CollisionIK: A Per-Instant Pose Optimization Method for Generating Robot Motions with Environment Collision Avoidance

Daniel Rakita, Haochen Shi, Bilge Mutlu, Michael Gleicher
Department of Computer Sciences, University of Wisconsin–Madison
{rakita,hshi7,bilge,gleicher}@cs.wisc.edu

***Abstract*— In this work, we present a per-instant pose optimization method that can generate configurations that achieve specified pose or motion objectives as best as possible over a sequence of solutions, while also simultaneously avoiding collisions with static or dynamic obstacles in the environment. We cast our method as a multi-objective, non-linear constrained optimization-based IK problem where each term in the objective function encodes a particular pose objective. We demonstrate how to effectively incorporate environment collision avoidance as a single term in this multi-objective, optimization-based IK structure, and provide solutions for how to spatially represent and organize external environments such that data can be efficiently passed to a real-time, performance-critical optimization loop. We demonstrate the effectiveness of our method by comparing it to various state-of-the-art methods in a testbed of simulation experiments and discuss the implications of our work based on our results.**

I. Introduction

Optimizing individual poses to achieve some objective is a common technique in robotics. For example, inverse kinematics (IK) is one instance of single pose optimization that involves optimizing a robot's joint-angle pose to match an input 6-DOF end-effector position and orientation goal. Single pose optimization has many practical benefits, such as its speed and comprehensible "single-input-to-single-output" nature. However, this paradigm is often insufficient when used iteratively to generate motions over a sequence of poses, a paradigm historically referred to as *per-frame IK* in animation [1], [2]. Problems with per-frame (or, per-instant) optimization often stem from the stream of independent poses lacking *temporal coherence*, *e.g.*, jerky motion or joint space discontinuities, or lacking *motion feasibility*, *e.g.*, self-collisions or collisions with obstacles in the environment. Thus, despite the speed and convenience of per-instant pose optimization, its shortcomings mean that larger motion planning or trajectory optimization frameworks are often needed in order to achieve coherent and feasible motions.

In this work, we present a per-instant pose optimization method, called *CollisionIK*, that optimizes single poses at given time points that achieve certain accuracy objectives as best as possible, *e.g.,* matching end-effector pose goals, without sacrificing temporal coherence and motion feasibility on a sequence of solutions. In particular, our method features environment collision avoidance as a sub-component, meaning that feasible motions that avoid collisions with static or dynamic obstacles can be generated on-the-fly in a per-instant manner without requiring a motion planner or trajectory optimizer. Our method can also incorporate joint smoothness objectives that use a short history of previous solutions to optimize over approximated derivatives. This per-instant paradigm, where smooth motions are iteratively constructed on-the-fly, effectively allows the robot to react and adapt its motion to complex or dynamic environments, all while still reflecting other objectives as best as possible.

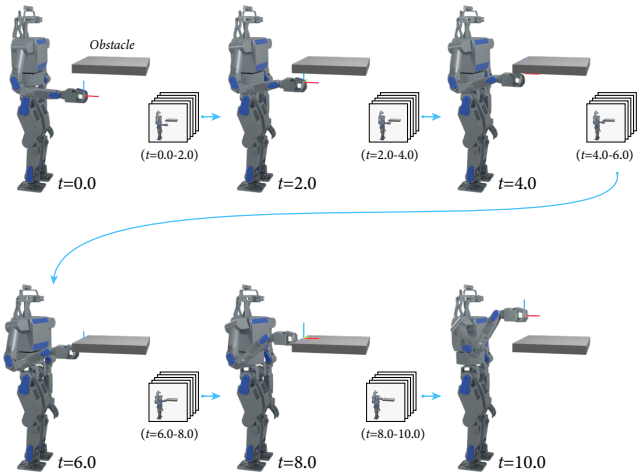

Fig. 1. In this work, we present a per-instant pose optimization method that is able to generate smooth, feasible paths on-the-fly without the use of a planner or trajectory optimizer. This example shows a robot avoiding a collision with a table while maintaining the same end-effector orientation throughout the motion.

We cast our method as a multi-objective, non-linear constrained optimization-based IK problem. The objective function is a weighted sum, where each term in the sum encodes a particular motion objective. The weights on the terms in the objective function set importances of the various terms and allows the optimization solver to *relax* certain objectives in favor of other, more important, terms in the case of competing objectives. In particular, our method favors motion feasibility objectives, such as avoiding collisions, over other objectives, such as matching end-effector pose goals. Throughout this work, we overview the structure of this overall optimization framework and highlight how this or similar frameworks extend well to the case of environment collision avoidance.

Our current work offers two technical contributions: (1) we demonstrate how to effectively incorporate environment collision avoidance as a single term in a multi-objective, optimization-based IK structure (§IV-B); (2) we provide initial solutions for how to spatially represent and organize

external environments such that data can be efficiently passed to a real-time, performance critical optimization loop (§IV-A, §IV-C); and (3) we provide open-source code that implements our proposed method: [*link will appear here*].

We assessed the efficacy of our method by running a testbed of simulation experiments (§V). We compared our method to the MoveIt! library Cartesian path controller [3] and *RelaxedIK* [4] on various simulated robots and tasks. We demonstrate that our method successfully avoids collisions with static or dynamic objects in the environment in real-time while consistently achieving additional motion objectives. Our evaluation also shows that our method scales well and maintains its efficient performance even in environments with many obstacles. We discuss the benefits and drawbacks of our method compared to the alternative approaches, such as speed, local minima, and motion accuracy trade-offs, and conclude with an overall discussion about the implications of our work based on our results (§VI).

II. Related Works

In this section, we highlight prior works that our method draws inspiration from in the areas of motion-planning, trajectory optimization, animation, and inverse kinematics.

Motion Planning— Generating robot motions that avoid collisions with obstacles in an environment is commonly addressed with a technique called *motion planning* [5]. This approach finds collision-free, feasible paths from a start state to a goal state in configuration space. Sampling-based motion planners often use random samples to bootstrap a search strategy and build a graph structure from start to goal [6]–[9]. Such planners are commonly guaranteed to find a solution if one exists, and some variants, such as RRT* [10], are also guaranteed to find the *shortest* feasible path in the limit.

While standard planners are effective at eventually finding collision-free solution paths, provided one exists, they are less adept at consistently finding sufficient solutions in a time-sensitive, real-time setting. Variants of these approaches have focused on the real-time aspects of this problem, such as Kroger et al. [11], who presented a real-time planning algorithm that allowed robots to avoid dynamic obstacles in real-time. Hauser [12] proposed an adaptive method to adjust a planning horizon time such that prediction of a future state that the robot will likely move toward and planning to said future state can be interleaved in a stable manner. Also, work by Murray et al. [13] accelerated road-map based path planning by creating custom hardware computer chips that check collision states in parallel.

As mentioned above, motion planning algorithms are effective at getting from a start point to a goal point, but it is difficult for these approaches to enforce what the path does between these boundary points. In contrast, our method tries to achieve any other motion objectives as best as possible on-the-fly in addition to environment collision avoidance, such as end-effector pose matching over time.

Trajectory Optimization— Trajectory optimization is an approach used to optimize motions to match desired motion qualities (see Betts [14] for a review). Trajectory optimization methods for robot motion, such as CHOMP [15], STOMP [16], and Trajopt [17] include environment collision avoidance techniques; however, the quality and convergence of the computed motion paths using these methods greatly depends on the quality of the initial condition. Further, these methods often formulate their environment collision-avoidance objective with respect to a pre-computed signed distance field of the environment, which is infeasible to routinely re-compute and update on-the-fly. Thus, these methods are generally not well suited for real-time, dynamic environments. In contrast, our method not only accommodates arbitrary motion objectives and constraints, but also accommodate real-time, dynamic environment collision avoidance, at the expense of global optimality.

Inverse kinematics— The process of calculating joint 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 [18] or Nakamura [19] for a full review). A main objective of IK solvers is to reliably match a given end-effector pose goal as quickly as possible. A state-of-the-art solver to achieve this central goal on is the optimization-based Trac-IK solver proposed by Beeson and Ames [20]. Recent work has showed the benefit of incorporating motion *feasibility* objectives, such as self-collision avoidance and singularity avoidance, in addition to the standard motion accuracy objectives mentioned above [4]. This was shown to be particularly effective for practical use cases such as teleoperation or shared control [21]–[23]. Our current work builds on this concept of providing both motion feasibility and accuracy in real-time inverse kinematics, though we are attempting to extend the definition of motion feasibility to include the avoidance of environment collisions in addition to kinematic singularities, self-collisions, and joint-space discontinuities.

III. Technical Overview

In this section, we provide background and notation for our problem, and overview our overall method.

A. Notation and Problem Statement

Suppose $\Theta \in \mathbb{R}^{\mathrm{n}}$ is an n-dimensional robot configuration. Next, consider $\mathbf{f}(t, \Theta)$ to be an *objective function* that maps a time value t and a robot configuration to some scalar objective output value, $f \in \mathbb{R}$. Note that this implies that the robot's objective can change with respect to time, t. The output of the objective function, f, signifies how well the joint configuration Θ reflects the objective specified by $\mathbf{f}$ at time t. Lastly, consider a set E consisting of obstacles in the environment. Each of the K obstacles in E will be considered a function $\mathbf{e}_{\mathrm{k}}(t)$ that maps time t to spatial information about the obstacle. This spatial information could take many forms, such as a signed distance field, occupancy map, triangulated mesh, etc. Regardless of spatial representation choice, we assume that there is some defined notion of distance between a robot configuration and an obstacle at time t, which we will denote as $\mathrm{d}(\Theta, \mathbf{e}_{\mathrm{k}}(t))$. We intentionally leave these definitions

as quite general in this section, and we detail the exact environment spatial representation and definition of distance we used in our method in §IV-A.

Using the notation above, the problem investigated in this work is to compute a joint configuration Θ at a given time t that minimizes $\mathbf{f}(t, \Theta)$, that may include sub-objectives like joint smoothness or matching end-effector pose goals, while maintaining a distance $\mathrm{d}(\Theta, \mathbf{e}_{\mathrm{k}}(t)) > \epsilon, \ \forall \mathrm{k}$, where ϵ is some reasonable cut-off distance between collision and non-collision. Additionally, the objective $\mathbf{f}$ at time $t+\delta$ and the spatial information about each obstacle at time $t+\delta$, for any $\delta > 0$, are unknown at time t. We also want to solve the above problem as fast as a possible since a fast run-time is essential for many applications.

B. Non-linear Optimization Structure

We cast the minimization problem posed above as a constrained non-linear optimization problem:

$$\begin{aligned} \Theta = \arg\min_{\Theta} \quad & \mathbf{f}(t, \Theta) \\ \text{s.t. } \mathbf{c}(\Theta) \ \geq \ & \mathbf{0}, \quad \mathrm{l_i} \leq \Theta_{\mathrm{i}} \leq \mathrm{u_i}, \forall \mathrm{i} \end{aligned} \tag{1}$$

Here, $\mathbf{c}$ is a set of inequality constraints and $\mathrm{l_i}$ and $\mathrm{u_i}$ values define the upper and lower bounds for the robot's joints.

We express our objective function as a weighted sum of individual motion goals as follows:

$$\mathbf{f}(t, \Theta) = \sum_{\mathrm{j}=1}^{\mathrm{J}} \mathrm{w_j} * \mathrm{f_j}(t, \Theta, \Omega_{\mathrm{j}}) \tag{2}$$

Here, $\mathrm{w_j}$ is a weight value for each term which sets an importance for a given objective term, and $\mathrm{f_j}$ is an objective term function that encodes a single sub-goal, with Ω_{j} being model parameters used to construct some loss function.

To facilitate combining of potentially many objectives, it is important to normalize each term such that their outputs are over a uniform range. For instance, a term with weight 2 will ideally hold twice as much importance in the optimization as a term with weight 1, regardless of the common output ranges of the two terms. To accomplish this normalization, we use the Groove parametric loss function proposed in prior work [4], [24], though any loss function that achieves effective normalization of multiple objectives should suffice. The Groove loss 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 an optimal point. This normalization loss is a Gaussian surrounded by a more gradual polynomial:

$$\begin{aligned} & \mathrm{f_j}(t, \Theta, \Omega_{\mathrm{j}}) = \\ & (-1)^n exp(\frac{-(\chi_{\mathrm{i}}(t, \Theta) - s)^2}{2c^2}) \ + r * (\chi_{\mathrm{i}}(t, \Theta) - s)^4 \end{aligned} \tag{3}$$

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.

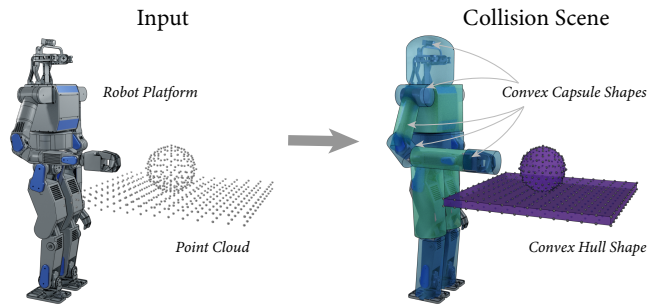

Fig. 2. (Left) Our method takes as input a robot platform and set of any collision objects as point clouds. (Right) Our method converts these inputs into a Collision Scene, comprised of convex capsule shapes around the robot links and convex hull shapes enveloping the point cloud objects.

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, higher values showing a steeper funneling into the Gaussian region and lower values flattening out the boundaries beyond the Gaussian. The scalar function χ assigns a numerical value to the current robot configuration that will serve as input to the loss function.

Our method includes seven objective terms and one constraint in the non-linear optimization structure above by default. The default objective terms encode: (1) end-effector position goal matching; (2) end-effector orientation goal matching; (3) minimized joint velocity; (4) minimized joint acceleration; (5) minimized joint jerk; (6) self-collision avoidance; and (7) environment collision avoidance. The one constraint is designed to avoid kinematic singularities. We model objectives 1–6 and the constraint based on prior work [4]. Derivative information about velocities, accelerations, and jerks are approximated using finite differencing over a short history of prior poses. We provide details on how we structure and pass spatial information into the environment collision avoidance objective in the following section. We note that this overall structure is modular and any of the above objectives or constraints can be removed and any additional objectives or constraints can be accommodated.

IV. Technical Details

In this section, we provide details on how we represent spatial information in our method and how we structure our collision avoidance objective that uses this information.

A. Environment Representation and Structure

As mentioned above, each collision function, $\mathbf{e}_{\mathrm{k}}(t)$, maps time to some spatial information about the collision object at that time. At a high level, our goal is to use convex shape representations that are fast and scalable for computing distances between collision objects and the robot's links. This technique that has been shown to be effective in many robotics and graphics applications [17], [25], [26], and we extend these approaches to integrate them within a per-instance optimization. Our method takes four steps to achieve this goal: (1) Each collision object is input as a point cloud representation; (2) Each point cloud is converted into a

convex hull object using the QuickHull algorithm [27]. If a particular collision object is not well represented as a convex hull, a decomposition algorithm could break down the object into convex sub-components [28], [29]; (3) Each convex hull object is updated at each given time t using some rigid transformation. This means that the $\mathbf{e}_k(t)$ mappings represent rigidly transformed convex hull shapes at time t; and (4) Each link of the robot is automatically wrapped in a convex shape (a capsule, by default). Each of these link objects is also updated at each given time t based on the robot's forward kinematics model associated with its joint state at that time. We refer to a function that maps t to the rigidly transformed convex shape wrapped around the n-th link as $\mathbf{l}_n(t, \Theta)$.

We refer to the collection of all all convex hull collision objects and all robot link convex shapes as the *Collision Scene* (illustrated in Figure 2). In the following section, we overview how we use this representation to compute the robot's distance to a collision state at any given time.

B. Environment Collision Avoidance Objective Term

In order to discourage the robot from colliding with the environment, it is necessary to define of what it means for the robot to be "close" to an obstacle. We encode a distance to a collision state using the following cost function:

$$\chi_c(t, \Theta(t)) = \sum_{\mathbf{e_k} \in \mathcal{A}} \sum_{n}^{N} \frac{(5\epsilon)^2}{dis(\mathbf{e}_k(t), \mathbf{l}_n(t, \Theta))^2} \tag{4}$$

Here, ϵ is some scalar value that signifies a reasonable cutoff distance between collision and non-collision. For example, in our prototype system, we use a value of $\epsilon = 0.02$ (represented in meters). The *dis* function computes the shortest straight-line distance between the input shapes, *i.e.*, a collision object and one of the robot's links. We compute the *dis* function using a Support Mapping computation, as this is an efficient way to find the shortest distance between two convex shapes [30]. Lastly, $\mathcal{A}$ is a set of "active" collision objects at the given time, t. Calculating this sum over just a subset of all collision objects maintains the efficiency and scalability of this computation when many obstacles are present in the Collision Scene. We overview how we filter the Collision Scene to select a subset of salient collision objects in the following section.

The cost function in Equation 4 was designed to be smooth and differentiable, as opposed to, for example, taking the minimum distance, such that it effectively mixes into a multi-objective, gradient-based optimization structure. We incorporate the cost function in Equation 4 into our optimization framework as a single objective term using the loss function in Equation 3. In our prototytpe system, we used Groove loss parameters of $n = 1$, $s = 0$, $c = 2.5$, and $r = 0.0035$. These values were selected to reflect the standard output range of the cost function. In particular, these loss function parameters ensure that this term's objective output significantly ramps up when the robot approaches a distance ϵ from an obstacle. Note that the cost function output is high when the robot is close to collision, thus the goal Gaussian region is negative in the loss function.

C. Collision Scene Filtering

As mentioned above, we only compute Equation 4 over a set of "active" obstacles in set $\mathcal{A}$. Filtering the obstacles and only considering a subset at any given time prevents the collision avoidance objective term from becoming too slow, especially in the presence of many environment obstacles. At every given time t, our method starts with all collision objects set as active and prunes this set based on two criteria: (1) All obstacles with a distance greater than some margin, Υ, from all of the robot's links at a given time will be removed from the active set; and (2) If $|\mathcal{A}| > N$ after step 1, only the N obstacles that have the highest cost according to Equation 4 are active. All other obstacles are removed from $\mathcal{A}$.

Our method efficiently achieves criterion 1 above by using a broad-phase and narrow-phase collision detection pipeline. The broad-phase step uses axis-aligned bounding box (AABB) hierarchies to quickly disregard obstacles that are guaranteed to not be within a distance of Υ. This phase uses a Dynamic Bounding Volume Tree to efficiently store, update, and query collision information, even in a dynamic environment. The narrow-phase performs ground truth distance checking only on the obstacles that were not culled by the broad-phase. Our prototype system uses an Υ margin distance of one meter, and sets the maximum number of active obstacles, N, as three.

V. Evaluation

In this section, we overview our experiments designed to assess the efficacy of our method.

A. Implementation Details

A prototype implementation of our method was configured as an extension of the Rust version of the *RelaxedIK* library[1]. Spatial data structures relating to the broad and narrow phase collision checking and convex hull conversions all use the `ncollide3d` library[2]. Our method uses the proximal averaged Newton-type Method (PANOC) optimization approach [31], implemented in the Rust Optimization Engine (OpEN) library. Our approach also works well with a variety of non-linear solvers, such as those offered by NLopt. All gradients in our work were computed using finite differencing. All evaluations were run on an AMD Ryzen 7 2700X Processor (3.70GHz) with 16GB RAM.

B. Evaluation Benchmark

We developed a set of three benchmark tasks to compare our method against alternative approaches. We will refer to these tasks as *Around Table*, *Square Tracing*, and *Isolated Rotations*.

The *Around Table* task involves the robot's end-effector being driven toward and through a table surface. The robot's goal in this task is to avoid colliding with the table and

[1] https://github.com/uwgraphics/relaxed_ik
[2] https://ncollide.org/

		# Env. Collisions	Mean Pos. Error (M)	Mean Rot. Error (Rads.)	Mean Velocity (Rads. / S)	Mean Accel. (Rads. / S^2)	Mean Jerk (Rads. / S^3)	# Self-Collisions	# Kinematic Singularities
Around Table	*CIK*	0	0.106 +- 0.065	0.005 +- 0.007	0.277 +- 0.173	0.012 +- 0.009	0.004 +- 0.004	0	0
	RIK	1365	0.038 +- 0.051	0.004 +- 0.006	0.219 +- 0.129	0.003 +- 0.005	0.002 +- 0.003	0	0
	MCC	26	0.216 +- 0.087	0.67 +- 0.386	1.533 +- 0.98	0.256 +- 0.46	0.511 +- 0.919	0	1
Square Tracing	*CIK*	0	0.101 +- 0.035	0.007 +- 0.003	0.588 +- 0.101	0.031 +- 0.02	0.02 +- 0.024	0	0
	RIK	2309	0.023 +- 0.016	0.006 +- 0.005	0.583 +- 0.12	0.013 +- 0.009	0.008 +- 0.009	0	0
	MCC	559	0.299 +- 0.028	0.954 +- 0.302	2.291 +- 0.761	0.476 +- 0.601	0.95 +- 1.203	0	5
Isolated Rotations	*CIK*	0	0.122 +- 0.079	0.063 +- 0.086	1.942 +- 0.262	0.096 +- 0.032	0.059 +- 0.055	0	0
	RIK	1202	0.057 +- 0.035	0.05 +- 0.057	1.997 +- 0.161	0.092 +- 0.024	0.045 +- 0.039	0	0
	MCC	2033	0.202 +- 0.101	1.477 +- 0.704	2.867 +- 1.537	1.22 +- 2.236	2.438 +- 4.474	0	9

Fig. 3. Results from Experiment 1. Range values represent standard deviation.

continue following the goal trajectory on the other side. The *Square Tracing* task involves the robot's end-effector tracing a square shape. A dynamic cube object approaches the robot on its way around the square, and the robot's goal is to avoid colliding with the cube and continue to follow the perimeter of the square as best as possible. Lastly, the *Isolated Rotations* task involves the robot's end-effector position remaining static, and the robot rotates its end-effector 90 degrees around all of its primary axes in sequence. While these rotations are happening, three dynamic sphere objects in the environment continuously encroach on the robot's space, and the goal is to avoid colliding with these spheres while still matching the specified end-effector rotations and static position point as best as possible.

The tasks listed above were all run on five simulated robots: a Universal Robots UR5 (6-DOF), a Rethink Robotics Sawyer (7-DOF), a Kinova Jaco (7-DOF), a Kuka IIWA (7-DOF), and the torso and right arm of the Rainbow Robotics Hubo+ (8-DOF). Each task was run 10 times per robot for every condition in our benchmark.

C. Evaluation Measures

We assessed eight primary measures in 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 kinematic singularities, total number of self-collisions, and total number of environment collisions.

D. Experiment 1: Comparisons with Alternative Approaches

In our first experiment, we compared our method to two alternative approaches: (1) *RelaxedIK* [4] (which we will refer to as *RIK*); and (2) the MoveIt Cartesian Controller [3] (which we will refer to as *MCC*). The *RIK* condition uses the open-source Rust version of RelaxedIK. It includes many objective terms in its objective function, such as end-effector pose matching, smooth joint motion, and self-collision avoidance, but does not include the environment collision avoidance methods discussed in this work.

The *MCC* controller uses the open-source MoveIt library. It involves three steps: (1) Compute a configuration that matches a desired end-effector pose goal using Trac-IK [20]; (2) Compute a feasible, collision-free path to the configuration found in step 1 using the RRT-Connect planner [9]; and (3) Move along the path found in step 2, and return to step 1 once the robot either completes the path or hits a dynamic collision along the path. If the planner ever does not find a feasible path to the given configuration in step 2, the robot maintains its current configuration, time advances some fixed step, and the approach goes back to step 1. In our evaluation, we "pause" the time parameter t on the trajectory and obstacles during step 2. Thus, this allows the planner to be "infinitely fast" from the perspective of the input trajectory and environment. We made this evaluative decision because our goal was to assess the general *MCC* approach under ideal conditions rather than potentially show that one implementation of one particular planner is too slow to keep up with the real-time demands of our benchmark.

Our results from Experiment 1 are summarized in Figure 3. We see that our method avoids collisions with the environment more effectively than both *RIK* and *MCC*. Also, our method achieves smoother motion and reflects the given end-effector pose goals more precisely than *MCC*. While *RIK* does avoid self-collisions and kinematic singularities (as reported in prior work), and reflects lower end-effector pose errors than our method, this extra accuracy comes at the high cost of many collisions with the environment.

E. Experiment 2: Local Minimum Test

In our second experiment, our goal was to assess if our method is prone to getting stuck in local minimum regions. This assessment involved moving the table closer to the robot in the *Around Table* task, such that the end-effector was guided more through the center of the table rather than close to the edge. We compared our method to the *MCC* approach described above on this task.

While our method still avoids colliding with the table, it does not ultimately find a way to get around the table to the other side. Thus, we see that our method is prone to falling into local minima, while other global planners do not get stuck in such regions. We note that while *MCC* did find a way around the table, this came at the expense of considerably high end-effector position and rotation errors. A graph of end-effector translation errors of the two conditions can be seen in Figure 5.

F. Experiment 3: Assessment of Objective Term Variants

In our third experiment, we compare results on our evaluation benchmark with adjusted objectives. This experiment was designed to show the flexibility of our approach in terms of accommodating different or dynamically changing objectives. Our first condition, *CIK*, uses the same position and orientation matching objectives as Experiments 1 and

		# Env. Collisions	Mean Pos. Error (M)	Mean Rot. Error (Rads.)	Mean Velocity (Rads. / S)	Mean Accel. (Rads. / S^2)	Mean Jerk (Rads. / S^3)	# Self-Collisions	# Kinematic Singularities
Around Table	*CIK*	0	0.106 +- 0.065	0.005 +- 0.007	0.277 +- 0.173	0.012 +- 0.009	0.004 +- 0.004	0	0
	CIK-3	0	0.086 +- 0.063	0.897 +- 0.669	0.238 +- 0.123	0.021 +- 0.012	0.021 +- 0.012	0	0
	CIK-A	0	0.076 +- 0.037	0.217 +- 0.127	0.267 +- 0.078	0.021 +- 0.012	0.014 +- 0.01	0	0
Square Tracing	*CIK*	0	0.101 +- 0.035	0.007 +- 0.003	0.588 +- 0.101	0.031 +- 0.02	0.02 +- 0.024	0	0
	CIK-3	0	0.052 +- 0.047	1.712 +- 0.438	0.585 +- 0.119	0.038 +- 0.012	0.03 +- 0.008	0	0
	CIK-A	0	0.061 +- 0.019	0.521 +- 0.313	0.699 +- 0.133	0.04 +- 0.014	0.032 +- 0.017	0	0
Isolated Rotations	*CIK*	0	0.122 +- 0.079	0.063 +- 0.086	1.942 +- 0.262	0.096 +- 0.032	0.059 +- 0.055	0	0
	CIK-3	0	0.062 +- 0.046	1.962 +- 0.699	0.572 +- 0.178	0.048 +- 0.041	0.045 +- 0.046	0	0
	CIK-A	0	0.108 +- 0.117	1.096 +- 0.811	1.095 +- 0.133	0.077 +- 0.059	0.071 +- 0.079	0	0

Fig. 4. Results from Experiment 3. Range values represent standard deviation.

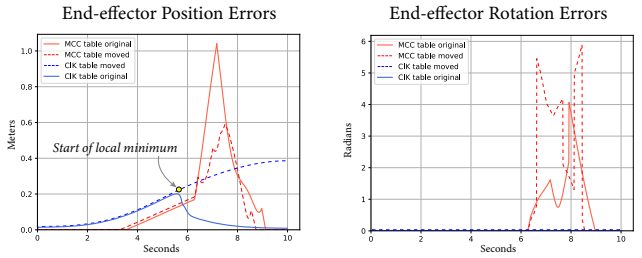

Fig. 5. End-effector position and rotation errors from our local minimum experiment. (Left) We see that our method falls into a local minimum when the table is moved closer to the robot. While *MCC* gets around the table using its global planning capibilities, this comes at the cost of high end-effector position and rotation errors.

2 above. Our second condition, *CIK-3*, only considers end-effector position goal matching, and does not try to match the orientation of goals as well. Lastly, our third condition, *CIK-A*, adaptively adjusts the weight on the orientation matching objectives on-the-fly, such that the importance of orientation matching is reduced when the robot is close to a collision state and raised to its standard value when the robot is not close to a collision.

Our results from Experiment 3 can be seen in Figure 4. We see that all of the evaluated variants of our method achieve their respective objectives, all while avoiding collisions with the environment. For example *CIK-3* reliably matches end-effector position goals, though it has a very high orientation matching error, while *CIK-A* achieves some orientation matching accuracy while still achieving higher positional accuracy than the baseline *CIK*. This demonstrates that our method is able to reflect a variety of different objectives, even those that dynamically change in real-time.

G. Experiment 4: Performance and Scalability Testing

In our final experiment, we tested the performance and scalability of our method in a real-time control setting. This experiment involved a user interactively driving the end-effector position and orientation of a simulated Sawyer robot in a ROS RViz environment using a keyboard controller. Obstacles placed in the environment were also able to be interactively moved by the user. Our ability to test our method on a real robot using a more robust control interface was hindered by the SARS-CoV2 pandemic.

Our first interactive test environment involved four sphere obstacles placed around a simulated robot. Our method avoided collisions with all obstacles during this test, and average run-time performance for this environment was approximately 500 microseconds per solve (2,000 Hz). Our second test environment was designed to test the scalability of our method, and involved 100 Stanford bunnies, each with over 30,000 vertices, around the environment. Each Stanford bunny took 18 milliseconds on average to compute a convex hull using the QuickHull algorithm. Our method again avoided collisions with all obstacles during this test, and average run-time performance for this environment was approximately 16 milliseconds per solve (60 Hz).

VI. Discussion

In this work, we presented a method for generating robot motions on-the-fly that avoid collisions with static or dynamic obstacles in an environment, while also simultaneously accommodating any other motion objectives. In this section, we discuss limitations and implications of our work.

A. Limitations

We note a number of limitations of our work that suggest future extensions. First, our method is prone to becoming trapped in local minima regions, as seen in Experiment 2. Further investigation is required to characterize when our method is susceptible to falling into local minima, and extensions of our work could explore ways to infuse more global path planning techniques within our optimization-based structure to get out of these minima [5]. Also, our work was tested in simulation with synthetic data and does not consider the challenges of real-world sensing or noisy data. Extensions of our work could explore how to efficiently accommodate sensed data form the environment and how to incorporate confidence bounds into our objectives in the case of noisy sensors.

B. Implications

We believe that our method could benefit various areas of robotics, such as teleoperation, shared-control, and reinforcement learning. For example, teleoperation and shared-control interfaces could utilize our method in home health-care, telenursing, or nuclear materials handling applications to mitigate collisions while still achieving other goals as best as possible. Also, unsupervised or semi-supervised learning agents could explore and manipulate their environments without high risk of collisions when forming their policies. We plan to explore these directions in future work.

References

[1] J. Lee and S. Y. Shin, "A hierarchical approach to interactive motion editing for human-like figures," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 1999, pp. 39–48.

[2] M. Gleicher, "Comparing constraint-based motion editing methods," *Graphical models*, vol. 63, no. 2, pp. 107–134, 2001.

[3] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ros topics]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[4] D. Rakita, B. Mutlu, and M. Gleicher, "RelaxedIK: Real-time Synthesis of Accurate and Feasible Robot Arm Motion," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[5] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[6] ——, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[7] L. Kavraki, P. Svestka, and M. H. Overmars, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*. Unknown Publisher, 1994, vol. 1994.

[8] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *Proceedings of International Conference on Robotics and Automation*, vol. 3. IEEE, 1997, pp. 2719–2726.

[9] J. J. Kuffner Jr and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *ICRA*, vol. 2, 2000.

[10] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[11] T. Kröger and F. M. Wahl, "Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 94–111, 2009.

[12] K. Hauser, "On responsiveness, safety, and completeness in real-time motion planning," *Autonomous Robots*, vol. 32, no. 1, pp. 35–48, 2012.

[13] S. Murray, W. Floyd-Jones, Y. Qi, D. J. Sorin, and G. Konidaris, "Robot motion planning on a chip." in *Robotics: Science and Systems*, 2016.

[14] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of guidance, control, and dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[15] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," 2009.

[16] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 4569–4574.

[17] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization." in *Robotics: science and systems*, vol. 9, no. 1. Citeseer, 2013, pp. 1–10.

[18] A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir, "Inverse kinematics techniques in computer graphics: A survey," in *Computer Graphics Forum*, vol. 37, no. 6. Wiley Online Library, 2018, pp. 35–58.

[19] Y. Nakamura, *Advanced robotics: redundancy and optimization*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[20] P. Beeson and B. Ames, "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, 2015, pp. 928–935.

[21] D. Rakita, B. Mutlu, and M. Gleicher, "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, 2017, pp. 361–370.

[22] D. Rakita, B. Mutlu, M. Gleicher, and L. M. Hiatt, "Shared control–based bimanual robot manipulation," *Science Robotics*, vol. 4, no. 30, 2019.

[23] D. Rakita, B. Mutlu, and M. Gleicher, "An autonomous dynamic camera method for effective remote teleoperation," in *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*. ACM, 2018, pp. 325–333.

[24] ——, "An analysis of relaxedik: an optimization-based framework for generating accurate and feasible robot arm motions," *Autonomous Robots*, vol. 44, no. 7, pp. 1341–1358, 2020.

[25] J. Pan, S. Chitta, and D. Manocha, "Fcl: A general purpose library for collision and proximity queries," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 3859–3866.

[26] S. A. Ehmann and M. C. Lin, "Accurate and fast proximity queries between polyhedra using convex surface decomposition," in *Computer Graphics Forum*, vol. 20, no. 3. Wiley Online Library, 2001, pp. 500–511.

[27] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software (TOMS)*, vol. 22, no. 4, pp. 469–483, 1996.

[28] T. Larsson and T. Akenine-Möller, "A dynamic bounding volume hierarchy for generalized collision detection," *Computers & Graphics*, vol. 30, no. 3, pp. 450–459, 2006.

[29] J.-M. Lien and N. M. Amato, "Approximate convex decomposition of polygons," *Computational Geometry*, vol. 35, no. 1-2, pp. 100–123, 2006.

[30] B. Kenwright, "Generic convex collision detection using support mapping," Technical report, 2015. URL https://www. semanticscholar. org/paper/Generic . . . , Tech. Rep., 2015.

[31] L. Stella, A. Themelis, P. Sopasakis, and P. Patrinos, "A simple and efficient algorithm for nonlinear model predictive control," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 1939–1944.