Propagative Distance Optimization for Constrained Inverse Kinematics

Yu Chen[1], Yilin Cai[1], Jinyun Xu[1],
Zhongqiang Ren[2], Guanya Shi[1], and Howie Choset[1]

[1] Carnegie Mellon University, Pittsburgh, PA 15213 USA
[2] Shanghai Jiao Tong University, Shanghai, 200240 China

Abstract. This paper investigates a constrained inverse kinematic (IK) problem that seeks a feasible configuration of an articulated robot under various constraints such as joint limits and obstacle collision avoidance. Due to the high-dimensionality and complex constraints, this problem is often solved numerically via iterative local optimization. Classic local optimization methods take joint angles as the decision variable, which suffers from non-linearity caused by the trigonometric constraints. Recently, distance-based IK methods have been developed as an alternative approach that formulates IK as an optimization over the distances among points attached to the robot and the obstacles. Although distance-based methods have demonstrated unique advantages, they still suffer from low computational efficiency, since these approaches usually ignore the chain structure in the kinematics of serial robots. This paper proposes a new method called propagative distance optimization for constrained inverse kinematics (PDO-IK), which captures and leverages the chain structure in the distance-based formulation and expedites the optimization by computing forward kinematics and the Jacobian propagatively along the kinematic chain. Test results show that PDO-IK runs up to two orders of magnitude faster than the existing distance-based methods under joint limits constraints and obstacle avoidance constraints. It also achieves up to three times higher success rates than the conventional joint-angle-based optimization methods for IK problems. The high runtime efficiency of PDO-IK allows the real-time computation (10−1500 Hz) and enables a simulated humanoid robot with 19 degrees of freedom (DoFs) to avoid moving obstacles, which is otherwise hard to achieve with the baselines.

Keywords: Kinematics · Distance Constraints · Articulated Robots.

1 Introduction

For an articulated robot consisting of rigid links and revolute joints, the inverse kinematics (IK) problem seeks joint angles (i.e., a configuration) such that the end effector(s) reach a given pose, which is a fundamental problem in robotics. This paper focuses on the constrained IK that requires finding a feasible configuration under various constraints, including joint angle limits, and collision

avoidance of workspace obstacles. Constrained IK can only be solved analytically for some specific robots. For the general case, constrained IK is usually formulated as constrained optimization problems and solved numerically via iterative local optimization. This local optimization often suffers from non-linearity due to the kinematic model of the robot, i.e., the mapping from the joint space, commonly represented with *angles*, to the task space, typically represented in the *Euclidean space* [13]. In particular, the kinematic model leads to complicated trigonometric constraints, which can trap the optimization in a local optimum that is highly sub-optimal or even infeasible, especially in the presence of high degrees of freedom (DoF) and cluttered workspace with many obstacles.

To address this challenge, an important class of methods in the literature is to eliminate the trigonometric mapping in the kinematic model by using distance-based optimization [17,9,15,13,7,14,18]. Instead of optimizing the joint angles, distance-based methods attach points to the robot and the obstacles, and reformulate the constrained IK as an optimization over the distances among these points. To name a few, Josep et al. [17] formulated the kinematics of 6-DoF serial robots using a distance matrix [5] and solve the IK with matrix completion leveraging Cayley-Menger determinant [19]. Marić and Giamou et al captures the sparsity of the distance matrix [14] and use sparse bounded-degree sum of squares relaxations [22] to solve IK for spherical joint robots without considering collisions. Marić et al. proposed a distance-geometric framework called Riemannian Trust Region (R-TR) [13] to solve the constrained IK by optimizing the distance matrix using Riemannian optimization.

Despite these advancements, distance-based methods still suffer from low computational efficiency. This paper proposes a new distance-based IK method called PDO-IK. Our key insight is that: Most existing distance-based methods for IK ignore the chain structure of the kinematics of serial robots after the reformulation, and purely focus on optimizing the distances among the points. In contrast, PDO-IK derives a new kinematic model based on the distance between points attached to the robot that captures and leverages such chain structures. PDO-IK computes the forward kinematics and Jacobian propagatively along the kinematic chain, which decomposes the kinematics model into a set of serially connected *units* and iteratively solve for one unit after another along the chain. In our formulation, such *units* are described using the points attached on each frame of robot link. This introduces two advantages: First, the computation of any unknown variable can re-use the variables that have already been computed in its neighbouring unit. Second, the matrix of the distances between points is sparse, which reduces the amount of variables to compute. These advantages allow fast computation of the forward kinematics and Jacobians, and thus expedite the overall optimization. In particular, our technical contributions include both (i) a novel distance-based formulation of the constrained IK, and (ii) an optimization algorithm using augmented Lagrangian based on the proposed formulation and the analysis of its runtime complexity.

For verification, we compare our PDO-IK against both a joint angle-based optimization method and some recent distance-based methods [13] as baselines

in various settings. The results show PDO-IK can often double or triple the success rates (i.e., finding a feasible solution within a runtime limit) of the joint angle method, and run up to two orders of magnitude faster than the existing distance-based methods, especially in cluttered workspace. In addition, PDO-IK demonstrates better numerical robustness than the baselines in the sense that PDO-IK can achieve a small numerical error tolerance that is below 10^{-4}, while the error tolerance of the baselines is often larger than 10^{-3}. Finally, we show the generalization capability of PDO-IK by applying it on a humanoid robot (19 DoFs) with the additional position constraints on the center of mass (CoM). The runtime efficiency of PDO-IK allows the real-time computation ($10-1500$ Hz) and enables the robot to avoid dynamic obstacles, which is otherwise hard to achieve with the baselines.

2 Preliminaries

Robot kinematics describes the relationship between the configuration space $\mathcal{C}$ and the task space $\mathcal{T}$. The mapping $F : \mathcal{C} \to \mathcal{T}$ is the forward kinematics (FK), while its inverse $F^{-1} : \mathcal{T} \to \mathcal{C}$ is the IK. In this paper, we focus on the end-effector pose objective defined in $\mathcal{T}$, joint limits as box constraints defined in $\mathcal{C}$, and collision avoidance constraints.

We consider a serial robot with M revolute joints [3]. We use i as the index for joints and links. For the robot bodies, $i = 1, \ldots, M$ and we define link 0 as the fixed base. Joint i lies between link $i-1$ and link i. We use vector $\boldsymbol{\theta} = [\theta_1, \ldots, \theta_M] \in \mathbb{R}^M$ to represent the joint angles with θ_i being the revolute angle of joint i. We also define unit vectors $\mathbf{x}_i$, $\mathbf{y}_i$, and $\mathbf{z}_i$ to denote the x-, y-, and z-axes of frame $\mathcal{F}_i$, which is the coordinate frame attached to link i. The obstacles in the environment are considered as clusters of points, with a total of N points. $\lceil \cdot \rceil$ denotes the ceiling function. $\sum(\cdot)$ denotes the sum of all elements within a vector or matrix.

2.1 Denavit–Hartenberg Parameters

The kinematic chain formulation in this paper builds upon the proximal Denavit–Hartenberg (DH) convention [4]. As shown in Fig. 1a, we attach the origin $\mathbf{o}_i$ of frame $\mathcal{F}_i$ to the revolute axis of joint i. The direction of the axis $\mathbf{z}_i$ aligns with the revolute axis of joint i. $\mathbf{x}_i$ is perpendicular to and intersects $\mathbf{z}_i$ and $\mathbf{z}_{i+1}$. $\mathbf{y}_i$ is defined by the right-hand rule with $\mathbf{x}_i$ and $\mathbf{z}_i$. The transformation from frame $\mathcal{F}_{i-1}$ to frame $\mathcal{F}_i$ is:

$$ {}^{i-1}\mathbf{T}_i = f(\theta_i; \alpha_{i-1}, a_{i-1}, d_i) = \begin{bmatrix} c_\theta & -s_\theta & 0 & a_{i-1} \\ s_\theta c_\alpha & c_\theta c_\alpha & -s_\alpha & d_i s_\alpha \\ s_\theta s_\alpha & c_\theta s_\alpha & c_\alpha & d_i c_\alpha \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1} $$

where $i \geq 1$, $c_\theta = \cos\theta_i$, $s_\theta = \sin\theta_i$, $c_\alpha = \cos\alpha_{i-1}$, $s_\alpha = \sin\alpha_{i-1}$, a_{i-1} is the revolute radius of $\mathcal{F}_i$ to axis $\mathbf{z}_{i-1}$, α_{i-1} is the angle from $\mathbf{z}_{i-1}$ to $\mathbf{z}_i$ about

[3] Tree-structure robots and parallel robots can also be handled by adding extra constraints. An example is tested in Sec. 5.3.

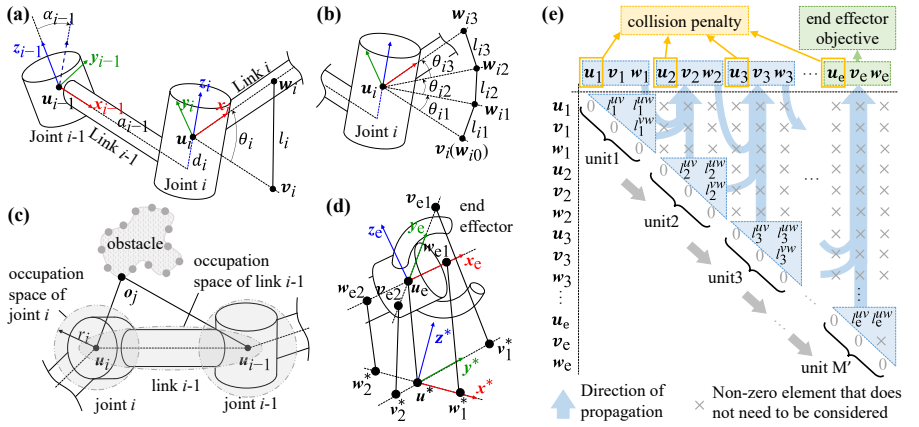

Fig. 1: Kinematics model, constraints, and objective under distance-based representation, as well as the propagation structure in our method. (a) The kinematic chain of a linkage of revolute joints. (b) Joint angle decomposition. (c) Collision avoidance constraint. (d) End effector pose objective. (e) The propagation structure in a diagonal matrix of distances in the forward rollout. The propagation in Jacobian computation follows the inverse direction of the forward rollout.

common normal, d_i is the offset of $\mathcal{F}_i$ to $\mathcal{F}_{i-1}$ along $\mathbf{z}_i$, and θ_i is the revolute angle from $\mathbf{x}_{i-1}$ to $\mathbf{x}_i$ about $\mathbf{z}_i$.

2.2 Quasi-Newton Method

Quasi-Newton method [6] is a variant of Newton's method when the Hessian is unavailable or expensive to compute, which iteratively finds the minimum of a function $f(x)$ by finding the root of its Jacobian ∇f. Quasi-Newton methods approximate Hessian in each iteration k given $f(x_k)$ and $\nabla f(x_k)$. Popular quasi-Newton methods include Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) [12], Broyden's method [2], etc.

3 Kinematics and Constraints Formulation

PDO-IK represents joint space, task space, and constraints with Euclidean distances between points attached on the robot and obstacles. This section primarily discusses our formulation of the robot kinematics chain (Sec. 3.1), joint limit constraints (Sec. 3.2), collision avoidance constraints (Sec. 3.3), and end effector pose objective (Sec. 3.4).

3.1 Kinematic Chain

We formulate the kinematic chain by modifying the proximal DH convention, replacing the joint angles with Euclidean distances between points. As shown in Fig. 1a, we first attach points to the link frames $\mathcal{F}_i$ and then re-parameterize the proximal DH matrix with the Euclidean distances among points.

We attach three points on $\mathcal{F}_i$. The first point $\mathbf{u}_i$ is attached to the origin $\mathbf{o}_i$ to represent the spatial position of joint i. The second point $\mathbf{v}_i$ is of distance l_i^{uv} away from $\mathbf{u}_i$ in the direction of $\mathbf{x}_{i-1}$. The third point $\mathbf{w}_i$ is of distance l_i^{uw} away from $\mathbf{u}_i$ in the direction of $\mathbf{x}_i$. The distance between $\mathbf{v}_i$ and $\mathbf{w}_i$, denoted as l_i^{vw}, is determined by θ_i using the Law of Cosines. The positions of these points and their confinements describe the robot's structure and motion:

$$\mathbf{u}_i = \mathbf{o}_i \tag{2a}$$

$$\mathbf{v}_i - \mathbf{u}_i = l_i^{uv}\mathbf{x}_{i-1} \tag{2b}$$

$$\mathbf{w}_i - \mathbf{u}_i = l_i^{wv}\mathbf{x}_i \tag{2c}$$

$$2l_i^{uv}l_i^{uw}\cos\theta_i = l_i^{uv\,2} + l_i^{uw\,2} - l_i^{vw\,2} \tag{2d}$$

We define the squared distance $L_i = l_i^{vw\,2}$ so that $\cos\theta_i \propto L_i$. For simplicity, we let $l_i^{uv} = l_i^{uw} = 1/\sqrt{2}$ such that

$$\cos\theta_i = 1 - L_i \tag{3a}$$

$$\sin\theta_i = \left(1 - \cos\theta_i{}^2\right)^{\frac{1}{2}} = \left(2L_i - L_i^2\right)^{\frac{1}{2}} \tag{3b}$$

Notably, Eq. 3b assumes that θ_i lies in the range $[0, \pi]$ [14], and accordingly, L_i lies in the range $[0, 2]$. This is because θ_i and $\pi - \theta_i$ correspond to the same value of L_i in our formulation, and we only consider the one that falls within $[0, \pi]$. This introduces an additional challenge when the joint angle limit θ_i is not a subset of $[0, \pi]$, which will be addressed in Sec. 3.2.

After attaching points on $\mathcal{F}_i$, we re-parameterize ${}^{i-1}\mathbf{T}_i$ by substituting Eq. 3a and Eq. 3b to Eq. 1:

$${}^{i-1}\mathbf{T}_i = g(L_i; \alpha_{i-1}, a_{i-1}, d_i) = \begin{bmatrix} 1 - L_i & -\left(2L_i - L_i^2\right)^{\frac{1}{2}} & 0 & a \\ c_\alpha\left(2L_i - L_i^2\right)^{\frac{1}{2}} & c_\alpha(1 - L_i) & -s_\alpha & ds_\alpha \\ s_\alpha\left(2L_i - L_i^2\right)^{\frac{1}{2}} & s_\alpha(1 - L_i) & c_\alpha & dc_\alpha \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

The transformation of frame $\mathcal{F}_i$ with respect to the world frame, which is $\mathbf{T}_i$, can be computed by recursively multiplying from ${}^0\mathbf{T}_1$ to ${}^{i-1}\mathbf{T}_i$. Then, we can extract the position of the joint $\mathbf{u}_i$ from $\mathbf{T}_i$:

$$\mathbf{T}_i = \prod_{p=1}^{i} {}^{p-1}\mathbf{T}_p \tag{5a}$$

$$\mathbf{u}_i = \left[\mathbf{T}_i^{(1,4)},\ \mathbf{T}_i^{(2,4)},\ \mathbf{T}_i^{(3,4)}\right]^\top \tag{5b}$$

3.2 Joint Limit Constraints

Joint limits refer to the restrictions on the range of motion for each joint. This paper focuses on the maximum and minimum joint angle limitations, which are formulated as box constraints $\theta_i \in [\theta_i^{\min}, \theta_i^{\max}]$. Our method handles the joint angle constraints by limiting the corresponding squared distance L_i between $L_i^{\min}$ and $L_i^{\max}$. Then, we convert the box constraints to equality constraints by introducing a squashing function based on the sigmoid function [8].

The formulation of L_i in Eq. 3b assumes that θ_i is within the range $[0, \pi]$. However, the joint angle limits in real-world robotic arms often exceed this range. We address this issue with angle decomposition. Let $\theta_i^{\min} = 0$ and $k = \theta_i^{\max}/\pi >$

0 so that $\theta_i \in [0, k\pi]$. We divide θ_i into $\lceil k \rceil$ sub-angles θ_{im} so that all of the sub-angles lie within the range of $[0, \pi]$:

$$\theta_i = \textstyle\sum_{m=1}^{\lceil k \rceil} \theta_{im}, \quad \theta_{im} \in \left[\theta_i^{\min}/\lceil k \rceil, k\pi/\lceil k \rceil\right] \subseteq [0, \pi] \tag{6}$$

Fig. 1b shows an example of angle decomposition when $m = 3$. By dividing joint angles into sub-angles, we further attach additional points $\mathbf{w}_{im}$ on the side of θ_{im}, and use l_m to describe the distance between $\mathbf{w}_{im}$ and $\mathbf{w}'_{i(m-1)}$. Again, we take the distance between $\mathbf{w}_{im}$ and $\mathbf{u}_i$ as $1/\sqrt{2}$. L_{im} is the corresponding squared distance of θ_{im} such that

$$\cos\theta_{im} = 1 - L_{im} \tag{7}$$

By substituting Eq. 6 into Eq. 7, the range of L_{im} is:

$$1 - \cos\left(\theta_i^{\min}/\lceil k \rceil\right) \leq L_{im} \leq 1 - \cos\left(k\pi/\lceil k \rceil\right) \tag{8}$$

With the above decomposition process, we can compute ${}^{i-1}\mathbf{T}_i$ using L_{im}:

$${}^{i-1}\mathbf{T}_i = \prod_{m=1}^{\lceil k \rceil} g(L_{im}) \tag{9}$$

So far, we have transformed the joint limit constraint to the box constraints on L_i or L_{im}. Box constraints are commonly handled by clamping the variables [10], penalizing over constraint violation [20,11], or converting to equality constraints [1,16]. In this work, we convert the box constraint to equality constraints using a squashing function $s(\omega)$ with the following properties [16]:

$$s(\omega) : \mathbb{R} \to (s^{\min}, s^{\max}) \tag{10a}$$

$$\frac{\mathrm{d}}{\mathrm{d}\omega} s(\omega) \geq 0 \tag{10b}$$

$$s^{\min} = \lim_{\omega \to -\infty} s(\omega), \; s^{\max} = \lim_{\omega \to \infty} s(\omega) \tag{10c}$$

where $\omega \in \mathbb{R}$ is the slack variable. We build our squashing function for L_i upon the sigmoid function [8]

$$\sigma(\omega_i) = \frac{1}{1 + e^{-\omega_i}} : \mathbb{R} \to (0, 1) \tag{11}$$

by linearly scaling $\sigma(\omega_i)$ with $(L_i^{\max} - L_i^{\min})$ and adding a bias term $L_i^{\min}$:

$$L_i = s(\omega_i) = \left(L_i^{\max} - L_i^{\min}\right) \sigma(\omega_i) + L_i^{\min} \tag{12a}$$

$$s(\omega_i) : \mathbb{R} \to (L_i^{\min}, L_i^{\max}) \tag{12b}$$

Eq. 12a and Eq. 12b bound the squared distance L_i within $(L_i^{\min}, L_i^{\max})$, which is a close approximation to $[L_i^{\min}, L_i^{\max}]$. The same constraint conversion can also be applied to L_{im}.

We arrange the slack variables ω_i or ω_{im} into a vector $\boldsymbol{\omega} \in \mathbb{R}^{M'}$, where $M' \geq M$ due to the presence of angle decomposition. In our optimization algorithm, which will be detailed in Sec. 4, we directly optimize over the slack variable $\boldsymbol{\omega}$. This brings several advantages at the cost of additional non-linearity: First, the squashing function is a smooth function that eliminates issues at box constraint boundaries where the gradient might be zero, discontinuous, or undefined. Moreover, the squashing function avoids the need for explicit checks and enforcement of box constraints during the optimization process.

3.3 Collision Avoidance Constraints

Collision avoidance constraints ensure that the robot does not overlap with obstacles. Distance-based IK methods utilize points attached to the robot and use the minimum distance between these points and obstacles for collision avoidance [13]. This models the robot's occupation space as a collection of spheres. Achieving an accurate occupation space requires attaching many points to the robot, making the collision avoidance constraints computationally expensive.

We propose a novel formulation of collision avoidance that only requires the points attached to joints to achieve full-body collision avoidance (Fig. 1c). Moreover, to better handle unstructured obstacles, our framework considers the obstacles as clusters of points rather than as spheres. This is inspired by the common use of LiDAR or depth cameras in robots, which detect and represent environmental obstacles as point clouds. Consequently, clusters of points, or point clouds, are a natural modality for obstacle representations. We separately consider the collision avoidance formulation for robot joints and links.

The occupation space of the joint is considered as a sphere, whose center is at $\mathbf{u}_i$ and the radius is r_i. For joint i, we require the distance between $\mathbf{u}_i$ and each obstacles point $\mathbf{o}_j$ no smaller than a minimum distance r_i:

$$c_{ij}^{\text{joint}} = r_i - ||\mathbf{u}_i - \mathbf{o}_j|| \leq 0 \tag{13}$$

We consider link i in a serial robot as a straight, thin, and long bar or similar shapes that start from point $\mathbf{u}_i$ and ends at $\mathbf{u}_{i+1}$. As the collision avoidance constraint for link i, we enforce the half sum of $||\mathbf{u}_i - \mathbf{o}_j||$ and $||\mathbf{u}_{i+1} - \mathbf{o}_j||$ to be equal or greater than a fixed distance a_i:

$$c_{ij}^{\text{link}} = 2a_i - (||\mathbf{u}_i - \mathbf{o}_j|| + ||\mathbf{u}_{i+1} - \mathbf{o}_j||) \leq 0 \tag{14}$$

Eq. 14 indicates that the occupation space of link i is bounded with a prolate spheroid, whose foci are $\mathbf{u}_i$ and $\mathbf{u}_{i+1}$ and semi-major axis is a_i. Let $c(\boldsymbol{\omega})$ be the vector containing c_{ij}^{joint} and c_{ij}^{link}, the collision avoidance constraint is:

$$c(\boldsymbol{\omega}) \leq 0 \tag{15}$$

3.4 End Effector Pose Objective

The end effector pose objective describes the error between the transformation of the end effector frame and the goal frame. We formulate this objective with the distance between a set of distinct points attached on the end effector frame and the goal frame. We attach $\mathbf{u}_\text{e}$ on the origin of $\mathcal{F}_\text{e}$. We then attach a set of distinct points $\mathbf{w}_{ep}$ and $\mathbf{v}_{ep}$ on $\mathbf{x}_\text{e}$ and $\mathbf{y}_\text{e}$, respectively. The position of these points could be computed from $\mathbf{T}_\text{e}$:

$$\mathbf{w}_{ep} = k_p\mathbf{x}_\text{e} + \mathbf{u}_\text{e},\ \mathbf{v}_{ep} = q_p\mathbf{y}_\text{e} + \mathbf{u}_\text{e} \tag{16}$$

where $\mathbf{x}_\text{e}$ and $\mathbf{y}_\text{e}$ can be extracted from $\mathbf{T}_\text{e}$. $k_p, q_p \in \mathbb{R}$ and $k_p, q_p \neq 0$. Similarly, we attach $\mathbf{u}^*$, $\mathbf{w}_i^*$, and $\mathbf{v}_i^*$ on the goal frame $\mathcal{F}^*$. Fig. 1d shows an example when $n_1 = n_2 = 2$.

We arrange the points attached on $\mathcal{F}_\text{e}$ into matrix $\mathbf{U}_\text{e} = [\mathbf{u}_\text{e}, \mathbf{w}_{\text{e}1}, ..., \mathbf{w}_{en1}, \mathbf{v}_{\text{e}1}, ..., \mathbf{v}_{en2}]^\top \in \mathbb{R}^{(n_1+n_2+1)\times 3}$ and goal points $\mathbf{u}_p^*$ into matrix $\mathbf{U}^* = [\mathbf{u}^*, \mathbf{w}_1^*, ...,$

$\mathbf{w}_{n1}^*, \mathbf{v}_1^*, ..., \mathbf{v}_{n2}^*]^\top \in \mathbb{R}^{(n_1+n_2+1)\times 3}$. The end effector pose objective can be formulated as:

$$J(\boldsymbol{\omega}) = \frac{1}{2}\sum \left((\mathbf{U}_\mathrm{e} - \mathbf{U}^*)^\top (\mathbf{U}_\mathrm{e} - \mathbf{U}^*)\right) \tag{17}$$

Besides the 6-DoF end effector pose objective, our method can also handle 5-DoF objective by setting $n_2 = 0$ or 3-DoF objective by setting $n_1 = n_2 = 0$.

4 Algorithm

So far, we have unified the formulation of kinematic chain, joint limit constraints, collision avoidance constraints and end effector pose objective into Euclidean distance representations. In this section, we introduce our method of solving the distance-based constrained IK.

We formulate the constrained IK as a local optimization problem over the slack variable $\boldsymbol{\omega}$ introduced in Sec. 3.2:

$$\begin{aligned} \min_{\boldsymbol{\omega}} \quad & J(\boldsymbol{\omega}) = \frac{1}{2}\sum (\mathbf{U}_\mathrm{e} - \mathbf{U}^*)^\top (\mathbf{U}_\mathrm{e} - \mathbf{U}^*) \\ \text{s.t.} \quad & \mathbf{U}_\mathrm{e} = FK(\boldsymbol{\omega}) \\ & c(\boldsymbol{\omega}) \leq 0 \end{aligned} \tag{18}$$

where FK is the forward kinematics computation.

The equality constraint, projecting from $\boldsymbol{\omega}$ to the end effector pose via the kinematic chain, can be directly incorporated into $J(\boldsymbol{\omega})$ by replacing $\mathbf{U}$e with $FK(\boldsymbol{\omega})$. Then, we convert the inequality constraint into penalty terms to form an augmented Lagrangian function $L\rho$. Let $c'(\boldsymbol{\omega}) = \max(0, c(\boldsymbol{\omega}))$, the augmented Lagrangian function is:

$$L_\rho(\boldsymbol{\omega}) = J(\boldsymbol{\omega}) + \boldsymbol{\mu}^\top c'(\boldsymbol{\omega}) + \frac{\rho}{2} c'(\boldsymbol{\omega})^\top c'(\boldsymbol{\omega}) \tag{19}$$

where $\boldsymbol{\mu}$ is the Lagrangian multiplier and ρ is the adjust penalty parameter. The constrained IK problem is formulated as finding $\boldsymbol{\omega}^* \in \mathbb{R}^{M'}$ such that

$$\boldsymbol{\omega}^* = \mathrm{argmin} L_\rho(\boldsymbol{\omega}) \tag{20}$$

4.1 Forward Rollout

The forward rollout procedure involves the computation of the forward kinematics FK and objective L_ρ. The forward kinematics computes the position of points attached on joints $\mathbf{U}$ given motion variable $\boldsymbol{\omega}$.

As shown in Fig. 1e and Algorithm 1, the forward rollout is computed propagatively along the direction of the kinematic chain from the base to the end effector. Fig. 1e illustrates a diagonal matrix of distances between points attached on robot. Each element represents the distance between points at the row and column indices. The propagation starts from $\mathbf{u}_0$, $\mathbf{w}_0$, and $\mathbf{v}_0$, which are assumed to be pre-known since they are typically stationary relative to the world frame. We can collect the set of $\mathbf{u}_i$, $\mathbf{w}_i$, and $\mathbf{v}_i$ as a *unit*, then the forward rollout solves for the ith unit and then moves to the $(i+1)$th unit. Additionally, the computation of the variables in the ith unit reuses the pre-computed variables in the $(i-1)$th unit.

Algorithm 1 Forward Rollout

$\mathbf{T}_0 \leftarrow \mathbf{T}_{\text{world}}$
for $i = 1, 2, ..., M$ **do**
  $L_i \leftarrow s(\omega_i)$ ▷ Eq. 12a
  ${}^{i-1}\mathbf{T}_i \leftarrow g(L_i)$ ▷ Eq. 4
  $\mathbf{T}_i \leftarrow \mathbf{T}_{i-1} \cdot {}^{i-1}\mathbf{T}_i$ ▷ Eq. 5a
  Compute $\mathbf{u}_i$ from $\mathbf{T}_i$ ▷ Eq. 5b
end for
$\mathbf{u}_{\text{e}} \leftarrow \mathbf{u}_M$
Compute $\mathbf{w}_{ep1}$ and $\mathbf{v}_{ep2}$ for $p_1 = 1, ..., n_1$ and $p_2 = 1, ..., n_2$.
$\mathbf{U}_{\text{e}} \leftarrow [\mathbf{u}_{\text{e}}, \mathbf{u}_{\text{e}1}, ..., \mathbf{u}_{\text{e}n}]^\top$. ▷ Eq. 16
Compute J from $\mathbf{U}_{\text{e}}$ and $\mathbf{U}^*$. ▷ Eq. 17
$L_\rho \leftarrow J$
for $j = 1, 2, ..., N$ **do**
  for $i = 1, 2, ..., M$ **do**
    $c_{ij}^{\text{joint}} \leftarrow r_i - ||\mathbf{u}_i - \mathbf{o}_j||$ ▷ Eq. 13
    $c_{ij}^{\text{link}} \leftarrow 2a_i - (||\mathbf{u}_i - \mathbf{o}_j|| + ||\mathbf{u}_{i+1} - \mathbf{o}_j||)$ ▷ Eq. 14
    $L_\rho \leftarrow L_\rho + \mu_{ij} c_{ij}^{\text{joint}} + \mu_{ij} c_{ij}^{\text{link}} + \frac{\rho}{2} {c_{ij}^{\text{joint}}}^2 + \frac{\rho}{2} {c_{ij}^{\text{link}}}^2$ ▷ Eq. 19
  end for
end for
return L_ρ

After FK computation, we compute the augmented Lagragian L_ρ, which is composed of end effector pose objective $J(\boldsymbol{\omega})$ and collision penalty $c(\boldsymbol{\omega})$. $J(\boldsymbol{\omega})$ is computed with $\mathbf{U}_{\text{e}}(\boldsymbol{\omega})$ obtained from FK. For collision penalties, we loop through every robot-attached point $\mathbf{u}_i$ and obstacle point $\mathbf{o}_j$ to compute c_{ij}^{joint} and c_{ij}^{link}.

4.2 Jacobian Computation

The Jacobian $\nabla_{\boldsymbol{\omega}} L_\rho$ is the derivatives of augmented Lagrangian L_ρ to the variables $\boldsymbol{\omega}$, which can be decomposed into the derivatives of end effector pose objective $\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega})$ and collision penalties $\nabla_{\boldsymbol{\omega}} c(\boldsymbol{\omega})$:

$$\nabla_{\boldsymbol{\omega}} L_\rho = \nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) + \boldsymbol{\mu} \nabla_{\boldsymbol{\omega}} c(\boldsymbol{\omega}) + \rho c(\boldsymbol{\omega}) \odot \nabla_{\boldsymbol{\omega}} c(\boldsymbol{\omega}) \tag{21a}$$

$$\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) = \frac{\mathrm{d}J}{\mathrm{d}\mathbf{T}_{\text{e}}} \frac{\mathrm{d}\mathbf{T}_{\text{e}}}{\mathrm{d}\boldsymbol{\omega}} \tag{21b}$$

$$\nabla_{\boldsymbol{\omega}} c(\boldsymbol{\omega}) = \sum_{i=1}^{M} \frac{\partial c}{\partial \mathbf{u}_i} \frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}\boldsymbol{\omega}} \tag{21c}$$

The key idea of Jacobian computation in our method is to propagate along the backward direction of the kinematic chain leveraging reverse accumulation. Notably, the Jacobian computation follows the forward rollout, allowing it to reuse the results from the forward rollout. As shown in Algorithm 2, we first compute the derivatives of collision penalties with respect to the position of points attached to the robot ($\partial c / \partial \mathbf{u}_i$). Then, we compute the derivative of the end effector pose objective to the end frame ($\partial J / \partial \mathbf{T}\text{e}$), and finally compute the derivatives of the position of the points to the variables ($\partial \mathbf{u}_i / \partial \boldsymbol{\omega}$). $\partial c / \partial \mathbf{u}_i$ in Eq. 21c is computed with

$$\frac{\partial c}{\partial \mathbf{u}_i} = \sum_{j=1}^{N} \left(\frac{d}{d\mathbf{u}_i} c_{ij}^{\text{joint}} + \frac{\partial}{\partial \mathbf{u}_i} c_{ij}^{\text{link}} + \frac{\partial}{\partial \mathbf{u}_i} c_{(i+1)j}^{\text{link}} \right) \tag{22a}$$

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{u}_i} c_{ij}^{\text{joint}} = (\mu_{ij} + \rho c_{ij}^{\text{joint}}) \frac{\mathbf{u}_i - \mathbf{o}_j}{||\mathbf{u}_i - \mathbf{o}_j||} \cdot \mathbb{1}\left(c_{ij}^{\text{joint}} \geq 0\right) \tag{22b}$$

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{u}_i} c_{ij}^{\text{link}} = (\mu_{ij} + \rho c_{ij}^{\text{link}}) \frac{\mathbf{u}_i - \mathbf{o}_j}{||\mathbf{u}_i - \mathbf{o}_j||} \cdot \mathbb{1}\left(c_{ij}^{\text{link}} \geq 0\right) \tag{22c}$$

Algorithm 2 Jacobian Computation

for $j = N, N-1, ..., 1$ **do**
  for $i = M, M-1, ..., 1$ **do**
    if $c_{ij}^{\text{joint}} > 0$ **then**
      $\mathbf{s}_1 \leftarrow (\mu_{ij} + \rho c_{ij}^{\text{joint}})(\mathbf{u}_i - \mathbf{o}_j)/||\mathbf{u}_i - \mathbf{o}_j||$ ▷ Eq. 22b
      Add $[\mathbf{s}_1^\top, 0]^\top$ to the last column of $\partial L_\rho / \partial \mathbf{T}_i$. ▷ Eq. 22a
    end if
    if $c_{ij}^{\text{link}} > 0$ **then**
      $\mathbf{s}_2 \leftarrow (\mu_{ij} + \rho c_{ij}^{\text{link}})(\mathbf{u}_i - \mathbf{o}_j)/||\mathbf{u}_i - \mathbf{o}_j||$
      Add $[\mathbf{s}_2^\top, 0]^\top$ to the last column of $\partial L_\rho / \partial \mathbf{T}_i$. ▷ Eq. 22c
      $\mathbf{s}_3 \leftarrow (\mu_{(ij} + \rho c_{(i-1)j}^{\text{link}})(\mathbf{u}_{i-1} - \mathbf{o}_j)/||\mathbf{u}_{i-1} - \mathbf{o}_j||$ ▷ Eq. 22c
      Add $[\mathbf{s}_3^\top, 0]^\top$ to the last column of $\partial L_\rho / \partial \mathbf{T}_{i-1}$. ▷ Eq. 22a
    end if
  end for
end for
$\partial L_\rho / \partial \mathbf{T}_M \leftarrow \partial J / \partial \mathbf{T}_{\text{e}}$ ▷ Eq. 23
for $i = M, M-1, ..., 1$ **do**
  $\partial L_\rho / \partial^{i-1}\mathbf{T}_i \leftarrow \mathbf{T}_{i-1}^\top \cdot (\partial L_\rho / \partial \mathbf{T}_i)$ ▷ Eq. 24b
  $\partial^{i-1}\mathbf{T}_i / \partial L_i \leftarrow g'(L_i)$ ▷ Eq. 24c
  $\partial L_i / \partial \omega_i \leftarrow (L_i^{\max} - L_i^{\min})\sigma(\omega_i)(1 - \sigma(\omega_i))$ ▷ Eq. 24d
  $\partial L_\rho / \partial \omega_i \leftarrow \left(\sum (\partial L / \partial^{i-1}\mathbf{T}_i) \cdot (\partial^{i-1}\mathbf{T}_i / \partial L_i)\right)(\partial L_i / \partial \omega_i)$ ▷ Eq. 24a
  $\partial L_\rho / \partial \mathbf{T}_{i-1} \leftarrow \partial L_\rho / \partial \mathbf{T}_{i-1} + (\partial L_\rho / \partial \mathbf{T}_i) \cdot^{i-1} \mathbf{T}_i^\top$ ▷ Eq. 25
end for
$\nabla_{\boldsymbol{\omega}} L_\rho \leftarrow [\partial L_\rho / \partial \omega_1, \partial L_\rho / \partial \omega_2, ..., \partial L_\rho / \partial \omega_i, ..., \partial L_\rho / \partial \omega_M]^\top$
return $\nabla_{\boldsymbol{\omega}} L_\rho$

where the term $\partial c_{(i+1)j}^{\text{link}} / \partial \mathbf{u}_i$ is dropped when $i = M$. The term $\mathbf{u}_i - \mathbf{o}_j$ and $||\mathbf{u}_i - \mathbf{o}_j||$ are already pre-computed in the forward rollout by Eq. 13. The term $\mathbb{1}(\cdot)$ is the indicator function, which equals to 1 when $(\cdot)$ is true otherwise 0. The term $\partial J / \partial \mathbf{T}_{\text{e}}$ in Eq. 21b is computed with:

$$\frac{\partial}{\partial \mathbf{T}_{\text{e}}} J = \begin{bmatrix} \sum_{p=1}^{n_1} k_p(\mathbf{w}_{ep} - \mathbf{w}^*) & \sum_{p=1}^{n_2} q_p(\mathbf{v}_{ep} - \mathbf{v}^*) & 0 & \mathbf{u}_{\text{e}} - \mathbf{u}^* \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{23}$$

where all the elements are already computed in Eq. 16 and Eq. 17. By computing the derivatives of L_ρ to the position of points attached on the robot, we can now solve for $\partial L_\rho / \partial \omega_i$ with:

$$\frac{\partial}{\partial \omega_i} L_\rho = \left(\sum \frac{\partial L_\rho}{\partial^{i-1}\mathbf{T}_i} \cdot \frac{\partial^{i-1}\mathbf{T}_i}{\partial L_i} \right) \frac{\partial L_i}{\partial \omega_i} \tag{24a}$$

$$\frac{\partial L_\rho}{\partial^{i-1}\mathbf{T}_i} = \mathbf{T}_{i-1}^\top \cdot \frac{\partial L_\rho}{\partial \mathbf{T}_i} \tag{24b}$$

$$\frac{\partial^{i-1}\mathbf{T}_i}{\partial L_i} = \frac{\mathrm{d}g(L_i)}{L_i} = \begin{bmatrix} -1 & -(1-L_i)\left(2L_i - L_i^2\right)^{-\frac{1}{2}} & 0\ 0 \\ c_\alpha (1-L_i)\left(2L_i - L_i^2\right)^{-\frac{1}{2}} & -c_\alpha & 0\ 0 \\ s_\alpha (1-L_i)\left(2L_i - L_i^2\right)^{-\frac{1}{2}} & -s_\alpha & 0\ 0 \\ 0 & 0 & 0\ 0 \end{bmatrix} \tag{24c}$$

$$\frac{\partial L_i}{\partial \omega_i} = (L_i^{\max} - L_i^{\min}) \frac{\mathrm{d}}{\mathrm{d}\omega_i} \sigma(\omega_i) = (L_i^{\max} - L_i^{\min}) \sigma(\omega_i)(1 - \sigma(\omega_i)) \tag{24d}$$

Algorithm 3 Inverse Kinematics

$\boldsymbol{\omega}_0^* \leftarrow \mathbf{0}$, $\boldsymbol{\mu} \leftarrow \mathbf{0}$, $\rho \leftarrow 1$, $\alpha \leftarrow 10$, $c_{\text{last}} \leftarrow \infty$.
for iteration $k = 1, 2, ..., k_{\max}$ **do**
  $\boldsymbol{\omega}_k^* \leftarrow L - BFGS(\boldsymbol{\omega}_{k-1}^*)$, where L_ρ is computed with Algorithm 1 and $\nabla_{\boldsymbol{\omega}_k} L_\rho$ is computed with Algorithm 2.
  if $\max(c(\boldsymbol{\omega}_k^*)) < c_{\text{tol}}$ **or** $\max(c(\boldsymbol{\omega}_k^*)) \geq \beta c_{\text{last}}$ **then**
    $\boldsymbol{\omega}^* \leftarrow \boldsymbol{\omega}_k^*$
    break
  end if
  $c_{\text{last}} \leftarrow \max(c(\boldsymbol{\omega}_k^*))$
  $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \rho c(\boldsymbol{\omega}_k^*)$
  $\rho \leftarrow \alpha\rho$
end for
Compute $\boldsymbol{\theta}^*$ with $\boldsymbol{\omega}^*$ ▷ Eq. 12a and Eq. 26
return $\boldsymbol{\theta}^*$

where $\mathbf{T}_{i-1}$ is already computed in Eq. 5a, $\sigma(\omega_i)$ is already computed in Eq. 11, and $\partial L_\rho / \partial \mathbf{T}_i$ can be recursively computed from joint $i+1$, collision penalty, and end effector pose objectives:

$$\frac{\partial L_\rho}{\partial \mathbf{T}_i} = \begin{cases} \frac{\partial c}{\partial \mathbf{T}_i} + \frac{\partial L_\rho}{\partial \mathbf{T}_{i+1}} \cdot^i \mathbf{T}_{i+1}^\top , & i < M \\ \frac{\partial c}{\partial \mathbf{T}_i} + \frac{\partial J}{\partial \mathbf{T}_{\text{e}}} , & i = M \end{cases} \tag{25}$$

4.3 Inverse Kinematics

We first solve Eq. 19 for a local optimal solution $\boldsymbol{\omega}^*$, and then compute $\boldsymbol{\theta}^*$ from $\boldsymbol{\omega}^*$. As shown in Algorithm 3, our method iteratively minimize L_ρ and update the Lagrangian multiplier $\boldsymbol{\mu}$ and ρ. Within each loop k, we use quasi-Newton method to solve for $\text{argmin} L_\rho$. The forward rollout and Jacobian computation are implemented following Algorithm 1 and Algorithm 2, respectively. The Hessian matrix are approximated with the Jacobian in quasi-Newton convention. In our framework, we use L-BFGS as our quasi-Newton-based solver. After solving for $\boldsymbol{\omega}_k^* = \text{argmin} L_\rho$, we update Lagrangian multipliers $\boldsymbol{\mu}$ and ρ. Our method checks $\max(c(\boldsymbol{\omega}_k^*))$ in every iteration and will terminate if $\max(c(\boldsymbol{\omega}_k^*)) < c_{\text{tol}}$ **or** $\max(c(\boldsymbol{\omega}_k^*)) \geq \beta c_{\text{last}}$, where $\beta < 1$ and $c_{\text{last}} = \max(c(\boldsymbol{\omega}_{k-1}^*))$.

By obtaining $\boldsymbol{\omega}^*$, we solve $\boldsymbol{\theta}^*$ from $\boldsymbol{\omega}^*$. We first compute L_i^* using Eq. 12a, and then compute θ_i^* from L_i^*:

$$\theta_i^* = \sum_{m=1}^{\lceil k \rceil} \arccos\left(1 - L_{im}^*\right) + \min\left(0, \theta_i^{\min}\right) \tag{26}$$

4.4 Complexity Analysis

Our optimization framework is a combination of augmented Lagrangian optimization and a quasi-Newton optimizer. It's complexity is determined by the complexity of a single iteration, which involves forward rollout (Algorithm 1), Jacobian computation (Algorithm 2), Hessian approximation, and variable update. The forward rollout and Jacobian computation propagate through all of the points attached on the robot and the points of the obstacles, with a complexity of $O(M'N)$. The complexity of Hessian approximation process in quasi-Newton solver is $O(M'^2)$. The complexity of variable update is $O(M')$. In summary, the complexity of the algorithm is $O(M'N + M'^2)$. Notably, M' is only determined

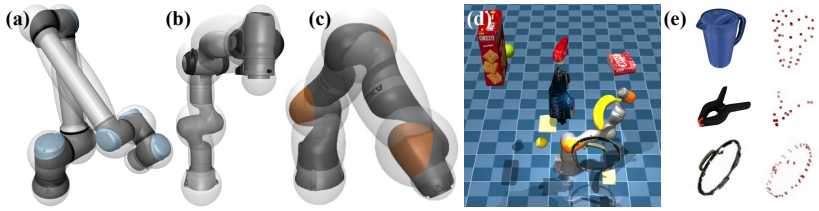

Fig. 2: Visualization of experimental setups. (a)-(c) Robot arm platforms (UR10, Franka, and KUKA). Their occupation space are composed of spheres or spheroids in our formulation, visualized here as translucent hulls. (d) An example of a KUKA robot in the environment with 9 random obstacles. (e) Visualization of obstacles as point clusters.

by the DoF of the robot and joint limit, which is commonly small (less than 20). On the other hand, N depends on the environment and could reach hundreds. Given a robot, the complexity of our algorithm is linear to the number of cluttered points in the environment.

5 Experiments

In this section, we conduct experiments to demonstrate the efficiency, effectiveness, and generalization capability of our algorithm.

5.1 Efficiency and Effectiveness Comparison

We conduct simulation experiments to benchmark the efficiency of our algorithm in runtime, as well as its effectiveness in handling end effector pose objective, collision avoidance constraints, and joint limit constraints.

We compare our method with three baselines, two of which are recent distance-based IK algorithms. The first baseline is R-TR [13], a distance-geometric-based IK algorithm that optimizes the distance matrix on the Riemannian manifold using the Trust Region method. The second baseline is Riemannian Conjugate Gradient (R-CG), also introduced in [13], which uses the Conjugate Gradient method to solve for the distance matrix. Since the baselines can only handle 3- or 5-DoF end effector poses, we focus on 5-DoF end effector pose objectives throughout the experiments. We further build a third baseline: a variant of PDO-IK, which employs the same collision avoidance constraints, end effector pose objectives, and optimization method as PDO-IK, but represents robot kinematics using joint angles and uses Limited-memory Broyden–Fletcher– Goldfarb–Shanno-Bounded (L-BFGS-B) [23] to directly handle joint limits as box constraints. We name this third baseline "Angle-LBFGS-B". For both PDO-IK and Angle-LBFGS-B, we set $\beta = 0.99$.

We test all the methods on 3 popular commercial robot arms: UR10, KUKA-IIWA, and Franka [4]. UR10 has 6 DoFs and every joint can rotate from $-360°$

[4] UR10: `https://www.universal-robots.com/`. KUKA-IIWA: `https://www.kuka.com/`. Franka: `https://franka.de/`.

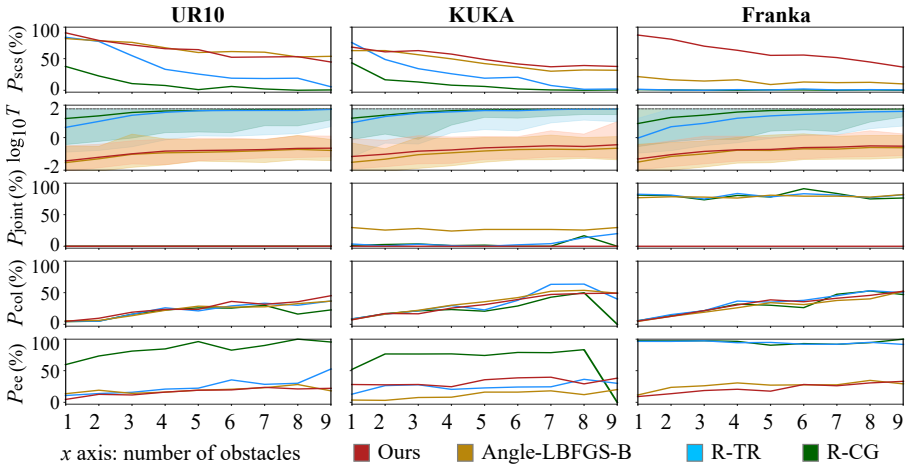

Fig. 3: Experimental results on UR10, KUKA, and Franka. The x-axis are the number of obstacles. The y-axis are success rate, logarithm of runtime in seconds, joint limit violation rate, collision rate, and end effector objective failure rate.

to 360°. KUKA-IIWA and Franka have 7 DoFs and have tighter joint limits. As mentioned in Sec. 3.3, our collision constraint formulation takes the occupation space of joints and links as sphere and spheroid, respectively. The occupation space of these robots is shown in Fig.2a-c.

We set up various scenarios with 1 to 9 random obstacles. For each number of obstacles and each robot, we generate 200 scenarios for experiments. We sample different objects form YCB dataset as obstacles [3], with their corresponding point clouds also provided. An example of an experiment scenario is shown in Fig.2d. The environment generation and method implementation for each scenario follow these steps:

1. Place the robot on the origin of the world frame, randomly sample a configuration $\boldsymbol{\theta}_{\text{rand}}$ from joint space $\mathcal{C}$ from a uniform distribution over the joint angle limits. Record the end effector pose as the target.
2. Randomly generate obstacles that are collision free to $\boldsymbol{\theta}_{\text{rand}}$. The position of the obstacles are randomly sampled from a uniform distribution within a specific range. The range of the x-, y-, and z-coordinates of the center of the objects are $[-0.6, 0.6]$, $[-0.6, 0.6]$, and $[0, 1.2]$ meters, respectively. The objects are randomly scaled with ratio of $[1, 3]$. We use Moveit! collision checker [5] to check for collision. The point cloud of the obstacle is downsampled with the Voxel Grid filter from the Point Cloud Library (PCL) [6] with voxel grid leaf size equals to 0.1m. Examples of obstacles and their point cloud are shown in Fig. 2e. The average amount of points in scenarios of 1 to 9 obstacles are 24, 47, 68, 87, 101, 110, 122, 130, and 136, respectively.

[5] Moveit!: `https://moveit.ros.org/`.
[6] Point Cloud Library: `https://pointclouds.org/`.

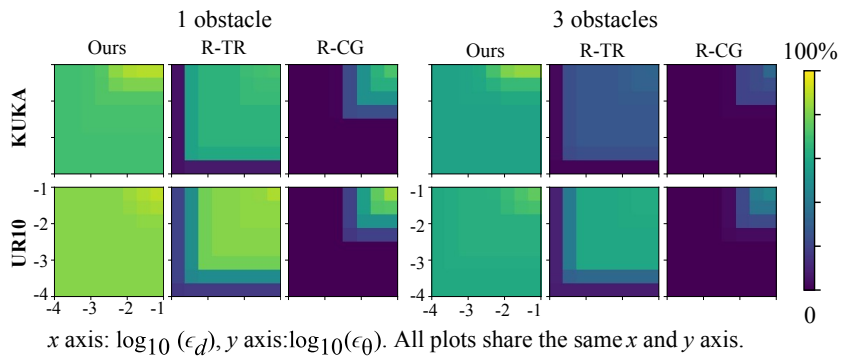

Fig. 4: Convergence precision experiments results.

3. Run each IK method given the target end effector pose and clusters of points. We use $\boldsymbol{\theta}_0 = \mathbf{0}$ as initialization. We set time limits of 60 seconds for each algorithm. All IK algorithms are implemented in Python on a desktop computer with Intel Core i9 CPU with 128 GB RAM.

We report the success rate P_{scs}, which is the percentage of experiments that satisfies the following criteria: (1) The solution is reported within the time limit. (2) The solution is collision-free to the obstacles detected by Moveit! collision checker. (3) The end effector position error ϵ_d and rotation error ϵ_θ are less than 0.01 m and 0.01 rad, respectively. (4) The joint limit violation is within 1% of the joint angle limit range. In addition, we also report joint limit failure rate P_{joint}, collision rate P_{col}, and end effector pose failure rate P_{ee}, which are the proportion of the number of solutions that fails to satisfy criteria (2), (3), and (4), respectively, to the number of total solutions generated. The logarithm of runtime $\log_{10} T$ where T is in seconds, is also reported for all IK methods.

The experiment results are shown in Fig.3. For all scenarios and robot platforms, our method achieves a comparable or higher success rate than the baselines, especially when the amount of obstacles in the environment gets higher. Moreover, our method runs up to two orders of magnitude faster than the previous distance-based methods. Our method also achieves lower or comparable P_{ee} compared to previous distance-based methods. The collision rate P_{col} of our method and the baselines are similar. Our method has 0 joint limit failure rate since our joint angle is strictly bounded with the squashing function. Although Angle-LBFGS-B has comparable or slightly faster speed than PDO-IK due to their similar approaches, it can only achieve a comparable success rate to PDO-IK on the UR10, which has 6 DoFs, with wide joint limits, and a simple kinematic chain. On robots with 7 DoFs with tight joint limits and more complex kinematic chains, such as the Franka, Angle-LBFGS-B performs much worse than PDO-IK. This comparison shows the advantage of distance-based representation over conventional angle-based representation.

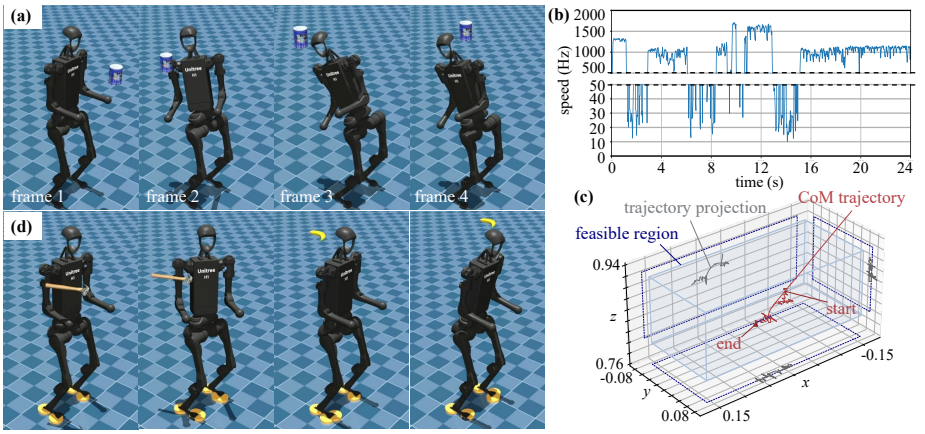

Fig. 5: Dynamic obstacle avoidance for humanoid robot. (a) Key frames of the H1 robot avoiding obstacles. (b) Speed of PDO-IK in C++ implementation. (c) The trajectory of CoM of the robot and its feasible region. (d) Demonstrations of some additional collision avoidance experiments.

5.2 Solution Accuracy Comparison

We check the solution accuracy with the optimal solution achieved among PDO-IK, R-TR, and R-CG. The solution accuracy measures how close the algorithm's final solution is to the true optimal solution. In this section, we compare the solution accuracy of PDO-IK, R-TR, and R-CG by counting the proportion of end effector objectives that satisfies different tolerance levels of ϵ_d and ϵ_θ.

Fig. 4 illustrates the success rate under different tolerance levels of ϵ_d and ϵ_θ on KUKA and UR10 when the number of obstacles is 1 and 3. Our method remains a high and relative consistent success rate when the tolerance varies from 10^{-4} to 10^{-1}. The success rate of R-TR and R-CG, however, significantly drop when the tolerance is is below 10^{-3} and 10^{-2}, respectively. This experiment shows that PDO-IK achieves higher solution accuracy than the baselines.

5.3 Humanoid Robot Avoiding Dynamic Obstacle

We further apply our algorithm on humanoid robot (Fig. 5). In this experiment, we let a can (`002_master_chef_can` from YCB dataset, its downsampled point cloud contains 168 points) fly to the H1 robot [7] in a pre-defined trajectory. The H1 robot is a humanoid robot that contains 19 DoFs (5 on each leg, 4 on each arm, 1 on the torso). The H1 robot needs to avoid the can in real time and remains its feet in a fixed position. Moreover, the CoM needs to maintain within a feasible region to ensure the stability of the robot. In this experiment, the left ankle of the robot is considered as the base link, which is fixed at $[0.2, 0, 0]^\top$ and the right ankle is considered as the end effector. We fix the right ankle by setting its objective position at $\mathbf{u}^* = [-0.2, 0, 0]^\top$.

[7] Unitree Robotics: `https://www.unitree.com/h1/`

For the robot stability, we add constraints to robot CoM position $\mathbf{c} \in \mathbb{R}^3$:

$$[-0.16, -0.07, 0.8]^\top < \mathbf{c} < [0.16, 0.075, 0.94]^\top \tag{27}$$

where $\mathbf{c}$ is the weighted combination of every links' center position:

$$\mathbf{c} = \sum_{i=1}^{M} m_i \mathbf{c}_i = \sum_{i=1}^{M} m_i [\mathbf{T}_{\text{mass},i}(1,4),\ \mathbf{T}_{\text{mass},i}(2,4),\ \mathbf{T}_{\text{mass},i}(3,4)]^\top \tag{28a}$$

$$\mathbf{T}_{\text{mass},i} = \mathbf{T}_i \cdot {}^i\mathbf{T}_{\text{mass},i} \tag{28b}$$

where m_i is the weight of link i and ${}^i\mathbf{T}_{\text{mass},i}$ is the transformation matrix of the CoM of link i with respect to its own reference frame. After PDO-IK solves for a feasible solution $\boldsymbol{\theta}^*$, we use a PD controller to control the motors to reach $\boldsymbol{\theta}^*$.

The algorithm is implemented in C++ on a laptop computer with AMD Ryzen 7 CPU with 16GB RAM. Fig. 5a shows a series of key frames of the simulation experiment on Mujoco [21]. The speed of the algorithm varies from 10 to 1500 Hz (Fig. 5b), indicating that the algorithm is capable for real-time collision avoidance in dynamic environments.

The trajectory of the CoM is shown in Fig. 5c. The blue box is the feasible region defined by Eq. 27. The CoM occasionally violates the CoM constraints of no more than 0.01m but will quickly return to the feasible region thereafter. Fig. 5d shows some additional demonstrations of the humanoid avoiding dynamic obstacles (`048_hammer` and `011_banana` from the YCB dataset).

6 Conclusions and Future Work

We present PDO-IK, a distance-based algorithm for constrained IK problems. It addresses inefficiencies in previous distance-based methods by leveraging the kinematic chain and new formulations for joint limit constraints, collision avoidance constraints, and end effector objectives. Experiments show that our method runs faster, can handle various constraints, and provide more accurate solutions than recent distance-based methods. Finally, experiments on the H1 humanoid robot demonstrate the generalization ability of our method and its usage for collision avoidance in dynamic environments.

Our method has several limitations to be addressed in our future work. First, the angle decomposition process introduces extra DoFs and reduces the speed of the algorithm. Additionally, the collision avoidance constraints assume joints are spherical and links are spheroidal, which might poorly approximate complex robot shapes. Better approximations could be achieved by attaching more points to the robot at the cost of potentially higher computational burden. Moreover, this paper doesn't consider robot self-collision but we believe such constraints can be achieved by constraining the distances between points attached on the robot. Finally, tests on different types of robot structures can be conducted to further broaden the application of the algorithm.

References

1. Baumgärtner, K., Wang, Y., Zanelli, A., Diehl, M.: Fast nonlinear model predictive control using barrier formulations and squashing with a generalized gauss-newton hessian. In: 2022 IEEE 61st Conference on Decision and Control (CDC). pp. 558–563. IEEE (2022)
2. Broyden, C.G.: A class of methods for solving nonlinear simultaneous equations. Mathematics of computation **19**(92), 577–593 (1965)
3. Calli, B., Singh, A., Walsman, A., Srinivasa, S., Abbeel, P., Dollar, A.M.: The ycb object and model set: Towards common benchmarks for manipulation research. In: 2015 international conference on advanced robotics (ICAR). pp. 510–517. IEEE (2015)
4. Craig, J.J.: Introduction to robotics. Pearson Educacion (2006)
5. Crippen, G.M., Havel, T.F., et al.: Distance geometry and molecular conformation, vol. 74. Research Studies Press Taunton (1988)
6. Dennis, Jr, J.E., Moré, J.J.: Quasi-newton methods, motivation and theory. SIAM review **19**(1), 46–89 (1977)
7. Giamou, M., Marić, F., Rosen, D.M., Peretroukhin, V., Roy, N., Petrović, I., Kelly, J.: Convex iteration for distance-geometric inverse kinematics. IEEE Robotics and Automation Letters **7**(2), 1952–1959 (2022)
8. Han, J., Moraga, C.: The influence of the sigmoid function parameters on the speed of backpropagation learning. In: International workshop on artificial neural networks. pp. 195–201. Springer (1995)
9. Han, L., Rudolph, L.: Inverse kinematics for a serial chain with joints under distance constraints. In: Robotics: Science and systems (2006)
10. Khokar, K., Beeson, P., Burridge, R.: Implementation of kdl inverse kinematics routine on the atlas humanoid robot. Procedia Computer Science **46**, 1441–1448 (2015)
11. Lantoine, G., Russell, R.: A hybrid differential dynamic programming algorithm for robust low-thrust optimization. In: AIAA/AAS Astrodynamics Specialist Conference and Exhibit. p. 6615 (2008)
12. Liu, D.C., Nocedal, J.: On the limited memory bfgs method for large scale optimization. Mathematical programming **45**(1-3), 503–528 (1989)
13. Marić, F., Giamou, M., Hall, A.W., Khoubyarian, S., Petrović, I., Kelly, J.: Riemannian optimization for distance-geometric inverse kinematics. IEEE Transactions on Robotics **38**(3), 1703–1722 (2021)
14. Marić, F., Giamou, M., Khoubyarian, S., Petrović, I., Kelly, J.: Inverse kinematics for serial kinematic chains via sum of squares optimization. In: 2020 IEEE International Conference on Robotics and Automation (ICRA). pp. 7101–7107. IEEE (2020)
15. Marić, F., Giamou, M., Petrović, I., Kelly, J.: Inverse kinematics as low-rank euclidean distance matrix completion. arXiv preprint arXiv:2011.04850 (2020)
16. Marti-Saumell, J., Solà, J., Mastalli, C., Santamaria-Navarro, A.: Squash-box feasibility driven differential dynamic programming. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 7637–7644. IEEE (2020)
17. Porta, J.M., Ros, L., Thomas, F.: Inverse kinematics by distance matrix completion (2005)
18. Porta, J.M., Ros, L., Thomas, F., Torras, C.: A branch-and-prune solver for distance constraints. IEEE Transactions on Robotics **21**(2), 176–187 (2005)

19. Sippl, M.J., Scheraga, H.A.: Cayley-menger coordinates. Proceedings of the National Academy of Sciences **83**(8), 2283–2287 (1986)
20. Tassa, Y., Mansard, N., Todorov, E.: Control-limited differential dynamic programming. In: 2014 IEEE International Conference on Robotics and Automation (ICRA). pp. 1168–1175. IEEE (2014)
21. Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 5026–5033. IEEE (2012). https://doi.org/10.1109/IROS.2012.6386109
22. Weisser, T., Lasserre, J.B., Toh, K.C.: Sparse-bsos: a bounded degree sos hierarchy for large scale polynomial optimization with sparsity. Mathematical Programming Computation **10**, 1–32 (2018)
23. Zhu, C., Byrd, R.H., Lu, P., Nocedal, J.: Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. ACM Transactions on mathematical software (TOMS) **23**(4), 550–560 (1997)