~~Noname manuscript No.~~
(will be inserted by the editor)

An Analysis of *RelaxedIK*: An Optimization-based Framework for Generating Accurate and Feasible Robot Arm Motions

Daniel Rakita · Bilge Mutlu · Michael Gleicher

Received: date / Accepted: date

Abstract We present a real-time motion-synthesis method for robot manipulators, called *RelaxedIK*, that is able to not only accurately match end-effector pose goals as done by traditional IK solvers, but also create smooth, feasible motions that avoid joint-space discontinuities, self-collisions, and kinematic singularities. To achieve these objectives on-the-fly, we cast the standard IK formulation as a weighted-sum non-linear optimization problem, such that motion goals in addition to end-effector pose matching can be encoded as terms in the sum. We present a normalization procedure such that our method is able to effectively make trade-offs to simultaneously reconcile many, and potentially competing, objectives. Using these trade-offs, our formulation allows features to be *relaxed* when in conflict with other features deemed more important at a given time. We compare performance against a state-of-the-art IK solver and a real-time motion-planning approach in several geometric and real-world tasks on seven robot platforms ranging from 5-DOF to 8-DOF. We show that our method achieves motions that effectively follow position and orientation end-effector goals without sacrificing motion feasibility, resulting

This research was supported by the National Science Foundation under award 1208632 and the University of Wisconsin–Madison Office of the Vice Chancellor for Research and Graduate Education with funding from the Wisconsin Alumni Research Foundation.

Daniel Rakita
Department of Computer Sciences
University of Wisconsin–Madison
E-mail: rakita@cs.wisc.edu

Bilge Mutlu
Department of Computer Sciences
University of Wisconsin–Madison
E-mail: bilge@cs.wisc.edu

Michael Gleicher
Department of Computer Sciences
University of Wisconsin–Madison
E-mail: gleicher@cs.wisc.edu

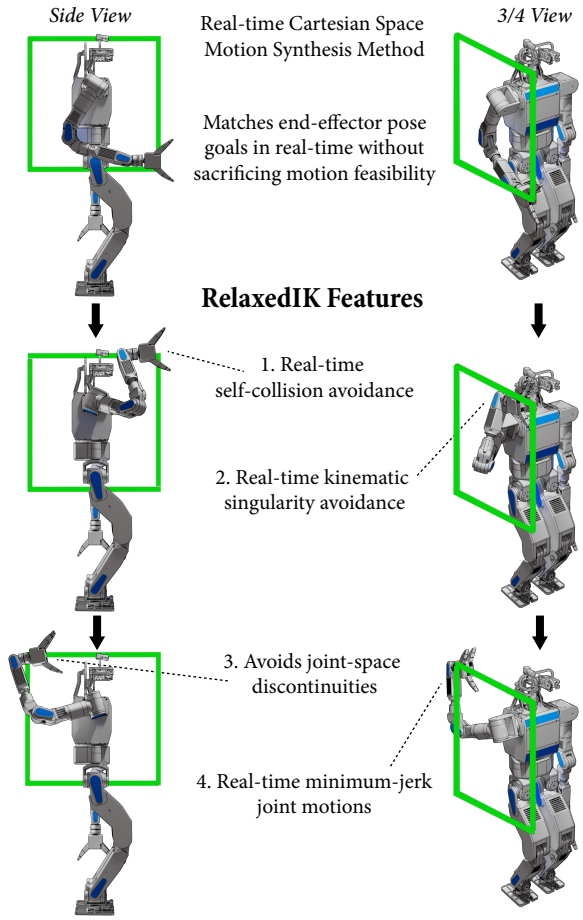

Fig. 1 In this paper, we present a method for generating accurate and feasible robot arm motions in real-time such that the arm not only can match end-effector pose goals, but it also avoids self-collisions, singularities, and joint-space discontinuities. This figure illustrates a DRC-Hubo+ robot performing the square-tracing task from our evaluation using *RelaxedIK*.

in more successful execution of tasks compared to the baseline approaches. We also empirically evaluate how our solver performs with different optimization solvers, gradient calculation methods, and choice of loss function in the objective function.

1 Introduction

To perform real-time tasks, a robotic manipulator must calculate how its joint angles should change at each update in order to meet kinematic goals rooted in its environment. For instance, a robot providing home-care assistance by

spoon-feeding an individual in a wheelchair would have to make real-time motion decisions to simultaneously meet many objectives, including the robot manipulator approaching the patient's head with smooth, self-collision-free motions, continuously updating the position and orientation of the spoon to account for potential head or torso motion, and keeping the spoon level such that the food does not spill. In this problem, and many other use cases for robotics, the robot must both accurately *match end-effector pose goals* and exhibit *motion feasibility*.

Prior approaches to matching end-effector goals while producing feasible motion provide only partial solutions. For instance, direct point-to-point methods provide accurate end-effector pose matching by solving inverse kinematics (IK) problems at each update. However, this approach does not guarantee feasible motion when generating a sequence of solutions and may result in behaviors such as exhibiting instantaneous jumps in joint space, causing damage to the robot through self collisions, and exhibiting unsafe behavior near kinematic singularities. Conversely, real-time motion-planning methods that calculate a path to predicted state as quickly as possible, *do* guarantee motion feasibility, but they do not ensure consistent matching of end-effector pose goals throughout a continuous motion. For instance, if the planner must compute a path between distant start and goal states, *i.e.*, when the IK solver exhibits a discontinuous jump after the prediction step, what the end-effector will do along the path between the Cartesian waypoints can be difficult to dictate.

In this paper, we present a novel real-time motion-synthesis method that simultaneously supports *end-effector pose goal matching* and *feasibility of motion*. We achieve accurate, feasible motion through a generalized IK solver, called *RelaxedIK*, that formulates the IK problem at each update as a weighted-sum non-linear optimization. Each term in the weighted sum encodes a specific motion objective, such as end-effector position/orientation goal matching, minimum-jerk joint motion, distance from self-collision state, etc. While parameter tuning can become unwieldy for multi-objective optimizations, we present a normalization procedure over the weighted sum terms that elicits expected and intuitive motion behavior. Because these objectives may be in conflict during motion, such as the robot trying to match a position goal within the volume of its body, the method automatically *relaxes* features that are in conflict with other features deemed more important at a given time. Our formulation does not rely on a specific optimization technique and provides sufficient solutions using many constrained non-linear optimization solvers.

Objective importances are specified through term weights and thus can be tailored to a specific task. Weights can even be dynamically adjusted during runtime to varying effect, such as decreasing relative importance on orientation matching when the robot must follow a fast-moving position goal (Rakita et al., 2017). Unlike many IK frameworks that achieve secondary goals through regularization techniques (Chiacchio et al., 1991; Mansard et al., 2009a,b; Nakamura, 1990; Siciliano, 1990), our method does not require any redun-

dant joints, and even provides as close as possible results on under-articulated robots.

Our method affords motion features that enable the creation of a sequence of feasible configurations, including minimum velocity, acceleration, and jerk joint motion; self-collision avoidance; and kinematic-singularity avoidance. While these features are more commonly found in offline trajectory-optimization and motion-planning methods (Kalakrishnan et al., 2011; Ratliff et al., 2009; Schulman et al., 2014), we utilize efficient and robust techniques to achieve real-time performance, such as using a neural network to approximate distance from collision states and singular value decomposition (SVD) to approximate distance to singular configurations.

We show the benefits of our method through various empirical tests that compare performance on several geometric and real-world tasks on seven robot platforms, ranging from 5-DOF to 8-DOF, against a state-of-the-art IK solver (Beeson and Ames, 2015) and a real-time motion-planning approach using the Open Motion Planning Library (OMPL) (Sucan et al., 2012). Our method achieves motions that effectively follow position and orientation end-effector goals without sacrificing motion feasibility, leading to more successful execution of tasks than the baseline approaches. In addition to the evaluations presented in our prior work (Rakita et al., 2018b), we also empirically evaluate how our solver performs with different optimization solvers, gradient calculation methods, and choice of loss function in the objective function. We discuss the main takeaways from our results, especially highlighting those that could pertain to real-world motion synthesis scenarios. A solver that implements the methods discussed in this work is available for download as open-source software at https://github.com/uwgraphics/relaxed_ik.

2 Related Work

The development of our method for accurate and smooth real-time motion synthesis draws from prior work in robotics, especially from inverse kinematics, teleoperation and controls, trajectory optimization, and motion planning, and from animation for methods that optimize over motions for real-time use.

Inverse Kinematics— The process of calculating joint-angle values on articulated chains that produce desired pose goals of end-effectors, called *inverse kinematics (IK)*, has been extensively studied in robotics and animation (see Aristidou et al. (2018) for a review of IK methods). The main objective of many IK solvers is to reliably match the end-effector goal as quickly as possible. A state-of-the-art solver to achieve these goals on robot chains is the Trac-IK solver proposed by Beeson and Ames (2015).

While 6-DOF chains generally have one solution to fully constrained position and orientation IK problems, prior research has attempted to take advantage of joint redundancy, if present, in order to achieve secondary goals by regularizing solutions—often called *task-priority IK* (Chiacchio et al., 1991; Chiaverini, 1997; Nakamura, 1990; Siciliano, 1990). Although we find inspira-

tion in the ability of these general regularization techniques to achieve supplementary objectives, our method does not require any redundancy and can even work on under-articulated robots. Hauser (2016) has formulated a framework for regularizing a smooth inverse of a multivariate function, such as a forward-kinematics model, such that the inverse is the same forward and backward along a path and that discontinuity boundaries are avoided as much as possible. This work has been shown to be effective with multiple redundant DOFs, but it has not been used on fully constrained IK problems. We believe that our methods complement the overall framework presented in this prior work.

In animation, work by Shin et al. (2001) introduced an IK technique for real-time articulated character puppetry that adjusted objectives on-the-fly depending on what is currently important. Our method is inspired by this idea of importance-based IK, as the terms in our weighted sum are formulated such that their respective motion features automatically relax if another more important term is in conflict. Also, the work by Baerlocher and Boulic (2004) presented an IK framework that is able to handle many competing constraints using multiple, hierarchically-based priority levels. The authors showed their method's ability to efficiently calculate a sequence of full-body postures with a high number of degrees of freedom and constraints. Our method draws on the idea of reconciling many potentially competing motion goals in a sequence of joint-space solutions, though we push these motion considerations to the objective function rather than using many hard constraints, as then the method can automatically arbitrate between different motion features with the goal of finding "close as possible" solutions without sacrificing motion feasibility.

Teleoperation and Controls—Synthesizing motions on the fly is particularly important in direct and shared control, as the system cannot look ahead to determine what motions will be required in the near future. The approaches described in seminal work, such as potential-field methods that afford real-time collision avoidance (Khatib, 1986) and kinematic-singularity-robust damped least-squares methods (Chiaverini, 1997; Maciejewski, 1990), serve as inspiration for our real-time motion-feasibility techniques. Work by Sentis and Khatib describes a method for controlling the motion of a humanoid by maintaining a hierarchy of motion constraints, operational tasks, and poses (Sentis and Khatib, 2005). The method is formulated such that lower priority tasks are projected onto the null space of higher priority constraints, guaranteeing that constraints will be met. Our method also contains many objectives and constraints of varying importances, but it is designed to be general enough to handle lower dimensional systems and constraints that do not have redundancy. Our prior work has shown the benefits of optimization-based methods for synthesizing motion across various task domains, including a method for real-time human-to-robot motion remapping to support intuitive teleoperation (Rakita et al., 2017), a motion-remapping technique used for motor task training (Rakita et al., 2018c), and a real-time motion synthesis method used to drive a robot camera to optimize a viewpoint for a remote teleoperation operator (Rakita et al., 2018a, 2019a). We have also applied our real-time mo-

tion synthesis approach for a bimanual shared-control method (Rakita et al., 2019c). In this work, the method moves the robot's arms to mimic the operator's arm movements, but provides on-the-fly assistance to help the user complete tasks more easily. This work shows the effectiveness of RelaxedIK in arbitrating between different motion and task objectives on-the-fly, as well as demonstrates that the method sufficiently scales up to higher-dimensional platforms.

End-Effector Path Following— While our method attempts to compute smooth, accurate, and feasible motions that follow end-effector pose goals in real-time, alternative methods have been proposed to address the end-effector trajectory tracing problem in an offline fashion. Work by Oriolo et al. considers searching through IK solutions organized in a graphical structure between a start and goal configuration using RRT-like search strategies Oriolo and Mongillo (2005); Oriolo and Vendittelli (2009). Work by Cefalo et al. presents a randomized motion planning method for redundant robot systems that generates cyclic motions that smoothly follow an end-effector trace while avoiding hard constraints such as collisions in the environment (Cefalo et al., 2013). Rakita et al. present a technique for finding feasible robot arm trajectories that pass through provided 6-DOF Cartesian-space end-effector paths with high accuracy by searching through a temporally organized graph of sufficient IK solutions (Rakita et al., 2019b). Praveena et al. present a solution that finds a set of approximate candidate paths that pass through a given end-effector trace, each with its own set of trade-offs, and affords users the option to provide sparse input to select a path that would be most fitting given the task at hand Praveena et al. (2019).

Real-time Motion Planning—Our work shares parallels with real-time motion planning techniques, which involve planning to predicted end-effector pose goals as fast as possible to meet real-time demands. Hauser (2012) provides an adaptive way to adjust the planning horizon time such that prediction and planning steps can be interleaved in a stable manner. Our work shares similar outcomes to this work, such as planning around obstacles in real-time. However, as we show in this work, controlling the end-effector pose en route to a predicted waypoint using motion-planning methods is difficult and ineffective for certain tasks. Additionally, Murray et al. (2016) present a real-time motion-planning approach that can solve for paths very quickly by reasoning about paths at the hardware level using a custom chip. While this approach enables feasible paths through joint space to be found nearly instantaneously, it does not provide solutions that enable the robot to follow precise Cartesian paths as done in our solution. To overcome this problem, Murray et al. (2016) used motion planning to find a path that exhibited an end-effector pose at the end point within 10 cm of the goal and then switched over to a Cartesian planner to precisely approach the goal.

3 Technical Overview

The main goal of our method is to calculate robot motions that match end-effector pose goals while also exhibiting motion feasibility. In this section, we provide a high level idea of how our method is structured to achieve both of these goals, leaving the mathematical treatment of our solutions for §4.

3.1 Problem Formulation

Our method is rooted as a standard inverse kinematics (IK) problem. At each system update, the method receives a goal position $\mathbf{p}_g$ and a goal orientation $\mathbf{q}_g$ for the end-effector and outputs joint angles corresponding to a desired robot state.

While standard IK approaches solely focus on matching end-effector pose goals as accurately and quickly as possible, our method also considers robot configuration feasibility upon a sequence of solutions. Throughout this work, we define *robot configuration feasibility* as meeting the criteria that the robot (1) exhibits sufficiently close consecutive solutions such that simple interpolation between these states is likely to be successful; (2) does not collide with itself, or any modeled obstacle, on any frame (which, coupled with criterion 1, should lead to a collision-free path over time); and (3) stays sufficiently far from kinematic singularities (when the robot's Jacobian matrix loses full column rank) at each time step. Given this notion of feasibility, we reformulate the standard IK problem as follows:

Match the end-effector pose goal corresponding to goal position $\mathbf{p}_g$ *and goal orientation* $\mathbf{q}_g$ *as precisely and quickly as possible without sacrificing robot configuration feasibility.*

We expect that solving discrete IK problems at each update with this central goal, each with an individual sense of feasibility, will in turn yield continuous and feasible motions upon a sequence of such solutions. This formulation does not consider end-effector pose matching as a hard constraint; pose goals may instead be *relaxed* if other, more important features will be met. This is a key insight in our method, as this affords feasible and smooth motions even when such a path does not exist passing through exact IK solutions. It is also this relaxation characteristic that allows our method to work without any joint redundancy, as our method will inherently "regularize" solutions in operational space if deemed necessary, even if a null-space is not present on a 6-DOF (or less) robot.

3.2 Importance-Based Inverse Kinematics

Because our method can relax certain features in favor of other features, it must offer intuitive and robust ways to set and tune relative weights between objectives. To achieve this goal, we draw from a concept called *importance-based inverse kinematics*, a technique pioneered in animation to drive real-time

performance capture (Shin et al., 2001). Prior work notes the key observation that the main objective in an IK problem can vary across scenarios, such as whether the animated character should match the general arm shape of the actor, e.g., when making a communicative gesture, or match the end-effector pose of the actor in space, e.g., when picking up an object. The primary way of setting objective term relative importances in through static *weight* values for each term.

While tuning an array of parameters can become unwieldy in multi-objective optimizations, we present a normalization procedure, outlined in §4.1, that ensures that the method reasons over values in a standard range. This procedure allows the weights described above to elicit expected behavior, making parameter tuning practical over numerous terms.

3.3 Optimization Overview

Given the varying objective importances outlined in the previous section, our method needs some way of reconciling many, potentially competing goals of different priorities in real-time. To achieve this, we use a non-linear constrained optimization formulation, which attempts to drive down the objective values of the various objective function terms, subject to a set of constraints.

We express the IK problem as follows:

$$\begin{aligned} \Theta = \underset{\Theta}{\arg\min} \quad & \mathbf{f}(\Theta) \text{ s.t. } \mathbf{c}_i(\Theta) \ \geq \ \mathbf{0}, \quad \mathbf{c}_e(\Theta) \ = \ \mathbf{0} \\ & l_i \leq \Theta_i \leq u_i, \forall i \end{aligned} \tag{1}$$

where $\mathbf{c}_i(\Theta)$ is a set of inequality constraints, $\mathbf{c}_e(\Theta)$ is a set of equality constraints, l_i and u_i values define the upper and lower bounds for the robot's joints, and $\mathbf{f}$ is an objective function. Our challenge is to encode our motion and feasibility goals within the constraints and objectives.

We express our objective function as a weighted sum of individual goals, such as end-effector position matching, end-effector orientation matching, minimum jerk joint motion, and distance to singularity, and formalize it as follows:

$$\mathbf{f}(\Theta) = \sum_{i=1}^{k} \ w_i \ * f_i(\Theta, \Omega_i) \tag{2}$$

Here, w_i is a static weight value for each term, as described in §3.2, which allows the user to incorporate prior knowledge about what terms are most important for a given task, and $f_i(\Theta, \Omega_i)$ is an objective-term function that encodes a single sub-goal, with Ω_i being model parameters used to construct a particular loss function. The exact structure of the $f_i(\Theta, \Omega_i)$ objective functions are covered in §4.1.

Our full optimization formulation is comprised of seven objective terms and two constraints. The objective terms encode the following kinematic goals: (1) End-effector position matching; (2) end-effector orientation matching; (3)

minimized joint velocity; (4) minimized joint acceleration; (5) minimized joint jerk; (6) minimized end-effector translational velocity; (7) and self-collision avoidance. The two constraints are designed to clamp joint velocities at each update and avoid kinematic singularities, respectively. These objectives and constraints are detailed throughout §4.

4 Technical Details

In this section, we cover the mathematical details that instantiate the high level ideas outlined in §3. We first cover the structure of our objective function, then detail the objective terms and constraints that comprise our full optimization.

4.1 Objective Function Structure

While a weighted-sum objective function affords expressiveness by encoding each motion goal as a single term in the sum, parameter tuning of the weights can become unwieldy, often leading to unstable or divergent behavior if care is not taken. Parameter tuning would be particularly troublesome in our Cartesian-space motion-synthesis approach, as many objectives may be in conflict at any given time. Ideally, the term weights would correspond to easily explainable behavior, such as a term with weight of two being twice as important as a term with weight of one in the optimization. This behavior is not observed using standard loss functions, such as quadratic, because optimized terms can be over different units at vastly different scales (e.g., joint-space velocities compared to Euclidean distances in operational space).

To facilitate combining objectives, we normalize each term using a parametric normalization function that is designed to scale each function to a uniform range. This function places a narrow "groove" around the goal values, a more gradual falloff away from the groove in order to better integrate with other objectives, and exhibits a consistent gradient that points towards the goal. We refer to this function as the "Groove loss" due to its groove-like shape at its center. We implement this normalization function as a Gaussian surrounded by a more gradual polynomial:

$$\begin{aligned} f_i(\Theta, \Omega_i) = (-1)^n exp(\frac{-(\chi_i(\Theta) - s)^2}{2c^2}) \\ + r * (\chi_i(\Theta) - s)^4 \end{aligned} \tag{3}$$

Here, the scalar values n, s, c, r form the set of model parameters Ω. Together, they shape the loss function to express the needs of a certain term. Here, $n \in \{0, 1\}$, which dictates whether the Gaussian is positive or negative. Negative Gaussian regions are areas of high "reward," while the optimization will push away from positive regions of high "cost." The value s shifts the function horizontally, and c adjusts the spread of the Gaussian region. The r value adjusts the transition between the polynomial and Gaussian regions,

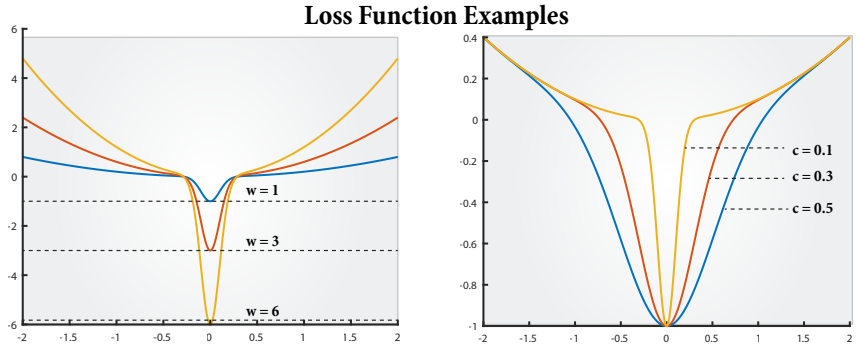

Fig. 2 Examples of the Groove loss function used in our weighted-sum objective. Left: Scalar multiplication by a weight fully controls the amplitude of the reward region. The w values illustrated here correspond to the w_i objective term weight values in Equation 2. Right: The value "c" controls the spread of the reward region.

higher values showing a steeper funneling into the Gaussian region and lower values flattening out the boundaries beyond the Gaussian. The scalar function $\chi(\Theta(t))$ assigns a numerical value to the current robot configuration that will serve as input to the loss function.

In our prototype solver described in §5, all parameter and weight values were selected empirically and were observed to work well in practice; however, using the normalization procedure described in this section, all parameters are robust to tuning for differing results. In the remainder of this section, we will outline the $\chi(\Theta(t))$ functions and model parameters used to formulate our motion-synthesis method.

4.2 End-Effector Position Matching

The first term in our weighted sum objective function involves matching up the robot's end effector position to a provided goal position $\mathbf{p}_g$. To achieve this goal, our solver minimizes the $L2$ error between the robot's end effector position given the joint configuration Θ and the goal position $\mathbf{p}_g$. Put formally, the objective term is formalized as:

$$\chi_p(\Theta) = || \ \mathbf{p}_g - FK(\Theta) \ ||_2 \tag{4}$$

Here, $FK(\Theta)$ signifies the end-effector position given joint angles Θ, calculated by the robot's forward kinematics model.

We inject this objective term value $\chi_p(\Theta)$ into the parametric loss function described in §4.1 using model parameters $n = 1$, $s = 0$, $c = 0.2$, and $r = 5.0$.

4.3 End Effector Orientation Matching

To match the robot's end-effector orientation to a provided goal quaternion $\mathbf{q}_g$, we introduce an objective term that will be minimized as the orientations align. We measure the difference between orientations as the magnitude of the rotation vector between them, $disp(\mathbf{q}_1, \mathbf{q}_2) = log(\mathbf{q}_1^{-1} * \mathbf{q}_2)$ (Lee, 2008). The objective term is therefore:

$$\chi_o(\Theta) = || \; disp(\; \mathbf{q}_g, \mathbf{q}[\hat{FK}(\Theta)] \;) \; ||_2 \tag{5}$$

Here, $\hat{FK}(\Theta)$ specifies the end-effector rotation frame at joint configuration Θ, calculated through the robot's forward kinematics model, and $\mathbf{q}[.]$ indicates a conversion from rotation matrix to quaternion.

Two quaternions can specify the same static orientation. This quaternion pair, $(iq_x + jq_y + kq_z + q_w)$ and $(-iq_x - jq_y - kq_z - q_w)$, are called *anti-podal equivalences*. While the two quaternions encode the same orientation, they produce different results when used in the quaternion displacement operator. Thus, in our orientation objective, we check the result of both anti-podal equivalences at each iteration, and we always minimize over the one with smaller displacement to always encourage convergence.

We add this objective term value $\chi_o(\Theta)$ into the parametric loss function described in §4.1 using model parameters $n = 1$, $s = 0$, $c = 0.2$, and $r = 5.0$.

4.4 Smooth Motion Synthesis

A main goal of our method is to produce smooth joint motion without exhibiting joint-space discontinuities. We achieve this goal using four objective terms and one hard constraint.

The first three smoothness objective terms strive to minimize joint velocity, acceleration, and jerk, respectively:

$$\chi_v(\Theta) = ||\dot{\Theta}||_2 \; ; \; \chi_a(\Theta) = ||\ddot{\Theta}||_2 \; ; \; \chi_j(\Theta) = ||\dddot{\Theta}||_2 \tag{6}$$

Velocity, acceleration, and jerk are approximated using backward finite differencing using a window of the past four solutions. Having smooth joint motion up to the third derivative is beneficial in terms of wear and tear on the robot. Prior work also shows this characteristic to be present when people move their arms to complete tasks (Flash and Hogan, 1985), suggesting that the generated motions may have a more human-like quality.

We also include an objective term that minimizes velocity in the robot's end-effector position space:

$$\chi_e(\Theta) = ||\dot{FK}(\Theta)||_2 \tag{7}$$

This term discourages large jumps in operational space, acts as a real-time filter, reduces motion jitters when performing fine-motion tasks, and facilitates

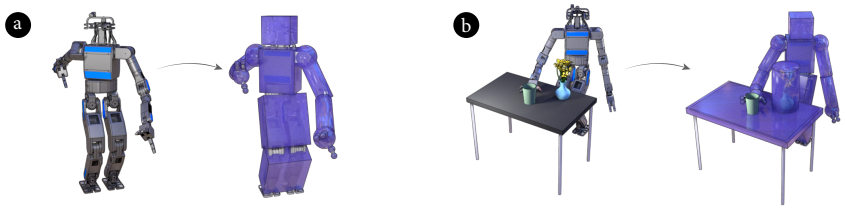

Fig. 3 (a) Our collision avoidance method starts by enveloping the robot's links and body components in geometric primitives. The method uses pairwise distance calculations between these geometric primitives to compute a potential function that characterizes a distance to a collision state. A neural network is trained to approximate this potential function to speed up the distance calculation, making it fast enough to serve as a single term in our optimization objective function. (b) Static environment objects can also be enclosed in collision objects, such as the table-top and vase seen here. After training the neural network, the robot will learn that a high collision cost is elicited when its links are close to the table or vase and will attempt to avoid these objects throughout its motions.

motions along straight lines. These four terms all use the same loss function model parameter values $n = 1$, $s = 0$, $c = 0.2$, and $r = 5.0$.

Lastly, because the aforementioned smoothing terms only encourage motion properties, but do not place any bounds in the case of errors, we include hard inequality constraints on individual joint velocities to further account for failure cases:

$$\mathbf{c}_{v_i} := |\dot{\Theta}_i| \ \leq v_i, \ \forall i \in \{1, ..., N\} \tag{8}$$

Here, v_i refers to the joint-velocity limit for joint i over a single update, and N is the number of robot DOFs.

4.5 Self-Collision Avoidance

A key feature of our real-time motion-synthesis method is to provide a way for the robot to avoid any self-collisions, even when using per-frame IK with no look-ahead or prediction. While existing methods can detect when a robot model is colliding with itself, a standard feature within the MoveIt! framework,[1] being alerted of a collision after it happens is not appropriate in real-time motion synthesis. Instead, our method can approximate *how imminent* the robot is to a collision state and favor configurations that are as far away as possible from self-collision states while still pursuing other goals.

Our approach follows two steps: (1) create a smooth, continuous function that approximates a self-collision cost given a joint state Θ, called $col(\Theta)$. This is essentially a *potential function*, congruent with prior collision-avoidance techniques, that is high when near collision and low otherwise (Khatib, 1986; Nakamura, 1990); and (2) train a neural network to learn the function from step 1 to speed up the collision approximation process by over two orders of

[1] http://moveit.ros.org/

magnitude, making this procedure fast enough to be optimized over in real-time.

While our initial prototype implementation presented in prior work used an approximation of the robot's geometry using line segments (Rakita et al., 2018b), the current work extends this idea to other geometric representations, such as cubes, capsules, or even full mesh models. The method automatically envelopes the robot's links in capsule objects, and the user may supplement the scene with other static geometric primitives, such as a thin cube in front of the robot to represent a table-top or a cylinder placed on the table to represent a vase. The method also supports manual specification of dynamic collision objects that are rigidly attached to the robot, such as a sphere object that surrounds the robot's end-effector, or a cube object specified for the robot's head. Throughout this work, we refer to these collision geometric primitives as *collision objects*. A visual representation of collision objects attached to and around a robot platform can be seen in Figure 3.

We start by characterizing the overall geometry of a robot arm by assessing distances between its collision objects in a non-collision state, provided by the user. This allows the method to discern when a collision is likely imminent, as opposed to two collision objects just being naturally close together in a safe state. We will refer to the sample configuration not in self-collision as Θ_s. We calculate the forward kinematics Θ_s, update the collision objects such that the items rigidly attached to the robot appropriately correspond to the new configuration, and store the orthogonal distances between all pairs of collision objects l_i and l_j in a table $d_{i,j}$. Note that $d_{i,j} = 0$ when l_i and l_j are adjacent, or when $i = j$.

Given these initial distances $d_{i,j}$, the method exponentially increases the self-collision cost as distances between collision objects l_i and l_j are observed to be increasingly less than their standard distance $d_{i,j}$. We use a sum of Gaussian terms to exponentially scale up the cost based on distance between all pairs of links, which are smooth and differentiable when taking gradients for optimization. The function is defined as follows:

$$\begin{aligned} col(\Theta) &= \sum_{i,j} b * exp(\frac{-dis(l_i, l_j)^2}{2c^2}) \\ c &= -d_{i,j}^2/(2 * log(1\text{e}{-}15/b)) \end{aligned} \tag{9}$$

Here, $dis(l_i, l_j)$ signifies the distance between collision objects corresponding to the query state Θ. The b value defines the amplitude of the Gaussian and normalizes a total range of return values, and the c value adjusts the spread of the Gaussian such that it starts to trend upwards only when $dis(l_i, l_j)$ is less than its standard distance $d_{i,j}$. When $c = 0$, i.e., when $d_{i,j} = 0$, the division by zero is manually avoided and nothing is added to the sum. In our prototype solver, we used a value of $b = 50$.

Depending on the complexity of the robot model and the number of supplemental collision objects specified in the environment, the function in Equation 9 checks all combinations of objects in approximately 5 - 20 *ms*. While this

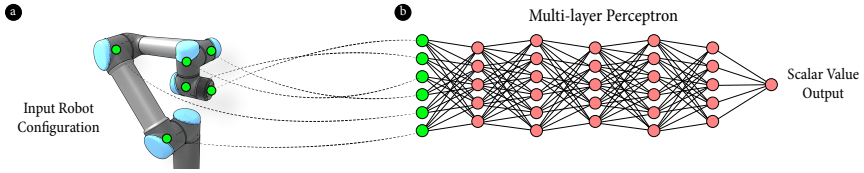

Fig. 4 Illustration of our collision avoidance neural network procedure. (a) The robot's joint points are calculated at the query configuration given the forward kinematics model. (b) The joint positions are concatenated together and are provided as inputs to the neural network. The network is a fully connected multi-layer perceptron with ReLU activations on all nodes. The output of the neural network is a single scalar value signifying an approximate distance to a collision state, where a higher value designates that the robot is closer to a collision.

performance may be sufficient for quick checks throughout run-time, it is not fast enough for real-time optimization where the full objective function may be called more than 100 times per solution. To speed up this process, we train a neural network to learn $col(\Theta)$, which then only requires a simple matrix multiplication for evaluation.

We used a multi-layer perceptron neural network with six hidden layers to learn $col(\Theta)$. We observed that concatenating the joint points $[j_1, j_2, ...j_N]$ as inputs worked considerably better than naïvely using the robot state Θ. This adds little overhead to the system as the forward kinematics are already being calculated for use by other objective terms. All of the six layers contains $N * 3 + 5$ nodes, such that each is slightly wider than the input vector. Each node uses a ReLU activation function. We used 100,000 training inputs by randomly generating states, and using outputs of the original $col(\Theta)$ function. We used the Adam solver to run the network optimization with a learning rate of $\eta = 0.001$. It takes about 10-25 minutes during preprocessing to generate all 100,000 input and output pairs, and about another 20 minutes to train the network.

Once the neural network is trained, we have a new function $col\_nn(\Theta)$ that sufficiently matches the outputs of $col(\Theta)$ but evaluates a cost in about 10 - 30 microseconds (dependent on the number of robot DOFs). This approximately 2-3 orders of magnitude gain in speed over $col(\Theta)$ enables real-time optimization. Our objective term is as follows:

$$\chi_c(\Theta) = col\_nn(\Theta) \tag{10}$$

This objective uses loss function model parameter values $n = 0$, $s = 0$, $c = 0.08$, and $r = 1.0$.

4.6 Kinematic Singularity Avoidance

Kinematic singularities are well studied in robotics (Gosselin and Angeles, 1990). These objectionable robot poses occur when the Jacobian matrix $\mathbf{J}(\Theta)$ that maps joint and end-effector tool velocities, i.e., $\dot{\mathbf{x}} = \mathbf{J}(\Theta)\dot{\Theta}$, loses full

column rank. Under these circumstances the chain may lock since an instantaneous change in one of the end-effector DOFs is unattainable. Further, when the Jacobian matrix is near singular, small changes in the end-effector tool space can induce large, diverging velocities in joint angle space, which is unsafe for many applications.

To avoid singular configurations in our motion synthesis method, we use the following approach: (1) find a metric that can approximate distance to a singularity; (2) characterize the robot's general manipulability during preprocessing by analyzing the singularity distance metric in many configurations; and (3) set a hard constraint that avoids configurations deemed to be close to singularities based on the analyses from step 2.

Because kinematic singularities occur when the Jacobian matrix loses full rank, we use a common metric that approximates distance to such a configuration, called the *matrix condition number*, which we denote as c. This value is found by taking the SVD of the matrix, then taking the ratio of the smallest singular value and the largest singular value: $c = \sigma_N/\sigma_1$. When this value is small, it indicates that the matrix is not well conditioned, and is close to losing full rank.

Because every robot arm has a distinct geometry and kinematic structure, the distribution of the conditioning number will vary for each arm. This characteristic of a particular robot arm is called its *manipulability* and is analyzed through a multi-dimensional object called a *manipulability ellipse* (Yoshikawa, 1985). We chose to analyze the *matrix condition number* of the Jacobian as a proxy distance to singularity over the Yoshikawa manipulability measure (Yoshikawa, 1985), because the condition number favors general *roundness* of the manipulability ellipse, rather than favoring a larger ellipse as a whole, which generalizes better across different robots (Nakamura, 1990).

To assess the properties of an arm's manipulability ellipse, we randomly sample 500,000 robot configurations during preprocessing and find the mean, μ_c, and standard deviation, std_c, of all condition values c. We make the model assumption that the condition-value random variable is approximately normal and set a hard constraint in the optimization such that configurations with condition values less than $\mu_c - b * std_c$ are avoided, for some scalar b. In our prototype solver, we used a value of $b = 2$, such that approximately the bottom 2.5% of configurations in terms of condition score will be avoided.

5 Experimental Evaluation

In this section, we outline the empirical tests carried out to validate our method. Specifically, we describe the prototype solver that instantiated our methods, provide details on our four experiments, and finally discuss our findings.

5.1 Prototype Details

To demonstrate the effectiveness of our method on various robot platforms and tasks, we implemented a prototype solver that instantiates our real-time motion-synthesis method. The performance intensive aspects of the solver are implemented in the Julia programming language, while higher level interfacing is specified in Python. The solver integrates natively with ROS, enabling real-time monitoring of optimization parameters and constraints, multi-threading, and communication with robot controllers. All of our testing was performed on an HP Pavilion laptop with an Intel Core 2.6 GHz i7-6700HQ CPU with 32 GB RAM.

Because our method requires information about the kinematic structure and geometry of the particular robot arm before run-time, it includes a one-time preprocessing step to gain this information prior to the use of the solver. This step takes as input a robot description in URDF format and initializes various procedures to learn certain geometric and kinematic features about the robot platform. The preprocessing step takes approximately 30–40 minutes, and the resulting output configuration files can be reused to seed the solver.

5.2 Experimental Procedure

All of our evaluations involved robot platforms executing five tasks, outlined in §5.3, simulated on seven robot platforms featuring 5 to 8 DOF arms, including the Fanuc LR Mate 200ic[2] (5-DOF), a Universal Robots UR5[3] (6-DOF), a Kinova Jaco[4] (6-DOF), a Rethink Robotics Sawyer[5] (7-DOF), a Kuka IIWA 7[6] (7-DOF), a Rainbow Robotics DRC-Hubo+ arm[7] (7-DOF), and a DRC-Hubo+ arm-and-waist rotation (8-DOF). We manually selected initial configurations for the robots such that all robots faced the same direction with matching end-effector orientations, and the tasks operated analogously across platforms.

Because real-time motion tasks are very sensitive to an initial configuration, we followed a randomization procedure on initial configurations to account for experimenter bias. For each trial, the system randomly generated a vector shorter than 0.2 m and calculated a random configuration based on this displaced starting position using Trac-IK. The maximum displacement was selected such that the robot always stayed within its manipulation envelope. We did not randomly offset the orientation, because the absolute directions of the end-effector's coordinate frame were often important for a given task. Each task was run with 100 random initial configurations.

[2] http://www.fanuc.eu/se/en/robots/robot-filter-page/lrmate-series
[3] https://www.universal-robots.com/products/ur5-robot/
[4] http://www.kinovarobotics.com/innovation-robotics/products/robot-arms/
[5] http://www.rethinkrobotics.com/sawyer/
[6] https://www.kuka.com/en-us/products/robotics-systems/lbr-iiwa
[7] http://www.rainbow-robotics.com/products_humanoid

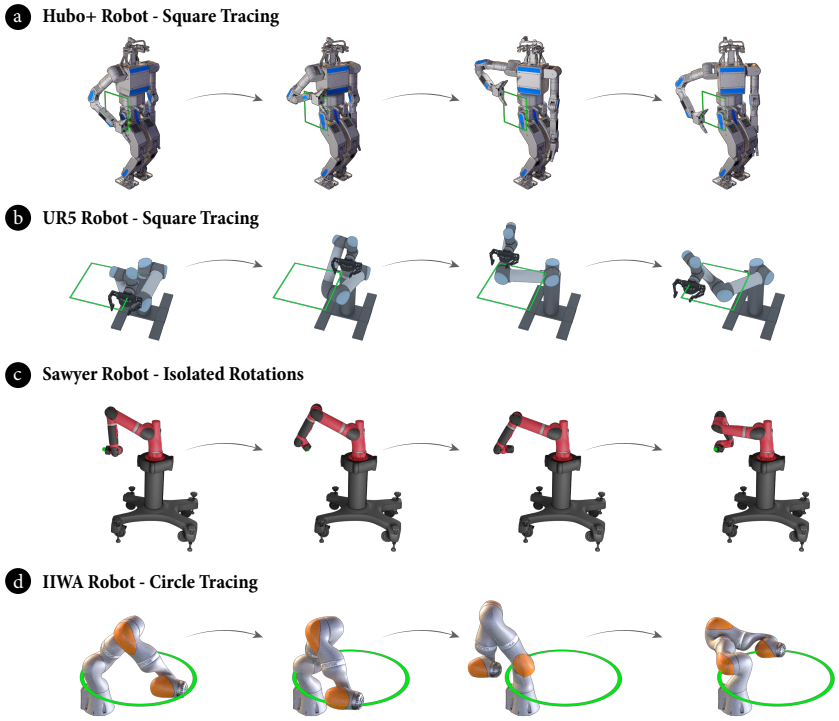

Fig. 5 Various robots exhibiting motions solved for on-the-fly using *RelaxedIK* in order to execute tasks used in our experimental test bed. (a)-(b) The Hubo+ and UR5 robots are performing the square-tracing task with smooth, accurate, and feasible motions. (c) The Sawyer robot is performing the isolated rotations task, where the robot rotates its end-effector around a single point. Note how the robot slightly relaxes the position-matching objective in the third and fourth images in order to avoid a kinematic singularity. (d) The IIWA robot is performing the circle tracing task. Note how the robot avoids the self-collision in the third image, even when the end-effector path goes through its static base, by automatically relaxing the position-matching objective and momentarily moving its end-effector from the path at those points to find close, yet still feasible, solutions.

Our test bed was run in simulation and consisted of a total of 1,535,500 discrete solutions, including seven robot platforms, five tasks, and 100 random initial configurations.

5.3 Experimental Tasks

Our experimental test bed consisted of five tasks, including three *geometric* tasks that enabled us to analytically assess the input curve if necessary and two *use-case* tasks involving a robot home-care assistant. The geometric tasks included *circle tracing*, *square tracing*, and *isolated rotations*. Tracing tasks involved the end-effector following a perfect a circle or a square centered at

the robot's base and scaled for each robot to span close to the robot's whole workspace range. The IK goal did not ease in and out at the square's corner, instead following a constant velocity even at the sharp corners. For the tracing tasks, the robot's end-effector remained static in its initial orientation. Isolated rotations involved the robot's end-effector rotating 180-degrees and back around yaw, pitch, and roll axes. No end-effector translation was present for this task. The two home-care-scenario tasks were *spoon feeding* and *cooking*. Spoon feeding involved the robot arm using a spoon to retrieve food from bowls placed around the workspace using a spoon and to offer the food to an individual in a wheelchair for feeding. Cooking was a two-arm task involving moving a pot from the stove top to the counter. Because the two arms have to coordinate, end-effector configurations and motion feasibility are both of particular importance, as highlighted in previous work (Sina Mirrazavi Salehian et al., 2016). For the home-care tasks, the end-effector traces were hand animated in a 3D-animation tool at 50 *Hz*. Examples of robots executing some of our tasks can be seen in Figure 5.

5.4 Measures

We assessed eight objective measures in all of our evaluations: mean position error (meters), mean rotational error (radians), mean joint velocity (rad/s), mean joint acceleration (rad/s^2), mean joint jerk (rad/s^3), total number of joint discontinuities, total number of singularities, and total number of self-collisions. We also report on the average solution time (seconds) for each condition.

5.5 Experiment 1: Comparing Against Alternative Real-time Synthesis Approaches

In our first experiment, we compared *RelaxedIK* to two alternative real-time motion synthesis approaches. We first present the comparisons used in the evaluation, then overview our results.

5.5.1 Experiment 1 Comparisons

We compared *RelaxedIK* against three alternative real-time motion-synthesis approaches. The first comparison is a direct point-to-point approach that uses a state-of-the-art IK solver, Trac-IK (Beeson and Ames, 2015), to perform per-update IK on the given end-effector pose goal. Trac-IK is an slsqp optimization-based IK formulation that minimizes the distance between the given pose goal and the pose of the end-effector, structured as a displacement of dual-quaternions. This formulation also minimizes velocity from a seed state to an optimized state. We seed Trac-IK with the configuration from the previous update. Our tests used the open-source C++ Trac-IK library (Beeson and Ames, 2015).

Our second comparison is a direct point-to-point approach using *IKFast*, an analytical inverse kinematics solver that can quickly find exact solutions for IK problems (Diankov, 2010). While IKFast can, in theory, find solutions for fully constrained IK problems on redundant robots with greater than six joints, it can only do so if the problem is reduced down to a 6-DOF problem, e.g., by fixing $d-6$ joints in the chain, where d is the number of joints on the redundant platform. If the redundant joints are *not* fixed, the number of exact IK solutions is infinite, and the solver has no way to reduce the set down to a single solution or set of reasonable solutions. Because there is not a systematic way to decide which redundant joint(s) in the chain to fix or what value to fix these redundant joints to at a given time to afford effective motion synthesis, we only tested IKFast on 6-DOF robots. If multiple valid solutions were found by IKFast on the 6-DOF robot tasks, the system chose the one closest to the robot's current configuration in terms of Euclidean distance.

Our third comparison is *real-time motion planning*, which predicts what pose the end-effector should have in the future, calculates a goal state corresponding to the predicted pose using an IK solver, plans a feasible motion from the current state to the goal state, and finally executes the trajectory along this planned path (Hauser, 2012). The planning and execution phases proceed as fast as possible to meet real-time demands. Our testing used the open-source OMPL (Sucan et al., 2012) motion planners that are integrated within the MoveIt! ROS package.

Our implementation used Trac-IK, as incorporated into MoveIt!, as the IK solver after the prediction step. We allow the IK solver to have perfect pose prediction up to 0.2 s ahead at the prediction step to prevent negative results due to poor prediction or an inadequate planning horizon to be able to observe real-time motion planning under ideal conditions.

For the motion-planning phase, we first use an RRT-Connect planner (Kuffner and LaValle, 2000) and then a PRM planner(Kavraki et al., 1996) as backup if the first planner failed to find a path. Because state-of-the-art techniques, such as parallelization using GPUs (Bialkowski et al., 2011) and hardware-level planning using custom chips (Murray et al., 2016), allow real-time use of motion planning, we also provide motion planners with as much time as they needed to converge, even when tracking the real-time goal in the test bed. We report solution times based on the implementation described above, although we make the assumption that these real-time approaches could keep up with such goals given their reported timing information.

We tested *RelaxedIK* with two different importance weight configurations. In *RelaxedIK (A)*, the configuration emphasized end-effector accuracy with the following weights: $\{w_p = 50, w_o = 40, w_v = 0.1, w_a = 1, w_j = 2, w_e = 0.1, w_c = 2\}$. The configuration in *RelaxedIK (B)* emphasized smoothness and feasibility with the following weights: $\{w_p = 10, w_o = 9, w_v = 5, w_a = 4, w_j = 3, w_e = 2, w_c = 5\}$.

5.5.2 Experiment 1 Results

Our results for Experiment 1 are summarized in Table 1. *RelaxedIK (A)* was shown to have higher end-effector accuracy than all other comparisons. *RelaxedIK (B)* did show some motion smoothness benefits, as seen by having lower joint velocity, acceleration, and jerk results than all other comparisons; however, these benefits come at the cost of inducing more end-effector position and rotation errors than *RelaxedIK (A)*. Both instantiations of *RelaxedIK* exhibited feasible motions on all solutions, without exhibiting any discontinuities, singularities, or self-collisions. In contrast, *direct point-to-point* using *Trac-IK* and *IKFast* encountered many of these errors, which would result in infeasible motions when run on a robot platform.

	Pos. Error	Rot. Error	Joint Velocity	Joint Accel.	Joint. Jerk	Number of Discontinuities	Number of Singularities	Number of Self-Collision	Solution Times
RelaxedIK (A)	0.0047 ± 0.003	0.0029 ± 0.008	0.0118 ± 0.013	0.0004 ± 0.001	0.0002 ± 0.001	0	0	0	0.0046 ± 0.008
RelaxedIK (B)	0.0118 ± 0.018	0.0284 ± 0.026	0.0101 ± 0.008	0.0001 ± 0.001	0.0001 ± 0.001	0	0	0	0.0046 ± 0.009
Trac-IK	0.0051 ± 0.005	0.0233 ± 0.040	0.0803 ± 0.084	0.113 ± 0.148	0.2110 ± 0.270	2,260	633	1,350	0.0029 ± 0.002
IKFast	0.017* ± 0.013	0.133* ± 0.130	0.048* ± 0.064	0.091* ± 0.083	0.179* ± 0.150	538*	43*	662*	0.000007 ± 0.00
Real-time MP	0.2523 ± 0.241	0.2517 ± 0.275	0.0357 ± 0.025	0.014 ± 0.013	0.0261 ± 0.025	0	9	0	0.0542 ± 0.039

Table 1 Summary of aggregated results from our Experiment 1. The measures are mean position error (meters), mean rotational error (radians), mean joint velocity (rad/s), mean joint acceleration (rad/s^2), mean joint jerk (rad/s^3), total number of joint discontinuities, total number of singularities, total number of self-collision, and mean solution time (seconds). The range value is standard deviation. The * on the IKFast condition values denotes partial results only over the 6-DOF robots.

Although the mean position and rotation errors were lower for *Trac-IK* than for *RelaxedIK(B)*, the errors induced by *RelaxedIK(B)* were due to feasibility considerations, i.e., avoiding acceleration and jerk disturbances which may cause the end effector to drift slightly behind the goals or slight deviations to prevent more serious problems such as self-collision or joint discontinuity. Further, if the reported end-effector pose matching errors would be too high for a given task, respective weights on position and orientation matching terms could be raised to exhibit a reduction in these errors. To illustrate, *RelaxedIK(A)* exhibited *lower* end-effector position and rotation errors than *Trac-IK* as the weights on motion accuracy were raised for this condition.

While *Trac-IK* and *IKFast* precisely hit end-effector poses when solutions existed, they often fell behind the end-effector pose goals when solutions could not be found. *Trac-IK* and *IKFast* also had major motion feasibility issues. For instance, *Trac-IK* resulted in $2,260$ discontinuities, 633 singularities, and $1,350$ self-collisions, and *IKFast* resulted in 538 discontinuities, 43 singularities, and 662 self-collisions on just the 6-DOF robots. In contrast, *RelaxedIK* did not exhibit such problems in any solution.

At a high level, *real-time motion planning* exhibited consistent motion feasibility, showing no joint discontinuities, but did not reliably get close to end-effector position and rotation goals. These errors followed one of two patterns: (1) when the motion planner had to interpolate a long path due to a

discontinuity, the end-effector had to deviate from the path to reach the goal state; and (2) when the motion planner failed to find a path, the robot stayed at its previous state, causing significant end-effector pose-matching errors. In contrast, *RelaxedIK* showed the same level of motion feasibility while reliably matching end-effector poses throughout the tasks.

Our method takes on average 4.6 *ms* to find a solution. While *Trac-IK* provided a solution in slightly less time (2.9 *ms* on average in our testing), and *IKFast* found solutions very quickly (7 microseconds on average), in many scenarios the feasibility benefits provided by our method may outweigh the cost of the extra computation time. Our results also indicate that the joint motion generated by *RelaxedIK* is considerably smoother than *direct point-to-point*, *IKFast* and *real-time motion planning*, demonstrating the feasibility of real-time minimum-jerk plans discussed in §4.

5.6 Experiment 2: Comparing Optimization Solvers

In our second experiment, we assess the performance of *RelaxedIK* using different optimization solvers, including derivative-based and non-derivative-based varieties. We first present the optimization solvers used in the evaluation before discussing our findings.

5.6.1 Experiment 2 Comparisons

We compared five optimization solvers in our evaluation, all of which are open-sourced and provided by the `NLopt` library.[8] The solvers used were COBYLA (constrained optimization by linear approximation) (Powell, 1994), BOBYQA (bound optimization by quadratic approximation) (Powell, 2009), MMA (method of moving asymptotes) (Svanberg, 2002), CCSAQ (conservative convex separable approximation) (Svanberg, 2002), and SLSQP (sequential least-squares quadratic programming) (Kraft, 1988). The COBYLA and BOBYQA solvers are non-derivative-based, while MMA, CCSAQ, and SLSQP solvers are derivative-based. In this experiment, all gradients for the derivative-based solvers were calculated using automatic-differentiation. For a comparison of different gradient calculation methods, refer to Experiment 3 in §5.7. All of these solvers are local and do not attempt to reach global optimality. An augmented-Lagrangian approach was used to apply unconstrained optimization solvers to constrained problems.

5.6.2 Experiment 2 Results

Our results for Experiment 2 are summarized in Table 2. At a high level, all of the optimization algorithms produced reasonably smooth and feasible results and returned solutions fast enough for real-time use. This highlights that our

[8] NLopt: https://nlopt.readthedocs.io/en/latest/

	Pos. Error	Rot. Error	Joint Velocity	Joint Accel.	Joint. Jerk	Number of Discontinuities	Number of Singularities	Number of Self-Collision	Solution Times
COBYLA	0.0138 ± 0.007	0.0100 ± 0.024	0.0107 ± 0.016	0.0028 ± 0.001	0.0041 ± 0.008	0	0	0	0.0111 ± 0.011
BOBYQA	0.0080 ± 0.007	0.0026 ± 0.003	0.0111 ± 0.012	0.0014 ± 0.003	0.0019 ± 0.001	0	0	0	0.0113 ± 0.009
MMA	0.0044 ± 0.005	0.0016 ± 0.040	0.0108 ± 0.012	0.0004 ± 0.003	0.0002 ± 0.003	0	0	0	0.0089 ± 0.009
CCSAQ	0.0043 ± 0.005	0.0016 ± 0.005	0.0110 ± 0.012	0.0004 ± 0.003	0.0002 ± 0.003	0	0	0	0.0092 ± 0.009
SLSQP	0.0047 ± 0.003	0.0029 ± 0.008	0.0118 ± 0.013	0.0004 ± 0.001	0.0002 ± 0.001	0	0	0	0.0046 ± 0.008

Table 2 Summary of aggregated results from our Experiment 2. The measures are mean position error (meters), mean rotational error (radians), mean joint velocity (rad/s), mean joint acceleration (rad/s^2), mean joint jerk (rad/s^3), total number of joint discontinuities, total number of singularities, total number of self-collision, and mean solution time (seconds). The range value is standard deviation.

method is generalizable and not tied to a particular non-linear constrained-optimization algorithm.

While all of the solvers performed reasonably well in the evaluation, two main points emerge upon further analysis. First, the derivative-based optimization algorithms performed better than the non-derivative-based algorithms. Specifically, we see that the COBYLA solver performs least favorably from the set of solvers, exhibiting higher end-effector errors, less smooth motion, and higher computational cost compared to the alternatives. We believe derivative-based optimization methods performed better because gradients and higher derivative information inherently carry rich information about a robot's motion qualities. To illustrate, consider how the robot's end-effector coordinates change when the robot's joint angles change. In this case, the different axes on the robot arm will have a different magnitude of effect on the robot's end-effector depending on where it is in the chain, *i.e.*, a rotation of the root joint will have a larger relative effect on the change in the robot's end-effector pose than the same angle rotation on the outermost joint in the chain. While this kind of kinematic phenomenon may be difficult for stochastic-based optimization algorithms to infer upon random steps in joint-space, this effect is inherently contained in the second-derivative information of most of our objective terms, with respect to the joint state. Specifically, the second-derivative of these terms can be exactly interpreted as how fast the gradient of the term outputs are changing based on each joint's rotation (this rate of change would be higher for the root joint than the outermost point). Thus, we believe that the derivative-based methods performed more favorably because they naturally maintain this sense of kinematic nuance through the use of gradients and approximated higher-order derivative information, making convergence a more dependable procedure. The second point we observed was that, while MMA and CCSAQ exhibited marginally better results over SLSQP in terms of end-effector pose error and joint smoothness, SLSQP provided comparable performance in about half the computation time. Based on these results, We suggest that SLSQP is the best algorithm to use for *RelaxedIK* in most situations where accuracy, feasibility, and computational efficiency are all considerations. However, in the case where computational efficiency is of lower priority, MMA and CCSAQ may provide motions with slightly improved accuracy and feasibility.

5.7 Experiment 3: Comparing Gradient Calculation Methods

Derivative-based optimization methods generally share a similar strategy: at each update, calculate the gradient of the objective function and move some distance along the gradient in the direction of quickest descent towards a minimum in the function landscape. All solvers within this paradigm depend on the *method* used to calculate gradients, each offering potential tradeoffs in different situations. In this experiment, we assess the performance of *RelaxedIK* using different gradient calculation methods on each of the derivative-based optimization solvers featured in Experiment 2.

As overviewed in Equation 2, our objective function is composed of multiple weighted terms, each encoding a desired motion quality:

$$\mathbf{f}(\Theta) = \sum_{i=1}^{k} w_i \ * f_i(\Theta, \Omega_i)$$

The gradient of the objective function with respect to Θ is:

$$\frac{\partial \mathbf{f}}{\partial \Theta} = \sum_{i=1}^{k} w_i \ * \frac{\partial f_i}{\partial \Theta}$$

The focus of this experiment was to assess if the method for calculating the gradient $\frac{\partial f_i}{\partial \Theta}$ terms has an effect on the overall optimization. We first overview the gradient methods used in the evaluation before discussing our findings.

5.7.1 Experiment 3 Comparisons

We compared two methods for computing gradient terms: automatic differentiation and finite differencing. Automatic differentiation, or AD, is a numerical method that propagates gradient information through all sub-routines needed to compute the value of a function. The final gradient is then constructed using the chain rule once the AD process reaches a base case with a known derivative. In this evaluation, we used the forward-mode AD package called `ForwardDiff` provided in the Julia programming language (Revels et al., 2016).

Finite differencing is a a numerical procedure that approximates a gradient by taking differences between the function at the original query point and other inputs slightly perturbed from the query point. The finite differencing implementation used in our evaluation is from the `Calculus` package provided in the Julia programming language.[9]

We note that the kinematic singularity constraint function discussed in §4.6 above is not able to be differentiated using AD as the gradient information is not able to pass through the singular value decomposition process. Thus, both comparisons in this section use finite differencing to calculate the gradient of this constraint function.

[9] https://github.com/JuliaMath/Calculus.jl

5.7.2 Experiment 3 Results

	Pos. Error	Rot. Error	Joint Velocity	Joint Accel.	Joint. Jerk	Number of Discontinuities	Number of Singularities	Number of Self-Collision	Solution Times
MMA (AD)	0.0044 ± 0.005	0.0016 ± 0.040	0.0108 ± 0.012	0.0004 ± 0.003	0.0002 ± 0.003	0	0	0	0.0089 ± 0.009
MMA (FD)	0.0045 ± 0.005	0.0016 ± 0.041	0.0108 ± 0.012	0.0004 ± 0.003	0.0002 ± 0.003	0	0	0	0.0290 ± 0.010
CCSAQ (AD)	0.0043 ± 0.005	0.0016 ± 0.005	0.0110 ± 0.012	0.0004 ± 0.003	0.0002 ± 0.003	0	0	0	0.0092 ± 0.009
CCSAQ (FD)	0.0044 ± 0.005	0.0017 ± 0.005	0.0110 ± 0.012	0.0004 ± 0.003	0.0002 ± 0.003	0	0	0	0.0279 ± 0.019
SLSQP (AD)	0.0047 ± 0.003	0.0029 ± 0.008	0.0118 ± 0.013	0.0004 ± 0.001	0.0002 ± 0.001	0	0	0	0.0046 ± 0.008
SLSQP (FD)	0.0047 ± 0.003	0.0029 ± 0.009	0.0114 ± 0.013	0.0004 ± 0.001	0.0002 ± 0.003	0	0	0	0.0126 ± 0.006

Table 3 Summary of aggregated results from our Experiment 3. The measures are mean position error (meters), mean rotational error (radians), mean joint velocity (rad/s), mean joint acceleration (rad/s^2), mean joint jerk (rad/s^3), total number of joint discontinuities, total number of singularities, total number of self-collision, and mean solution time (seconds). The range value is standard deviation.

Our results for Experiment 3 are summarized in Table 3. We see that automatic differentiation and finite differencing provide very similar accuracy and feasibility in output motions. While, in theory, automatic differentiation would be expected to provide improved results because of its exactness (accurate down to machine precision), we believe results were similar here because locally optimal solutions are close by in state-space and the small errors induced by finite differencing do not have much room to propagate and propel the solution away from the local minimum.

We observed the automatic differentiation conditions to be 2–3 times faster at finding solutions than the finite differencing conditions. We believe AD is faster because it only needs to take a single pass through the objective function, while finite differencing has to make many calls to the function to make its approximation. Because the `ForwardDiff` package propagates gradient information very efficiently using operator overloading, it requires little computational overhead when performing its single pass through the objective function and results in faster optimization outputs than the finite differencing approach.

5.8 Experiment 4: Comparing Loss Functions

In our fourth experiment, we assess the performance of RelaxedIK using different loss functions in the objective function. As discussed in §4.1, we present a novel loss function, called the "Groove loss", which normalizes the many objective terms into a standard range. This experiment was designed to assess whether the Groove loss exhibits advantages over more standard loss functions. We first overview the loss functions used in the evaluation, then discuss our results.

5.8.1 Experiment 4 Comparisons

In this experiment, our goal was to assess the relative contribution of our Groove loss function on the results presented in the prior experiments. To achieve this, we compared results on our evaluation test bed using our Groove loss function to a more standard quadratic loss function. In contrast to the formulation specified in Equation 3, our objective terms take the following form using just a quadratic loss:

$$f_i(\Theta) = \chi_i(\Theta)^2 \tag{11}$$

We used the same two sets of weighting parameters used in Experiment 1 for both loss functions. Specifically, these weighting parameters are $\{w_p = 50, w_o = 40, w_v = 0.1, w_a = 1, w_j = 2, w_e = 0.1, w_c = 2\}$ for *RelaxedIK A*, which should emphasize motion accuracy, and $\{w_p = 10, w_o = 9, w_v = 5, w_a = 4, w_j = 3, w_e = 2, w_c = 5\}$ for *RelaxedIK B*, which should emphasize motion feasibility and smoothness.

5.8.2 Experiment 4 Results

	Pos. Error	Rot. Error	Joint Velocity	Joint Accel.	Joint. Jerk	Number of Discontinuities	Number of Singularities	Number of Self-Collision	Solution Times
Groove (A)	0.0047 ± 0.003	0.0029 ± 0.008	0.0118 ± 0.013	0.0004 ± 0.001	0.0002 ± 0.001	0	0	0	0.0046 ± 0.008
Groove (B)	0.0118 ± 0.018	0.0284 ± 0.026	0.0101 ± 0.008	0.0001 ± 0.001	0.0001 ± 0.001	0	0	0	0.0046 ± 0.009
Quadratic(A)	0.0884 ± 0.096	0.1361 ± 0.262	0.0359 ± 0.086	0.0182 ± 0.061	0.0229 ± 0.079	79	0	0	0.0067 ± 0.095
Quadratic(B)	0.1001 ± 0.089	0.0168 ± 0.241	0.0114 ± 0.019	0.0018 ± 0.008	0.0020± 0.011	0	0	0	0.0071 ± 0.085

Table 4 Summary of aggregated results from our Experiment 4. The measures are mean position error (meters), mean rotational error (radians), mean joint velocity (rad/s), mean joint acceleration (rad/s^2), mean joint jerk (rad/s^3), total number of joint discontinuities, total number of singularities, total number of self-collision, and mean solution time (seconds). The range value is standard deviation.

Our results for Experiment 4 are summarized in Table 4. At a high level, we observe that the Groove loss versions of *RelaxedIK* far outperform the versions using a standard quadratic loss functions. These results indicate that the Groove loss function structure plays an integral role in the improved results presented in the experiments presented above and serves as a substantial contribution of this work as a whole. We suggest that the Groove loss function may provide similar advantages for any optimization problem where the objective function is structured as a weighted sum with many terms. We show that the Groove loss is able to reconcile many potentially competing objectives and effectively make tradeoffs in such situations.

6 General Discussion

In this paper, we presented a real-time motion-synthesis method for robot manipulators to reliably match end-effector pose goals while considering motion

feasibility objectives on-the-fly. Our contributions included the Groove loss function that both normalizes terms over a standard range and allows motion features to smoothly relax in favor of other more important objectives, introducing a collision avoidance neural network approach for quickly computing a distance to a self-collision state, and demonstrating how to effectively incorporate many motion objectives and constraints into a non-linear optimization framework to effectively synthesize robot motions in real-time. We also showed through many empirical tests that our method performs more favorably than state-of-the-art baselines including direct Trac-IK or real-time motion planning on numerous tasks and robot platforms, and we provided additional details on other components of the method, such as choices of optimization solver, gradient calculation method, and objective term loss functions.

Our method has a number of limitations that suggest many extensions. First, because we rely on a general constrained non-linear optimization formulation, our method provides no guarantees of convergence. We instead provide substantial empirical evidence of the robustness of our method in practice. Additionally, certain guarantees can be achieved by integrating our method as the IK solver in an overall real-time motion-planning framework, thus falling back on the completeness and feasibility guarantees of the motion planner as a backup.

While the current work only explored setting objective term relative importances through defining *static weights* for each term prior to run-time, defining *dynamic weighting functions* for each objective that can adjust relative importances on-the-fly could be beneficial. To illustrate, precise end-effector pose matching would be very important when a sewing robot is threading the needle, but smooth, minimum-jerk joint motion would be more important when the robot is making broad motions to pull the thread through the fabric. While prior works have demonstrated that such dynamic weights work in practice, (Rakita et al., 2017; Shin et al., 2001), extensions to this work could investigate their overall effectiveness and influence on convergence properties.

While the overall framework presented in this paper may generalize to consider force or torque based objectives and constraints, we have not yet explored this possibility and plan to consider dynamics, particularly how exerted forces and moments could fit into our relaxation framework, in our future work. Lastly, while the current implementation of our method is sufficiently fast for real-time use, and is about 4–5 times faster than the original *RelaxedIK* implementation reported on in prior work (Rakita et al., 2018b), it is still marginally slower than standard IK solvers. Although the feasibility benefits may outweigh the cost of the additional computation time in many scenarios, we will pursue ways of speeding up our method so that it can generalize to more domains and more easily work as a subroutine within larger frameworks.

The results presented in this work show that our method works for creating accurate and feasible robot arm motions one frame at a time. We believe that such a per-frame approach for synthesizing accurate and feasible robot arm motions could enable more effective applications that involve robots reacting to external stimuli or generally acting under uncertainty. We will continue to

expand upon this approach and evaluate its efficacy in such reactive scenarios, such as for telemanipulation, shared-control, active vision, or active policy learning.

References

Aristidou A, Lasenby J, Chrysanthou Y, Shamir A (2018) Inverse kinematics techniques in computer graphics: A survey. In: Computer Graphics Forum, Wiley Online Library, vol 37, pp 35–58

Baerlocher P, Boulic R (2004) An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. The visual computer 20(6):402–417

Beeson P, Ames B (2015) TRAC-IK: An open-source library for improved solving of generic inverse kinematics. In: 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), IEEE, pp 928–935

Bialkowski J, Karaman S, Frazzoli E (2011) Massively parallelizing the RRT and the RRT. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, pp 3513–3518

Cefalo M, Oriolo G, Vendittelli M (2013) Planning safe cyclic motions under repetitive task constraints. In: 2013 IEEE International Conference on Robotics and Automation, IEEE, pp 3807–3812

Chiacchio P, Chiaverini S, Sciavicco L, Siciliano B (1991) Closed-loop inverse kinematics schemes for constrained redundant manipulators with task space augmentation and task priority strategy. The International Journal of Robotics Research 10(4):410–425

Chiaverini S (1997) Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. IEEE Transactions on Robotics and Automation 13(3):398–410

Diankov R (2010) Automated construction of robotic manipulation programs. PhD thesis, Carnegie Mellon University, Robotics Institute, URL `http://www.programmingvision.com/rosen_diankov_thesis.pdf`

Flash T, Hogan N (1985) The coordination of arm movements: an experimentally confirmed mathematical model. Journal of neuroscience 5(7):1688–1703

Gosselin C, Angeles J (1990) Singularity analysis of closed-loop kinematic chains. IEEE Transactions on Robotics and Automation 6(3):281–290

Hauser K (2012) On responsiveness, safety, and completeness in real-time motion planning. Autonomous Robots 32(1):35–48

Hauser K (2016) Continuous pseudoinversion of a multivariate function: Application to global redundancy resolution. In: 12th International Workshop on the Algorithmic Foundations of Robotics

Kalakrishnan M, Chitta S, Theodorou E, Pastor P, Schaal S (2011) STOMP: Stochastic trajectory optimization for motion planning. In: 2011 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp 4569–4574

Kavraki LE, Svestka P, Latombe JC, Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE transactions on Robotics and Automation 12(4):566–580

Khatib O (1986) Real-time obstacle avoidance for manipulators and mobile robots. The international journal of robotics research 5(1):90–98

Kraft D (1988) A software package for sequential quadratic programming. Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt

Kuffner JJ, LaValle SM (2000) RRT-connect: An efficient approach to single-query path planning. In: 2000 IEEE International Conference on Robotics and Automation (ICRA), IEEE, vol 2, pp 995–1001

Lee J (2008) Representing rotations and orientations in geometric computing. IEEE Computer Graphics and Applications 28(2):75–83

Maciejewski AA (1990) Dealing with the ill-conditioned equations of motion for articulated figures. IEEE Computer Graphics and Applications 10(3):63–71

Mansard N, Khatib O, Kheddar A (2009a) A unified approach to integrate unilateral constraints in the stack of tasks. IEEE Transactions on Robotics 25(3):670–685

Mansard N, Stasse O, Evrard P, Kheddar A (2009b) A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks. In: 2009 International Conference on Advanced Robotics, IEEE, pp 1–6

Murray S, Floyd-Jones W, Qi Y, Sorin DJ, Konidaris G (2016) Robot motion planning on a chip. In: Robotics: Science and Systems

Nakamura Y (1990) Advanced robotics: redundancy and optimization. Addison-Wesley Longman Publishing Co., Inc.

Oriolo G, Mongillo C (2005) Motion planning for mobile manipulators along given end-effector paths. In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation, IEEE, pp 2154–2160

Oriolo G, Vendittelli M (2009) A control-based approach to task-constrained motion planning. In: 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, pp 297–302

Powell MJ (1994) A direct search optimization method that models the objective and constraint functions by linear interpolation. In: Advances in optimization and numerical analysis, Springer, pp 51–67

Powell MJ (2009) The bobyqa algorithm for bound constrained optimization without derivatives. Cambridge NA Report NA2009/06, University of Cambridge, Cambridge pp 26–46

Praveena P, Rakita D, Mutlu B, Gleicher M (2019) User-guided offline synthesis of robot arm motion from 6-dof paths. In: IEEE International Conference on Robotics and Automation (ICRA), IEEE

Rakita D, Mutlu B, Gleicher M (2017) A motion retargeting method for effective mimicry-based teleoperation of robot arms. In: Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction, ACM, pp 361–370

Rakita D, Mutlu B, Gleicher M (2018a) An autonomous dynamic camera method for effective remote teleoperation. In: Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction, ACM

Rakita D, Mutlu B, Gleicher M (2018b) RelaxedIK: Real-time Synthesis of Accurate and Feasible Robot Arm Motion. In: Proceedings of Robotics: Science and Systems, Pittsburgh, Pennsylvania, DOI 10.15607/RSS.2018.XIV.043

Rakita D, Mutlu B, Gleicher M, Hiatt LM (2018c) Shared dynamic curves: A shared-control telemanipulation method for motor task training. In: Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction, ACM

Rakita D, Mutlu B, Gleicher M (2019a) Remote telemanipulation with adapting viewpoints in visually complex environments. In: Proceedings of Robotics: Science and Systems, FreiburgimBreisgau, Germany, DOI 10.15607/RSS.2019.XV.068

Rakita D, Mutlu B, Gleicher M (2019b) Stampede: A discrete-optimization method for solving pathwise-inverse kinematics. In: IEEE International Conference on Robotics and Automation (ICRA), IEEE

Rakita D, Mutlu B, Gleicher M, Hiatt LM (2019c) Shared control–based bimanual robot manipulation. Science Robotics 4(30):eaaw0955

Ratliff N, Zucker M, Bagnell JA, Srinivasa S (2009) CHOMP: Gradient optimization techniques for efficient motion planning. In: 2009 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp 489–494

Revels J, Lubin M, Papamarkou T (2016) Forward-mode automatic differentiation in julia. arXiv:160707892 [csMS] URL `https://arxiv.org/abs/1607.07892`

Schulman J, Duan Y, Ho J, Lee A, Awwal I, Bradlow H, Pan J, Patil S, Goldberg K, Abbeel P (2014) Motion planning with sequential convex optimization and convex collision checking. The International Journal of Robotics Research 33(9):1251–1270

Sentis L, Khatib O (2005) Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. International Journal of Humanoid Robotics 2(04):505–518

Shin HJ, Lee J, Shin SY, Gleicher M (2001) Computer puppetry: An importance-based approach. ACM Transactions on Graphics (TOG) 20(2):67–94

Siciliano B (1990) Kinematic control of redundant robot manipulators: A tutorial. Journal of Intelligent & Robotic Systems 3(3):201–212

Sina Mirrazavi Salehian S, Figueroa N, Billard A (2016) Coordinated multi-arm motion planning: Reaching for moving objects in the face of uncertainty. In: Proceedings of Robotics: Science and Systems

Sucan IA, Moll M, Kavraki LE (2012) The open motion planning library. IEEE Robotics & Automation Magazine 19(4):72–82

Svanberg K (2002) A class of globally convergent optimization methods based on conservative convex separable approximations. SIAM journal on optimization 12(2):555–573

Yoshikawa T (1985) Manipulability of robotic mechanisms. The International Journal of Robotics Research 4(2):3–9