Propagative Distance Optimization for Constrained Inverse Kinematics

Yu Chen

CMU-RI-TR-24-48

August 6th, 2024

School of Computer Science
The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania

Thesis Committee:
Howie Choset, *chair*
Guanya Shi
Chao Cao

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

To all people and robots that are important to my life.

Abstract

This work 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 work 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.

Acknowledgments

I would like to express my deepest gratitude to my family. Your unwavering support, encouragement, and understanding have been the bedrock upon which I have built my academic journey.

I am immensely grateful to my mentors and people who provided me guidance: Dr. Howie Choset, Dr. Guanya Shi, Dr. Zhongqiang Ren, Mr. Shuo Yang, Mr. Ben Brown, Mr. Lu Li, and Mrs. Yizhu Gu. Your expertise, patience, and insightful feedback have been invaluable throughout this process. Your dedication to me has been a source of constant motivation and inspiration. Thank you for believing in me and for your continuous support.

I would also like to extend my heartfelt thanks to my colleagues and friends: Ms. Jinyun Xu, Mr. Yilin Cai, Mr. Fujun Ruan, Mr. Chao Cao, etc. Your camaraderie, assistance, and moral support have made this journey more enjoyable and manageable. I am fortunate to have you by my side, and I cherish the moments we have shared together.

Contents

1 Introduction **1**

2 Related Works **5**
2.1 Angle-based Formulations for Inverse Kinematics 5
2.2 Distance-based Formulation for Inverse Kinematics 7

3 Preliminaries **11**
3.1 Denavit–Hartenberg Parameters 11
3.2 Quasi-Newton Method 13
3.3 Augmented Lagrangian 13

4 Kinematics Framework **15**
4.1 Kinematic Chain 15
4.2 Joint Limit Constraints 17
4.3 Collision Avoidance Constraints 19
4.4 End Effector Pose Objective 21

5 Kinematics Algorithm **23**
5.1 Forward Rollout 24
5.2 Jacobian Computation 26
5.3 Inverse Kinematics 28
5.4 Complexity Analysis 29

6 Experiments **31**
6.1 Efficiency and Effectiveness Comparison 31
6.2 Solution Accuracy Comparison 36
6.3 Humanoid Robot Avoiding Dynamic Obstacle 37

7 Conclusions **41**

A DH Parameters in Experiments **43**
A.1 KUKA 44
A.2 Franka 45
A.3 UR10 46
A.4 H1 47

Bibliography **51**

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

3.1 Link frame attachment on articulated robots. 12

4.1 Distance-based kinematic chain formulation. 16
4.2 Example of angle decomposition for joint limit formulation. 17
4.3 Collision avoidance constraints formulation visualization. 20
4.4 Example of end effector objective formulation. 21

5.1 Visualization of propagative computation in forward kinematics. . . . 24

6.1 Robot arm platforms (UR10, Franka, and KUKA) and their occupation space visualized as translucent hulls. 32
6.2 An example of the scene generated. 33
6.3 Experimental results. 34
6.4 Convergence precision experiments results on KUKA robot. 35
6.5 Convergence precision experiments results on UR10 robot. 36
6.6 Convergence precision experiments results on Franka robot. 37
6.7 Key frames of dynamic obstacle avoidance demonstration for humanoid robot. 38
6.8 Speed of the algorithm throughout dynamic obstacle avoidance. . . . 39
6.9 Trajectory of the CoM of the humanoid robot. 40

List of Tables

A.1 DH convention for KUKA robot in PDO-IK formulation. 44
A.2 DH convention for Franka robot in PDO-IK formulation. 45
A.3 DH convention for UR10 robot in PDO-IK formulation. 46
A.4 DH convention for H1 robot in PDO-IK formulation from number 1 to 25. 47
A.5 DH convention for H1 robot in PDO-IK formulation from number 26 to 50. 48
A.6 DH convention for H1 robot in PDO-IK formulation from number 50 to 70. 49

Chapter 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 work 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* [19]. 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 [10, 12, 17, 18, 19, 22, 23]. 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. [22] formulated the kinematics of 6-DoF serial robots

using a distance matrix [6] and solve the IK with matrix completion leveraging Cayley-Menger determinant [24]. Marić and Giamou et al captures the sparsity of the distance matrix [17] and use sparse bounded-degree sum of squares relaxations [29] to solve IK for spherical joint robots without considering collisions. Marić et al. proposed a distance-geometric framework called Riemannian Trust Region (R-TR) [19] to solve the constrained IK by optimizing the distance matrix using Riemannian optimization. Works of distance-based IK will be detailed in chapter 2.

Despite these advancements, distance-based methods still suffer from low computational efficiency. This work is based on our previous conference submission [4], which 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 [19] 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 numer-

ical 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.

Chapter 2

Related Works

In this chapter, we review previous work on IK solvers for articulated robots with revolute joints. First, we introduce various types of IK solvers based on the widely applied joint space formulation using angles. Then, we discuss recent developments in distance-based solvers.

2.1 Angle-based Formulations for Inverse Kinematics

The majority of IK solvers represent the joint space of articulated robots using joint angles $\boldsymbol{\theta}$. This approach is inspired by the fact that revolute joints are directly driven by motors, and their angles can be read from sensors.

The IK of a non-redundant robot can be solved using analytical techniques to derive explicit equations for joint variables in terms of the end-effector's desired position and orientation. This involves formulating the kinematic equations based on the robot's Denavit– Hartenberg (DH) parameters. By systematically applying trigonometric identities and algebraic manipulation, the kinematic equations can be reduced to a set of solvable polynomial equations. Analytic solutions are highly efficient and precise. They are particularly feasible for robots with simple kinematic chains and a small number of DoFs, where the mathematical complexity remains manageable.

On the contrary, robots with redundant DoFs cannot be solved analytically. One feasible approach for such cases is to search within the range of joint angle limits. However, this can be computationally expensive when the dimension of the system is considerably high. As a result, people tend to solve the IK for robots with redundant DoFs using numerical optimization solvers.

The pseudo-inverse method [30] is a widely used technique for IK, which calculates joint configurations to achieve a desired end-effector position and orientation. It is a first-order optimization solver that only uses the Jacobian for optimization. The method starts by linearizing the relationship between small changes in joint angles $\delta\boldsymbol{\theta}$ and small changes in the end-effector pose, mapping these changes with the Jacobian matrix. Instead of directly solving the inverse of the Jacobian, which might not always exist or be stable, the method computes its pseudo-inverse. Therefore, the pseudo-inverse method handles arbitrary DoFs by finding a solution that minimizes the error in a least-squares sense. By using the pseudo-inverse, the method mitigates problems caused by singularities and unstable solutions. However, given the complexity of the pseudo-inverse method, $O(mn^2)$ where m and n are the larger and smaller dimensions of the Jacobian, respectively, such IK solvers can be computationally intensive, especially for high-dimensional systems.

another option for first-order optimization IK solvers that use the transpose of the Jacobian instead of its inverse or pseudo-inverse. The Jacobian transpose method computes the change in joint angles $\Delta\boldsymbol{\theta}$ by multiplying a step size α, the transpose of the Jacobian, and the end-effector pose error. When both α and the end-effector error are sufficiently small, this method can rapidly converge. The choice of the step size α is critical; if it is too large, the method may oscillate, and if it is too small, it may converge slowly.

Although capable of handling various numbers of constraints and DoFs, the first-order optimization solvers mentioned above are often ill-behaved near singularities and are unable to converge to exact singular configurations. Therefore, second-order optimization methods, which compute the direction of each step in optimization iterations based on the Hessian or its approximations, are applied to IK solvers for faster convergence and more precise solutions.

Gauss-Newton methods are popular second-order approaches to solve IK. Deo et al. [8] model IK as a non-linear least squares problem and use the Gauss-Newton

method and Levenberg-Marquardt iteration to solve it. The Hessian matrix is only computed when the end-effector target is not reachable. When the target is reachable, the Hessian matrix is approximated with $J^\top J$, where J is the Jacobian. Compared to first-order models, this method is insensitive to the reachability of the desired position. It converges to a joint configuration that places the end-effector at the closest possible distance from the desired goal, irrespective of whether the goal is within the manipulator workspace or not. Thus, convergence is ensured even if the goal position corresponds to a singular configuration of the manipulator.

Second-order solvers can also be combined with first-order solvers. The damped least squares method [21, 27] is a blend of the Gauss-Newton algorithm and gradient descent. It uses a damping factor λ to control the step size and direction. If λ is large, the algorithm behaves like gradient descent. If λ is small, it behaves like Gauss-Newton. The damped least squares method requires careful tuning of the damping factor λ. Wang et al. [28] use a combined optimization approach that first finds a coarse solution with cyclic coordinate descent and then fine-tunes the solution with the Broyden-Fletcher-Shanno solver. This method is capable of handling joint limits, robots with arbitrary DoFs, and collision avoidance.

2.2 Distance-based Formulation for Inverse Kinematics

Distance-based optimization is an important class of literature for IK formulation. Instead of optimizing the joint angles, distance-based methods attach points to the robot and the obstacles and formulate the constrained IK as an optimization over the distances among these points. As a variant of the angle-based formulation, the motivation for the distance-based formulation is to avoid the complex mapping from joint space to task space caused by trigonometric constraints.

Josep et al. [23] proposed a distance-based strategy for IK aimed at applications where a system of constraints on the relative positions of a set of objects must be solved. This includes position analysis of serial and parallel robots, contact analysis of polyhedral models, or the automatic generation of constraint-specified designs or assemblies. This method involves searching for some of the a priori unknown

distances by leveraging the Cayley-Menger determinant [24]. They use a branch-and-prune technique to find these distances, iteratively eliminating regions from the space of distances that cannot contain any solution. This elimination is accomplished by applying redundant necessary conditions derived from the theory of distance geometry.

Han et al. [12] parameterized the IK problem of serial robots. They focus on IK for either a spatial chain with spherical joints or a planar chain with revolute joints. This method introduces a new set of geometric parameters, which are combinations of anchored diagonal lengths and triangle orientations rather than joint angles, and uses an exact formulation to express the IK as a system of linear inequalities. Under their formulation, the set of solutions for an IK problem specified by the positions of the two end points of the last link, and more generally the set of solutions for all IK problems, is essentially piecewise convex. However, their method ignores other constraints such as joint limits and link collision-free constraints.

Marić and Giamou et al. proposed a distance formulation of optimization-based IK as a nearest point problem with a fast sum of squares solver. They capture the sparsity of the distance matrix [17] and use sparse bounded-degree sum of squares relaxations [29] to solve IK. Their method focuses on robots with spherical joints without considering collisions. This approach has the advantages of post-hoc certification of global optimality and a runtime that scales polynomially with the number of DoF. They prove that such convex relaxation leads to a globally optimal solution when certain conditions, which are common and practical, are met.

Marić et al. also proposed a distance-geometric framework called R-TR [18, 19] to solve constrained IK by optimizing the distance matrix using Riemannian optimization. They formalize the equivalence of distance-based IK and the distance geometry problem for a large class of articulated robots and task constraints. Furthermore, they use the connection between distance geometry and low-rank matrix completion to find IK solutions by completing a partial Euclidean distance matrix through local optimization. They additionally parameterize the space of Euclidean distance matrices with the Riemannian manifold of fixed-rank Gram matrices, allowing them to leverage a variety of mature Riemannian optimization methods. They also use bound smoothing to generate informed initializations and improve convergence. R-TR achieves higher success rates than traditional angle-based techniques, especially on

problems that involve many workspace constraints.

Giamou et al. [10] formulate IK problems with complex workspace constraints as a convex feasibility problem whose low-rank feasible points provide exact solutions based on distance. Their method solves IK with a sequence of semidefinite programs whose objectives are designed to encourage low-rank minimizers. Their formulation expresses the intrinsic robot geometry and obstacle avoidance as simple linear matrix equations and inequalities.

Recent distance-based IK formulations demonstrate more effective performance in constrained IK computations compared to angle-based algorithms, particularly in managing joint limits and collision avoidance. However, these frameworks often fail to consider the structure of the kinematic chain. As a result, they still suffer from low computational efficiency.

Chapter 3

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 [1]. 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.

3.1 Denavit–Hartenberg Parameters

The DH convention is a standardized methodology in robot kinematics for modeling the relationship between adjacent links in a robotic arm. It involves assigning coordinate frames to each link and defining four parameters: link length, link twist,

[1] Tree-structure robots and parallel robots can also be handled by adding extra constraints. An example is tested in Sec. 6.3.

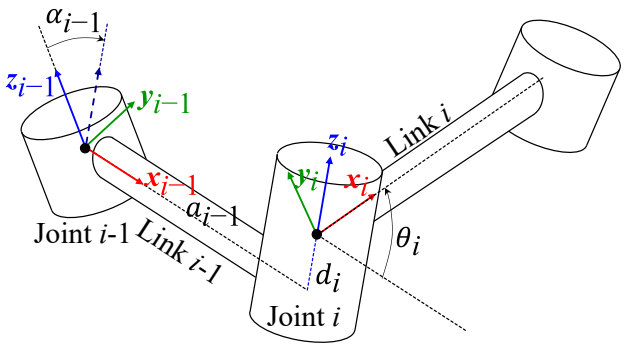

Figure 3.1: Link frame attachment on articulated robots.

link offset, and joint angle. These parameters systematically describe the spatial relationships between the links, facilitating the transformation of coordinates from one joint to another.

The kinematic chain formulation in this paper builds upon the proximal DH convention [5]. The proximal-DH convention is a variation of the DH convention that modifies the placement of the coordinate frames. In this convention, the frames are placed at the proximal end of each link rather than the distal end, which can lead to different parameter values and transformations. The choice between the DH and proximal-DH conventions depends on specific application requirements and desired computational efficiency in robotic system modeling and analysis.

As shown in Fig. 3.1, 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:

$$
{}^{i-1}\mathbf{T}_i = f(\theta_i; \alpha_{i-1}, a_{i-1}, d_i) = \begin{bmatrix} c_\theta & -s_\theta & 0 & a_{i-1} \\ s_\theta c_\alpha & c_\theta c_\alpha & -s_\alpha & d_i s_\alpha \\ s_\theta s_\alpha & c_\theta s_\alpha & c_\alpha & d_i c_\alpha \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}
$$

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 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 $\mathbf{z}_i$.

3.2 Quasi-Newton Method

The quasi-Newton method [7] is a powerful iterative optimization technique used to find the minimum of a function $f(x)$ by finding the root of its Jacobian ∇f. Unlike traditional Newton's method, which requires the computation of second-order derivatives, quasi-Newton methods approximate the Hessian matrix using updates based on first-order derivatives. This approach enhances efficiency by avoiding the computational cost of Hessian evaluations in each iteration. Instead, it refines an approximation of the inverse Hessian matrix in each iteration k given $f(x_k)$ and $\nabla f(x_k)$, adjusting the search direction to progressively hone in on the function's minimum. Widely employed in numerical optimization, quasi-Newton methods like Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) [16] and Broyden's method [2] strike a balance between computational feasibility and convergence speed, making them invaluable in solving complex optimization problems.

3.3 Augmented Lagrangian

The Augmented Lagrangian method is a strategy used for solving constrained optimization problems, particularly those with inequality constraints. It combines elements of both penalty and Lagrangian methods to handle constraints effectively. The method introduces an augmented Lagrangian function that penalizes violations of constraints, incorporating a penalty parameter that increases as violations persist. This augmented Lagrangian formulation transforms the constrained problem into a sequence of unconstrained problems. The augmented Lagrangian $L_\rho(\boldsymbol{x})$ is defined as:

$$L_\rho = f(\boldsymbol{x}) + \lambda^\top g(\boldsymbol{x}) + \frac{\rho}{2} g(\boldsymbol{x})^\top g(\boldsymbol{x}) \tag{3.2}$$

where $\boldsymbol{x}$ represents the optimization variables, λ denotes the Lagrange multipliers associated with the constraints $g(\boldsymbol{x})$. $f(\boldsymbol{x})$ is the objective function to be minimized subject to constraints $g(\boldsymbol{x}) \leq 0$, and ρ is a penalty parameter controlling the strength of the penalty function. The method iteratively updates $\boldsymbol{x}$, λ, and ρ to drive $g(\boldsymbol{x})$ towards zero and thereby satisfy the constraints while minimizing $f(\boldsymbol{x})$. Such update strategy effectively balances the trade-off between constraint satisfaction and the

minimization of the objective function, offering robustness and convergence guarantees for a wide range of optimization problems.

Chapter 4

Kinematics Framework

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. 4.1), joint limit constraints (Sec. 4.2), collision avoidance constraints (Sec. 4.3), and end effector pose objective (Sec. 4.4).

4.1 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. 4.1, 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:

$$\mathbf{u}_i = \mathbf{o}_i \tag{4.1a}$$

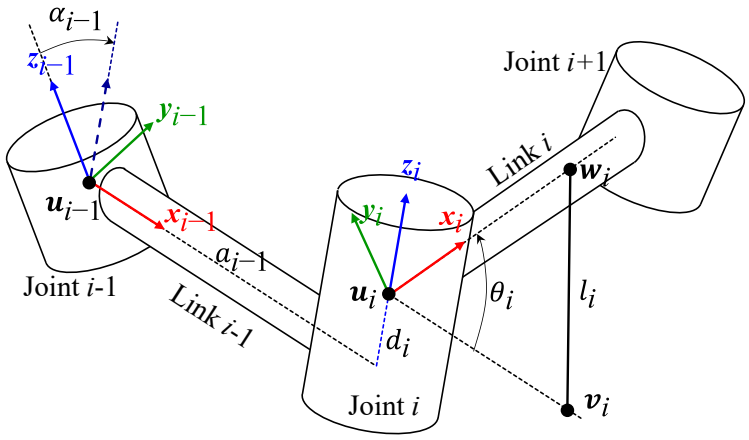

Figure 4.1: Distance-based kinematic chain formulation.

$$\mathbf{v}_i - \mathbf{u}_i = l_i^{uv}\mathbf{x}_{i-1} \tag{4.1b}$$

$$\mathbf{w}_i - \mathbf{u}_i = l_i^{wv}\mathbf{x}_i \tag{4.1c}$$

$$2l_i^{uv}l_i^{uw}\cos\theta_i = l_i^{uv2} + l_i^{uw2} - l_i^{vw2} \tag{4.1d}$$

We define the squared distance $L_i = l_i^{vw2}$ so that $\cos\theta_i \propto L_i$. For simplicity, we let $l_i^{uv} = l_i^{uw} = 1/\sqrt{2}$ such that

$$\cos\theta_i = 1 - L_i \tag{4.2a}$$

$$\sin\theta_i = \left(1 - \cos\theta_i{}^2\right)^{\frac{1}{2}} = \left(2L_i - L_i^2\right)^{\frac{1}{2}} \tag{4.2b}$$

Notably, Eq. 4.2b assumes that θ_i lies in the range $[0, \pi]$ [17], 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. 4.2.

After attaching points on $\mathcal{F}_i$, we re-parameterize ${}^{i-1}\mathbf{T}_i$ by substituting Eq. 4.2a and Eq. 4.2b to Eq. 3.1:

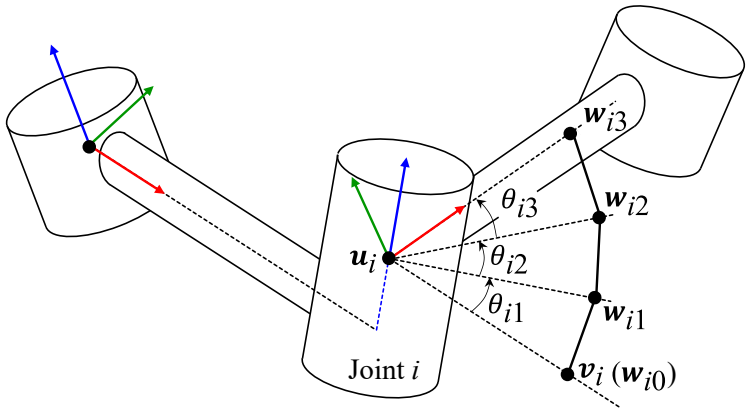

Figure 4.2: Example of angle decomposition for joint limit formulation.

$$
{}^{i-1}\mathbf{T}_i = g(L_i; \alpha_{i-1}, a_{i-1}, d_i) = \begin{bmatrix} 1 - L_i & -\left(2L_i - L_i^2\right)^{\frac{1}{2}} & 0 & a_{i-1} \\ c_\alpha\left(2L_i - L_i^2\right)^{\frac{1}{2}} & c_\alpha(1 - L_i) & -s_\alpha & d_i s_\alpha \\ s_\alpha\left(2L_i - L_i^2\right)^{\frac{1}{2}} & s_\alpha(1 - L_i) & c_\alpha & d_i c_\alpha \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}
$$

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 $\mathbf{T}_i$:

$$
\mathbf{T}_i = \prod_{p=1}^{i} {}^{p-1}\mathbf{T}_p \tag{4.4a}
$$

$$
\mathbf{u}_i = \left[\mathbf{T}_i^{(1,4)},\ \mathbf{T}_i^{(2,4)},\ \mathbf{T}_i^{(3,4)}\right]^\top \tag{4.4b}
$$

4.2 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 [11].

The formulation of L_i in Eq. 4.2b 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 $[0, \pi]$:

$$\theta_i = \textstyle\sum_{m=1}^{\lceil k \rceil} \theta_{im}, \qquad \theta_{im} \in \left[\theta_i^{\min}/\lceil k \rceil, k\pi/\lceil k \rceil\right] \subseteq [0, \pi] \tag{4.5}$$

Fig. 4.2 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

$$\cos\theta_{im} = 1 - L_{im} \tag{4.6}$$

By substituting Eq. 4.5 into Eq. 4.6, the range of L_{im} is:

$$1 - \cos\left(\theta_i^{\min}/\lceil k \rceil\right) \leq L_{im} \leq 1 - \cos\left(k\pi/\lceil k \rceil\right) \tag{4.7}$$

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

$${}^{i-1}\mathbf{T}_i = \prod_{m=1}^{\lceil k \rceil} g(L_{im}) \tag{4.8}$$

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 [14], penalizing over constraint violation [15, 25], or converting to equality constraints [1, 20]. In this work, we convert the box constraint to equality constraints using a squashing function $s(\omega)$ with the following properties [20]:

$$s(\omega) : \mathbb{R} \to (s^{\min}, s^{\max}) \tag{4.9a}$$

$$\frac{\mathrm{d}}{\mathrm{d}\omega} s(\omega) \geq 0 \tag{4.9b}$$

$$s^{\min} = \lim_{\omega \to -\infty} s(\omega), \; s^{\max} = \lim_{\omega \to \infty} s(\omega) \tag{4.9c}$$

where $\omega \in \mathbb{R}$ is the additional decision variable. We build our squashing function for L_i upon the sigmoid function [11]

$$\sigma(\omega_i) = \frac{1}{1 + e^{-\omega_i}} : \mathbb{R} \to (0, 1) \tag{4.10}$$

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

$$L_i = s(\omega_i) = \left(L_i^{\max} - L_i^{\min}\right) \sigma(\omega_i) + L_i^{\min} \tag{4.11a}$$

$$s(\omega_i) : \mathbb{R} \to (L_i^{\min}, L_i^{\max}) \tag{4.11b}$$

Eq. 4.11a and Eq. 4.11b bounds 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. 5, 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.

4.3 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 [19]. 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.

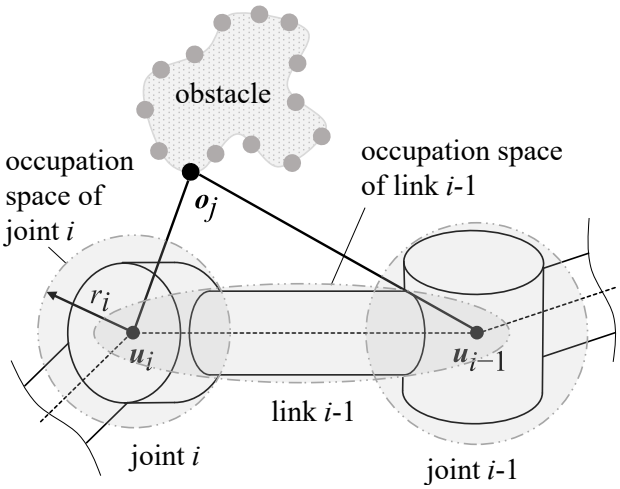

Figure 4.3: Collision avoidance constraints formulation visualization.

We propose a novel formulation of collision avoidance that only requires the points attached to joints to achieve full-body collision avoidance (Fig. 4.3). 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:

$$c_{ij}^{\text{joint}} = r_i - ||\mathbf{u}_i - \mathbf{o}_j|| \leq 0 \tag{4.12}$$

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:

$$c_{ij}^{\text{link}} = 2a_i - (||\mathbf{u}_i - \mathbf{o}_j|| + ||\mathbf{u}_{i+1} - \mathbf{o}_j||) \leq 0 \tag{4.13}$$

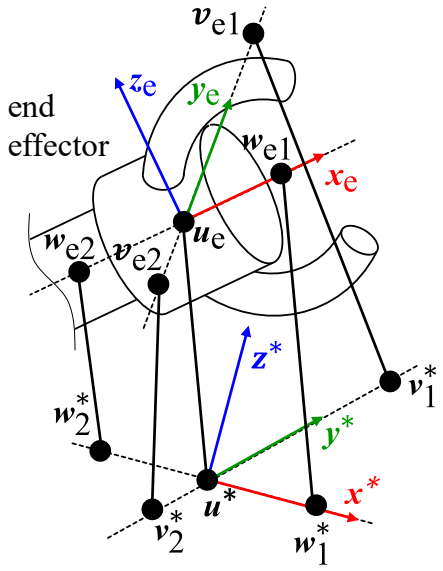

Figure 4.4: Example of end effector objective formulation.

Eq. 4.13 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:

$$c(\boldsymbol{\omega}) \leq 0 \tag{4.14}$$

4.4 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 $\mathbf{T}_\text{e}$:

$$\mathbf{w}_{ep} = k_p\mathbf{x}_\text{e} + \mathbf{u}_\text{e}, \;\; \mathbf{v}_{ep} = q_p\mathbf{y}_\text{e} + \mathbf{u}_\text{e} \tag{4.15}$$

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. 4.4 shows an example when $n_1 = n_2 = 2$.

We arrange the points attached on $\mathcal{F}_\mathrm{e}$ into matrix $\mathbf{U}_\mathrm{e} = [\mathbf{u}_\mathrm{e}, \mathbf{w}_{\mathrm{e}1}, ..., \mathbf{w}_{\mathrm{e}n1}, \mathbf{v}_{\mathrm{e}1}, ..., \mathbf{v}_{\mathrm{e}n2}]^\top \in \mathbb{R}^{(n_1+n_2+1)\times 3}$ 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:

$$J(\boldsymbol{\omega}) = \frac{1}{2} \sum \left((\mathbf{U}_\mathrm{e} - \mathbf{U}^*)^\top (\mathbf{U}_\mathrm{e} - \mathbf{U}^*) \right) \tag{4.16}$$

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 $n_1 = n_2 = 0$.

Chapter 5

Kinematics 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. 4.2:

$$\begin{aligned} \min_{\boldsymbol{\omega}} \quad & J(\boldsymbol{\omega}) = \frac{1}{2}\sum (\mathbf{U}_{\mathrm{e}} - \mathbf{U}^*)^\top (\mathbf{U}_{\mathrm{e}} - \mathbf{U}^*) \\ \text{s.t.} \quad & \mathbf{U}_{\mathrm{e}} = FK(\boldsymbol{\omega}) \\ & c(\boldsymbol{\omega}) \leq 0 \end{aligned} \tag{5.1}$$

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:

$$L_\rho(\boldsymbol{\omega}) = J(\boldsymbol{\omega}) + \boldsymbol{\mu}^\top c'(\boldsymbol{\omega}) + \frac{\rho}{2} c'(\boldsymbol{\omega})^\top c'(\boldsymbol{\omega}) \tag{5.2}$$

where λ is the Lagrangian multiplier and ρ is the adjust penalty parameter. The

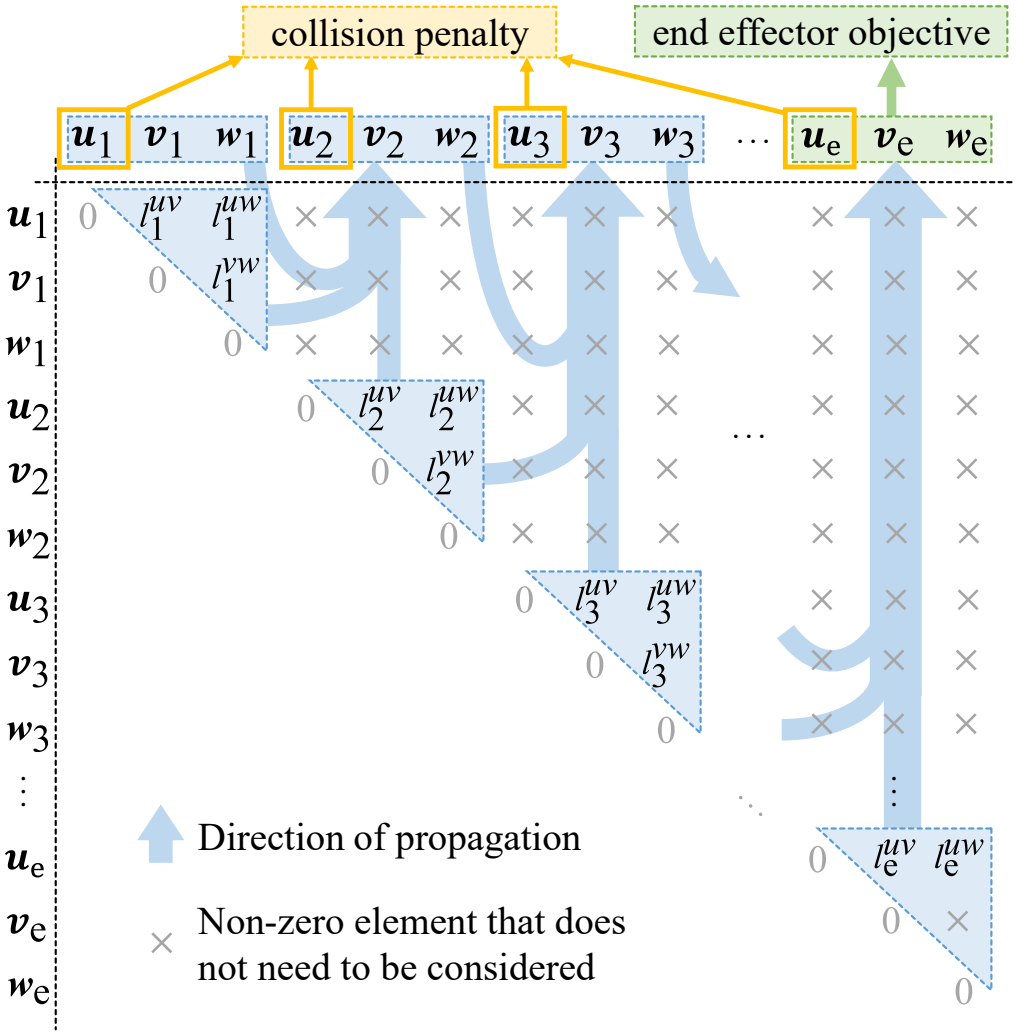

Figure 5.1: Visualization of propagative computation in forward kinematics.

constrained IK problem is formulated as finding $\boldsymbol{\omega}^* \in \mathbb{R}^{M'}$ such that

$$\boldsymbol{\omega}^* = \text{argmin} L_\rho(\boldsymbol{\omega}) \tag{5.3}$$

5.1 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 $\boldsymbol{\omega}$.

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. 4.11a
  ${}^{i-1}\mathbf{T}_i \leftarrow g(L_i)$ ▷ Eq. 4.3
  $\mathbf{T}_i \leftarrow \mathbf{T}_{i-1} \cdot {}^{i-1}\mathbf{T}_i$ ▷ Eq. 4.4a
  Compute $\mathbf{u}_i$ from $\mathbf{T}_i$ ▷ Eq. 4.4b
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. 4.15
Compute J from $\mathbf{U}_{\text{e}}$ and $\mathbf{U}^*$. ▷ Eq. 4.16
$L_\rho \leftarrow J$
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. 4.12
    $c_{ij}^{\text{link}} \leftarrow 2a_i - (||\mathbf{u}_i - \mathbf{o}_j|| + ||\mathbf{u}_{i+1} - \mathbf{o}_j||)$ ▷ Eq. 4.13
    $L_\rho \leftarrow L_\rho + \mu_{ij} c_{ij}^{\text{joint}} + \mu_{ij} c_{ij}^{\text{link}} + \frac{\rho}{2} c_{ij}^{\text{joint}^2} + \frac{\rho}{2} c_{ij}^{\text{link}^2}$ ▷ Eq. 5.2
  end for
end for
return L_ρ

As shown in Fig. 5.1 and Algorithm 1, the forward rollout is computed propagatively along the direction of the kinematic chain from the base to the end effector. Fig. 5.1 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.

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}.

5.2 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 $\nabla_{\boldsymbol{\omega}} c(\boldsymbol{\omega})$:

$$\nabla_{\boldsymbol{\omega}} L_\rho = \nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) + \boldsymbol{\mu} \nabla_{\boldsymbol{\omega}} c(\boldsymbol{\omega}) + \rho c(\boldsymbol{\omega}) \odot \nabla_{\boldsymbol{\omega}} c(\boldsymbol{\omega}) \tag{5.4a}$$

$$\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) = \frac{\mathrm{d}J}{\mathrm{d}\mathbf{T}_\mathrm{e}} \frac{\mathrm{d}\mathbf{T}_\mathrm{e}}{\mathrm{d}\boldsymbol{\omega}} \tag{5.4b}$$

$$\nabla_{\boldsymbol{\omega}} c(\boldsymbol{\omega}) = \sum_{i=1}^{M} \frac{\partial c}{\partial \mathbf{u}_i} \frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}\boldsymbol{\omega}} \tag{5.4c}$$

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 ($\partial J / \partial \mathbf{T}\mathrm{e}$), 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. 5.4c is computed with

$$\frac{\partial c}{\partial \mathbf{u}_i} = \sum_{j=1}^{N} \left(\frac{d}{d\mathbf{u}_i} c_{ij}^{\text{joint}} + \frac{\partial}{\partial \mathbf{u}_i} c_{ij}^{\text{link}} + \frac{\partial}{\partial \mathbf{u}_i} c_{(i+1)j}^{\text{link}} \right) \tag{5.5a}$$

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{u}_i} c_{ij}^{\text{joint}} = (\mu_{ij} + \rho c_{ij}^{\text{joint}}) \frac{\mathbf{u}_i - \mathbf{o}_j}{||\mathbf{u}_i - \mathbf{o}_j||} \cdot \mathbb{1}\left(c_{ij}^{\text{joint}} \geq 0 \right) \tag{5.5b}$$

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{u}_i} c_{ij}^{\text{link}} = (\mu_{ij} + \rho c_{ij}^{\text{link}}) \frac{\mathbf{u}_i - \mathbf{o}_j}{||\mathbf{u}_i - \mathbf{o}_j||} \cdot \mathbb{1}\left(c_{ij}^{\text{link}} \geq 0 \right) \tag{5.5c}$$

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. 4.12. The term $\mathbb{1}(\cdot)$ is the indicator function, which equals to 1 when $(\cdot)$ is true otherwise 0. The term $\partial J / \partial \mathbf{T}_\mathrm{e}$

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. 5.5b
      Add $[\mathbf{s}_1^\top, 0]^\top$ to the last column of $\partial L_\rho / \partial \mathbf{T}_i$. ▷ Eq. 5.5a
    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. 5.5c
      $\mathbf{s}_3 \leftarrow (\mu_{(ij} + \rho c_{(i-1)j}^{\text{link}})(\mathbf{u}_{i-1} - \mathbf{o}_j)/||\mathbf{u}_{i-1} - \mathbf{o}_j||$ ▷ Eq. 5.5c
      Add $[\mathbf{s}_3^\top, 0]^\top$ to the last column of $\partial L_\rho / \partial \mathbf{T}_{i-1}$. ▷ Eq. 5.5a
    end if
  end for
end for
$\partial L_\rho / \partial \mathbf{T}_M \leftarrow \partial J / \partial \mathbf{T}_\text{e}$ ▷ Eq. 5.6
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. 5.7b
  $\partial^{i-1}\mathbf{T}_i / \partial L_i \leftarrow g'(L_i)$ ▷ Eq. 5.7c
  $\partial L_i / \partial \omega_i \leftarrow (L_i^{\max} - L_i^{\min})\sigma(\omega_i)(1 - \sigma(\omega_i))$ ▷ Eq. 5.7d
  $\partial L_\rho / \partial \omega_i \leftarrow (\sum (\partial L / \partial^{i-1}\mathbf{T}_i) \cdot (\partial^{i-1}\mathbf{T}_i / \partial L_i))\,(\partial L_i / \partial \omega_i)$ ▷ Eq. 5.7a
  $\partial L_\rho / \partial \mathbf{T}_{i-1} \leftarrow \partial L_\rho / \partial \mathbf{T}_{i-1} + (\partial L_\rho / \partial \mathbf{T}_i) \cdot^{i-1} \mathbf{T}_i^\top$ ▷ Eq. 5.8
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 $\nabla_{\boldsymbol{\omega}} L_\rho$

in Eq. 5.4b is computed with:

$$\frac{\partial}{\partial \mathbf{T}_\text{e}} J = \begin{bmatrix} \sum_{p=1}^{n_1} k_p(\mathbf{w}_{ep} - \mathbf{w}^*) & \sum_{p=1}^{n_2} q_p(\mathbf{v}_{ep} - \mathbf{v}^*) & 0 & \mathbf{u}_\text{e} - \mathbf{u}^* \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.6}$$

where all the elements are already computed in Eq. 4.15 and Eq. 4.16. 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:

$$\frac{\partial}{\partial \omega_i} L_\rho = \left(\sum \frac{\partial L_\rho}{\partial^{i-1}\mathbf{T}_i} \cdot \frac{\partial^{i-1}\mathbf{T}_i}{\partial L_i} \right) \frac{\partial L_i}{\partial \omega_i} \tag{5.7a}$$

$$\frac{\partial L_\rho}{\partial {}^{i-1}\mathbf{T}_i} = \mathbf{T}_{i-1}^\top \cdot \frac{\partial L_\rho}{\partial \mathbf{T}_i} \tag{5.7b}$$

$$\frac{\partial {}^{i-1}\mathbf{T}_i}{\partial L_i} = \frac{\mathrm{d}g(L_i)}{L_i} = \begin{bmatrix} -1 & -(1-L_i)\left(2L_i - L_i^2\right)^{-\frac{1}{2}} & 0 & 0 \\ c_\alpha\left(1-L_i\right)\left(2L_i - L_i^2\right)^{-\frac{1}{2}} & -c_\alpha & 0 & 0 \\ s_\alpha\left(1-L_i\right)\left(2L_i - L_i^2\right)^{-\frac{1}{2}} & -s_\alpha & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.7c}$$

$$\frac{\partial L_i}{\partial \omega_i} = (L_i^{\max} - L_i^{\min})\frac{\mathrm{d}}{\mathrm{d}\omega_i}\sigma(\omega_i) = (L_i^{\max} - L_i^{\min})\sigma(\omega_i)(1-\sigma(\omega_i)) \tag{5.7d}$$

where $\mathbf{T}_{i-1}$ is already computed in Eq. 4.4a, $\sigma(\omega_i)$ is already computed in Eq. 4.10, and $\partial L_\rho / \partial \mathbf{T}_i$ can be recursively computed from joint $i+1$, collision penalty, and end effector pose objectives:

$$\frac{\partial L_\rho}{\partial \mathbf{T}_i} = \begin{cases} \frac{\partial c}{\partial \mathbf{T}_i} + \frac{\partial L_\rho}{\partial \mathbf{T}_{i+1}} \cdot {}^{i}\mathbf{T}_{i+1}^\top & , \ i < M \\ \frac{\partial c}{\partial \mathbf{T}_i} + \frac{\partial J}{\partial \mathbf{T}_\mathrm{e}} & , \ i = M \end{cases} \tag{5.8}$$

5.3 Inverse Kinematics

We first solve Eq. 5.2 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 $\mathrm{argmin} L_\rho$. 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 $\boldsymbol{\omega}_k^* = \mathrm{argmin} L_\rho$, we update Lagrangian multipliers $\boldsymbol{\mu}$ and ρ. Our method checks $\max(c(\boldsymbol{\omega}_k^*))$ in every iteration and will terminate if $\max(c(\boldsymbol{\omega}_k^*)) < c_\mathrm{tol}$ **or** $\max(c(\boldsymbol{\omega}_k^*)) \geq \beta c_\mathrm{last}$, where $\beta < 1$ and $c_\mathrm{last} = \max(c(\boldsymbol{\omega}_{k-1}^*))$.

By obtaining $\boldsymbol{\omega}^*$, we solve $\boldsymbol{\theta}^*$ from $\boldsymbol{\omega}^*$. We first compute L_i^* using Eq. 4.11a, and then compute θ_i^* from L_i^*:

$$\theta_i^* = \sum_{m=1}^{\lceil k \rceil} \arccos\left(1 - L_{im}^*\right) + \min\left(0, \theta_i^{\min}\right) \tag{5.9}$$

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. 4.11a and Eq. 5.9
return $\boldsymbol{\theta}^*$

5.4 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 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 $O(N)$, which is linear to the number of cluttered points in the environment.

On the contrary, the complexity of previous distance-based methods is quadratic or cubic relative to the number of environment points. For instance, a previous distance-based method, R-TR, introduced by Marić [19], has a complexity of $O(N^3)$. There are two reasons for the higher complexity of R-TR compared to our method. First, R-TR forms a dense graph of points attached to the robot, resulting in a

much denser distance matrix compared to ours. Second, R-TR requires a bound smoothing process [13] with a complexity of $O(N^3)$ to improve performance at the cost of speed. The bound smoothing technique can find all-pairs shortest paths as informed initializations that improve the convergence of the algorithm. R-TR utilizes the Floyd–Warshall algorithm [9], which is of $O(N^3)$ complexity, to solve the all-pairs shortest path problem. The runtime of bound smoothing significantly increases when the number of environment points N is considerable.

Chapter 6

Experiments

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

6.1 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 [19], 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 [19], 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) [31] 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 $\beta = 0.99$.

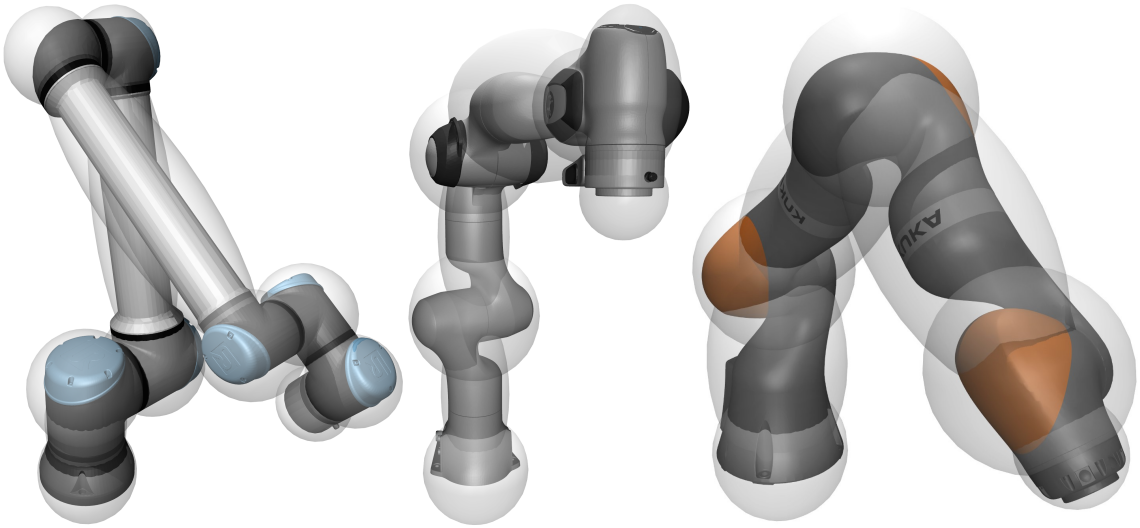

Figure 6.1: Robot arm platforms (UR10, Franka, and KUKA) and their occupation space visualized as translucent hulls.

We test all the methods on 3 popular commercial robot arms: UR10 [1], KUKA-IIWA [2], and Franka [3]. UR10 has 6 DoFs and every joint can rotate from $-360°$ to $360°$. KUKA-IIWA and Franka have 7 DoFs and have tighter joint limits. As mentioned in Sec. 4.3, 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.6.1. Their DH parameters used in the experiments are shown in Appendix A.1$\sim$ A.3.

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.6.2. 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

[1]UR10: https://www.universal-robots.com/.
[2]KUKA-IIWA: https://www.kuka.com/.
[3]Franka: https://franka.de/.

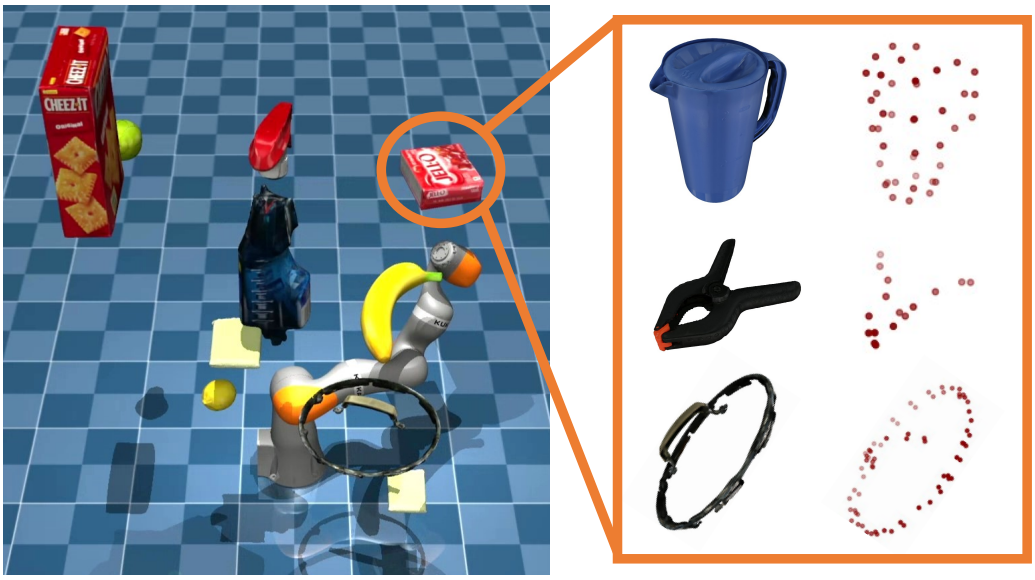

Figure 6.2: An example of the scene generated.

$\boldsymbol{\theta}_{\text{rand}}$ 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 [4] to check for collision. The point cloud of the obstacle is downsampled with the Voxel Grid filter from the Point Cloud Library (PCL) [5] with voxel grid leaf size equals to 0.1m. Examples of obstacles and their point cloud are shown in Fig. 6.2. The average amount of points in scenarios of 1 to 9 obstacles are 24, 47, 68, 87, 101, 110, 122, 130, and 136, respectively.
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.

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

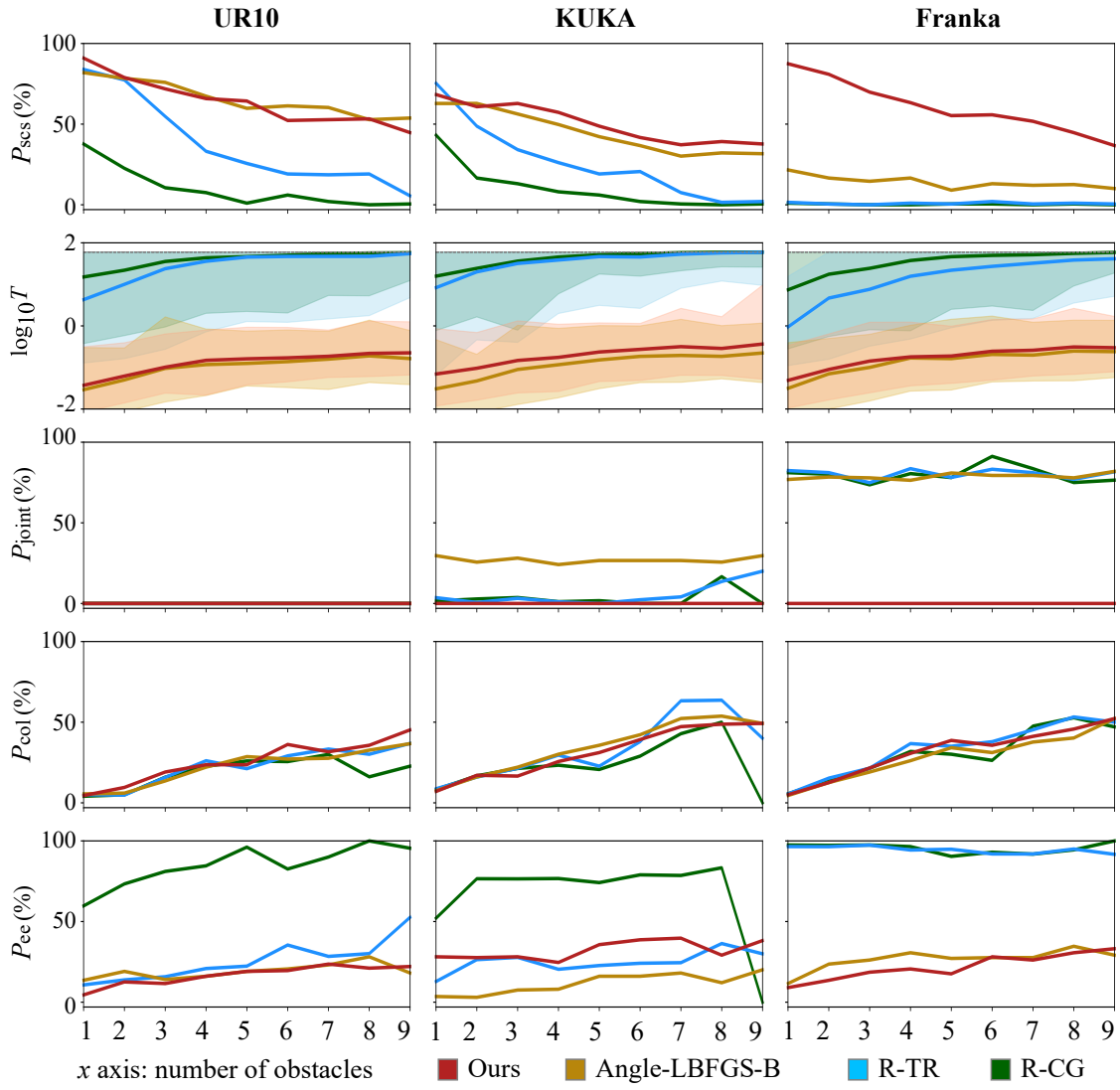

Figure 6.3: Experimental results.

We report the success rate P_{scs}, which is the 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

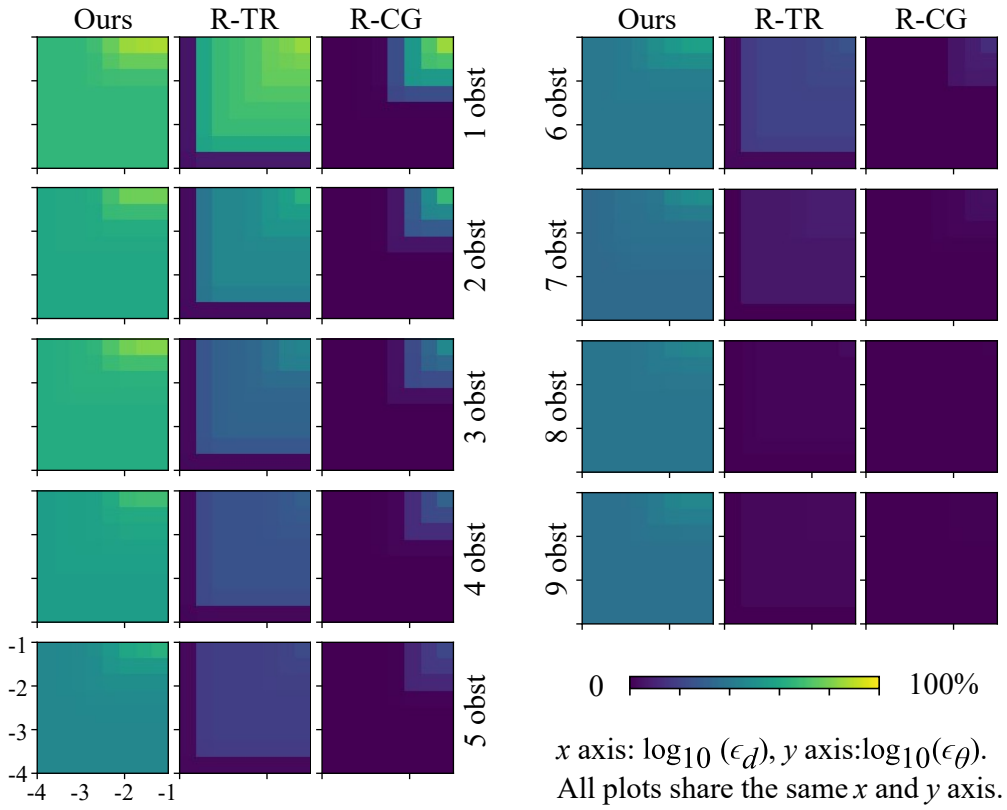

Figure 6.4: Convergence precision experiments results on KUKA robot.

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.6.3. 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

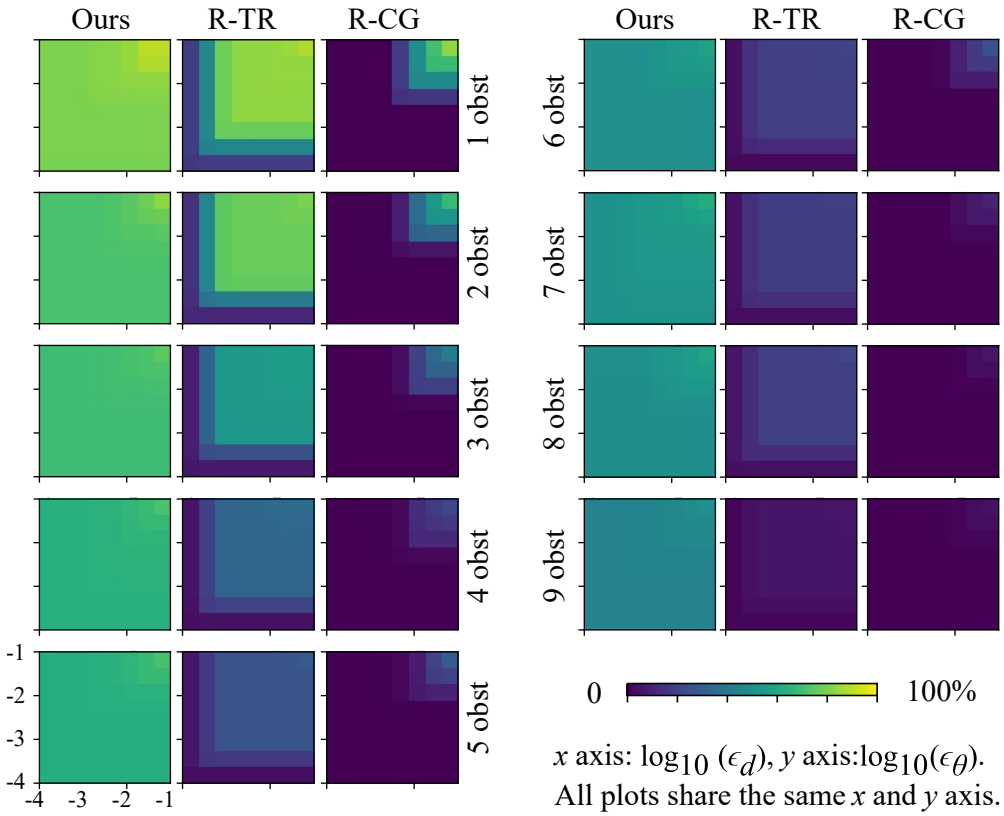

Figure 6.5: Convergence precision experiments results on UR10 robot.

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.

6.2 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. 6.4, Fig. 6.5, and Fig. 6.6 illustrate the success rate under different tolerance

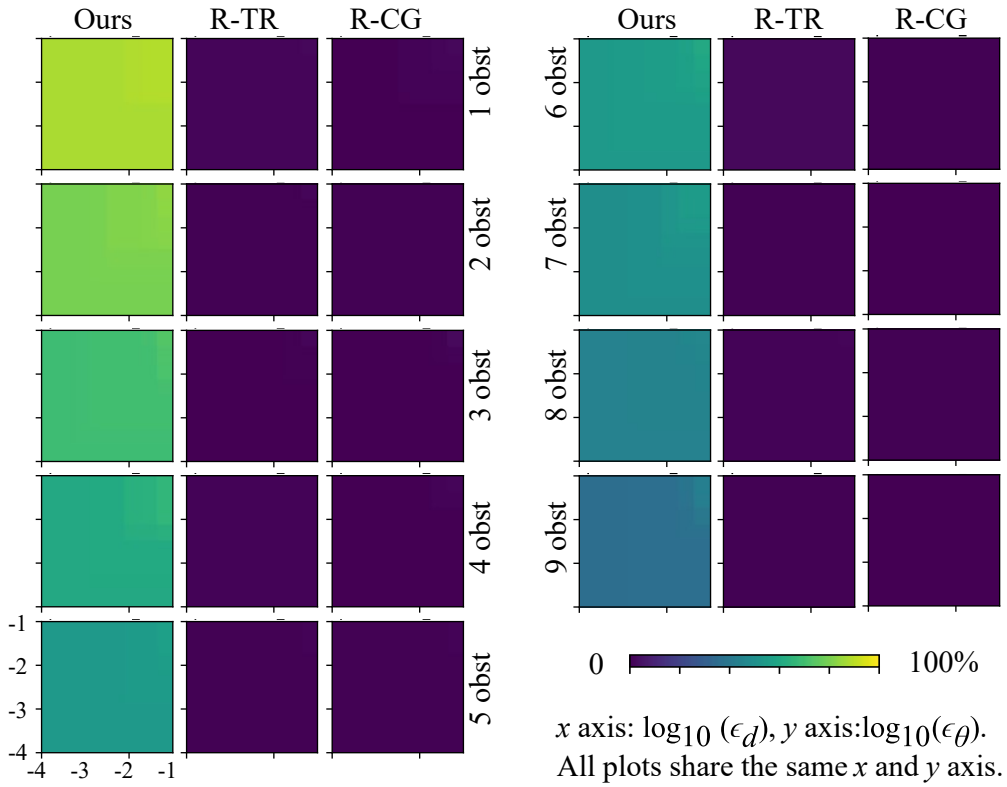

Figure 6.6: Convergence precision experiments results on Franka robot.

levels of ϵ_d and ϵ_θ on KUKA, UR10, and Franka when the number of obstacles is from 1 to 9. 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.

6.3 Humanoid Robot Avoiding Dynamic Obstacle

We further apply our algorithm on humanoid robot. 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 [6] 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

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

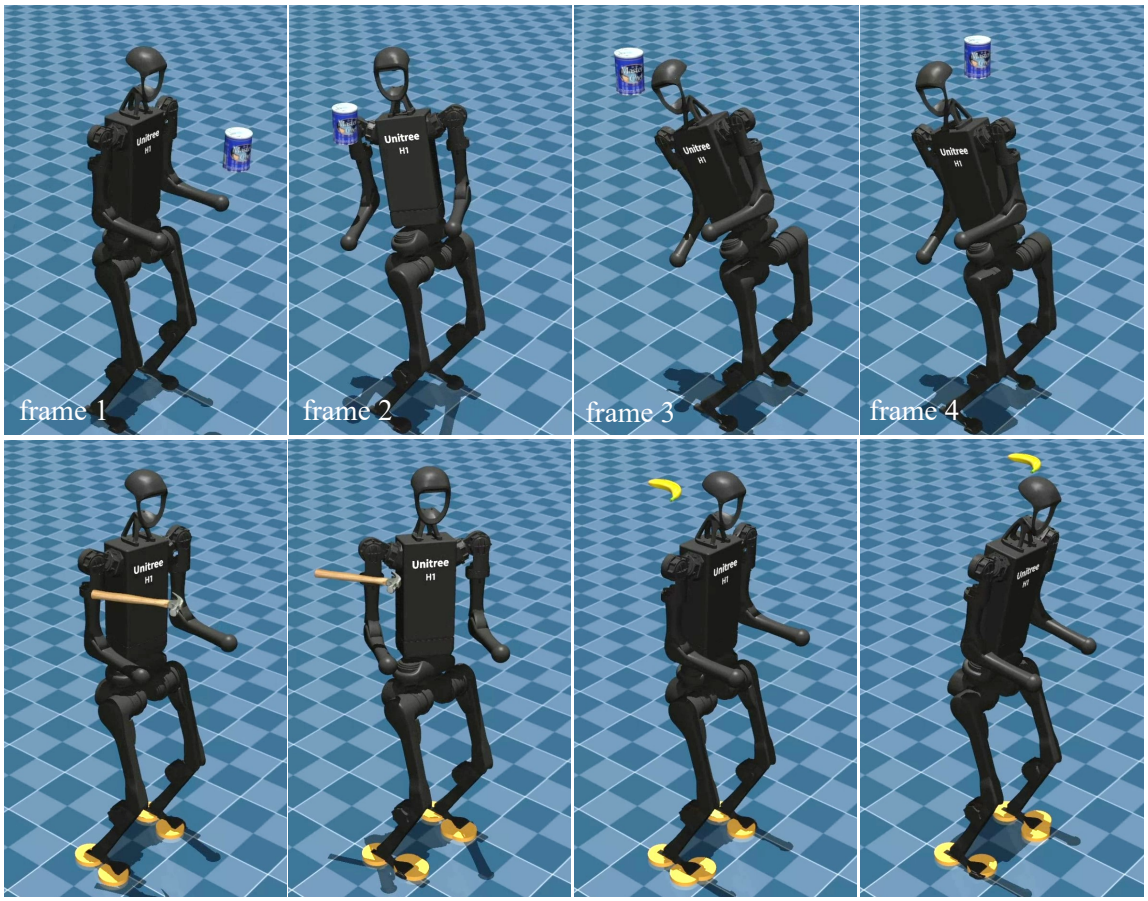

Figure 6.7: Key frames of dynamic obstacle avoidance demonstration for humanoid robot.

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 $\mathbf{u}^* = [-0.2, 0, 0]^\top$. The DH parameters used in the experiments are shown in Appendix A.4.

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

$$[-0.16, -0.07, 0.8]^\top < \mathbf{c} < [0.16, 0.075, 0.94]^\top \tag{6.1}$$

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

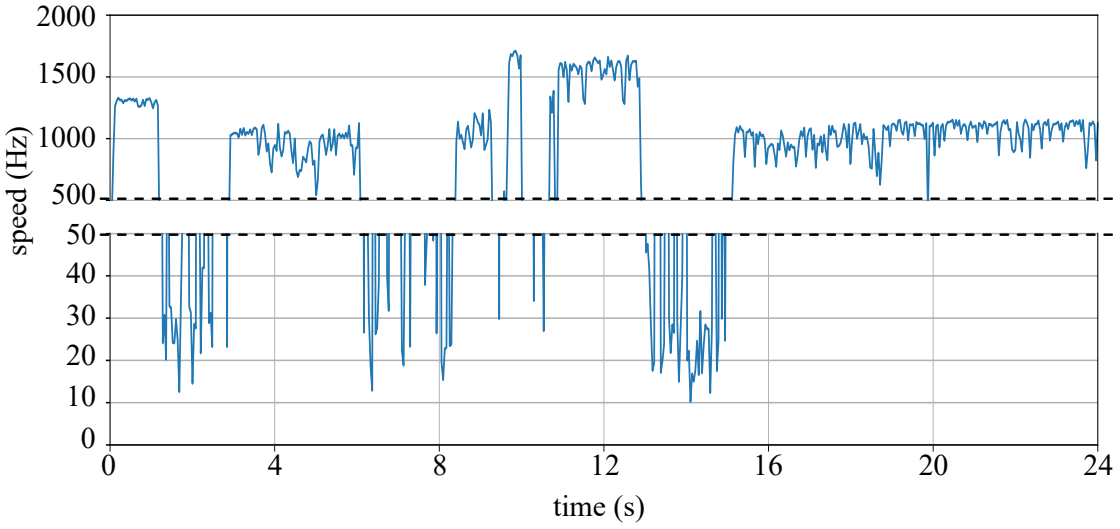

Figure 6.8: Speed of the algorithm throughout dynamic obstacle avoidance.

$$\mathbf{c} = \sum_{i=1}^{M} m_i \mathbf{c}_i = \sum_{i=1}^{M} m_i [\mathbf{T}_{\text{mass},i}(1,4),\ \mathbf{T}_{\text{mass},i}(2,4),\ \mathbf{T}_{\text{mass},i}(3,4)]^\top \tag{6.2a}$$

$$\mathbf{T}_{\text{mass},i} = \mathbf{T}_i \cdot^i \mathbf{T}_{\text{mass},i} \tag{6.2b}$$

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. The upper row of Fig. 6.7 shows a series of key frames of the simulation experiment on Mujoco [26]. The lower row shows some additional demonstrations of the humanoid avoiding dynamic obstacles (`048_hammer` and `011_banana` from the YCB dataset). The speed of the algorithm varies from 10 to 1500 Hz (Fig. 6.8), indicating that the algorithm is capable for real-time collision avoidance in dynamic environments.

The trajectory of the CoM is shown in Fig. 6.9. The blue box is the feasible region defined by Eq. 6.1. The CoM occasionally violates the CoM constraints of no more

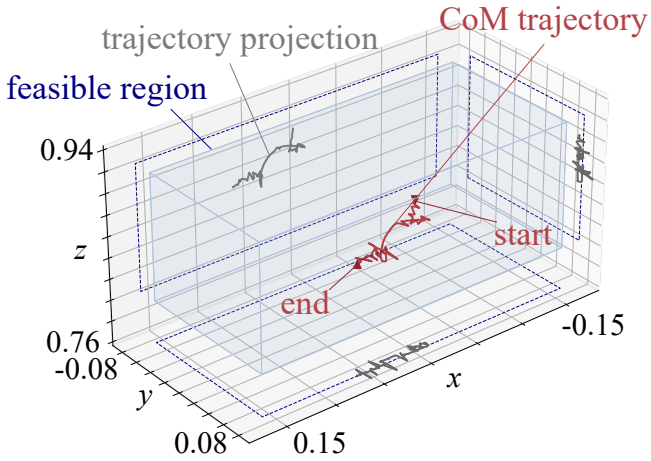

Figure 6.9: Trajectory of the CoM of the humanoid robot.

than 0.01m but will quickly return to the feasible region thereafter.

Chapter 7

Conclusions

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, this thesis does not provide a theoretical guarantee or analysis of why distance-based methods outperform angle-based methods, which remains a valuable open problem. Second, 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. Furthermore, tests on different types of robots, including robots with floating bases, can be conducted to further broaden the application of the algorithm. Finally, the constrained IK problem focuses on finding a feasible configuration of the robot for a given task, but

it cannot provide a path to the computed configuration. An exciting direction for future research is to utilize the distance-based formulation of PDO-IK in motion planning tasks.

Appendix A

DH Parameters in Experiments

This chapter includes the DH conventions we used in the experiments for PDO-IKacross robot platforms of KUKA (Tab. A.1), Franka (Tab. A.2), UR10 (Tab. A.3), and H1 (Tab. A.4 to A.6). In this chapter, we use L_i, a_{i-1}, α_{i-1}, and d_i to denote the DH parameters described in Sec. 3.1 of the ith frame.

A.1 KUKA

i	L_i	a_{i-1}	α_{i-1}	d_i
1	$0.0038 \sim 1.9962$	0	0	0.36
2	$0.0038 \sim 1.9962$	0	0	0
3	$2 \sim 2$	0	0	0
4	$0.1340 \sim 1.8660$	0	$-\frac{\pi}{2}$	0
5	$0.1340 \sim 1.8660$	0	0	0
6	$2 \sim 2$	0	0	0
7	$0.0038 \sim 1.9962$	0	$\frac{\pi}{2}$	0.42
8	$0.0038 \sim 1.9962$	0	0	0
9	$2 \sim 2$	0	0	0
10	$0.1340 \sim 1.8660$	0	$\frac{\pi}{2}$	0
11	$0.1340 \sim 1.8660$	0	0	0
12	$2 \sim 2$	0	0	0
13	$0.0038 \sim 1.9962$	0	$-\frac{\pi}{2}$	0.4
14	$0.0038 \sim 1.9962$	0	0	0
15	$2 \sim 2$	0	0	0
16	$0.1340 \sim 1.8660$	0	$-\frac{\pi}{2}$	0
17	$0.1340 \sim 1.8660$	0	0	0
18	$2 \sim 2$	0	0	0
19	$0.0010 \sim 1.9990$	0	$\frac{\pi}{2}$	0
20	$0.0010 \sim 1.9990$	0	0	0
21	$2 \sim 2$	0	0	0
22	$0 \sim 0$	0	0	0.126

Table A.1: DH convention for KUKA robot in PDO-IK formulation.

A.2 Franka

i	L_i	a_{i-1}	α_{i-1}	d_i
1	$0.0075 \sim 1.9926$	0	0	0.333
2	$0.0075 \sim 1.9926$	0	0	0
3	$2 \sim 2$	0	0	0
4	$0.2284 \sim 1.7716$	0	$-\frac{\pi}{2}$	0
5	$0.2284 \sim 1.7716$	0	0	0
6	$2 \sim 2$	0	0	0
7	$0.0075 \sim 1.9926$	0	$\frac{\pi}{2}$	0.316
8	$0.0075 \sim 1.9926$	0	0	0
9	$2 \sim 2$	0	0	0
10	$1.0349 \sim 1.9994$	0.1	$\frac{\pi}{2}$	0
11	$1.0349 \sim 1.9994$	0	0	0
12	$0 \sim 0$	0	0	0
13	$0.0075 \sim 1.9926$	-0.1	$-\frac{\pi}{2}$	0.384
14	$0.0075 \sim 1.9926$	0	0	0
15	$2 \sim 2$	0	0	0
16	$0.2868 \sim 1.8870$	0	$\frac{\pi}{2}$	0
17	$0.2868 \sim 1.8870$	0	0	0
18	$1.0001 \sim 1.0001$	0	0	0
19	$0.0075 \sim 1.9926$	0.1	$\frac{\pi}{2}$	0
20	$0.0075 \sim 1.9926$	0	0	0
21	$2 \sim 2$	0	0	0
22	$0 \sim 0$	0	0	0.107

Table A.2: DH convention for Franka robot in PDO-IK formulation.

A.3 UR10

i	L_i	a_{i-1}	α_{i-1}	d_i
1	$0 \sim 2$	0	0	0.127
2	$0 \sim 2$	0	0	0
3	$0 \sim 2$	0	0	0
4	$0 \sim 2$	0	0	0
5	$0 \sim 2$	0	$-\frac{\pi}{2}$	0
6	$0 \sim 2$	0	0	0
7	$0 \sim 2$	0	0	0
8	$0 \sim 2$	0	0	0
9	$0 \sim 2$	0.6	0	0
10	$0 \sim 2$	0	0	0
11	$0 \sim 2$	0	0	0
12	$0 \sim 2$	0	0	0
13	$0 \sim 2$	0.6	0	0.164
14	$0 \sim 2$	0	0	0
15	$0 \sim 2$	0	0	0
16	$0 \sim 2$	0	0	0
17	$0 \sim 2$	0	$-\frac{\pi}{2}$	0.116
18	$0 \sim 2$	0	0	0
19	$0 \sim 2$	0	0	0
20	$0 \sim 2$	0	0	0
21	$0 \sim 2$	0	$\frac{\pi}{2}$	0.092
22	$0 \sim 2$	0	0	0
23	$0 \sim 2$	0	0	0
24	$0 \sim 2$	0	0	0

Table A.3: DH convention for UR10 robot in PDO-IK formulation.

A.4 H1

i	L_i	a_{i-1}	α_{i-1}	d_i
1	$1 \sim 1$	0	$-\frac{\pi}{2}$	0
2	$0.3596 \sim 1.6404$	0	$-\frac{\pi}{2}$	0
3	$1.1741 \sim 1.1741$	0	0	0
4	$0.0852 \sim 1.9148$	0.4	0	0
5	$0.2198 \sim 0.2198$	0	0	0
6	$0 \sim 2$	0.4	0	0
7	$0.9992 \sim 0.9992$	0	0	0
8	$1 \sim 1$	0	$-\frac{\pi}{2}$	0
9	$0.5831 \sim 1.4169$	0.1	0	0
10	$1 \sim 1$	0	0	0
11	$1 \sim 1$	0	$\frac{\pi}{2}$	0
12	$1 \sim 1$	-0	0	0
13	$0.5831 \sim 1.4169$	0	0	0
14	$1 \sim 1$	0	0	0
15	$2 \sim 2$	0.1	0	-0.174
16	$0 \sim 0$	0	π	0
17	$1 \sim 1$	-0.1	0	-0.174
18	$0.5831 \sim 1.4169$	0	0	0
19	$1 \sim 1$	0	0	0
20	$0.0773 \sim 1.9227$	0	0	0
21	$0.0773 \sim 1.9227$	0	0	0
22	$2 \sim 2$	0	0	0
23	$1 \sim 1$	0	0	0
24	$0.5831 \sim 1.4169$	0	$\frac{\pi}{2}$	0
25	$1 \sim 1$	0	0	0

Table A.4: DH convention for H1 robot in PDO-IK formulation from number 1 to 25.

i	L_i	a_{i-1}	α_{i-1}	d_i
26	$1 \sim 1$	0.2	0	0.430
27	$0.0092 \sim 1.9908$	0	-1.134	0
28	$0.0092 \sim 1.9908$	0	0	0
29	$2 \sim 2$	0	0	0
30	$1 \sim 1$	-0.2	0	0.430
31	$0.0092 \sim 1.9908$	0	-2.007	0
32	$0.0092 \sim 1.9908$	0	0	0
33	$2 \sim 2$	0	0	0
34	$1 \sim 1$	-0.1	0	0
35	$0 \sim 2$	0	$-\frac{\pi}{2}$	0
36	$0.9992 \sim 0.9992$	0	0	0
37	$0 \sim 0$	0	0	0.057
38	$0 \sim 0$	0	0	-0.057
39	$1 \sim 1$	0	-0.436	0
40	$1 \sim 1$	0	0.436	0
41	$0.2405 \sim 1.7595$	0	$\frac{\pi}{2}$	0
42	$0.2405 \sim 1.7595$	0	$\frac{\pi}{2}$	0
43	$0.2405 \sim 1.7595$	0	0	0
44	$0.2405 \sim 1.7595$	0	0	0
45	$1.1847 \sim 1.1847$	0	0	0
46	$1.1847 \sim 1.1847$	0	0	0
47	$0.0852 \sim 1.9148$	0.4	0	0
48	$0.2198 \sim 0.2198$	0	0	0
49	$1 \sim 1$	0.1	0	0
50	$1 \sim 1$	0.1	0	0

Table A.5: DH convention for H1 robot in PDO-IK formulation from number 26 to 50.

i	L_i	a_{i-1}	α_{i-1}	d_i
51	$0.0089 \sim 1.9911$	0	$\frac{\pi}{2}$	0
52	$0.0089 \sim 1.9911$	0	$\frac{\pi}{2}$	0
53	$0.0089 \sim 1.9911$	0	0	0
54	$0 \sim 0$	0	0	0
55	$0.0089 \sim 1.9911$	0	0	0
56	$2 \sim 2$	0	0	0
57	$0.3596 \sim 1.6404$	0.4	0	0
58	$1.1741 \sim 1.1741$	0	0	0
59	$0 \sim 0$	0	0	-0.198
60	$0 \sim 0$	0	0	-0.198
61	$0.1780 \sim 1.8220$	0	$-\frac{\pi}{2}$	0
62	$0.1780 \sim 1.8220$	0	$-\frac{\pi}{2}$	0
63	$0.1780 \sim 1.8220$	0	0	0
64	$0.1780 \sim 1.8220$	0	0	0
65	$1.7776 \sim 1.7776$	0	0	0
66	$1.7776 \sim 1.7776$	0	0	0
67	$0 \sim 0$	0.3	0	0
68	$0 \sim 0$	0.3	0	0
69	$0 \sim 0$	0	$\frac{\pi}{2}$	0.200
70	$0 \sim 0$	0	0	0.700

Table A.6: DH convention for H1 robot in PDO-IK formulation from number 50 to 70.

Bibliography

[1] Katrin Baumgärtner, Yizhen Wang, Andrea Zanelli, and Moritz Diehl. 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)*, pages 558–563. IEEE, 2022. 4.2

[2] Charles G Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of computation*, 19(92):577–593, 1965. 3.2

[3] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *2015 international conference on advanced robotics (ICAR)*, pages 510–517. IEEE, 2015. 6.1

[4] Yu Chen, Yilin Cai, Jinyun Xu, Zhongqiang Ren, Guanya Shi, and Howie Choset. Propagative distance optimization for constrained inverse kinematics. *arXiv preprint arXiv:2406.11572*, 2024. 1

[5] John J Craig. *Introduction to robotics*. Pearson Educacion, 2006. 3.1

[6] Gordon M Crippen, Timothy F Havel, et al. *Distance geometry and molecular conformation*, volume 74. Research Studies Press Taunton, 1988. 1

[7] John E Dennis, Jr and Jorge J Moré. Quasi-newton methods, motivation and theory. *SIAM review*, 19(1):46–89, 1977. 3.2

[8] Arati S Deo and Ian D Walker. Adaptive non-linear least squares for inverse kinematics. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 186–193. IEEE, 1993. 2.1

[9] Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5 (6):345–345, 1962. 5.4

[10] Matthew Giamou, Filip Marić, David M Rosen, Valentin Peretroukhin, Nicholas Roy, Ivan Petrović, and Jonathan Kelly. Convex iteration for distance-geometric inverse kinematics. *IEEE Robotics and Automation Letters*, 7(2):1952–1959, 2022. 1, 2.2

[11] Jun Han and Claudio Moraga. The influence of the sigmoid function parameters

on the speed of backpropagation learning. In *International workshop on artificial neural networks*, pages 195–201. Springer, 1995. 4.2, 4.2

[12] Li Han and Lee Rudolph. Inverse kinematics for a serial chain with joints under distance constraints. In *Robotics: Science and systems*, 2006. 1, 2.2

[13] Timothy F Havel. Distance geometry: Theory, algorithms, and chemical applications. *Encyclopedia of Computational Chemistry*, 120:723–742, 1998. 5.4

[14] Karan Khokar, Patrick Beeson, and Rob Burridge. Implementation of kdl inverse kinematics routine on the atlas humanoid robot. *Procedia Computer Science*, 46: 1441–1448, 2015. 4.2

[15] Gregory Lantoine and Ryan Russell. A hybrid differential dynamic programming algorithm for robust low-thrust optimization. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, page 6615, 2008. 4.2

[16] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989. 3.2

[17] Filip Marić, Matthew Giamou, Soroush Khoubyarian, Ivan Petrović, and Jonathan Kelly. Inverse kinematics for serial kinematic chains via sum of squares optimization. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7101–7107. IEEE, 2020. 1, 2.2, 4.1

[18] Filip Marić, Matthew Giamou, Ivan Petrović, and Jonathan Kelly. Inverse kinematics as low-rank euclidean distance matrix completion. *arXiv preprint arXiv:2011.04850*, 2020. 1, 2.2

[19] Filip Marić, Matthew Giamou, Adam W Hall, Soroush Khoubyarian, Ivan Petrović, and Jonathan Kelly. Riemannian optimization for distance-geometric inverse kinematics. *IEEE Transactions on Robotics*, 38(3):1703–1722, 2021. 1, 2.2, 4.3, 5.4, 6.1

[20] Josep Marti-Saumell, Joan Solà, Carlos Mastalli, and Angel Santamaria-Navarro. Squash-box feasibility driven differential dynamic programming. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7637–7644. IEEE, 2020. 4.2

[21] Yoshihiko Nakamura and Hideo Hanafusa. Inverse kinematic solutions with singularity robustness for robot manipulator control. 1986. 2.1

[22] Josep M Porta, Lluís Ros, and Federico Thomas. Inverse kinematics by distance matrix completion. 2005. 1

[23] Josep M Porta, Lluís Ros, Federico Thomas, and Carme Torras. A branch-and-prune solver for distance constraints. *IEEE Transactions on Robotics*, 21(2): 176–187, 2005. 1, 2.2

[24] Manfred J Sippl and Harold A Scheraga. Cayley-menger coordinates. *Proceedings*

of the National Academy of Sciences, 83(8):2283–2287, 1986. 1, 2.2

[25] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175. IEEE, 2014. 4.2

[26] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109. 6.3

[27] Charles W Wampler. Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):93–101, 1986. 2.1

[28] L-CT Wang and Chih-Cheng Chen. A combined optimization method for solving the inverse kinematics problems of mechanical manipulators. *IEEE Transactions on Robotics and Automation*, 7(4):489–499, 1991. 2.1

[29] Tillmann Weisser, Jean B Lasserre, and Kim-Chuan Toh. Sparse-bsos: a bounded degree sos hierarchy for large scale polynomial optimization with sparsity. *Mathematical Programming Computation*, 10:1–32, 2018. 1, 2.2

[30] Daniel E Whitney. Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on man-machine systems*, 10(2):47–53, 1969. 2.1

[31] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)*, 23(4):550–560, 1997. 6.1