박사학위논문
Ph.D. Dissertation

동적 장애물로부터의 안전한 사용자 명령 수행을 위한 로봇 팔 움직임 계획

Robot Arm Motion Planning for Safe Execution of User Commands against Dynamic Obstacles

2023

강 민 철 (姜 旻 澈 Kang, Mincheul)

한국과학기술원

Korea Advanced Institute of Science and Technology

박사학위논문

동적 장애물로부터의 안전한 사용자 명령 수행을 위한 로봇 팔 움직임 계획

2023

강민철

한국과학기술원

전산학부

동적 장애물로부터의 안전한 사용자 명령 수행을 위한 로봇 팔 움직임 계획

강민철

위 논문은 한국과학기술원 박사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2022년 11월 23일

심사위원장 윤성의 (인)
심사위원 조성호 (인)
심사위원 명현 (인)
심사위원 김영준 (인)
심사위원 박대형 (인)

Robot Arm Motion Planning for Safe Execution of User Commands against Dynamic Obstacles

Mincheul Kang

Advisor: Sung-Eui Yoon

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

Daejeon, Korea
November 23, 2022

Approved by

Sung-Eui Yoon
Professor of School of Computing

The study was conducted in accordance with Code of Research Ethics[1].

---

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

DCS 강민철. 동적 장애물로부터의 안전한 사용자 명령
수행을 위한 로봇 팔 움직임 계획. 전산학부. 2023년. 57+iv 쪽. 지도교수:
윤성의. (영문 논문)
Mincheul Kang. Robot Arm Motion Planning for Safe Execution of
User Commands against Dynamic Obstacles. School of Computing. 2023.
57+iv pages. Advisor: Sung-Eui Yoon. (Text in English)

초 록

원격 조작은 사용자의 명령에 의해 로봇을 조종하는 것으로, 핵 발전소나 의료 시설과 같은 특수한 장소에서 사람이 하기에 위험하거나 어려운 일을 대신하는데 많이 사용되어 왔다. 최근에는 집과 편의점 같은 우리 주변 환경으로 영역을 확장해 직접 그 장소에 가지 않고 로봇을 통해 일을 수행하기 위한 연구들이 진행되고 있다. 우리 주변 환경은 다양하고 동적인 장애물들이 많기 때문에, 이런 장애물들로부터 안전한 사용자 명령 수행이 필요하다. 장애물 회피에는 많은 계산량이 소요되는 반면, 사용자는 명령에 즉각적인 수행을 바란다. 그렇기에 장애물 회피의 계산량을 줄이면서도 사용자의 명령을 따르는 조인트 구성을 찾는 것이 중요한 문제이다.

본 논문에서는 다양하고 동적인 장애물로부터 안전한 사용자 명령 수행을 위한 로봇 팔 움직임 계획법을 제안한다. 또한, 본 논문은 로봇의 엔드-이펙터에 주어지는 사용자의 명령에 대해 다양한 조인트 구성들을 가지는 여유 자유도 로봇을 대상으로 한다. 그에 따라, 본 논문은 사용자 명령에 대한 다양한 조인트 구성 중 충돌 없는 조인트 구성을 찾는 것과 사용자의 부주의로 인한 위험한 명령을 안전하게 조정하는 것을 다룬다. 구체적으로, 1) 장애물을 피해 주어진 엔드-이펙터 경로를 따라가는 조인트 경로를 생성하기 위한 경로 최적화 방법을 제안한다. 다음으로, 2) 동적 장애물이 있는 환경에서 실시간으로 주어지는 사용자 명령을 수행하기 위한 딥러닝 기반의 실시간 역기구학 기법을 제안한다. 마지막으로, 3) 사용자의 부주의로 인한 위험한 명령을 안전하도록 조정하기 위한 강화학습 기반의 사용자 명령 조정법을 제안한다. 제안한 방법은 실제 센서 데이터와 로봇을 이용해서 실험하였고, 다양하고 동적인 장애물들로부터 사용자 명령 수행의 안전성을 높임을 보였다.

핵 심 낱 말 움직임 계획, 충돌 회피, 역기구학, 여유 자유도 로봇, 원격 조작

Abstract

Remote manipulation is to control a robot from a user command and has been primarily used to perform dangerous or difficult tasks for humans in special places such as nuclear power plants or medical facilities. Recently, the scope of remote manipulation has expanded to environments around us (e.g., homes and convenience stores), and studies have been conducted to perform tasks through robots without directly going to those places. Since environments around us have many different and dynamic obstacles, it is necessary to perform user commands safely from these obstacles. Obstacle avoidance requires high computational costs, whereas users expect immediate execution of commands. Therefore, it is an important issue to find collision-free joint configurations that follow user commands while reducing the amount of computation for obstacle avoidance.

In this dissertation, we present robot arm motion planning approaches for safe execution of user commands against various and dynamic obstacles. This dissertation also targets a redundant manipulator, which has various joint configurations for a user command given to the manipulator's end-effector. Accordingly, this dissertation deals with finding a collision-free joint configuration among various solutions for a user command and adjusting a risky command due to the user's unpredictability or carelessness to be safe. Specifically, we propose 1) a trajectory optimization method for generating joint configurations that follow a given end-effector path avoiding obstacles. We also propose 2) real-time collision-free inverse kinematics based on deep learning to perform consecutive user's real-time commands in dynamic environments. Lastly, we propose 3) a reinforcement learning-based user command adjustment method to adjust risky commands caused by the user's carelessness to be safe. We showed that our proposed methods increase safety from various and dynamic obstacles through experiments using the real robot and sensor data.

Keywords Motion Planning, Collision Avoidance, Inverse Kinematics, Redundant Manipulator, Remote Manipulation

Contents

Contents i
List of Tables iii
List of Figures iv

Chapter 1. Introduction 1
1.1 List of Related Papers 3

Chapter 2. TORM: Fast and Accurate Trajectory Optimization of Redundant Manipulator given an End-Effector Path 4
2.1 Introduction 4
2.2 Related Work 5
2.2.1 Inverse Kinematics 5
2.2.2 Motion Planning for Following an End-effector Path 6
2.2.3 Constrained Motion Planning 7
2.2.4 Gradient descent for a constrained problem 7
2.2.5 Avoiding local minima for trajectory optimization 7
2.3 Background 7
2.3.1 Problem Definition 8
2.3.2 Jacobian-based Inverse Kinematics 8
2.3.3 Optimization-based Motion Planning 9
2.4 Trajectory Optimization 10
2.4.1 Motivation and Overview 10
2.4.2 Two-Stage Gradient Descent 11
2.4.3 Adaptive Exploration 13
2.4.4 New Trajectory Generation 14
2.4.5 Trajectory Constraints 15
2.5 Experiments and Analysis 16
2.5.1 Experiment setting 16
2.5.2 Comparison with prior methods 18
2.5.3 Analysis of our proposed methods 21
2.5.4 Analysis with different robots 23

Chapter 3. RCIK: Real-Time Collision-Free Inverse Kinematics using a Collision-Cost Prediction Network 24
3.1 Introduction 24

3.2 Related Work 24
3.2.1 Inverse Kinematics 25
3.2.2 Collision Avoidance for Manipulation Planning 26
3.3 Overview 26
3.3.1 Problem Definition 26
3.3.2 Motivation 27
3.4 Real-time Collision-free Inverse Kinematics 28
3.4.1 IK candidates 28
3.4.2 Objective Function 29
3.4.3 Collision-Cost Prediction Network 30
3.5 Experiments And Analysis 32
3.5.1 Experiments in dynamic environments 33
3.5.2 Analysis 34

Chapter 4. Towards Safe Remote Manipulation: User Command Adjustment based on Risk Prediction for Dynamic Obstacles 37
4.1 Introduction 37
4.2 Related Work 38
4.2.1 Inverse kinematics for remote manipulation 38
4.2.2 Dynamic obstacle avoidance 39
4.3 Background 39
4.3.1 Problem definition 39
4.3.2 Reinforcement Learning 40
4.4 Approach 40
4.4.1 Overview 40
4.4.2 User command adjustment 41
4.4.3 Risk prediction for dynamic obstacles 42
4.4.4 Training details 43
4.5 Experiments 44
4.5.1 Evaluation 44
4.5.2 Real robot test 46

Chapter 5. Conclusion 48

Bibliography 49

Acknowledgments in Korean 56

Curriculum Vitae in Korean 57

List of Tables

2.1 Notation table . . . 9

2.2 Results of different methods, CHOMP, Trajopt, EIGS, and ours in four problems including external obstacles. The bold text indicates the smallest pose error, and the underline indicates the second smallest pose error. . . . 17

2.3 Results of different methods, CHOMP, Trajopt, EIGS, RelaxedIK, Stampede, and ours in two non-obstacle problems. RelaxedIk is a real-time planner and does not provide better trajectories with more planning time; we provide its results while it cannot be compared in the equal-time comparison. RelaxedIK and Stampede experimented with non-obstacle problems since these methods only check self-collision. The bold text indicates the smallest pose error, and the underline indicates the second smallest pose error. . . . 18

2.4 Results of the number of iterations and explorations during 50 seconds for optimization-based approaches, CHOMP, Trajopt, and ours. Parenthesis indicates the time (s) for one update of its optimizer. We regard one update of Trajopt as one execution of the quadratic solver. . . . 20

2.5 Analysis of components of our proposed method by substituting them to alternative methods, the simple integration (SI), iterative exploration (IE), and extracting sub-sampled poses at 10 and 50 intervals. The last row shows the results of our complete method. . . . 20

3.1 Construction times of the SDF with varying resolutions and computation time of collision costs for 50 IK candidates using the SDF and the CCPN. . . . 33

3.2 Results with 20 experiments with different methods in dynamic environments. . . . 34

3.3 Results of using the SDF with varying resolutions and the CCPN trained by varying w_{rank} in the validation dataset. The number of unique data indicates the number of different collision costs among 50 IK candidates. . . . 35

3.4 Results of different methods with varying configurations of robots in static environments. RelaxedIK only considers self-collision, and thus is tested in two non-obstacle scenes. . . . 35

4.1 Results in two kinds of obstacle environments: both static and dynamic obstacles, and only static obstacles. . . . 44

4.2 The mean and standard deviation of difference between ρ computed from Eq. 4.6 and $\hat{\rho}$ predicted by the RPN. . . . 46

List of Figures

1.1 These figures show (a) our desired output and the test results when there is no consideration of (b) obstacle avoidance and (c) joint velocity limit between commands; the joint velocity limit is to maintain continuity between commands. The red and blue lines indicate the given commands and computed end-effector movements, respectively. 2

1.2 This figure shows the end-effector pose for a user command and its IK solutions for the redundant manipulator, seven-DoF Fetch manipulator. The manipulator has infinity joint configurations for one end-effector pose; we visualize 100 joint configurations. To avoid obstacles, the robot links associated with the end-effector (red circle) depend on the user command. The other links (blue circle) are determined by considering obstacles under various possibilities. 2

1.3 These figures outline our proposed methods for the safe execution of user commands. 3

2.1 These figures show a sequence of maneuvering the Fetch manipulator to follow the specified end-effector path (red lines). Our method generates the trajectory that accurately follows the given end-effector path, while avoiding obstacles such as the pack of A4 paper and the table. 5

2.2 This figure shows our problem that is synthesizing a feasible and accurate trajectory $\boldsymbol{\xi}$ for a given end-effector path X. The red line is X, which is approximated by end-effector poses $\widetilde{X} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ (green dots). The trajectory $\boldsymbol{\xi}$ is computed at end-effector poses $\widetilde{X}$. The part of the synthesized trajectory shows that the end-effector follows the red line, and the sequence of joint configurations is smooth and collision-free, while avoiding obstacles such as the blue box and the table. 8

2.3 This figure shows an abstraction of our overall algorithm. The blue boxes represent the iterative process, and one exploration means the refinement process starting from a newly generated trajectory. 10

2.4 This shows the illustration of optimization progress for a simple integration using Eq. 2.7, and our TSGD (Eq. 2.9). Each gray contour line represents equal F_{pose}, and the center, marked $\otimes$, of innermost contour has the smallest F_{pose}. The simple integration (a) is difficult to reach the minimum F_{pose} due to conflicts between three different terms; black arrows represent the update process of the simple integration through the sum of three functional gradients. On the other hand, our TSGD approaches to the minimum F_{pose} through alternatively updating the trajectory toward the feasible set C shown in the red region using ∇F_{obs} and ∇F_{smooth} (green arrow) and updating the trajectory toward the minimum F_{pose} using ∇F_{pose} (blue arrow). 11

2.5 This illustrates two problematic cases for ineffective exploration; the information of this figure can be seen in Figure 2.4. Note that these figures are examples of one exploration from a newly generated trajectory, ξ_0. (a) is the case where F_{pose} gradually increases due to the strong tendency to make a feasible trajectory. In contrast, (b) is the case where F_{pose} is gradually reduced, but it is still not feasible. 12

2.6 This figure shows sub-sampled poses S (blue dots), extracted from finely divided end-effector poses $\widetilde{X}$ (red dots), in the problem of square tracing and writing “hello”. From S, we construct a new trajectory ξ_{new} that starts with random IK solutions and is found in a greedy manner minimizing Eq. 2.7. (a,d) and (b,e) are extracted uniformly at 10 and 50, respectively. (c) and (f) are the results computed by the path simplification using the DP algorithm. () represents the number of extracted sub-sampled poses as blue dots. (b) has too many S, increasing the time to generate a new trajectory $\boldsymbol{\xi}_{new}$. On the other hand, (d) has a small number of S, which is not enough to make an potentially good trajectory. On the other hand, we apply the path simplification to extract the appropriate S (c,f) regardless of various forms of paths. 14

2.7 This shows six problems with three different robots, Kuka 7-DoF, Fetch 7-DoF, and Hubo 8-DoF. Four problems, (a) to (d), include external obstacles and two problems, (e) and (f), do not have external obstacles. The red line represents the given end-effector path. 16

2.8 This shows the pose error over the planning time of different methods in three different problems. Since (a) and (c) include external obstacles, RelaxedIK and Stampede are excluded from the experiments. Also, the start configurations of (a) and (c) are fixed. We visualize graphs once an initial solution is computed. 19

2.9 This shows the pose error as a function of the planning time of various ablated methods. Our complete method contains two-stage gradient descent (TSGD), adaptive exploration (AE), and path simplification (PS). We substitute them with the simple integration (SI), the iterative exploration (IE) with the fixed number of iterations, and extracting sub-sampled poses at 10 and 50 intervals. The gray dotted lines represent the results shown in Table 2.5. 21

2.10 This shows the pose error over the planning time of different methods with three different robots in “S” tracing with two boxes. 22

3.1 These figures show the progress of our method in a dynamic environment. (a) Our method performs consecutively given 6-DoF commands while avoiding a dynamic obstacle, a cart in this case. (b) To avoid the cart, we compute a collision cost, F_{col}, using our collision-cost prediction network (CCPN). The CCPN uses an occupancy grid updated through sensor data to reflect changing environments immediately. Finally, our method finds a desired joint configuration that allows the robot to follow the given command accurately while avoiding obstacles. Our algorithm conducts all of these operations within 30ms for each command, mainly due to CCPN. 25

3.2 This shows an overview of our approach. (a) Our method synthesizes a desired joint configuration q_{des} that matches the target end-effector pose x_{tar} as a 6-DoF command and has the motion feasibility considering the current configuration q_{cur}. (b) First, we generate IK candidates Q_{cand} with high feasibility based on an optimization-based IK approach. (c) With checking constraints, such as collision and joint velocity limits, for Q_{cand}, we select the best one using our objective function for collision avoidance with obstacles and smoothness with previous joint configurations. (d) For 6-DoF commands given consecutively, denoted with the yellow arrow, our method sequentially synthesizes joint configurations that accurately realize the commands while avoiding collision and achieving smoothness, as indicated by the red arrow. 26

3.3 These figures show (a) the simplified robot links, spheres B, used for manipulation and (b) signed distance field for external obstacles, the table and gray box. The color of cells represents the distance from the closest obstacles; as the distance from a cell to the obstacle becomes smaller, its color gets closer to the red, and vice-versa to blue. 29

3.4 This figure shows the collision-cost prediction network (CCPN), which consists of two feature extractors and one regression module. 30

3.5 These figures show one of the generated environments to obtain training data. From (a) randomly distributed obstacles, we construct (b) an occupancy grid with updates from sensor data. 31

3.6 These figures show our experiment in the dynamic environments using a real robot, the Fetch manipulator. The robot receives 6-DoF commands consecutively to trace (a) circle and (b) square, while avoiding an incoming book in the direction of the green arrow. Additionally, (b) includes a static obstacle, a table. Note that we give 6-DoF commands consecutively to follow the yellow line. 32

4.1 This shows the visualization of the user monitor screen for remote manipulation. In the visualized situation, the user gives commands (red arrow) to the robot to move a water bottle (orange boxes) from the dotted magenta box to the direction of the magenta arrow, and a person suddenly appears in the camera's field of view (blue arrow). Our method predicts high risk of collision, which is expressed between 0 and 1, and for safe remote manipulation, it adjusts the user's command accordingly to move the robot away from the obstacle (green arrow). The cells of the occupancy grid are colored according to the z-axis value. 38

4.2 This shows the overall system flow of our approach. 40

4.3 This shows the structure of obstacle avoidance command network (OACN). The encoder is a 3-D convolutional network following the obstacle feature extractor of RCIK [1]. 41

4.4 This shows our risk prediction network (RPN) for dynamic obstacles. The RPN recurrently takes joint positions x_q^{pos} and an occupancy grid difference ΔO. We apply VoxNet [2] to extract the feature of an occupancy grid difference ΔO. Furthermore, we utilize the gate recurrent unit (GRU) [3] to preserve previous robot state and obstacle information. 42

4.5 (a) shows an example of a randomly generated obstacle environment. Gray boxes are static obstacles, and the orange box is a dynamic obstacle. The green arrow indicates the moving direction of the dynamic obstacle. (b) shows the occupancy grid difference ΔO between O_t and O_{t-1} constructed from sensor data for (a). Each cell in O is represented as 1 if there is an obstacle or 0 otherwise. Therefore, in the occupancy grid difference, 1 (blue boxes) and -1 (red boxes) indicates newly detected and disappeared obstacles, respectively. From the occupancy grid difference, we can see the moving direction of the dynamic obstacle, but the noise data due to sensor noise occurs near static obstacles (dotted black circles). 43

4.6 These figures show examples of our results in (a) an environment with static and dynamic obstacles and (b) an environment with only static obstacles. Orange and gray boxes are dynamic and static obstacles, respectively, and red and green arrows indicate the magnitude and direction of given command and adjusted command, respectively. The numbers in the upper right of figures are predicted risks for dynamic obstacles $\hat{\rho}$s. (a) shows that as the dynamic obstacle approaches to the manipulator, our method predicts a high $\hat{\rho}$ and adjusts the given command to avoid the obstacle. In contrast, as the dynamic obstacle moves away from the manipulator, our method follows the given commands, predicting a low $\hat{\rho}$. (b) shows that our method robustly copes with the noisy sensor data by predicting a near-zero $\hat{\rho}$ in a static environment. 45

4.7 These figures show tested problems in a real environment. Red arrow and blue arrow indicate the moving direction of the user commands and a person, respectively. In both problems, there is a risk of collision since the moving direction of the user commands and the person intersect. (a) is that the user gives commands to the robot to pick the black water bottle (red dotted box) and the person moves to pick the green water bottle (blue dotted box). (b) is that the user gives commands to the robot to transfer the water bottle and the person suddenly appears from behind the robot. . 46

Chapter 1. Introduction

Remote manipulation is controlling a robot from a user's command at a distance. It has been primarily used to perform tasks on behalf of humans at special-purpose sites. For example, surgical robots perform sophisticated operations that could be difficult for humans [4], and remote robots in hazardous places, such as nuclear power plants [5], reduce the risk of human beings by putting them in place of humans. Recently, the scope of remote manipulation has been expanded to environments around us, such as convenience stores (e.g., Telexistence[1]) and homes [6–9]. In environments around us, there are various obstacles, both static and dynamic, and avoiding such obstacles is necessary for the safe execution of user commands.

In this dissertation, a user gives a command to the robot arm's end-effector in the Cartesian space, and our target robot is a redundant manipulator, which has multiple joint configurations for one end-effector pose; the redundant manipulator has a controllable degree-of-freedom (DoF) greater than six, which is the DoF of the Cartesian space. Generally, inverse kinematics (IK) methods find a joint configuration from the command in the Cartesian space. However, basic inverse kinematic methods (e.g., KDL[2], IKfast [10], and Trac-IK [11]) do not take into account several constraints to follow user commands, such as collision with obstacles, joint velocity limits between commands, and kinematic singularity (Figure 1.1) [12]. Several works [12–15] point out the necessity of these constraints and consider the constraints to follow user commands. On the other hand, various issues still remain, such as accuracy, computation time, applicability to real robots, and consideration of dynamic and external obstacles.

A user anticipates that a robot will respond to commands immediately, whereas avoiding various and dynamic obstacles require high computational costs. This is because collision detection and distance computation between robot links and obstacles must be repeatedly performed in order to avoid the obstacles. Furthermore, it is difficult to handle dynamic obstacles due to their uncertain and sudden movement. Therefore, this dissertation focus on reducing the computational overhead of collision avoidance and handling dynamic obstacles.

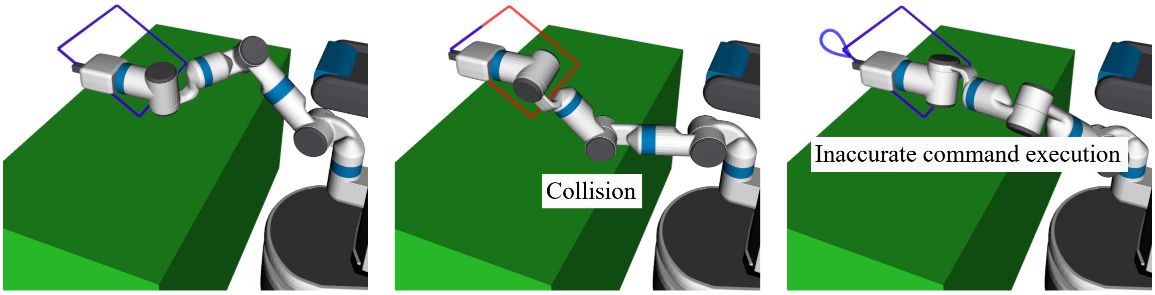
To overcome the challenges mentioned earlier, we present robot arm motion planning approaches for the safe execution of user commands in environments surrounding various and dynamic obstacles. Considering the characteristic of following a user command, we divide the robot arm into two parts. One is that the robot links associated with the end-effector (red circle in the Figure 1.2) depend on the user command. Other robot links (blue circle in the Figure 1.2) have various possibilities for the user command since our target robot is a redundant manipulator. Therefore, we deal with finding collision-free joint configurations from user commands and adjusting a risky command caused by the user's mistake or unfamiliar control skills to be safe. Also, this dissertation aims to apply our proposed methods in a real environment. To do that, the proposed methods use a sensor data and handle its noise, which is the difficulty of a real environment. In addition, this dissertation introduces the existing methods and discusses improvements of our proposed methods.

We first solve a problem to follow a given end-effector path while avoiding obstacles. Prior works [13, 14] to follow the path suggest sampling-based IK methods (Figure 1.3a). These methods can achieve a global optimal trajectory, but requires excessive computation to cover multiple solutions for a redundant manipulator. To efficiently search a solution space, optimization-based motion planning approaches [16, 17] synthesize a trajectory that follows the given path by adding the task constraint to their optimizer. However, these planners have difficulties dealing with diverse constraints and complex paths since they tend to fall into a local minimum. In Chapter 2,

---

[1] https://tx-inc.com

[2] http://wiki.ros.org/kdl



(a) Desired output (b) No considering obstacle avoidance (c) No considering joint velocity limits between commands

Figure 1.1: These figures show (a) our desired output and the test results when there is no consideration of (b) obstacle avoidance and (c) joint velocity limit between commands; the joint velocity limit is to maintain continuity between commands. The red and blue lines indicate the given commands and computed end-effector movements, respectively.

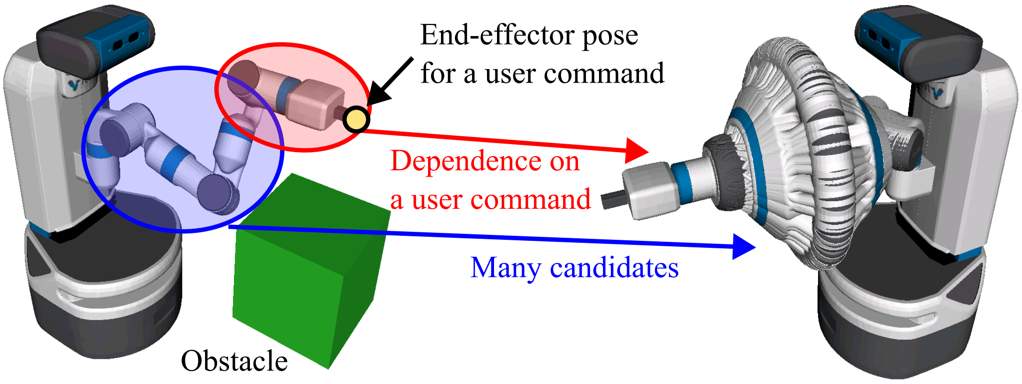
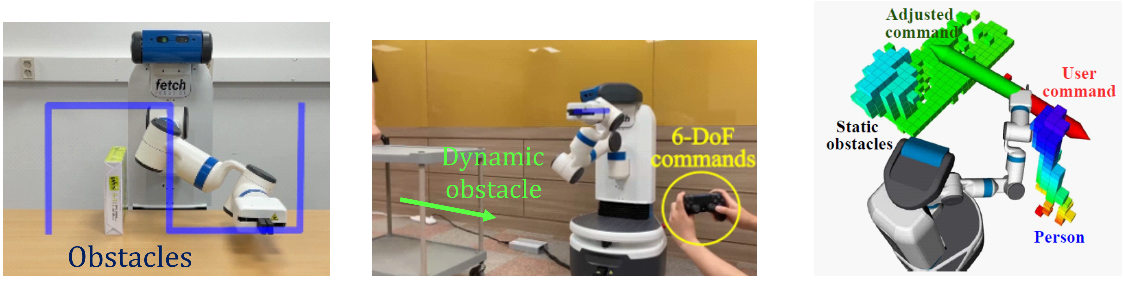


Figure 1.2: This figure shows the end-effector pose for a user command and its IK solutions for the redundant manipulator, seven-DoF Fetch manipulator. The manipulator has infinity joint configurations for one end-effector pose; we visualize 100 joint configurations. To avoid obstacles, the robot links associated with the end-effector (red circle) depend on the user command. The other links (blue circle) are determined by considering obstacles under various possibilities.

we present a trajectory optimization of a redundant manipulator (TORM) for synthesizing a trajectory that is accurately following a given path while satisfying several constraints, joint position and velocity limits, kinematic singularity, and collisions with obstacles. Our method integrates the Jacobian-based IK solving method and an optimization-based motion planning approach to holistically incorporate these constraints into our optimization process.

Next, we solve a problem of calculating solutions in real-time for consecutively given user commands (Figure 1.3b). As prior methods, RelaxedIK [12] and CollisionIK [15] suggest a non-linear optimization methods to solve the problem in real-time. On the other hand, these approaches do not take into account difficulties of a real environment, such as dynamic obstacles or sensor noise. In Chapter 3, we present real-time collision-free inverse kinematics (RCIK) to handle dynamic obstacles in real-time. Our method reduces the massive computational overhead associated with collision using a neural network. Our network estimates a collision cost by grasping environment information from an occupancy grid updated from sensor data in real-time.



(a) Trajectory optimization for an end-effector path (b) Real-time collision-free inverse kinematic (c) Risk-aware user command adjustment

Figure 1.3: These figures outline our proposed methods for the safe execution of user commands.

Lastly, we solve a problem of adjusting a risky user commands to be safe unlike the previous two methods dealing with executable user commands (Figure 1.3c). Real-time IK solvers [1, 15] have the characteristic of following a user's command, whereas a user can make risky commands due to unexpected situations or insufficient control skill. In the case of static obstacles, we can keep the safety by naïvely stopping a robot, even if the user's command has a possibility of causing a collision. On the other hand, it is difficult to guarantee the safety against dynamic obstacles with uncertain motions [18, 19]. In Chapter 4, we present a risk-aware user command adjustment method that predicts the risk of dynamic obstacles and then adjusts a user command based on the predicted risk. Also, we handle sensor noises using a deep neural network to apply our method in a real environment.

1.1 List of Related Papers

This dissertation is partially related to the following published papers:

- **Mincheul Kang** and Sung-Eui Yoon, *Analysis and Acceleration of TORM: Optimization-based Planning for Path-wise Inverse Kinematics*, Autonomous Robots, vol. 46, pp. 599-615, 2022.
- **Mincheul Kang**, Yoonki Cho, and Sung-Eui Yoon, *RCIK: Real-Time Collision-Free Inverse Kinematics Using a Collision-Cost Prediction Network*, IEEE Robotics and Automation Letters (RA-L), vol. 7, no. 1, pp. 610-617, 2021.
- **Mincheul Kang**, Heechan Shin, Donghyuk Kim, and Sung-Eui Yoon, *TORM: Fast and Accurate Trajectory Optimization of Redundant Manipulator given an End-Effector Path*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020.

Chapter 2. TORM: Fast and Accurate Trajectory Optimization of Redundant Manipulator given an End-Effector Path

2.1 Introduction

Remote control of various robots has been one of the main challenges in robotics, while it is commonly used for cases where it is difficult or dangerous for a human to perform tasks [20, 21]. In this remote control scenario, a robot has to follow the task command accurately while considering its surrounding environment and the constraints of the robot itself.

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

Path-wise IK approaches [12, 13] solve this problem using non-linear optimization by considering aforementioned constraints (Sec. 2.2.1). These approaches avoid self-collisions using a neural network, but they do not deal with collisions for external obstacles. On the other hand, prior methods [14, 22] based on a motion planning approach consider external obstacles and use IK solutions to synthesize a trajectory following the desired path in the Cartesian space. By simply using IK solutions, however, these methods have time or structural difficulties in getting a highly-accurate solution (Sec. 2.2.2).

Optimization-based motion planners [16, 17] can make a trajectory that follows the given path by adding the kinematic constraints to their optimizer. However, these planners have difficulty dealing with diverse constraints and complex paths since it tends to fall into a local minimum (Sec. 2.2.3). Our method is also an optimization-based approach, but we alleviate this problem by performing an appropriate update method for a path-wise IK problem and exploring various trajectories efficiently.

In this work, we present a trajectory optimization of a redundant manipulator (TORM) for synthesizing a trajectory that is accurately following a given path as well as smooth and collision-free against the robot itself and external obstacles (Figure 2.1). Our method incorporates these properties into the optimization process by holistically integrating the Jacobian-based IK solving method and an optimization-based motion planning approach. For effective optimization, we consider different properties of our objectives through a two-stage update manner, which alternates making a constraint-satisfying trajectory and following a given end-effector path accurately (Sec. 2.4.2). To avoid local minima within our optimization process, we iteratively perform exploration for other alternative trajectories. We also accelerate the iterative exploration by adaptively deciding the stop of each exploration based on the observation of optimization results (Sec. 2.4.3). In addition, in order to strike a good balance between the quality and generation time of trajectory for the exploration, we apply a path simplification to extract sub-sampled poses with appropriate intervals and generate the trajectory using the random IK solution at the extracted poses (Sec. 2.4.4).

To compare our method with the prior methods, CHOMP [16], Trajopt [17], RelaxedIK [12], Stampede [13] and the work of Holladay et al. [14], we test four scenes with external obstacles and two non-obstacle problems. Furthermore, we test three different configurations of robots, Kuka 7-DoF, Fetch 7-DoF, and Hubo 8-DoF (Sec. 2.5.1).

Overall, we observe that our method achieves more accurate solutions given the equal planning time over

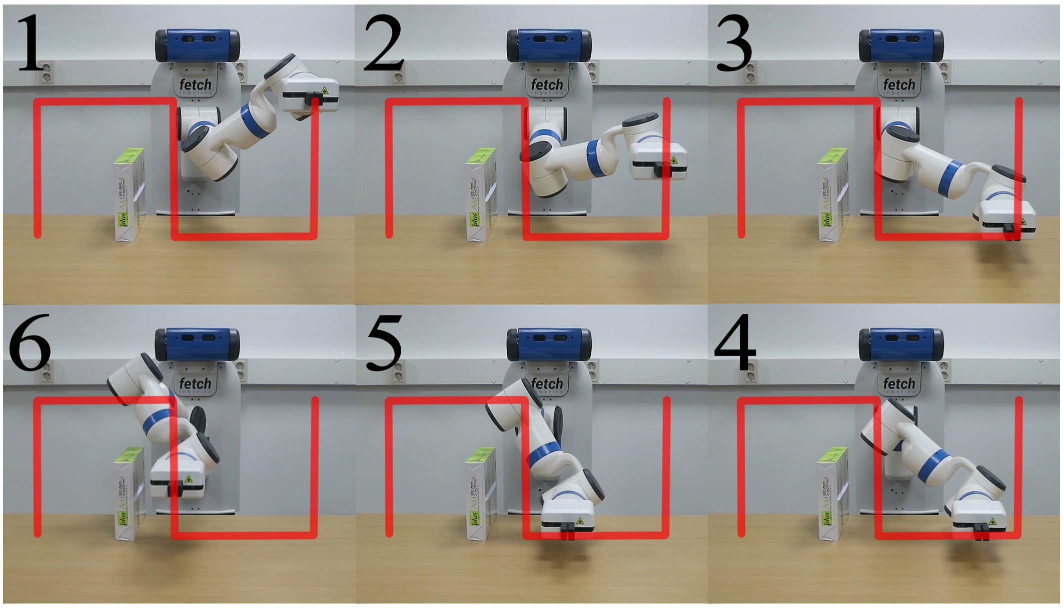


Figure 2.1: These figures show a sequence of maneuvering the Fetch manipulator to follow the specified end-effector path (red lines). Our method generates the trajectory that accurately follows the given end-effector path, while avoiding obstacles such as the pack of A4 paper and the table.

the tested prior methods. Also, our method robustly minimizes the pose error reasonably fast with the anytime property, which quickly finds an initial trajectory and refines the trajectory [23] (Sec. 2.5.2). This result mainly arises from the holistic optimization process. To show the benefits of our proposed methods, we test by replacing some of our methods with other methods (Sec. 2.5.3) and analyze the results of three different configurations of robots (Sec. 2.5.4).

2.2 Related Work

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

2.2.1 Inverse Kinematics

Inverse kinematics (IK) has been studied widely to find a joint configuration from an end-effector pose [24]. In the case of a redundant manipulator where our target robots belong to, there can be multiple joint configurations from a single end-effector pose. For this problem, several techniques for quickly finding solutions have been proposed [10, 25]. In particular, task-priority IK algorithms [26–28] prioritize solutions based on an objective function for each purpose, e.g., kinematic singularity or constraint task.

Most methods that synthesize a trajectory geometrically constrained for an end-effector pose use the Jacobian matrix for finding a feasible solution [29–32]. Unfortunately, the Jacobian matrix is the first derivative of the vector-valued function with multiple variables and thus it can cause false-negative failures by getting stuck in local minima or convergence failures due to the limits of the joint angle [11].

Trac-IK [11] points out the problem and improves success rates by using randomly selected joint configurations and sequential quadratic programming. Nonetheless, its solutions do not guarantee continuity of a sequence of joint configurations to make a feasible trajectory [12]. In summary, the traditional IK approaches have different strengths and weaknesses for synthesizing a feasible trajectory.

To get alleviated solutions, many studies have presented optimization techniques with objective functions for synthesizing a feasible trajectory with matching end-effector poses. [33] use a geometric and temporal optimization to generate a dynamically-feasible trajectory from a sketch input. Recently, [12] proposed a real-time approach using a weighted-sum non-linear optimization, called RelaxedIK, to solve the IK problem for a sequence of motion inputs. Since the collision checking has a relatively large computational overhead, RelaxedIK uses a neural network for fast self-collision avoidance.

Based on RelaxedIK, two studies [13, 34] are proposed for synthesizing a highly-accurate trajectory on off-line. One of them is Stampede [13], which finds an optimal trajectory using a dynamic programming algorithm in a discrete-space-graph that is built by samples of IK solutions. The other work [34] generates multiple candidate trajectories from multiple starting configurations and then selects the best trajectory with a user guide by allowing a deviation if there are risks of self-collisions or kinematic singularities.

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

2.2.2 Motion Planning for Following an End-effector Path

Motion planning involves a collision-free trajectory from a start configuration to a goal configuration. Based on the framework of motion planning, two methods [14,22] are presented to follow the desired end-effector path in Cartesian space. One [22] uses an optimization-based method, specifically Trajopt [17], by applying the discrete Fréchet distance that approximately measures the similarity of two curves. Although the optimization-based motion planning approaches quickly find a collision-free trajectory using efficient collision avoidance methods, these approaches can be stuck in local minima due to several constraints (e.g., joint limits and collisions). To assist its optimizer, this approach separately plans a trajectory by splitting the end-effector path as a set of waypoints and then sampling an IK solution at each pose.

Its subsequent work [14] points out the limitation of the prior work that samples only one IK solution for each pose. Considering this property, it presents a sampling-based approach that iteratively updates a layered graph by sampling new waypoints and IK solutions. However, this method needs a long planning time to get a highly-accurate solution, even with lazy collision checking to reduce the computational overhead.

Even though these methods find a collision-free trajectory by utilizing a motion planning approach and random IK solutions, it is hard to get a highly-accurate solution due to time or structural constraints. To overcome these difficulties, our method incorporates the Jacobian-based IK solving method into our optimization process, instead of using only IK solutions. The aforementioned path-wise IK approaches also use the objective function to minimize the end-effector pose error, but do not combine it with the objective function to avoid external obstacles. On the other hand, our approach holistically integrates these different objectives and constraints within an optimization framework, inspired by an optimization-based motion planning approach, CHOMP [16], and effectively computes refined trajectories based on our two-stage gradient descent method.

2.2.3 Constrained Motion Planning

A constrained motion planning (CMP) additionally considers kinematic or dynamic constraints in addition to finding a collision-free trajectory [17, 35–37]. In the case of the kinematic constraint, e.g., pulling a drawer or opening a door [29], the CMP plans the trajectory while limiting movement to specific transitional or rotational axes. Our problem is similar to having the kinematic constraint on all transitional and rotational axes to match a given end-effector path. However, most CMP planners do not have a structure to solve the problem, especially sampling-based planners [35, 37].

Although an optimization-based motion planner can solve the problem by adding the kinematic constraints to their optimizer with finely divided end-effector poses in a similar manner of the path-wise IK approaches [12, 13], it is difficult to deal with diverse constraints and objectives, since it tends to fall into a local minimum. For handling this problem, we present a two-stage gradient descent to refine a trajectory to be feasible and accurate with a given end-effector path, and an adaptive exploration to escape local minima efficiently.

2.2.4 Gradient descent for a constrained problem

Gradient descent is a first-order iterative method of gradually finding a parameter value to minimize an objective function. While an unconstrained problem only focuses on minimizing a given objective function, a constrained problem is important to find a parameter value that minimizes the objective function in a feasible set [38].

For solving a constrained problem, the projected gradient descent (PGD) conducts a projection onto a feasible set after minimizing a primary objective function. Another method is conditional gradient descent (CGD), as known as the Frank-Wolfe algorithm [39], which finds a good feasible direction using a local linear approximation of the objective function and then moves along the chosen direction in each step.

Based on these algorithms, various works solving constrained problems extend and apply to fit their applications, e.g., image reconstruction and spike estimation [40–42]. We approach our path-wise problem as a constrained problem that aims to reduce the error with the target end-effector poses and puts collision and joint speed limits as constraints. We aim to minimize the error with the poses and solve the problem through our two-stage gradient descent based on the PGD concept.

2.2.5 Avoiding local minima for trajectory optimization

Optimization-based approaches usually synthesize the desired trajectory quickly but have a problem of falling into local minima. To escape local minima, STOMP [43] conducts a stochastic optimization, and CHOMP [16] applies the simulated annealing [44], one of the meta-heuristic algorithms, for its optimization. Recently, [45] use the Beetle Antennae Search algorithm [46] to improve the robustness of their trajectory optimization.

Another way to avoid local minima is to restart by changing parameter values or initial trajectory [11, 47, 48]. Similarly, we iteratively explore newly created trajectories to avoid local minima. Furthermore, we accelerate the exploration by deciding the restart based on the observation of past optimization results.

2.3 Background

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

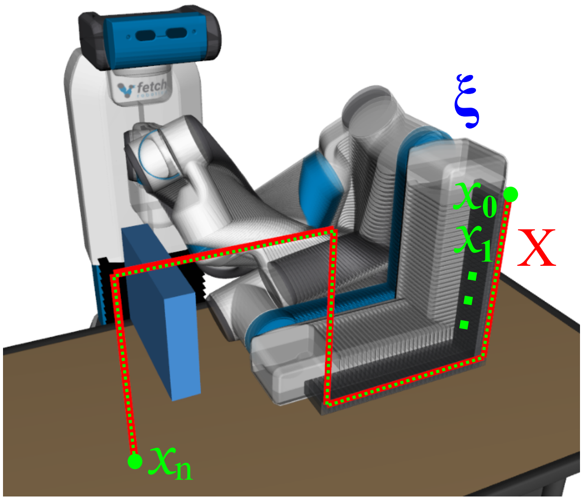


Figure 2.2: This figure shows our problem that is synthesizing a feasible and accurate trajectory $\boldsymbol{\xi}$ for a given end-effector path X. The red line is X, which is approximated by end-effector poses $\widetilde{X} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ (green dots). The trajectory $\boldsymbol{\xi}$ is computed at end-effector poses $\widetilde{X}$. The part of the synthesized trajectory shows that the end-effector follows the red line, and the sequence of joint configurations is smooth and collision-free, while avoiding obstacles such as the blue box and the table.

2.3.1 Problem Definition

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

Our target robot is a redundant manipulator that has multiple joint configurations given an end-effector pose; if the manipulator has a controllable degree of freedom (DoF) greater than six, it has infinitely many valid solutions. Among many candidates, we synthesize a set of joint configurations as a trajectory to follow the given end-effector path accurately, while avoiding collisions for external obstacles and the robot itself (Figure 2.2).

We solve the problem by applying the waypoint parameterization [49] of the path that finely splits the given end-effector path, $X \approx \widetilde{X} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$; $\boldsymbol{x} \in \mathbb{R}^6$ is an end-effector pose. We then compute the joint configurations for end-effector poses $\widetilde{X}$. As a result, the trajectory is approximated as: $\boldsymbol{\xi} \approx [\boldsymbol{q}_0 \;\; \boldsymbol{q}_1 \;\; \dots \;\; \boldsymbol{q}_n]^T \subset \mathbb{R}^{(n+1)\times d}$, where d is the DoF of a manipulator. When a start configuration $\boldsymbol{q}_0$ is given, we compute the joint configurations from its next configuration, $\boldsymbol{q}_1$, to the goal configuration $\boldsymbol{q}_n$. In our problem, however, the start configuration may not be given, while an end-effector path is given.

To solve our path-wise problem, we present a trajectory optimization of a redundant manipulator (TORM) that holistically integrates the Jacobian-based IK method and an optimization-based motion planning approach. Main notations are summarized in Table 2.1.

2.3.2 Jacobian-based Inverse Kinematics

To match a given target end-effector path, our optimizer is based on the Jacobian-based IK method. Many prior techniques explained in Sec. 2.2.1 utilize the Jacobian-based IK method, since it is very accurate with fast convergence [11]. In this work, we combine it with an optimization-based approach to quickly get a highly-accurate trajectory, while avoiding collisions with obstacles and achieving smoothness of joints.

Table 2.1: Notation table

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

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

$$F_{pose}(\boldsymbol{\xi}) = \frac{1}{2}\sum_{i=0}^{n} \|\boldsymbol{x}_i - FK(\boldsymbol{q}_i)\|^2, \tag{2.1}$$

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

$$\nabla F_{pose}(\boldsymbol{q}_i) = \boldsymbol{J}^T(\boldsymbol{x}_i - FK(\boldsymbol{q}_i)), \tag{2.2}$$

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

2.3.3 Optimization-based Motion Planning

For making a feasible trajectory, we adopt an optimization-based motion planning approach, specifically CHOMP [16], which synthesizes a smooth and collision-free trajectory based on the covariant gradient descent. This approach significantly reduces the planning time by incorporating an efficient collision avoidance method into its optimization process. By incorporating an optimization-based motion planning method with these advantages into our method, we can quickly get the desired solution, while avoiding collisions against external obstacles and the robot itself.

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

$$U(\boldsymbol{\xi}) = F_{obs}(\boldsymbol{\xi}) + \lambda F_{smooth}(\boldsymbol{\xi}), \tag{2.3}$$

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

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

$$F_{obs}(\boldsymbol{\xi}) = \sum_{i=0}^{n-1}\sum_{p}^{P}\left(\frac{1}{2}\Big(c(\boldsymbol{x}_{i+1,p}) + c(\boldsymbol{x}_{i,p})\Big)\left\|\boldsymbol{x}_{i+1,p} - \boldsymbol{x}_{i,p}\right\|\right), \tag{2.4}$$

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

$$\nabla F_{obs}(\boldsymbol{q}_i) = \sum_{p}^{P} \boldsymbol{J}_p^T \left(\|\boldsymbol{x}_{i,p}^{'}\| [(\boldsymbol{I} - \hat{\boldsymbol{x}}_{i,p}^{'} \hat{\boldsymbol{x}}_{i,p}^{'T}) \nabla c(\boldsymbol{x}_{i,p}) - c(\boldsymbol{x}_{i,p}) \boldsymbol{\kappa}] \right), \tag{2.5}$$

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

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

$$F_{smooth}(\boldsymbol{\xi}) = \frac{1}{2} \sum_{i=0}^{n-1} \left\| \frac{\boldsymbol{q}_{i+1} - \boldsymbol{q}_i}{\Delta t} \right\|^2. \tag{2.6}$$

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

2.4 Trajectory Optimization

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

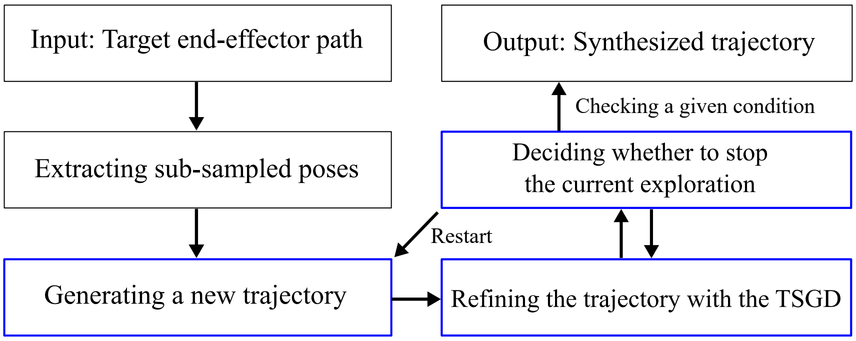

Input: Target end-effector path
Output: Synthesized trajectory
Checking a given condition
Extracting sub-sampled poses
Deciding whether to stop the current exploration
Restart
Generating a new trajectory
Refining the trajectory with the TSGD

Figure 2.3: This figure shows an abstraction of our overall algorithm. The blue boxes represent the iterative process, and one exploration means the refinement process starting from a newly generated trajectory.

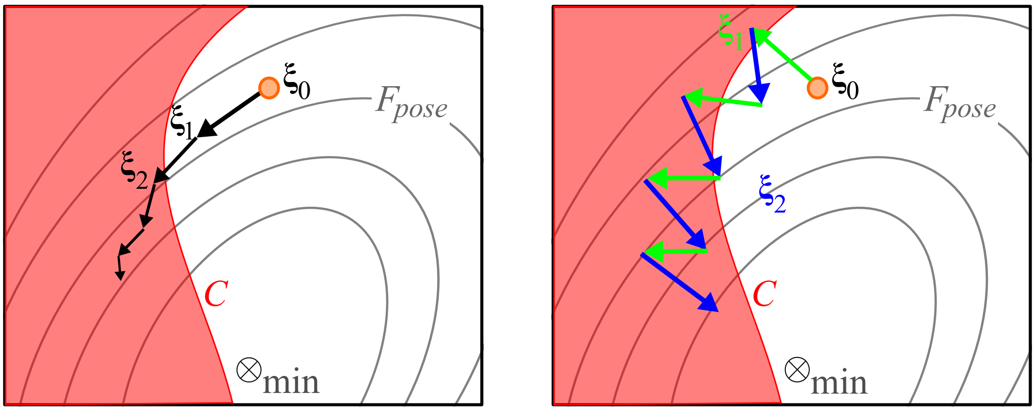
2.4.1 Motivation and Overview

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

$$U(\boldsymbol{\xi}) = F_{pose}(\boldsymbol{\xi}) + \lambda_1 F_{obs}(\boldsymbol{\xi}) + \lambda_2 F_{smooth}(\boldsymbol{\xi}). \tag{2.7}$$

We, however, found that a naïve approach cannot get a highly-accurate solution due to conflicts among each functional gradient when updating a trajectory (Figure 2.4a).

Additionally, this optimization-based update method has a probability of falling into local minima due to its local nature. One way of escaping local minima is to perform a fixed number of local updates by repeatedly



(a) Simple integration

(b) TSGD

Figure 2.4: This shows the illustration of optimization progress for a simple integration using Eq. 2.7, and our TSGD (Eq. 2.9). Each gray contour line represents equal F_{pose}, and the center, marked ⊗, of innermost contour has the smallest F_{pose}. The simple integration (a) is difficult to reach the minimum F_{pose} due to conflicts between three different terms; black arrows represent the update process of the simple integration through the sum of three functional gradients. On the other hand, our TSGD approaches to the minimum F_{pose} through alternatively updating the trajectory toward the feasible set C shown in the red region using ∇F_{obs} and ∇F_{smooth} (green arrow) and updating the trajectory toward the minimum F_{pose} using ∇F_{pose} (blue arrow).

restarting from different values [47]. However, it is rather unclear how many updates we need to perform for our task.

To alleviate the aforementioned problems, we first present a two-stage gradient descent (TSGD) that optimizes a trajectory by dividing two parts: making a trajectory to be feasible and matching a given end-effector path. The TSGD mainly focuses on minimizing the pose error, while achieving the feasibility as satisfying various constraints (Sec. 2.4.2).

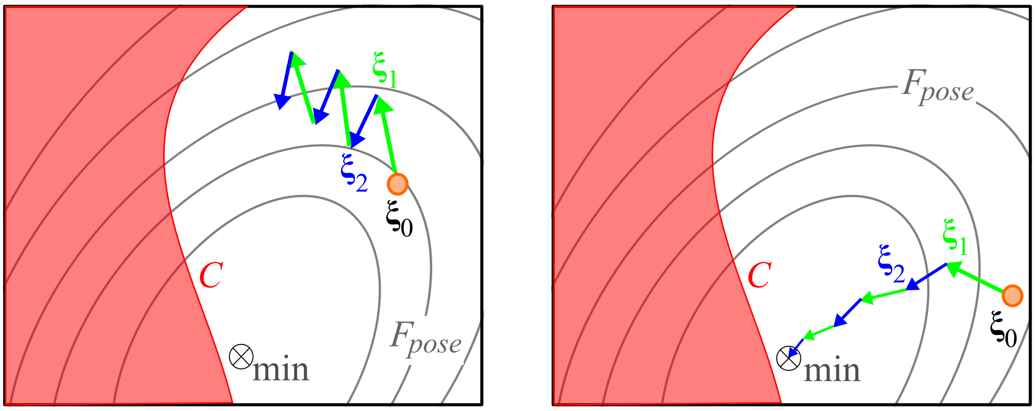
We also propose an adaptive exploration (AE) to avoid local minima efficiently. The AE explores different new trajectories with the TSGD and adaptively decides whether to stop the current exploration instead of repeating a fixed number of updates. The stop criterion is decided through the observation of optimization results (Sec. 2.4.3).

Overall, our algorithm iteratively performs two parts, generating a new trajectory, $\boldsymbol{\xi}_{new}$, (Sec. 2.4.4) and updating the trajectory (Figure 2.3). An initial trajectory is created in the same way as generating a new trajectory. The update part locally refines the trajectory using our TSGD, along with examining whether or not to restart the current exploration. During the iterative process, we find the trajectory that has the smallest F_{pose} with satisfying the feasibility (Sec. 2.4.5). This process continues until satisfying a given condition, e.g., running time or cost.

2.4.2 Two-Stage Gradient Descent

Our goal is to get a highly-accurate solution with satisfying the feasibility. To achieve the goal, we holistically integrate the Jacobian-based IK with an optimization-based motion planning approach. When simply combining their terms as shown in Eq. 2.7, we can get a solution through iterative refinement of a trajectory, but the computed trajectory tends to be sub-optimal, because it is updated from the weighted sum of different functional gradients computed for different, even worse conflicting, purposes (Figure 2.4).

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



(a) Case where F_{pose} is increasingly growing

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

Figure 2.5: This illustrates two problematic cases for ineffective exploration; the information of this figure can be seen in Figure 2.4. Note that these figures are examples of one exploration from a newly generated trajectory, ξ_0. (a) is the case where F_{pose} gradually increases due to the strong tendency to make a feasible trajectory. In contrast, (b) is the case where F_{pose} is gradually reduced, but it is still not feasible.

is defined as a constrained problem:

$$\underset{\boldsymbol{\xi}}{\operatorname{argmin}}\ F_{pose}(\boldsymbol{\xi}), \quad \text{subject to} \quad \boldsymbol{\xi} \in C, \tag{2.8}$$

where C is a feasible set.

Our TSGD solves Eq. 2.8 by iteratively performing two parts consisting of updating to make a feasible trajectory and updating the trajectory to match closer to the minimum point in terms of F_{pose}, marked as ⊗ (Figure 2.4b). We design our TSGD inspired by the projected gradient descent that conducts a projection onto a feasible set C after minimizing a primary objective function to solve a constrained problem. Instead of the projection, we update the trajectory to be feasible by minimizing F_{obs} and F_{smooth}. Since the feasible set C cannot be computed at once due to the complexity of the constraints [16, 17, 36], we find a feasible trajectory by repeatedly minimizing F_{obs} and F_{smooth}.

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

$$\boldsymbol{\xi}_{i+1} = \begin{cases} \boldsymbol{\xi}_i - \eta_1 \boldsymbol{A}^{-1}(\nabla F_{obs} + \lambda \nabla F_{smooth}), & \text{if } i \text{ is odd,} \\ \boldsymbol{\xi}_i - \eta_2 \nabla F_{pose}, & \text{otherwise,} \end{cases} \tag{2.9}$$

where η is a learning rate, and $\boldsymbol{A}$ is from an equally transformed representation of the smooth term (see the bottom of Sec. 2.3.3). $\boldsymbol{A}^{-1}$ acts to retain smoothness and to accelerate the optimization by having a small amount of impact on the overall trajectory. ∇F_{obs} and ∇F_{smooth} using $\boldsymbol{A}^{-1}$ have a computational benefit by giving an influence between successive joint configurations, which is shown in [16, 50]. On the other hand, ∇F_{pose} does not apply $\boldsymbol{A}^{-1}$ to compute highly-accurate joint configurations matched for each finely divided end-effector pose.

Our TSGD needs a more number of iterations by performing two separate updates over the simple integration (Eq. 2.7). Nevertheless, we found that our method shows a faster convergence speed than the simple integration (Figure 2.9); in our experiment, our TSGD shows at least 100 times less pose error over the simple integration (Table 2.5). This is thanks to the TSGD, which effectively resolves conflicts between constraints.

2.4.3 Adaptive Exploration

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

To effectively avoid getting stuck in local minima, we suggest an adaptive exploration (AE) that can explore new, different trajectories, while examining whether to stop each exploration part or not. Note that a redundant manipulator can have multiple joint configurations at a single pose, thus we can construct many candidate trajectories. By utilizing this property, we generate new, different trajectories, which are locally refined based on our TSGD.

An iterative exploration, starting with new values, has been used to avoid local minima in works based on the gradient descent [11,47]. On the other hand, the iterative exploration has the drawback of computational waste by performing a fixed number of iterations for each exploration.

To improve the effectiveness of the exploration, we set a stopping criteria inspired by accelerated gradient methods [51, 52], which can dramatically improve the convergence rate through a heuristic restart scheme. Our AE restarts when it is likely that the exploration progress is ineffective, or makes little change in the recently updated trajectories as a local minimum.

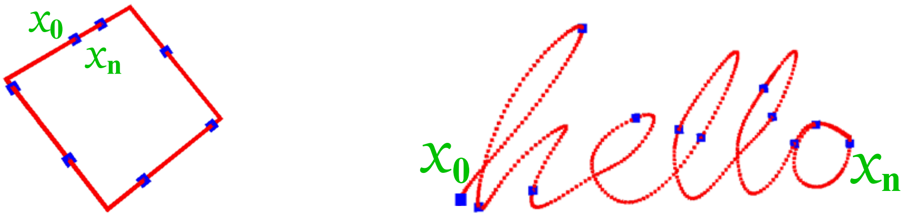
We have observed that our optimizer has poor exploration in typically two cases. One of the cases is that F_{pose} is increasing as the force to satisfy the constraints is strong (Figure 2.5a). The other case is that F_{pose} is gradually reduced, but any trajectories do not achieve the feasibility (Figure 2.5b). These cases may occur as one of the two stages has a greater impact on the optimization, but are more affected by an initial trajectory, $\boldsymbol{\xi}_0$. When the initial trajectory is not good enough, such as violating the constraints in many parts of the trajectory, finding the desired solution only with local updates is difficult. Such importance of initial values in optimization-based planning has been discussed in several works [11, 33]. Hence, we can accelerate our optimization by restarting from a new trajectory when the current exploration is judged to the aforementioned cases.

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

$$\beta = \frac{1}{m}\sum_{i=1}^{m}(F_{pose}^{u} - F_{pose}^{u-i})/i, \tag{2.10}$$

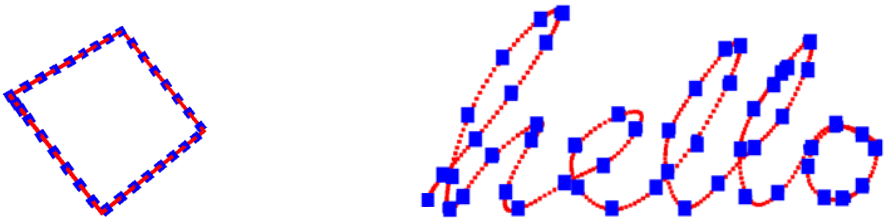
where u is the current iteration index; if $u < m$, we do not execute this process, and we set m to 5. Specifically, we first examine whether the current exploration falls into a local minimum by checking $|\beta| < 1 \times 10^{-6}$. Next, we examine whether the current exploration corresponds to two cases of ineffective exploration (Figure 2.5). The first case (Figure 2.5a) where F_{pose} is gradually increasing is checked by $\beta > 0.1$. The second case (Figure 2.5b) where F_{pose} is enough reduced without the feasibility is checked by simultaneously testing $|\beta| < 1 \times 10^{-3}$ and examining whether the constraints are satisfied. Note that the exploration is continued if the aforementioned, three conditions are not satisfied.

We decide the restart from a new trajectory through a simple, yet effective restart scheme. As a result, our approach shows a faster convergence than performing an exploration by a fixed number of iterations (Table 2.5). This is mainly thanks to the adaptive iterations.



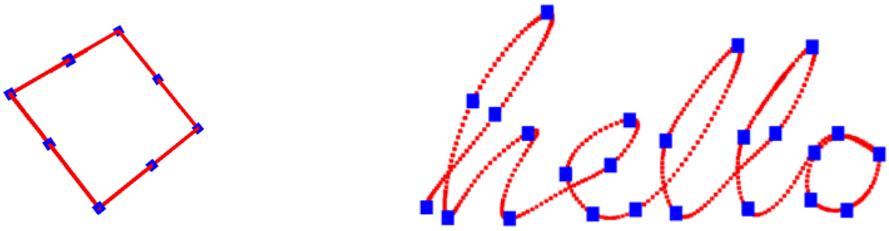
(a) 50 intervals (8)

(b) 50 intervals (13)



(c) 10 intervals (33)

(d) 10 intervals (57)



(e) Path simplification (9)

(f) Path simplification (25)

Figure 2.6: This figure shows sub-sampled poses S (blue dots), extracted from finely divided end-effector poses $\widetilde{X}$ (red dots), in the problem of square tracing and writing "hello". From S, we construct a new trajectory ξ_{new} that starts with random IK solutions and is found in a greedy manner minimizing Eq. 2.7. (a,d) and (b,e) are extracted uniformly at 10 and 50, respectively. (c) and (f) are the results computed by the path simplification using the DP algorithm. () represents the number of extracted sub-sampled poses as blue dots. (b) has too many S, increasing the time to generate a new trajectory $\boldsymbol{\xi}_{new}$. On the other hand, (d) has a small number of S, which is not enough to make an potentially good trajectory. On the other hand, we apply the path simplification to extract the appropriate S (c,f) regardless of various forms of paths.

2.4.4 New Trajectory Generation

Each exploration part generates a new trajectory $\boldsymbol{\xi}_{new}$. We strive to find a potentially good trajectory for our local optimization, which considers our objectives with smoothness, avoiding obstacles, and following a given path. Because merely connecting start and goal configurations can result in a sub-optimal solution, especially in cases of complex scenarios (e.g., Figure 2.6 and environments with external obstacles) [22].

Overall, we consider random configurations and choose one that minimizes three different terms (Eq. 2.7) in a greedy manner for generating an initial trajectory. As the first step, we extract sub-sampled poses, S, from the end-effector poses $\widetilde{X}$, since working with more poses tends to increase the complexity of generating a trajectory. We can uniformly extract S at γ intervals from $\widetilde{X}$, but it is challenging to set γ for effectively handling various end-effector paths (Figure 2.6). For example, when the interval is small, the generation time increases. On the other hand, when the interval is large, the quality of the path may deteriorate.

To handle this trade-off problem, we extract the most important poses that can well maintain the shape of a given path. To compute such important poses, we apply the path simplification method, specifically Douglas-

Peucker (DP) algorithm [53]. In our method, the DP recursively extracts $\widetilde{X}$ finding the furthest pose between two poses that are initially start and end poses. This recursive process stops when the furthest pose is closer than a given distance.

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

In the next step, we find suitable joint configurations at the sub-sampled poses. For the first, sub-sampled pose as the start end-effector pose $\boldsymbol{x}_0$, we simply compute a random IK solution at the pose if a start configuration is not given. For its next pose, we compute j random IK solutions at the pose and greedily select one that minimizes Eq. 2.7 when connecting it with the configuration of the previous pose; we set j to 100. Lastly, we connect chosen joint configurations through linear interpolation for generating a new trajectory, which is then locally refined by our TSGD.

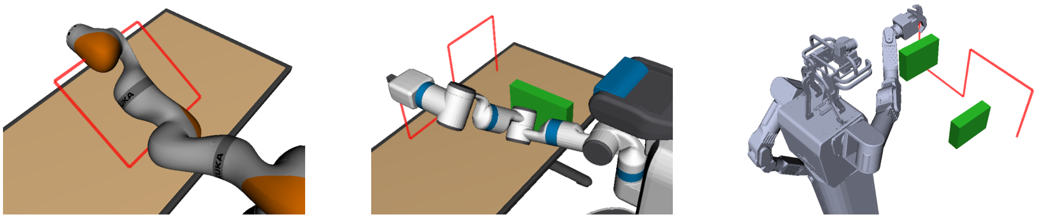
Our trajectory generation method sometimes generates a poor quality of trajectory due to its random nature. Even though we can reduce the randomness by increasing the number of sub-sampled poses with many IK solutions, it exponentially increases the generation time. We, therefore, extract sub-sampled poses at appropriate intervals using the path simplification to maintain a balance between generation time and quality of trajectory. Furthermore, we compensate for the randomness of the generated trajectory by exploring various trajectories through adaptive exploration (Sec. 2.4.3).

2.4.5 Trajectory Constraints

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

In addition to the collision checking, a manipulator commonly has several constraints that must satisfy lower and upper limits of joints, velocity limits, and kinematic singularities for joints. In the case of lower and upper limits of joints, it is traditionally handled by performing L_1 projection that resets the violating joints value to its limit value. To retain smoothness, we use a smooth projection used by CHOMP [16] during the update process. The smooth projection uses the Riemannian metric $\boldsymbol{A}^{-1}$. In other words, an updated trajectory, $\tilde{\boldsymbol{\xi}}$, is defined as $\tilde{\boldsymbol{\xi}} = \boldsymbol{\xi} + \alpha \boldsymbol{A}^{-1}\boldsymbol{v}$, where $\boldsymbol{v}$ is the vector of joint values, and α is a scale constant. This process is repeated until there is no violation.

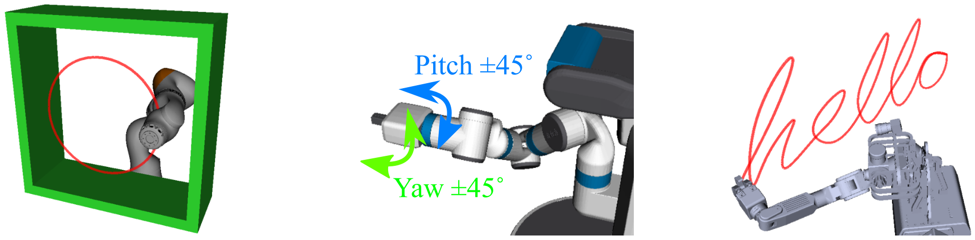
For other constraints like the velocity limit, we check them together while checking collisions. The velocity limit for joints is checked by computing the velocities of joints between $\boldsymbol{q}_{i-1}$ and $\boldsymbol{q}_i$ for a given time interval, Δt. Another constraint is the kinematic singularity that is a point where the robot is unstable, and it can occur when the Jacobian matrix loses full rank. To check kinematic singularities, we use the manipulability measure [54] used by RelaxedIK [12]. At a high level, it avoids making the manipulability measure less than a certain value that is computed by random samples. Note that our method returns the lowest cost trajectory guaranteed through checking constraints for constructing a feasible trajectory.



(a) Square tracing with the table

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

(c) "S" tracing with two boxes



(d) Circle tracing within a box

(e) Rotation task

(f) Writing "hello"

Figure 2.7: This shows six problems with three different robots, Kuka 7-DoF, Fetch 7-DoF, and Hubo 8-DoF. Four problems, (a) to (d), include external obstacles and two problems, (e) and (f), do not have external obstacles. The red line represents the given end-effector path.

2.5 Experiments and Analysis

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

We report the average performance by performing 20 tests with a machine that has 3.6 GHz Intel i7-9700 CPU and 32 GB RAM. In this experiment, we compute the pose error with target end-effector poses using a weighted sum of the Euclidean distances of the translational and rotational parts, which was used in the work of [14]; the adopted weight of the rotational distance over the translational distance is 0.17. In addition, we construct the target end-effector poses for calculating the pose error by adding one more between the poses used for planning. This is because the error calculation using only poses used for planning may not be precise in determining the concordance rate with a given end-effector path, since our target robot, a redundant manipulator, has various joint configurations for one pose.

2.5.1 Experiment setting

We prepare six problems (Figure 2.7) with external obstacles and two non-obstacle problems to evaluate and compare our method with prior methods, CHOMP [16], Trajopt [17], RelaxedIK [12], Stampede [13], and the work of [14]. In this section, we call the work of [14] EIGS, taken from their paper title.

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

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

Table 2.2: Results of different methods, CHOMP, Trajopt, EIGS, and ours in four problems including external obstacles. The bold text indicates the smallest pose error, and the underline indicates the second smallest pose error.

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

		Kuka 7-DoF					Fetch 7-DoF					Hubo 8-DoF				
		PE / SD	TL	NF	IST	PT	PE / SD	TL	NF	IST	PT	PE / SD	TL	NF	IST	PT
Square tracing with the table (fixed $\boldsymbol{q}_0$)	CHOMP	2.25e-2 / 7.9e-3	9.4	0	0.4	50	2.17e-2 / 3.3e-3	8.5	0	**0.6**	50	2.82e-2 / 1.3e-2	11.0	0	5.5	50
	Trajopt	3.33e-3 / 8.0e-3	7.9	0	3.4		1.46e-2 / 2.7e-3	10.3	1	4.5		7.80e-3 / 5.6e-3	11.1	5	7.3	
	EIGS	6.39e-3 / 1.3e-3	24.1	0	5.2		5.11e-3 / 1.2e-3	25.6	0	3.5		6.90e-3 / 1.1e-3	36.6	0	6.0	
	Ours	**6.37e-6** / 1.3e-6	7.9	0	**0.7**		**6.33e-6** / 1.0e-6	8.3	0	0.8		**1.51e-5** / 6.9e-6	10.8	0	**3.1**	
"S" tracing with the table and one box (fixed $\boldsymbol{q}_0$)	CHOMP	5.94e-3 / 8.3e-4	6.0	0	4.7	50	1.53e-2 / 2.7e-3	7.1	0	7.9	50	7.2e-2 / 4.6e-2	13.5	0	10.1	50
	Trajopt	4.61e-6 / 7.6e-6	5.9	0	3.3		5.70e-2 / 5.8e-2	8.7	2	3.8		9.05e-3 / 1.0e-2	10.0	7	8.3	
	EIGS	5.50e-3 / 8.1e-4	25.1	0	3.6		4.51e-3 / 7.2e-4	21.6	0	2.9		8.50e-3 / 1.8e-3	29.2	0	5.2	
	Ours	**1.72e-6** / 6.4e-7	6.0	0	**1.1**		**6.00e-6** / 2.1e-6	7.0	0	**1.2**		**6.85e-5** / 4.7e-5	11.7	0	**2.6**	
"S" tracing with the table and one box	CHOMP	3.66e-2 / 4.2e-2	7.6	0	4.3	50	7.52e-3 / 1.3e-3	6.1	0	6.8	50	2.31e-2 / 1.4e-2	9.8	1	6.9	50
	Trajopt	6.69e-6 / 5.3e-6	7.1	3	3.2		7.30e-6 / 3.9e-6	6.7	7	3.8		1.90e-3 / 4.9e-3	8.3	9	**3.6**	
	EIGS	4.64e-3 / 6.1e-4	24.0	0	3.9		5.02e-3 / 1.1e-3	18.8	0	3.2		8.34e-3 / 1.8e-3	31.4	0	6.2	
	Ours	**1.80e-6** / 1.9e-6	5.6	0	**1.2**		**1.35e-6** / 1.3e-6	5.7	0	**1.4**		**1.85e-5** / 1.6e-5	8.4	0	3.7	
"S" tracing with two boxes (fixed $\boldsymbol{q}_0$)	CHOMP	1.53e-2 / 3.4e-3	8.4	1	5.0	50	3.22e-2 / 1.3e-3	11.0	0	2.3	50	4.29e-2 / 2.0e-2	11.2	2	11.8	50
	Trajopt	2.21e-2 / 2.6e-2	9.4	4	**1.2**		8.54e-5 / 9.1e-5	10.8	6	2.9		5.16e-3 / 6.7e-3	10.2	9	**3.4**	
	EIGS	1.70e-2 / 1.0e-3	33.9	0	3.7		8.62e-3 / 5.4e-3	25.5	0	3.0		9.68e-3 / 1.2e-3	26.5	0	5.1	
	Ours	**9.26e-6** / 2.9e-6	8.2	0	1.4		**1.36e-5** / 2.6e-6	9.7	0	**1.8**		**1.12e-4** / 7.1e-4	10.2	0	4.9	
Circle tracing with surrounding obstacles (fixed $\boldsymbol{q}_0$)	CHOMP	1.83e-2 / 7.4e-4	7.8	0	**0.6**	50	1.68e-2 / 7.6e-4	7.0	0	2.4	50	4.77e-2 / 2.5e-2	8.5	1	7.4	50
	Trajopt	3.54e-5 / 4.5e-5	7.5	0	3.4		2.26e-2 / 2.4e-2	8.7	8	1.8		1.15e-2 / 1.3e-2	7.8	6	15.7	
	EIGS	4.72e-3 / 1.0e-3	24.9	0	3.9		5.40e-3 / 1.2e-3	16.9	0	2.7		1.06e-2 / 4.2e-3	24.2	0	**6.3**	
	Ours	**7.20e-6** / 4.5e-7	7.8	0	0.7		**7.27e-6** / 1.5e-6	7.5	0	**1.3**		**1.10e-4** / 3.8e-4	7.8	0	**6.3**	
Circle tracing with surrounding obstacles	CHOMP	2.91e-2 / 1.5e-2	7.8	0	**0.6**	50	1.16e-2 / 3.0e-3	6.9	0	2.0	50	2.67e-2 / 2.0e-2	8.7	3	9.4	50
	Trajopt	8.53e-6 / 6.3e-6	7.7	3	4.3		2.89e-5 / 5.1e-5	6.9	9	2.3		1.91e-3 / 6.0e-3	7.9	8	5.7	
	EIGS	5.34e-3 / 6.5e-4	23.1	0	4.2		6.05e-3 / 9.9e-4	14.8	0	2.9		6.93e-3 / 1.4e-3	19.4	0	5.9	
	Ours	**4.96e-6** / 3.2e-6	7.4	0	0.8		**2.30e-6** / 1.3e-6	6.8	0	**0.8**		**1.13e-5** / 1.3e-5	8.0	0	**5.5**	

We used the code of RelaxedIK and Stampede that are provided by authors through their websites. For RelaxedIK, Stampede, and our method, the end-effector paths of all problems are finely divided and are fed into all the tested methods; in our experiment, the distance between two divided end-effector poses is about $0.005m$ for translation and $0.02rad$ for rotation, following the protocol adopted in [13]. On the other hand, EIGS initially splits the end-effector path and gradually breaks down the path during the planning.

In CHOMP and Trajopt, we modified their setting for solving the existing constrained motion planning problem (Sec. 2.2.3) to solve the path-wise problem. Their setting considers various kinds of kinematic constraints, and we gave the kinematic constraint on all transitional and rotational axes with the finely divided end-effector poses. Also, we set CHOMP and Trajopt to find a trajectory that minimizes pose error while satisfying the constraints as in our method (Sec. 2.4.5). In addition, we gave an initial trajectory computed by our new trajectory generation method (Sec. 2.4.4), since both planners cannot solve our problems mostly without the initial trajectory.

We mainly evaluate whether the synthesized trajectory accurately follows the given end-effector path. Note that reported results were extracted from feasible trajectories satisfying the given constraints (Sec. 2.4.5).

Table 2.3: Results of different methods, CHOMP, Trajopt, EIGS, RelaxedIK, Stampede, and ours in two non-obstacle problems. RelaxedIk is a real-time planner and does not provide better trajectories with more planning time; we provide its results while it cannot be compared in the equal-time comparison. RelaxedIK and Stampede experimented with non-obstacle problems since these methods only check self-collision. The bold text indicates the smallest pose error, and the underline indicates the second smallest pose error.

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

		Kuka 7-DoF					Fetch 7-DoF					Hubo 8-DoF				
		PE / SD	TL	NF	IST	PT	PE / SD	TL	NF	IST	PT	PE / SD	TL	NF	IST	PT
Rotation task without obstacles	RelaxedIK	3.93e-2 / 3.7e-3	8.9	0	-	6.3	7.31e-2 / 4.3e-3	12.1	0	-	6.1	8.29e-2 / 6.6e-3	8.9	0	-	7.3
	Stampede	5.85e-5 / 1.8e-6	9.5	0	28.5	28.5	**6.49e-5** / 1.9e-6	12.4	0	35.7	35.7	2.58e-4 / 8.2e-5	9.8	0	38.0	38.0
	CHOMP	7.33e-3 / 3.3e-3	8.7	0	**0.3**		2.51e-2 / 7.8e-3	14.3	0	**0.5**		2.83e-2 / 1.1e-2	11.7	0	**1.2**	
	Trajopt	2.85e-3 / 3.7e-3	6.6	0	1.1		5.85e-4 / 8.3e-4	12.6	0	2.0		1.49e-2 / 2.2e-2	10.7	4	6.3	
	EIGS	1.40e-2 / 4.2e-3	17.1	0	4.2		8.30e-3 / 1.5e-3	17.8	0	4.3		7.41e-3 / 1.6e-3	19.2	0	5.6	
	Ours	**5.53e-5** / 3.3e-5	8.9	0	0.5		6.96e-5 / 3.8e-6	13.3	0	0.6		**2.21e-4** / 1.3e-4	10.5	0	1.5	
Writing “hello” without obstacles	RelaxedIK	4.10e-2 / 2.1e-3	19.2	0	-	16.7	5.04e-2 / 4.6e-3	21.6	0	-	16.0	5.29e-2 / 8.7e-3	27.4	0	-	17.2
	Stampede	3.28e-5 / 4.7e-5	20.3	0	30.4	30.4	4.71e-5 / 6.6e-5	24.8	0	38.3	38.3	**9.10e-5** / 2.0e-4	29.7	0	45.1	45.1
	CHOMP	1.27e-1 / 2.8e-2	36.0	0	**1.1**		3.76e-2 / 3.7e-3	26.3	0	**1.2**		6.43e-2 / 3.5e-2	32.5	0	**4.5**	
	Trajopt	2.42e-3 / 2.6e-2	25.5	0	5.5		1.54e-3 / 5.9e-2	28.4	2	6.9		1.71e-2 / 6.0e-3	31.1	4	7.1	
	EIGS	2.23e-2 / 3.8e-3	49.2	0	11.7		1.7e-2 / 2.1e-3	36.9	0	11.4		1.7e-2 / 1.9e-3	49.0	0	13.2	
	Ours	**3.13e-5** / 1.0e-5	22.5	0	1.5		**4.36e-5** / 3.6e-6	26.0	0	1.3		3.3e-4 / 5.3e-4	31.8	0	5.2	

2.5.2 Comparison with prior methods

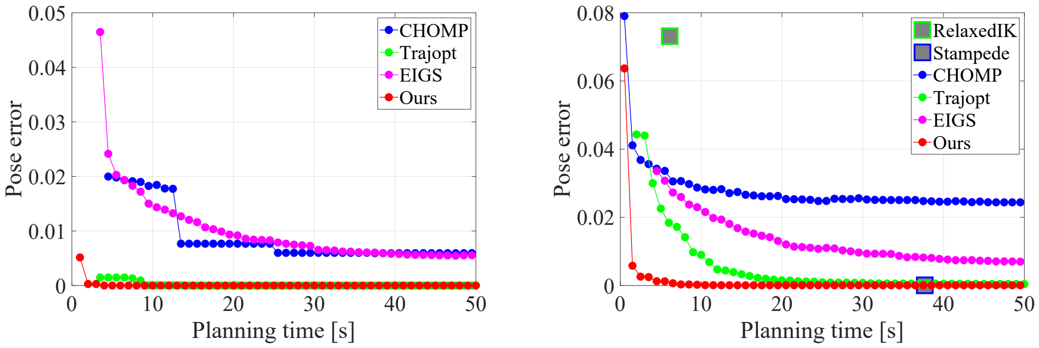
In four problems including external obstacles, we fix the start configuration $\boldsymbol{q}_0$ located close to the obstacles in order to see how well different methods can avoid external obstacles (Figure 2.7). For non-obstacle problems and two obstacle problems, we do not fix $\boldsymbol{q}_0$ to see how different methods handle $\boldsymbol{q}_0$ during the planning process.

Table 2.2 and Table 2.3 show the overall results of different methods, including three different robots. Figure 2.8 shows the trajectory quality of different methods, as we have more planning time up to the maximum planning budget of 50 seconds. On the other hand, Table 2.3 shows a snapshot result of the trajectory computed at a specific planning time.

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

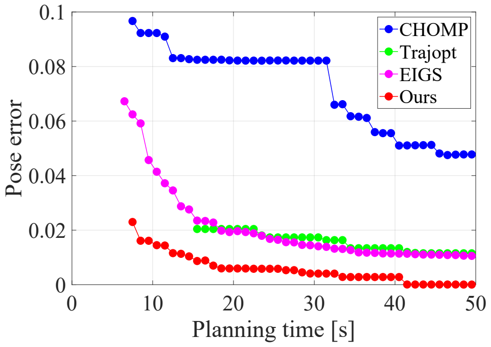
CHOMP, Trajopt, and our method find an initial solution faster than other methods in most problems (Table 2.2). This is because these techniques set an initial trajectory using our new trajectory generation method. It also supports that our new trajectory generation method quickly synthesizes a potentially good trajectory (Sec. 2.4.4).

Nonetheless, CHOMP shows the highest pose error in most problems, even though it gradually reduces the pose error over planning time. Our method is based on CHOMP, but we achieved a highly-accurate solution thanks to the update through our TSGD by combining the Jacobian-based IK approach and CHOMP. In addition, Table 2.4 shows the computational advantage of our TSGD combined with the Jacobian-based IK approach compared to CHOMP considering various kinds of kinematic constraints; in CHOMP, multiple matrix calculations are performed to take into account various kinematic constraints (see [16]).



(a) Square tracing using the Kuka 7-DoF

(b) Rotation task using the Fetch 7-DoF



(c) Circle tracing using the Hubo 8-DoF

Figure 2.8: This shows the pose error over the planning time of different methods in three different problems. Since (a) and (c) include external obstacles, RelaxedIK and Stampede are excluded from the experiments. Also, the start configurations of (a) and (c) are fixed. We visualize graphs once an initial solution is computed.

Trajopt has the highest number of failures due to falling into a local minimum, even though it had good results in some problems, e.g., “S” tracing with two boxes using Fetch 7-DoF. Especially, there are many failures in the two problems, “S” tracing with two boxes and circle tracing, surrounded by obstacles in the given end-effector path. This is because Trajopt uses the second-order method, so the update time is long, making it difficult to explore various trajectories during a given planning time. On the other hand, our method did not record failures by exploring new trajectories thanks to the very fast update of our TSGD based on gradient descent (Table 2.4).

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

EIGS refines a trajectory by progressively sampling new waypoints and IK solutions. Nonetheless, our method improves the quality of the trajectory over EIGS, as we have more planning time (Figure 2.8). This improvement is mainly because our method incorporates the IK solving method into the optimization process instead of simply using IK solutions.

In Table 2.2, EIGS shows the longest length of the computed trajectory on average for all problems; we measure the length of a trajectory using the Euclidean distance. EIGS does not consider the distance in C-space, since it only checks the similarity measure of two curves using the discrete Fréchet distance [14]. On the other

Table 2.4: Results of the number of iterations and explorations during 50 seconds for optimization-based approaches, CHOMP, Trajopt, and ours. Parenthesis indicates the time (s) for one update of its optimizer. We regard one update of Trajopt as one execution of the quadratic solver.

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

Table 2.5: Analysis of components of our proposed method by substituting them to alternative methods, the simple integration (SI), iterative exploration (IE), and extracting sub-sampled poses at 10 and 50 intervals. The last row shows the results of our complete method.

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

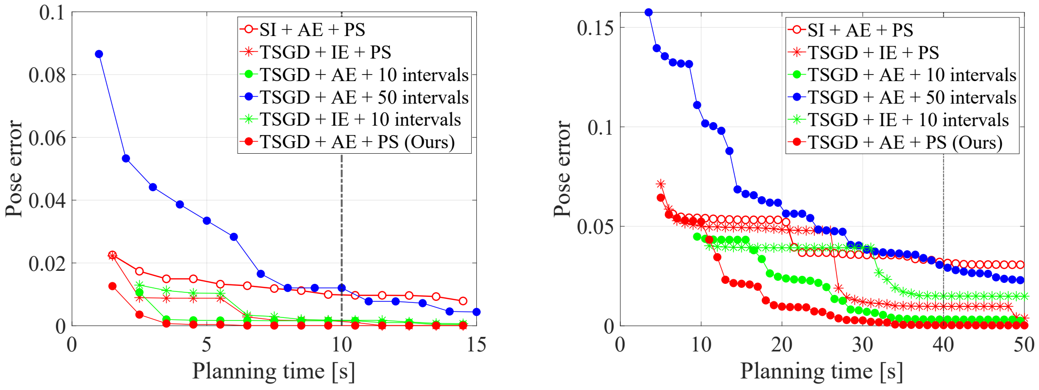
TL: Trajectory length (rad). IST: Initial solution time (s). PT: Planning time (s).

hand, other methods take into account the smoothness between joint configurations and thus generate shorter trajectories than EIGS.

Stampede is also a sampling-based approach like EIGS, but it generates a highly-accurate solution on average (Table 2.3). Stampede does not deal with external obstacles, but uses a neural network to check self-collision quickly. However, it takes a long time to get an initial solution, because it has to samples IK solutions at all the end-effector poses. These results show that Stampede and EIGS have different pros and cons.

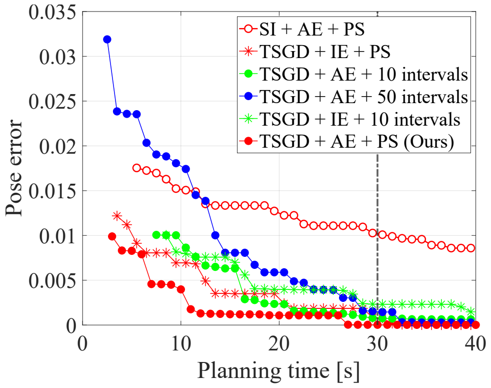
RelaxedIK is a real-time planner, thus it quickly finds a solution (Table 2.3). However, its accuracy is much lower than other methods. In conclusion, RelaxedIK shows real-time performance by quickly optimizing the joint configuration for one pose, but it is difficult to obtain a highly-accurate trajectory.

Overall, our method finds an initial solution quite quickly with a high pose error, but improves its quality as we have more planning time (Figure 2.8). Also, our method has a lower pose error with a lower standard deviation on average than other methods, as shown in Table 2.2 and Table 2.3. These results support that our optimization process efficiently reduces the pose error, while satisfying several constraints.



(a) Writing "hello" using the Fetch 7-DoF

(b) Writing "hello" using the Hubo 8-DoF



(c) Square tracing using the Hubo 8-DoF

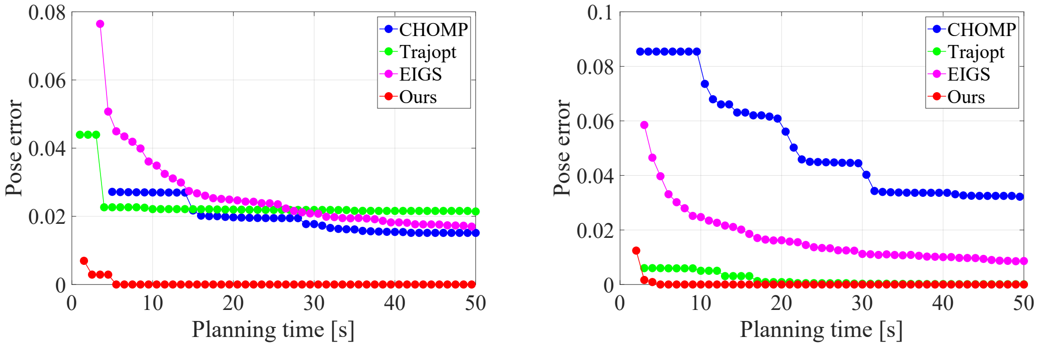
Figure 2.9: This shows the pose error as a function of the planning time of various ablated methods. Our complete method contains two-stage gradient descent (TSGD), adaptive exploration (AE), and path simplification (PS). We substitute them with the simple integration (SI), the iterative exploration (IE) with the fixed number of iterations, and extracting sub-sampled poses at 10 and 50 intervals. The gray dotted lines represent the results shown in Table 2.5.

2.5.3 Analysis of our proposed methods

To see the benefits of components of our proposed method, we conduct ablation study. We first substitute the two-stage gradient descents (TSGD) to the simple integration (SI) updating three functional gradients at once. Next, we test the iterative exploration (IE) iterated a fixed number of update instead of our adaptive exploration (AE). Finally, we compare different ways of extracting sub-sampled poses, 10 and 50 intervals, and the chosen path simplification (PS).

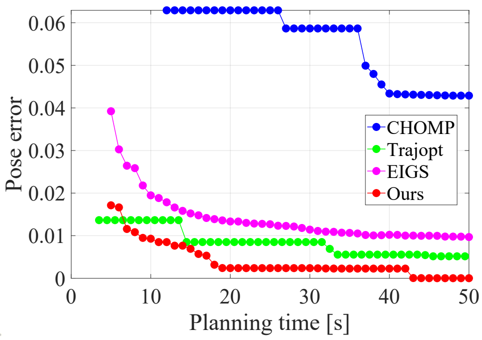
Table 2.5 shows the results of aforementioned methods for three different problems, writing "hello" using the Fetch 7-DoF and the Hubo 8-DoF, and square tracing using the Hubo 8-DoF; the start configuration of the square tracing is fixed. Figure 2.9 also shows the results, as we have more planning time up to the maximum planning budget of 50 seconds.

As shown in Figure 2.9a, it is hard to check the big difference between different methods due to its fast convergence when we use the Fetch 7-DoF. On the other hand, we can confirm the pros of proposed methods in experiments using the Hubo 8-DoF (Figure 2.9b and Figure 2.9c). We analyze the results for different configurations of robots in Sec. 2.5.4.



(a) Using Kuka 7-DoF

(b) Using Fetch 7-DoF



(c) Using Hubo 8-DoF

Figure 2.10: This shows the pose error over the planning time of different methods with three different robots in "S" tracing with two boxes.

Using the SI has a higher pose error on average than our full method, and it also has the highest min. pose error among 20 tests, compared to other tested methods (Table 2.5). These results demonstrate that the different functional gradients conflict with each other. Therefore, the TSGD prevents the competition of different functional gradients and is useful to get a highly-accurate solution. Note that the SI is different approach from CHOMP. The SI is the simple integration of F_{obs}, F_{smooth}, and F_{pose}, the objective of the Jacobian-based IK, while CHOMP is a reconfiguration of the original update rule to consider various kinds of kinematic constraints [16]; we set the kinematic constraints on all transitional and rotational axes to match a given end-effector path.

Our method shows the fast convergence rather than using the IE (see green and red lines with dots Figure 2.9). In Table 2.5, the max. pose errors of our method are also lower than using the IE at a specific planning time. These results indicate that the AE makes a good decision to stop the current exploration based on the observation of optimization results.

Extracting sub-sampled poses S at 50 intervals shows the fastest result of finding an initial solution in the problem of writing "hello", while 10 intervals is the slowest. On the other hand, extracting S at 10 intervals shows the better performance than 50 intervals. The time of generating a trajectory is reduced as having smaller sub-sampled poses, but we can get a low-quality path if sub-sampled poses are too few. To generate a trajectory, 10 intervals, 50 intervals, and the PS take 7.2, 3.8, and 2.1 seconds in the writing "hello" problem using the Hubo 8-DoF. Although the generation time of the PS is in the middle of 10 and 50 intervals, the PS shows the better performance than others. These results demonstrate that the PS strikes a good balance between the generation time and trajectory quality.

In summary, these results show that our proposed method synthesizes highly-accurate trajectories, while effectively getting out of local minima. Furthermore, we can see that the proposed method is more accelerated, about 43 times faster on average across all the tested cases, than the conference version of TORM (the green line with star dots in Figure 2.9).

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

2.5.4 Analysis with different robots

To analyze the results with different robots, we compare results in the problem of “S” tracing with two boxes by using three different robots. It is difficult to directly compare them even with the same problem, but we can see overall tendency of the results. Figure 2.10 shows the results of “S” tracing with two boxes by using different robots, Kuka 7-DoF, Fetch 7-DoF, and Hubo 8-DoF.

In Figure 2.10, our method quickly minimizes the pose error, as having more planning time. Within the given 50 seconds, we also get highly-accurate results across different problems (Table 2.2). In our method, nevertheless, using the Hubo 8-DoF shows a slower convergence speed than using the Fetch 7-DoF and Kuka 7-DoF. One of main reasons is an increase in DoF, which requires higher computational overhead of the overall method. Other methods also show that using the Hubo 8-DoF generally has worse performances compared to using other robots (Table 2.2).

In our method, the biggest difference arises due to the number of spheres approximating the robot model to get the collision costs effectively. The Hubo 8-DoF has 163 spheres, while the Fetch 7-DoF and Kuka 7-DoF have 41 and 27, respectively. As a result, the computation time of the Hubo 8-DoF is 120ms, and the computation time of the Kuka 7-DoF and the Fetch 7-DoF is approximately 20ms for one trajectory (Eq. 2.4) in the problem of “S” tracing with two boxes.

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

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

Chapter 3. RCIK: Real-Time Collision-Free Inverse Kinematics using a Collision-Cost Prediction Network

3.1 Introduction

Remote control of a robotic manipulator has been used to perform tasks on behalf of humans at special-purpose sites such as nuclear power plants [5] and hospitals during telesurgery [4]. Recent research on the remote control of robots has been expanding to a diverse set of environments, including the home environment [8]. Due to this broadening diversity, robots are now expected to handle more difficult challenges when avoiding various and dynamic obstacles, while also accurately following human commands.

A robotic manipulator is capable of human-like movement given its many joints, but it has various constraints, e.g., avoiding collisions with obstacles and kinematic singularities, and maintaining the continuity of joint configurations. Among the various constraints, collision avoidance from obstacles incurs significant computational overhead and is one of the most important issues in many prior approaches [12, 15] to achieve real-time performance when executing human commands.

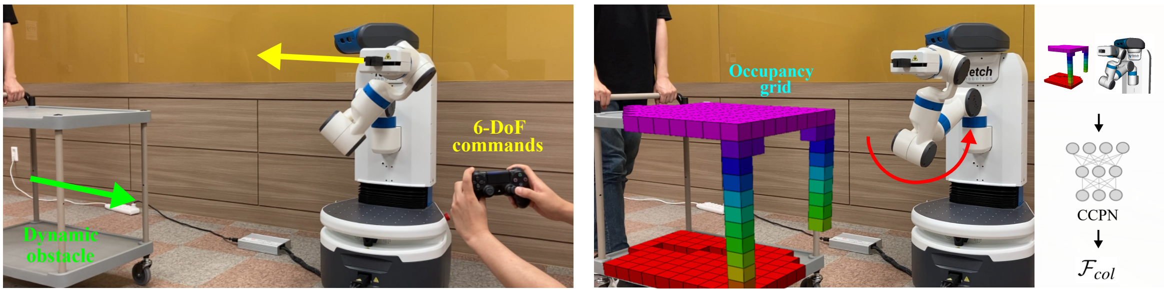
Recently, TORM [55] reduces the computational overhead required for external static obstacles by applying the collision avoidance method using the signed distance field (SDF) used in CHOMP [16]. However, the SDF is hard to deal with dynamic obstacles in real-time due to its long construction time.

Main Contributions. In this work, we present real-time collision-free inverse kinematics (RCIK) using a novel, collision-cost prediction network (CCPN) to reduce the massive computational overhead associated with collision avoidance. Our method performs consecutively given six-Degrees-of-Freedom (DoF) commands in environments, including dynamic obstacles (Figure 3.1). We generate IK candidates with high feasibility based on an optimization-based IK approach (Sec. 3.4.1). We then check whether generated IK candidates are satisfied with constraints, collision, joint velocity limits, and kinematic singularities. Among the IK candidates satisfying the constraints, we select the best configuration through our objective function, consisting of collision avoidance and the smoothness of joints to support consecutively given commands (Sec. 3.4.2). For handling dynamic obstacles, our network estimates a collision cost by grasping environment information from an occupancy grid updated from sensor data in real-time (Sec. 3.4.3).

To evaluate our method, we test three dynamic problems using a real robot, the Fetch manipulator, and four static problems for three different configurations of robots (Sec. 3.5.1). We observe that our method finds collision-free joint configurations in real-time in both static and dynamic environments, thanks to our deep neural network estimating collision costs quickly and accurately. We also analyze our method by comparing over the SDF (Sec. 3.5.2).

3.2 Related Work

In this section, we discuss prior studies in the areas of inverse kinematics and collision avoidance for manipulation planning.



(a) We send consecutive 6-DoF commands to a robot, and a moving cart obstructs the robot.

(b) The robot avoids the cart by computing F_{col} using our collision-cost prediction network (CCPN), while following consequtively given commmands.

Figure 3.1: These figures show the progress of our method in a dynamic environment. (a) Our method performs consecutively given 6-DoF commands while avoiding a dynamic obstacle, a cart in this case. (b) To avoid the cart, we compute a collision cost, F_{col}, using our collision-cost prediction network (CCPN). The CCPN uses an occupancy grid updated through sensor data to reflect changing environments immediately. Finally, our method finds a desired joint configuration that allows the robot to follow the given command accurately while avoiding obstacles. Our algorithm conducts all of these operations within 30ms for each command, mainly due to CCPN.

3.2.1 Inverse Kinematics

Traditional inverse kinematics (IK) has been widely studied to determine a joint configuration for a target end-effector pose quickly and successfully [24]. For example, IKFast [10] and the work of Sinha et al. [25] analytically solve equations of a complex kinematics chain to reduce the computation time. Additionally, Trac-IK [11] improves the success rate of the Jacobian-based IK method by restarting from a random joint configuration and applying sequential quadratic programming instead of the Jacobian. Nevertheless, these methods encounter difficulty when following a given sequence of end-effector poses, called *path-wise IK*, because there is no consideration of the continuity on generated joint configurations and constraints (e.g., collisions, joint velocity limits, and kinematic singularities).

Considering the aforementioned constraints, RelaxedIK [12] solves the path-wise IK by means of weighted-sum non-linear optimization. Based on RelaxedIK, Stampede [13] synthesizes an optimal trajectory from a discrete graph built in consideration of several constraints. These methods use a neural network for self-collision avoidance to reduce the computational overhead, but do not consider external obstacles.

To handle external obstacles, Holladay et al. [14, 22] suggest two methods based on optimization-based and sampling-based motion planning. These works apply the discrete Fréchet distance to approximate the similarity with a given end-effector path in the Cartesian space. Moreover, TORM [55] improves speed and accuracy by integrating the Jacobian-based IK method and an optimization-based motion planning approach.

Recently, CollisionIK [15] deals with static and dynamic obstacles in real-time by computing the distance between robot links and obstacles using the convex hull approach. On the other hand, it does not take into account difficulties of a real environment, such as sensor noise.

We also propose a real-time approach to solve path-wise IK problems in an environment surrounding static and dynamic obstacles, but we overcome the difficulties of a real environment using an occupancy grid and a deep neural network. An occupancy grid is updated in real-time while eliminating sensor noise based on a probabilistic update rule [56], and our deep neural network quickly predicts collision costs by grasping the environment information from the occupancy grid.

3.2.2 Collision Avoidance for Manipulation Planning

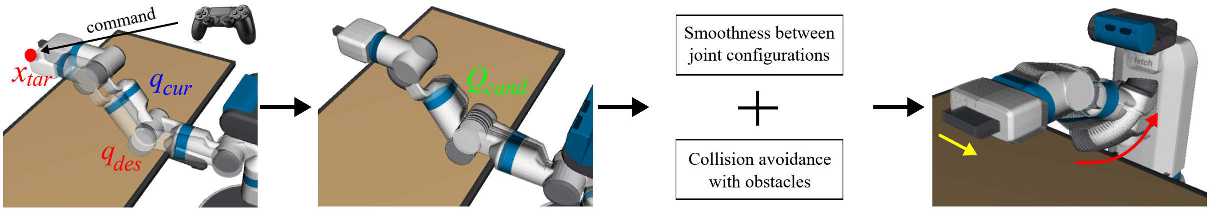
A SDF has been used in many manipulation planning methods, especially in relation to optimization-based approaches [16,55], because the SDF helps to avoid collision efficiently by computing a quantitative collision cost using the distance information from obstacles. On the other hand, it is difficult to use the SDF in dynamic environments due to its construction time. Although selective dynamic updating [57,58] and parallel processing [59,60] methods can reduce the construction time, constructing the SDF still incurs high computational overhead, especially when used to handle dynamic environments in real-time.

For a dynamic environment, several works have used simplified methods, such as approximating obstacles to spheres [61], using the closest distance from the obstacles [62]. Recently, Kew et al. [63] suggest a neural estimator for predicting the shortest distance. This estimator serves as a collision detector, improving the performance of a motion planner. On the other hand, this estimator can be used only within the learned objects because it is learned by changing only the positions of the objects.

Unlike the work of Kew et al. [63], our deep neural network reflects changing environments immediately without any restrictions on objects by grasping environment information from an occupancy grid updated through sensor data in real-time [56]. Moreover, our network is not used as a collision detector and is used to select a desired joint configuration away from obstacles among generated joint configurations for a target end-effector pose in our sampling-based approach.

3.3 Overview

In this section, we introduce the problem we aim to solve, followed by giving motivations of our work.


command
x_{tar}
q_{cur}
q_{des}
Q_{cand}
Smoothness between joint configurations
+
Collision avoidance with obstacles

(a) 6-DoF command (b) IK candidates (c) Cost function (d) Execution results

Figure 3.2: This shows an overview of our approach. (a) Our method synthesizes a desired joint configuration q_{des} that matches the target end-effector pose x_{tar} as a 6-DoF command and has the motion feasibility considering the current configuration q_{cur}. (b) First, we generate IK candidates Q_{cand} with high feasibility based on an optimization-based IK approach. (c) With checking constraints, such as collision and joint velocity limits, for Q_{cand}, we select the best one using our objective function for collision avoidance with obstacles and smoothness with previous joint configurations. (d) For 6-DoF commands given consecutively, denoted with the yellow arrow, our method sequentially synthesizes joint configurations that accurately realize the commands while avoiding collision and achieving smoothness, as indicated by the red arrow.

3.3.1 Problem Definition

Our goal is to develop a collision-free IK solver in real-time for accurately performing consecutively given 6-DoF commands in environments with static and dynamic obstacles. The command is provided as a target end-effector pose, $x_{tar} \subset \mathbb{R}^6$, and we deal with a redundant manipulator that can have many joint configurations for one end-effector pose x_{tar}; a redundant manipulator generally has a DoF value that exceeds 6. Among many solutions, we find a desired joint configuration, $q_{des} \subset \mathbb{R}^D$, where D is the DoF of the robot, considering consecutively given 6-DoF commands and various constraints.

We now describe our assumption for a given command and the criteria of finding q_{des}. First, we assume that a command given to a robot from a user is considering the robot specifications, a given time, and an environment. To perform the command accurately, our method finds q_{des} that accurately matches x_{tar}. When receiving an invalid command, e.g., collision due to moving obstacles and out of possible moving range, we stop the robot's movement for safety and wait for a valid command.

Next, our problem undertakes collision avoidance in static and dynamic environments and supports the continuity of synthesized joint configurations for consecutively given commands. To avoid a collision, we find q_{des} far away from obstacles among many joint configurations for x_{tar}, since we aim to follow the command exactly.

Lastly, q_{des} must satisfy various constraints, and we call it having *motion feasibility* in this chapter. The requirement of the *motion feasibility* include the following: the manipulator 1) must not collide with obstacles, 2) it does not violate the joint velocity limit in order to be able to move from the current configuration, q_{cur}, within the given time, t, and 3) it does not enter kinematic singularities, which occur when the Jacobian matrix loses the full rank.

In summary, we target to solve real-time inverse kinematics for consecutively given 6-DoF commands, while satisfying the requirement of motion feasibility in both static and dynamic environments.

3.3.2 Motivation

A real-time approach has to compute the desired solution during a given short time t, i.e., 30ms in our problem. Therefore, reducing the computational overhead is a critical issue for our problem. As explained in Sec. 3.2.1, many works have proposed methods to reduce the computational overhead of obstacle avoidance, which takes up a lot of computation. Among them, TORM [55] uses the signed distance field (SDF) to reduce the computational overhead for collision avoidance. TORM improves the trajectory optimization performance by quickly synthesizing an initial trajectory using IK samples generated by the traditional IK method.

Inspired by TORM, our method generates IK candidates, $Q_{cand} = \{q^1_{cand}, q^2_{cand}, \ldots\}$, for x_{tar} based on an optimization-based IK, one of the traditional-IK methods, and then selects the best q_{des} that has the motion feasibility and minimizes our objective function (Figure 3.2). To follow consecutive commands, our objective function considers collision avoidance of obstacles and the continuity of joint configurations.

Due to selecting one of generated IK candidates, increasing the possibility of having the motion feasibility of IK candidates is an important factor for improving the success rate of our approach. To obtain such IK candidates, especially satisfying the joint velocity limits, we utilize an optimization-based IK approach that has the property of obtaining a solution depending on the initial configuration [64, 65]. By maneuvering the initial configuration close to q_{cur}, we can generate IK candidates that have the potential to satisfy the joint velocity limits given the short time t.

The SDF can reduce the computational overhead of collision avoidance by calculating the distance information between obstacles in advance. The SDF also makes it possible to compute a quantitative cost, F_{col}, for collisions with obstacles. Despite these advantages, it is difficult to apply the SDF to dynamic environments due to its long construction time. To overcome this problem, we predict F_{col} using a deep neural network without constructing the SDF. Our network uses an occupancy grid updated from sensor data as the input to reflect changing environments in real-time. F_{col} is used to select an IK candidate far away from obstacles among generated IK candidates.

3.4 Real-time Collision-free Inverse Kinematics

Figure 3.2 shows the system flow. In this section, we explain how the proposed method generates IK candidates based on an optimization-based IK and how it assesses the motion feasibility. We then introduce the method that selects the best candidate from among all IK candidates using our objective functions, considering collision avoidance with obstacles and the continuity of joint configurations. We also propose a collision-cost prediction network (CCPN) for avoiding dynamic obstacles.

3.4.1 IK candidates

While a redundant manipulator has various joint configurations for a target end-effector pose x_{tar}, joint configurations with the motion feasibility are limited. In particular, considering the given short time and velocity limits of joints, the possible moving range of joints from the current joint configuration q_{cur} is very narrow. Finding many IK candidates within possible moving range in a short given amount of time can increase the performance of our approach. To generate IK candidates Q_{cand} within the most likely joint bounds, we use an optimization-based IK approach and set its initial configuration, q_{init}^{opt-IK}, to randomly generated configurations using a Gaussian distribution (Figure 3.2(b)); we use the Jacobian-based IK solver.

An optimization-based IK method starts with the initial configuration q_{init}^{opt-IK} and finds a joint configuration q for x_{tar} by iteratively minimizing the error with x_{tar} in a Cartesian space. By repeatedly minimizing the error, it finds q that is close to the initial configuration q_{init}^{opt-IK} [64–66]. Also, different initial configurations can generate diverse solutions for a redundant manipulator [65, 66].

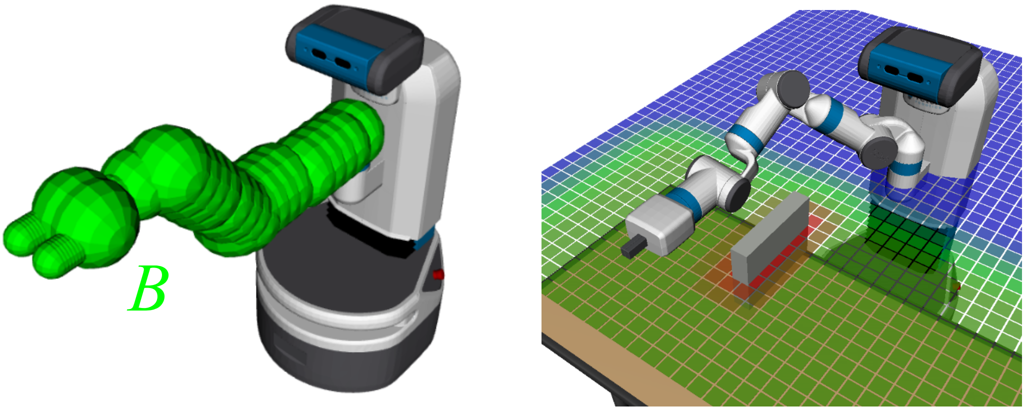
Inspired these properties, we create an initial configuration of optimization-based IK q_{init}^{opt-IK} by modifying q_{cur} using the multivariate Gaussian distribution, N:

$$q_{init}^{opt-IK} \sim N(q_{cur}, diag(\sigma_1^2, \sigma_2^2, ..., \sigma_D^2)), \tag{3.1}$$

where $diag(\cdot)$ is a diagonal, covariance matrix of size D and σ_i is the standard deviation for considering the properties of the i-th joint. $diag(\cdot)$ controls how closely IK candidates are generated from the current configuration q_{cur}. Specifically, σ_i of $diag(\cdot)$ is determined by $\mu \times v_i$, where v_i is the maximum velocity of the i-th joint and μ is a constant that serves to adjust the closeness of IK candidates from q_{cur}. When μ is too small, it is difficult to obtain diverse Q_{cand}, whereas it is difficult to obtain Q_{cand} satisfying the joint velocity limits when μ is too large; thus, we experimentally found $\mu = 0.1$ to strike the balance between diversity and generating many feasible Q_{cand}. In addition, we consider v because the maximum velocity of each joint $v = \{v_1, v_2, ..., v_D\}$ can differ.

There is no guarantee that Q_{cand} generated by the aforementioned process have the motion feasibility. We thus check the constraints for collision, joint velocity limits, and kinematic singularities. Note that we delay collision checking until after we calculate our cost function to reduce the computational overhead of collision detection; we use FCL [67], a commonly used collision checker. The joint velocity limit checks whether the manipulator can move from q_{cur} to the generated Q_{cand} during the given time t. In the case of kinematic singularities, we use the lower bound of the manipulability [54] obtained from random configurations, as is used by RelaxedIK [12]. In short, we check Q_{cand} where the manipulability value is greater than the lower bound.

When checking the constraints for Q_{cand}, there can be no feasible solution due to invalid command or insufficient Q_{cand} caused by the short time constraints and the randomness of our generation method. In those cases, we stop the robot's movement by following prior methods of handling dynamic obstacles [18, 68]. Since no algorithm guarantees success in the problem of dealing with unknown dynamic obstacles in real-time, the prior methods have decided to stop the robot's movement for robot safety [18, 68]. Furthermore, our problem deals with an unknown



(a) Simplification of robot links. (b) Signed distance field for external obstacles.

Figure 3.3: These figures show (a) the simplified robot links, spheres B, used for manipulation and (b) signed distance field for external obstacles, the table and gray box. The color of cells represents the distance from the closest obstacles; as the distance from a cell to the obstacle becomes smaller, its color gets closer to the red, and vice-versa to blue.

future command from a user. To increase the likelihood of success against unknown commands and obstacles in the future, we select a joint configuration away from obstacles among Q_{cand} through our objective function.

3.4.2 Objective Function

For one target end-effector pose x_{tar}, we can use all of Q_{cand} with the motion feasibility. However, to execute consecutive commands, it is necessary to consider the continuity between joint configurations and collision avoidance with obstacles. Because a collision can occur from an unknown future command and obstacles, we focus on selecting a joint configuration that is far away from obstacles among generated Q_{cand}. In this work, we select the configuration with the smallest value among Q_{cand} with the motion feasibility using the objective function, U, which numerically expresses the aforementioned properties.

We model the objective function U to have two cost terms for collision avoidance with obstacles, F_{col}, and smoothness between joint configurations, F_{smooth}:

$$U = F_{smooth} + w_{col} F_{col}, \tag{3.2}$$

where w_{col} is the weight for F_{col} in our objective function; we empirically set w_{col} to 2.0.

F_{col} represents a quantitative collision cost determined by calculating the approximate distance between robot links and obstacles. We first introduce a method of computing F_{col} using the SDF and then explain our deep neural network for efficiently avoiding dynamic obstacles (Sec. 3.4.3).

The SDF can significantly reduce the computational overhead during the planning stage by calculating the distance information for static obstacles in advance. Thus, many works [16, 55] use the SDF to improve the performance of their optimization-based planners. For the effective use of the SDF, we simplify the robot links used for manipulation into spheres, $B = \{b_1, b_2, ..., b_n\}$ (Figure 3.3). Through this method, we can calculate an approximate distance, $\tilde{d_i}$, from the nearest obstacles to the i-th sphere b_i, simply as $SDF(p_{b_i}) - r_{b_i}$, where $p_{b_i} \subset \mathbb{R}^3$ and r_{b_i} are correspondingly the center position and radius of b_i, and $SDF(p_{b_i})$ returns the SDF value at p_{b_i}.

Unlike most existing optimization-based methods that calculate a cost considering continuity between joint configurations [16,55], we compute F_{col} only for a joint configuration q because the desired joint configuration q_{des}

is very close to the current configuration q_{cur} to satisfy the joint velocity limits given the short time t. Therefore, F_{col} using the SDF is represented as:

$$F_{col} = \sum_{b}^{|B|} \max\left(\varepsilon - (SDF(P(q,b)) - r_b), 0\right), \tag{3.3}$$

where $P(q,b)$ is the partial forward kinematics that computes the position of a sphere $b \in B$ at a joint configuration q, and ε is the clearance value to make zero when the distance from the obstacle is far enough; ($SDF(\cdot) < \varepsilon$), and we set ε to $1.0m$. In short, F_{col} increases as the distance between obstacles and robot links decreases.

F_{smooth} in Eq. 3.2 serves to synthesize smooth joint configurations for consecutive commands. We consider the joint velocity and acceleration from the past three configurations. F_{smooth} is formulated via:

$$F_{smooth} = \|q_i - q_{i-1}\|^2 + \|\dot{q}_i - \dot{q}_{i-1}\|^2. \tag{3.4}$$

3.4.3 Collision-Cost Prediction Network

Although the SDF helps to accelerate the calculation of a collision cost F_{col}, it is difficult to use in dynamic environments due to its long construction time, e.g., 61ms for a $0.025m$ resolution. Furthermore, as the resolution of the SDF increases, longer construction times become necessary. Unfortunately, our approach needs a high-resolution of the SDF to accurately differentiate F_{col} of the IK candidates Q_{cand}, which are located very close to each other to satisfy the joint velocity limits given the short time t.

For a real-time approach in dynamic environments, we present a collision-cost prediction network (CCPN) that estimates F_{col} of Q_{cand} directly without constructing a SDF. We set the ground-truth data to collision costs computed from Eq. 3.3 using the SDF with a high-resolution of $0.005m$. Although the high-resolution SDF requires a long construction time and large amount of memory, it can provide very accurate distance information from obstacles. In addition, our CCPN serves to select a joint configuration far away from obstacles among Q_{cand}, and thus it learns to predict not only the value of F_{col} but also the ranking order for F_{col} of Q_{cand}.

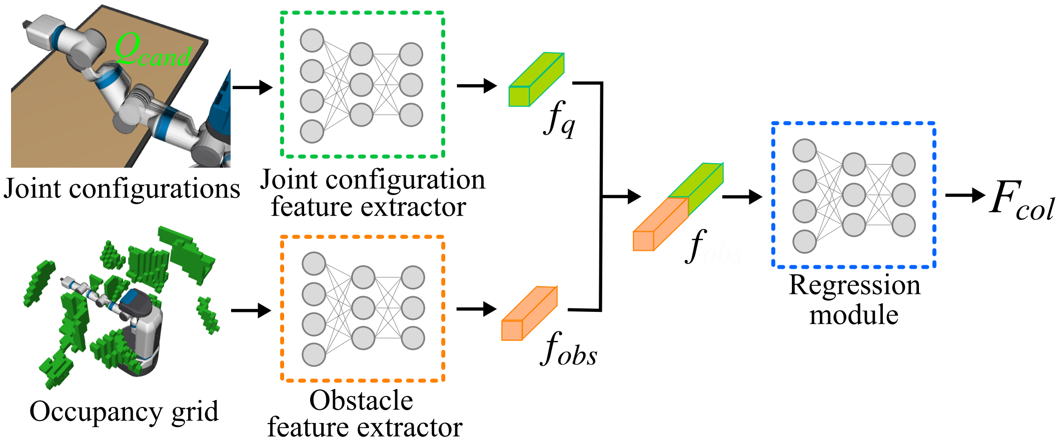
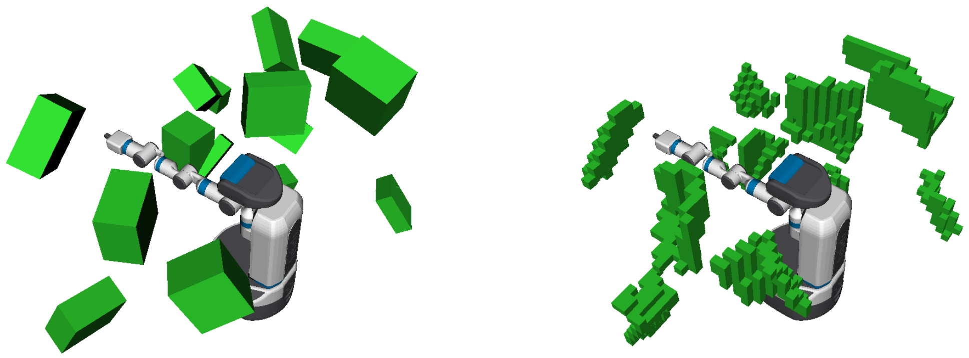


Figure 3.4: This figure shows the collision-cost prediction network (CCPN), which consists of two feature extractors and one regression module.

Network architecture The CCPN aims to estimate collision costs close to computation by Eq. 3.3. As explained in Sec. 3.4.2, Eq. 3.3 is computed from the distance information of the obstacles in the SDF and the positions of the target robot links, as simplified links B according to a joint configuration q. We design the architecture of the CCPN by reflecting such information of Eq. 3.3.



(a) Randomly distributed obstacles.

(b) Occupancy grid.

Figure 3.5: These figures show one of the generated environments to obtain training data. From (a) randomly distributed obstacles, we construct (b) an occupancy grid with updates from sensor data.

As shown in Figure 3.4, the CCPN extracts deep features from a joint configuration and an occupancy grid, and the regression module predicts F_{col} by concatenating the extracted two features. Although the CCPN has a simple architecture consisting of two feature extractors and one regression module, it is suitable for our real-time approach; it takes 2.3 ms to compute collision costs of 50 IK candidates.

At a high level, the regression module is designed to behave using an approach similar to that in Eq. 3.3 by predicting collision costs from the concatenated feature out of two features; the joint configuration feature, f_q, and the obstacle feature, f_{obs}. f_q has position information pertaining to the robot links, and f_{obs} contains obstacle information from an occupancy grid in the role of the SDF.

An occupancy grid includes information about obstacles and is updated from sensor data, while removing sensor noise based on the probabilistic update rule [56]. For real-time update, we set the resolution of the occupancy grid to $0.05m$ and the size of the occupancy grid is $40 \times 40 \times 40$. Our network extracts f_{obs} from the occupancy grid of $0.05m$, and f_{obs} serves as the high-resolution SDF. Similarly, some super-resolution and shape representation methods using a deep neural network have produced high-resolution results even with low-resolution inputs [69, 70].

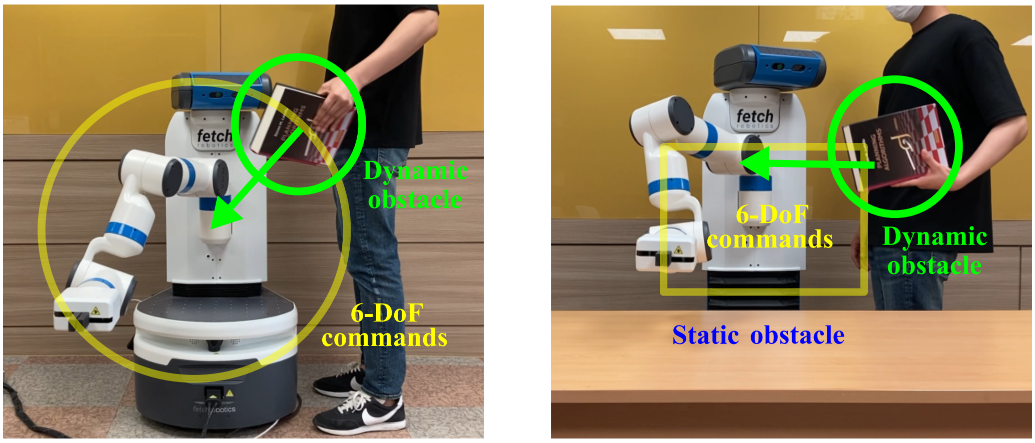
In the detailed structure of the CCPN, the feature extractor of the joint configuration has six one-dimensional (D) convolution layers and two linear layers. The feature extractor of the environment has a structure identical to that of the feature extractor of the joint configuration, except it has 3-D convolution layers instead of 1-D convolution layers. In addition, the regression module is a multi-layer perceptron with six hidden layers. We leverage batch normalization and a ReLU activation function between all layers in the CCPN.

Loss functions Prediction results from the CCPN are used to select one of Q_{cand} in conjunction with F_{smooth}. Therefore, it is important for the predicted values and, more importantly, the predicted ranking among Q_{cand}, to close to those of the ground-truth data. We use the mean squared error (MSE) loss, L_{MSE}, to match the ground-truth data, and the ranking loss, L_{rank}, to select one of Q_{cand} far away from obstacles:

$$loss = L_{MSE} + w_{rank} L_{rank}, \tag{3.5}$$

where w_{rank} is the weight for L_{rank} in our loss function.

Some deep learning approaches [71, 72] have shown that considering the ranking order of the data is effective when selecting the desired data. Inspired by these approaches, we also aim to preserve the ranking order of two different estimations for our method to select the best joint configuration among the IK candidates Q_{cand}.



(a) Circle tracing

(b) Square tracing

Figure 3.6: These figures show our experiment in the dynamic environments using a real robot, the Fetch manipulator. The robot receives 6-DoF commands consecutively to trace (a) circle and (b) square, while avoiding an incoming book in the direction of the green arrow. Additionally, (b) includes a static obstacle, a table. Note that we give 6-DoF commands consecutively to follow the yellow line.

L_{rank} is measured by comparing a data pair, and we make pairs by dividing a subset of the training data used in one training iteration, as mini-batch data, in half. For this type of data pair, L_{rank} computes the difference between two prediction values, $\hat{y_1}$ and $\hat{y_2}$, as to whether their ranking order is incorrect. Otherwise, L_{rank} is 0. Specifically, L_{rank} is formulated as:

$$\begin{aligned} L_{rank} &= \max(\Gamma(y_1, y_2) \times (\hat{y_1} - \hat{y_2}), 0), \\ s.t. \quad \Gamma(y_1, y_2) &= \begin{cases} 1, & y_1 < y_2, \\ -1, & \text{otherwise}, \end{cases} \end{aligned} \tag{3.6}$$

where y_1 and y_2 are the ground-truth values for $\hat{y_1}$ and $\hat{y_2}$, respectively.

Training dataset To train the CCPN, we construct datasets that consist of 500 random joint configurations in 2000 different environments for one million data instances in total. Some prior learning-based motion planning methods [73, 74] constructed datasets from randomly generated obstacles to cover a diverse set of environments. By adopting this protocol, we construct the environments by randomly positioning obstacles with arbitrary sizes (Figure 3.5). From each environment, we extract occupancy grids and F_{col} of the joint configurations as the ground-truth data through using Eq. 3.3 with the high resolution of SDF; the SDF is built from the extracted occupancy grid, same as the network input information.

The CCPN is trained using the Adam optimizer with an initial learning rate of 5×10^{-5} and set the mini-batch size to 128. The training process takes 100 epochs, and the learning rate is decayed to 5×10^{-6} at 50 epoch.

3.5 Experiments And Analysis

In this section, we introduce our experiment setting and discuss the experimental results. Various experiments are tested on a machine that has a 3.60GHz Intel i7-9700K CPU, 32GB of RAM, and a RTX 2080 Ti graphics card.

Table 3.1: Construction times of the SDF with varying resolutions and computation time of collision costs for 50 IK candidates using the SDF and the CCPN.

Fetch 7-DoF	SDF			CCPN
Resolution (m)	0.005	0.025	0.05	-
Construction time (ms)	8400	61	7	**0**
Computation time (ms)	17.5	6.0	4.5	**2.3**

3.5.1 Experiments in dynamic environments

In dynamic environments, we tested our method using a real robot, the Fetch manipulator; for sensing dynamic obstacles, we used the 3-D RGB-D camera mounted on the robot head. To test the usability of the proposed method, CCPN, we also compare its results against using a SDF. A SDF cannot be used directly in a dynamic environment, especially when the construction time is longer than the given time, 30ms. We therefore test the SDF with a resolution of $0.05m$, which has a low construction time, $7ms$ (Table 3.1). In addition, we test two types of CCPNs learned with and without L_{rank}.

We prepare three dynamic problems with different obstacles and 6-DoF commands. The first problem is to carry out the command in the form of a straight line transmitted using the joystick, while avoiding an incoming cart (Figure 3.1). Next, the second and third problems consecutively give 6-DoF commands to trace a circle and a square, while avoiding an approaching book (Figure 3.6). The third problem also involves a table as a static obstacle. The 6-DoF commands for three problems are provided consecutively, and our method progressively computes a joint configuration for each single command; our method does not know the future commands. In addition, since these dynamic problems are designed to evaluate the proposed method, the dynamic obstacle deliberately approaches the robot.

To determine the obstacles, we update the occupancy grid from sensor data in real-time [56]. As implementation details, we filter out sensor data corresponding to the manipulator, the spheres B shown in Figure 3.3a, as those data are not part of the environment. For the real-time update of the occupancy grid, including the filtering process, we used the occupancy grid with a resolution of $0.05m$; its update time is $30ms$ on average. The updated occupancy grid is used to check the collision with obstacles and as input to our CCPN for grasping the environment information; in the case of static obstacles, we can use prior obstacle information when precise collision checking is required, e.g., narrow passage. It should be noted self-collision detection is performed with the robot geometry information.

In addition, conducting experiments in real dynamic environments always involves different speeds of dynamic obstacles and timings of the human commands, which is inappropriate for fairly testing different methods. For a fair evaluation of different methods, we use a recorded data consisting of sensor data and consecutive commands obtained from a real experiment. Concretely, we prepare 20 different recorded data for each problem through real experiments; the 20 recorded data have slightly different speeds of dynamic obstacles and timings of the commands.

From the recorded data, we evaluate whether the computed joint configurations accurately match the given end-effector poses, while achieving the motion feasibility in dynamic problems. We measure the position and orientation error by adding one more between a current and a given end-effector pose for precise computation, since our target robot, a redundant manipulator, has multiple joint configurations for a target end-effector pose. We also regard failure when we do not find a joint configuration with the motion feasibility for each problem; in

Table 3.2: Results with 20 experiments with different methods in dynamic environments.

Fetch 7-DoF	Straight (Figure 3.1)			Circle (Figure 3.6a)			Square (Figure 3.6b)		
	PE	OE	NF	PE	OE	NF	PE	OE	NF
SDF (0.05m res.)	5.5e-5	1.0e-4	12	-	-	20	4.3e-5	7.1e-5	15
CCPN ($w_{rank} = 0$)	5.6e-5	9.5e-5	7	5.0e-5	7.8e-5	9	1.8e-4	1.1e-4	6
CCPN ($w_{rank} = 10$)	6.3e-5	1.1e-4	**3**	5.3e-5	8.4e-5	**1**	6.6e-5	1.0e-4	**2**

PE: Position error (m). OE: Orientation error (rad). NF: Number of failures.

this case, we stop the movement of the robot.

Table 3.2 shows the results in dynamic environments with different methods using a 0.05m resolution of SDF and two types of CCPNs trained by only L_{MSE} and by L_{MSE} and L_{rank} together. Using a 0.05m resolution of the SDF, which satisfies the real-time performance requirement, shows the highest number of failures among the tested methods, as its resolution cannot accurately distinguish F_{col} of nearly located Q_{cand}. Moreover, the CCPN learned w/ L_{rank} shows fewer failures than the CCPN w/o L_{rank} due to being learned to distinguish F_{col} of Q_{cand}.

When measuring the position and orientation errors for the success cases, all methods show reasonably low errors. This is because all methods generate Q_{cand} through our generation method, and generated Q_{cand} have fairly low errors for x_{tar} to follow a given command.

In conclusion, our proposed method performs consecutively given commands in real-time while avoiding dynamic and static obstacles effectively. A quantitative analysis of the proposed method will be discussed in the next sub-section.

3.5.2 Analysis

To analyze our proposed method, we construct validation sets using three different configurations of robots, Kuka iiwa with 7-DoF, and the Fetch manipulator with 7 and 8 DoF. We also evaluate the performance of the CCPN with varying values of w_{rank} and the SDF with different resolutions.

To evaluate our proposed method in various environments, we prepare 200 environments with randomly distributed obstacles as shown in Figure 3.5a. Each environment has ten random end-effector poses, and we obtain 50 IK candidates for one end-effector pose. The reason we use 50 IK candidates as a standard is that our approach generates about 125 IK candidates on average for a given 30ms, and about 50 of them satisfy the constraints except for collision. We ensure that the 50 candidates have different collision costs so as to evaluate whether the predicted collision costs have the correct ranking; we compute the collision costs at a resolution of 0.005m of SDF to achieve high accuracy. Furthermore, the 50 IK candidates are located within a very narrow joint boundaries considering the joint velocity limit and given short time.

From the validation sets, we compute F_{col} for the different tested methods. We evaluate how well estimated F_{col} of 50 IK candidates have similar ranking orders among them compared to the corresponding ground-truth values. To measure the similarity, we measure the ranking orders of the predicted outcomes and do the same procedure for the ground-truth values, after which we compute the L1 distance between them. In this work, we refer to this metric as the "ranking error". Intuitively, the low ranking error implies that it is highly possible to select an IK candidate far away from obstacles among various IK candidates.

Table 3.3 shows the ranking error of different SDFs and CCPNs with varying configurations of the robots. Overall, the 0.05m resolution of the SDF has the highest ranking error. We also measure how unique the values

Table 3.3: Results of using the SDF with varying resolutions and the CCPN trained by varying w_{rank} in the validation dataset. The number of unique data indicates the number of different collision costs among 50 IK candidates.

		SDF w/ varying resolutions		CCPN w/ varying w_{rank}		
		0.025m	0.05m	0	10	20
Kuka 7-DoF	Ranking error	7.9	12.6	5.9	**5.8**	**5.8**
	# of unique data	13	6	**50**		
Fetch 7-DoF	Ranking error	7.5	11.1	7.0	**6.6**	6.9
	# of unique data	20	16	**50**		
Fetch 8-DoF	Ranking error	8.0	9.1	8.0	**7.6**	7.7
	# of unique data	44	43	**50**		

Table 3.4: Results of different methods with varying configurations of robots in static environments. RelaxedIK only considers self-collision, and thus is tested in two non-obstacle scenes.

		Square tracing		"S" tracing		Rotation task		Writing "hello"	
		PE	OE	PE	OE	PE	OE	PE	OE
Kuka 7-DoF	RelaxedIK	-	-	-	-	5.4e-2	2.9e-1	3.9e-2	1.6e-2
	Ours	4.8e-5	3.6e-4	3.5e-5	6.3e-5	**3.0e-5**	**2.8e-4**	**3.2e-5**	**7.3e-5**
Fetch 7-DoF	RelaxedIK	-	-	-	-	5.6e-2	9.9e-2	4.6e-2	2.4e-2
	Ours	4.0e-5	6.9e-5	5.0e-5	7.2e-5	**6.4e-5**	**1.7e-4**	**6.3e-5**	**1.3e-4**
Fetch 8-DoF	RelaxedIK	-	-	-	-	4.1e-2	9.5e-2	3.8e-2	1.7e-2
	Ours	5.4e-5	8.9e-5	6.3e-5	6.6e-5	**6.1e-5**	**1.6e-4**	**7.3e-5**	**1.1e-4**

PE: Position error (m). OE: Orientation error (rad).

for the SDF and CCPN are. The tested SDF has many more duplicate values over the tested CCPN. Although the accuracy of the SDF can be improved as we increase the resolution, its construction time increases exponentially (Table 3.1), becoming inappropriate for our real-time purpose.

On the other hand, the CCPN does not require construction time at runtime and has fewer ranking errors than the SDF with resolutions of $0.05m$ and even $0.025m$. Furthermore, the CCPN computes collision costs of 50 IK candidates nearly twice as fast compared to the use of tested SDF with the $0.05m$ resolution (Table 3.1). Owing to such performances, our method achieved an average of 90% success rate for three dynamic problems. These results indicate that the CCPN has computational advantages for our real-time approach in dynamic environments and selects a joint configuration away from obstacles among Q_{cand}.

In addition, the CCPN which underwent learning through L_{MSE} and L_{rank} shows smaller ranking errors (Table 3.3) and higher success rates in dynamic environments than the CCPN trained by only L_{MSE} (Table 3.2). In particular, the greatest difference was found in the circle problem, as shown in the middle column of Table 3.2. This improvement stems from the fact that L_{rank} reduces the ranking error of the CCPN, allowing a better IK candidate to be selected.

Lastly, we test our method for three different configurations of robots in four static problems, which were used in earlier work [55]. Table 3.4 shows the results of different methods with varying configurations of robots in the four tested static problems. We compare our method with RelaxedIK in open-space environments where only self-collisions matter. Note that RelaxedIK does not consider external obstacles. In two problems without external obstacles, our method shows higher accuracy than RelaxedIK. Our method also successfully solved two problems with external obstacles. These results indicate that the proposed approach can find the desired joint configuration q_{des} that exactly matches the target end-effector pose x_{tar} for the given short time ($30ms$) while ensuring motion feasibility.

Chapter 4. Towards Safe Remote Manipulation: User Command Adjustment based on Risk Prediction for Dynamic Obstacles

4.1 Introduction

Real-time remote manipulation has been primarily used in special sites (e.g., medical facilities or nuclear power plants) that require sophisticated or hazardous work [4,5]. Recent studies [6,7,9] have expanded the scope of remote manipulation to environments around us, such as convenience stores (e.g., Telexistence[1]) and homes. In environments around us, there are various obstacles, both static and dynamic, and avoiding such obstacles is a critical issue for safe remote manipulation.

Recent proposed inverse kinematics (IK) methods for real-time remote manipulation, e.g., CollisionIK [15] and RCIK [1], handle static and dynamic obstacles to find collision-free joint configurations from consecutively given user's commands. These IK approaches have the characteristic of following a user's command, and thus the user's judgment on giving proper commands also plays a big role in obstacle avoidance. However, a user makes judgments by observing a restricted environment through a camera, and thus there is a possibility that the user may encounter unexpected situations for giving commands in remote manipulation (Figure 4.1).

In the case of static obstacles, we can keep the safety by naïvely stopping a robot, even if the user's command has a possibility of causing a collision. However, it is difficult to guarantee the safety against dynamic obstacles with uncertain motion [18, 19]. To increase the probability of avoiding dynamic obstacles, it is necessary to identify the danger of dynamic obstacles in advance and take action to avoid them. For safe remote manipulation, we aim to adjust user's commands to avoid obstacles according to the risk of dynamic obstacles. Conversely, when there is no risk of dynamic obstacles, a robot follows the user's command so as not to interfere with the user's desired tasks, e.g., picking an object on a table.

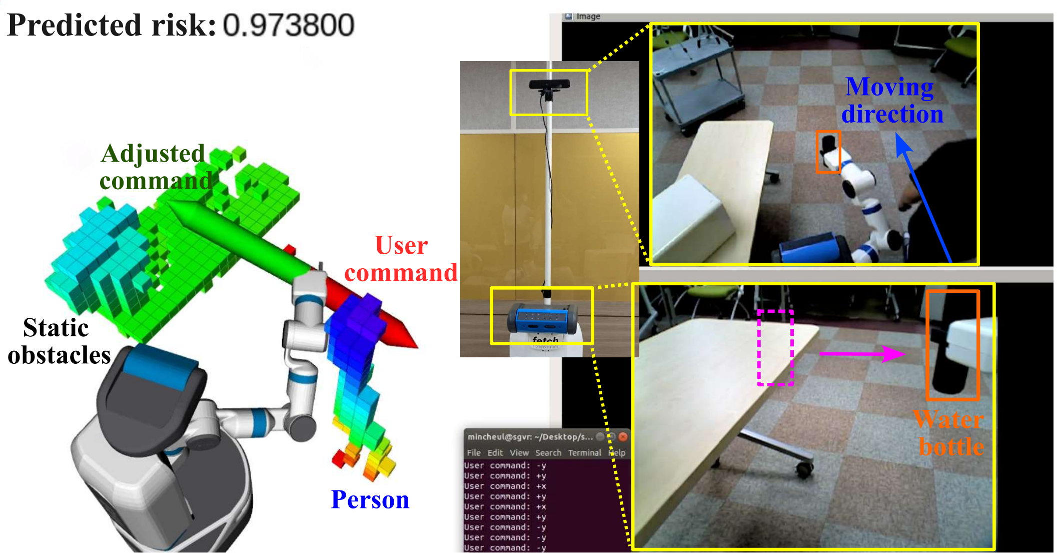
Main contributions. In this work, we present a risk-aware user command adjustment method to adjust risky commands that can cause collisions with dynamic obstacles due to a user's unpredictability or carelessness (Figure 4.1). We suggest a risk prediction network (RPN) and an obstacle avoidance command network (OACN) to minimize the delay of remote operation and to handle sensor noise, which is the difficulty of a real environment. Using two networks, our method predicts the risk of dynamic obstacles and then adjusts a user command to avoid obstacles depending on the predicted risk.

We model the risk of dynamic obstacles utilizing the proximity between robot links and dynamic obstacles, and the RPN learns the modeled risk values with consecutive robot state and obstacle information (Sec. 4.4.3); we use occupancy grids updated from sensor data in real-time to grasp obstacle information. Based on the predicted risk from the RPN, our method decides an adjusted command between a user command and the output of the OACN. We train the OACN via reinforcement learning so that the final adjusted command avoids obstacles when the predicted risk is high, otherwise, we follow the user's commands (Sec. 4.4.2).

We evaluate our proposed method in environments including dynamic obstacles varying velocities and environments including only static obstacles (Sec. 4.5.1). We show that our method improves the safeness in dynamic environments where a collision occurs when naïvely following given commands. We also show that our RPN robustly handles sensor noise by predicting the risk of dynamic obstacles close to zero in static environments. Lastly, our method shows a fast computation time of $3ms$, and we verify the feasibility of our method in real

---

[1] `https://tx-inc.com`



```
User command: -y
User command: +y
User command: +x
User command: +y
User command: +x
User command: +y
User command: -y
User command: -y
User command: -y
```

Figure 4.1: This shows the visualization of the user monitor screen for remote manipulation. In the visualized situation, the user gives commands (red arrow) to the robot to move a water bottle (orange boxes) from the dotted magenta box to the direction of the magenta arrow, and a person suddenly appears in the camera's field of view (blue arrow). Our method predicts high risk of collision, which is expressed between 0 and 1, and for safe remote manipulation, it adjusts the user's command accordingly to move the robot away from the obstacle (green arrow). The cells of the occupancy grid are colored according to the z-axis value.

environments (Sec. 4.5.2).

4.2 Related Work

In this section, we introduce inverse kinematics methods related to remote manipulation and motion planning approaches for dynamic obstacle avoidance.

4.2.1 Inverse kinematics for remote manipulation

In remote manipulation, a user can easily control a manipulator by giving a command to the manipulator's end-effector in the Cartesian space. Accordingly, inverse kinematics (IK) techniques have been widely used in remote manipulation to compute a joint configuration for the user command [13, 14, 75]. Conventional IK approaches (e.g., IKfast [10] and Trac-IK [11]) focus on finding a joint configuration that matches the given end-effector pose. However, to synthesize feasible joint configurations for consecutively given user commands, it is necessary to consider several constraints, such as collision, continuity of joints, and kinematic singularity [1].

RelaxedIK [12] handles several constraints using an optimization-based technique and uses a neural network for real-time remote manipulation to quickly avoid self-collision. Extending RelaxedIK, CollisionIK [15] avoids static and dynamic obstacles by adding a collision avoidance term that quickly computes the shortest distance between convexified robot links and obstacles using the QuickHull algorithm. However, this method has difficulty coping with sensor noise in a real environment.

RCIK [1] also handles static and dynamic obstacles, while overcoming the difficulty of a real environment by utilizing deep learning with an occupancy grid via real-time updates based on probability [56]. In addition, RCIK accurately follows given commands through a sampling-based IK approach. Although the high accuracy is one of the important factors in remote manipulation, it requires a user's proper and fast judgment to avoid collisions of robot links, especially related to the end-effector. Since users cannot always make perfect decisions in a timely manner due to carelessness or unforeseen circumstances, we aim to adjust user commands to reduce the risk of collision for safe remote manipulation.

4.2.2 Dynamic obstacle avoidance

Motion uncertainty of dynamic obstacles threatens safe robot movement and makes it difficult to plan a collision-free robot motion for reaching a target position [76]. Classical motion planning approaches [18, 77–79] handling dynamic obstacles perform iterative replanning in a short time considering the sensing cycle. To overcome the short replanning time, Vannoy et al. [18] suggest a method that generates multiple initial trajectories to a target configuration and then iteratively updates the trajectories. Hauser et al. [77] present an adaptive time-stepping approach for replanning, considering responsiveness, safety, and completeness of planning results. In addition, Park et al. [78] present GPU-based parallel processing to accelerate optimization-based replanning. Recently, learning-based approaches [80–82] have been studied to quickly replan a trajectory for reaching a target position using a fast inference time of a neural network.

As another way, online adaption methods [83–86] avoid dynamic obstacles by adaptively modifying a given trajectory using a dynamic potential field. Furthermore, several works [19, 87] construct a learning-based safety layer that iteratively checks collisions on a planned trajectory and modifies the collision part of the trajectory to be safe.

These methods consider avoiding dynamic obstacles but use special equipment (e.g., marker or motion capture system) or object detection to figure out dynamic obstacles in a real environment with static and dynamic obstacles. This is because real sensor data for static obstacles is not constant due to sensor noise. Also, it is time-consuming to figure out the position and velocity of dynamic obstacles. Our method uses consecutive sensor data without special equipment and predicts the risk of dynamic obstacles while handling sensor noise. According to the predicted risk, our method determines the final command between the user's command and a command to avoid obstacles.

4.3 Background

4.3.1 Problem definition

Our goal is to perform safe remote manipulation in environments including static and dynamic obstacles. In this work, we deal with real-time remote manipulation in which a user consecutively gives a command, $\Delta x_u \in \mathbb{R}^6$, to the end-effector in the Cartesian space. Δx indicates the amount of movement for the end-effector, and a target end-effector pose, x_u, from a user command is calculated by adding a current end-effector pose, x_{cur}, and Δx_u; $x_u = x_{cur} + \Delta x_u$. For x_u, a real-time IK solver, such as RCIK [1], synthesizes a joint configuration, q_u, considering various constraints, e.g., continuity of joints, collision avoidance, and kinematic singularity. In this work, we mainly consider a redundant manipulator with multiple joint configurations for one end-effector pose; a redundant manipulator has greater than six degrees of freedom (DoF).

In real-time remote manipulation, a user's decision-making greatly affects the process of avoiding obstacles since a robot is designed to follow the user command Δx_u. In the case of static obstacles, we can maintain the safety from collisions by naïvely stopping a robot, since we can check whether joint configuration for x_u is collision or not [1]. On the other hand, it is more difficult to avoid collision with dynamic obstacles that have even uncertain movement. Unfortunately, a user may give commands that lead to collisions due to the user's unfamiliar control

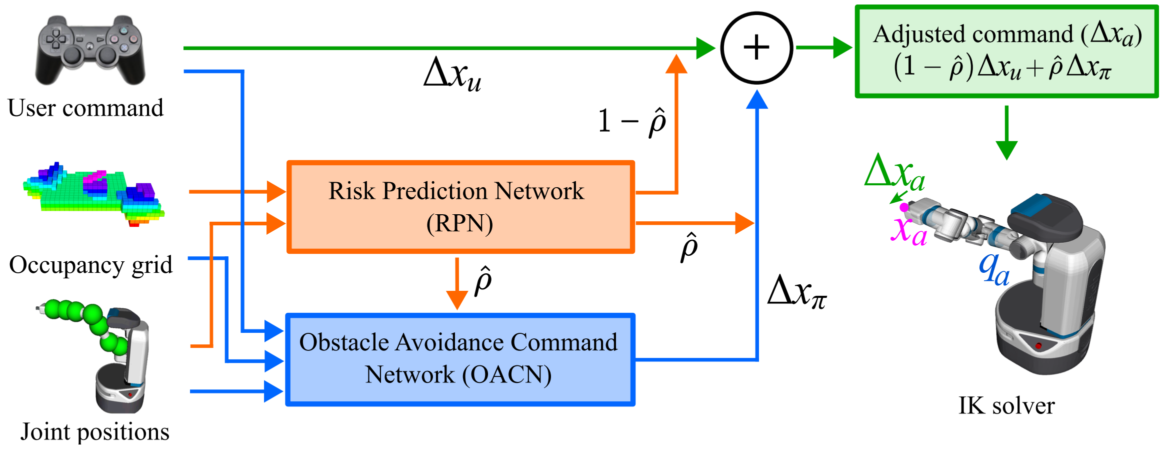

User command
Occupancy grid
Joint positions
Δx_u
$1-\hat{\rho}$
Risk Prediction Network (RPN)
$\hat{\rho}$
$\hat{\rho}$
Obstacle Avoidance Command Network (OACN)
Δx_π
+
Adjusted command (Δx_a)
$(1-\hat{\rho})\Delta x_u + \hat{\rho}\Delta x_\pi$
Δx_a
x_a
q_a
IK solver

Figure 4.2: This shows the overall system flow of our approach.

skills or unexpected situations. Therefore, there is a strong demand to protect a robot from risky commands that can cause collision with dynamic obstacles.

In this work, our problem is to cope with the danger of dynamic obstacles where adjusting the user command is essential. To solve our problem, we have to figure out the risk of dynamic obstacles in advance, and adjust the user's commands to avoid obstacles considering the risk of dynamic obstacles. When no risk of dynamic obstacles is expected, we simply follow user command.

In addition, we handle the noise problem of real sensor data to apply our method on a real environment and aim to solve our problem as quickly as possible to reduce the delay of real-time remote manipulation; in real-time IK solvers [1, 12, 15], the time interval, Δt, between commands is within $30ms$ according to the sensing cycle.

4.3.2 Reinforcement Learning

We use reinforcement learning to solve our problem by formulating our problem as Markov decision process (MDP) defined by a tuple: (S, A, P, R, γ), where S, A, P, R, and $\gamma \in [0,1)$ are the state space, the action space, a transition function, a reward function, and a discount factor, respectively [88]; more details can be found in Sec. 4.4.2. We find an optimal policy, $\pi^* : S \rightarrow A$, that maximizes a discounted cumulative reward, G, via the soft actor-critic (SAC) [89] framework. G is denoted as $\sum_{t=0}^{T} \gamma^t \mathbb{E}[R(s_t, a_t) + \alpha H(\pi(\cdot|s_t))]$, where s_t and a_t are the subset of S and A, respectively, at each time step $t \in [0, T]$, and α and H are regulation coefficients for controlling the stochasticity of π and the entropy of π, respectively.

4.4 Approach

In this section, we give an overview of our method and then describe our approach in detail.

4.4.1 Overview

To solve our problem, we present a risk-aware user command adjustment method using deep learning. We adopt a learning-based approach to quickly adjust a risky user command for dynamic obstacles and cope with sensor noise. Figure 4.2 shows our system flow. Our method consists of two kinds of networks: risk prediction network (RPN) and obstacle avoidance command network (OACN).

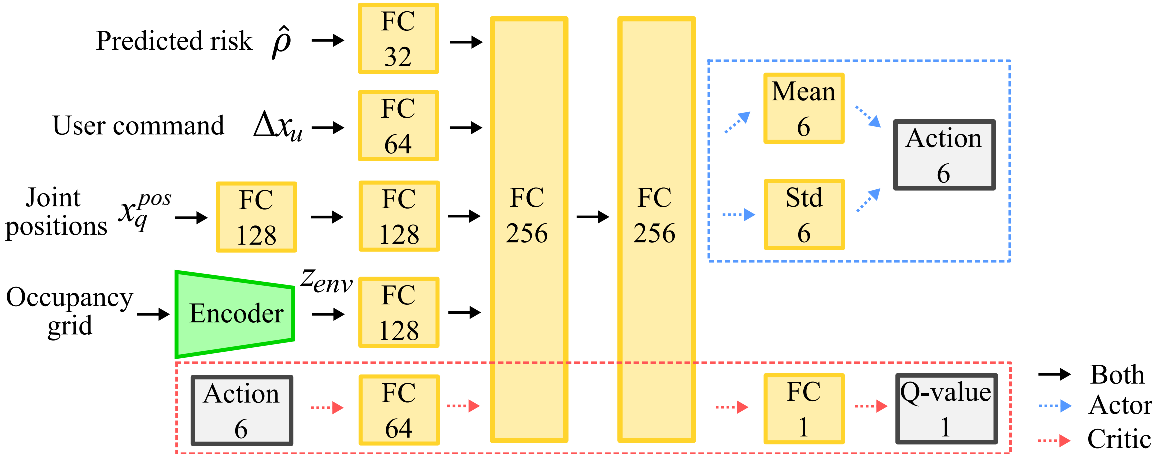


Figure 4.3: This shows the structure of obstacle avoidance command network (OACN). The encoder is a 3-D convolutional network following the obstacle feature extractor of RCIK [1].

The RPN predicts the risk of dynamic obstacles, $\hat{\rho} \in [0,1]$, in environments including static obstacles (Sec. 4.4.3). To identify the risk of dynamic obstacles, the RPN uses consecutive robot state information and occupancy grids updated in real-time from sensor data. According to the predicted risk $\hat{\rho}$, our method decides an adjusted command, Δx_a, between the user command Δx_u and a command, Δx_π, generated by OACN:

$$\Delta x_a = (1-\hat{\rho})\Delta x_u + \hat{\rho}\Delta x_\pi. \tag{4.1}$$

The OACN is learned via reinforcement learning to find an optimal Δx_π considering the predicted risk, the user command Δx_u, current robot state, and obstacle information (Sec. 4.4.2); note that an optimal Δx_π is computed in a way that Δx_a gets the maximum reward. Finally, to find a joint configuration, we give the adjusted command Δx_a to a real-time IK solver considering collision avoidance; specifically, we use RCIK [1].

4.4.2 User command adjustment

We decide an adjusted command Δx_a by combining a user command Δx_u and a command Δx_π generated by the OACN according to the predicted risk $\hat{\rho}$ (Eq. 4.1). We learn the OACN through reinforcement learning so that Δx_a avoids obstacles at high $\hat{\rho}$ and follows a user's command at low $\hat{\rho}$. To achieve the desired Δx_a, we introduce our MDP setup based on $\hat{\rho}$.

We learn our policy π, as the OACN, to maximize a discounted cumulative reward G for Δx_a. Our policy π generates $a_t \in \mathbb{R}^6$ from the current state s_t (Figure 4.3), and we synthesize Δx_π from a_t:

$$\Delta x_\pi = diag(\boldsymbol{\lambda}_{pos}, \boldsymbol{\lambda}_{ori})a_t, \tag{4.2}$$

where each element of a_t has a value between -1 to 1, and $diag(\boldsymbol{\lambda}_{pos}, \boldsymbol{\lambda}_{ori})$ is a diagonal matrix to control the amount of movement of the end-effector during the short Δt; we set $\boldsymbol{\lambda}_{pos}$ and $\boldsymbol{\lambda}_{ori}$ to $[0.01, 0.01, 0.01]$ and $[0.05, 0.05, 0.05]$, respectively.

We define s_t as a tuple: $(\hat{\rho}, \Delta x_u, x_q^{pos}, z_{env})$, where $x_q^{pos} \in \mathbb{R}^{d\times 3}$ is the 3-dimensional (D) position for d joints, and z_{env} is an obstacle feature compressed from an occupancy grid, O. We extract z_{env} from a pre-trained encoder, which is part of the variational autoencoder (VAE) [90].

Risk-based reward function. We introduce our reward function based on the predicted risk $\hat{\rho}$ to synthesize the desired Δx_a. We define the reward function so that when $\hat{\rho}$ is high, Δx_a moves away from obstacles, and when $\hat{\rho}$

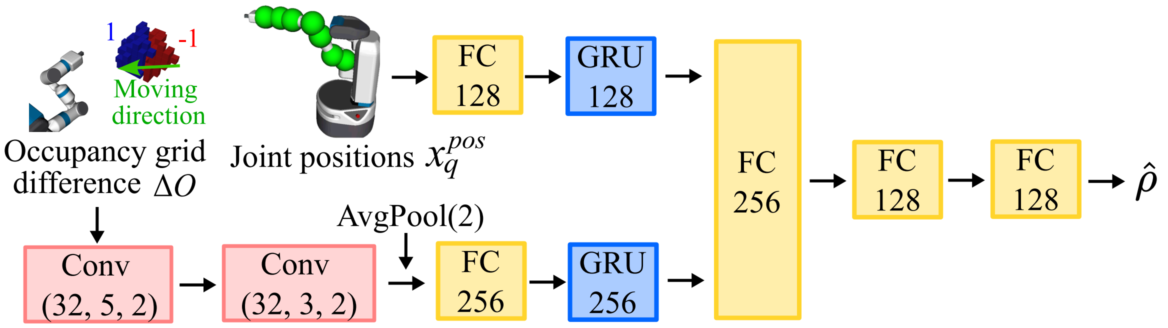


Figure 4.4: This shows our risk prediction network (RPN) for dynamic obstacles. The RPN recurrently takes joint positions x_q^{pos} and an occupancy grid difference ΔO. We apply VoxNet [2] to extract the feature of an occupancy grid difference ΔO. Furthermore, we utilize the gate recurrent unit (GRU) [3] to preserve previous robot state and obstacle information.

is low, Δx_a gets closer to Δx_u:

$$R = \begin{cases} (1-\hat{\rho})R_{match} + \hat{\rho}R_{obs}, & \text{if IK solution exists,} \\ R_{fail}, & \text{otherwise,} \end{cases} \tag{4.3}$$

where R_{match} is to match Δx_u, and R_{obs} is to avoid obstacles. When there is no IK solution for Δx_a due to collision or inaccessibility, we set R to R_{fail}; we set R_{fail} to -10.

We compute R_{match} and R_{obs} by comparing Δx_u and Δx_a. Considering Δt, the difference between Δx_u and Δx_a is very small. To amplify the difference, we apply a normalization function to R_{match} and R_{obs}. Also, it can control the relative importance of the two terms. We design a normalization function in the form of a cubic function: $F(v, \boldsymbol{w}) = w_0(w_1 v + w_2)^3$. Although the value of a cubic function can increase infinitely, we can amplify the value to a limited range, since the difference between Δx_u and Δx_a is bounded.

R_{obs} indicates whether the end-effector pose x_a from Δx_a is further away from obstacles than x_u from Δx_u:

$$R_{obs} = F(dist_{x_a} - dist_{x_u}, \boldsymbol{w}_{obs}), \tag{4.4}$$

where $dist$ is a distance between obstacles and the end-effector; we set $\boldsymbol{w}_{obs}$ to $[1/3, 200, 0]$. We simply use the distance with the end-effector instead of the whole arm. This is because an IK solver considering collision avoidance has already taken into account the distance between the whole arm and obstacles. Furthermore, the distance for the whole arm makes it difficult to converge the policy π since it is not constant for a given command.

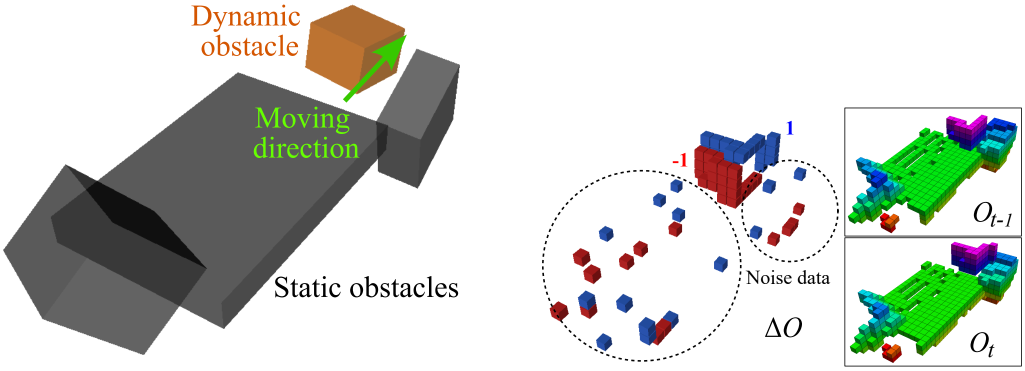
R_{match} represents the difference between Δx_u and Δx_a:

$$R_{match} = F(e_{pos} + \lambda_{match} e_{ori}, \boldsymbol{w}_{match}), \tag{4.5}$$

where e_{pos} and e_{ori} are the position and orientation error [11] between Δx_a and Δx_u, and λ_{match} is a constant to calibrate different units of position (m) and orientation (rad); we set $\boldsymbol{w}_{match}$ to $[-1, 100, -2]$ and λ_{match} to 0.17, as used in [75];

4.4.3 Risk prediction for dynamic obstacles

We aim to quickly predict the risk of dynamic obstacles in environments that contain both static and dynamic obstacles. To this end, we model the risk of dynamic obstacles, ρ, and present a risk prediction network (RPN) that learns ρ from consecutive robot state and obstacle information (Figure 4.4). To figure out dynamic obstacles,



(a) Obstacle environment

(b) Occupancy grid difference

Figure 4.5: (a) shows an example of a randomly generated obstacle environment. Gray boxes are static obstacles, and the orange box is a dynamic obstacle. The green arrow indicates the moving direction of the dynamic obstacle. (b) shows the occupancy grid difference ΔO between O_t and O_{t-1} constructed from sensor data for (a). Each cell in O is represented as 1 if there is an obstacle or 0 otherwise. Therefore, in the occupancy grid difference, 1 (blue boxes) and -1 (red boxes) indicates newly detected and disappeared obstacles, respectively. From the occupancy grid difference, we can see the moving direction of the dynamic obstacle, but the noise data due to sensor noise occurs near static obstacles (dotted black circles).

the RPN recurrently takes an occupancy grid difference, ΔO, between O_t and O_{t-1}, motivated by the work of Villegas et al. [91] as the optical flow estimation method using an image difference.

Ideally, using the occupancy grid difference would remove the information on static obstacles, but in reality, it is difficult to obtain information on dynamic obstacles due to sensor noise. Even though an occupancy grid O is updated by the probabilistic rule to reduce sensor noise, it cannot completely eliminate sensor noise near static obstacles as shown in Figure 4.5. Therefore, we learn the RPN from a dataset including sensor noises to cope with the sensor noises; we give the Gaussian noise to sensor data and set the standard deviation to 0.008.

Modeling of the risk for dynamic obstacles. Most of the approaches [15,62,92] dealing with dynamic obstacles have used a distance between a robot and the obstacles and considered their moving direction and velocity. Inspired by these methods, we model the risk of dynamic obstacles ρ as the proximity between robot links and dynamic obstacles, and its amount of change. ρ is represented using the collision cost used in [75,93]:

$$\rho = \sigma\Big(max\big(col_{q_t}^{dyn} + \lambda_\rho \sum_{i=1}^{N} (col_{q_{t-i+1}}^{dyn} - col_{q_{t-i}}^{dyn})/i, 0\big)\Big), \tag{4.6}$$

where λ_ρ is a weight for the amount of collision cost change, and N is the number of col_q^{dyn} to consider; we set λ_ρ to 2.0 and N to 3. col_q^{dyn} is a collision cost for dynamic obstacles, which is computed from the distance between spheres surrounding the robot links and dynamic obstacles. When a robot is close to dynamic obstacles, col_q^{dyn} has a large value. $\sigma(v) = tanh(v)$ is to represent ρ between 0 to 1 for easily computing Δx_a between Δx_u and Δx_π (Eq. 4.1).

4.4.4 Training details

We train the OACN via reinforcement learning using the SAC [89] algorithm and the RPN via supervised learning by gathering learning data during the training of the OACN. We prepare three kinds of obstacle scenes, including 1) static obstacles, 2) dynamic obstacles, and 3) both static and dynamic obstacles.

Table 4.1: Results in two kinds of obstacle environments: both static and dynamic obstacles, and only static obstacles.

	Static and dynamic obstacles					Static obstacles
Velocity range of dynamic obs. (m/s)	[0.4, 0.5)	[0.5, 0.6)	[0.6, 0.7)	[0.7, 0.8)	[0.8, 0.9]	-
	Success rate (%)					Command error (mean, std)
RCIK [1]	0					-
Ours	93	86	83	78	74	6.6×10^{-5}, 8.1×10^{-5}

Command error: difference between Δx_u and Δx_a ($e_{pos} + \lambda_{match} e_{ori}$).

Each scene has box-shaped obstacles with random sizes and positions and has no more than five fixed obstacles and no more than two dynamic obstacles (Figure 4.5a). Dynamic obstacles have velocity between $0.4 m/s$ and $0.9 m/s$. The velocity of the end-effector is less than $0.57 m/s$ considering λ_{pos} and the short Δt; we set Δt to $30 ms$ in simulation environments, considering the sensing cycle.

To recognize obstacles, we construct an occupancy grid O in real-time using the SuperRay [56]; the resolution of O is $0.05m$ and the size of O is $40 \times 40 \times 40$. Also, we used the extended sensing range to recognize the entire obstacle in the Gazebo simulator [94].

We set user commands to move a robot from a random start configuration, q_s, to a random goal pose, x_g, since it is difficult for a user to give commands by intervening the learning process. We decide a user command by considering simple linear movement; note that the case where a user command is not feasible, we do not use it to learn our networks. Each scene is terminated when a robot reaches x_g, or there is no IK solution for Δx_a due to collision or inaccessibility.

As one command means one step, we train the OACN during one million steps with 1×10^{-5} learning rate, 4096 batch size, 10^6 replay buffer size, $\gamma = 0.99$, and 0.995 polyak for target network update. We update the OACN 100 times every $1,000$ steps using the Adam optimizer [95].

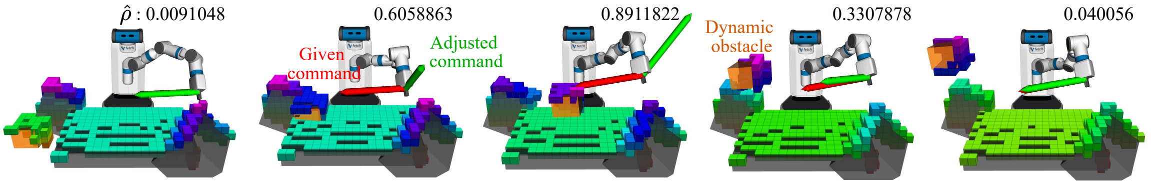
We construct the RPN dataset with $60,000$ groups, and each group consists of 16 consecutive joint positions, occupancy grids, and ρs. We train the RPN during 100 epochs using the adam optimizer with 1×10^{-5} learning rate and the mean square error (MSE) loss. In addition, we use the ten thousand occupancy grids to train the encoder of the OACN. We train the VAE [90] for the pre-trained encoder with the same condition as the RPN.

4.5 Experiments

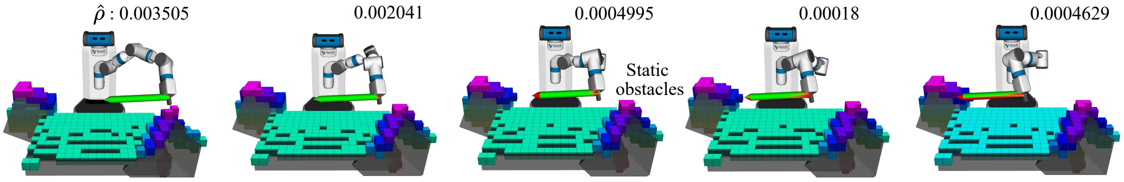
In this section, we describe our experimental setting and discuss our experimental results. Our experiments are tested on a machine equipped with a 3.60 GHz Intel i7-9700 K CPU and an RTX 2080 Ti graphics card. In these experiments, we use the Fetch manipulator with 7-DoF.

4.5.1 Evaluation

To evaluate our method, we prepare two kinds of environments one consisting of both static and dynamic obstacles and the other consisting of only static obstacles. In each environment, we construct $1,000$ problems with different configuration of obstacles and commands (Figure 4.5a); the generation method of obstacles and



(a) Result in an environment with static and dynamic obstacles



(b) Result in an environment with only static obstacles

Figure 4.6: These figures show examples of our results in (a) an environment with static and dynamic obstacles and (b) an environment with only static obstacles. Orange and gray boxes are dynamic and static obstacles, respectively, and red and green arrows indicate the magnitude and direction of given command and adjusted command, respectively. The numbers in the upper right of figures are predicted risks for dynamic obstacles $\hat{\rho}$s. (a) shows that as the dynamic obstacle approaches to the manipulator, our method predicts a high $\hat{\rho}$ and adjusts the given command to avoid the obstacle. In contrast, as the dynamic obstacle moves away from the manipulator, our method follows the given commands, predicting a low $\hat{\rho}$. (b) shows that our method robustly copes with the noisy sensor data by predicting a near-zero $\hat{\rho}$ in a static environment.

commands is the same as making the training scenes (Sec. 4.4.4).

In the environments including both static and dynamic obstacles, we measure the success rate of avoiding dynamic obstacles at different velocities of dynamic obstacles. For reliable evaluation, we extract $1,000$ problems that cause collisions with dynamic obstacles when following given commands. In contrast, in the environments including only static obstacles, we extract $1,000$ problems where there is no collision with obstacles when following given command. In these problems, we measure whether the RPN predicts the risk of dynamic obstacles close to zero handling sensor noise that occurs near static obstacles; note that sensor data for obstacles in all problems include sensor noise.

Table 4.1 shows the results of RCIK [1] and our method on the various problems. RCIK is our baseline approach and is used in our method to find a joint configuration for a given command. Since RCIK is a method of finding a joint configuration for a given command with high accuracy, we can verify that our method appropriately adjusts user commands according to the risk of dynamic obstacles. Accordingly, RCIK fails on all problems involving dynamic obstacles that are set up to collide with dynamic obstacles when following given commands.

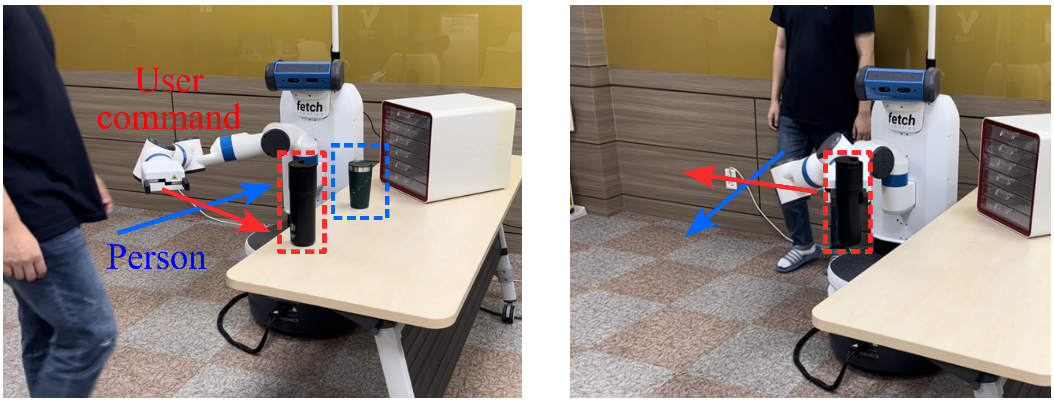
On the other hand, our method achieves a maximum success rate of 93% and a minimum of 74% throughout different velocity ranges of dynamic obstacles; the tested dynamic obstacles are all faster than the maximum velocity of the end-effector in the uniaxial direction, $0.33m/s$. These results indicate that our method can improve the collision avoidance rate by adjusting the given commands at risk of the collision to avoid obstacles in advance based on $\hat{\rho}$ (Figure 4.6a), even though the success rate decreases as the velocity of dynamic obstacles increases.

As shown in the Figure 4.6, when the dynamic obstacle is moving away from the manipulator or absent, our method shows accurately following given commands by predicting a low $\hat{\rho}$. Consequently, the difference between Δx_u and Δx_a is less than 0.0001 on average in $1,000$ different static environments (Table 4.1). This result is thanks to the RPN predicting a low $\hat{\rho}$ by robustly handling sensor noise (Table 4.2).

For a more detailed analysis of the RPN, we prepare a validation set of $2,000$ groups each for different types

Table 4.2: The mean and standard deviation of difference between ρ computed from Eq. 4.6 and $\hat{\rho}$ predicted by the RPN.

$\lvert\rho - \hat{\rho}\rvert$	No obstacle	Static obstacles	Dynamic obstacles	Both
Mean	3.85×10^{-3}	8.44×10^{-3}	5.89×10^{-2}	6.69×10^{-2}
Std	3.57×10^{-3}	3.21×10^{-2}	8.65×10^{-2}	1.03×10^{-1}



(a) Moving to pick the water bottle (b) Moving to transfer the water bottle

Figure 4.7: These figures show tested problems in a real environment. Red arrow and blue arrow indicate the moving direction of the user commands and a person, respectively. In both problems, there is a risk of collision since the moving direction of the user commands and the person intersect. (a) is that the user gives commands to the robot to pick the black water bottle (red dotted box) and the person moves to pick the green water bottle (blue dotted box). (b) is that the user gives commands to the robot to transfer the water bottle and the person suddenly appears from behind the robot.

of obstacles: no obstacles, only static obstacles, only dynamic obstacles, and both static and dynamic obstacles. Each group contains 16 consecutive joint positions, occupancy grids, and ρs calculated using Eq. 4.6; when there are no obstacles and only static obstacles, ρs are all zero. We measure the difference between ρ and $\hat{\rho}$ predicted by the RPN.

Table 4.2 shows the mean and standard deviation (std) of the difference in the validation set. It shows the largest mean difference of 0.067 and standard deviation of 0.1 in the environment including both static and dynamic obstacles. These are quite small numbers and indicate that the RPN can predict a value close to ρ we model. Moreover, the low difference in the environment containing only static obstacles (0.008) supports that the RPN can robustly cope with noisy sensor data.

In conclusion, we demonstrate through various experiments that our method can increase safety from dynamic obstacles by adjusting a user command to avoid obstacles based on the risk prediction for dynamic obstacles.

4.5.2 Real robot test

We tested our method using the real fetch manipulator to verify the feasibility of our method in a real world. In our test environment, mimicking a remote manipulation task environment, the user observes the robot state and obstacles, and gives commands to the robot using a keyboard. For the observation, we installed a camera over the head to check a wide range and additionally used a camera mounted on the robot head (Figure 4.1).

As shown in the Figure 4.7, we tested our method in situations where the user transmits the commands with a risk of collision to the robot without recognizing the movement of a person, since the person suddenly appears in the camera's field of view. In these experiments, we showed that our method can predict the risk of dynamic obstacles from real sensing data and adjust the user commands at risk of collision in a safe direction. Our detailed experimental results can be seen in the attached video.

In addition, we measure the computation time of our method. Our OACN and RPN takes about $0.6ms$ and $1.0ms$, respectively, and RCIK takes about $28.5ms$ on average. As a result, the computation time of our method is about $32ms$ on average including other computational processes, e.g., GPU allocation ($1.5ms$). Accordingly, our method can improve safety without much delay by adjusting the user's command with approximately $3ms$ additional time.

Chapter 5. Conclusion

In this dissertation, we have proposed three methods for the safe execution of user commands against various and dynamic obstacles. We have designed 1) a trajectory optimization method to follow a given end-effector path while avoiding obstacles and 2) a real-time inverse kinematics solver for consecutively given user commands in dynamic environments. Also, we have constructed 3) a user command adjustment method to handle risky commands that can cause collisions with dynamic obstacles due to a user's unpredictability or carelessness. Lastly, we have verified the feasibility of our proposed methods using the real, Fetch manipulator.

In Chapter 2, we have presented the trajectory optimization of a redundant manipulator (TORM) that holistically incorporates three important properties into the trajectory optimization process by integrating the Jacobian-based IK solving method and an optimization-based motion planning approach. Given different properties, we have suggested the two-stage gradient descent to follow a given end-effector path and to make a feasible trajectory. We have also performed adaptive exploration to avoid local minima effectively. We have shown the benefits of our method over the five prior techniques in environments w/ and w/o external obstacles using three different types of robots. Our method has robustly minimized the pose error in a progressive manner and achieved a highly-accurate trajectory at a reasonable time compared to other methods.

In Chapter 3, we have presented real-time collision-free inverse kinematics (RCIK) for performing consecutive 6-DoF commands accurately in environments including static and dynamic obstacles. For real-time solving, we have proposed a simple method that generates IK candidates with a high probability of achieving the motion feasibility. We have also suggested a collision-cost prediction network for handling dynamic obstacles. We have demonstrated the benefits that our method, which can accurately perform the consecutively given commands while avoiding obstacles in real-time.

In Chapter 4, we have proposed a risk-aware user command adjustment method to avoid the risk of dynamic obstacles. Our method predicts the risk of dynamic obstacles and adjusts a user command to avoid collisions with obstacles according to the predicted risk. We have showed the feasibility of our method towards safe execution of user commands through various experiments in simulation and real robot experiments.

We have shown that our proposed methods improve the safety of user command executions through various experiments using real robot and sensor data. Nevertheless, it is difficult to guarantee safety against dynamic obstacles, and the proposed methods can be enhanced by using more meaningful information for obstacles. Although our proposed methods use simple sensor data to figure out the accessibility of dynamic obstacles, we can utilize semantic information about dynamic obstacles (e.g., expected route and intention to move). We hope that this dissertation inspires future research on robot arm motion planning for the safe execution of user commands against various and dynamic obstacles.

Bibliography

[1] Mincheul Kang, Yoonki Cho, and Sung-Eui Yoon, "RCIK: Real-time collision-free inverse kinematics using a collision-cost prediction network", *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 1, pp. 610–617, 2021.

[2] Daniel Maturana and Sebastian Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 922–928.

[3] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling", in *NIPS 2014 Workshop on Deep Learning*, 2014.

[4] Cai Meng, Tianmiao Wang, Wusheng Chou, Sheng Luan, Yuru Zhang, and Zengmin Tian, "Remote surgery case: robot-assisted teleneurosurgery", in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2004, vol. 1, pp. 819–823.

[5] Shinji Kawatsuma, Mineo Fukushima, and Takashi Okada, "Emergency response by robots to fukushima-daiichi accident: summary and lessons learned", *Industrial Robot: An International Journal*, vol. 39, no. 5, pp. 428–435, 2012.

[6] Marcus Mast, Zdeněk Materna, Michal Španěl, Florian Weisshardt, Georg Arbeiter, Michael Burmester, Pavel Smrž, and Birgit Graf, "Semi-autonomous domestic service robots: Evaluation of a user interface for remote manipulation and navigation with focus on effects of stereoscopic display", *International Journal of Social Robotics*, vol. 7, no. 2, pp. 183–202, 2015.

[7] Zdeněk Materna, Michal Španěl, Marcus Mast, Vítězslav Beran, Florian Weisshardt, Michael Burmester, and Pavel Smrž, "Teleoperating assistive robots: a novel user interface relying on semi-autonomy and 3d environment mapping", *Journal of Robotics and Mechatronics*, vol. 29, no. 2, pp. 381–394, 2017.

[8] Gal Gorjup, Anany Dwivedi, Nathan Elangovan, and Minas Liarokapis, "An intuitive, affordances oriented telemanipulation framework for a dual robot arm hand system: On the execution of bimanual tasks", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 3611–3616.

[9] Eimei Oyama, Kohei Tokoi, Ryo Suzuki, Sousuke Nakamura, Naoji Shiroma, Norifumi Watanabe, Arvin Agah, Hiroyuki Okada, and Takashi Omori, "Augmented reality and mixed reality behavior navigation system for telexistence remote assistance", *Advanced Robotics*, vol. 35, no. 20, pp. 1223–1241, 2021.

[10] Rosen Diankov, "Automated construction of robotic manipulation programs", 2010.

[11] Patrick Beeson and Barrett Ames, "TRAC-IK: An open-source library for improved solving of generic inverse kinematics", in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE, 2015, pp. 928–935.

[12] Daniel Rakita, Bilge Mutlu, and Michael Gleicher, "RelaxedIK: Real-time synthesis of accurate and feasible robot arm motion.", in *Robotics: Science and Systems (RSS)*, 2018.

[13] Daniel Rakita, Bilge Mutlu, and Michael Gleicher, “STAMPEDE: A discrete-optimization method for solving pathwise-inverse kinematics”, in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 3507–3513.

[14] Rachel Holladay, Oren Salzman, and Siddhartha Srinivasa, “Minimizing task-space fréchet error via efficient incremental graph search”, *IEEE Robotics and Automation Letters (RA-L)*, vol. 4, no. 2, pp. 1999–2006, 2019.

[15] Daniel Rakita, Haochen Shi, Bilge Mutlu, and Michael Gleicher, “Collisionik: A per-instant pose optimization method for generating robot motions with environment collision avoidance”, in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 9995–10001.

[16] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa, “CHOMP: Covariant hamiltonian optimization for motion planning”, *The International Journal of Robotics Research (IJRR)*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[17] John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel, “Finding locally optimal, collision-free trajectories with sequential convex optimization”, in *Robotics: Science and Systems (RSS)*. Citeseer, 2013, vol. 9, pp. 1–10.

[18] John Vannoy and Jing Xiao, “Real-time adaptive motion planning (ramp) of mobile manipulators in dynamic environments with unforeseen changes”, *IEEE Transactions on Robotics (T-RO)*, vol. 24, no. 5, pp. 1199–1212, 2008.

[19] Jakob Thumm and Matthias Althoff, “Provably safe deep reinforcement learning for robotic manipulation in human environments”, in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022.

[20] Kapil D Katyal, Christopher Y Brown, Steven A Hechtman, Matthew P Para, Timothy G McGee, Kevin C Wolfe, Ryan J Murphy, Michael DM Kutzer, Edward W Tunstel, Michael P McLoughlin, et al., “Approaches to robotic teleoperation in a disaster scenario: From supervised autonomy to direct control”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2014, pp. 1874–1881.

[21] Philip Long, Tarik Keleştemur, Aykut Özgün Önol, and Taşkin Padir, “optimization-based human-in-the-loop manipulation using joint space polytopes”, in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 204–210.

[22] Rachel M Holladay and Siddhartha S Srinivasa, “Distance metrics and algorithms for task space path optimization”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 5533–5540.

[23] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller, “Anytime motion planning using the rrt”, in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 1478–1483.

[24] Samuel R Buss, “Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods”, *IEEE Journal of Robotics and Automation*, vol. 17, no. 1-19, pp. 16, 2004.

[25] Anirban Sinha and Nilanjan Chakraborty, “Geometric search-based inverse kinematics of 7-dof redundant manipulator with multiple joint offsets”, in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 5592–5598.

[26] Pasquale Chiacchio, Stefano Chiaverini, Lorenzo Sciavicco, and Bruno Siciliano, "Closed-loop inverse kinematics schemes for constrained redundant manipulators with task space augmentation and task priority strategy", *The International Journal of Robotics Research (IJRR)*, vol. 10, no. 4, pp. 410–425, 1991.

[27] Stefano Chiaverini, "Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators", *Transactions on Robotics and Automation*, vol. 13, no. 3, pp. 398–410, 1997.

[28] Oussama Kanoun, Florent Lamiraux, and Pierre-Brice Wieber, "Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task", *IEEE Transactions on Robotics (T-RO)*, vol. 27, no. 4, pp. 785–792, 2011.

[29] Mike Stilman, "Task constrained motion planning in robot joint space", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2007, pp. 3074–3081.

[30] Dmitry Berenson, Siddhartha S Srinivasa, Dave Ferguson, and James J Kuffner, "Manipulation planning on constraint manifolds", in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2009, pp. 625–632.

[31] Nikolaus Vahrenkamp, Dmitry Berenson, Tamim Asfour, James Kuffner, and Rüdiger Dillmann, "Humanoid motion planning for dual-arm manipulation and re-grasping tasks", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2009, pp. 2464–2470.

[32] Tobias Kunz and Mike Stilman, "Manipulation planning with soft task constraints", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 1937–1942.

[33] Jingru Luo and Kris Hauser, "Interactive generation of dynamically feasible robot trajectories from sketches using temporal mimicking", in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2012, pp. 3665–3670.

[34] Pragathi Praveena, Daniel Rakita, Bilge Mutlu, and Michael Gleicher, "User-guided offline synthesis of robot arm motion from 6-dof paths", in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8825–8831.

[35] Beobkyoon Kim, Terry Taewoong Um, Chansu Suh, and Frank C Park, "Tangent bundle rrt: A randomized algorithm for constrained motion planning", *Robotica*, vol. 34, no. 1, pp. 202, 2016.

[36] Riccardo Bonalli, Abhishek Cauligi, Andrew Bylard, and Marco Pavone, "Gusto: Guaranteed sequential trajectory optimization via sequential convex programming", in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6741–6747.

[37] Zachary Kingston, Mark Moll, and Lydia E Kavraki, "Exploring implicit spaces for constrained sampling-based planning", *The International Journal of Robotics Research (IJRR)*, vol. 38, no. 10-11, pp. 1151–1178, 2019.

[38] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe, *Convex optimization*, Cambridge university press, 2004.

[39] Marguerite Frank, Philip Wolfe, et al., "An algorithm for quadratic programming", *Naval research logistics quarterly*, vol. 3, no. 1-2, pp. 95–110, 1956.

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

[41] Harshit Gupta, Kyong Hwan Jin, Ha Q Nguyen, Michael T McCann, and Michael Unser, "Cnn-based projected gradient descent for consistent ct image reconstruction", *IEEE transactions on medical imaging*, vol. 37, no. 6, pp. 1440–1453, 2018.

[42] Yann Traonmilin, Jean-François Aujol, and Arthur Leclaire, "Projected gradient descent for non-convex sparse spike estimation", *IEEE Signal Processing Letters*, vol. 27, pp. 1110–1114, 2020.

[43] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal, "STOMP: Stochastic trajectory optimization for motion planning", in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 4569–4574.

[44] Peter JM Van Laarhoven and Emile HL Aarts, "Simulated annealing", in *Simulated annealing: Theory and applications*, pp. 7–15. Springer, 1987.

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

[46] Xiangyuan Jiang and Shuai Li, "BAS: Beetle antennae search algorithm for optimization problems", *ArXiv*, vol. abs/1710.10724, 2017.

[47] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan, "Fast random walk with restart and its applications", in *Sixth international conference on data mining*. IEEE, 2006, pp. 613–622.

[48] Yinghao Zhao, Li Yan, Yu Chen, Jicheng Dai, and Yuxuan Liu, "Robust and efficient trajectory replanning based on guiding path for quadrotor fast autonomous flight", *Remote Sensing*, vol. 13, no. 5, pp. 972, 2021.

[49] Tamar Flash and Renfrey B Potts, "Communication: Discrete trajectory planning", *The International Journal of Robotics Research (IJRR)*, vol. 7, no. 5, pp. 48–57, 1988.

[50] Martin Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent", in *International Conference on Machine Learning (ICML)*, 2003, pp. 928–936.

[51] Brendan O'donoghue and Emmanuel Candes, "Adaptive restart for accelerated gradient schemes", *Foundations of computational mathematics*, vol. 15, no. 3, pp. 715–732, 2015.

[52] Donghwan Kim and Jeffrey A Fessler, "Adaptive restart of the optimized gradient method for convex optimization", *Journal of Optimization Theory and Applications*, vol. 178, no. 1, pp. 240–263, 2018.

[53] David H Douglas and Thomas K Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature", *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[54] Tsuneo Yoshikawa, "Manipulability of robotic mechanisms", *The International Journal of Robotics Research (IJRR)*, vol. 4, no. 2, pp. 3–9, 1985.

[55] Mincheul Kang, Heechan Shin, Donghyuk Kim, and Sung-eui Yoon, "TORM: Fast and accurate trajectory optimization of redundant manipulator given an end-effector path", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020.

[56] Youngsun Kwon, Donghyuk Kim, Inkyu An, and Sung-eui Yoon, "Super rays and culling region for real-time updates on grid-based occupancy maps", *IEEE Transactions on Robotics (T-RO)*, vol. 35, no. 2, pp. 482–497, 2019.

[57] Chonhyon Park, Jia Pan, and Dinesh Manocha, "ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments", in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2012.

[58] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1366–1373.

[59] Avneesh Sud, Miguel A Otaduy, and Dinesh Manocha, "Difi: Fast 3d distance field computation using graphics hardware", in *Computer Graphics Forum (CGF)*. Wiley Online Library, 2004, vol. 23, pp. 557–566.

[60] Hongfeng Yu, Jinrong Xie, Kwan-Liu Ma, Hemanth Kolla, and Jacqueline H Chen, "Scalable parallel distance field construction for large-scale applications", *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, vol. 21, no. 10, pp. 1187–1200, 2015.

[61] Liangliang Zhao, Jingdong Zhao, Hong Liu, and Dinesh Manocha, "Efficient inverse kinematics for redundant manipulators with collision avoidance in dynamic scenes", in *International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2018, pp. 2502–2507.

[62] Dong Han, Hong Nie, Jinbao Chen, and Meng Chen, "Dynamic obstacle avoidance for manipulators using distance calculation and discrete detection", *Robotics and Computer-Integrated Manufacturing*, vol. 49, pp. 98–104, 2018.

[63] J Chase Kew, Brian Ichter, Maryam Bandari, Tsang-Wei Edward Lee, and Aleksandra Faust, "Neural collision clearance estimator for fast robot motion planning", *arXiv preprint arXiv:1910.05917*, 2019.

[64] Pyung Chang, "A closed-form solution for inverse kinematics of robot manipulators with redundancy", *IEEE Journal on Robotics and Automation*, vol. 3, no. 5, pp. 393–403, 1987.

[65] He Zhang, Sebastian Starke, Taku Komura, and Jun Saito, "Mode-adaptive neural networks for quadruped motion control", *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, pp. 1–11, 2018.

[66] VVMJ Satish Chembuly and Hari Kumar Voruganti, "An optimization based inverse kinematics of redundant robots avoiding obstacles and singularities", in *Proceedings of the Advances in Robotics*, pp. 1–6. 2017.

[67] Jia Pan, Sachin Chitta, and Dinesh Manocha, "Fcl: A general purpose library for collision and proximity queries", in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2012, pp. 3859–3866.

[68] Peter Lehner, Arne Sieverling, and Oliver Brock, "Incremental, sensor-based motion generation for mobile manipulators in unknown, dynamic environments", in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 4761–4767.

[69] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox, "Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs", in *IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2088–2096, IEEE.

[70] Wenming Yang, Xuechen Zhang, Yapeng Tian, Wei Wang, Jing-Hao Xue, and Qingmin Liao, "Deep learning for single image super-resolution: A brief review", *IEEE Transactions on Multimedia*, vol. 21, no. 12, pp. 3106–3121, 2019.

[71] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender, "Learning to rank using gradient descent", in *International Conference on Machine Learning (ICML)*, 2005, pp. 89–96.

[72] Donggeun Yoo and In So Kweon, "Learning loss for active learning", in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 93–102.

[73] Ahmed H Qureshi, Anthony Simeonov, Mayur J Bency, and Michael C Yip, "Motion planning networks", in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2118–2124.

[74] Xiang Li, Qixin Cao, Mingjing Sun, and Ganggang Yang, "Fast motion planning via free c-space estimation based on deep neural network", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 3542–3548.

[75] Mincheul Kang and Sung-Eui Yoon, "Analysis and acceleration of torm: optimization-based planning for path-wise inverse kinematics", *Autonomous Robots*, pp. 599–615, 2022.

[76] Baisravan HomChaudhuri, Lee Smith, Lydia Tapia, et al., "Safety, challenges, and performance of motion planners in dynamic environments", in *Robotics research*, pp. 793–808. Springer, 2020.

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

[78] Chonhyon Park, Jia Pan, and Dinesh Manocha, "Real-time optimization-based planning in dynamic environments using gpus", in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2013, pp. 4090–4097.

[79] Daniel Kappler, Franziska Meier, Jan Issac, Jim Mainprice, Cristina Garcia Cifuentes, Manuel Wüthrich, Vincent Berenz, Stefan Schaal, Nathan Ratliff, and Jeannette Bohg, "Real-time perception meets reactive motion generation", *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 3, pp. 1864–1871, 2018.

[80] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn, "Universal planning networks: Learning generalizable representations for visuomotor control", in *International Conference on Machine Learning (ICML)*. PMLR, 2018, pp. 4732–4741.

[81] Mohamed El-Shamouty, Xinyang Wu, Shanqi Yang, Marcel Albus, and Marco F Huber, "Towards safe human-robot collaboration using deep reinforcement learning", in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4899–4905.

[82] Kei Ota, Devesh Jha, Tadashi Onishi, Asako Kanezaki, Yusuke Yoshiyasu, Yoko Sasaki, Toshisada Mariyama, and Daniel Nikovski, "Deep reactive planning in dynamic environments", in *Conference on Robot Learning (CoRL)*. PMLR, 2021, pp. 1943–1957.

[83] Dae-Hyung Park, Heiko Hoffmann, Peter Pastor, and Stefan Schaal, "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields", in *IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2008, pp. 91–98.

[84] Heiko Hoffmann, Peter Pastor, Dae-Hyung Park, and Stefan Schaal, "Biologically-inspired dynamical systems for movement generation: Automatic real-time goal adaptation and obstacle avoidance", in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2009, pp. 2587–2592.

[85] Huan Tan, Erdem Erdemir, Kazuhiko Kawamura, and Qian Du, “A potential field method-based extension of the dynamic movement primitive algorithm for imitation learning with obstacle avoidance”, in *IEEE International Conference on Mechatronics and Automation*. IEEE, 2011, pp. 525–530.

[86] Andrej Gams, Tadej Petrič, Martin Do, Bojan Nemec, Jun Morimoto, Tamim Asfour, and Aleš Ude, “Adaptation and coaching of periodic motion primitives through physical and visual interaction”, *Robotics and Autonomous Systems*, vol. 75, pp. 340–351, 2016.

[87] Tu-Hoa Pham, Giovanni De Magistris, and Ryuki Tachibana, “Optlayer-practical constrained optimization for deep reinforcement learning in the real world”, in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6236–6243.

[88] Richard S Sutton and Andrew G Barto, *Reinforcement learning: An introduction*, MIT press, 2018.

[89] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al., “Soft actor-critic algorithms and applications”, *International Conference on Machine Learning (ICML)*, 2018.

[90] Diederik P. Kingma and Max Welling, “Auto-encoding variational bayes”, in *International Conference on Learning Representations (ICLR)*, 2014.

[91] Ruben Villegas, Jimei Yang, Seunghoon Hong, Xunyu Lin, and Honglak Lee, “Decomposing motion and content for natural video sequence prediction”, in *International Conference on Learning Representations (ICLR)*. ICLR, 2017.

[92] Liangliang Zhao, Jingdong Zhao, and Hong Liu, “Solving the inverse kinematics problem of multiple redundant manipulators with collision avoidance in dynamic environments”, *Journal of Intelligent & Robotic Systems*, vol. 101, no. 2, pp. 1–18, 2021.

[93] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa, “CHOMP: Gradient optimization techniques for efficient motion planning”, in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2009, pp. 489–494.

[94] Nathan Koenig and Andrew Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2004, vol. 3, pp. 2149–2154.

[95] Diederick P Kingma and Jimmy Ba, “Adam: A method for stochastic optimization”, in *International Conference on Learning Representations (ICLR)*, 2015.

Acknowledgments in Korean

로봇의 인공지능 연구를 하고 싶다는 막연한 생각에 시작했던 대학원 과정이었는데, 7년 반이라는 오랜 기간이 지나 결실을 맺게 되었습니다. 쉽지 않은 시간이었지만, 이겨내기 위해 열심히 노력했기에 후회보다는 보람되고 한 단계 성장했던 시간으로 여기고자 합니다.

먼저, 부족한 저를 끊임없는 연구 지도와 가르침으로 이끌어 주신 윤성의 교수님께 감사하다는 말씀 전하고 싶습니다. 연구뿐만 아니라 세상 살아가는데 필요한 많은 것을 배웠습니다. 교수님 가르침 잊지 않고 살아가겠습니다. 그리고 바쁜 일정에도 논문 심사 맡아주시고 지도해 주신 조성호 교수님, 명현 교수님, 김영준 교수님, 박대형 교수님께도 감사의 말씀을 드립니다.

7년 반이라는 시간 동안, SGVR에서 정말 많은 사람들과 좋은 인연 맺었습니다. 연구실이라는 낯선 공간에 들어왔었는데 이제는 이곳이 너무 익숙해져 버렸습니다. 연구실에서 함께 생활했던 선후배분들이 있었기에 쉽지 않은 생활 이겨낼 수 있었고, 좋은 추억 남길 수 있었습니다. 특히, 오랜 시간 같이 지내면서 로봇팀을 이끌어 주었던 김동혁 박사님, 권용선 박사님, 응원해 주시고 좋은 말씀해 주셨던 김수민 박사님 감사합니다. 졸업하면서 죽기 전에 볼 수 있냐고 했는데 아직도 못 보고 있는 pio 박사님, 잘 지내고 있으리라 믿습니다. 같이 졸업하는 태영군, 곧 졸업할 인규형, 우빈군에게도 좋은 추억 만들어줘서 감사하다는 말 전하고 싶습니다. 그리고 연구 도와주면서 많은 이야기 나눴던 희찬군, 윤기군, 민성군에게도 졸업할 수 있게 도와줘서 고맙습니다. 또한, 낯선 환경 속에서 쉽게 적응할 수 있도록 도와주었던 현철이형, 웅직이형, 윤석이형, 정수형, 재형군, 그리고 연구실 친구 홍선이, 함께 지냈던 영기군, 치완군, 진혁군, 형열군 외 모든 졸업한 선후배님께 고맙다는 말 전하고 싶습니다. 앞으로 좋은 연구 이어가면서 SGVR을 빛낼 이미지 재윤군, 준식군, 김민철형, 렌더링 재윤군, 세빈군, 충수형, 문성주 연구원님, 우재군, 규범군, 주민양, 주형군, 우정군에게도 응원의 말을 전합니다. 진원군도 졸업 축하합니다! Thanks to Xu Yin, Guoyuan An, and Yaxin Wang. I wish you good luck for your future! 그리고 연구만 집중할 수 있게 도와주셨던 김슬기나 선생님 감사합니다.

마지막으로, 항상 무한한 지지를 보내준 우리 가족에게 감사하다는 말 전하고 싶습니다. 제가 스스로 선택하고 그 선택에 책임질 수 있도록 키워주시고 믿어주신 부모님께 감사드리며, 항상 지지해 주었던 동생들에게도 고맙습니다. 또한, 내 소중한 친구들에게 항상 응원해 주고 지지해 줘서 고맙다는 말 전하고 싶으며, 항상 힘이 되어주는 김윤지님에게도 감사의 말씀 전합니다.

정말 많은 분들의 도움을 받았기에 힘든 시간들 이겨내고 성장할 수 있었습니다. 앞으로 새로운 환경에서 다른 어려움이 있겠지만, 대학원 과정에서의 경험으로 슬기롭게 이겨나가겠습니다. 이런 고마움 잊지 않고 살아가겠습니다. 정말 감사합니다!

Curriculum Vitae in Korean

이 름: 강 민 철

생 년 월 일: 1990년 05월 02일

학 력

2009. 3. – 2015. 2.	충남대학교 메카트로닉스 공학과 (학사)
2015. 8. – 2017. 8.	한국과학기술원 로봇공학학제전공 (석사)
2017. 8. – 2023. 2.	한국과학기술원 전산학부 (박사)

연 구 업 적

1. **Mincheul Kang**, Minsung Yoon, and Sung-Eui Yoon, *Towards Safe Remote Manipulation: User Command Adjustment based on Risk Prediction for Dynamic Obstacles* (under review).
2. Minsung Yoon, **Mincheul Kang**, Daehyung Park, and Sung-Eui Yoon, *Reinforcement Learning based Initialization of Trajectory Optimization for Path-following Problems of Redundant Manipulators* (under review).
3. **Mincheul Kang** and Sung-Eui Yoon, *Analysis and Acceleration of TORM: Optimization-based Planning for Path-wise Inverse Kinematics*, Autonomous Robots, 2022.
4. **Mincheul Kang**, Yoonki Cho, and Sung-Eui Yoon, *RCIK: Real-Time Collision-Free Inverse Kinematics Using a Collision-Cost Prediction Network*, IEEE Robotics and Automation Letters (RA-L), 2021.
5. **Mincheul Kang**, Heechan Shin, Donghyuk Kim, and Sung-Eui Yoon, *TORM: Fast and Accurate Trajectory Optimization of Redundant Manipulator given an End-Effector Path*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020.
6. **Mincheul Kang**, Donghyuk Kim, and Sung-Eui Yoon, *Harmonious Sampling for Mobile Manipulation Planning*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019.
7. Donghyuk Kim, **Mincheul Kang**, and Sung-Eui Yoon, *Volumetric Tree*: Adaptive Sparse Graph for Effective Exploration of Homotopy Classes*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019.
8. Hongsun Choi, **Mincheul Kang**, Youngsun Kwon, and Sung-Eui Yoon, *An Objectness Score for Accurate and Fast Detection during Navigation*, The World Congress on Advances in Nano, Bio, Robotics and Energy (ANBRE), 2019.
9. **Mincheul Kang**, Youngsun Kwon, and Sung-Eui Yoon, *Automated Task Planning using Object Arrangement Optimization*, International Conference on Ubiquitous Robots (UR), 2018.