Auton Robot (2021) manuscript No.
(will be inserted by the editor)

Analysis and Acceleration of TORM: Optimization-based Planning for Path-wise Inverse Kinematics

Mincheul Kang[1] · **Sung-Eui Yoon**[1]

the date of receipt and acceptance should be inserted later

Abstract A redundant manipulator can have many trajectories for joints that follow a given end-effector path in the Cartesian space, since it has multiple inverse kinematics solutions per end-effector pose. While maintaining accuracy with the given end-effector path, it is challenging to quickly synthesize a feasible trajectory that satisfies robot-specific constraints and is collision-free against obstacles, especially when the given end-effector path passes around obstacles. In this paper, we present a trajectory optimization of a redundant manipulator (TORM) to synthesize a trajectory that follows a given end-effector path accurately, while achieving smoothness and collision-free manipulation. Our method holistically incorporates three desired properties into the trajectory optimization process by integrating the Jacobian-based inverse kinematics solving method and an optimization-based motion planning approach. Specifically, we optimize a trajectory using two-stage gradient descent to reduce potential competition between different properties during the update. To avoid falling into local minima, we iteratively explore different candidate trajectories with our local update. We also accelerate our optimizer by adaptively determining the stop of the current exploration based on the observation of optimization results. We compare our method with five prior methods in test scenes, including external obstacles and two non-obstacle problems. Furthermore, we analyze our optimizer performance by experimenting with three different configurations of robots. Our method robustly minimizes the pose error in a progressive manner while satisfying various desirable properties.

Keywords Kinematics · Motion Planning · Trajectory Optimization · Redundant Manipulator

Mincheul Kang
E-mail: mincheul.kang@kaist.ac.kr

Sung-Eui Yoon
E-mail: sungeui@kaist.edu, corresponding author

[1]Scalable Graphics, Vision and Robotics (SGVR) Lab, School of Computing, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea

1 INTRODUCTION

Remote control of various robots has been one of the main challenges in robotics, while it is commonly used for cases where it is difficult or dangerous for a human to perform tasks (Katyal et al., 2014; Long et al., 2019). In this remote control scenario, a robot has to follow the task command accurately while considering its surrounding environment and the constraints of the robot itself.

In the case of a redundant manipulator that this paper focuses on, a sequence of finely-specified joint configurations is required to follow the end-effector path in the Cartesian space accurately. Traditionally, inverse kinematics (IK) has been used to determine joint configurations given an end-effector pose. The traditional IK, however, cannot consider the continuity of configurations, collision avoidance, and kinematic singularities that arise when considering to follow the end-effector path.

Path-wise IK approaches (Rakita et al., 2018, 2019) solve this problem using non-linear optimization by considering aforementioned constraints (Sec. 2.1). These approaches avoid self-collisions using a neural network, but they do not deal with collisions for external obstacles. On the other hand, prior methods (Holladay et al., 2019; Holladay and Srinivasa, 2016) based on a motion planning approach consider external obstacles and use IK solutions to synthesize a trajectory following the desired path in the Cartesian space. By simply using IK solutions, however, these methods have time or structural difficulties in getting a highly-accurate solution (Sec. 2.2).

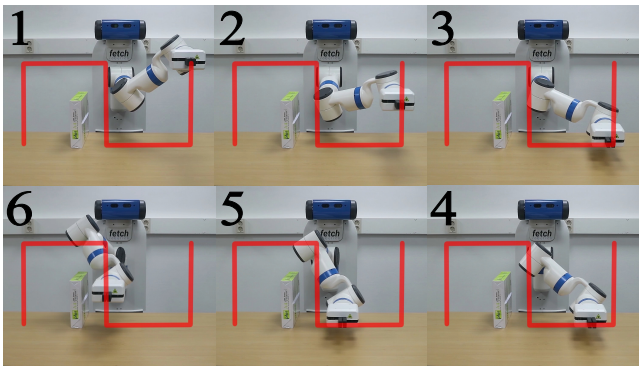


Fig. 1 These figures show a sequence of maneuvering the Fetch manipulator to follow the specified end-effector path (red lines). Our method generates the trajectory that accurately follows the given end-effector path, while avoiding obstacles such as the pack of A4 paper and the table.

Optimization-based motion planners (Schulman et al., 2013; Zucker et al., 2013) can make a trajectory that follows the given path by adding the kinematic constraints to their optimizer. However, these planners have difficulty dealing with diverse constraints and complex paths since it tends to fall into a local minimum (Sec. 2.3). Our method is also an optimization-based approach, but we alleviate this problem by performing an appropriate update method for a path-wise IK problem and exploring various trajectories efficiently.

In this work, we present a trajectory optimization of a redundant manipulator (TORM) for synthesizing a trajectory that is accurately following a given path as well as smooth and collision-free against the robot itself and external obstacles (Fig. 1). Our method incorporates these properties into the optimization process by holistically integrating the Jacobian-based IK solving method and an optimization-based motion planning approach. For effective optimization, we consider different properties of our objectives through a two-stage update manner, which alternates making a constraint-satisfying trajectory and following a given end-effector path accurately (Sec. 4.2). To avoid local minima within our optimization process, we iteratively perform exploration for other alternative trajectories. We also accelerate the iterative exploration by adaptively deciding the stop of each exploration based on the observation of optimization results (Sec. 4.3). In addition, in order to strike a good balance between the quality and generation time of trajectory for the exploration, we apply a path simplification to extract sub-sampled poses with appropriate intervals and generate the trajectory using the random IK solution at the extracted poses (Sec. 4.4).

To compare our method with the prior methods, CHOMP (Zucker et al., 2013), Trajopt (Schulman et al., 2013), RelaxedIK (Rakita et al., 2018), Stampede (Rakita et al., 2019) and the work of Holladay et al. (Holladay et al., 2019), we test four scenes with external obstacles and two non-obstacle problems. Furthermore, we test three different configurations of robots, Kuka 7-DoF, Fetch 7-DoF, and Hubo 8-DoF (Sec. 5.1).

Overall, we observe that our method achieves more accurate solutions given the equal planning time over the tested prior methods. Also, our method robustly minimizes the pose error reasonably fast with the anytime property, which quickly finds an initial trajectory and refines the trajectory (Karaman et al., 2011) (Sec. 5.2). This result mainly arises from the holistic optimization process. To show the benefits of our proposed methods, we test by replacing some of our methods with other methods (Sec. 5.3) and analyze the results of three different configurations of robots (Sec. 5.4).

The synthesized results of tested problems and real robot verification can be seen in the attached video. Our implemented code is available at http://github.com/cheulkang/TORM.

This paper is an extension of the conference version of TORM (Kang et al., 2020) and shares the problem statement and preliminaries with the original work (Sec. 1 and Sec. 3). The main additions are accelerated methods from the original methods (Sec. 4) and various experiments and analyses to show the applicability of the proposed methods (Sec. 5). In addition, this paper includes more detailed descriptions of the proposed methods and introduces broader related works than the original paper (Sec. 2). The detailed additions from the original work are as follows:

1. Provide a detailed description for a two-stage gradient descent (TSGD) (Sec. 4.2) and a introduction of the projected gradient descent for solving a constrained problem on which the TSGD is based (Sec. 2.4).
2. Present an adaptive exploration that accelerates the existing iterative exploration by adding stop criteria based on observing the optimization results (Sec. 4.3).
3. Improve the generation of different trajectories, specifically extracting sub-sampled poses applying a path simplification method instead of extracting the poses at regular intervals (Sec. 4.4).
4. Add the comparison with optimization-based motion planners considering the kinematic constraints (Sec. 5.2), CHOMP and Trajopt, including the explanation of the constrained motion planning (Sec. 2.3).
5. Add the results and analysis for new two test scenes that are more difficult than the prior problems (Sec. 5.1), and the comparison with three different configurations of robots, Kuka 7-DoF, Fetch-7-DoF, and Hubo 8-DoF (Sec. 5.4).

2 RELATED WORK

In this section, we discuss prior studies in the fields of inverse kinematics and motion planning for matching the desired end-effector path.

2.1 Inverse Kinematics

Inverse kinematics (IK) has been studied widely to find a joint configuration from an end-effector pose (Buss, 2004). In the case of a redundant manipulator where our target robots belong to, there can be multiple joint configurations from a single end-effector pose. For this problem, several techniques for quickly finding solutions have been proposed (Diankov, 2010; Sinha and Chakraborty, 2019). In particular, task-priority IK algorithms (Chiacchio et al., 1991; Chiaverini, 1997; Kanoun et al., 2011) prioritize solutions based on an objective function for each purpose, e.g., kinematic singularity or constraint task.

Most methods that synthesize a trajectory geometrically constrained for an end-effector pose use the Jacobian matrix for finding a feasible solution (Berenson et al., 2009; Kunz and Stilman, 2012; Stilman, 2007; Vahrenkamp et al., 2009). Unfortunately, the Jacobian matrix is the first derivative of the vector-valued function with multiple variables and thus it can cause false-negative failures by getting stuck in local minima or convergence failures due to the limits of the joint angle (Beeson and Ames, 2015).

Trac-IK (Beeson and Ames, 2015) points out the problem and improves success rates by using randomly selected joint configurations and sequential quadratic programming. Nonetheless, its solutions do not guarantee continuity of a sequence of joint configurations to make a feasible trajectory (Rakita et al., 2018). In summary, the traditional IK approaches have different strengths and weaknesses for synthesizing a feasible trajectory.

To get alleviated solutions, many studies have presented optimization techniques with objective functions for synthesizing a feasible trajectory with matching end-effector poses. Luo and Hauser (2012) use a geometric and temporal optimization to generate a dynamically-feasible trajectory from a sketch input. Recently, Rakita et al. (2018) proposed a real-time approach using a weighted-sum non-linear optimization, called RelaxedIK, to solve the IK problem for a sequence of motion inputs. Since the collision checking has a relatively large computational overhead, RelaxedIK uses a neural network for fast self-collision avoidance.

Based on RelaxedIK, two studies (Praveena et al., 2019; Rakita et al., 2019) are proposed for synthesizing a highly-accurate trajectory on off-line. One of them is Stampede (Rakita et al., 2019), which finds an optimal trajectory using a dynamic programming algorithm in a discrete-space-graph that is built by samples of IK solutions. The other work (Praveena et al., 2019) generates multiple candidate trajectories from multiple starting configurations and then selects the best trajectory with a user guide by allowing a deviation if there are risks of self-collisions or kinematic singularities.

The aforementioned methods, called path-wise IK methods, optimize joint configurations at finely divided end-effector poses. These optimization methods synthesize an accurate and feasible trajectory satisfied with several constraints, i.e., continuity of configurations, collision avoidance, and kinematic singularities. Inspired by this strategy, we propose a trajectory optimization of a redundant manipulator (TORM) to get a collision-free and highly-accurate solution. Unlike prior path-wise IK works, our method considers self-collision as well as external obstacles, thanks to tight integration of an efficient collision avoidance method using a signed distance field.

2.2 Motion Planning for Following an End-effector Path

Motion planning involves a collision-free trajectory from a start configuration to a goal configuration. Based on the framework of motion planning, two methods (Holladay et al., 2019; Holladay and Srinivasa, 2016) are presented to follow the desired end-effector path in Cartesian space. One (Holladay and Srinivasa, 2016) uses an optimization-based method, specifically Trajopt (Schulman et al., 2013), by applying the discrete Fréchet distance that approximately measures the similarity of two curves. Although the optimization-based motion planning approaches quickly find a collision-free trajectory using efficient collision avoidance methods, these approaches can be stuck in local minima due to several constraints (e.g., joint limits and collisions). To assist its optimizer, this approach separately plans a trajectory by splitting the end-effector path as a set of waypoints and then sampling an IK solution at each pose.

Its subsequent work (Holladay et al., 2019) points out the limitation of the prior work that samples only one IK solution for each pose. Considering this property, it presents a sampling-based approach that iteratively updates a layered graph by sampling new waypoints and IK solutions. However, this method needs a long planning time to get a highly-accurate solution, even with lazy collision checking to reduce the computational overhead.

Even though these methods find a collision-free trajectory by utilizing a motion planning approach and random IK solutions, it is hard to get a highly-accurate solution due to time or structural constraints. To overcome these difficulties, our method incorporates the Jacobian-based IK solving method into our optimization process, instead of using only IK solutions. The aforementioned path-wise IK approaches also use the objective function to minimize the

end-effector pose error, but do not combine it with the objective function to avoid external obstacles. On the other hand, our approach holistically integrates these different objectives and constraints within an optimization framework, inspired by an optimization-based motion planning approach, CHOMP (Zucker et al., 2013), and effectively computes refined trajectories based on our two-stage gradient descent method.

2.3 Constrained Motion Planning

A constrained motion planning (CMP) additionally considers kinematic or dynamic constraints in addition to finding a collision-free trajectory (Bonalli et al., 2019; Kim et al., 2016; Kingston et al., 2019; Schulman et al., 2013). In the case of the kinematic constraint, e.g., pulling a drawer or opening a door (Stilman, 2007), the CMP plans the trajectory while limiting movement to specific transitional or rotational axes. Our problem is similar to having the kinematic constraint on all transitional and rotational axes to match a given end-effector path. However, most CMP planners do not have a structure to solve the problem, especially sampling-based planners (Kim et al., 2016; Kingston et al., 2019).

Although an optimization-based motion planner can solve the problem by adding the kinematic constraints to their optimizer with finely divided end-effector poses in a similar manner of the path-wise IK approaches (Rakita et al., 2018, 2019), it is difficult to deal with diverse constraints and objectives, since it tends to fall into a local minimum. For handling this problem, we present a two-stage gradient descent to refine a trajectory to be feasible and accurate with a given end-effector path, and an adaptive exploration to escape local minima efficiently.

2.4 Gradient descent for a constrained problem

Gradient descent is a first-order iterative method of gradually finding a parameter value to minimize an objective function. While an unconstrained problem only focuses on minimizing a given objective function, a constrained problem is important to find a parameter value that minimizes the objective function in a feasible set (Boyd et al., 2004).

For solving a constrained problem, the projected gradient descent (PGD) conducts a projection onto a feasible set after minimizing a primary objective function. Another method is conditional gradient descent (CGD), as known as the Frank-Wolfe algorithm (Frank et al., 1956), which finds a good feasible direction using a local linear approximation of the objective function and then moves along the chosen direction in each step.

Based on these algorithms, various works solving constrained problems extend and apply to fit their applications, e.g., image reconstruction and spike estimation (Gupta et al., 2018; Joulin et al., 2014; Traonmilin et al., 2020). We approach our path-wise problem as a constrained problem that aims to reduce the error with the target end-effector poses and puts collision and joint speed limits as constraints. We aim to minimize the error with the poses and solve the problem through our two-stage gradient descent based on the PGD concept.

2.5 Avoiding local minima for trajectory optimization

Optimization-based approaches usually synthesize the desired trajectory quickly but have a problem of falling into local minima. To escape local minima, STOMP (Kalakrishnan et al., 2011) conducts a stochastic optimization, and CHOMP (Zucker et al., 2013) applies the simulated annealing (Van Laarhoven and Aarts, 1987), one of the meta-heuristic algorithms, for its optimization. Recently, Khan et al. (2021) use the Beetle Antennae Search algorithm (Jiang and Li, 2017) to improve the robustness of their trajectory optimization.

Another way to avoid local minima is to restart by changing parameter values or initial trajectory (Beeson and Ames, 2015; Tong et al., 2006; Zhao et al., 2021). Similarly, we iteratively explore newly created trajectories to avoid local minima. Furthermore, we accelerate the exploration by deciding the restart based on the observation of past optimization results.

3 Background

In this section, we define the path-wise IK problem we aim to solve and then give the background based on previous major studies.

3.1 Problem Definition

The path-wise IK problem is to find a trajectory, $\boldsymbol{\xi}$, that matches a given end-effector path, X, as well as satisfies various constraints, i.e., collisions with obstacles, joint velocity limits, and kinematic singularities. The trajectory $\boldsymbol{\xi}$ is a sequence of joint configurations, and the end-effector path, X, is defined in the six-dimensional Cartesian space.

Our target robot is a redundant manipulator that has multiple joint configurations given an end-effector pose; if the manipulator has a controllable degree of freedom (DoF) greater than six, it has infinitely many valid solutions. Among many candidates, we synthesize a set of joint configurations as a trajectory to follow the given end-effector path

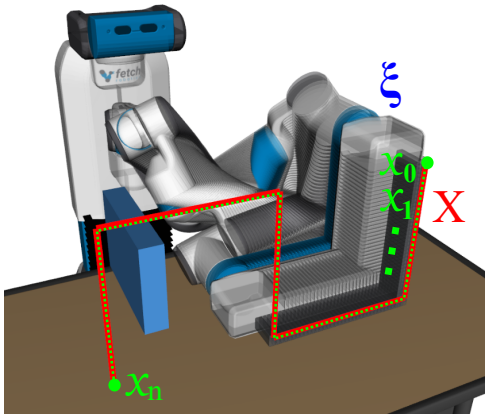


Fig. 2 This figure shows our problem that is synthesizing a feasible and accurate trajectory $\boldsymbol{\xi}$ for a given end-effector path X. The red line is X, which is approximated by end-effector poses $\widetilde{X} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ (green dots). The trajectory $\boldsymbol{\xi}$ is computed at end-effector poses $\widetilde{X}$. The part of the synthesized trajectory shows that the end-effector follows the red line, and the sequence of joint configurations is smooth and collision-free, while avoiding obstacles such as the blue box and the table.

accurately, while avoiding collisions for external obstacles and the robot itself (Fig. 2).

We solve the problem by applying the waypoint parameterization (Flash and Potts, 1988) of the path that finely splits the given end-effector path, $X \approx \widetilde{X} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$; $\boldsymbol{x} \in \mathbb{R}^6$ is an end-effector pose. We then compute the joint configurations for end-effector poses $\widetilde{X}$. As a result, the trajectory is approximated as: $\boldsymbol{\xi} \approx [\boldsymbol{q}_0 \ \boldsymbol{q}_1 \ \dots \ \boldsymbol{q}_n]^T \subset \mathbb{R}^{(n+1)\times d}$, where d is the DoF of a manipulator. When a start configuration $\boldsymbol{q}_0$ is given, we compute the joint configurations from its next configuration, $\boldsymbol{q}_1$, to the goal configuration $\boldsymbol{q}_n$. In our problem, however, the start configuration may not be given, while an end-effector path is given.

To solve our path-wise problem, we present a trajectory optimization of a redundant manipulator (TORM) that holistically integrates the Jacobian-based IK method and an optimization-based motion planning approach. Main notations are summarized in Table 1.

3.2 Jacobian-based Inverse Kinematics

To match a given target end-effector path, our optimizer is based on the Jacobian-based IK method. Many prior techniques explained in Sec. 2.1 utilize the Jacobian-based IK method, since it is very accurate with fast convergence (Beeson and Ames, 2015). In this paper, we combine it with an optimization-based approach to quickly get a highly-accurate trajectory, while avoiding collisions with obstacles and achieving smoothness of joints.

The Jacobian-based IK computes a joint configuration by iteratively minimizing the pose error, F_{pose}, between the target and current end-effector pose. Since our work considers the given end-effector path, not a single pose, we utilize the finely divided end-effector poses $\widetilde{X}$ from the given path. Consequently, F_{pose} for a trajectory is defined as:

$$F_{pose}(\boldsymbol{\xi}) = \frac{1}{2}\sum_{i=0}^{n} \|\boldsymbol{x}_i - FK(\boldsymbol{q}_i)\|^2, \tag{1}$$

where $FK(\boldsymbol{q}_i)$ computes the end-effector pose at the i-th joint configurations $\boldsymbol{q}_i$ using forward kinematics (FK). Note that n is the number of end-effector poses $\widetilde{X}$ divided by the waypoint parameterization and $\boldsymbol{q}_i$ represents the i-th joint configuration corresponding to the i-th end-effector pose $\boldsymbol{x}_i$. In this equation, $\|\boldsymbol{x}_i - FK(\boldsymbol{q}_i)\|^2$ can be represented as $\frac{1}{2}(\boldsymbol{x}_i - FK(\boldsymbol{q}_i))^T(\boldsymbol{x}_i - FK(\boldsymbol{q}_i))$. Accordingly, we can derive the functional gradient of the pose term, ∇F_{pose}, by the following:

$$\nabla F_{pose}(\boldsymbol{q}_i) = \boldsymbol{J}^T(\boldsymbol{x}_i - FK(\boldsymbol{q}_i)), \tag{2}$$

where $\boldsymbol{J} = \frac{d\boldsymbol{x}}{d\boldsymbol{q}} \in \mathbb{R}^{6\times d}$ is the Jacobian matrix.

Table 1 Notation table

Notation	Description
$\widetilde{X}$	Target end-effector poses that are finely divided from the given end-effector path X
$\boldsymbol{\xi}$	Set of joint configurations corresponding to $\widetilde{X}$
$\boldsymbol{q}_i$	Joint configuration on the trajectory $\boldsymbol{\xi}$ at i-th end-effector pose $\boldsymbol{x}_i$
$\boldsymbol{J}$	Jacobian matrix, i.e., $\frac{d\boldsymbol{x}}{d\boldsymbol{q}} \in \mathbb{R}^{6\times d}$
P	Set of body points in the workspace approximating the geometric shape of the manipulator
$\boldsymbol{x}(\boldsymbol{q}, p)$	Partial forward kinematics from the manipulator base to a body point $p \in P$ at a configuration $\boldsymbol{q}$

3.3 Optimization-based Motion Planning

For making a feasible trajectory, we adopt an optimization-based motion planning approach, specifically CHOMP (Zucker et al., 2013), which synthesizes a smooth and collision-free trajectory based on the covariant gradient descent. This approach significantly reduces the planning time by incorporating an efficient collision avoidance method into its optimization process. By incorporating an optimization-based motion planning method with these advantages into our method, we can quickly get the desired solution, while avoiding collisions against external obstacles and the robot itself.

CHOMP models an objective function consisting of avoiding collisions and achieving smoothness:

$$U(\boldsymbol{\xi}) = F_{obs}(\boldsymbol{\xi}) + \lambda F_{smooth}(\boldsymbol{\xi}), \tag{3}$$

where λ is a regularization constant. While minimizing the objective function, CHOMP finds a collision-free trajectory from a start configuration to a goal configuration.

F_{obs} quantitatively measures proximity to obstacles using a signed distance field that can be calculated from the geometry of workspace obstacles. The robot body is simplified into spheres, which serve as conservative bounding volumes for efficient computation. Overall, F_{obs} for a trajectory can be calculated highly fast and is formulated as:

$$F_{obs}(\boldsymbol{\xi}) = \sum_{i=0}^{n-1}\sum_{p}^{P}\left(\frac{1}{2}\Big(c(\boldsymbol{x}_{i+1,p}) + c(\boldsymbol{x}_{i,p})\Big)\left\|\boldsymbol{x}_{i+1,p} - \boldsymbol{x}_{i,p}\right\|\right), \tag{4}$$

where $\boldsymbol{x}_{i,p}$ is partial forward kinematics, i.e., a position of a body point $p \in P$ in the workspace at a configuration $\boldsymbol{q}_i$ and $c(\cdot)$ is an obstacle cost computed from the signed distance field. At a high level, it approximately measures the sum of penetration depths between the robot body and the obstacles. Further, ∇F_{obs} can be derived as the following:

$$\nabla F_{obs}(\boldsymbol{q}_i) = \sum_{p}^{P}\boldsymbol{J}_p^T\left(\|\boldsymbol{x}_{i,p}^{'}\|[(\boldsymbol{I} - \hat{\boldsymbol{x}}_{i,p}^{'}\hat{\boldsymbol{x}}_{i,p}^{'T})\nabla c(\boldsymbol{x}_{i,p}) - c(\boldsymbol{x}_{i,p})\boldsymbol{\kappa}]\right), \tag{5}$$

where $\hat{\boldsymbol{x}}_{i,p}^{'}$ is the normalized velocity vector, and $\boldsymbol{\kappa} = \|\boldsymbol{x}_{i,p}^{'}\|^{-2}(\boldsymbol{I} - \hat{\boldsymbol{x}}_{i,p}^{'}\hat{\boldsymbol{x}}_{i,p}^{'T})\boldsymbol{x}^{''}$ is the curvature vector.

F_{smooth} measures dynamical quantities, i.e., the squared sum of derivatives, to encourage the smoothness between joint configurations:

$$F_{smooth}(\boldsymbol{\xi}) = \frac{1}{2}\sum_{i=0}^{n-1}\left\|\frac{\boldsymbol{q}_{i+1} - \boldsymbol{q}_i}{\Delta t}\right\|^2. \tag{6}$$

Using a finite difference method, F_{smooth} is represented to $F_{smooth} = \frac{1}{2}\parallel \boldsymbol{K}\boldsymbol{\xi} + \boldsymbol{k} \parallel^2 = \frac{1}{2}\boldsymbol{\xi}^T\boldsymbol{A}\boldsymbol{\xi} + \boldsymbol{\xi}^T\boldsymbol{b} + c$, where $\boldsymbol{K}$ and $\boldsymbol{k}$ are the matrix and vector for a finite-difference, $\boldsymbol{A} = \boldsymbol{K}^T\boldsymbol{K}$, $\boldsymbol{b} = \boldsymbol{K}^T\boldsymbol{k}$, and $c = \boldsymbol{k}^T\boldsymbol{k}/2$. We can then simply derive ∇F_{smooth} as $\boldsymbol{A}\boldsymbol{\xi} + \boldsymbol{b}$.

4 Trajectory Optimization

In this section, we describe the motivation and overview of our approach, followed by giving a detailed explanation of our proposed methods.

4.1 Motivation and Overview

A simple way to integrate the Jacobian-based IK and the optimization-based approach is to combine their objective function. Specifically, we can approach the path-wise problem by iteratively updating the trajectory to minimize the cost of the objective function consisting of three different terms:

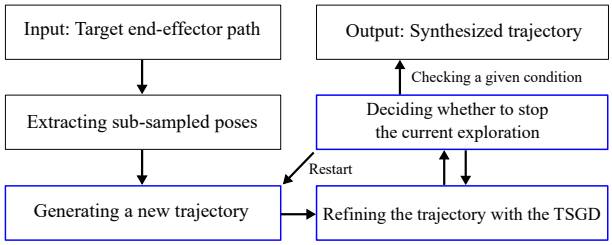


Fig. 3 This figure shows an abstraction of our overall algorithm. The blue boxes represent the iterative process, and one exploration means the refinement process starting from a newly generated trajectory.

$$U(\boldsymbol{\xi}) = F_{pose}(\boldsymbol{\xi}) + \lambda_1 F_{obs}(\boldsymbol{\xi}) + \lambda_2 F_{smooth}(\boldsymbol{\xi}). \tag{7}$$

We, however, found that a naïve approach cannot get a highly-accurate solution due to conflicts among each functional gradient when updating a trajectory (Fig. 4(a)).

Additionally, this optimization-based update method has a probability of falling into local minima due to its local nature. One way of escaping local minima is to perform a fixed number of local updates by repeatedly restarting from different values (Tong et al., 2006). However, it is rather unclear how many updates we need to perform for our task.

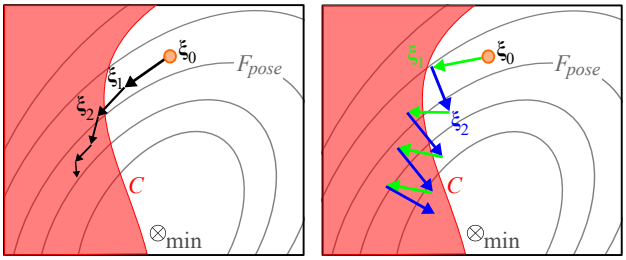
To alleviate the aforementioned problems, we first present a two-stage gradient descent (TSGD) that optimizes a trajectory by dividing two parts: making a trajectory to be feasible and matching a given end-effector path. The TSGD mainly focuses on minimizing the pose error, while achieving the feasibility as satisfying various constraints (Sec. 4.2).

We also propose an adaptive exploration (AE) to avoid local minima efficiently. The AE explores different new trajectories with the TSGD and adaptively decides whether to stop the current exploration instead of repeating a fixed number of updates. The stop criterion is decided through the observation of optimization results (Sec. 4.3).

Overall, our algorithm iteratively performs two parts, generating a new trajectory, $\boldsymbol{\xi}_{new}$, (Sec. 4.4) and updating the trajectory (Fig. 3). An initial trajectory is created in the same way as generating a new trajectory. The update part locally refines the trajectory using our TSGD, along with examining whether or not to restart the current exploration. During the iterative process, we find the trajectory that has the smallest F_{pose} with satisfying the feasibility (Sec. 4.5). This process continues until satisfying a given condition, e.g., running time or cost.

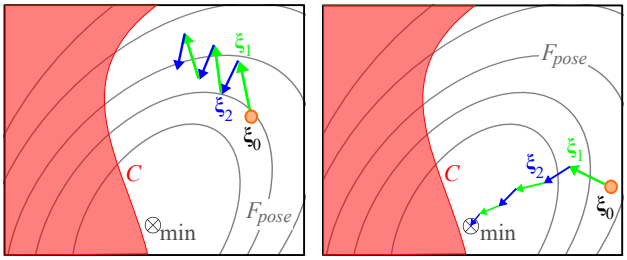
4.2 Two-Stage Gradient Descent

Our goal is to get a highly-accurate solution with satisfying the feasibility. To achieve the goal, we holistically integrate



(a) Simple integration (b) TSGD

Fig. 4 This shows the illustration of optimization progress for a simple integration using Eq. 7, and our TSGD (Eq. 9). Each gray contour line represents equal F_{pose}, and the center, marked $\otimes$, of innermost contour has the smallest F_{pose}. The simple integration (a) is difficult to reach the minimum F_{pose} due to conflicts between three different terms; black arrows represent the update process of the simple integration through the sum of three functional gradients. On the other hand, our TSGD approaches to the minimum F_{pose} through alternatively updating the trajectory toward the feasible set C shown in the red region using ∇F_{obs} and ∇F_{smooth} (green arrow) and updating the trajectory toward the minimum F_{pose} using ∇F_{pose} (blue arrow).



(a) Case where F_{pose} is increasingly growing (b) Case where F_{pose} is reducing without the feasibility

Fig. 5 This illustrates two problematic cases for ineffective exploration; the information of this figure can be seen in Fig. 4. Note that these figures are examples of one exploration from a newly generated trajectory, ξ_0. (a) is the case where F_{pose} gradually increases due to the strong tendency to make a feasible trajectory. In contrast, (b) is the case where F_{pose} is gradually reduced, but it is still not feasible.

the Jacobian-based IK with an optimization-based motion planning approach. When simply combining their terms as shown in Eq. 7, we can get a solution through iterative refinement of a trajectory, but the computed trajectory tends to be sub-optimal, because it is updated from the weighted sum of different functional gradients computed for different, even worse conflicting, purposes (Fig. 4).

To reduce conflicts between different functional gradients, we present a two-stage gradient descent (TSGD) that focuses on minimizing F_{pose} and considers other factors as constraints. Specifically, our optimization problem is defined as a constrained problem:

$$\underset{\boldsymbol{\xi}}{\text{argmin}}\ F_{pose}(\boldsymbol{\xi}), \quad \text{subject to} \quad \boldsymbol{\xi} \in C, \tag{8}$$

where C is a feasible set.

Our TSGD solves Eq. 8 by iteratively performing two parts consisting of updating to make a feasible trajectory and updating the trajectory to match closer to the minimum point in terms of F_{pose}, marked as $\otimes$ (Fig. 4(b)). We design our TSGD inspired by the projected gradient descent that conducts a projection onto a feasible set C after minimizing a primary objective function to solve a constrained problem. Instead of the projection, we update the trajectory to be feasible by minimizing F_{obs} and F_{smooth}. Since the feasible set C cannot be computed at once due to the complexity of the constraints (Bonalli et al., 2019; Schulman et al., 2013; Zucker et al., 2013), we find a feasible trajectory by repeatedly minimizing F_{obs} and F_{smooth}.

Concretely, our TSGD is repeated in which each odd iteration updates the trajectory using ∇F_{obs} and ∇F_{smooth}, and in which even iteration updates the trajectory using ∇F_{pose}:

$$\boldsymbol{\xi}_{i+1} = \begin{cases} \boldsymbol{\xi}_i - \eta_1 \boldsymbol{A}^{-1}(\nabla F_{obs} + \lambda \nabla F_{smooth}), & \text{if } i \text{ is odd,} \\ \boldsymbol{\xi}_i - \eta_2 \nabla F_{pose}, & \text{otherwise,} \end{cases} \tag{9}$$

where η is a learning rate, and $\boldsymbol{A}$ is from an equally transformed representation of the smooth term (see the bottom of Sec. 3.3). $\boldsymbol{A}^{-1}$ acts to retain smoothness and to accelerate the optimization by having a small amount of impact on the overall trajectory. ∇F_{obs} and ∇F_{smooth} using $\boldsymbol{A}^{-1}$ have a computational benefit by giving an influence between successive joint configurations, which is shown in Zinkevich (2003); Zucker et al. (2013). On the other hand, ∇F_{pose} does not apply $\boldsymbol{A}^{-1}$ to compute highly-accurate joint configurations matched for each finely divided end-effector pose.

Our TSGD needs a more number of iterations by performing two separate updates over the simple integration (Eq. 7). Nevertheless, we found that our method shows a faster convergence speed than the simple integration (Fig. 9); in our experiment, our TSGD shows at least 100 times less pose error over the simple integration (Table 4). This is thanks to the TSGD, which effectively resolves conflicts between constraints.

4.3 Adaptive Exploration

Our update rule based on gradient descent has a probability to fall into sub-optimal due to its local nature. Furthermore, there can be many surrounding local minima in our solution space, since we consider several different properties of constraints, e.g., collisions, velocity limits, and matching a given path.

To effectively avoid getting stuck in local minima, we suggest an adaptive exploration (AE) that can explore new, different trajectories, while examining whether to stop each exploration part or not. Note that a redundant manipulator

can have multiple joint configurations at a single pose, thus we can construct many candidate trajectories. By utilizing this property, we generate new, different trajectories, which are locally refined based on our TSGD.

An iterative exploration, starting with new values, has been used to avoid local minima in works based on the gradient descent (Beeson and Ames, 2015; Tong et al., 2006). On the other hand, the iterative exploration has the drawback of computational waste by performing a fixed number of iterations for each exploration.

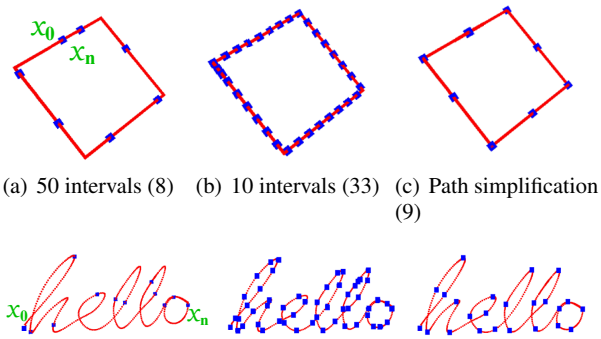
To improve the effectiveness of the exploration, we set a stopping criteria inspired by accelerated gradient methods (Kim and Fessler, 2018; O'donoghue and Candes, 2015), which can dramatically improve the convergence rate through a heuristic restart scheme. Our AE restarts when it is likely that the exploration progress is ineffective, or makes little change in the recently updated trajectories as a local minimum.

We have observed that our optimizer has poor exploration in typically two cases. One of the cases is that F_{pose} is increasing as the force to satisfy the constraints is strong (Fig. 5(a)). The other case is that F_{pose} is gradually reduced, but any trajectories do not achieve the feasibility (Fig. 5(b)). These cases may occur as one of the two stages has a greater impact on the optimization, but are more affected by an initial trajectory, $\boldsymbol{\xi}_0$. When the initial trajectory is not good enough, such as violating the constraints in many parts of the trajectory, finding the desired solution only with local updates is difficult. Such importance of initial values in optimization-based planning has been discussed in several works (Beeson and Ames, 2015; Luo and Hauser, 2012). Hence, we can accelerate our optimization by restarting from a new trajectory when the current exploration is judged to the aforementioned cases.

The AE makes a decision to restart by checking whether the exploration progress corresponds to either one of the two cases or falls into a local minimum. To do that, we set a restart scheme based on the past optimization results, specifically pose error F_{pose} and feasibility; the results are calculated after even iteration in the TSGD. Our restart scheme examines the change tendency of F_{pose} approximately by computing the average slope of past m F_{pose} based on the current F_{pose}:

$$\beta = \frac{1}{m}\sum_{i=1}^{m}(F_{pose}^{u} - F_{pose}^{u-i})/i, \tag{10}$$

where u is the current iteration index; if $u < m$, we do not execute this process, and we set m to 5. Specifically, we first examine whether the current exploration falls into a local minimum by checking $|\beta| < 1 \times 10^{-6}$. Next, we examine whether the current exploration corresponds to two cases of ineffective exploration (Fig. 5). The first case (Fig. 5(a)) where F_{pose} is gradually increasing is checked by $\beta > 0.1$.



(a) 50 intervals (8) (b) 10 intervals (33) (c) Path simplification (9)

(d) 50 intervals (13) (e) 10 intervals (57) (f) Path simplification (25)

Fig. 6 This figure shows sub-sampled poses S (blue dots), extracted from finely divided end-effector poses $\widetilde{X}$ (red dots), in the problem of square tracing and writing "hello". From S, we construct a new trajectory ξ_{new} that starts with random IK solutions and is found in a greedy manner minimizing Eq. 7. (a,d) and (b,e) are extracted uniformly at 10 and 50, respectively. (c) and (f) are the results computed by the path simplification using the DP algorithm. () represents the number of extracted sub-sampled poses as blue dots. (b) has too many S, increasing the time to generate a new trajectory $\boldsymbol{\xi}_{new}$. On the other hand, (d) has a small number of S, which is not enough to make an potentially good trajectory. On the other hand, we apply the path simplification to extract the appropriate S (c,f) regardless of various forms of paths.

The second case (Fig. 5(b)) where F_{pose} is enough reduced without the feasibility is checked by simultaneously testing $|\beta| < 1 \times 10^{-3}$ and examining whether the constraints are satisfied. Note that the exploration is continued if the aforementioned, three conditions are not satisfied.

We decide the restart from a new trajectory through a simple, yet effective restart scheme. As a result, our approach shows a faster convergence than performing an exploration by a fixed number of iterations (Table 4). This is mainly thanks to the adaptive iterations.

4.4 New Trajectory Generation

Each exploration part generates a new trajectory $\boldsymbol{\xi}_{new}$. We strive to find a potentially good trajectory for our local optimization, which considers our objectives with smoothness, avoiding obstacles, and following a given path. Because merely connecting start and goal configurations can result in a sub-optimal solution, especially in cases of complex scenarios (e.g., Fig. 6 and environments with external obstacles) (Holladay and Srinivasa, 2016).

Overall, we consider random configurations and choose one that minimizes three different terms (Eq. 7) in a greedy manner for generating an initial trajectory. As the first step, we extract sub-sampled poses, S, from the end-effector poses $\widetilde{X}$, since working with more poses tends to increase the complexity of generating a trajectory. We can uniformly extract S at γ intervals from $\widetilde{X}$, but it is challenging to set γ

for effectively handling various end-effector paths (Fig. 6). For example, when the interval is small, the generation time increases. On the other hand, when the interval is large, the quality of the path may deteriorate.

To handle this trade-off problem, we extract the most important poses that can well maintain the shape of a given path. To compute such important poses, we apply the path simplification method, specifically Douglas-Peucker (DP) algorithm (Douglas and Peucker, 1973). In our method, the DP recursively extracts $\widetilde{X}$ finding the furthest pose between two poses that are initially start and end poses. This recursive process stops when the furthest pose is closer than a given distance.

In the case of having a long distance between two adjacent sub-sampled poses computed from the DP, there is a probability to create poor quality of trajectory. Accordingly, we additionally extract S so that there are no more than 50 poses between the two adjacent poses. Note that this extraction process is a one-time operation that is cached and reused during all the exploration processes.

In the next step, we find suitable joint configurations at the sub-sampled poses. For the first, sub-sampled pose as the start end-effector pose $\boldsymbol{x}_0$, we simply compute a random IK solution at the pose if a start configuration is not given. For its next pose, we compute j random IK solutions at the pose and greedily select one that minimizes Eq. 7 when connecting it with the configuration of the previous pose; we set j to 100. Lastly, we connect chosen joint configurations through linear interpolation for generating a new trajectory, which is then locally refined by our TSGD.

Our trajectory generation method sometimes generates a poor quality of trajectory due to its random nature. Even though we can reduce the randomness by increasing the number of sub-sampled poses with many IK solutions, it exponentially increases the generation time. We, therefore, extract sub-sampled poses at appropriate intervals using the path simplification to maintain a balance between generation time and quality of trajectory. Furthermore, we compensate for the randomness of the generated trajectory by exploring various trajectories through adaptive exploration (Sec. 4.3).

4.5 Trajectory Constraints

During the optimization process, we may find a low-cost solution, but it can violate several constraints. For example, a trajectory can have collisions with obstacles or self-collisions, even though the trajectory accurately follows a given end-effector pose. Hence, we check collisions every time we find a better trajectory during our optimization process.

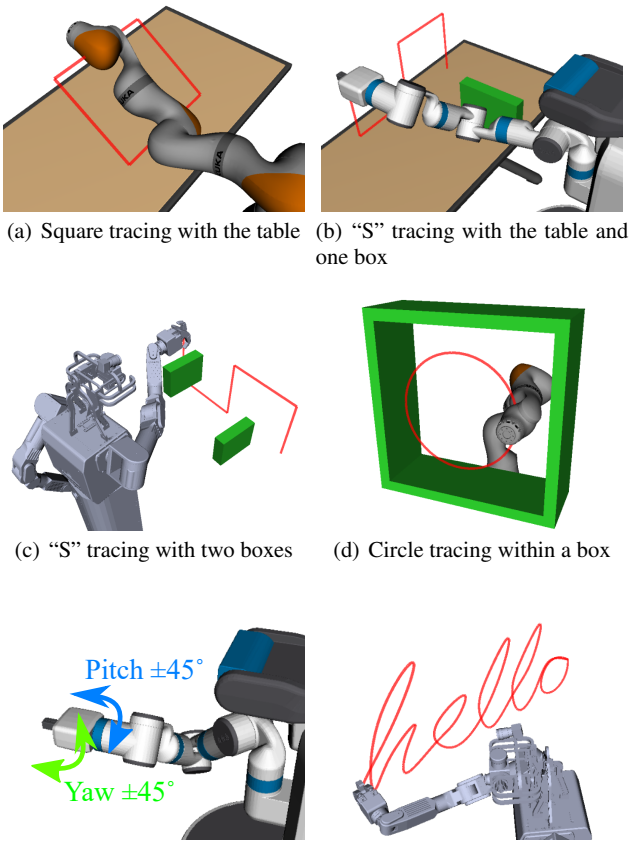
In addition to the collision checking, a manipulator commonly has several constraints that must satisfy lower and upper limits of joints, velocity limits, and kinematic singularities for joints. In the case of lower and upper limits of joints, it is traditionally handled by performing L_1 projection that resets the violating joints value to its limit value. To retain smoothness, we use a smooth projection used by CHOMP (Zucker et al., 2013) during the update process. The smooth projection uses the Riemannian metric $\boldsymbol{A}^{-1}$. In other words, an updated trajectory, $\tilde{\boldsymbol{\xi}}$, is defined as $\tilde{\boldsymbol{\xi}} = \boldsymbol{\xi} + \alpha \boldsymbol{A}^{-1}\boldsymbol{v}$, where $\boldsymbol{v}$ is the vector of joint values, and α is a scale constant. This process is repeated until there is no violation.

For other constraints like the velocity limit, we check them together while checking collisions. The velocity limit for joints is checked by computing the velocities of joints between $\boldsymbol{q}_{i-1}$ and $\boldsymbol{q}_i$ for a given time interval, Δt. Another constraint is the kinematic singularity that is a point where the robot is unstable, and it can occur when the Jacobian matrix loses full rank. To check kinematic singularities, we use the manipulability measure (Yoshikawa, 1985) used by RelaxedIK (Rakita et al., 2018). At a high level, it avoids making the manipulability measure less than a certain value that is computed by random samples. Note that our method returns the lowest cost trajectory guaranteed through checking constraints for constructing a feasible trajectory.

5 Experiments and Analysis

In this section, we provide various experiment results, and discussions of our method and other prior works using three different configurations of robots, Kuka 7-DoF, Fetch 7-DoF, and Hubo 8-DoF. Furthermore, we test our method with the real Fetch robot. The results are shown in the accompanying video.

We report the average performance by performing 20 tests with a machine that has 3.6 GHz Intel i7-9700 CPU and 32 GB RAM. In this experiment, we compute the pose error with target end-effector poses using a weighted sum of the Euclidean distances of the translational and rotational parts, which was used in the work of Holladay et al. (2019); the adopted weight of the rotational distance over the translational distance is 0.17. In addition, we construct the target end-effector poses for calculating the pose error by adding one more between the poses used for planning. This is because the error calculation using only poses used for planning may not be precise in determining the concordance rate with a given end-effector path, since our target robot, a redundant manipulator, has various joint configurations for one pose.



(a) Square tracing with the table (b) "S" tracing with the table and one box

(c) "S" tracing with two boxes (d) Circle tracing within a box

(e) Rotation task (f) Writing "hello"

Fig. 7 This shows six problems with three different robots, Kuka 7-DoF, Fetch 7-DoF, and Hubo 8-DoF. Four problems, (a) to (d), include external obstacles and two problems, (e) and (f), do not have external obstacles. The red line represents the given end-effector path.

5.1 Experiment setting

We prepare six problems (Fig. 7) with external obstacles and two non-obstacle problems to evaluate and compare our method with prior methods, CHOMP (Zucker et al., 2013), Trajopt (Schulman et al., 2013), RelaxedIK (Rakita et al., 2018), Stampede (Rakita et al., 2019), and the work of Holladay et al. (2019). In this section, we call the work of Holladay et al. (2019) EIGS, taken from their paper title.

Four problems with obstacles are the square tracing with the table, the "S" tracing with the table and blue box, the "S" tracing with two boxes, and the circle tracing with surrounding obstacles. At these problems, we do not test for RelaxedIK and Stampede, since these prior methods did not consider external obstacles.

To compare ours against those prior methods, we prepare two non-obstacle problems, rotating ±45 degrees in the direction of pitch and yaw, and writing "hello", by adapting problems used in those methods; we just change writing "icra" to "hello".

We used the code of RelaxedIK and Stampede that are provided by authors through their websites. For RelaxedIK, Stampede, and our method, the end-effector paths of all problems are finely divided and are fed into all the tested methods; in our experiment, the distance between two divided end-effector poses is about $0.005m$ for translation and $0.02rad$ for rotation, following the protocol adopted in Rakita et al. (2019). On the other hand, EIGS initially splits the end-effector path and gradually breaks down the path during the planning.

In CHOMP and Trajopt, we modified their setting for solving the existing constrained motion planning problem (Sec. 2.3) to solve the path-wise problem. Their setting considers various kinds of kinematic constraints, and we gave the kinematic constraint on all transitional and rotational axes with the finely divided end-effector poses. Also, we set CHOMP and Trajopt to find a trajectory that minimizes pose error while satisfying the constraints as in our method (Sec. 4.5). In addition, we gave an initial trajectory computed by our new trajectory generation method (Sec. 4.4), since both planners cannot solve our problems mostly without the initial trajectory.

We mainly evaluate whether the synthesized trajectory accurately follows the given end-effector path. Note that reported results were extracted from feasible trajectories satisfying the given constraints (Sec. 4.5).

5.2 Comparison with prior methods

In four problems including external obstacles, we fix the start configuration $\boldsymbol{q}_0$ located close to the obstacles in order to see how well different methods can avoid external obstacles (Fig. 7). For non-obstacle problems and two obstacle problems, we do not fix $\boldsymbol{q}_0$ to see how different methods handle $\boldsymbol{q}_0$ during the planning process.

Table 2 shows the overall results of different methods, including three different robots. Fig. 8 shows the trajectory quality of different methods, as we have more planning time up to the maximum planning budget of 50 seconds. On the other hand, Table 2 shows a snapshot result of the trajectory computed at a specific planning time.

For the non-obstacle problems, the rotation task and writing "hello", we report results by the initial solution time of Stampede, instead of 50 seconds; within 50 seconds, Stampede computed only a single trajectory. Note that in the case of RelaxedIK, a real-time planner, it quickly synthesizes one trajectory at one execution, but its computed trajectory tends to be low-quality.

CHOMP, Trajopt, and our method find an initial solution faster than other methods in most problems (Table 2). This is because these techniques set an initial trajectory using our new trajectory generation method. It also supports that our new trajectory generation method quickly synthesizes a potentially good trajectory (Sec. 4.4).

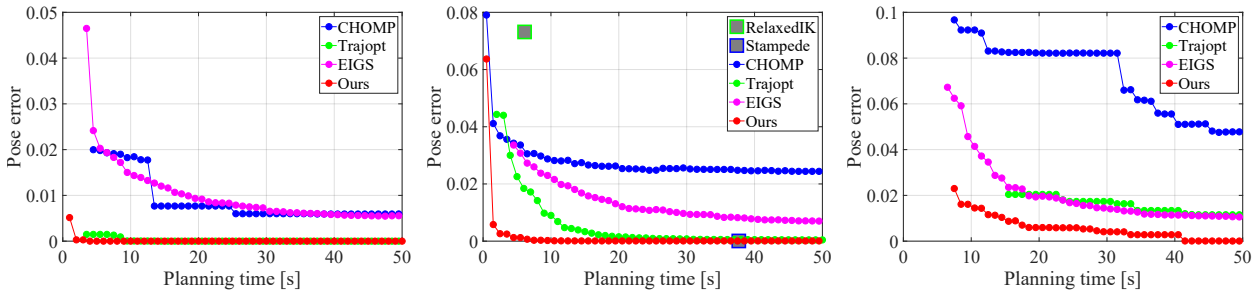
Table 2 Results of different methods, CHOMP, Trajopt, EIGS, RelaxedIK, Stampede, and ours given an equal planning time. RelaxedIk is a real-time planner and does not provide better trajectories with more planning time; we provide its results while it cannot be compared in the equal-time comparison. RelaxedIK and Stampede experimented with non-obstacle problems since these methods only check self-collision. The bold text indicates the smallest pose error, and the underline indicates the second smallest pose error.

PE: Pose error. SD: Standard Deviation. TL: Trajectory length (rad). NF: Number of failures. IST: Initial solution time (s). PT: Planning time (s).

		Kuka 7-DoF					Fetch 7-DoF					Hubo 8-DoF				
		PE / SD	TL	NF	IST	PT	PE / SD	TL	NF	IST	PT	PE / SD	TL	NF	IST	PT
Square tracing with the table (fixed $\boldsymbol{q}_0$)	CHOMP	2.25e-2 / 7.9e-3	9.4	0	0.4	50	2.17e-2 / 3.3e-3	8.5	0	**0.6**	50	2.82e-2 / 1.3e-2	11.0	0	5.5	50
	Trajopt	3.33e-3 / 8.0e-3	7.9	0	3.4		1.46e-2 / 2.7e-3	10.3	1	4.5		7.80e-3 / 5.6e-3	11.1	5	7.3	
	EIGS	6.39e-3 / 1.3e-3	24.1	0	5.2		5.11e-3 / 1.2e-3	25.6	0	3.5		6.90e-3 / 1.1e-3	36.6	0	6.0	
	Ours	**6.37e-6** / 1.3e-6	7.9	0	**0.7**		**6.33e-6** / 1.0e-6	8.3	0	0.8		**1.51e-5** / 6.9e-6	10.8	0	**3.1**	
"S" tracing with the table and one box (fixed $\boldsymbol{q}_0$)	CHOMP	5.94e-3 / 8.3e-4	6.0	0	4.7	50	1.53e-2 / 2.7e-3	7.1	0	7.9	50	7.2e-2 / 4.6e-2	13.5	0	10.1	50
	Trajopt	4.61e-6 / 7.6e-6	5.9	0	3.3		5.70e-2 / 5.8e-2	8.7	2	3.8		9.05e-3 / 1.0e-2	10.0	7	8.3	
	EIGS	5.50e-3 / 8.1e-4	25.1	0	3.6		4.51e-3 / 7.2e-4	21.6	0	2.9		8.50e-3 / 1.8e-3	29.2	0	5.2	
	Ours	**1.72e-6** / 6.4e-7	6.0	0	**1.1**		**6.00e-6** / 2.1e-6	7.0	0	**1.2**		**6.85e-5** / 4.7e-5	11.7	0	**2.6**	
"S" tracing with the table and one box	CHOMP	3.66e-2 / 4.2e-2	7.6	0	4.3	50	7.52e-3 / 1.3e-3	6.1	0	6.8	50	2.31e-2 / 1.4e-2	9.8	1	6.9	50
	Trajopt	6.69e-6 / 5.3e-6	7.1	3	3.2		7.30e-6 / 3.9e-6	6.7	7	3.8		1.90e-3 / 4.9e-3	8.3	9	**3.6**	
	EIGS	4.64e-3 / 6.1e-4	24.0	0	3.9		5.02e-3 / 1.1e-3	18.8	0	3.2		8.34e-3 / 1.8e-3	31.4	0	6.2	
	Ours	**1.80e-6** / 1.9e-6	5.6	0	**1.2**		**1.35e-6** / 1.3e-6	5.7	0	**1.4**		**1.85e-5** / 1.6e-5	8.4	0	3.7	
"S" tracing with two boxes (fixed $\boldsymbol{q}_0$)	CHOMP	1.53e-2 / 3.4e-3	8.4	1	5.0	50	3.22e-2 / 1.3e-3	11.0	0	2.3	50	4.29e-2 / 2.0e-2	11.2	2	11.8	50
	Trajopt	2.21e-2 / 2.6e-2	9.4	4	**1.2**		8.54e-5 / 9.1e-5	10.8	6	2.9		5.16e-3 / 6.7e-3	10.2	9	**3.4**	
	EIGS	1.70e-2 / 1.0e-3	33.9	0	3.7		8.62e-3 / 5.4e-3	25.5	0	3.0		9.68e-3 / 1.2e-3	26.5	0	5.1	
	Ours	**9.26e-6** / 2.9e-6	8.2	0	1.4		**1.36e-5** / 2.6e-6	9.7	0	**1.8**		**1.12e-4** / 7.1e-4	10.2	0	4.9	
Circle tracing with surrounding obstacles (fixed $\boldsymbol{q}_0$)	CHOMP	1.83e-2 / 7.4e-4	7.8	0	**0.6**	50	1.68e-2 / 7.6e-4	7.0	0	2.4	50	4.77e-2 / 2.5e-2	8.5	1	7.4	50
	Trajopt	3.54e-5 / 4.5e-5	7.5	0	3.4		2.26e-2 / 2.4e-2	8.7	8	1.8		1.15e-2 / 1.3e-2	7.8	6	15.7	
	EIGS	4.72e-3 / 1.0e-3	24.9	0	3.9		5.40e-3 / 1.2e-3	16.9	0	2.7		1.06e-2 / 4.2e-3	24.2	0	**6.3**	
	Ours	**7.20e-6** / 4.5e-7	7.8	0	0.7		**7.27e-6** / 1.5e-6	7.5	0	**1.3**		**1.10e-4** / 3.8e-4	7.8	0	**6.3**	
Circle tracing with surrounding obstacles	CHOMP	2.91e-2 / 1.5e-2	7.8	0	**0.6**	50	1.16e-2 / 3.0e-3	6.9	0	2.0	50	2.67e-2 / 2.0e-2	8.7	3	9.4	50
	Trajopt	8.53e-6 / 6.3e-6	7.7	3	4.3		2.89e-5 / 5.1e-5	6.9	9	2.3		1.91e-3 / 6.0e-3	7.9	8	5.7	
	EIGS	5.34e-3 / 6.5e-4	23.1	0	4.2		6.05e-3 / 9.9e-4	14.8	0	2.9		6.93e-3 / 1.4e-3	19.4	0	5.9	
	Ours	**4.96e-6** / 3.2e-6	7.4	0	0.8		**2.30e-6** / 1.3e-6	6.8	0	**0.8**		**1.13e-5** / 1.3e-5	8.0	0	**5.5**	
Rotation task without obstacles	RelaxedIK	3.93e-2 / 3.7e-3	8.9	0	-	6.3	7.31e-2 / 4.3e-3	12.1	0	-	6.1	8.29e-2 / 6.6e-3	8.9	0	-	7.3
	Stampede	5.85e-5 / 1.8e-6	9.5	0	28.5	28.5	**6.49e-5** / 1.9e-6	12.4	0	35.7	35.7	2.58e-4 / 8.2e-5	9.8	0	38.0	38.0
	CHOMP	7.33e-3 / 3.3e-3	8.7	0	**0.3**		2.51e-2 / 7.8e-3	14.3	0	**0.5**		2.83e-2 / 1.1e-2	11.7	0	**1.2**	
	Trajopt	2.85e-3 / 3.7e-3	6.6	0	1.1		5.85e-4 / 8.3e-4	12.6	0	2.0		1.49e-2 / 2.2e-2	10.7	4	6.3	
	EIGS	1.40e-2 / 4.2e-3	17.1	0	4.2		8.30e-3 / 1.5e-3	17.8	0	4.3		7.41e-3 / 1.6e-3	19.2	0	5.6	
	Ours	**5.53e-5** / 3.3e-5	8.9	0	0.5		6.96e-5 / 3.8e-6	13.3	0	0.6		**2.21e-4** / 1.3e-4	10.5	0	1.5	
Writing "hello" without obstacles	RelaxedIK	4.10e-2 / 2.1e-3	19.2	0	-	16.7	5.04e-2 / 4.6e-3	21.6	0	-	16.0	5.29e-2 / 8.7e-3	27.4	0	-	17.2
	Stampede	3.28e-5 / 4.7e-5	20.3	0	30.4	30.4	4.71e-5 / 6.6e-5	24.8	0	38.3	38.3	**9.10e-5** / 2.0e-4	29.7	0	45.1	45.1
	CHOMP	1.27e-1 / 2.8e-2	36.0	0	**1.1**		3.76e-2 / 3.7e-3	26.3	0	**1.2**		6.43e-2 / 3.5e-2	32.5	0	**4.5**	
	Trajopt	2.42e-3 / 2.6e-2	25.5	0	5.5		1.54e-3 / 5.9e-2	28.4	2	6.9		1.71e-2 / 6.0e-3	31.1	4	7.1	
	EIGS	2.23e-2 / 3.8e-3	49.2	0	11.7		1.7e-2 / 2.1e-3	36.9	0	11.4		1.7e-2 / 1.9e-3	49.0	0	13.2	
	Ours	**3.13e-5** / 1.0e-5	22.5	0	1.5		**4.36e-5** / 3.6e-6	26.0	0	1.3		3.3e-4 / 5.3e-4	31.8	0	5.2	

Nonetheless, CHOMP shows the highest pose error in most problems, even though it gradually reduces the pose error over planning time. Our method is based on CHOMP, but we achieved a highly-accurate solution thanks to the update through our TSGD by combining the Jacobian-based IK approach and CHOMP. In addition, Table 3 shows the computational advantage of our TSGD combined with the Jacobian-based IK approach compared to CHOMP considering various kinds of kinematic constraints; in CHOMP, multiple matrix calculations are performed to take into account various kinematic constraints (see Zucker et al. (2013)).

Trajopt has the highest number of failures due to falling into a local minimum, even though it had good results in some problems, e.g., "S" tracing with two boxes using Fetch 7-DoF. Especially, there are many failures in the two prob-



(a) Square tracing using the Kuka 7-DoF (b) Rotation task using the Fetch 7-DoF (c) Circle tracing using the Hubo 8-DoF

Fig. 8 This shows the pose error over the planning time of different methods in three different problems. Since (a) and (c) include external obstacles, RelaxedIK and Stampede are excluded from the experiments. Also, the start configurations of (a) and (c) are fixed. We visualize graphs once an initial solution is computed.

Table 3 Results of the number of iterations and explorations during 50 seconds for optimization-based approaches, CHOMP, Trajopt, and ours. Parenthesis indicates the time (s) for one update of its optimizer. We regard one update of Trajopt as one execution of the quadratic solver.

		Circle tracing		Writing "hello"	
		# of iter.	# of explor.	# of iter.	# of explor.
Kuka 7-DoF	CHOMP	751 (0.05)	-	204 (0.23)	-
	Trajopt	13 (2.5)	-	10 (4.3)	-
	Ours	542 (0.03)	61	392 (0.06)	29
Fetch 7-DoF	CHOMP	776 (0.05)	-	208 (0.20)	-
	Trajopt	11 (3.0)	-	8 (5.4)	-
	Ours	582 (0.03)	60	561 (0.06)	10
Hubo 8-DoF	CHOMP	256 (0.15)	-	86 (0.43)	-
	Trajopt	8 (4.9)	-	5 (7.8)	-
	Ours	173 (0.12)	14	115 (0.30)	7

lems, "S" tracing with two boxes and circle tracing, surrounded by obstacles in the given end-effector path. This is because Trajopt uses the second-order method, so the update time is long, making it difficult to explore various trajectories during a given planning time. On the other hand, our method did not record failures by exploring new trajectories thanks to the very fast update of our TSGD based on gradient descent (Table 3).

From comparison with optimization-based motion planning considering geometric constraints, we can see that our proposed method effectively optimizes the path-wise IK problem to synthesize a highly-accurate solution while avoiding local minima. In addition, although optimization-based approaches generally have difficulty in changing a start configuration $\boldsymbol{q}_0$ during the planning process, our method is quite straightforward to change $\boldsymbol{q}_0$ thanks to adaptively exploring new trajectories including $\boldsymbol{q}_0$. As a result, our method did not record failures in all problems.

EIGS refines a trajectory by progressively sampling new waypoints and IK solutions. Nonetheless, our method improves the quality of the trajectory over EIGS, as we have more planning time (Fig. 8). This improvement is mainly because our method incorporates the IK solving method into the optimization process instead of simply using IK solutions.

In Table 2, EIGS shows the longest length of the computed trajectory on average for all problems; we measure the length of a trajectory using the Euclidean distance. EIGS does not consider the distance in C-space, since it only checks the similarity measure of two curves using the discrete Fréchet distance (Holladay et al., 2019). On the other hand, other methods take into account the smoothness between joint configurations and thus generate shorter trajectories than EIGS.

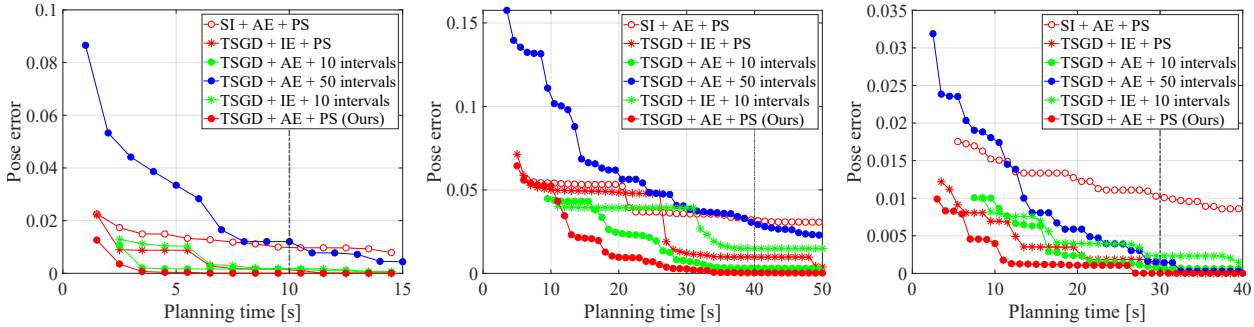
Stampede is also a sampling-based approach like EIGS, but it generates a highly-accurate solution on average (Table 2). Stampede does not deal with external obstacles, but uses a neural network to check self-collision quickly. However, it takes a long time to get an initial solution, because it has to samples IK solutions at all the end-effector poses (Table 2). These results show that Stampede and EIGS have different pros and cons.

RelaxedIK is a real-time planner, thus it quickly finds a solution (Table 2). However, its accuracy is much lower than other methods. In conclusion, RelaxedIK shows real-time performance by quickly optimizing the joint configuration for one pose, but it is difficult to obtain a highly-accurate trajectory.

Overall, our method finds an initial solution quite quickly with a high pose error, but improves its quality as we have more planning time (Fig. 8). Also, our method has a lower pose error with a lower standard deviation on average than other methods, as shown in Table 2. These results support that our optimization process efficiently reduces the pose error, while satisfying several constraints.

5.3 Analysis of our proposed methods

To see the benefits of components of our proposed method, we conduct ablation study. We first substitute the two-stage



(a) Writing "hello" using the Fetch 7-DoF (b) Writing "hello" using the Hubo 8-DoF (c) Square tracing using the Hubo 8-DoF

Fig. 9 This shows the pose error as a function of the planning time of various ablated methods. Our complete method contains two-stage gradient descent (TSGD), adaptive exploration (AE), and path simplification (PS). We substitute them with the simple integration (SI), the iterative exploration (IE) with the fixed number of iterations, and extracting sub-sampled poses at 10 and 50 intervals. The gray dotted lines represent the results shown in Table 4.

Table 4 Analysis of components of our proposed method by substituting them to alternative methods, the simple integration (SI), iterative exploration (IE), and extracting sub-sampled poses at 10 and 50 intervals. The last row shows the results of our complete method.

Problem			Writing "hello" using Fetch 7-DoF						Writing "hello" using Hubo 8-DoF						Square tracing using Hubo 8-DoF					
Method			Pose error			TL	IST	PT	Pose error			TL	IST	PT	Pose error			TL	IST	PT
Update	Exploration	Extracting S	Avg.	Min.	Max.				Avg.	Min.	Max.				Avg.	Min.	Max.			
SI	Adaptive	PS	9.7e-3	2.9e-3	2.5e-2	28.4	1.4	10	3.2e-2	5.8e-3	1.3e-1	32.1	6.6	40	1.3e-2	4.8e-3	2.2e-2	13.3	5.4	30
TSGD	Adaptive	50 intervals	1.2e-2	4.1e-5	8.7e-2	25.9	0.9		2.1e-2	**4.7e-5**	8.4e-2	25.8	3.6		1.5e-3	**2.3e-6**	2.3e-2	9.8	3.6	
TSGD	Iterative	PS	1.6e-3	**4.0e-5**	3.2e-2	27.8	1.4		9.8e-3	5.2e-5	1.5e-1	31.7	4.9		7.6e-4	7.6e-6	1.5e-2	11.2	3.4	
TSGD	Adaptive	10 intervals	1.6e-3	6.7e-5	1.3e-2	34.7	2.5		3.2e-3	1.1e-4	2.7e-2	46.4	9.6		7.7e-4	8.6e-5	6.0e-3	31.4	5.5	
TSGD	Iterative	10 intervals	1.7e-3	8.1e-5	1.2e-2	33.1	2.4		4.1e-3	1.2e-4	2.4e-2	50.5	12.2		2.3e-3	9.9e-5	1.9e-2	31.8	6.4	
TSGD	Adaptive	PS	**5.6e-5**	4.4e-5	**1.0e-4**	26.7	1.3		**3.3e-4**	5.1e-5	**2.4e-3**	31.8	5.2		**2.6e-5**	7.6e-6	**1.1e-4**	10.6	3.2	

TL: Trajectory length (rad). IST: Initial solution time (s). PT: Planning time (s).

gradient descents (TSGD) to the simple integration (SI) updating three functional gradients at once. Next, we test the iterative exploration (IE) iterated a fixed number of update instead of our adaptive exploration (AE). Finally, we compare different ways of extracting sub-sampled poses, 10 and 50 intervals, and the chosen path simplification (PS). Note that the IE and extracting sub-sampled poses at 10 intervals were proposed in our earlier work, the conference version of TORM (Kang et al., 2020).

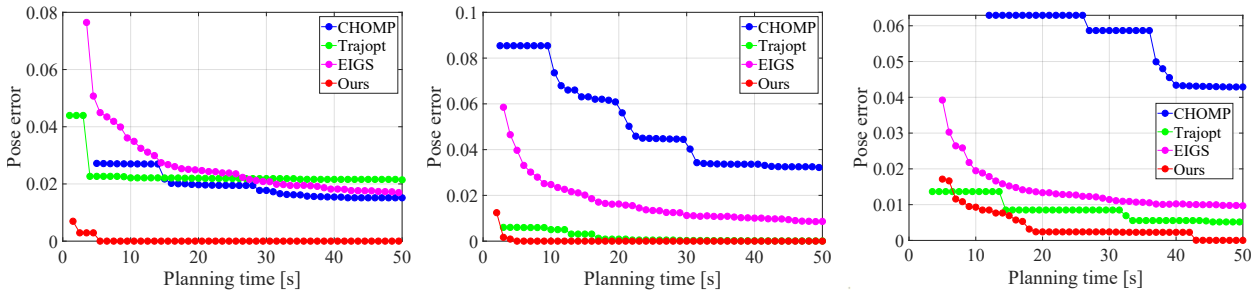
Table 4 shows the results of aforementioned methods for three different problems, writing "hello" using the Fetch 7-DoF and the Hubo 8-DoF, and square tracing using the Hubo 8-DoF; the start configuration of the square tracing is fixed. Fig. 9 also shows the results, as we have more planning time up to the maximum planning budget of 50 seconds.

As shown in Fig. 9(a), it is hard to check the big difference between different methods due to its fast convergence when we use the Fetch 7-DoF. On the other hand, we can confirm the pros of proposed methods in experiments using the Hubo 8-DoF (Fig. 9(b) and Fig. 9(c)). We analyze the results for different configurations of robots in Sec. 5.4.

Using the SI has a higher pose error on average than our full method, and it also has the highest min. pose error among 20 tests, compared to other tested methods (Table 4). These results demonstrate that the different functional gradients conflict with each other. Therefore, the TSGD prevents the competition of different functional gradients and is useful to get a highly-accurate solution. Note that the SI is different approach from CHOMP. The SI is the simple integration of F_{obs}, F_{smooth}, and F_{pose}, the objective of the Jacobian-based IK, while CHOMP is a reconfiguration of the original update rule to consider various kinds of kinematic constraints (Zucker et al., 2013); we set the kinematic constraints on all transitional and rotational axes to match a given end-effector path.

Our method shows the fast convergence rather than using the IE (see green and red lines with dots Fig. 9). In Table 4, the max. pose errors of our method are also lower than using the IE at a specific planning time. These results indicate that the AE makes a good decision to stop the current exploration based on the observation of optimization results.

Extracting sub-sampled poses S at 50 intervals shows the fastest result of finding an initial solution in the problem of writing "hello", while 10 intervals is the slowest. On the other hand, extracting S at 10 intervals shows the better performance than 50 intervals. The time of generating a trajectory is reduced as having smaller sub-sampled poses, but we can get a low-quality path if sub-sampled poses are too



(a) Using Kuka 7-DoF (b) Using Fetch 7-DoF (c) Using Hubo 8-DoF

Fig. 10 This shows the pose error over the planning time of different methods with three different robots in "S" tracing with two boxes.

few. To generate a trajectory, 10 intervals, 50 intervals, and the PS take 7.2, 3.8, and 2.1 seconds in the writing "hello" problem using the Hubo 8-DoF. Although the generation time of the PS is in the middle of 10 and 50 intervals, the PS shows the better performance than others. These results demonstrate that the PS strikes a good balance between the generation time and trajectory quality.

In summary, these results show that our proposed method synthesizes highly-accurate trajectories, while effectively getting out of local minima. Furthermore, we can see that the proposed method is more accelerated, about 43 times faster on average across all the tested cases, than the conference version of TORM (the green line with star dots in Fig. 9).

Time analysis of our proposed methods. In our optimizer, performing TSGD, generating new trajectories, and checking constraints of our method take 70%, 26%, and 4% of the overall running time on average; the TSGD is frequently iterated to refine the trajectory, as the main update operation. The PS is executed once, and it is calculated very quickly in about 0.25 seconds for the writing "hello" problem that has the longest path in our problems.

5.4 Analysis with different robots

To analyze the results with different robots, we compare results in the problem of "S" tracing with two boxes by using three different robots. It is difficult to directly compare them even with the same problem, but we can see overall tendency of the results. Fig. 10 shows the results of "S" tracing with two boxes by using different robots, Kuka 7-DoF, Fetch 7-DoF, and Hubo 8-DoF.

In Fig. 10, our method quickly minimizes the pose error, as having more planning time. Within the given 50 seconds, we also get highly-accurate results across different problems (Table 2). In our method, nevertheless, using the Hubo 8-DoF shows a slower convergence speed than using the Fetch 7-DoF and Kuka 7-DoF. One of main reasons is an increase in DoF, which requires higher computational overhead of the overall method. Other methods also show that using the Hubo 8-DoF generally has worse performances compared to using other robots (Table 2).

In our method, the biggest difference arises due to the number of spheres approximating the robot model to get the collision costs effectively. The Hubo 8-DoF has 163 spheres, while the Fetch 7-DoF and Kuka 7-DoF have 41 and 27, respectively. As a result, the computation time of the Hubo 8-DoF is 120ms, and the computation time of the Kuka 7-DoF and the Fetch 7-DoF is approximately 20ms for one trajectory (Eq. 4) in the problem of "S" tracing with two boxes.

Even though the Fetch robot has torso, head, and lower body in addition to an arm, we use only the arm of Fetch, and thus other parts of the Fetch are treated to be fixed. Since the fixed part can be regarded as a fixed obstacle, the amount of calculation is reduced by constructing a distance field in advance. On the other hand, the Hubo 8-DoF has few fixed parts by using the torso; its fixed part is only the lower body. Therefore, the Hubo 8-DoF has a higher computational amount of collision costs compared to other robots.

Although there is a difference in the amount of calculation depending on the robot configurations, abbreviating the robot model to spheres is cost-effective. Accordingly, our method can quickly avoid collisions with obstacles than other prior methods, resulting in fast convergence.

6 Conclusion and Future work

In this paper, we have presented the trajectory optimization of a redundant manipulator (TORM) that holistically incorporates three important properties into the trajectory optimization process by integrating the Jacobian-based IK solving method and an optimization-based motion planning approach. Given different properties, we have suggested the two-stage gradient descent to follow a given end-effector path and to make a feasible trajectory. We have also performed adaptive exploration to avoid local minima effectively.

We have shown the benefits of our method over the five prior techniques in environments w/ and w/o external obsta-

cles using three different types of robots. Our method has robustly minimized the pose error in a progressive manner and achieved a highly-accurate trajectory at a reasonable time compared to other methods. Further, we have verified the feasibility of our synthesized trajectory using the real, Fetch manipulator.

Even though our method efficiently avoids external obstacles and the robot itself by using the model abbreviated as spheres and signed distance field, a high amount of computation still exists, especially when the robot configuration is complicated due to the high DoF robot. Moreover, our new trajectory generation method is reasonably fast and effective for our task, but it sometimes generates a poor trajectory due to its randomness. Although we address the problem by exploring different trajectories, it would be more desirable to reduce the randomness of the generation for certain tasks.

As future research directions, we would like to relieve the aforementioned problems by applying deep learning to our method. Recently, collision avoidance methods (Kew et al., 2019; Li et al., 2021) using deep learning have been used for motion planning, and these methods can be extended to our method to solve the problem of complexity of robot configurations. We also believe that the trajectory generation using deep learning will lead to improvement in generation speed and mitigation of randomness.

Acknowledgements We appreciate the anonymous reviewers for constructive comments and insightful suggestions. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021R1A4A1032582 and No. 2019R1A2C3002833).

References

Beeson, Patrick, & Barrett Ames (2015). TRAC-IK: An open-source library for improved solving of generic inverse kinematics. *In Humanoids*, pp 928–935. IEEE.

Berenson, Dmitry, Siddhartha S Srinivasa, Dave Ferguson, & James J Kuffner (2009). Manipulation planning on constraint manifolds. *In ICRA*, pp 625–632. IEEE.

Bonalli, Riccardo, Abhishek Cauligi, Andrew Bylard, & Marco Pavone (2019). GuSTO: Guaranteed sequential trajectory optimization via sequential convex programming. *In 2019 International Conference on Robotics and Automation (ICRA)*, pp 6741–6747. IEEE.

Boyd, Stephen, Stephen P Boyd, & Lieven Vandenberghe (2004). *Convex optimization*. Cambridge university press.

Buss, Samuel R (2004). Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, 17(1-19):16.

Chiacchio, Pasquale, Stefano Chiaverini, Lorenzo Sciavicco, & Bruno Siciliano (1991). Closed-loop inverse kinematics schemes for constrained redundant manipulators with task space augmentation and task priority strategy. *IJRR*, 10(4):410–425.

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

Diankov, Rosen (2010). Automated construction of robotic manipulation programs.

Douglas, David H, & Thomas K Peucker (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122.

Flash, Tamar, & Renfrey B Potts (1988). Communication: Discrete Trajectory Planning. *IJRR*, 7(5):48–57.

Frank, Marguerite, Philip Wolfe, et al. (1956). An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110.

Gupta, Harshit, Kyong Hwan Jin, Ha Q Nguyen, Michael T McCann, & Michael Unser (2018). CNN-based projected gradient descent for consistent CT image reconstruction. *IEEE transactions on medical imaging*, 37(6):1440–1453.

Holladay, Rachel, Oren Salzman, & Siddhartha Srinivasa (2019). Minimizing task-space fréchet error via efficient incremental graph search. *RA-L*, 4(2):1999–2006.

Holladay, Rachel M, & Siddhartha S Srinivasa (2016). Distance metrics and algorithms for task space path optimization. *In IROS*, pp 5533–5540. IEEE.

Jiang, Xiangyuan, & Shuai Li (2017). BAS: Beetle Antennae Search Algorithm for Optimization Problems. *ArXiv*, abs/1710.10724.

Joulin, Armand, Kevin Tang, & Li Fei-Fei (2014). Efficient image and video co-localization with frank-wolfe algorithm. *In European Conference on Computer Vision*, pp 253–268. Springer.

Kalakrishnan, Mrinal, Sachin Chitta, Evangelos Theodorou, Peter Pastor, & Stefan Schaal (2011). STOMP: Stochastic trajectory optimization for motion planning. *In ICRA*, pp 4569–4574. IEEE.

Kang, Mincheul, Heechan Shin, Donghyuk Kim, & Sung-Eui Yoon (2020). TORM: Fast and Accurate Trajectory Optimization of Redundant Manipulator given an End-Effector Path. *In IROS*. IEEE.

Kanoun, Oussama, Florent Lamiraux, & Pierre-Brice Wieber (2011). Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task. *T-RO*, 27(4):785–792.

Karaman, Sertac, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, & Seth Teller (2011). Anytime motion planning using the RRT. *In ICRA*, pp 1478–1483. IEEE.

Katyal, Kapil D, Christopher Y Brown, Steven A Hechtman, Matthew P Para, Timothy G McGee, Kevin C Wolfe, Ryan J Murphy, Michael DM Kutzer, Edward W Tunstel, Michael P McLoughlin, et al. (2014). Approaches to robotic teleoperation in a disaster scenario: From supervised autonomy to direct control. *In IROS*, pp 1874–1881. IEEE.

Kew, J Chase, Brian Ichter, Maryam Bandari, Tsang-Wei Edward Lee, & Aleksandra Faust (2019). Neural Collision Clearance Estimator for Fast Robot Motion Planning.

Khan, Ameer Tamoor, Shuai Li, & Xuefeng Zhou (2021). Trajectory optimization of 5-link biped robot using beetle antennae search. *IEEE Transactions on Circuits and Systems II: Express Briefs*.

Kim, Beobkyoon, Terry Taewoong Um, Chansu Suh, & Frank C Park (2016). Tangent bundle RRT: A randomized algorithm for constrained motion planning. *Robotica*, 34(1):202.

Kim, Donghwan, & Jeffrey A Fessler (2018). Adaptive restart of the optimized gradient method for convex optimization. *Journal of Optimization Theory and Applications*, 178(1):240–263.

Kingston, Zachary, Mark Moll, & Lydia E Kavraki (2019). Exploring implicit spaces for constrained sampling-based planning. *IJRR*, 38(10-11):1151–1178.

Kunz, Tobias, & Mike Stilman (2012). Manipulation planning with soft task constraints. *In IROS*, pp 1937–1942. IEEE.

Li, Linjun, Yinglong Miao, Ahmed H Qureshi, & Michael C Yip (2021). MPC-MPNet: Model-Predictive Motion Planning Networks for Fast, Near-Optimal Planning under Kinodynamic Constraints. *RA-L*, 6(3):4496–4503.

Long, Philip, Tarik Keleştemur, Aykut Özgün Önol, & Taşkin Padir (2019). optimization-Based Human-in-the-Loop Manipulation Using Joint Space Polytopes. *In ICRA*, pp 204–210. IEEE.

Luo, Jingru, & Kris Hauser (2012). Interactive generation of dynamically feasible robot trajectories from sketches using temporal mimicking. *In ICRA*, pp 3665–3670. IEEE.

O'donoghue, Brendan, & Emmanuel Candes (2015). Adaptive restart for accelerated gradient schemes. *Foundations of computational mathematics*, 15(3):715–732.

Praveena, Pragathi, Daniel Rakita, Bilge Mutlu, & Michael Gleicher (2019). User-Guided Offline Synthesis of Robot Arm Motion from 6-DoF Paths. *In ICRA*, pp 8825–8831. IEEE.

Rakita, Daniel, Bilge Mutlu, & Michael Gleicher (2018). RelaxedIK: Real-time Synthesis of Accurate and Feasible Robot Arm Motion. *In RSS*.

Rakita, Daniel, Bilge Mutlu, & Michael Gleicher (2019). STAMPEDE: A Discrete-Optimization Method for Solving Pathwise-Inverse Kinematics. *In ICRA*, pp 3507–3513. IEEE.

Schulman, John, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, & Pieter Abbeel (2013). Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization. *In RSS*, volume 9, pp 1–10. Citeseer.

Sinha, Anirban, & Nilanjan Chakraborty (2019). Geometric Search-Based Inverse Kinematics of 7-DoF Redundant Manipulator with Multiple Joint Offsets. *In ICRA*, pp 5592–5598. IEEE.

Stilman, Mike (2007). Task constrained motion planning in robot joint space. *In IROS*, pp 3074–3081. IEEE.

Tong, Hanghang, Christos Faloutsos, & Jia-Yu Pan (2006). Fast random walk with restart and its applications. *In Sixth international conference on data mining*, pp 613–622. IEEE.

Traonmilin, Yann, Jean-François Aujol, & Arthur Leclaire (2020). Projected gradient descent for non-convex sparse spike estimation. *IEEE Signal Processing Letters*, 27:1110–1114.

Vahrenkamp, Nikolaus, Dmitry Berenson, Tamim Asfour, James Kuffner, & Rüdiger Dillmann (2009). Humanoid motion planning for dual-arm manipulation and re-grasping tasks. *In IROS*, pp 2464–2470. IEEE.

Van Laarhoven, Peter JM, & Emile HL Aarts (1987). Simulated annealing. In Simulated annealing: Theory and applications, pp 7–15. Springer.

Yoshikawa, Tsuneo (1985). Manipulability of robotic mechanisms. *IJRR*, 4(2):3–9.

Zhao, Yinghao, Li Yan, Yu Chen, Jicheng Dai, & Yuxuan Liu (2021). Robust and Efficient Trajectory Replanning Based on Guiding Path for Quadrotor Fast Autonomous Flight. *Remote Sensing*, 13(5):972.

Zinkevich, Martin (2003). Online convex programming and generalized infinitesimal gradient ascent. *In Proceedings of the 20th international conference on machine learning*, pp 928–936.

Zucker, Matt, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, & Siddhartha S Srinivasa (2013). CHOMP: Covariant hamiltonian optimization for motion planning. *IJRR*, 32(9-10):1164–1193.