

Path Planning for Continuum Arms

BY

BRANDON HANS MENG

A DISSERTATION SUBMITTED TO THE SCHOOL OF COMPUTING, JARVIS
COLLEGE OF COMPUTING AND DIGITAL MEDIA OF DEPAUL UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE
OF
DOCTOR OF PHILOSOPHY

DEPAUL UNIVERSITY
CHICAGO, ILLINOIS

2026

DePaul University
College of Computing and Digital Media
Dissertation Verification Form

This doctoral dissertation has been read and approved by the dissertation committee below according to the requirements of the Computer and Information Systems PhD program and DePaul University.

Name: Brandon Hans Meng

Title of dissertation:

Path Planning for Continuum Arms

Date of Dissertation Defense: May 20th, 2026

Iyad Kanj, Ph.D., DePaul University
Dissertation Advisor*

Isuru Godage, Ph.D., Texas A&M University
1st Reader

Hunter B. Gilbert, Ph.D., Louisiana State University
2nd Reader

Umer Huzaifa, Ph.D., DePaul University
3rd Reader

David Hanley, Ph.D., DePaul University
4th Reader

** A copy of this form has been signed, but may only be viewed after submission and approval of FERPA request letter.*

Dissertation Committee

Dr. Iyad Kanj, DePaul University (Advisor)

Dr. Isuru Godage, Texas A&M University (External)

Dr. Hunter B. Gilbert, Louisiana State University (External)

Dr. Umer Huzaifa, DePaul University

Dr. David Hanley, DePaul University

To my family and friends, for your tireless support.

Abstract

Continuum arms are robotic manipulators that rely on continuous bending and deformation for motion. They may be constructed from soft materials, rigid elements, or hybrid combinations of rigid and compliant components. Many such devices exhibit whole-body compliance, which can enable safer interaction with humans in collaborative environments such as industrial manufacturing and healthcare. In this dissertation, we study a hybrid-construction multisection continuum arm in which multiple continuum sections are joined in series. This structure enables more complex motions and planning tasks. However, the same characteristics that grant these devices their dexterity and compliance also complicate path planning, rendering it a largely open problem for continuum arms.

Even modest improvements in planning reliability or path smoothness can significantly impact the practicality of such robotic devices. Existing path planning methods for continuum arms often have limitations that restrict their effectiveness in more complex planning tasks, particularly due to nonlinear kinematics, local minima, and singularities. This dissertation develops planning algorithms for continuum arms in path tracing, static obstacle avoidance, and dynamic obstacle avoidance, and further extends these methods to coordinated planning for dual continuum arms. These approaches draw on graph theory, inverse kinematics, and sampling-based planning to generate smooth and reliable paths.

Taken together, the methods developed in this dissertation form a progression of planning methods for continuum arms, beginning with graph-theoretic and inverse kinematic approaches for reliable planning, and extending to anticipatory, temporal, and sampling-based methods targeted at path tracing, static obstacle avoidance, and dynamic obstacle avoidance. The dissertation then extends these ideas to a dual-arm model through refinement methods and sampling-based planners for coordinated tracing and obstacle avoidance. Collectively, these results show that smooth and reliable path planning for continuum arms can be achieved across both single-arm and dual-arm systems, and across static and dynamic environments.

Keywords: Path Planning, Dynamic Path Planning, Dual-Arm Planning, Continuum Arms, Sampling-based Path Planning, RRT, PRM

Contents

Dissertation Committee	i
Abstract	iii
List of Figures	viii
List of Tables	xi
Acknowledgments	xiii
1 Introduction	1
1.1 Background	1
1.2 Motivation for Continuum Arm Robots	2
1.3 Motivation for Path Planning of Continuum Arms	2
1.4 Research Objectives	3
1.5 Dissertation Structure	4
1.6 Dissertation Contributions	5
1.7 List of Research Articles	6
2 Preliminaries	7
2.1 Prototype Arm	7
2.1.1 Derivation of a Reduced DoF Mapping	8
2.1.2 Actuator Range Derivation	9
2.1.3 Kinematic Model	10
2.1.4 Dynamic Simulator	11
2.2 Kinematics	11
2.2.1 $\mathcal{C}$ -space, $\mathcal{J}$ -space, and $\mathcal{W}$ -space	11
2.2.2 Forward and Inverse Kinematics	13
2.2.3 Dual-Arm Model	13
2.3 Path Planning Preliminaries	14
2.3.1 Path Planning Problem	14
2.3.2 Dynamic Path Planning in Discrete Time	15
2.4 Graph Theory	15
2.4.1 Graphs	15
2.4.2 Dijkstra's Algorithm	16
2.4.3 A* Algorithm	16
2.4.4 Bellman-Ford Algorithm	16

2.4.5	Temporal Graphs	17
2.5	Sampling-based Path Planning Algorithms	18
2.5.1	PRM	18
2.5.2	RRT	19
2.5.3	RRT*	19
2.5.4	Time as a Smoothness Metric	20
3	Literature Review	21
3.1	Continuum Robotics	21
3.1.1	Materials	22
3.1.2	Actuation	23
3.1.3	Applications	23
3.1.4	Challenges	23
3.2	Seminal Sampling-Based Planners	24
3.2.1	Problem Description	24
3.2.2	PRM	24
3.2.3	RRT	25
3.2.4	Asymptotic Optimality	26
3.2.5	Seminal Variants	27
3.3	RRT Variants	28
3.3.1	Redundant Manipulators	29
3.3.2	Continuum Manipulators	29
3.4	PRM Variants	29
3.4.1	Continuum Manipulators	29
3.5	Dynamic Planning	30
3.5.1	Reactive Dynamic Planners	31
3.5.2	Non-Reactive Dynamic Planners	31
3.6	Artificial Potential Fields	31
3.6.1	Artificial Potential Fields	32
4	Graph-Theoretic Planning	34
4.1	Look-up Table Approach	34
4.1.1	Cubes-Graph	35
4.1.2	Offline Look-up Table	35
4.1.3	Auxiliary Graph	35
4.1.4	Path Planning	35
4.1.5	Test Environment	37
4.1.6	Results	38
4.2	Inverse Kinematics for Improved Configuration Generation	40
4.2.1	Cube Configuration Generation	41
4.2.2	Path Planning	41
4.2.3	Time as a Metric	42
4.2.4	Test Environment	42
4.2.5	Trajectory Smoothness without Obstacles	43
4.2.6	Trajectory Smoothness with Obstacles	45

5	Anticipatory Path Planning	46
5.1	Anticipatory Path Planning Algorithm	46
5.1.1	Cubes-Graph, Cube Path, and Layered Graph	47
5.1.2	Smoothness Measure	47
5.1.3	Temporal Model and Obstacle Prediction	48
5.1.4	Temporal Cubes-Graph and Path	48
5.1.5	Path Planning	49
5.1.6	Other Approaches	50
5.1.7	Test Environment	51
5.1.8	Results	52
6	Rapidly-Exploring Random Tree Path Planning	54
6.1	$\mathcal{W}$ -space RRT*	54
6.1.1	$\mathcal{W}$ -space - $\mathcal{C}$ -space Projection Via Jacobians	55
6.1.2	Null Space	55
6.1.3	$\mathcal{W}$ -space Exploration	56
6.1.4	The RRT* Algorithm	57
6.1.5	$\mathcal{C}$ -space RRT*	58
6.1.6	Other Approaches	59
6.1.7	Comparison of Approaches	59
6.1.8	Category 1: Obstacle Avoidance	60
6.1.9	Category 2: Trajectory Tracking	60
6.2	Dynamic $\mathcal{W}$ -space RRT*	62
6.2.1	The Planner	62
6.2.2	Obstacle Inflation	63
6.2.3	Distance Record	63
6.2.4	Backup Counter	64
6.2.5	Test Environment	64
6.2.6	Results	66
6.2.7	Continuum Arm Prototype Testing	67
7	Probabilistic Roadmap Path Planning	71
7.1	Temporal $\mathcal{J}$ -space PRM	71
7.1.1	Temporal Local Planner	71
7.1.2	Temporal Graphs for Local Planning	72
7.1.3	Dynamic $\mathcal{J}$ -space Planning	74
7.1.4	Test Environment	77
7.1.5	Results	78
8	Sampling-Based Planning for Dual Continuum Arms	81
8.1	State Refiners	82
8.1.1	General Refiner Framework	82
8.1.2	Section Jacobian and Local IK	82
8.1.3	Refinement Loop	83
8.1.4	Refining Strategies	84
8.1.5	Test Environment	86
8.1.6	Results	86

8.2	Dual-Arm IK Methods	87
8.2.1	Simultaneous IK	87
8.2.2	Sequential IK	87
8.2.3	Midpoint-Orientation IK	88
8.2.4	Test Environment	88
8.2.5	Results	88
8.3	Dual-Arm RRT*	89
8.3.1	RRT* Formulation	89
8.3.2	Sampling Strategy	90
8.3.3	Growth Strategy	90
8.3.4	Multi-Root Seeding	91
8.3.5	Multi-Step Planning for Tracing	92
8.4	Dual-Arm PRM	92
8.4.1	PRM Formulation	93
8.4.2	Many-Goal Strategy	93
8.4.3	Edge Connection with Refinement	93
8.5	Planner Evaluation	94
9	Conclusion	99
9.1	Overview	99
9.2	Summary of Work	99
9.2.1	Graph-Theoretic Planning for Continuum Arms	99
9.2.2	Improved Configuration Generation via Inverse Kinematics	100
9.2.3	Anticipatory Path Planning in Dynamic Environments	100
9.2.4	$\mathcal{W}$ -space RRT* for Static Obstacle Avoidance and Tracing	100
9.2.5	Dynamic $\mathcal{W}$ -space RRT*	101
9.2.6	Temporal $\mathcal{J}$ -space PRM for Dynamic Planning	101
9.2.7	Dual-Arm State Refinement and Inverse Kinematics	101
9.2.8	Sampling-Based Planning for Dual Continuum Arms	102
9.3	Future Research Directions	102
	Bibliography	103

List of Figures

2.1	Left: A multisection continuum arm at rest, Middle: A multisection continuum arm shown actuated	7
2.2	(A) Schematic of a general n -section continuum arm and (B) cross-section of a continuum section showing the actuator arrangement and inextensible backbone in the neutral axis.	8
2.3	Left: The curve parameters and their relationship to the arm's shape. Right: Schematic of a multisection continuum arm with 3 sections. Note that, as L_i is constant, $\lambda_i = \frac{L_i}{\phi_i}$	10
2.4	Actuator range of the reduced mapping.	10
2.5	An example graph with labeled vertices and edges. The nodes are labeled with the letters A-I.	15
2.6	An illustration of a temporal graph with $\mu = 1$ and $d = 0$ and its corresponding static graph [17].	17
3.1	A collection of continuum arms is shown with a label at the bottom right of each. (A) A multisection continuum arm constructed of soft pneumatic muscles[32]. (B) A soft planar continuum arm constructed of pneumatic actuators[33]. (C/D) A tendon-driven continuum arm with a novel magnetic locking scheme shown in [34] and [35], respectively. (E) A multisection hybrid continuum arm made with pneumatic muscles and a rigid inextensible backbone (from the authors of [1]). (F/G) A concentric tube robot with 3 concentric tubes shown in [36] and [37], respectively. (H) A continuum arm using a spine with expandable air chambers [38]. (I) A tendon-driven continuum arm made from inexpensive parts [39].	22
3.2	A comparison between a rapidly-exploring random tree (Left) and a probabilistic roadmap (Right). The starting configuration is shown in green and the target configuration is shown in red. A single obstacle is shown as a grey diamond.	26
3.3	A tree generated by the RRT* algorithm. This image is taken from multimedia materials associated with [21].	27
3.4	An image of all of the feasible configurations generated during the construction of a PRM for concentric tube robots. These configurations are generated rapidly using a parallel IK system and display configurations that are not colliding with the exterior wall. This image is taken from [77]	30
3.5	A depiction of a continuum manipulator being influenced by an APF. The tip of the arm and the obstacles (orange) share the same polarization (purple) so they are repulsed. The tip and the goal tip position share opposing polarization (purple vs yellow) so the two are attracted to each other.	32

4.1	Illustrative test case where both the Look-up Table approach (Right) and the IK (Left) successfully found solutions. No obstacles were present in the workspace. The top point is the starting point and the black dot is the end point.	39
4.2	An example test case where the Look-up Table approach (Left) converged to a solution, but the IK (Right) was unsuccessful. No obstacles were present in the workspace. The rightmost point is the beginning point and the leftmost black dot is the ending point . . .	39
4.3	One scenario in which the Look-up Table approach (right) was able to find a path from the starting point to the end point amidst obstacles whereas the IK (left) planner failed.	40
4.4	A comparison for the cube (26,26,15) of the configurations stored in the Look-up Table vs. the Cube Configuration Generation Method	41
4.5	A 3D view of the Sequential-IK approach simulated on the circle, rotated cube, and figure-8 trajectory. The traced path is shown in black and the target trajectory is shown in blue.	44
4.6	A top down view of the results from the Dynamic Simulator for a circle, rotated cube, and figure-8 trajectory. The trajectory traced by the dynamic simulator (red) is displayed over the calculated trajectory (blue).	45
5.1	A comparison of trajectories generated by the T_2 (top) and T_3 (bottom) approaches for the same trial. The trajectory is shown at 5 steps of progress, with step 1 and step 5 showing the start and goal states, respectively. Objects are shown in orange and the goal tip position is shown in purple. Note that the H , H_s , and PF approaches could not generate a valid trajectory for this trial.	52
6.1	A comparison of a trajectory produced by the $\mathcal{W}$ -space RRT* (left) and $\mathcal{C}$ -space RRT* (right) for the obstacle avoidance task. The target point is shown in purple.	61
6.2	A comparison of a trajectory traced by the Look-up Table (top left), the Refined-IK (top right), $\mathcal{C}$ -space RRT* (bottom left), and $\mathcal{W}$ -space RRT* (bottom right).	68
6.3	A path generated by WRRT*D performed in Environment 3. Frames are shown of the initial position (1), 33% progress (2), 66% progress (3), and the final position (4)	69
6.4	Path planning progression of the continuum arm prototype under (a) single and (b) two obstacles.	70
7.1	An illustration of a temporal graph with all $\lambda = 1$. The edges and times corresponding to an earliest arrival path from vertex (a) to vertex (l) are highlighted in red. The equivalent edge stream representation is also shown.	72
7.2	An example of the TJPRM approach creating a plan through the Orbit Environment (1). The trajectory is shown at 5 steps of progress, with step 1 and step 5 showing the start and goal states, respectively. In the accompanying video, this is Orbit Trial 2. . . .	78
7.3	An example of the TJPRM approach creating a plan through the Hurricane Environment (2). The trajectory is shown at 5 steps of progress, with step 1 and step 5 showing the start and goal states, respectively. In the accompanying video, this is Hurricane Trial 2.	79
7.4	An example of the TJPRM approach creating a plan through the Random Environment (3). The trajectory is shown at 5 steps of progress, with step 1 and step 5 showing the start and goal states, respectively. In the accompanying video, this is Random Trial 3. .	79

8.1	A path generated by the Dual-Arm PRM (top) and RRT* (bottom) approaches for a trial with 6 obstacles. The target midpoint is shown in purple. Frames are shown of the initial position (1), 25% progress (2), 50% progress (3), 75% progress (4), and the final position (5).	95
8.2	A path generated by the Dual-Arm PRM approach for a trial with 6 obstacles. The target midpoint is shown in purple. Frames are shown of the initial position (1), 25% progress (2), 50% progress (3), 75% progress (4), and the final position (5). Note that the Dual-Arm RRT* approach could not produce a path for this trial.	95
8.3	An example of a circle on the XY plane traced by the PRM (left) and RRT* (right) planners. The target midpoint is shown in purple. The midpoint trajectory is shown in black, the Arm 1 trajectory is shown in red, and the Arm 2 trajectory is shown in blue.	97
8.4	An example of a circle on the XZ plane traced by the PRM (left) and RRT* (right) planners. The target midpoint is shown in purple. The midpoint trajectory is shown in black, the Arm 1 trajectory is shown in red, and the Arm 2 trajectory is shown in blue.	98

List of Tables

2.1	Notation used in Section 2.2	12
3.1	Seminal Sampling-Based Path Planners	28
4.1	Summary of the numerical results.	38
4.2	Summary of Time Results for small shapes (above line) and large shapes (below line) . .	44
4.3	Summary of Time Results for trajectories with obstacles	45
5.1	Simulations of obstacles between start and end points	52
6.1	Summary of Computation Time for Obstacle Avoidance (For each approach, the mean DynSim-time, the corresponding variance σ^2 , and the mean computation time for 100 random trials in seconds.)	60
6.2	Summary of DynSim-Time Results for small shapes (above line) and large shapes (below line). (All numbers represent the DynSim-time in seconds. Variance is included for the RRT* approaches.)	61
6.3	Summary of Computation Time Results for small shapes (above line) and large shapes (below line). (All numbers represent computation time in seconds.)	62
6.4	Orbit Environment	66
6.5	Wave Environment	66
6.6	Random Environment	66
7.1	Orbit Environment	79
7.2	Hurricane Environment	79
7.3	Random Environment	80
8.1	Refiner benchmark results under the $\mathcal{C}$ -space and $\mathcal{J}$ -space representations.	86
8.2	Comparison of inverse kinematics methods under the $\mathcal{J}$ -space and $\mathcal{C}$ -space representations. .	89
8.3	Obstacle-avoidance results for the dual-arm planners. (Success is reported as a percentage, and time is reported in seconds.)	96
8.4	Shape-tracing results for the dual-arm planners. (Success is reported as a percentage, time is reported in seconds, and — indicates that no successful trial was obtained.) . . .	98

List of Algorithms

1	Layered $\mathcal{J}$ -space Path Planning	36
2	Global-IK Subroutine	42
3	Sequential-IK Subroutine	43
4	Temporal cubes-graph Construction	49
5	Temporal Shortest Path Approach	50
6	$\mathcal{W}$ -space RRT*	58
7	Horizon	64
8	$\mathcal{W}$ -space RRT* Dynamic	65
9	Temporal Local Planner	73
10	Temporal $\mathcal{J}$ -space PRM (TJPRM)	77
11	General Dual-Arm Refinement Loop	83
12	Dual-Arm RRT*	92
13	Dual-Arm PRM	94

Acknowledgments

First and foremost, I offer my deepest gratitude to my doctoral advisor, Dr. Iyad Kanj of DePaul University, and to Dr. Isuru Godage of Texas A&M University. Their mentorship, generosity with their time, and support have shaped every stage of this work. Dr. Kanj's guidance, patience, and encouragement were instrumental throughout my doctoral studies, and his influence has profoundly shaped both this dissertation and my growth as a researcher. Dr. Godage's insight, collaboration, and continued investment in my development have likewise had a lasting impact on both this research and my broader academic trajectory. I am deeply grateful to both of them for the confidence they placed in me and for the standards of scholarship they consistently encouraged me to pursue.

I am also sincerely grateful to the members of my dissertation committee, Dr. Hunter B. Gilbert of Louisiana State University, Dr. Umer Huzaifa of DePaul University, and Dr. David Hanley of DePaul University, for their thoughtful feedback, insightful questions, and constructive criticism throughout this process. Their expertise and perspective strengthened this dissertation in many important ways.

I gratefully acknowledge the Jarvis College of Computing and Digital Media and the School of Computing at DePaul University for their financial support through the Computing and Digital Media Award for doctoral studies, as well as the support I received through the GAANN fellowship. I would also like to thank Dr. Amber Settle for her mentorship and the generous time she devoted to helping me when I began teaching.

I also owe a great deal to Dr. Dimuthu Arachchige for his support, frequent collaboration, guidance, and friendship. I thank Jiahao Deng, with whom I wrote my first publication. I am likewise thankful to Dr. Bradley Dirks, Dr. Skylar Wurster, and Dr. Chase Bloch, whose friendship, collaboration, and conversations encouraged and inspired me throughout this journey. To my friends and fellow DePaul alumni, Giancarlo and Alan, and to my long-time friends, Lee and Ben, I am grateful for their constant support and enduring friendship.

I am deeply grateful to my parents, Lynne and Hans, and to my brothers, Kyle and Tyler, for their unwavering belief in me, their encouragement, and their constant support. Without them, this dissertation would not have been possible. Finally, I am thankful to my fiancée, Laura, whose patience, encouragement, and steadfast support carried me through to the finish line.

This dissertation was only possible because of the support, guidance, and generosity of so many people in my life. I am profoundly grateful to all of you.

Chapter 1

Introduction

1.1 Background

Path planning is the problem of generating a sequence of “states” that navigate a robotic device from a starting state to a goal state, avoiding obstacles and adhering to the robot’s physical constraints. This challenge, often referred to as the piano mover’s problem, spans numerous domains, each presenting unique challenges and specific requirements due to the diversity of robotic models. As robotic models have advanced, a wide variety of path planning schemes have been attempted. Though closed-form and exact methods were initially explored for use in robotics, their applicability proved limited, as closed-form solutions are only available for restricted models and exact methods tend to become computationally impractical in complicated environments or real-time settings. Inverse kinematic (IK) based approaches showed early promise in simple robotic models, but as robots have grown more complicated, standalone IK-based planning has become increasingly unreliable, and practical IK solutions often depend on numerical techniques and optimization. Among the various strategies employed to address this problem, sampling-based planning has emerged as particularly promising and widely adopted. These approaches rely on random sampling configurations within the robot configuration space ($\mathcal{C}$ -space) to explore the state space of the robot and produce a sequence of states defining a trajectory. The rapidly-exploring random tree (RRT) and probabilistic roadmap (PRM) path planning algorithms stand out as significant contributions in this area, for which much additional work has been performed. These approaches have been adapted to a wide variety of robotic models to perform planning in a great number of domains. These approaches have also been augmented in a vast number of ways to improve the results created by these algorithms through modifications to sampling and exploration.

Path planning is broadly categorized into mobile planning and manipulation planning. Mobile planning deals with scenarios where a robot must navigate through an environment, focusing on movement within a space. Classic examples of such devices include autonomous vehicles, drones, and automated vacuum cleaners. For these devices, a plan consists of a series of spatial positions leading the robot from its starting point to its destination. Conversely, manipulation planning involves robotic devices with appendages fixed to a base, typically engaged in moving or interacting with objects. This category includes robotic arms used in manufacturing, surgery, and other applications requiring precise object manipulation. Traditionally, robots have been made from rigid materials, offering well-defined motion and composition similar to other machines with rigid bodies. However, advances in materials science have paved the way for innovative construction methods and actuation techniques. Notably, the use of deformation and bending has proven to be highly effective in applications demanding

great dexterity and passive safety. Continuum manipulators, which employ a deformable design for movement, exemplify this innovation, extending the capabilities of robotic devices to tasks that are difficult for conventional rigid-link robots.

1.2 Motivation for Continuum Arm Robots

Conventional manipulators, characterized by rigid links and discrete joints, contrast sharply with continuum manipulators, which achieve motion through continuous bending and deformation. This fundamental difference gives continuum devices a high degree of shape flexibility, allowing them to attain a wide variety of configurations for complex tasks without requiring specialized end-effectors. Such flexibility suggests strong potential in obstacle avoidance and whole-body manipulation. However, it also introduces additional complexity into their kinematics, particularly in inverse kinematics and path planning.

Continuum manipulators may be constructed from materials spanning a spectrum from soft to rigid, with different designs offering different trade-offs in flexibility, dexterity, and payload capacity. Many of these devices are also bio-inspired, drawing conceptual inspiration from biological appendages such as elephant trunks or octopus tentacles. To achieve their motion, continuum manipulators employ a variety of actuation strategies, including tendon-driven systems, pneumatic actuation, and concentric tube designs. These actuation methods enable bending, extension, contraction, and twisting, and multiple continuum sections may be joined in series to provide even greater dexterity and shape diversity.

As a result of these properties, applications of continuum robotics are broad and growing, particularly in areas requiring high flexibility and safety, such as surgery, space exploration, and human-robot collaboration. Their ability to maneuver around obstacles while reducing the risk of harmful contact makes them well suited to operation in constrained or sensitive environments. These same characteristics also make continuum arms attractive for manipulation tasks that benefit from whole-body compliance and the ability to reach around clutter.

At the same time, continuum manipulators present notable challenges in modeling, control, and path planning. Their compliant nature is more difficult to model than that of traditional manipulators composed of rigid links and joints. Moreover, these devices rely on structural deformation for motion, which leads to highly nonlinear kinematics. In the model considered in this dissertation, approximating each section as a circular arc also gives rise to problematic cases, particularly near the straight-arm pose. As a result, during path planning and numerical inverse kinematic computation, it may be difficult to determine when the arm is approaching a local minimum. In addition, singularities and local minima in this setting are not necessarily isolated points, but may instead occupy broader singular regions or neighborhoods.

Despite these challenges, continuum robotics holds strong promise for enabling robotic systems to operate more effectively in demanding environments. Ongoing research seeks to harness the flexibility, dexterity, and safety of these devices while addressing the substantial difficulties they pose in control and planning.

1.3 Motivation for Path Planning of Continuum Arms

Though research into the design and construction of continuum manipulators has seen rapid development in recent years, path planning for such devices remains largely unresolved. Continuum arms present a distinct combination of planning difficulties arising from their compliant structure, nonlinear kinematics,

and high-dimensional state representations. In particular, the redundancy, and often hyper-redundancy, of these systems can make many traditional planning methodologies difficult to apply reliably or efficiently.

A central difficulty lies in the use of inverse kinematics (IK) to relate the configuration space ($\mathcal{C}$ -space) of the robot to its workspace ($\mathcal{W}$ -space). For continuum arms, straightforward numerical IK can be particularly unreliable, since numerical solutions may encounter local minima, singular regions, or knotting-like behaviors in which the arm folds into undesirable shapes. As a result, planning strategies that rely too directly on IK, such as inverse kinematic chains or potential-field-style methods, may suffer in reliability or path quality. These challenges motivate the development of path planning methods that can leverage kinematic tools when useful without depending exclusively on standalone IK.

Among the most promising families of methods for this purpose are sampling-based planners. By building connectivity through random sampling in the robot’s state space, these approaches are often better suited to high-dimensional planning problems than methods that rely on explicit global parameterizations or repeated inverse kinematic solves. Two of the most prominent methods are Probabilistic Roadmaps (PRM) and Rapidly-exploring Random Trees (RRT). PRM utilizes roadmap construction and reuse across multiple queries, while RRT incrementally explores the space from a starting state toward a goal. Their asymptotically optimal variants, PRM* and RRT*, further improve path quality over time.

For continuum arms, sampling-based approaches are especially attractive because they can be adapted to different state representations and combined with numerical kinematic tools in a flexible way. However, their application to continuum arms is not straightforward. Configuration generation, local planning, collision checking, and path smoothness all require special consideration for deformable, highly nonlinear robotic models. Accordingly, effective path planning for continuum arms requires methods capable of addressing path tracing, static obstacle avoidance, and dynamic obstacle avoidance in a coherent manner.

In addition, many practical manipulation tasks are naturally cooperative rather than single-arm. Two continuum arms can enlarge the reachable workspace, maintain a desired geometric relationship between contact points, and coordinate around obstacles while performing tracing or manipulation-like tasks that would be difficult for a single arm alone. At the same time, this added capability introduces substantial planning complexity, since valid motions must satisfy not only obstacle avoidance requirements, but also inter-arm constraints, self-collision avoidance, and coordination between the two arms. These considerations make dual-arm continuum planning a natural and important extension of the single-arm planning problem.

1.4 Research Objectives

This dissertation aims to develop effective and reliable path planning methods for continuum arms in order to broaden the practical use of such devices. To this end, it presents a collection of planning approaches that address several central challenges in continuum arm planning, including nonlinear and unreliable inverse kinematics, high-dimensional state spaces, path smoothness, and dynamic obstacles.

A primary objective of this work is to improve the reliability of path planning for continuum arms beyond what is typically achievable through straightforward IK-based strategies alone. This objective is pursued through methods that incorporate graph-theoretic structure and kinematic information in ways that reduce dependence on standalone inverse kinematics while preserving the ability to generate high-quality trajectories. A related objective is to improve trajectory smoothness without sacrificing reliability, particularly in tracing and obstacle avoidance tasks.

Another central objective of the dissertation is to develop and adapt sampling-based planning methods for continuum arms. Because these methods are well suited to high-dimensional planning problems, they provide a powerful framework for addressing the complexity of continuum arm state spaces. In particular, this work develops RRT* and PRM-based approaches, together with the kinematic and local-planning tools needed to make them effective for continuum arm models in both static and dynamic settings.

This work also addresses planning in dynamic environments. In particular, it investigates anticipatory and temporal planning methods that incorporate prediction and temporal reasoning in order to navigate moving obstacles while maintaining trajectory quality. These dynamic-planning ideas are studied both in their own right and in conjunction with sampling-based methods.

Finally, the dissertation considers the extension from single-arm to dual-arm continuum planning. In this setting, the problem is not only to generate feasible trajectories, but also to maintain coordination between two arms while satisfying inter-arm constraints and avoiding collisions with obstacles, the environment, and the arms themselves. This motivates the development of methods for coupled-state generation, refinement, inverse kinematics, and coordinated planning in the dual-arm case.

Taken together, these objectives reflect a progression from reliable planning for single continuum arms, to sampling-based and dynamic planning in more complex environments, and ultimately to coordinated planning for dual-arm continuum systems.

1.5 Dissertation Structure

The structure of this dissertation is outlined as follows:

Chapter 1 sets the stage for the research by providing the background, motivation, research objectives, contributions of the dissertation, and related publications. It also outlines the overall structure of the dissertation, offering readers a roadmap of what to expect.

Chapter 2 presents the foundational material for the dissertation, including the prototype arm and its kinematic model, the $\mathcal{C}$ -space, $\mathcal{J}$ -space, and $\mathcal{W}$ -space representations, and the path planning problem in both static and dynamic settings. It also reviews the graph-theoretic and sampling-based planning concepts used throughout the chapters that follow.

Chapter 3 provides a review of continuum arms, sampling-based planning, and path planning algorithms for continuum arms. This includes discussions on redundant manipulators, challenges in high-dimensional exploration, and the application of RRT/PRM strategies to continuum systems.

Chapter 4 details graph-theoretic planning approaches tailored to the continuum arm model, incorporating graph-theoretic principles and an offline pre-computation phase. It also introduces inverse-kinematics-based configuration generation to reduce offline computation and improve path smoothness.

Chapter 5 addresses dynamic path planning through an anticipatory approach. This method employs a temporal graph together with a specialized subset of the look-up table to forecast obstacle movement and take proactive planning actions.

Chapter 6 presents sampling-based planning methods based on RRT*. In particular, it develops a $\mathcal{W}$ -space RRT* framework for smooth tracing and static obstacle avoidance, and then extends this framework to dynamic environments.

Chapter 7 presents a temporal $\mathcal{J}$ -space PRM for dynamic path planning. This chapter develops a temporal local planner and a temporal-graph-based formulation for validating motion under moving obstacles.

Chapter 8 extends the planning problem to a dual-arm continuum-robot system. It introduces state refinement methods and dual-arm inverse kinematics methods, and then develops dual-arm RRT* and PRM planners for coordinated tracing and obstacle avoidance.

Chapter 9 concludes the dissertation by summarizing the main results, revisiting the progression of methods developed throughout the work, and discussing directions for future research.

1.6 Dissertation Contributions

This dissertation advances the field of path planning for continuum robotic arms through the following contributions:

1. It develops a graph-theoretic look-up-table approach for continuum arm path planning, emphasizing reliability and robustness in the generation of feasible trajectories.
2. It augments this graph-theoretic framework with rapid inverse-kinematics-based configuration generation, reducing offline computation while improving the diversity of reachable poses and the smoothness of the resulting motion.
3. It introduces an anticipatory dynamic planning method that combines offline planning structure with temporal reasoning and obstacle prediction in order to navigate dynamic environments.
4. It develops a $\mathcal{W}$ -space RRT* framework for continuum arms that uses Jacobian-based projection and specialized $\mathcal{W}$ -space exploration to achieve smooth tracing and effective static-obstacle avoidance.
5. It extends this sampling-based methodology to dynamic environments, yielding a reactive planning method capable of adapting to moving obstacles.
6. It develops a temporal $\mathcal{J}$ -space PRM for dynamic planning, including a temporal local planner and temporal-graph-based reasoning for validating motion through changing environments.
7. It introduces state refinement methods and dual-arm inverse kinematics methods for generating valid coupled states for a dual-arm continuum robot system.
8. It develops dual-arm sampling-based planners, including PRM- and RRT*-based approaches, for coordinated tracing and obstacle avoidance under dual-arm constraints.

Collectively, these contributions show that smooth and reliable path planning for continuum arms can be achieved across single-arm and dual-arm systems, and across static and dynamic environments.

1.7 List of Research Articles

- **Peer-reviewed Journal Articles**

1. B. H. Meng et al., “RRT*-based path planning for continuum arms,” *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6830–6837, 2022, Presented at IROS 2022 in Kyoto. DOI: [10.1109/LRA.2022.3174257](https://doi.org/10.1109/LRA.2022.3174257)

- **Peer-reviewed Conference Articles**

2. B. H. Meng et al., “Anticipatory path planning for continuum arms in dynamic environments,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 7815–7820. DOI: [10.1109/ICRA48506.2021.9560952](https://doi.org/10.1109/ICRA48506.2021.9560952)
3. B. H. Meng et al., “Smooth path planning for continuum arms,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 7809–7814. DOI: [10.1109/ICRA48506.2021.9560982](https://doi.org/10.1109/ICRA48506.2021.9560982)
4. B. H. Meng et al., “Path planning for continuum arms in dynamic environments,” in *2024 IEEE 7th International Conference on Soft Robotics (RoboSoft)*, 2024, pp. 900–905. DOI: [10.1109/RoboSoft60065.2024.10521950](https://doi.org/10.1109/RoboSoft60065.2024.10521950)

- **In-progress/accepted Journal Articles**

5. B. H. Meng et al., “Temporal joint-space planning for continuum arms in dynamic environments,” *IEEE Robotics and Automation Letters*, 2026

- **Co-authored Articles**

6. J. Deng et al., “Near-optimal smooth path planning for multisection continuum arms,” *CoRR*, vol. abs/1812.03615, 2018. arXiv: [1812.03615](https://arxiv.org/abs/1812.03615). [Online]. Available: <http://arxiv.org/abs/1812.03615>

Chapter 2

Preliminaries

2.1 Prototype Arm



Figure 2.1: Left: A multisection continuum arm at rest,
Middle: A multisection continuum arm shown actuated

In Figure 2.1 there is an image of the prototype continuum arm for which we develop the proposed path planning algorithms. Figure 2.2 shows the schematic of an n -section continuum arm, in which the sections are identical and are enumerated starting from the base section (index 1) attached to the task-space coordinate system, $\{O\}$. The i^{th} continuum section, shown in Figure 2.2, is actuated by three extending-mode pneumatic muscle actuators (PMA) that are affixed on rigid terminating plates at either end at distance $r_i \in \mathbb{R}^+$ from the neutral axis, and that are $\frac{2\pi}{3}$ radians apart. Each continuum section has an inextensible rigid chain of length $L_i \in \mathbb{R}^+$ in its neutral axis. The unactuated length of the PMAs is $l_i \in \mathbb{R}^+$, and the maximum length change is $l_{max} \in \mathbb{R}^+$. The actuator length vector of the continuum section is $\mathbf{q}_i = [l_{i1}, l_{i2}, l_{i3}]^T$, where $l_{ij} \in [0, l_{max}] \ \forall j \in \{1, 2, 3\}$. Individual continuum

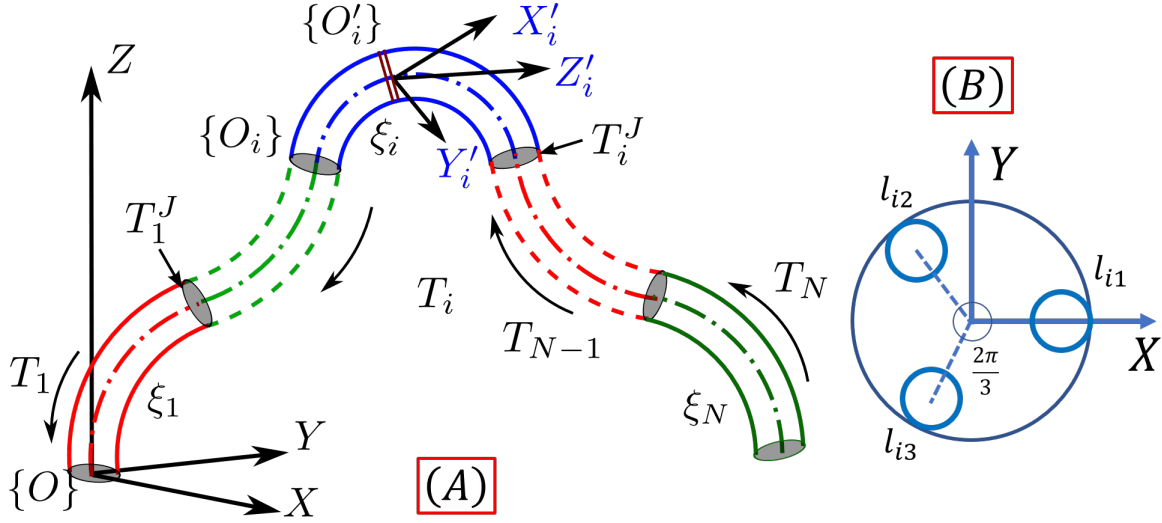


Figure 2.2: (A) Schematic of a general n -section continuum arm and (B) cross-section of a continuum section showing the actuator arrangement and inextensible backbone in the neutral axis.

sections are joined together using rigid joints that introduce $\sigma \in \mathbb{R}_0^+$ linear displacement along the $+Z$ axis and $\gamma \in \mathbb{R}_0$ angular displacement about the $+Z$ axis of $\{O_i\}$.

Along the length of the continuum section, PMAs are constrained to maintain r_i clearance to the neutral axis. Consequently, any differential length changes of PMAs due to different pressure inputs cause the continuum section to bend in a circular arc or extend (when length changes are equal). The subsequent derivations rely on the assumption that the continuum sections bend in circular arc shapes. This is a reasonable assumption. As shown in [7] and [8], under operating conditions where the continuum arms are not subjected to large external forces, continuum sections satisfy this condition.

2.1.1 Derivation of a Reduced DoF Mapping

The presence of an inextensible backbone introduces an over-constrained system. Unlike the pneumatically-actuated continuum arms without such a backbone (see [8], [9]), for any i^{th} section, the length changes are governed by

$$\sum_k l_{ik} = 0, \quad (2.1)$$

where physically this means that any extension of actuators results in the contraction of the others.

As a result, without loss of generality, the complete kinematic model for a continuum section has to be derived from the principles of force/moment balance. However, in this work, by utilizing the above constraint, we will introduce a reduced, two-DoF model. In this approach, we will disregard the third actuator, l_{i3} , and instead use l_{i1}, l_{i2} , and the backbone length to derive a complete mapping. The objective of our approach is to reduce the overall DoF of the continuum arm, without betraying the physical behavior.

By definition, the bending of a continuum section can be described by three curve parameters, the radius λ_i of the circular arc, the angle ϕ_i subtended by the arc, and the angle θ_i between the $+X$ axis and the bending plane [7]. Figure 2.3 shows the schematic of the i^{th} section bending in a circular arc with the arc parameters λ_i, ϕ_i , and θ_i .

Utilizing the arc geometry, we can derive the following relations between the actuator length changes and the curve parameters. A similar and detailed exposition of the methodology, developed for continuum arms without constraining backbones, is given in [7].

$$\begin{aligned}
L_i &= \lambda_i \phi_i, \\
L_i + l_{i1} &= \left(\lambda_i - r_i \sin \left(\theta_i - \frac{\pi}{6} \right) \right) \phi_i, \text{ and} \\
L_i + l_{i2} &= \left(\lambda_i - r_i \sin \left(\theta_i - \frac{5\pi}{6} \right) \right) \phi_i.
\end{aligned} \tag{2.2}$$

By manipulating the relations given in (2.2), we can derive the curve parameters as functions of the length changes as

$$\begin{aligned}
\theta_i &= \arctan \left(l_{i2} - l_{i1}, \sqrt{3}(l_{i1} + l_{i2}) \right), \\
\phi_i &= \frac{2\sqrt{l_{i1}^2 + l_{i1}l_{i2} + l_{i2}^2}}{r_i\sqrt{3}}, \text{ and} \\
\lambda_i &= \frac{L_i}{\phi_i}.
\end{aligned} \tag{2.3}$$

Using (2.3), the homogeneous transformation matrix (HTM) for the i^{th} section can be derived as

$$\begin{aligned}
\mathbf{T}_i &= \mathbf{R}_Z(\theta_i) \mathbf{P}_X(\lambda_i) \mathbf{R}_Y(\xi_i \phi_i) \mathbf{P}_X(-\lambda_i) \mathbf{R}_Z(-\theta_i) \cdots \\
\mathbf{P}_Z(\sigma_i) \mathbf{R}_Z(\gamma_i) &= \begin{bmatrix} \mathbf{R}_i & \mathbf{p}_i \\ \mathbf{0} & 1 \end{bmatrix},
\end{aligned} \tag{2.4}$$

where $\mathbf{P}_X \in \mathbb{SE}^3$, $\mathbf{R}_Z \in \mathbb{SO}^3$, and $\mathbf{R}_Y \in \mathbb{SO}^3$ are HTM that denote translation along the +X axis, rotation about the +Z and +Y axes, respectively. $\mathbf{R}_i : (\mathbf{q}_i, \xi_i) \mapsto \mathbb{SO}^3$ is the resultant rotation matrix and $\mathbf{p}_i : (\mathbf{q}_i, \xi_i) \mapsto \mathbb{R}^3$ is the position vector. The scalar ξ_i denotes any point along the neutral axis, where $\xi_i = 0$ is the base where $\{O'_i\} \equiv \{O_i\}$ and $\xi_i = 1$ is the tip of the continuum section. We then apply the 18th order multivariate Taylor series expansion on the terms of (2.4) to obtain a numerically efficient and stable modal form of the HTM (see [7]).

2.1.2 Actuator Range Derivation

In order to preserve the full reachable motion of the continuum section, the two actuator ranges have to be modified. For instance, the actual PMAs only extend during operation, but as we have omitted l_{i3} , to describe the motion contributions of l_{i3} , as per (2.1), we will modify the ranges of l_{i1} and l_{i2} . Under the premise that PMA extension is proportional to the differential pressure input, using the resulting curve parameters given by (2.3) with the length constraint (2.1), one can easily derive the correct ratio of pressures given to PMAs.

Noting the maximum bending, $\phi_{i,max} = \pi$, of the continuum sections, we can find the valid actuator combinations that result in $\phi_i \leq \pi$, as shown in Figure 2.4. We identified this valid length change range by comparing the actuator combinations in the range $[-0.04, 0.04]$ for $L_i = 0.15 \text{ m}$ and $r_i = 0.0125 \text{ m}$. The resulting valid actuator range is given by the rotated ellipse

$$g(l_{i1}, l_{i2}) = al_{i1}^2 + bl_{i1}l_{i2} + cl_{i2}^2 + dl_{i1} + el_{i2} + f, \tag{2.5}$$

where $a = -0.5766$, $b = 0.5789$, $c = -0.5766$, $d = 0$, $e = 0$, and $f = 0.0007$. This means that if $g(l_{i1}, l_{i2}) \geq 0$ in (2.5), then the value pair (l_{i1}, l_{i2}) is a valid combination, and results in $\phi_i \leq \pi$.

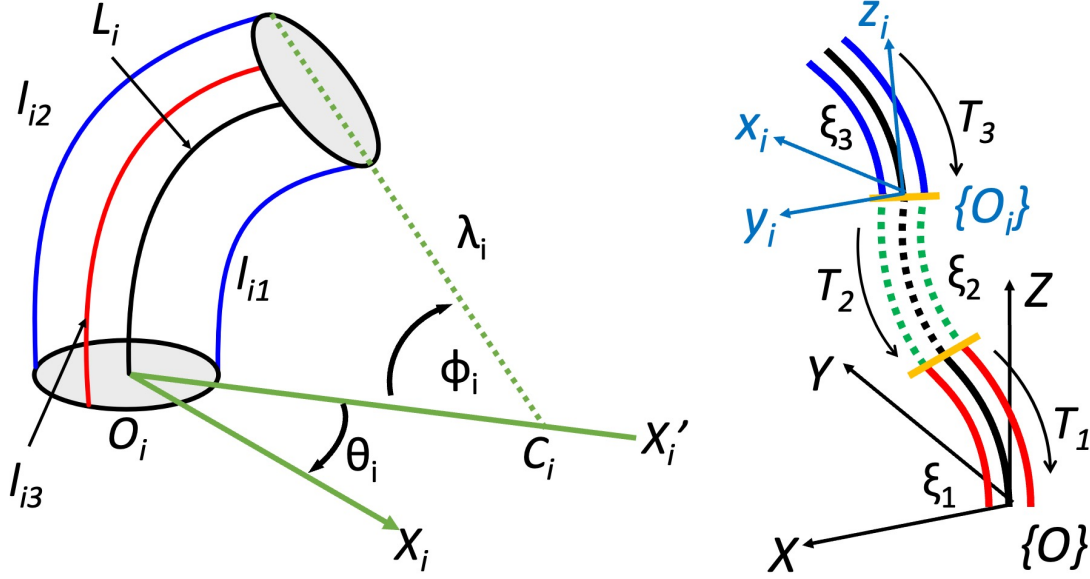


Figure 2.3: Left: The curve parameters and their relationship to the arm's shape. Right: Schematic of a multisection continuum arm with 3 sections. Note that, as L_i is constant, $\lambda_i = \frac{L_i}{\phi_i}$.

2.1.3 Kinematic Model

Having obtained the section-level HTM, we now extend this model to the full multisection arm through serial composition. Employing the continuum section HTM given in (2.4) and the kinematics principles of serial robot chains, the HTM of the i^{th} section with respect to the workspace coordinate system $\{O\}$, $\mathbf{T}^i : (q^i, \xi_i) \mapsto \mathbb{SE}^3$, is given by

$$\mathbf{T}^i = \prod_{k=1}^i \mathbf{T}_k = \begin{bmatrix} \mathbf{R}^i & \mathbf{p}^i \\ \mathbf{0} & 1 \end{bmatrix}. \quad (2.6)$$

The HTM in (2.6) can be expanded to obtain the recursive form of the kinematics as

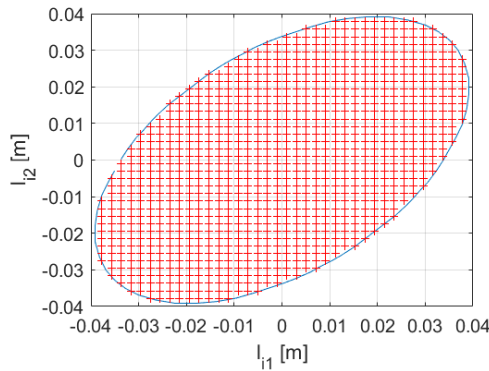


Figure 2.4: Actuator range of the reduced mapping.

$$\begin{aligned}\mathbf{R}^i &= \mathbf{R}^{i-1} \mathbf{R}_i, \\ \mathbf{p}^i &= \mathbf{p}^{i-1} + \mathbf{R}^{i-1} \mathbf{p}_i,\end{aligned}\tag{2.7}$$

where $\mathbf{R}^{i-1} : (\mathbf{q}^{i-1}) \mapsto \mathbb{SO}^3$ and $\mathbf{p}^{i-1} : (\mathbf{q}^{i-1}) \mapsto \mathbb{R}^3$ are the section tip rotation matrix and position vector of the preceding continuum section, respectively. Notice the absence of ξ_i , as $\xi_k = 1, \forall k < i$, by the definition of ξ_i (see [7]).

2.1.4 Dynamic Simulator

In addition to the kinematic model, this work uses a dynamic simulator to evaluate whether planned trajectories can be executed under more realistic system dynamics. We extend the continuum arm dynamic model developed in [8] to model the inextensible continuum manipulator dynamics. The model was implemented in the Matlab Simulink environment with a joint space PI controller for each DoF (proportional gain, $P = 7$ and integral gain, $I = 420$). Here, the dynamic model incorporates the PMA dynamics [10], where input pressure controls the length variation of PMAs. Thus, in the controller design, we employed joint value saturation limits ($0 - 6$ bars to imitate the 6-bar input pressure limitation), joint slew rate (from experimental data showing ± 1 bar/sec), and 25 ms delay to the sensor feedback loop. In this work, the dynamic model is used to assess the quality of trajectories generated by various path planning algorithms. This simulator tracks the arm under realistic dynamics. To judge the quality of a path, we expand or compress the execution time of the path until the dynamic simulator is able to track the arm with less than 0.01cm root mean squared error at the tip. If two paths are otherwise identical, a shorter time would indicate higher quality. The dynamic simulator requires that a time be associated with every transition between states.

2.2 Kinematics

Table 2.1 provides a summary of the notations discussed in this section.

2.2.1 $\mathcal{C}$ -space, $\mathcal{J}$ -space, and $\mathcal{W}$ -space

In robotics, a robot may be described in terms of a *state space*, which consists of all states that can be used to represent the system. A state is an n -dimensional description that fully specifies the robot's pose or shape, where n corresponds to the robot's degrees of freedom (DoF). A state is denoted by q . The configuration space ($\mathcal{C}$ -space), joint space ($\mathcal{J}$ -space), and workspace ($\mathcal{W}$ -space) are defined, and are denoted by the sets $\mathcal{C}$, $\mathcal{J}$, and $\mathcal{W}$, respectively. Throughout this work, the terms “ $\mathcal{C}$ -space” (respectively, “ $\mathcal{J}$ -space” and “ $\mathcal{W}$ -space”) are used when referring to these spaces conceptually in prose, and the symbols $\mathcal{C}$, $\mathcal{J}$, and $\mathcal{W}$ are used when referring to them mathematically as sets.

For the continuum arm considered here, two equivalent state representations are used. The arm is described by curve parameters in one representation, and by actuator-length values in the other. These correspond to representations in $\mathcal{C}$ -space and $\mathcal{J}$ -space, respectively, so that

$$q \in \mathcal{C} \quad \text{or} \quad q \in \mathcal{J}.$$

For the reduced model introduced in Section 2.1.1, two equivalent representations of the same state are provided by the configuration (curve parameter) and joint space (actuator length) descriptions. Thus, they are interchangeable, and q is used generically unless the chosen representation must be emphasized. When necessary, $q^{\mathcal{C}}$ is written for a state in the curve-parameter representation and $q^{\mathcal{J}}$ for the corresponding state in the joint space representation.

Symbol	Description
$\mathcal{C}$	Configuration space ($\mathcal{C}$ -space)
$\mathcal{J}$	Joint space ($\mathcal{J}$ -space)
$\mathcal{W}$	Workspace ($\mathcal{W}$ -space)
q	Robot state (configuration)
$q^{\mathcal{C}}$	State in the curve-parameter representation ($\mathcal{C}$ -space)
$q^{\mathcal{J}}$	State in the joint space representation ($\mathcal{J}$ -space)
p	Workspace point in $\mathcal{W}$
ϕ_i	Bending angle of section i
θ_i	Orientation of the bending plane for section i
l_{i1}, l_{i2}	Actuator length changes for section i
$\mathcal{C}_{\text{free}}$	Collision-free subset of $\mathcal{C}$
$\mathcal{C}_{\text{obs}}$	Collision subset of $\mathcal{C}$
q_1, q_2	States of the first and second arms in the dual-arm system
$q = [q_1^T \ q_2^T]^T$	State of the dual-arm system
L_{rod}	Fixed tip-to-tip distance between the two arms
$p_1(q), p_2(q)$	Tip positions of the first and second arms
$m(q)$	Midpoint of the tip-to-tip coupling; effective end-effector position

Table 2.1: Notation used in Section 2.2

Under these representations, a full state of an i -section continuum arm may be written in either of two equivalent forms. In the curve-parameter representation,

$$q^{\mathcal{C}} = [\phi_1, \theta_1, \phi_2, \theta_2, \dots, \phi_i, \theta_i]^T,$$

and in the joint space representation,

$$q^{\mathcal{J}} = [l_{11}, l_{12}, l_{21}, l_{22}, \dots, l_{i1}, l_{i2}]^T.$$

Thus, each section contributes two parameters in either representation. Accordingly, for an i -section arm under the reduced model, each representation is $2i$ -dimensional. Equivalently, letting $n = 2i$, each representation is n -dimensional. Although $\mathcal{C}$ denotes the curve-parameter representation, the term “configuration” is used generically to refer to a state of the robot, independent of its parameterization.

In the presence of obstacles, the collision-free portion of $\mathcal{C}$ is denoted by $\mathcal{C}_{\text{free}}$, while the set of configurations that result in collision is denoted by $\mathcal{C}_{\text{obs}}$. These subsets satisfy

$$\mathcal{C}_{\text{free}} \cap \mathcal{C}_{\text{obs}} = \emptyset \quad \text{and} \quad \mathcal{C}_{\text{free}} \cup \mathcal{C}_{\text{obs}} = \mathcal{C}.$$

In addition to the state space, a robot operates in a *workspace*, denoted by $\mathcal{W}$. The workspace describes the ambient physical space in which the robot exists. A point in $\mathcal{W}$ is typically given by an m -dimensional tuple corresponding to a physical position, and possibly an orientation, of some point of interest on the robot, such as the tip or a point along the arm.

For some robotic systems, such as certain mobile robots, the state space and $\mathcal{W}$ may coincide. For manipulators, however, the state space is often of higher dimensionality than $\mathcal{W}$, reflecting the fact that multiple internal states may realize the same task-space objective. This mismatch in dimensionality is referred to as *redundancy*. A robot is considered redundant when its state space has more dimensions than are strictly necessary to specify its task-space objective in the workspace. In such cases, multiple

states may correspond to the same workspace point. This is particularly important for continuum manipulators, where the arm shape may vary considerably even when the tip occupies the same location in $\mathcal{W}$.

2.2.2 Forward and Inverse Kinematics

Forward Kinematics refers to the process by which a state from the robot’s state space is mapped to a point in the $\mathcal{W}$ -space. In this work, a state is denoted by q , where $q \in \mathcal{C}$ or $q \in \mathcal{J}$, depending on the representation being used, and a workspace point is denoted by $p \in \mathcal{W}$. Thus, a state q is mapped to the corresponding physical point or points in the workspace. Depending on the application, these workspace quantities may describe the position and possibly orientation of the tip, or of other points along the robot’s structure.

Conversely, *Inverse Kinematics* (IK) involves the determination of a state q that corresponds to a desired workspace point $p \in \mathcal{W}$. In robotic systems where the dimensionality of the state space matches that of the relevant workspace description, a direct one-to-one mapping may exist. More commonly, however, the state space has higher dimensionality than the $\mathcal{W}$ -space, in which case the mapping is not unique. Robots with this property are termed *redundant*, meaning that multiple states may correspond to the same workspace point.

For redundant manipulators, IK problems are typically solved using numerical techniques such as iterative optimization. In continuum arms, this process can be particularly challenging, as convergence may occur to a local minimum and a viable state may fail to be identified. To help mitigate this issue, an “initial state” is often required by numerical IK methods, guiding the search toward an appropriate solution among the multiple feasible candidates.

2.2.3 Dual-Arm Model

We now extend the single-arm continuum model of Section 2.1 to a coupled dual-arm system. The model consists of two identical three-section continuum arms, each with 6 degrees of freedom, yielding 12 degrees of freedom in total.

Let $q = [q_1^T \ q_2^T]^T$ denote a state of the dual-arm system, where q_1 and q_2 are the states of the first and second arm, respectively. As in Section 2.2.1, each arm may be represented either in the curve-parameter representation or in the actuator-length representation. Accordingly, the dual-arm system admits both a $\mathcal{C}$ -space and a $\mathcal{J}$ -space description, obtained by stacking the corresponding 6-dimensional states of the two individual arms.

The two arms are mounted at fixed base locations separated along the x -axis. If the midpoint between the two bases is taken as the origin of the task-space coordinate system, then the bases are displaced by $\pm L_{\text{rod}}/2$ along the x -axis. In the implementation considered here, each continuum section has length $L_s = 0.15 \text{ m}$, the arm radius is $r = 0.0125 \text{ m}$, and the base separation is $L_{\text{rod}} = 0.15 \text{ m}$.

The tips of the two arms are constrained to remain a fixed distance L_{rod} apart. Thus, if $p_1(q)$ and $p_2(q)$ denote the tip positions of the first and second arm, respectively, then the corresponding fixed-length constraint is given by

$$\|p_1(q) - p_2(q)\| = L_{\text{rod}}.$$

For convenience, we refer to this fixed-length coupling in terms of a rod of length L_{rod} . This introduces a closed-chain coupling between the two manipulators, so that the feasibility of one arm is no longer independent of the other.

2.3. PATH PLANNING PRELIMINARIES

Rather than using the tip of a single arm as the task-space quantity of interest, we use the midpoint of this fixed-length coupling,

$$m(q) = \frac{p_1(q) + p_2(q)}{2},$$

as the effective end-effector position of the dual-arm system. In this way, a target point in $\mathcal{W}$ -space specifies the desired end-effector pose of the coupled system through the position of the midpoint, while still allowing the individual arm shapes to vary subject to the fixed-length constraint and collision constraints.

The forward kinematics of the dual-arm system are inherited directly from the single-arm model in Subsection 2.2.2. For each arm, the per-section homogeneous transformation matrices are composed as before, with the appropriate base offset applied prior to evaluating points along the arm. Thus, the dual-arm model preserves the kinematic structure of the single-arm prototype while introducing additional coupling, redundancy, and collision considerations that make inverse kinematics and path planning substantially more difficult.

Unless otherwise stated, a state of a single arm is considered valid/feasible only if it satisfies the bounds of the chosen representation, avoids self-collision, and avoids collision with any obstacles present in the environment. A dual-arm state is considered feasible only if, in addition, it respects the fixed-length constraint and avoids collision between the two arms.

2.3 Path Planning Preliminaries

2.3.1 Path Planning Problem

Given an initial state q_s and a goal state or goal region, the path planning problem is to determine a feasible path that moves the robot from the initial condition to the goal while satisfying the constraints of the system. In the present setting, planning may be carried out in either the $\mathcal{C}$ -space or the $\mathcal{J}$ -space, depending on the chosen representation. Thus, a state q is used generically unless the representation must be emphasized.

Let $\mathcal{Q}$ denote the state space used for planning, where $\mathcal{Q} = \mathcal{C}$ or $\mathcal{Q} = \mathcal{J}$. In a static environment, let $\mathcal{O} \subseteq \mathcal{W}$ denote the set of obstacles in the workspace, and let $\mathcal{B}(q) \subseteq \mathcal{W}$ denote the subset of the workspace occupied by the robot when it is in state q . A state $q \in \mathcal{Q}$ is said to be valid if it satisfies the bounds and constraints of the chosen representation and is collision-free. For a single arm, this means that q does not result in self-collision and that

$$\mathcal{B}(q) \cap \mathcal{O} = \emptyset.$$

For the dual-arm system, validity is defined similarly, with the additional requirements that the rigid rod constraint is satisfied and that the two arms do not collide with one another.

Accordingly, the path planning problem is to determine a path

$$\pi : [0, 1] \rightarrow \mathcal{Q}$$

such that $\pi(0) = q_s$, $\pi(1) \in \mathcal{Q}_g$ for some goal set $\mathcal{Q}_g \subseteq \mathcal{Q}$, and $\pi(s)$ remains valid for all $s \in [0, 1]$. In addition, the path must respect the motion constraints of the robot and the representation in which planning is performed.

For continuum manipulators, this problem is particularly challenging due to redundancy, nonlinearity, and the presence of obstacles. Multiple arm shapes may correspond to similar or identical task-space objectives, while differing significantly in feasibility, collision behavior, and execution quality. As a result, the objective of path planning is not merely to reach the goal, but to do so through a sequence of feasible states that is suitable for the arm's physical motion.

2.3.2 Dynamic Path Planning in Discrete Time

In the dynamic environments considered in later chapters, time is modeled discretely as

$$t \in \mathbb{N}.$$

At each tracing or execution step t , let $\mathcal{O}(t) \subseteq \mathcal{W}$ denote the true obstacle set in the workspace. Thus, $\mathcal{O}(t)$ represents the portion of the workspace occupied by obstacles at time t .

Using the same robot-occupancy notation as above, a state q is said to be valid at time t if

$$\mathcal{B}(q) \cap \mathcal{O}(t) = \emptyset.$$

The dynamic path planning problem is then to determine a time-indexed sequence of states

$$\{q(t)\}_{t=t_0}^{t_f},$$

such that each state $q(t)$ is valid at its corresponding time t , consecutive states satisfy the motion constraints of the robot, and the resulting sequence reaches a desired goal state or goal region.

2.4 Graph Theory

2.4.1 Graphs

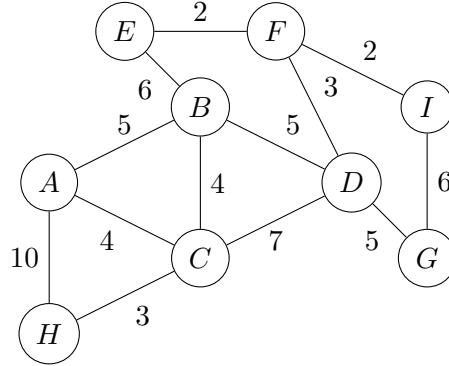


Figure 2.5: An example graph with labeled vertices and edges. The nodes are labeled with the letters A-I.

Graphs are mathematical structures used to represent relationships between data points. A graph G is defined by two finite sets: a nonempty set of *vertices* $V(G)$, and a set of *edges* $E(G)$, where each edge connects a pair of vertices known as its *endpoints*. Typically, a graph is denoted by $G = (V, E)$, by which its two fundamental components are captured.

A *directed graph* (or *digraph*) is similarly defined by a set of vertices and a set of directed edges. In this setting, each directed edge is represented as an ordered pair of vertices, indicating direction from one vertex to another. Thus, if (u, v) is a directed edge, it is understood to travel from vertex u to vertex v .

A graph is said to be *weighted* when each edge is assigned a real-valued quantity by a weight function. More precisely, a weighted graph, whether directed or undirected, is associated with a function w that maps each edge to its corresponding weight. These weights are often used to represent quantities such as distance, cost, or traversal time. For additional background on graph theory, see [11], [12].

2.4.2 Dijkstra’s Algorithm

Dijkstra’s algorithm [13] is a classic method for solving the single-source shortest path problem in a graph with non-negative weights. Exploration of the graph is performed starting from a source vertex until a destination or target vertex is reached, potentially resulting in all vertices being examined to construct a shortest path tree.

The process is initiated by setting a source vertex and, if desired, a target vertex. Initially, every vertex is assigned a “travel cost” of infinity, representing the minimum known cost to travel from the source to that vertex, except for the source vertex, which is assigned a cost of zero. The algorithm operates in a loop, beginning with a “current” vertex from which all adjacent, unvisited vertices are examined. These neighboring vertices are assigned a tentative travel cost, calculated by adding the travel cost of the current vertex to the cost of moving from the current vertex to the neighbor. If this tentative cost is smaller than the currently recorded cost, the neighbor’s value is updated accordingly. This update is referred to as a *relaxation* of the corresponding edge. After all neighbors have been evaluated, the current vertex is marked as “visited,” indicating that its shortest-path cost has been finalized. This process repeats until every reachable vertex has been visited, culminating in the shortest path from the source to the target being determined by tracing the sequence of predecessor edges.

As negative weights cannot be handled by Dijkstra’s algorithm, incorrect results may be produced when such edges are present. Moreover, it is designed to find the shortest paths from a single source to all other vertices in the graph, not just between two specific points.

2.4.3 A* Algorithm

A* [14] is a graph search algorithm for finding a least-cost path between a source vertex and a target vertex. A* has much in common with Dijkstra’s algorithm (Section 2.4.2), though a heuristic function is employed to direct exploration. In this way, the search is biased toward the goal, rather than relying on more uniform exploration.

As in Dijkstra’s algorithm, each vertex is associated with a cost-to-come value representing the current best known cost from the source to that vertex. In addition, each vertex is associated with a heuristic estimate of the remaining cost from that vertex to the target. These two values are combined into an evaluation function

$$f(v) = g(v) + h(v),$$

where $g(v)$ is the known cost from the source to vertex v , and $h(v)$ is the estimated cost from v to the goal. At each iteration, the vertex with the smallest f -value is selected as the current vertex, and its outgoing edges are relaxed in the same general manner as in Dijkstra’s algorithm.

A* is complete under standard assumptions on the search graph and heuristic. In practice, A* is often significantly faster than Dijkstra’s algorithm when a meaningful heuristic is available, since exploration is directed toward the target rather than treating all directions equally. However, a heuristic that is ill-suited to the graph structure can meaningfully reduce this advantage and degrade performance.

2.4.4 Bellman-Ford Algorithm

The Bellman-Ford algorithm [15], [16] is a well-known method for solving the single-source shortest path problem in a weighted directed graph (digraph). Unlike Dijkstra’s algorithm, which is typically faster but requires non-negative edge weights, the Bellman-Ford algorithm can accommodate negative-weight edges, thereby offering greater flexibility.

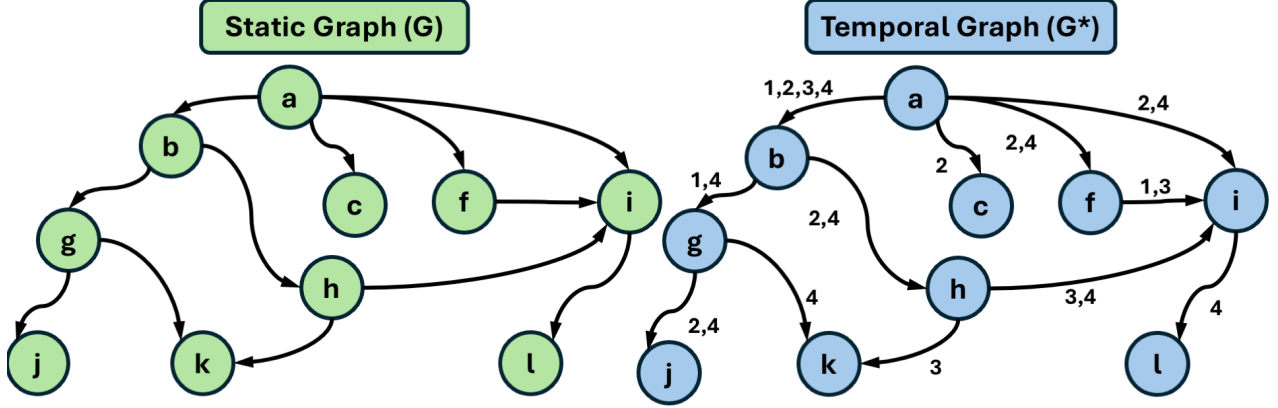


Figure 2.6: An illustration of a temporal graph with $\mu = 1$ and $d = 0$ and its corresponding static graph [17].

Starting similarly to Dijkstra’s algorithm (Subsection 2.4.2), a source vertex is selected and each vertex’s travel cost is set to infinity, except for the source vertex, which is assigned a cost of zero. The core of the algorithm is the repeated application of *relaxation*. In this procedure, for a directed edge (a, b) with weight w connecting vertices a and b , it is assessed whether the cost of reaching b can be reduced by passing through a . This is done by comparing whether

$$\text{cost}(b) > \text{cost}(a) + w.$$

If the condition is met, b ’s travel cost is updated to $\text{cost}(a) + w$, and a is set as the new predecessor of b . Since the maximum length of a shortest path in a graph without cycles is $|V| - 1$, where $|V|$ is the vertex count, the relaxation process is repeated for $|V| - 1$ iterations over all edges.

Finally, negative-weight cycles are checked for by performing one more relaxation iteration. If any cost decreases during this step, the presence of a negative-weight cycle reachable from the source is indicated, and thus no finite shortest path solution exists for affected vertices. If no changes occur, the shortest path is successfully output.

2.4.5 Temporal Graphs

In order to model an environment with temporal constraints, a *temporal graph* can be used; by this graph, it is possible to model which parts of the environment are accessible/available (e.g., devoid of obstacles) during certain times, which is crucial for path planning in such environments. As the name suggests, temporal graphs are graphs whose edges are open only during certain time windows and closed otherwise [17], [18]. In the representation used here, based on that of Wu et al. [17], a temporal graph is defined as a directed weighted multigraph $G^* = (V^*, E^*)$, where V^* represents the multiset of vertices and E^* represents the multiset of temporal edges. Each temporal edge $e \in E^*$ is defined as a tuple (u, v, w, λ) , where $u, v \in V^*$ are the start and end vertices of the edge, respectively, w is the time at which the edge becomes available (starting time), and λ is the time taken to traverse the edge (traversal time). The quantity $w + \lambda$ denotes the ending time of the edge. It is assumed that each temporal edge remains valid for the entirety of its traversal interval, i.e., over $[w, w + \lambda]$. In particular, if an edge were to become invalid at any point during this interval (e.g., due to obstacle interference), it would not be included in E^* .

For completeness, it is noted that in some settings (e.g., anticipatory planning), temporal edges may additionally be associated with a weight or cost. While this is not required for the base definition, such extensions can be incorporated without altering the structure of the temporal graph.

2.5. SAMPLING-BASED PATH PLANNING ALGORITHMS

Figure 2.6 shows an example of a temporal graph together with its corresponding static counterpart. In the figure, it is assumed that $\lambda = 1$ for all edges. The values shown on the temporal edges indicate the times at which those edges become available. It is noted that, for example, the path $A \rightarrow I \rightarrow L$ is valid in the static graph but not in the corresponding temporal graph. Instead, the temporal graph requires the path $A \rightarrow B \rightarrow H \rightarrow I \rightarrow L$.

A temporal path may be written as a sequence of temporal edges

$$\{(v_i, v_{i+1}, w_i, \lambda_i)\}_{i=0}^{L-1},$$

where each traversal occurs entirely within its valid time interval. A time-respecting temporal path must satisfy

$$w_{i+1} \geq w_i + \lambda_i, \quad i = 0, \dots, L-2. \quad (2.8)$$

This condition enforces that traversals occur sequentially and remain within their corresponding valid intervals $[w_i, w_i + \lambda_i]$. Since traversal times are positive, the starting times along a temporal path are strictly increasing.

In this context, the *edge stream* representation of a temporal graph is defined as an ordered sequence of its temporal edges, sorted by starting time. Edges with the same starting time are considered equivalent in the edge stream, and their relative ordering may be arbitrary. For example, if a temporal graph contains the edges

$$(v_1, v_2, 2, 5), (v_2, v_4, 4, 1), (v_3, v_2, 1, 1),$$

then the corresponding edge stream is

$$[(v_3, v_2, 1, 1), (v_1, v_2, 2, 5), (v_2, v_4, 4, 1)].$$

A temporal graph and its corresponding edge stream are illustrated in Figure 7.1.

2.5 Sampling-based Path Planning Algorithms

2.5.1 PRM

The Probabilistic Roadmap (PRM) [19] method is a sampling-based path planning algorithm in which a roadmap representation of the state space is constructed and then queried to find a valid path between a start and goal state. The planning process is separated by PRM into two distinct phases: a *construction phase* and a *query phase*.

In the construction phase, a set of states $q \in \mathcal{C}$ is randomly sampled from the state space. Each valid sampled state is treated as a vertex in the roadmap, and connections are attempted between nearby vertices. These connections are established using a local planner, by which it is determined whether a feasible, collision-free transition exists between two states. If such a transition exists, an edge is added between the corresponding vertices.

The notion of “nearness” is defined using a neighborhood structure, typically based on a distance metric in the state space. For each sampled vertex, candidate connections are evaluated within this neighborhood, and only those that satisfy the feasibility constraints are retained. As more samples are added, the roadmap becomes a more accurate approximation of the underlying state space connectivity, particularly in complicated regions.

Once the roadmap has been constructed, the query phase begins. The start state q_s and goal state q_g are inserted into the roadmap and connected to nearby vertices using the same local planning procedure. If both states lie within the same connected component of the graph, a path can be extracted using

2.5. SAMPLING-BASED PATH PLANNING ALGORITHMS

a graph search algorithm such as Dijkstra’s or A*. If no such connection exists, additional sampling may be required to improve the roadmap.

The key idea behind PRM is that reusable information about the structure of the state space is encoded by the roadmap. As a result, PRM is particularly well-suited for multi-query settings, where many planning problems are solved within the same environment.

2.5.2 RRT

The Rapidly-exploring Random Tree (RRT) [20] method is a path planning algorithm in which a path from an initial state to a target state is incrementally built by randomly sampling the state space. The algorithm operates by constructing a graph structure whose nodes correspond to states q . This structure is a tree ($\mathcal{T}$), which is initialized at a single root node corresponding to the initial state q_s and is expanded outward without creating cycles.

At each iteration, a state q_{rand} is sampled from the state space, defining a direction for growth. In some iterations, this sample is biased toward the goal state q_g , a strategy known as *goal biasing*. The nearest node in the tree to q_{rand} is then identified, and an attempt is made to extend the tree toward this sampled state.

This extension is performed using a *steering* procedure, by which a new state q' is generated along the path from the nearest node q toward q_{rand} , at a distance δ from q . If the resulting state and the transition from q to q' are valid, a new node corresponding to q' is added to the tree, and an edge is created between the two nodes. In this way, nodes are incrementally added to the tree, with each node representing a state in the state space.

The tree $\mathcal{T}$ continues to grow in this manner until a node is added that is sufficiently close to the target state and can be connected via a valid transition. Due to the cycle-free structure of the tree, once the goal is reached, a path can be obtained by retracing the parent-child relationships from the goal node back to the root.

A fixed number of iterations is designated by RRT for attempting node insertions. If the algorithm does not reach the target within these attempts, it is suggested that a path may not be achievable with the current number of samples. In such cases, the RRT can be executed again with a greater number of iterations, or it may be concluded that no feasible path exists under the current settings.

2.5.3 RRT*

RRT*[21] is a variant of RRT in which a cost measure is employed to indicate the quality of the tree $\mathcal{T}$ and to distinguish between a low-quality and high-quality path while the tree is *refined* during exploration. The *cost* between two nodes is defined as a metric specifying the cost of transitioning between the states corresponding to these two nodes. The cost between two nodes $n, m \in \mathcal{T}$ is denoted $\zeta(n, m)$. For a node n in $\mathcal{T}$, let $P = (n_s = n_0, n_1, \dots, n_k = n)$ be the unique path in $\mathcal{T}$ from the starting node n_s to n . The *path cost* of n is defined as

$$v(n) = \sum_{i=0}^{k-1} \zeta(n_i, n_{i+1}). \quad (2.9)$$

To refine the tree, upon insertion of a new node n' , it is checked whether n' “opens up” shorter paths to some existing nodes in $\mathcal{T}$. This process of examining the vicinity or *neighborhood* of a newly-inserted node and changing edges in the tree is referred to as *rewiring*. The neighborhood is defined by a distance $\alpha \geq 0$. More formally, when a node n' is inserted in $\mathcal{T}$, if some node m within distance α of

2.5. SAMPLING-BASED PATH PLANNING ALGORITHMS

n' satisfies $\zeta(n', m) + v(n') < v(m)$, then a shorter path exists to m (via n'). In this case, the parent of m in $\mathcal{T}$ is replaced with n' , and the cost of m is updated to reflect this change.

2.5.4 Time as a Smoothness Metric

In the kinematic sense, smooth paths are generally characterized by gradual and small changes over time in the corresponding arm states. While smoothness may be assessed in a number of ways, here we focus on smoothness as it pertains to the arm's joint space. Usually, the amount of change between consecutive joint space states is inversely proportional to the smoothness of the corresponding path. Paths with smaller changes between consecutive joint space states are more easily and quickly tracked with feedback controllers without causing oscillation issues during tracking. Thus, by maximizing smoothness in the trajectory, the speed of the continuum arm can be increased without causing oscillations during controller tracking. The $\mathcal{J}$ -space velocity of each actuator is incorporated in path planning to compute a smooth trajectory and to accurately relate the generated paths with the actuated movement of the continuum arm.

To that end, first, the notion of a velocity threshold, V_{th} , is defined as the maximum PMA velocity. A change limit, V_δ , is also defined below, which determines the maximum allowable change in velocity between two consecutive configurations on the path. In the path generation, two consecutive configurations q and q^* and their corresponding length changes $l_{i,j}$ and $l_{i,j}^*$ are considered, where $i = 1, 2, 3$ denotes the arm's section and $j = 1, 2$ denotes the actuator within a section.

For section $i \in \{1, 2, 3\}$ and actuator $j \in \{1, 2\}$, the shortest movement time, $t_{i,j}$, between $l_{i,j}$ and $l_{i,j}^*$, is calculated as

$$t_{i,j} = \frac{|l_{i,j}^* - l_{i,j}|}{V_{th}}. \quad (2.10)$$

Since no actuator can actuate in time less than $t_{i,j}$, the minimum time to actuate between q and q^* , $t_{shortest}$, is calculated as

$$t_{shortest} = \max\{t_{i,j}\}, \quad i = 1, 2, 3, \quad j = 1, 2. \quad (2.11)$$

After computing $t_{shortest}$, actuation of the robot from q to q^* is performed in this shortest possible time.

Note that there are situations where (2.10) does not hold. For example, if the velocity of an actuator were to change from negative to positive, it is possible that the magnitude of the change could be quite large despite being within the velocity threshold. To mitigate this, the use of a maximum change V_δ is proposed such that $V_\delta - V_{th} > 0$. If a sign change occurs, the new maximum actuator velocity becomes $V_\delta - V_{th}$ to ensure that this large change occurs slowly. Note that this velocity threshold is used for normalizing the trajectory with a common actuator velocity. Once a time trajectory is generated, the duration of this trajectory is modified until a smooth trajectory is obtained under desired constraints during empirical tests.

Chapter 3

Literature Review

This literature review offers an examination of continuum arms and explores common planning techniques and their application to such devices. Continuum arms represent a unique class of robotic manipulators, characterized by their compliant construction and novel bending mechanisms, which enable movement and actuation. These robots are particularly valuable in fields requiring high degrees of dexterity, such as advanced manufacturing, healthcare, and human-robot collaboration.

In this literature review, we also examine the existing research on path planning for continuum-arm devices. These manipulators present a unique set of challenges, including their intricate kinematics and often redundant design. Despite these challenges, there are several existing approaches that have been demonstrated to generate quality trajectories for the continuum devices in question.

While the literature addressing path planning for continuum arms is relatively sparse, there are many planners whose application is related to these devices and highly relevant to their discussion. Notably, rapidly-exploring random trees (RRT) and probabilistic roadmaps (PRM), two popular sampling-based planners, show significant potential for continuum arms.

As these continuum devices are targeted for demanding applications, effective path planning approaches for such robots remain an important problem. This section also includes a brief overview of planners developed specifically for these devices, highlighting the importance of this research in enabling the use of continuum arms in demanding application domains.

3.1 Continuum Robotics

Conventional robotics are often defined by their stiff components and, in the case of manipulations, rigid-link arm design. These devices create motion at their joints, often with revolute actuators. In many domains, rigid-link robots are paired with end-effectors that enable these devices to perform a number of complex manipulation tasks, as is seen in industrial manufacturing applications.

Continuum manipulators, in contrast, are defined by their continuous bending, frequently due to elastic deformation. Such devices perform movement primarily by modifying the shape of their curve, in addition to other motions like extension, contraction, and twisting. Such continuum devices are said to have an infinite number of degrees of freedom (DoF), as their signature curve resembles a rigid-link manipulator with infinite joints and infinitely-small links [22], [23], [24], [25], [26]. This property is said to make a continuum manipulator *hyper-redundant*. The unique design of these robots allows such devices to assume a vast array of novel spatial-shapes, creating potential advantages in areas like obstacle avoidance. Additionally, such devices are suited to perform “whole-body manipulation”, wherein an arm can perform a manipulation task without the need for a specialized end-effector[27].

3.1. CONTINUUM ROBOTICS

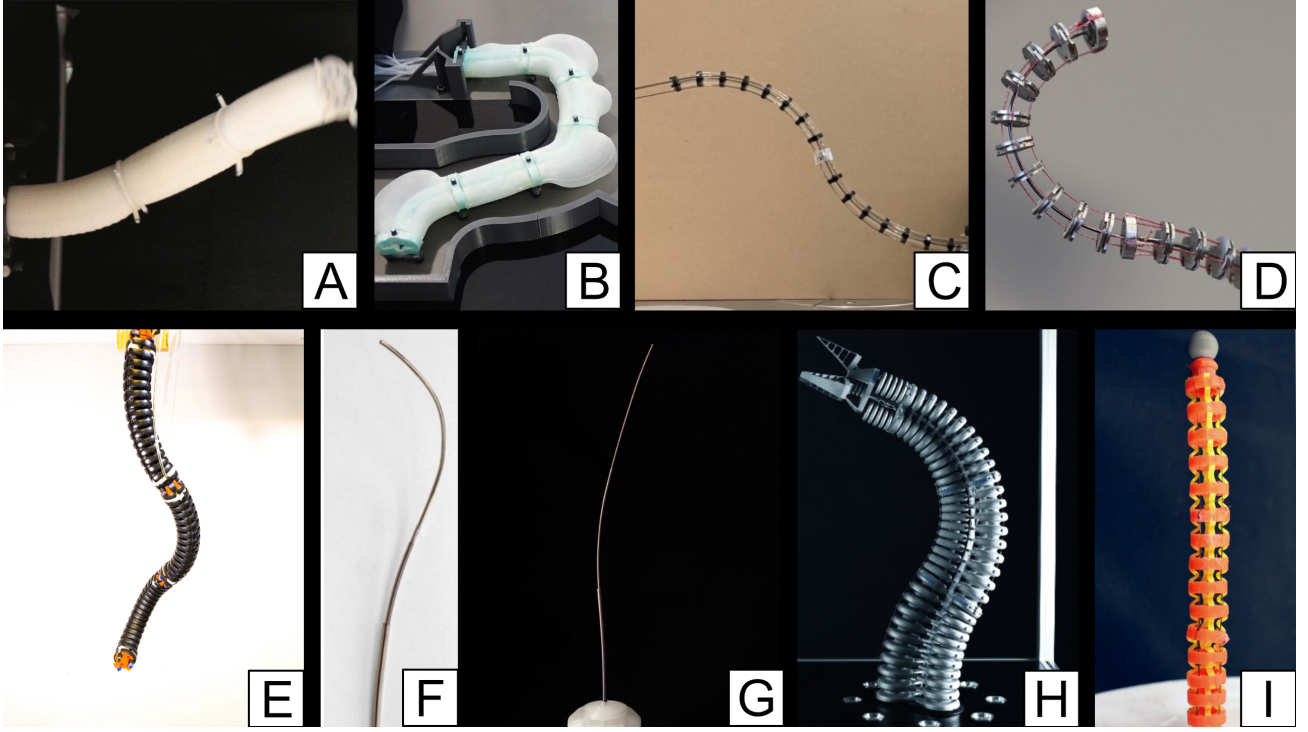


Figure 3.1: A collection of continuum arms is shown with a label at the bottom right of each.

(A) A multisection continuum arm constructed of soft pneumatic muscles[32]. (B) A soft planar continuum arm constructed of pneumatic actuators[33]. (C/D) A tendon-driven continuum arm with a novel magnetic locking scheme shown in [34] and [35], respectively. (E) A multisection hybrid continuum arm made with pneumatic muscles and a rigid inextensible backbone (from the authors of [1]). (F/G) A concentric tube robot with 3 concentric tubes shown in [36] and [37], respectively. (H) A continuum arm using a spine with expandable air chambers [38]. (I) A tendon-driven continuum arm made from inexpensive parts [39].

The kinematics of such devices are often more complicated than rigid-link robotics, which can be problematic in applications of inverse kinematics and path planning.

3.1.1 Materials

Though the design and construction of continuum manipulators is outside of the scope of this survey, these devices are discussed at a high-level for the purposes of completeness and to elucidate their complexity when applied to path planning. Continuum manipulators can be created through a wide variety of models. Broadly speaking, continuum devices can be constructed from materials ranging from rigid to soft, with trade-offs between both construction types. Robots constructed with soft materials are capable of assuming a wider array of shapes and can be useful in situations where dexterity is important, though their payload capacity is limited. Robots constructed with rigid components, on the other hand, are capable of handling larger payloads, but at the cost of flexibility and human-friendliness. Many continuum robots are constructed around a backbone, which provides structure around which bending can occur. Hybrid continuum arms utilize both rigid and soft materials (often with a backbone) to get the best features from both materials. Furthermore, continuum manipulators are often bio-inspired, meaning that their construction resembles physical appendages found on creatures like elephants, monkeys, octopuses, etc. Some examples of such bio-inspired devices are shown in [23], [28], [29], [30], [31].

3.1.2 Actuation

The discussion of actuation methods of a continuum arm robot is outside of the scope of this dissertation. However, a few specific models are highlighted in the following paragraph. Many actuation methods involve using a material or device that is capable of deforming into a curve, and then arranging them selectively to create a manipulation-capable motion. One of the most common methods is a tendon-driven construction, where a number of cables are run parallel to the length of the arm. In such devices, an individual cable's length can be increased or decreased to create a bend in the arm. These cables can be combined with a number of other devices, such as a backbone, locks, magnetic rods, etc to influence characteristics like the degree of bending or the strength of the arm. Examples of tendon/cable driven continuum arms are included in [31], [34], [35], [40]. Similarly, pneumatic, shape-memory, and hydraulic devices can also be used to create movement. For such robots, a deformable material is arranged in bundles or around a backbone in a way that bending/straightening individual components forces a curve in the arm. Examples of these continuum arms are included in [41], [42], [43]. For all of the aforementioned actuators, several shorter continuum arms can be joined serially to create a multisection continuum arm. Each section of this arm can be controlled independently, allowing for more shapes, at the cost of design and modeling complexity. Multisection continuum arms are demonstrated in [7], [44], [45]. Lastly, several precurved tubes can be nested inside of each other to create a concentric-tube robot. Such devices are actuated by translating and rotating individual tubes in the robot to modify the shape of the robot. We encourage the reader to review [22], [24], [46], [47] for more information on the design, modeling, and control of continuum manipulators. See Figure 3.1 for images of several different continuum manipulator models.

3.1.3 Applications

Continuum robotics is an active area of research, and a number of continuum devices are introduced each year for new applications. A few of these promising applications are detailed here. These devices excel in domains where they are able to utilize their unique shape and flexibility to avoid obstacles and avoid harming humans. For example, concentric tube continuum arms have shown promise in surgical applications, as was demonstrated in [48], [49]. Additionally, the models developed in [50], [51] were designed for minimally-invasive inspection tasks for outer-space applications. Furthermore, hybrid continuum models, such as those discussed in [1], [7], [52], have a mix of payload and safety that make them ideal to serve in human-robot collaborative domains as co-robots [53]. Examples of these domains include human mobility aid, complex manufacturing, and engineering. Complex engineering applications have also been discussed for robots in difficult environments, such as underwater work.

3.1.4 Challenges

Though continuum manipulators have many qualities that make their use in real-world applications desirable, they also present several challenges for path planning. While this is outside the scope of this dissertation, the inverse kinematics problem for continuum manipulators is nontrivial and remains an active area of research. Since many continuum manipulators are redundant, IK can be computationally costly and may fail more often than in conventional robotics. Singularities and local minima are challenges for the planning and IK of many robotic models, but they are especially pronounced for continuum manipulators due to their atypical $\mathcal{C}$ -space.

In addition, the $\mathcal{C}$ -space of such manipulators is often highly complex, so that linear trajectories in $\mathcal{C}$ -space may correspond to undesirable motion in $\mathcal{W}$ -space. This means that planners often must construct more sophisticated paths in order to reduce auxiliary movement. A further challenge is that

the practical difficulty of path planning for continuum manipulators depends heavily on the model and representation used for the robot. In many cases, reduced-order geometric models are preferred during planning because they make configuration generation, interpolation, and collision checking more tractable. More detailed mechanics-based models may better capture the behavior of the robot, but they also increase the computational burden of planning. As a result, the performance of a planner for continuum arms is often tied not only to the planner itself, but also to how the arm is represented during planning.

3.2 Seminal Sampling-Based Planners

Sampling-based planners are a category of path planners that, broadly speaking, rely on random sampling of a robotic space. Their strategy generally revolves around randomly selecting points in the $\mathcal{C}$ -space (though sometimes in the $\mathcal{W}$ -space), checking their validity, and then using these random samples to construct a corresponding topological graph. By following a path contained in this graph, a sequence of configurations constituting a robotic path plan can be outputted and used to move the robot.

Sampling-based path planners were a very hot area of research in the 1980s and 1990s where a vast number of new random planning schemes were being devised. One of the earliest contributions in this field was the Randomized Path Planner (RPP) [54], which combined artificial potential fields (discussed in Section 3.6) with random motion to overcome issues of minima. There were other randomized planners at this time as well, notably [55]. Concurrently, there were planners shown in [56], [57] that relied on grid-search to solve the path planning problem. There were also other planners, shown in [58], [59] that demonstrated early examples of sampling-based planning. The first seminal work, probabilistic roadmaps, was published in [19]. There was an explosion of investigation into PRM before another seminal planner, rapidly-exploring random trees, was published in [20], [60]. These two planners constitute the vast majority of sampling-based path planners and were specifically designed to handle robots with a high-dimensional $\mathcal{C}$ -space.

3.2.1 Problem Description

Sampling-based planners generally involve creating a graph $G = (V, E)$ that is embedded in the $\mathcal{C}$ -space. Let $V = \{c \in \mathcal{C}_f \mid c \text{ is a randomly sampled configuration}\}$ be the set of configurations and let $E = \{(a, b) \mid a, b \in \mathcal{C}_f \text{ and have a valid, collision-free transition}\}$ be the set of edges. To find a path, we add initial and goal configurations c_s, c_g to G and search for a path between them.

3.2.2 PRM

Probabilistic Roadmaps (PRM) [19] is a sampling-based planning method that relies on a randomized roadmap (a network of interconnected points). This approach relies on two phases: A learning phase and a query phase.

During the learning phase, a roadmap is constructed. First, random points are sampled in the $\mathcal{C}$ -space with uniform probability. When a point is sampled, the planner adds a corresponding node and then tries to create edges to nearby nodes if they are contained within the node’s *neighborhood* and there exists a collision-free transition between them. A node’s neighborhood is a ball centered around the node with fixed radius that contains all nodes that are “sufficiently close” for an edge to be formed. A *local planner* is used to verify that such a transition exists and, assuming the transition is valid, the edge is created. Typically, the output from the local planner is not saved and is reproduced

3.2. SEMINAL SAMPLING-BASED PLANNERS

when outputting a path. Once the roadmap is generated, it can be *enhanced* through a number of strategies. Though the $\mathcal{C}$ -space is sampled randomly, more points may be needed in difficult areas, such as areas near or between obstacles, to ensure that a path exists. During the enhancement phase, nodes are added selectively in such difficult areas so that they are sufficiently sampled and contain enough nodes to be included in the roadmap. In the original publication, random walks starting from nodes near obstacles are used.

During the query phase, the planner attempts to insert into the roadmap nodes corresponding to the initial configuration c_s and the goal configuration c_g . These nodes are inserted using the aforementioned neighborhood and local planner strategy. Once these nodes are added, the planner checks to see whether they are in the same connected component. If they are, then a path between the nodes exists and can be generated and outputted using a shortest-path algorithm [13] and the local planner (or by simply following the child-parent structure if no cycles exist in the roadmap [19]). However, it may be that one of these initial or goal nodes cannot be inserted, or that they are not contained within the same connected component. In this case, either there does not exist a path between these nodes, or the learning phase did not sufficiently sample the $\mathcal{C}$ -space. In either case, the planner can return to the learning phase or report that no path was found and terminate.

3.2.3 RRT

Rapidly-exploring random trees (RRT) [20] is another sampling-based planning method that iteratively constructs a path outward from the initial node towards a target node. This approach explores the $\mathcal{C}$ -space through random sampling to grow a tree of nodes. A tree is a graph structure that has one root node and grows outwards without cycles. The growth of the tree begins with a starting node corresponding to the c_s . In each iteration, a point is chosen as the growth direction. In some iterations, this direction point is selected as a random point in the $\mathcal{C}$ -space, but in other iterations, the tree growth can be biased by growing in the direction of the target point. Once a direction point is selected, a new node is then attempted to be inserted into the tree using *steering*. This new node is steered by selecting it to be along the straight-line between the nearest node and the direction point at some small fixed distance. The algorithm verifies that the new node (and corresponding configuration) are valid and that the transition between the new node and nearest node is collision free and valid. The tree continues to grow until a new node is inserted that is sufficiently close to the target node and for which there is a valid transition. Since there are no cycles in this tree structure, once the target node is inserted, the chain of parent-child edges can be followed and recorded until the initial node is reached. Using this chain, a path is generated and outputted. In RRT, the number of iterations where a node insertion is attempted is a fixed number. If the RRT algorithm performs the maximum number of iterations without reaching the target, then a path is said not to exist for the number of samples. As with PRM, the RRT algorithm can be rerun with a larger number of iterations, or it can be reported that a path does not exist.

Completeness in path planning is the property that if a path exists, it is discovered in finite time and outputted [61]. Both RRT and PRM are known to be probabilistically complete [21], meaning that the probability that a path is found, if one exists, approaches 1 as the number of iterations or insertions reaches infinity. This is a powerful property, meaning that the existence or non-existence of a path is closely related to how long is spent looking for one. There exist a number of complete planners [62], but such planners often require far more computation time than RRT or PRM.

PRM and RRT differ in another key way: queries. Since PRM constructs a roadmap that samples the entire $\mathcal{C}$ -space, it is possible that the roadmap could be stored or reused for path planning queries beyond the original one. Furthermore, the roadmap could be modified slightly if the environment has

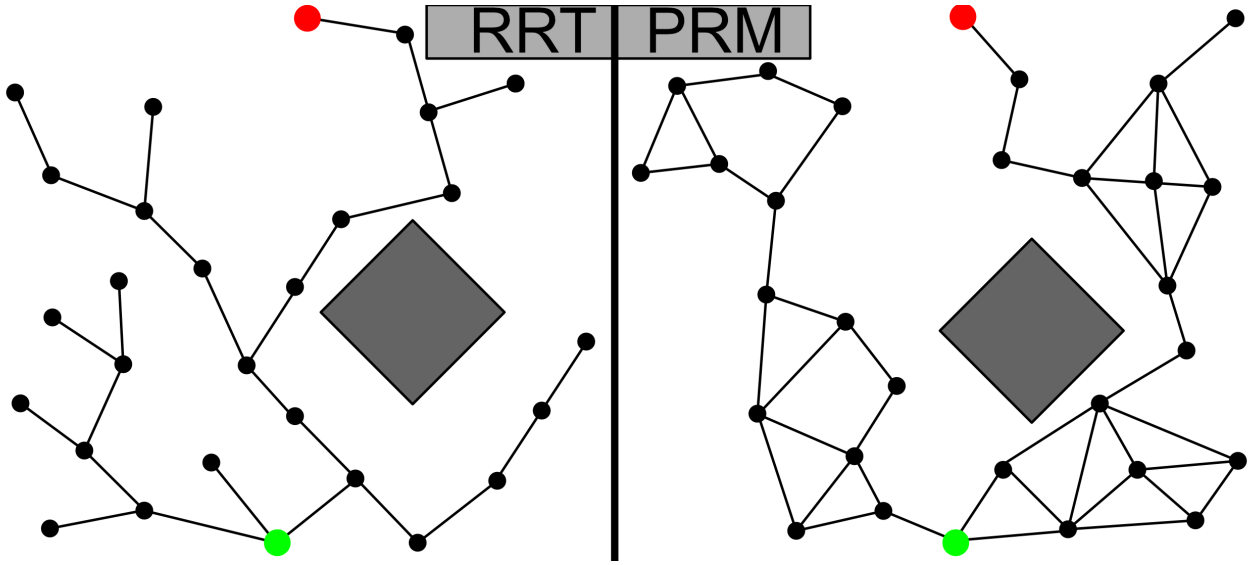


Figure 3.2: A comparison between a rapidly-exploring random tree (Left) and a probabilistic roadmap (Right). The starting configuration is shown in green and the target configuration is shown in red. A single obstacle is shown as a grey diamond.

changed due to the introduction, removal, or movement of obstacles. To achieve this, one would simply take the existing roadmap, perform any additional learning or enhancement, and then try to attach the new start and goal nodes. This is called a *multi-query planner*. RRT, on the other hand, does not share this utility. Since the tree is grown from an initial root node, if a new planning query is requested, there is very little likelihood that the existing tree could provide any use to solve this query. For that reason, a new tree must be grown for each query. Therefore, RRT is called a *single-query planner*. Though this may seem like a key advantage of PRM, RRT trees are far faster to calculate than roadmaps, as only the area relevant to an individual query are explored during planning. These differences are often contemplated when selecting one or the other to use for a specific planning domain. An image comparing a roadmap to a tree is shown in Figure 3.2.

3.2.4 Asymptotic Optimality

A more recent contribution to sampling-based path planning is the introduction of *asymptotically optimal* versions of RRT and PRM. The optimality of a path relates to the *cost* of a path generated by a planner. Cost refers to a measure used to evaluate the quality or efficiency of a path generated by an algorithm. This cost could be based on various criteria, such as the length of the path, the amount of energy required, time taken, etc. Typically, costs are assigned to edges in the graph G , creating a weighted graph and qualifying the cost of transitions between nodes. In path planners, asymptotic optimality is the property that a path generated by a planner converges towards the optimal path as the number of insertions/iterations reaches infinity. Asymptotically optimal planners are not just able to find a feasible path from the start to the goal, but to improve the quality of this path over time, approaching the best possible solution as the number of iterations increases. The concept of cost is central to this optimization process, as the algorithm continuously seeks to reduce the cost, thereby yielding increasingly efficient or shorter paths within the configuration space. The seminal work on asymptotically optimal planners was published in [21] and establishes the creation of two new planners: RRT* and PRM*.

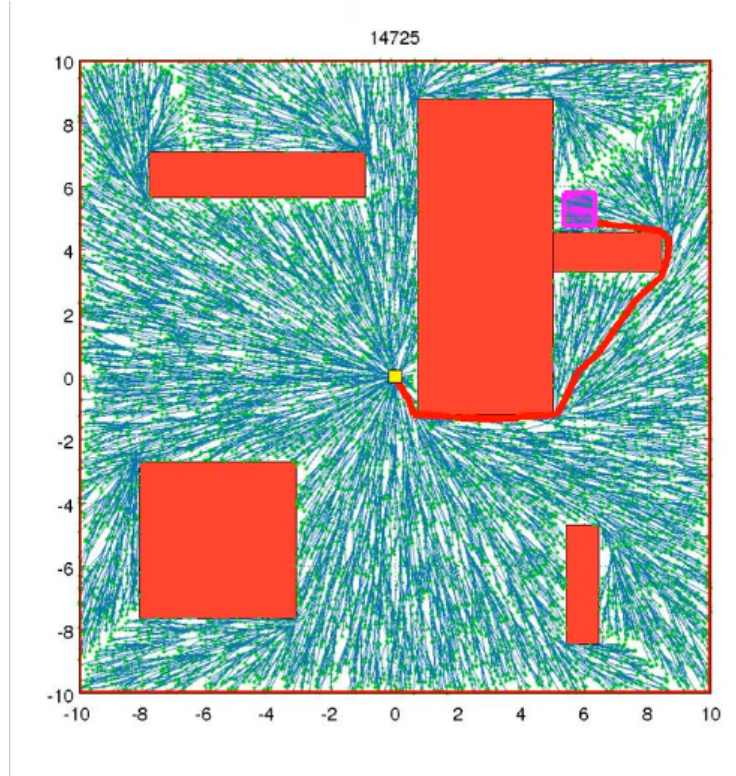


Figure 3.3: A tree generated by the RRT* algorithm. This image is taken from multimedia materials associated with [21].

Both approaches are modified so that the graph G is weighted, and that edges between nodes correspond to some “cost” of transition. PRM* achieves optimality by reducing the size of the roadmap’s neighborhood as the number of nodes is inserted. This way, new nodes are connected to a decaying radius of nodes. The authors in [21] provide a proof that this small change achieves asymptotic optimality. On the other hand, RRT* achieves asymptotic optimality through the use of a new step: rewiring. Rewiring is the process of changing the parent node of a node in a tree such that the change of parent reduces the overall path cost from the root to the node in question. During tree growth, whenever a node is inserted, the total cost from the root node to the current node is calculated. Next, this node checks all of its neighbors to see if they should be rewired. If the total path cost of the current node plus the cost of transitioning to the neighbor is lower than the neighbor’s current path cost, then the current node becomes the new parent of the neighbor. Again, the authors in [21] provide a proof that shows that RRT* is asymptotically optimal. RRT* is one of the most important recent contributions to this field, and has seen a number of variants introduced each year. See Figure 3.3 for an example of an RRT* tree.

3.2.5 Seminal Variants

PRM and RRT are quite simple algorithms with minimal implementation details. After these planners were published, a number of authors followed up with variants that changed or improved some features of these algorithms. RRT*, for example, is considered a variant of RRT and has proven to be just as popular as the original. We detail here some of the most important RRT and PRM variants before discussing further variants in later sections. See Section 3.3 and Section 3.4 for more variants and those specific to continuum arms.

3.3. RRT VARIANTS

Table 3.1: Seminal Sampling-Based Path Planners

Planner	Probab. Complete	Asympt. Optimal	Graph Structure	Space Sampled	Query Type
RRT[20], [60]	Yes	No	Tree	$\mathcal{C}$ -space	Single-query
PRM[19]	Yes	No	Roadmap	$\mathcal{C}$ -space	Multi-query Single-query and Multi-Query
Lazy-PRM[63]	Yes	No	Lazy Roadmap	$\mathcal{C}$ -space	Single-query
RRT-Connect[64]	Yes	No	Multi-tree	$\mathcal{C}$ -space	Single-query
RRT*[21]	Yes	Yes	Tree	$\mathcal{C}$ -space	Single-query
PRM*[21]	Yes	Yes	Roadmap	$\mathcal{C}$ -space	Multi-query
RRG[65]	Yes	Yes	Connected Roadmap	$\mathcal{C}$ -space	Single-query Single-query and Multi-Query
RRF[66]	Yes	No	Forest	$\mathcal{C}$ -space	Single-query and Multi-Query
Lazy PRM*[21], [67], [68]	Yes	Yes	Lazy Roadmap	$\mathcal{C}$ -space	Single-query
Lazy RRG[67]	Yes	Yes	Connected Roadmap	$\mathcal{C}$ -space	Single-query
RRT#[69]	Yes	Yes	Tree	$\mathcal{C}$ -space	Single-query
RRTX[70]	Yes	Yes	Tree	$\mathcal{C}$ -space	Single-query

One of the first and most important variants to arise was Lazy-PRM. This variant of PRM chooses not to evaluate the edges between nodes during the learning phase, and only checks during the query phase. This means that during the query phase, a number of edges and nodes may be invalidated, necessitating a return to the learning phase. As such, this variant is better suited for single-queries, but can be applied to multi-queries if the roadmap is saved. A variant of LazyPRM called LazyPRM* was introduced in [67]. This algorithm shares the lazy growth scheme of Lazy-PRM, but introduces the star method for neighborhood size.

RRT-Connect [64] is a very notable variant that introduces the use of multiple RRT trees during growth. In this algorithm, two trees are grown: one originating from the target node, and one originating from the initial node. Tree growth alternates between each of the trees, until a node from one tree is connected to a node from another. In addition to random sampling, new nodes in one tree are steered or biased towards the nearest node in the other tree. The authors in [65] proposed an algorithm similar to RRT*, but that does not enforce the tree property. This algorithm was also shown to be asymptotically optimal. Lastly, the authors in [66] demonstrated that they could use multiple invocations of RRT to create a collection of trees also called a *forest*. Once this forest is generated it is updated and reused like a roadmap. This approach was also applied to dynamic planning.

RRTX and RRT# are two state-of-the-art sampling planners that have been developed very recently. Though the former was designed for dynamic planning, they have both been shown to be very efficient for static planning. These planners rely on a relaxation scheme to improve path quality. These are discussed in subsequent sections, but are mentioned here due to their importance.

3.3 RRT Variants

A number of variants have been proposed for RRT over the years. Given its simplicity to implement and excellent efficiency, it is a very popular solution to the path planning problem. These planners have been applied to a vast array of domains and robotic models, so there is a vast array of planner

variants and techniques applied to RRT for specific models. However, this planner is also very popular for mobile robots, so there are portions of this work are not applicable to continuum manipulators. There does not exist a very large body of work surrounding continuum arm applications of RRT. However, given that RRT is designed for high-dimensional search, there are a number of planners that are relevant to continuum manipulators despite not being directly applied to them. In this section, we provide a survey of those recent works whose purpose is related or directly applied to continuum manipulators.

3.3.1 Redundant Manipulators

One of the easiest ways to handle redundant manipulators is to try to reduce the DoF of the spaces used for planning. In [71], the authors introduce $\mathcal{W}$ -space path planning using Jacobian-based IK methods and a $\mathcal{W}$ -space Voronoi biasing to reduce the complexity of the functional search space. Bidirectional RRT* methods [72], [73] have also shown themselves to be useful in the redundant planning space, as their simultaneous planning allows for more rapid exploration of complicated spaces. One of the most important and recent contributions to this field is RRTSharp [69], a static planner that uses relaxation-style methods. RRTSharp iteratively refines the path towards the best possible solution, making it ideal where path quality is paramount. This method was also shown to be very performant, only recently being overshadowed by [70] (discussed in Section 3.5). The authors in [74] demonstrated an RRT implementation for redundant manipulators that was able to use a follow-the-leader style motion with additional constraints on joint angle to create smooth movement in the $\mathcal{W}$ -space.

3.3.2 Continuum Manipulators

One of the most recent works applying RRT* to continuum robots was presented in [1] which, similar to $\mathcal{W}$ -space planning [71], uses a modified Jacobian-based method paired with a modified RRT* to reduce the complexity of the search space. Subsequently, the authors in [75] augment a RRT* planner with a cross-entropy estimation method. In this method, they are able to sample paths in areas that are estimated to contain high-quality paths. The authors of [76] embed safety constraints into a RRT* planner such that it produces smoother trajectories. This approach was also demonstrated for dynamic planning.

3.4 PRM Variants

Similar to RRT, numerous variants of the Probabilistic Roadmap Method (PRM) have been developed over the years, reflecting its popularity and versatility in addressing path planning challenges. PRM’s straightforward implementation and effectiveness make it a widely adopted approach across diverse domains and robotic architectures. It is particularly prevalent in applications involving mobile robots, leading to a rich array of specialized PRM techniques and variants tailored to such models. However, the direct application of PRM to continuum manipulators is less explored, despite its potential suitability for high-dimensional search problems characteristic of continuum robotics. This gap suggests an opportunity for adapting and extending PRM methodologies to continuum manipulators.

3.4.1 Continuum Manipulators

A redundant PRM is presented in [78] that uses a scheme called a virtual force field that enables the exploration of narrow passages. The authors in [77] presented an interesting PRM-based planning

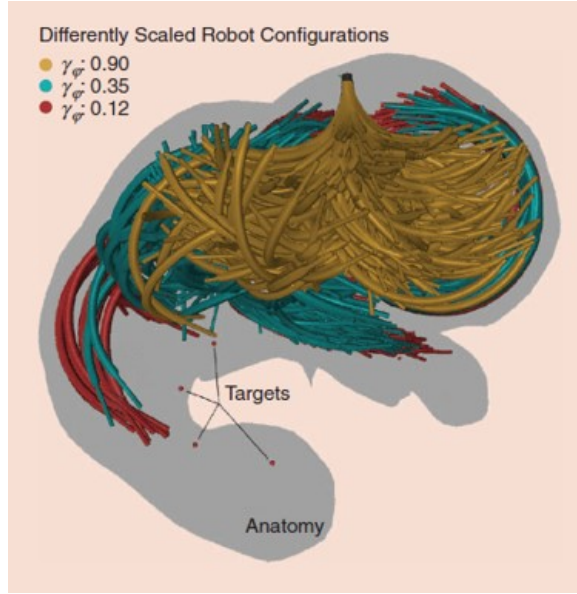


Figure 3.4: An image of all of the feasible configurations generated during the construction of a PRM for concentric tube robots. These configurations are generated rapidly using a parallel IK system and display configurations that are not colliding with the exterior wall. This image is taken from [77]

scheme. Their surgical application required that obstacles be represented as a boundary wall, and the motion generated by the PRM was subject to a number of constraints to ensure the safety of the patient. Furthermore, their real-time application required that the PRM be generated and reformed very rapidly to account for changes in the body. They also needed the concentric tube to allow for human-intervention, wherein a human could modify the plan by manipulating a controller. They utilized a highly-parallelized IK scheme, wherein new collision-free configurations are generated in real time to facilitate rapid reconstruction of the roadmap, and IK calls for movement outside of the roadmap.

3.5 Dynamic Planning

Dynamic path planning extends the foundational concepts of traditional path planning to environments where obstacles and constraints are not static but change over time. In dynamic path planning, the robot not only has to find a sequence of states leading from an initial position to a goal position but must also adapt to the evolving nature of the environment. This adaptation requires the planner to continuously or periodically reassess its path, accommodating moving obstacles, or altering environmental conditions. Unlike static environments where a once computed path remains valid, dynamic environments pose unique challenges that demand real-time or near real-time responsiveness and robustness in path planning. The dynamic aspect introduces complexities such as predicting the movement of obstacles and planning under uncertainty, significantly increasing the computational demands of the problem. This problem variant mirrors real-world scenarios more closely, where environments are rarely static and unforeseen changes are common. Sampling-based path planning has been a common solution to this problem, updating the existing trees or roadmaps as the environment changes. Though dynamic planning is a complex topic on its own, a small collection of sampling-based planners for dynamic environments are mentioned here.

3.5.1 Reactive Dynamic Planners

Reactive dynamic path planners address a challenging version of the dynamic path planning problem, focusing on unpredictable environments. In reactive planning, algorithms are assumed to have no prior knowledge about the movement of obstacles and instead respond in real time to the changing environment. These planners prioritize quick decision-making and flexibility, since environments may change rapidly and unexpectedly. This approach is crucial in scenarios where precomputed paths are quickly invalidated by environmental changes, making such planners useful in applications like human-robot collaboration or surgery, where reactive avoidance is important.

Some reactive planners, such as [79], [80] generate initial plans before updating them anytime a collision is near. The approach in [79] utilizes a way-point scheme wherein the planner attempts to reach recently disconnected configurations to try to re-enter “safe” areas. The approach in [80], on the other hand, chooses to invalidate dead branches of the tree and regrow them as quickly as possible. The planner in [81] similarly performs a “pruning” operation to carefully remove dead branches before rapidly regrowing the tree in critical areas. Another such algorithm is LRF [82], which uses a forest structure and creates “escape paths” from incoming obstacles before reconstructing the tree. The HIRO algorithm [83], rather than reconstructing RRT trees, classifies the points of a roadmap into static and dynamic parts, so that only the relevant portions will be recomputed. Additionally, it uses fuzzy collision detection to find these dynamic areas without the need for costly conventional detection. The LTR* algorithm [84] uses a lazy tree-based sampling scheme (like RRT and Lazy PRM*) to enable quick planning of initial paths with relevant reconstruction, while speeding up subsequent queries using an “experience graph”. The approach presented in [85] similarly uses an updating PRM structure.

The most recent and state-of-the-art sampling-based planner for this setting is RRTX [70]. This approach is related to RRTSharp and performs better than it in static settings, while also being able to handle dynamic planning tasks. RRTX uses a relaxation-like scheme, wherein it iteratively improves its solution after it is generated. It is tailored for dynamic environments, continuously updating the path as conditions change, thus maintaining a valid and optimal route over time.

3.5.2 Non-Reactive Dynamic Planners

Non-reactive dynamic path planners (often represented as temporal planners) focus on pre-computing paths in environments where changes are predictable or occur over longer time scales. These planners typically rely on models or predictions about the environment’s dynamics, planning paths that account for expected changes without needing real-time adjustments. The strength of non-reactive planners lies in their ability to optimize paths considering future states of the environment, which is particularly useful in applications with known or regular patterns of change. However, they are less adaptable to sudden, unforeseen alterations in the environment, making them more suited to scenarios where changes are either slow-moving or can be accurately predicted in advance.

One of the most recent contributions in this area is Temporal PRM [86], an approach that combines PRM with time-aware A* [14]. Similarly, the planner in [87] combines linear temporal logic with an iterative sampling strategy to generate trajectories. Likewise, the approach in [88] uses signal temporal logic together with sampling-based strategies to solve dynamic planning problems.

3.6 Artificial Potential Fields

Though the focus of this survey has been primarily on RRT and PRM, there is another planner that is highly relevant to continuum manipulators: artificial potential fields. These planners are relevant

3.6. ARTIFICIAL POTENTIAL FIELDS

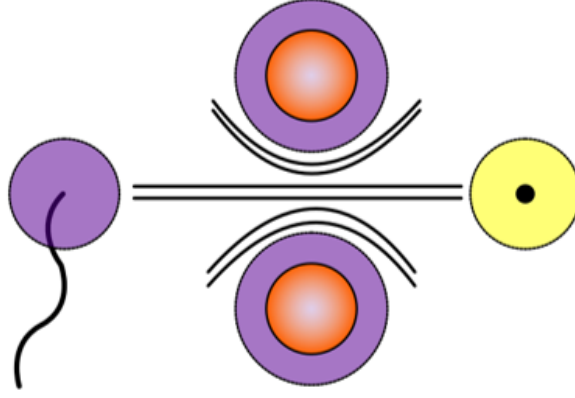


Figure 3.5: A depiction of a continuum manipulator being influenced by an APF. The tip of the arm and the obstacles (orange) share the same polarization (purple) so they are repulsed. The tip and the goal tip position share opposing polarization (purple vs yellow) so the two are attracted to each other.

to nearly every type of robot and are often applied early when testing new models. Furthermore, potential fields are often used as a tool in more sophisticated planning methods, frequently as a biasing or obstacle-avoidance mechanism. We explore their usage for continuum manipulators in the following subsections.

3.6.1 Artificial Potential Fields

Artificial potential fields (APF) [89] are a near-ubiquitous approach for path planning and have been applied to a wide array of robotic models. APFs function like magnets, simulating attractive and repulsive forces to move a robot from an initial state to a target state by calculating the net change in these forces. In manipulators, these APFs often involve IK [90], as the repulsive and attractive forces must be calculated in the $\mathcal{W}$ -space and then mapped to $\mathcal{C}$ -space forces. An interesting characteristic of such planners is that they work for both static and dynamic planning cases. In the static case, assuming that the repulsive forces do not prevent progress towards the target, a path can be found by iterating over the forces. However, in the dynamic case, the position and therefore strength of the repulsive forces are changing as the obstacles move. This means that APFs are a natural solution to the dynamic path planning problem. However, these approaches are not successful in situations where narrow corridors between obstacles must be navigated, or in situations where the obstacle motion repeatedly causes the robot to traverse backwards (away from the goal). For more information on APFs, we point the reader towards standard robotics texts like [91], [92]. Figure 3.5 depicts a continuum arm and an APF.

As mentioned in previous sections, one of the seminal works on random path planning was presented in [54]. This approach used a random-walk scheme to overcome issues of minima and singularities (a notable challenge for continuum arms). Though these general APF implementations are powerful, there have been several papers published on the use of APFs for redundant manipulators. These approaches are largely either trying to use techniques to avoid Jacobian-based IK [93], [94], or to improve upon either the APF or IK in a way that overcomes the challenges of IK [95], [96], [97], [98]. There are also approaches that combine RRT or PRM with APF, such as [99], [100], [101], [102] often to bias growth towards a target and away from obstacles.

There also exist implementations of APF for continuum manipulators. In [103], the authors present a simple APF that is applied to a multisection pneumatic continuum arm using a Jacobian method, though this method was shown to produce low quality paths in [2]. The researchers referenced in [104]

3.6. ARTIFICIAL POTENTIAL FIELDS

devised a similar APF strategy with significant modifications for application to continuum arms, though their work specifically targets a three-section cable-driven model. Similarly, the authors in [105], [106] use $\mathcal{W}$ -space Jacobians to create and apply forces to the continuum arm to achieve dynamic planning. Tangentially related is the Jacobian-based planner in [107] which uses a similar Jacobian model and an APF-like approach to motion planning. The approach presented in [90] eschews the Jacobian method and uses a Virtual Guiding pipeline to calculate configuration changes.

Chapter 4

Graph-Theoretic Planning

In this chapter, we explore two distinct graph-theoretic approaches to the path planning problem: the Look-up Table and IK-based graph-theoretic solutions. The development of these approaches aims to address the limitations encountered in IK-chain based solutions for path planning. Despite their ability to produce smooth trajectories, IK-chain based solutions often struggle with reliability issues, frequently encountering local minima and the knotting problem [108], which are particularly prevalent in continuum manipulator models.

To circumvent these challenges, the Look-up Table approach employs a $\mathcal{W}$ -space discretization strategy, breaking down the $\mathcal{W}$ -space into a structured cubic grid. This method leverages a precomputed offline look-up table that links specific configurations with corresponding cubes. Additionally, a layered graph is integrated within this cubic grid, enabling the application of the Bellman-Ford algorithm to identify viable paths. The resulting trajectories from this technique have been demonstrated to be significantly more reliable than those derived from IK-chain methods, while still maintaining a degree of smoothness deemed acceptable. However, this approach is significantly dependent on precomputation, which may lead to high computational costs and a lack of flexibility, as the table cannot be easily modified once generated.

The second approach retains the same general graph-theoretic framework and cubic decomposition of the $\mathcal{W}$ -space, but replaces the offline table with an online bulk IK-generation mechanism. In this method, IK solutions are used to populate cubes with candidate configurations, after which the layered graph and Bellman-Ford procedure are applied as before. Although this approach is generally slower than using a look-up table, it provides greater flexibility and often yields smoother trajectories, particularly in tasks that involve tracing and navigating around obstacles.

4.1 Look-up Table Approach

The Look-up Table approach begins with an offline enumeration of a discretized subset of the $\mathcal{J}$ -space of the continuum arm, obtained by discretizing the actuator inputs of the three arm sections to a granularity that remains computationally feasible while preserving useful path-planning quality. These configurations are associated with cubes in a cubic decomposition of the $\mathcal{W}$ -space, yielding a reusable offline look-up table. Online path planning is then performed by computing a cube path in the $\mathcal{W}$ -space and constructing a query-specific auxiliary graph over the configurations associated with the cubes along that path. We proceed to the details.

4.1. LOOK-UP TABLE APPROACH

4.1.1 Cubes-Graph

To support path planning in the $\mathcal{W}$ -space, we discretize the 3-dimensional $\mathcal{W}$ -space by creating a cubic grid, in which each cube has the same dimension (1 unit). From this cubic grid, we construct a graph whose vertices are the cubes in the grid, and whose edges correspond to adjacent cubes. We refer to this graph as the *cubes-graph* and denote it by G_B . An edge in G_B has a weight equal to the Euclidean distance between the centers of the two cubes corresponding to the endpoints of the edge. This cubes-graph is generated offline and reused across planning queries. From this graph, it is possible to compute a path between any two points in the $\mathcal{W}$ -space by identifying their corresponding cubes and searching over G_B .

Given two points p_s and p_g in the 3-dimensional $\mathcal{W}$ -space, we first determine the two cubes B_s and B_g containing p_s and p_g , respectively. We then use G_B to find the shortest—with respect to Euclidean distance—*cube path* between B_s and B_g , which is a sequence of adjacent cubes in the cubic grid that starts at B_s and ends at B_g . This can be done by applying Dijkstra’s algorithm to G_B . Let

$$P_{sg} = (B_0 = B_s, B_1, B_2, \dots, B_r = B_g).$$

4.1.2 Offline Look-up Table

A key component of this approach is a precomputed offline look-up table that associates cubes in the $\mathcal{W}$ -space with configurations of the continuum arm. After discretizing the actuator inputs of the three arm-sections, we enumerate a large finite collection of configurations in the $\mathcal{J}$ -space. For each sampled configuration $q \in \mathcal{J}$, we compute the corresponding tip point $p(q)$ in the $\mathcal{W}$ -space, determine the cube B containing that point, and store q in the list associated with B . Thus, each cube is assigned a set of candidate configurations whose tip positions lie within that cube.

More formally, for each cube B in the cubic decomposition of the $\mathcal{W}$ -space, we define

$$Q(B) = \{q \in \mathcal{J} : p(q) \in B\},$$

where $p(q)$ denotes the tip position of the continuum arm under configuration q . The collection of these sets constitutes the offline look-up table, which is generated once and reused across planning queries.

4.1.3 Auxiliary Graph

We define an auxiliary graph, G_Q , whose vertices correspond to configurations drawn from a precomputed (offline) enumeration of the $\mathcal{J}$ -space, obtained by discretizing the actuator inputs, and whose edges correspond to adjacent configurations, i.e., configurations that are obtained from a small change. Since one of our goals is achieving smooth path planning, we associate weights with the edges of the auxiliary graph that reflect the change in orientation between the two configurations that are endpoints of the same edge. More specifically, for each $q^J \in \mathcal{J}$, we calculate the corresponding $q^C \in \mathcal{C}$, and use the curve parameters in (2.3) to measure the Euclidean distance between the corresponding curve-parameter representations. This value is then used as the edge weight in G_Q . Thus, G_Q captures feasible local transitions between nearby configurations.

4.1.4 Path Planning

In the path planning problems under consideration, we are given two points, a starting point p_s and a goal point p_g , in a 3-dimensional $\mathcal{W}$ -space, and we wish to compute a smooth p_s - p_g path for the

Algorithm 1: Layered $\mathcal{J}$ -space Path Planning

Input: Start point p_s ; goal point p_g ; cubes-graph G_B
Output: A sequence of configurations as a path P

- 1 Determine cubes B_s, B_g containing p_s, p_g ;
- 2 Compute shortest cube path $P_{sg} = (B_0 = B_s, \dots, B_r = B_g)$ in G_B ;
- 3 **for** $i = 1$ **to** r **do**
- 4 Retrieve configuration set Q_i corresponding to cube B_i from the offline look-up table;
- 5 **end**
- 6 Let Q_0 be the singleton containing the configuration corresponding to p_s ;
- 7 **if** *obstacles present* **then**
- 8 **for** $i = 0$ **to** r **do**
- 9 Remove invalid configurations from Q_i ;
- 10 **end**
- 11 **end**
- 12 Initialize cost of configuration in Q_0 to 0;
- 13 Set cost of all other configurations in $\bigcup_{i \in \{0, \dots, r\}} Q_i$ to ∞ ;
- 14 **for** $i = 0$ **to** $r - 1$ **do**
- 15 **forall** *adjacent configurations* (q, q') *with* $q \in Q_i$ *and* $q' \in Q_{i+1}$ **do**
- 16 Compute edge weight using curve-parameter distance;
- 17 Relax(q, q');
- 18 **end**
- 19 Remove Q_i from memory;
- 20 **end**
- 21 Select configuration $q^* \in Q_r$ with minimum cost;
- 22 Generate configuration path P from the configuration in Q_0 to q^* ;
- 23 **return** P ;

continuum arm. We consider both cases: where the $\mathcal{W}$ -space is obstacle-free and where it contains obstacles.

Given the start point p_s and the goal point p_g , we first determine the two cubes B_s and B_g containing p_s and p_g , respectively, and then find a shortest cube path P_{sg} between them in the cubes-graph G_B . Using the offline look-up table from Subsection 4.1.2, we retrieve the set of configurations Q_i whose corresponding points in the $\mathcal{W}$ -space fall within cube B_i , for $i = 1, \dots, r$; we let Q_0 be the singleton containing the configuration corresponding to the initial point p_s . In the case where obstacles are present, we purge from each Q_i the configurations that are invalid because they correspond to a position in which the continuum arm intersects an obstacle in the $\mathcal{W}$ -space. In order to test that, for each configuration, we keep track of intermediate points in the $\mathcal{W}$ -space along the three sections of the continuum arm, and use those points to detect intersections with obstacles.

The goal now becomes to compute a shortest path in G_Q that starts at Q_0 , proceeds to visit a vertex in each Q_i , for $i = 1, \dots, r - 1$ (in the listed order), and ends at a vertex in Q_r . To compute such a path, and in order to reduce memory storage (as storing the whole configurations graph G_Q turns out to be quite inefficient) and exploit parallelism, we employ a shortest-path algorithm that takes advantage of the structure of the layered graph, G_L , whose vertex set consists of the configurations in

$$\bigcup_{i \in \{0, \dots, r\}} Q_i,$$

4.1. LOOK-UP TABLE APPROACH

and whose edge set consists of those edges in G_Q that join configurations in consecutive layers Q_i and Q_{i+1} , for $i \in \{0, \dots, r-1\}$.

For this purpose, we apply a variant of the Bellman-Ford algorithm (see Subsection 2.4.4 or [12]) for computing a shortest path between the single configuration in Q_0 and every vertex in Q_r . Here, we do not insist on reaching p_g , and settle for any configuration corresponding to a point in B_g , and hence close enough to p_g , i.e., within distance $\sqrt{3} \cdot d$ from p_g , where d is the cube dimension. The algorithm starts by initializing the cost of the configuration in Q_0 to 0, and the cost of every other configuration in G_L to ∞ , which is a sentinel value that is assumed to be larger than any concrete number. The algorithm employs the subroutine **Relax**(u, v), which operates on edge uv and relaxes uv by updating $cost(v)$ to become $cost(u) + wt(u, v)$, in case the current value of $cost(v)$ exceeds $cost(u) + wt(u, v)$. The algorithm then performs r iterations, where in iteration i , $i = 0, \dots, r-1$, the algorithm relaxes all edges between layer Q_i and layer Q_{i+1} in G_L . This computation is parallelized by running it separately on the set of all edges incident to the same node in Q_{i+1} . Moreover, after iteration i , the vertices in Q_i are no longer useful during the rest of the algorithm, and hence can be removed from memory and replaced with those in Q_{i+2} . At the end of step $r-1$, we have computed the shortest path between the single configuration in Q_0 and every vertex in Q_r . The shortest among all those paths yields the desired configuration-space path. The pseudocode for this method is provided in Algorithm 1.

4.1.5 Test Environment

We implemented and tested our algorithm on a Windows-10 computer, with 64-bit CPU, 6 core 3.20 GHz processor, and 64 GB RAM. Several considerations were made throughout the project to reduce the CPU and memory load. First of all, most of the calculations were performed using the “NumPy” Python library [109]. The array and matrix operations in “NumPy” are more efficient than the standard library, as they are optimized to use less memory and CPU time. We encoded the cube indices and the coordinates of the points as decimal numbers to save memory. In order to determine the points in the $\mathcal{W}$ -space, given their corresponding configurations, we relied on large matrix multiplication. To speed up this process, two features were used. First of all, the matrix is modified so that it can be vectorized; this has the effect of parallelization without any explicit threads and improves performance. The other feature we used is the “JIT” compiler of the Numba Library [110] to improve performance. (For instance, the function that calculates the shortest distance between points on the path is aggressively vectorized.) These techniques brought the time taken to generate the $\mathcal{J}$ -space graph down from 16 hours to 10 minutes. To avoid recalculating the graph each time, we used the NumPy save function and the “pickle” python library¹ to store the graph, cube list, and point list on disk for easy reuse. Lastly, in order to perform graph operations, we used the python library “igraph”[111]. This is a high-performance library that handles all of the graph generation and simple path planning for the cubes.

To test our algorithm, we discretized the three sections of the continuum arm; each section has 2 DoF, and each DoF was uniformly discretized into (roughly) 26 steps. This resulted in a sampling of 688 configurations for the tip of each section. Thus, the number of vertices in the $\mathcal{J}$ -space graph is 325,660,672. Based on the points in the $\mathcal{W}$ -space corresponding to the sampled configurations, we placed a bounding box of X - Y - Z dimensions 82 cm $\times$ 82 cm $\times$ 74 cm that contains all these points. This bounding box was discretized into cubes, each of dimension 1 cm.

We ran 2,000 tests, 1000 tests in which the $\mathcal{W}$ -space is obstacle-free, and 1000 tests in which the $\mathcal{W}$ -space contains either 2 or 3 spherical-shape obstacles. The starting and ending points of the path were generated randomly from the $\mathcal{W}$ -space (by randomly choosing two configurations), and so was

¹<https://docs.python.org/3/library/pickle.html>

4.1. LOOK-UP TABLE APPROACH

the center of the spherical-shape obstacles; the radius of the obstacles was randomly chosen from the interval [3 cm, 25 cm]. We then located the two cubes containing the starting and ending points; computed the shortest cube-path between these two cubes in the cubes-graph; and then ran our algorithm on the corresponding layered graph resulting from the cubes on this cube-path, as explained in the previous section. The IK approach was run on the same generated test cases. To implement the IK approach, we used the Sequential Least Squares Programming (SLSQP) method provided in the Python Scipy Package. This method allows for obstacle representation using constraints.

4.1.6 Results

Table 4.1 below summarizes the results obtained for both approaches. The third column of the table shows the success rate of the IK approach and the fourth column shows the average time (in seconds) taken by the IK approach (overall cases tested). Columns 5 and 6 show the corresponding data for our algorithm (Alg), given in the previous section. Figure 4.1 shows a test case in an obstacle-free space where both the Look-up Table and the IK approaches succeeded in finding a solution. Given the (continuous) nature of the IK approach, the resulting path generated by the IK approach is shorter and smoother than the one computed by the Look-up Table approach. Our approach is limited by the rough discretizations of the $\mathcal{J}$ -space and $\mathcal{W}$ -space in order to ensure a feasible computation. The reliability of our approach stems from the fact that it has a priori access to the entire (discretized) $\mathcal{J}$ -space, which it could use to avoid “knotting” or running into local minima. As shown in Figure 4.2, in this example our planner was able to find a path, whereas the IK approach did not even converge to a solution. It can be seen that the base and the midsections of the continuum arm form a Knot [108], as the IK approach tries to minimize the distance to the target point but the mid and base sections are already at the maximum bending. Consequently, the IK approach failed to yield a solution. This is the common reason behind the failure of IK-based continuum arm path planners. Since the $\mathcal{J}$ -space graph—constructed according to a proper discretization—allows us to capture a wide range of motion for the continuum arm covering the entire task-space, our algorithm has the ability to adjust itself, when started from certain initial configurations, to reach the destination point, in situations where the IK approach, starting from the same initial configurations, is incapable of doing so, and diverges to points that are far away from the destination points.

Table 4.1: Summary of the numerical results.

Scenario	# Cases	I. K. Rate	I. K. Time	Alg. Rate	Alg. Time
No obstacles	1000	74%	0.01	100%	75.43s
Obstacles	1000	68.67%	0.45	100%	85.55s

4.1. LOOK-UP TABLE APPROACH

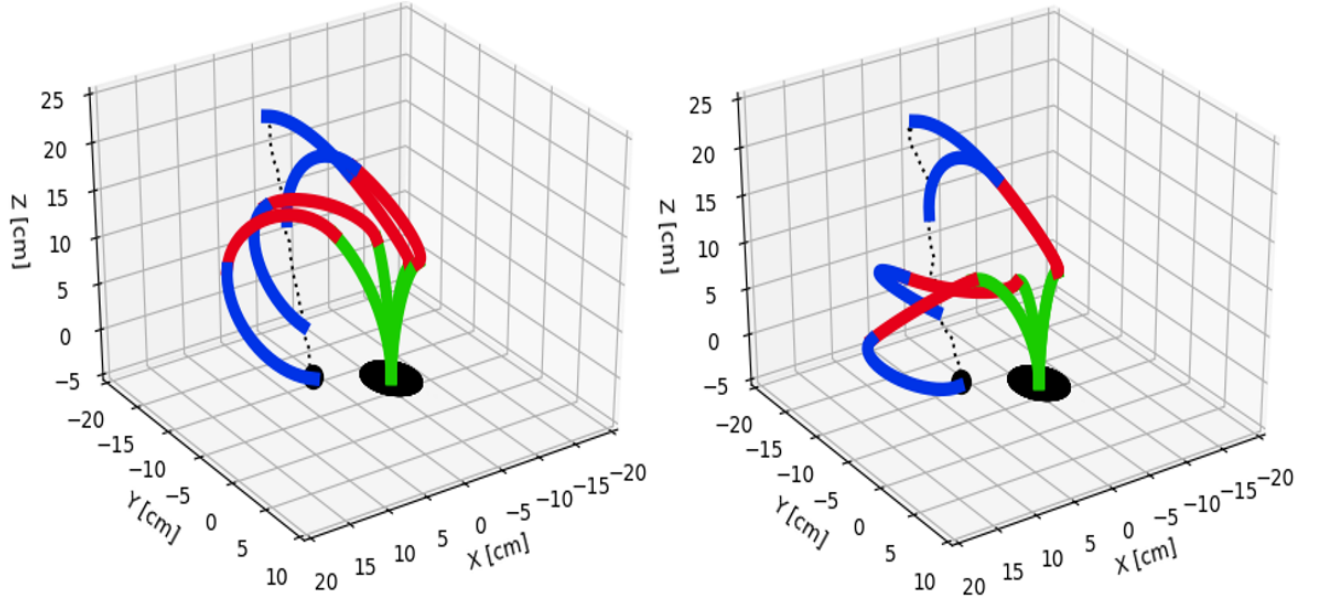


Figure 4.1: Illustrative test case where both the Look-up Table approach (Right) and the IK (Left) successfully found solutions. No obstacles were present in the workspace. The top point is the starting point and the black dot is the end point.

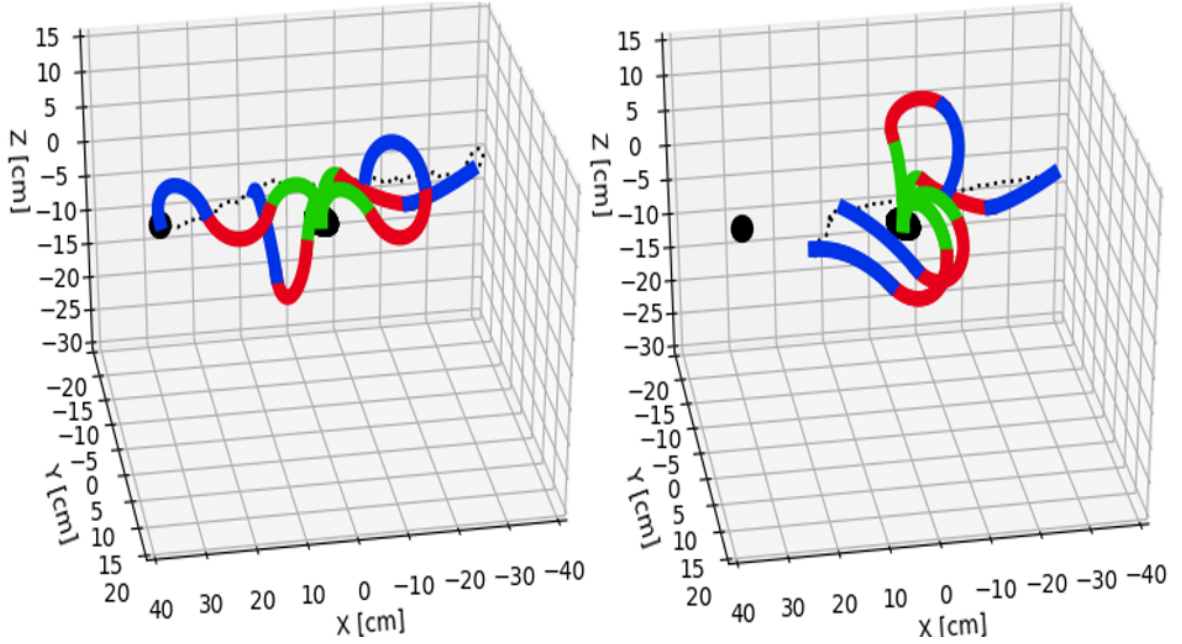


Figure 4.2: An example test case where the Look-up Table approach (Left) converged to a solution, but the IK (Right) was unsuccessful. No obstacles were present in the workspace. The rightmost point is the beginning point and the leftmost black dot is the ending point.

4.2. INVERSE KINEMATICS FOR IMPROVED CONFIGURATION GENERATION

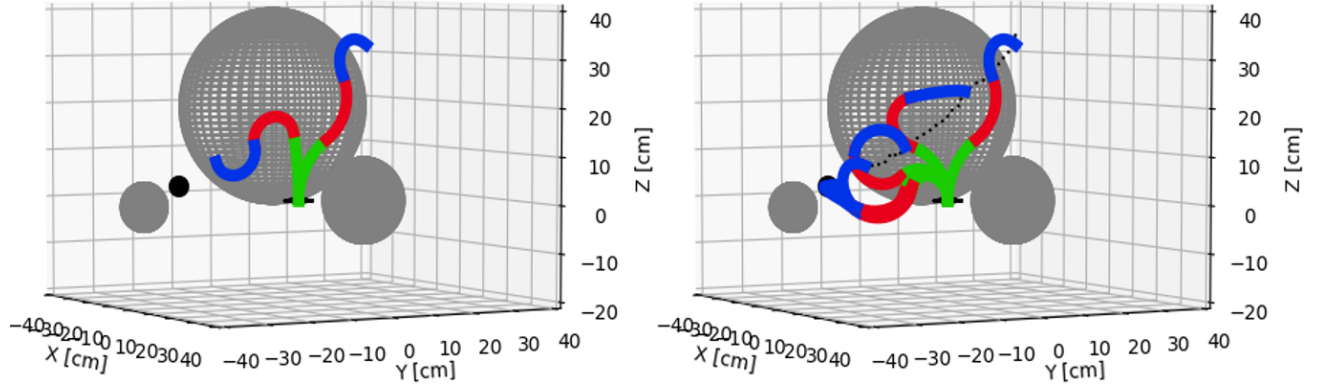


Figure 4.3: One scenario in which the Look-up Table approach (right) was able to find a path from the starting point to the end point amidst obstacles whereas the IK (left) planner failed.

Figure 4.3 shows a test case with obstacles. Due to the highly-constrained nature of the problem, unlike the obstacle-free scenario, in this case, it is not possible to generate a meaningful shortest path between the starting and end point. Therefore, in favor of a fair comparison with the IK approach, we removed the shortest path requirement so that the IK planner is only required to find *a* path. In the IK planner, we formulated the obstacle avoidance as a constrained function; we considered ten points along a continuum section (30 points total), and the constraint function must maintain a positive distance between all the points and obstacles. In contrast, the Look-up Table planner solves for the shortest path. As shown in Figure 4.3, in this test, the IK failed to converge. Similarly to the obstacle-free test case shown in Figure 4.2, the IK planner was stuck in a local minima (knot configuration), whereas, due to the a priori knowledge of the $\mathcal{W}$ -space, our planner was able to find poses that circumvent configurations associated with local minima.

Since we generate the obstacles randomly, there is the possibility of having cases where there is no solution; for instance, this is the case if the obstacles form a separator in the $\mathcal{W}$ -space, disconnecting the starting point from the endpoint. We discarded such cases when computing the success rate in Table 4.1, and computed the success rate of IK as a percentage of the successful cases, i.e., in which a path exists. (Note that, given how our planner works, it is able to find a solution if there is a path.) Consequently, the Look-up Table approach is near-optimal and reliable for planning paths of multisection continuum arms in spaces with or without obstacles. In addition, it facilitates the introduction of multiple optimization criteria—such as smooth trajectories, in contrast to the IK approaches, which will increasingly fail as the number of constraints grow.

4.2 Inverse Kinematics for Improved Configuration Generation

In this section, we describe an IK-based graph-theoretic path planning method that uses online IK-based configuration generation in place of the precomputed offline look-up table of Subsection 4.1.2. More specifically, we use IK to generate a rich set of configurations whose corresponding $\mathcal{W}$ -space points lie within the vicinity of a $\mathcal{W}$ -space target point. Here, richness refers to sets of configurations that exhibit a wide variety of valid spatial shapes. We use an objective function that takes a starting configuration and minimizes the distance between the arm tip and the target point.

As in the Look-up Table approach, the procedure begins by generating a cube path. The cubes along this path are then populated on-the-fly using IK. Afterwards, the layered graph is formed, and path planning is performed using the same general framework as in Section 4.1. This formulation allows for greater flexibility in the number of configurations generated per cube, the size of the cubes,

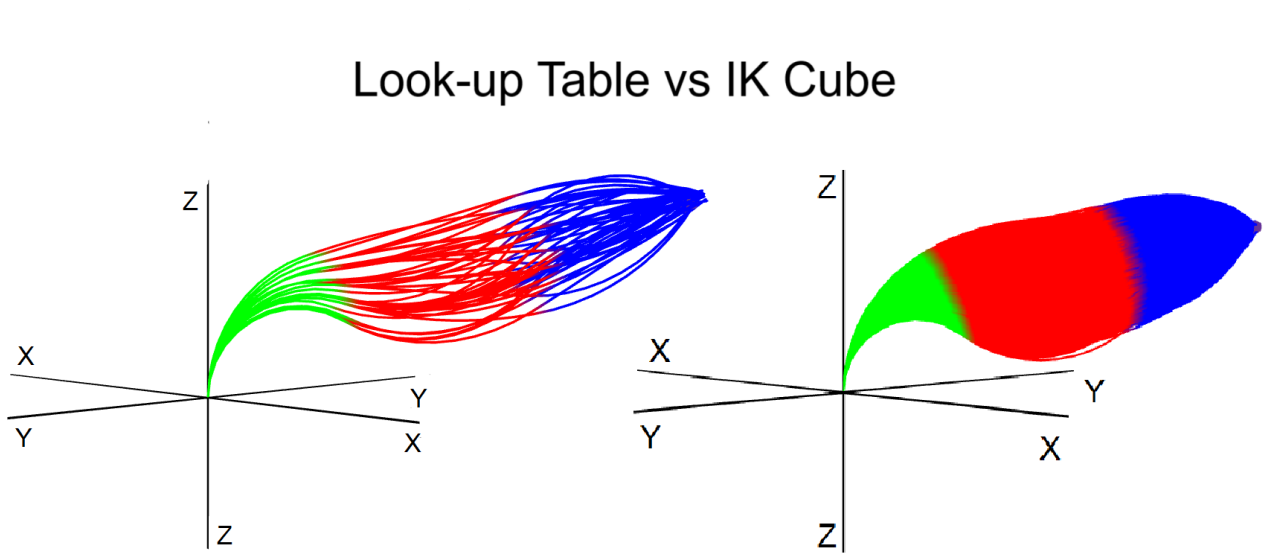


Figure 4.4: A comparison for the cube (26,26,15) of the configurations stored in the Look-up Table vs. the Cube Configuration Generation Method

and the amount of precomputation required for path planning. Additionally, cubes that are empty or contain too few valid configurations can be discarded and replaced with alternative cubes that admit valid or more diverse configuration sets.

4.2.1 Cube Configuration Generation

In order to generate a dense set of configurations in a cube B , we rely on multiple invocations of IK at the point located at the center of B . We begin by generating a large number of random starting configurations. Afterwards, we invoke IK with each of these starting configurations to find a target configuration that places the tip within some small distance of the center of B . This procedure generates a dense set of configurations within B . We refer to this procedure as *Cube Configuration Generation*. Figure 4.4 shows a comparison of the richness between a cube populated with the Cube Configuration Generation method and the look-up table.

4.2.2 Path Planning

Using the Cube Configuration Generation method, we describe an IK-based path planning procedure that does not rely on the offline look-up table of Subsection 4.1.2. As discussed in Subsection 4.1.4, path planning in the Look-up Table approach proceeds by generating a cube path and populating it with configurations from a look-up table. Here, these cubes are instead populated using IK. We consider two variants of this procedure, corresponding to two subroutines: the *Global-IK* subroutine and the *Sequential-IK* subroutine. The Global-IK subroutine populates the first cube using the Cube Configuration Generation method. Each subsequent cube is also populated using the Cube Configuration method by using random starting points. The Sequential-IK subroutine populates the first cube using the Cube Configuration Generation method. Each subsequent cube is populated with the configurations from the previous cube used as starting points. This means that only the first cube is generated using random starting points, and each subsequent cube is populated using the output from previous invocations. After populating the cubes, a path can be generated using the layered graph

4.2. INVERSE KINEMATICS FOR IMPROVED CONFIGURATION GENERATION

Algorithm 2: Global-IK Subroutine

Input: A cube path P
Output: A list of configuration sets L

```

1 foreach cube in  $P$  do
2   Create a set of configurations  $S = \emptyset$ ;
3   repeat  $n$  times
4     Generate a random starting configuration start;
5     Use the cube's center point as the target;
6     Use the Cube Configuration Generation method as the objective function;
7     Use IK with parameters start, target, and objective function to generate a (new)
      configuration and add it to  $S$ ;
8   end
9   Add  $S$  to  $L$ ;
10 end
11 return  $L$ 

```

and weight metrics discussed in Section 4.1. These methods, when paired with the two subroutines, form the *Global-IK Path Planning Approach* and *Sequential-IK Path Planning Approach* respectively. See Algorithms 2 and 3 for the pseudocode of the two subroutines.

4.2.3 Time as a Metric

In Section 4.1, the Euclidean distance between configurations was used as edge-weight when constructing the layered graph. This metric prioritizes the smallest overall change in configurations in the path. Here, we propose to use time, as described in Subsection 2.5.4, as the edge-weight in the layered graph so that the new metric prioritizes the shortest time instead. That is, for consecutive configurations q_{k-1} and q_k , the weight between them is calculated as $weight(q_{k-1}, q_k) = t_{shortest}$. Intuitively, the path that can be traced by the continuum arm with a high velocity would have less overall changes with respect to its configurations.

4.2.4 Test Environment

Our algorithms were implemented and tested on a Windows-10 computer, with a 64-bit Intel i7-9700K CPU, 8 core 3.60 GHz processor, and 32 GB RAM. We use the Look-up Table approach from Section 4.1 as a basis for comparison with the IK-based approaches. It is implemented as described in Section 4.1. To test the Look-up Table algorithm from Section 4.1, we used the offline enumeration method discussed in Subsection 4.1.2 to generate 327,082,769 configurations. Both of the IK-based approaches and the Look-up Table approach were implemented to be time weighted, as discussed in Subsection 4.2.3.

All approaches used a cube size/dimension of 1 *cm* for a fair comparison, though the IK approaches are capable of using any cube size without re-enumeration. When the Look-up Table approach was run with smaller cube sizes, there were many holes in the graph that lead to jagged, nonsmooth paths. These holes may be filled with a denser $\mathcal{C}$ -space, but that would necessitate a re-enumeration of all of the configurations, which was a stated shortcoming of this approach.

To test our approaches, we generated two sets of varied target trajectories that the IK-based and Look-up Table approaches had to trace. These included lines with obstacles and demonstrative geometric shapes. The first set of shapes were small and placed mostly in the center of the $\mathcal{W}$ -space.

4.2. INVERSE KINEMATICS FOR IMPROVED CONFIGURATION GENERATION

Algorithm 3: Sequential-IK Subroutine

```

Input: A cube path  $P$ 
Output: A list of configuration sets  $L$ 
1 for cube  $c$  in  $P$  do
2   Create a set of configurations  $S = \emptyset$ ;
3   if cube is first then
4     repeat  $n$  times
5       Generate a random starting configuration  $start$ ;
6       Use the cube's center point as the  $target$ ;
7       Use the Cube Configuration Generation method as the objective function;
8       Use IK with parameters  $start$ ,  $target$ , and objective function to generate a (new)
        configuration and add it to  $S$ ;
9   end
10  else
11    foreach configuration  $c$  in the previous cube do
12       $start = c$ ;
13      Use the cube's center point as the  $target$ ;
14      Use the Cube Configuration Generation method as the objective function;
15      Use IK with parameters  $start$ ,  $target$ , and objective function to generate a (new)
        configuration and add it to  $S$ ;
16    end
17  end
18  Add  $S$  to  $L$ ;
19 end
20 return  $L$ 

```

An additional, larger set of trajectories was created that focused on cubes towards the edge of the $\mathcal{W}$ -space. On both sets of trajectories, paths were generated that closely traced the desired trajectory with the arm tip. Next, these generated paths were validated using the dynamic model (discussed in Section 2.1.4). We stretched or compressed the time for these paths until the Root Mean Square tip error between the generated path and the dynamic path, which we use as the acceptable error, was less than 1 *cm*. Any error larger than that would indicate oscillation and instability. In some cases, this may not be possible, meaning that the path is unsuccessful. Otherwise, successful paths are deemed “smoother” if they have a lower time.

4.2.5 Trajectory Smoothness without Obstacles

Circles, figure-8 (the shape resembling the infinity symbol), and rotated cube trajectories were created for the arm to trace. Circles and figure-8 trajectories were two dimensional geometric shapes with a fixed Z coordinate. Rotated cubes were generated as fully three dimensional trajectories.

Table 4.2 shows the time needed to complete the tracing of the shapes. The small shapes are displayed above the double lines, and the large shapes are displayed below it. *LT* is the Look-up Table approach, while *IK-Global* and *IK-Seq* are the Global-IK and Sequential-IK approaches, respectively. “Failure” indicates that the trajectory could not be computed with a given approach. The term *DSF* indicates that the dynamic simulator could not find a tip error less than 1 *cm*. A shorter time indicates

4.2. INVERSE KINEMATICS FOR IMPROVED CONFIGURATION GENERATION

Table 4.2: Summary of Time Results for small shapes (above line) and large shapes (below line)

Approach	LT Time(s)	IK-Global Time(s)	IK-Seq Time(s)
Circle 1	24.46	16.08	23.67
Circle 2	13.81	11.01	12.90
Figure-8	20.82	17.03	18.02
Rotated Cube	110.14	86.45	83.53
Circle 1	148.96	64.59	64.52
Circle 2	DSF	68.37	67.19
Circle 3	Failure	57.71	57.01
Figure 8	81.61	53.71	56.47
Rotated Cube	245.34	116.78	131.64

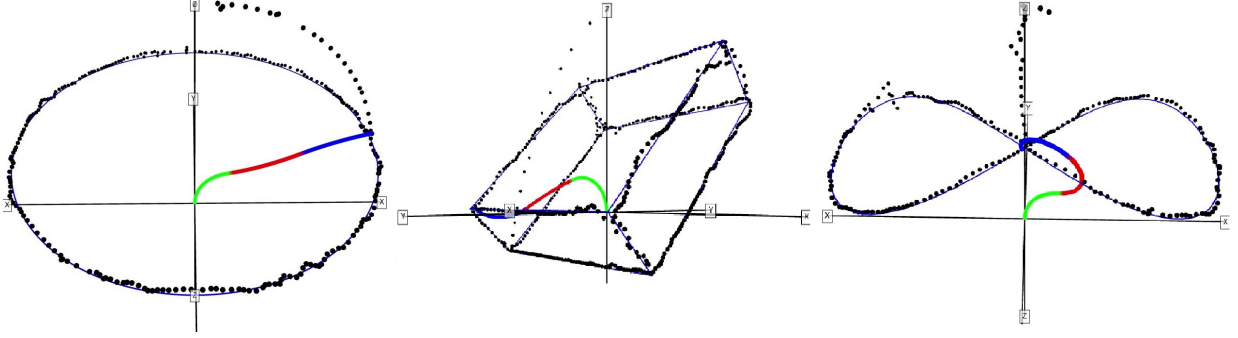


Figure 4.5: A 3D view of the Sequential-IK approach simulated on the circle, rotated cube, and figure-8 trajectory. The traced path is shown in black and the target trajectory is shown in blue.

that the dynamic simulator was able to simulate the arm moving at a higher velocity. As discussed before, shorter times are desirable, so assuming no oscillation in our generated trajectories, a shorter time indicates a better performance for the algorithm.

In Table 4.2, we see that there are not substantial differences in time between the Look-up Table, Global-IK, and Sequential-IK approaches for the small shapes. However, it seems that IK-based approaches have a slight edge. Given that most of the redundancy in the $\mathcal{W}$ -space occurs in the center, the cubes towards the center of the $\mathcal{W}$ -space have quite rich and dense sets of configurations. In the case of the smaller trajectories, this means that the approaches are similar in the overall number of configurations that can be used, hence the similar times. For the large shapes in Table 4.2, however, this is not the case. The larger trajectories focus on cubes towards the edge of the $\mathcal{W}$ -space where there is far less redundancy. Given how extended the arm must be to reach these points, there are fewer unique spatial shapes to assume towards the edge of the $\mathcal{W}$ -space. In the case of the Look-up Table algorithm, the restricted discretization of the $\mathcal{C}$ -space limits the number of configurations in each one of the edge cubes. However, the IK-based approaches, with no offline configurations, can find an acceptable rich set of configurations. When applied to path planning, the IK-based approaches have far more configurations to choose from, meaning that IK-based approaches have a distinct advantage.

Figure 4.6 compares the desired and achieved trajectories for a circle, figure-8, and rotated cube as traced by the Look-up Table and IK-based approaches. Both trajectories have a tip error of less than 1 cm. Despite similar tip errors, the IK-based results are smoother and had a lower time in each case. Figure 4.5 shows the arm tracing large circle, figure-8, and rotated cube trajectories in 3D for the Sequential-IK approach.

4.2. INVERSE KINEMATICS FOR IMPROVED CONFIGURATION GENERATION

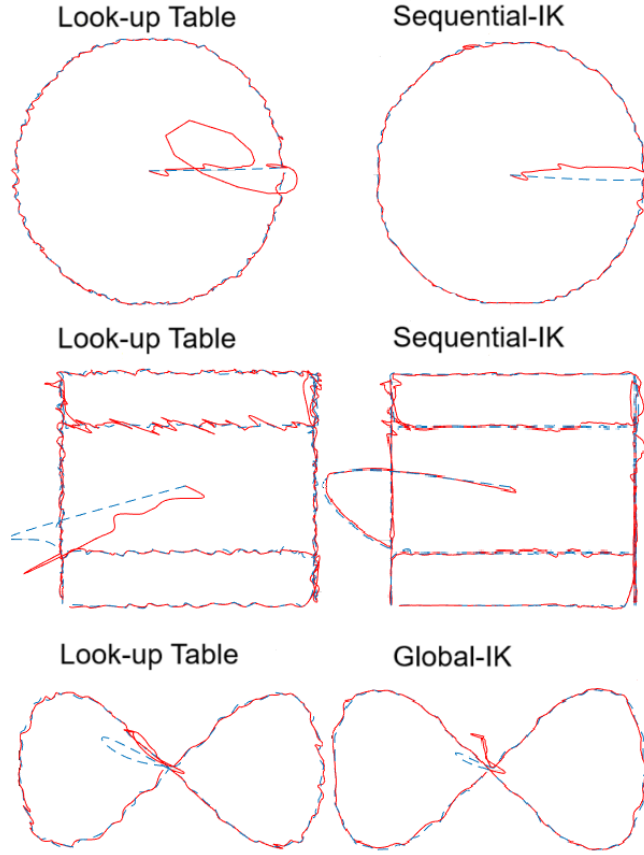


Figure 4.6: A top down view of the results from the Dynamic Simulator for a circle, rotated cube, and figure-8 trajectory. The trajectory traced by the dynamic simulator (red) is displayed over the calculated trajectory (blue).

Table 4.3: Summary of Time Results for trajectories with obstacles

Approach	LT Time(s)	IK-Global Time(s)	IK-Seq Time(s)
Obstacle Trial 1	14.34	16.43	9.44
Obstacle Trial 2	12.89	14.22	7.11

4.2.6 Trajectory Smoothness with Obstacles

We also performed tests with straight-line trajectories in the presence of obstacles as a comparison to the results in Subsection 4.1.6. Starting and ending points were given along with three obstacles. These were spherical obstacles placed randomly within the arm's reach and had a random radius in the range of $[4, 15]$. Table 4.3 shows the times for all three approaches. The results here very closely mimic those without obstacles.

Chapter 5

Anticipatory Path Planning

Continuum arms, characterized by their exceptional dexterity and flexible actuation, are ideally suited for operation within highly dynamic environments. These unique attributes, however, contribute to the complexity of the dynamic path planning problem, a task further complicated by factors such as redundancy and intricate kinematics inherent to continuum arms. This chapter is dedicated to exploring an anticipatory path planning strategy specifically tailored for continuum arms, integrating methodologies from the Look-up Table approach (referenced in Chapter 4) with the novel inclusion of a temporal graph.

The temporal graph framework facilitates an efficient exploration of graph structures annotated with time stamps, enabling the system to account for changes in the environment over time. By merging this temporal dimension with the spatial organization of the cube-graph structure, alongside a predictive model for dynamic obstacles, the Anticipatory approach is able to plan through temporally “available” segments of the graph. A comparison shows that this Anticipatory path planning technique markedly surpasses the reliability of both simplistic heuristic strategies and the conventional potential field method, presenting a significant advancement in the domain of path planning for continuum arms in dynamic settings.

5.1 Anticipatory Path Planning Algorithm

In this section, we develop a temporal path planning scheme for dynamic environments. The scheme utilizes obstacle prediction and temporal graphs to accomplish path planning in a dynamic environment. We give a high-level description of this scheme before proceeding to the details.

Suppose that the arm tip is at some current cube B_{cur} (initially the starting cube B_s). First, we predict the motion of the obstacles for h steps in the future based on the historical data of the obstacles’ motion that we have collected. Second, we construct a (local) temporal cubes-graph, G_T , centered around B_{cur} , that extends h hops in all directions by employing both the cubes-graph (described in Section 4.1) and our prediction of the obstacles’ motion. The temporal cubes-graph G_T allows us to model the availability and unavailability of certain areas in the $\mathcal{W}$ -space during certain time windows. Finally, we compute a subpath (of cubes) in this temporal cubes-graph from B_{cur} to the cube B_ℓ that is the closest cube in G_T to the goal cube B_g . The above process is then repeated starting at B_ℓ until B_g is reached.

5.1.1 Cubes-Graph, Cube Path, and Layered Graph

We briefly recall the cubes-graph, cube path, and layered graph constructions introduced in Chapter 4, which form the static planning backbone used in our Anticipatory framework.

To discretize the $\mathcal{W}$ -space, we construct a cubic grid in which each cube has the same dimension. From this grid, we define the cubes-graph, G_B , whose vertices correspond to cubes and whose edges connect adjacent cubes. Each edge is assigned a weight equal to the Euclidean distance between the centers of its corresponding cubes. This structure allows us to efficiently compute paths between regions of the $\mathcal{W}$ -space. Given a starting point p_s and a goal point p_g , we identify the corresponding cubes B_s and B_g , and compute a cube path $P = (B_s, \dots, B_g)$ in G_B using a shortest-path algorithm such as Dijkstra's algorithm. This cube path provides a sequence of regions in the $\mathcal{W}$ -space through which the arm must pass.

Based on this cube path, we construct a layered graph of configurations. First, the $\mathcal{J}$ -space is enumerated (with respect to a chosen discretization), and each configuration is mapped to its corresponding $\mathcal{W}$ -space point. For each cube in P , we collect the set of configurations whose $\mathcal{W}$ -space points lie within that cube, forming a sequence of configuration sets. Any configurations that result in collisions with obstacles are removed by checking intermediate points along the arm. The layered graph is then defined by connecting configurations in consecutive layers: each configuration in one cube is connected to all configurations in the next cube. The weight of an edge between two configurations is defined using the smoothness metric described in Section 5.1.2. A shortest path through this layered graph is then computed using the modified Bellman-Ford algorithm, yielding a smooth $\mathcal{J}$ -space path. These constructions enable path planning in static environments. In the anticipatory setting, we extend this framework by incorporating temporal information and obstacle prediction.

5.1.2 Smoothness Measure

Smoothness is a key measure of the quality of a path for continuum robots. For the present algorithm, we evaluate smoothness using the corresponding curve-parameter representation. Section 2.1 defines the shape curvature parameters θ , ϕ , and λ . Recall that, as discussed earlier, a configuration may be represented either in joint-space form as q^J or in curve-parameter form as q^C , where each section of the arm is described by a pair (ϕ_i, θ_i) . Here, although the path itself is generated over joint-space configurations, we use the corresponding curve-parameter representation only for measuring smoothness. The orientation of the robot's arm can be represented as a vector whose coordinates/components are these parameters.

For a given path, its smoothness can be measured as the cumulative change in orientations over pairs of consecutive configurations along the path. This change is quantified using the Euclidean distance between the orientations of consecutive configurations.

More specifically, consider two consecutive joint-space configurations q_k^J and q_{k+1}^J along a path, and let q_k^C and q_{k+1}^C denote their corresponding curve-parameter representations. Let (ϕ_i^k, θ_i^k) and $(\phi_i^{k+1}, \theta_i^{k+1})$ denote the parameters of section $i \in \{1, 2, 3\}$ for these configurations. The smoothness contribution between q_k^J and q_{k+1}^J is defined as

$$\Delta(q_k^J, q_{k+1}^J) = \sum_{i=1}^3 \sqrt{(\phi_i^{k+1} - \phi_i^k)^2 + (\theta_i^{k+1} - \theta_i^k)^2}.$$

5.1. ANTICIPATORY PATH PLANNING ALGORITHM

For a joint-space path $P = (q_1^J, \dots, q_K^J)$, the smoothness of P is given by

$$S(P) = \sum_{k=1}^{K-1} \Delta(q_k^J, q_{k+1}^J).$$

Our goal is to compute an obstacle-avoiding path P in the $\mathcal{J}$ -space that minimizes this cumulative value $S(P)$. See Sections 2.1 and 2.2.1 for more details about the curvature parameters.

5.1.3 Temporal Model and Obstacle Prediction

As discussed in Chapter 2, dynamic path planning is treated in discrete time. In the anticipatory setting considered here, let $t \in \mathbb{N}$ denote the current planning step, and let $\mathcal{O}(t) \subseteq \mathcal{W}$ denote the true obstacle set observed at that step. Unlike a purely reactive planner, the present method does not rely only on the current obstacle state. Instead, it uses recent obstacle observations to predict obstacle motion over a finite horizon.

More specifically, for each obstacle, the planner stores its most recent observed positions in a fixed-length history buffer H . This historical data is then used to generate predicted future obstacle sets

$$\hat{\mathcal{O}}(t+1), \hat{\mathcal{O}}(t+2), \dots, \hat{\mathcal{O}}(t+h),$$

where $h \in \mathbb{N}$ is the prediction horizon. Thus, H represents the stored past observations, while $\hat{\mathcal{O}}(t+j)$ represents the predicted obstacle occupancy at future step $t+j$.

In this work, obstacle prediction is performed using cubic interpolation [112]. This is a widely used methodology for making predictions based on a small set of data [113]. For a given amount of input data, cubic interpolation constructs a polynomial of degree 3, with time as the variable, that agrees with the observed obstacle positions. This polynomial is then evaluated at future time steps to predict obstacle locations.

As the obstacles move, the r most recent observed locations of each obstacle are maintained in H . Though r may be chosen as desired, in general r should be at least 4, since 4 data points are needed to determine a polynomial of degree 3. At each time step, the oldest observation is removed and the current obstacle position is inserted. Using this history, predictions are generated for the next h time steps, thereby producing the predicted obstacle sets $\hat{\mathcal{O}}(t+1), \dots, \hat{\mathcal{O}}(t+h)$. These predicted obstacle sets are then used in the following subsection to determine which temporal transitions remain feasible during the planning horizon.

5.1.4 Temporal Cubes-Graph and Path

Suppose that the arm tip has reached some current cube B_{cur} at planning step t , and that the predicted obstacle sets $\hat{\mathcal{O}}(t+1), \dots, \hat{\mathcal{O}}(t+h)$ have been computed. A *temporal cubes-graph* G_T is then constructed, and a path in G_T is computed as follows.

Using Breadth-First Search [114], starting from B_{cur} , we find all cubes in the cubes-graph G_B within h hops from B_{cur} . We then define a temporal subgraph G_T whose vertices are the cubes within hop-distance h from B_{cur} , and in which $(u, v, w, \lambda = 1, d)$ is an edge if the arm can move from the cube B_u corresponding to u to the cube B_v corresponding to v during the interval $[w, w+1]$ without colliding with the predicted obstacle occupancy over that interval. Here, d is the Euclidean distance between the centers of B_u and B_v , and we assume that the traversal time λ between any two adjacent cubes is 1.

5.1. ANTICIPATORY PATH PLANNING ALGORITHM

Finally, a local target cube B_ℓ in G_T is selected such that B_ℓ is a closest cube in G_T to the final destination cube B_g . Using a shortest temporal path algorithm [17], a sequence of cubes is found in the temporal cubes-subgraph from B_{cur} to B_ℓ . If B_g is more than h hops away from the current cube, the above process of generating a temporal cubes path is repeated starting from B_ℓ .

We note that, due to the temporal requirement, the shortest path between two vertices in a temporal graph cannot be computed using folklore shortest path algorithms such as Dijkstra’s algorithm. However, a similar efficient algorithm to Dijkstra’s was proposed in [17] and is used in this dissertation. As such, d is not included in the definition of a temporal edge provided in Section 2.4.5 but is added here to enable this algorithm. Algorithm 4 illustrates the construction of the temporal cubes-graph starting at the current cube B_{cur} , for a given number h of hops. The algorithm uses a similar idea to Breadth-First Search.

Algorithm 4: Temporal cubes-graph Construction

input : Current cube B_{cur} , current planning step t , prediction horizon h , goal cube B_g ,
predicted obstacle sets $\hat{\mathcal{O}}(t+1), \dots, \hat{\mathcal{O}}(t+h)$
output : A temporal cubes-subgraph G_T and a cube B_ℓ

- 1 $G_T := \emptyset$;
- 2 Use Breadth-First Search to find the set Ω_{cur} of cubes in G_B within hop-distance h from B_{cur} ;
- 3 Define the vertices of G_T to be the vertices of G_B corresponding to the cubes in Ω_{cur} ;
- 4 Use the predicted obstacle sets $\hat{\mathcal{O}}(t+1), \dots, \hat{\mathcal{O}}(t+h)$ to construct the edges of G_T ; more specifically, there is an edge (u, v, w, λ, d) , where $\lambda = 1$ and d is the Euclidean distance between the centers of the cubes corresponding to u and v , if the arm can move from B_u to B_v without collision during the time interval $[w, w+1]$;
- 5 Using the coordinates of the centers of the cubes in G_T , find a cube B_ℓ in G_T that is closest (w.r.t. Euclidean distance) to B_g ;
- 6 **return** G_T, B_ℓ ;

5.1.5 Path Planning

The Anticipatory path planning algorithm combines obstacle prediction and the temporal cubes-graph model to construct a sequence of configurations that navigates the arm tip from a starting point to a goal point. We assume that we are given an initial configuration q^0 corresponding to the arm tip’s initial position. This configuration is mapped to its corresponding $\mathcal{W}$ -space point p_s . First, we bin the starting point p_s into the starting cube B_s and the goal point p_g into the goal cube B_g . We then wait until enough obstacle observations have been collected to initialize the history buffer H .

Next, the temporal cubes-subgraph at B_s is constructed using the process described in Subsection 5.1.4. After a sequence of cubes from B_s to an intermediate target cube B_ℓ is found, a layered configurations graph L is constructed as described in Subsection 5.1.1 using the configurations in each of these cubes. The first layer includes the starting configuration q^0 , and the final layer includes all configurations whose corresponding $\mathcal{W}$ -space points fall in B_ℓ .

The Bellman-Ford algorithm is run on L , and a sequence of configurations P_L is computed. The first configuration in P_L places the arm tip at p_s (inside B_s), and the last configuration, denoted q^ℓ , places the arm tip inside B_ℓ . These configurations in P_L are appended to a global path ρ , and then the arm progresses to configuration q^ℓ . If the goal point has not been reached, the temporal cubes-graph is reconstructed centered around cube B_ℓ , and the first layer of the new layered graph consists only of configuration q^ℓ .

Algorithm 5: Temporal Shortest Path Approach

input : Starting configuration q^0 , goal point p_g , prediction horizon h
output : A list of configurations ρ

- 1 $\rho := \emptyset$;
- 2 Let p_s be the $\mathcal{W}$ -space point corresponding to q^0 ;
- 3 Observe obstacle motion until enough historical data has been collected to initialize the history buffer H ;
- 4 Let $t := 0$;
- 5 Let $B_{\text{cur}} = B_s$, where B_s is the cube containing p_s , and let B_g be the cube containing the goal point p_g ;
- 6 Let $q_{\text{cur}} := q^0$;
- 7 **repeat** until $B_{\text{cur}} = B_g$
 - 8 Observe the current obstacle set $\mathcal{O}(t)$ and update the history buffer H ;
 - 9 Use H to predict the obstacle motion for the next h steps and compute $\hat{\mathcal{O}}(t+1), \dots, \hat{\mathcal{O}}(t+h)$;
 - 10 Apply Algorithm 4 with B_{cur} , current planning step t , prediction horizon h , goal cube B_g , and predicted obstacle sets $\hat{\mathcal{O}}(t+1), \dots, \hat{\mathcal{O}}(t+h)$, to construct G_T and determine a local target cube B_ℓ in G_T ;
 - 11 Compute the shortest temporal cube path P in G_T from B_{cur} to B_ℓ ;
 - 12 Compute the layered graph L corresponding to P , with first layer consisting only of q_{cur} ;
 - 13 Find a shortest path P_L of configurations in L by running the Bellman-Ford algorithm;
 - 14 Append P_L to ρ ;
 - 15 Let q_{cur} be the last configuration in P_L , and let $B_{\text{cur}} := B_\ell$;
 - 16 Increment the planning step t ;
- 17 **end**
- 18 **return** ρ ;

The process of constructing a temporal cubes-graph, finding a shortest temporal cubes path, constructing a layered graph, and finding a sequence of configurations is repeated until the tip of the arm falls inside the final destination cube B_g . At this point, the global path ρ is returned. See Algorithm 5 for the details.

5.1.6 Other Approaches

In order to evaluate our approach, we compared it against the folklore potential field approach used for path planning of rigid robots. We also implemented two simple greedy heuristics as baseline methods. The empirical results obtained using these greedy heuristics show that the Anticipatory path planning approach for continuum arms is superior to baseline methods, while the empirical results obtained using the potential field approach show that the Anticipatory method is also superior to some folklore path planning methods for rigid robots in dynamic environments.

First Heuristic Approach

One straightforward approach that may be used by the continuum arm to avoid mobile obstacles is to stop (or slow down) and wait for the obstacles to clear the way. This heuristic approach imitates the static obstacle planning approach in Section 4.1 but with modifications to account for the changing state of the graph. Since the obstacles are moving, the cube path must be constantly recomputed. At

5.1. ANTICIPATORY PATH PLANNING ALGORITHM

each time step, we check to see if the next cube in the cube path is free of obstacles. If so, we compute the smoothest transition between the configurations in the current cube and the next cube using the method discussed in Subsection 5.1.2. If the next cube contains an obstacle, that cube must be purged from the cubes-graph and then a new cube path is reconstructed. This proceeds until the arm collides with an obstacle, indicating a failure, or until the arm tip reaches the goal point p_g .

Second Heuristic Approach

The second heuristic approach is also based on the static path planning discussed in Section 4.1. However, rather than recomputing a new path every time a cube is unavailable, the arm waits until the obstacle moves away. At each time step, we check to see if any obstacles are colliding with the next cube in the cube path. If not, the smoothest transition between the configurations in the current and next cube is calculated using the method discussed in Subsection 5.1.2. If the obstacle is in the next cube, however, we simply wait until the obstacle is no longer colliding with the cube. This continues until the arm either reaches the goal point p_g or collides with an obstacle.

Potential Field Approach

A *potential Field* [115] is a greedy method of path planning that uses attractive and repulsive forces to move the arm tip towards a goal point p_g . A potential-field path planning approach for continuum arms was given in [103]. A similar approach was developed in [104]. Our implementation closely follows that in [103] and we compare our results to those obtained in [103]. The approach in [103] is a greedy approach that is unreliable in the presence of obstacles. When multiple obstacles are in the $\mathcal{W}$ -space, the magnitude and spread of repelling forces create a repulsive “wall” that prevents the arm from moving towards the goal. Additionally, in a dynamic environment, changes in the position of obstacles (and repelling forces) could reverse progress that the arm has already made towards the goal point. These two issues, taken together, can cause the arm to move wildly back and forth.

5.1.7 Test Environment

In our implementation of the temporal approach, we used the built-in *interp1d* function provided by the *SciPy* library [116] to perform interpolation. This function allows us to input recent obstacle observations stored in the history buffer H , and retrieve predicted future obstacle positions over a limited number of time steps. In our implementation, predictions are generated from the past 5 observed positions of each obstacle.

In order to test the performance of each approach fairly, we randomly generated test cases with moving obstacles that followed certain criteria. Each approach was tested on these trials, and their relative performance was recorded.

The task for each algorithm is to output a sequence of configurations that brings the arm tip from a starting point p_s to a goal point p_g . If an obstacle collides with the arm at any point, that trial is considered a failure. Otherwise, the task is complete when the arm tip is within some small distance of the goal point p_g . For each number of obstacles $n \in \{1, 3, 5, 6\}$, we generate 100 test cases. For each n , we ran each approach on each of the 100 test cases and keep track of the average running time (in seconds) and the success rate over all 100 test cases. Obstacles are modeled as spheres with their radius randomly chosen as either 2 cm or 3 cm. These obstacles orbit around random points along the straight line joining p_s to p_g . The radius of these orbits is selected randomly in the range of 6 cm to 15 cm. The ranges of the x -axis, y -axis, and z -axis of the entire $\mathcal{W}$ -space are approximately $[-40.20\text{cm}, 40.20\text{cm}]$, $[-40.20\text{cm}, 40.20\text{cm}]$, and $[-29.95\text{cm}, 45\text{cm}]$, respectively. For purposes of simplicity, we treat time as

5.1. ANTICIPATORY PATH PLANNING ALGORITHM

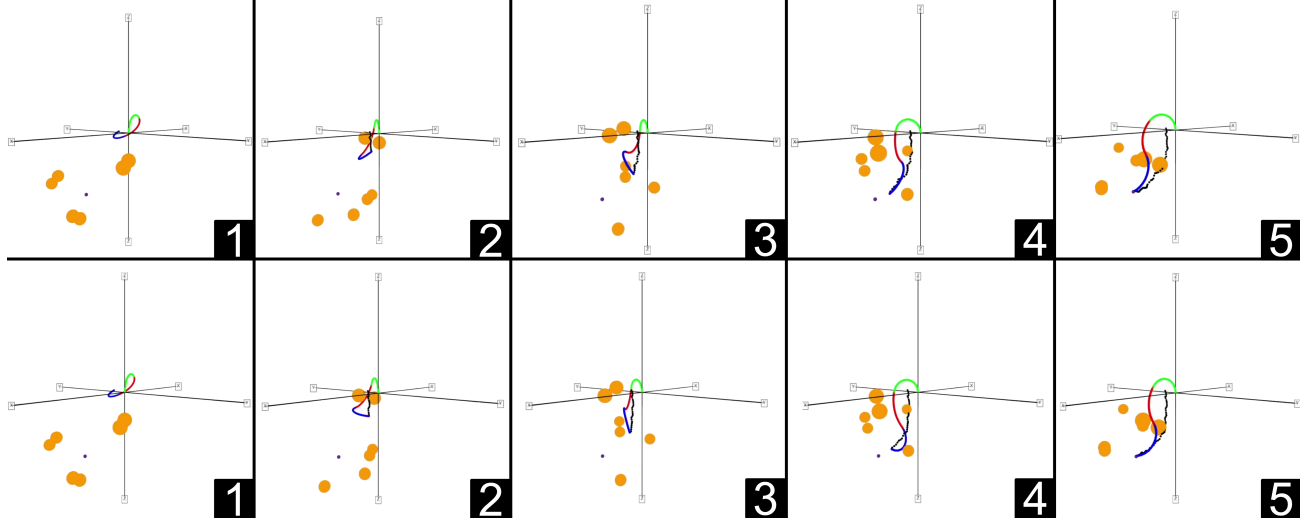


Figure 5.1: A comparison of trajectories generated by the T_2 (top) and T_3 (bottom) approaches for the same trial. The trajectory is shown at 5 steps of progress, with step 1 and step 5 showing the start and goal states, respectively. Objects are shown in orange and the goal tip position is shown in purple. Note that the H , H_s , and PF approaches could not generate a valid trajectory for this trial.

Table 5.1: Simulations of obstacles between start and end points

Algo. \ # Obs	1		3		5		6	
T_2 Time(s)/Succ	31.86	92%	39.37	69%	35.01	55%	32.58	46%
T_3 Time(s)/Succ	42.56	93%	50.61	72%	42.98	56%	40.19	46%
H Time(s)/Succ	10.41	58%	21.30	20%	30.31	14%	33.36	4%
H_s Time(s)/Succ	5.31	58%	12.23	21%	17.32	10%	18.57	4%
PF Time(s)/Succ	39.07	17%	41.22	17%	37.93	19%	70.27	19%

a discrete unit. Time is discretized into steps of 1 second each. In all of our simulations, we assume the obstacles move at a relatively low speed. Speed is a randomly generated number that is no greater than 1.5 cm/s. This is a reasonable assumption, as today’s high-speed cameras can capture moving objects at the frequency of multiple frames per second, which means the object moves only a small distance between frames [117].

5.1.8 Results

Table 5.1 shows the simulation results. We compare five different algorithms. T_2 and T_3 in the table refer to the temporal approaches that run on the next 2 and 3 predicted positions of obstacles while H and H_s refer to the first and second heuristic approaches, respectively. PF refers to the potential field approach. A comparison of the T_2 and T_3 approaches is shown in Figure 5.1.

As indicated from the simulation results, both temporal approaches perform significantly better than the heuristic approaches and potential field approach in all cases. Understandably, as the number of obstacles increased, the performance of all approaches decreased. However, the temporal approaches were consistently the best performing algorithms. As discussed earlier, the limitations of potential fields prevented the arm tip from reaching the target point in most cases. Though this was expected with high numbers of obstacles, the potential field approach also underperformed with few obstacles.

5.1. ANTICIPATORY PATH PLANNING ALGORITHM

The heuristic approaches were also understandably worse than the temporal approach as they do not consider the predicted motion of the obstacles. Although the larger prediction window should give T_3 an advantage over T_2 , in some of our simulation cases, we observed that T_2 performed better than T_3 . This is mainly caused by the fact that our prediction algorithm (cubic interpolation) does not predict the obstacle motions at the third time step as accurately as the first two.

Chapter 6

Rapidly-Exploring Random Tree Path Planning

This chapter discusses the application of RRT*, an advanced variant of RRT, in addressing the unique path planning challenges presented by continuum arms. Recognized for generating high-quality trajectories with low computational demand and notable reliability, sampling-based strategies like RRT are traditionally tailored for the state space of robotic systems. Nevertheless, the complexity escalates significantly for robots with a high degree of freedom, where the expansive, high-dimensional state space incurs the so-called *curse of dimensionality*, leading to prohibitive search times.

The intricate kinematics and high DoF inherent to continuum arms exacerbate these challenges, rendering standard RRT approaches less effective. To counteract this, the chapter introduces a novel RRT* methodology that combines workspace ($\mathcal{W}$ -space) search with a Jacobian-based configuration generation technique. When applied to scenarios involving static obstacles and trajectory tracing tasks, this $\mathcal{W}$ -space-oriented RRT* method consistently outperforms traditional techniques, generating superior trajectories for trajectory tracing.

The $\mathcal{W}$ -space RRT* method is further adapted to navigate environments populated with dynamic obstacles, a critical capability for continuum arms designed for highly dynamic settings. The dynamic RRT* variant presented herein demonstrates comparable reliability to the Anticipatory path planning method discussed in Chapter 5 for scenarios characterized by non-chaotic motion. Notably, it exhibits superior performance in environments with chaotic or unpredictable movements, demonstrating its robustness and adaptability.

6.1 $\mathcal{W}$ -space RRT*

RRT* is a widely-used algorithm for path planning that is often employed to explore the state space of a robot. A $\mathcal{C}$ -space based RRT* implementation for continuum arms faces several challenges, primarily due to their complex and highly non-linear kinematics. For instance, a linear trajectory in the $\mathcal{C}$ -space may correspond to a highly-undesirable, non-linear path in the $\mathcal{W}$ -space. Additionally, reorientation, when the arm makes large shifts in the pose without progressing along the path, can occur when two unrelated configurations occur sequentially in a path. This poses inherent hindrances for a $\mathcal{C}$ -space-based implementation of RRT* that is capable of achieving high-quality path planning. Therefore, for continuum arms, an RRT* implementation should preferably be based on $\mathcal{W}$ -space exploration, especially when obstacles are present in the environment.

Generating a mapping from the $\mathcal{W}$ -space to the $\mathcal{C}$ -space of continuum arms is a challenging task. For instance, a straightforward mapping of $\mathcal{W}$ -space points to $\mathcal{C}$ -space points by separate invocation of IK does not work, as separate invocations may produce uncoordinated spatial shapes; this issue can create an undesired movement for the continuum arm. There exist schemes for $\mathcal{W}$ -space to $\mathcal{C}$ -space mappings based on Jacobians that do not require the use of IK[7], [118]. Though this approach may also run into issues of convergence, it is possible to overcome these issues using randomness.

We overcome the above challenges with a RRT* algorithm for continuum arms that is based on $\mathcal{W}$ -space exploration. This algorithm uses a Jacobian-based method to iteratively estimate the IK solutions, along with considerations for hyper-redundancy, to make tree growth easier. Variance is added to the Jacobian-based method to avoid “getting stuck” in valleys of the $\mathcal{W}$ -space and $\mathcal{C}$ -space.

To start, a tree is grown outward from a starting configuration. Candidate nodes are created by sampling the $\mathcal{W}$ -space, steering, and then using the Jacobian-based method to find suitable corresponding configurations. On some iterations, the goal point p_g is considered in the formulation, instead of a randomly sampled point, to bias the tree growth towards the goal. Nodes that are found to collide with obstacles are discarded, and a queue of unsuccessful nodes is kept to avoid unnecessary exploration of unpromising tree branches. The procedure continues until the tree reaches the goal point p_g , and a corresponding configuration of p_g is computed. We now proceed to the details.

6.1.1 $\mathcal{W}$ -space - $\mathcal{C}$ -space Projection Via Jacobians

This method uses $\mathcal{C}$ -space and $\mathcal{W}$ -space displacements and velocities with their corresponding Jacobian to find a new configuration using a starting configuration and a desired change in tip position. Using forward kinematics, the continuum arm tip position, p , can be extracted from $T(q) = [R(q) \ p(q); 0 \ 1]$ where $q \in \mathcal{C}$ is a configuration and T is the HTM mapping the $\mathcal{C}$ -space to the $\mathcal{W}$ -space. The tip velocity can be described as $\dot{p} = J(q)\dot{q}$ where $\dot{q}$ is configuration velocity. Given a change in configuration δq , we can approximate a change in the position vector of the tip of the arm as $\delta p \approx J(q)\delta q$ where J is the Jacobian of p . Similarly, we can approximate a change in configuration δq as

$$\delta q \approx J_w(q)^\dagger \delta p, \quad (6.1)$$

where $J_w(q)$ is a weighted Jacobian [119] that ensures the validity of configurations generated using this method and $[\]^\dagger$ denotes the Moore-Penrose pseudo-inverse operator. As δp increases, so does the error in this approximation. Using (6.1), we can use a small change in tip position to calculate a new configuration $q' = q + \delta q$ as

$$q + \delta q \approx q + J_w(q)^\dagger \delta p \quad (6.2)$$

Thus, the new configuration vector will be adjacent to the current configuration q . To use this formulation, an initial configuration q with corresponding tip p must first be selected. Then, a target point $\hat{p}$ is also selected, and δp is calculated as $\delta p = \hat{p} - p$. These values are plugged into (6.2), and we find a new configuration $q' = q + \delta q$ that places the tip of the arm at $\hat{p}$. We refer to this process as the Jacobian configuration-generation method (JCG).

6.1.2 Null Space

In the context of the continuum arm, the null space can correspond to the redundancy of the arm, or rather to the configurations that produce the same result at the tip. We can exploit the null space to modify the position/spatial shape of the intermediate sections of the arm while still producing the desired change in tip position.

We can describe the null space of the aforementioned Jacobian as $I - J^\dagger J$, where I is the identity matrix.

With the consideration of the null space, variance can be added to (6.2) using random configuration velocities. We start with the initial configuration q . First, a random configuration q_k is picked from the $\mathcal{C}$ -space. Then, q_k is used to create a “random” configuration velocity. A configuration q'_k is found that is within some small distance from q in the direction of $q_k - q$. Next, a random $\mathcal{C}$ -space velocity μ is found by calculating $\mu = q'_k - q$. Lastly, the null space $I - J^\dagger J$ is multiplied by μ . We apply μ to (6.2) and obtain the following approximation

$$q + \delta q \approx q + J_w^\dagger \delta p + (I - J^\dagger J)\mu. \quad (6.3)$$

6.1.3 $\mathcal{W}$ -space Exploration

Exploration of the $\mathcal{W}$ -space only occurs using viable $\mathcal{C}$ -space points that correspond to the tip of the continuum arm. This ensures the tree contains only feasible nodes. The exploration is performed using a tree of nodes $\mathcal{T}$ where each node represents a valid $\mathcal{C}$ -space and $\mathcal{W}$ -space vector pair. Each node $n \in \mathcal{T}$ is associated with a configuration q_n and its corresponding tip position in $\mathcal{W}$ -space.

A tree is grown outward from some initial node n_s with corresponding configuration q_{n_s} . To perform exploration, we start by randomly selecting a point r in the $\mathcal{W}$ -space. A node n is found in the tree whose tip point has the shortest distance to r . We then use the steering process to find a point in the direction of r from n at distance δ ; a new node n' is inserted into $\mathcal{T}$ that corresponds to that point. Using the difference between n' and n and the configuration q_n of n , we apply the JCG method to find a configuration $q_{n'}$ corresponding to n' .

Given that the JCG method computes an approximate solution, it may be that the distance between n' and the $\mathcal{W}$ -space point corresponding to $q_{n'}$ is large. If this distance is larger than a certain threshold (specified based on the simulation results), we say that the invocation of the JCG method “failed” and deem the configuration $q_{n'}$ “invalid”. Given the aforementioned use of random null space vectors, it may be that the JCG method both succeeds and fails for the same input on different invocations. Assuming that the configuration is acceptable, we compute and update n' using $q_{n'}$. Additionally, the distance δ is selected to be small such that the error produced by the JCG method is also small.

During the rewiring process, a node m may be rewired so that a new node n' becomes the parent of m . Therein, we calculate a new configuration for m (using the JCG method) between n' and m . If the JCG method fails and the distance between the current and new tip position of m is large, m is not updated. We explore the $\mathcal{W}$ -space with the goal of finding the goal point p_g and inserting it into $\mathcal{T}$. When a node n' is inserted into $\mathcal{T}$, the Euclidean distance between p_g and n' is calculated. If this distance is within some small acceptable margin, n' is marked as the goal node n_g . There now exists a path from the initial node n_s to the goal node n_g in $\mathcal{T}$.

Cost

To maximize the quality of a generated path, movement at the tip must proceed smoothly while minimizing unnecessary movement in the intermediate sections of the arm. The cost $\zeta(m, n)$ between two nodes $m, n \in \mathcal{T}$ is defined as the Euclidean distance between their corresponding arm tip points. Note that this does not take into account movement in the intermediate sections of the arm.

Goal-Biasing

The growth of the tree can be directed towards the goal using *Goal-biasing*. In this scheme, random sampling occurs every $\omega - 1$ iterations of RRT*. However, in the ω^{th} iteration, the random node

is replaced with the goal point p_g . In some applications where quality paths are prioritized over exploration, randomness can be reduced by decreasing ω . We refer to the process of selecting the goal instead of a random node as *guiding*, and the ω^{th} iteration is called a *guiding iteration*. If a guiding iteration using the goal point p_g and a node n that is closest to p_g fails to produce a configuration (due to obstacle collision, local minima, etc.), then no additional node is added. In a future iteration, it may be possible that the same node n is attempted again if no other node is closer to p_g , meaning that if the tree has not grown in the direction of the goal, the algorithm could become stuck. Although repeated invocations of the JCG method can produce different results (due to the null space variance), it is possible that there is no feasible configuration, and n would be repeatedly attempted with no forward progress/growth. To avoid this scenario, a queue is created that has a size equaling some fixed percentage of $\mathcal{T}$'s size. Any node in this queue cannot be used during a guiding iteration. This encourages guided tree growth in other sections of the tree.

Starting Configurations

Since the JCG method relies on an initial configuration to produce a result, additional starting configurations or *auxiliary nodes* can be added to increase the diversity of the configurations included in the tree. This increased diversity can be useful in certain domains, like obstacle avoidance, where having a wide variety of spatial-shapes is important. These nodes are generally selected so that their corresponding tip points inhabit different areas of the $\mathcal{W}$ -space near the starting configuration.

Obstacles

Obstacles may or may not be present in the $\mathcal{W}$ -space during path planning. If obstacles are present, collisions can be checked before the node is inserted into the tree. Using the configuration corresponding to a node n' and the HTM, points along the intermediate sections of the arm can be calculated. Each of these points is checked for collision with obstacles. If such a collision exists, n' is not inserted into the tree. As a consequence of choosing the step size δ to be small (to minimize the error produced by the JCG method), the tree grows slowly with small gaps between nodes. With small changes in tip position, collision detection is performed at very small intervals.

6.1.4 The RRT* Algorithm

The algorithm (see Algorithm 6) starts with an initial configuration q_{n_s} , a goal tip position p_g , and the minimum number of iterations i . A tree is created, and the starting node n_s and auxiliary starting nodes are inserted (lines 1–2). Then in line 3, the expansion of the tree begins with goal-biasing occurring every ω iterations. During guided iterations (lines 4–11), p_g is selected as the goal point. The node n with the smallest Euclidean distance between n 's tip and p_g is selected (line 5). The JCG method is used with n to create a new node n' assuming this node is valid and does not collide with obstacles (lines 7–8). The rewire routine is invoked (line 9). During non-guided iterations (lines 12–20), a random $\mathcal{W}$ -space point r is sampled. Next, the node n with the smallest Euclidean distance between n 's tip and r is found (line 14). The tip of n is steered towards r , then the JCG method is invoked with n to create a new node n' , assuming the node is valid and does not collide with obstacles (lines 16–17). After a node is added to the tree, the rewire subroutine is invoked (line 18).

Algorithm 6: $\mathcal{W}$ -space RRT*

Input: Initial configuration q_{n_s} ; goal point p_g ; minimum number of iterations i
Output: A sequence of configurations as a path P

- 1 Create tree $\mathcal{T}$ using q_{n_s} and p_g ;
- 2 Create auxiliary starting-configurations and add to $\mathcal{T}$;
- 3 **while** no node in $\mathcal{T}$ has reached p_g and number of iterations $< i$ **do**
- 4 **if** guiding iteration **then**
- 5 Find a node n nearest to p_g ;
- 6 Create a new node n' by steering from n to p_g ;
- 7 **if** n' is valid and is obstacle-free **then**
- 8 Calculate a configuration $q_{n'}$ for n' using the JCG method;
- 9 Insert n' into $\mathcal{T}$ and rewire;
- 10 **end**
- 11 **end**
- 12 **else**
- 13 Generate a random $\mathcal{W}$ -space point r ;
- 14 Find a node n nearest to r ;
- 15 Create a new node n' by steering from n to r ;
- 16 **if** n' is valid and is obstacle-free **then**
- 17 Calculate a configuration $q_{n'}$ for n' using the JCG method;
- 18 Insert n' into $\mathcal{T}$ and rewire;
- 19 **end**
- 20 **end**
- 21 Check to see if p_g has been reached;
- 22 **end**
- 23 Generate a path P by following $\mathcal{T}$ from the goal node;
- 24 **return** P ;

6.1.5 $\mathcal{C}$ -space RRT*

As another basis for comparison with other relevant approaches, a $\mathcal{C}$ -space RRT* approach for the continuum arm model under consideration was implemented. The $\mathcal{C}$ -space implementation follows to a large extent the description of the $\mathcal{W}$ -space RRT* implementation (i.e., in terms of steering, goal-biasing, etc.), except that the tree growth occurs in the $\mathcal{C}$ -space (and hence, there is no need for the invocation of the JCG method). Each node $n \in \mathcal{T}$ is associated with a configuration q_n . Note that nodes in the tree $\mathcal{T}$ will still need to be mapped to the $\mathcal{W}$ -space to perform tasks such as obstacle detection. Once a path to a target node in $\mathcal{T}$ is found, no additional mapping is required as the corresponding sequence of configurations can simply be outputted.

Although $\mathcal{C}$ -space RRT/RRT* approaches are the standard for many robotic models, there are model-specific challenges that prevent a $\mathcal{C}$ -space RRT* approach for the model under consideration from generating a high-quality path as alluded to in Section 6.1. Since path planning requires specifying a target end-effector position, IK must be employed to generate the corresponding configuration. While this behaves well in free path planning, it becomes a serious issue in trajectory tracking. In this task, the arm tip must stop a series of intermediate points or *stops* to have the effect of “tracking”. To accomplish this task, multiple invocations of IK are needed to generate configurations corresponding

to each of the stops. These invocations are performed with random starting configurations, as chaining together IK invocations leads to the knotting problem alluded to earlier. This leads to an uncoordinated movement along the trajectory, resulting in poor-quality paths.

Some considerations are made to mitigate the above-mentioned issues in the implementation. First, the cost $\zeta(m, n)$ is updated so that, between two nodes $m, n \in \mathcal{T}$, it is defined as the Euclidean distance between their corresponding configurations q_m and q_n . Second, to address the issue of uncoordinated movement along the trajectory, multiple (as opposed to one) configurations corresponding to each intermediate trajectory point were generated, and the best configuration (w.r.t. the Euclidean distance from the previous point) is chosen.

6.1.6 Other Approaches

We compare our $\mathcal{W}$ -space RRT* to the Look-up Table and the IK-based approaches introduced in Chapter 4, as well as the $\mathcal{C}$ -space RRT* approach described in Section 6.1.5. All these approaches were designed specifically for the continuum arm model under consideration. We summarize the approaches in Chapter 4 briefly for the sake of comparison.

The approaches in Chapter 4 are graph-theoretic and rely on a decomposition of the $\mathcal{W}$ -space. The Look-up Table approach works by first computing a discretization of the $\mathcal{C}$ -space, calculating the corresponding tip points, and then storing the configurations and points in a look-up table. Next, the $\mathcal{W}$ -space is discretized into a grid of cubes, and a path is found in the grid. Then, configurations are mapped to cubes if the corresponding tip position falls inside the cube. Lastly, a layered graph is constructed where each layer consists of all of the configurations in one of the cubes in the path of cubes. Section 4.2 discussed two IK-based approaches that had very similar results; we only discuss/compare one of them for the sake of brevity and refer to it as the Refined-IK approach. The Refined-IK approach is similar to the Look-up Table approach (Section 4.1), but trades the look-up table for real-time invocation of IK. A cube is “populated” with configurations using repeated invocations of IK. Refined-IK specifically uses the configurations from one cube as a starting point for additional invocations of IK. Both approaches suffer from runtime issues. The Look-up Table approach requires the generation of an offline look-up table which, depending on the density, can take more than 30 minutes to compute. The Refined-IK approach, on the other hand, requires repeated calls to IK, which requires substantial computation time.

6.1.7 Comparison of Approaches

Each of the algorithms is evaluated using two categories of tests. They will be evaluated by both their computation time (the amount of time required to compute a solution) and DynSim-time (the output of the dynamic simulator discussed in Section 2.1.4). In some of these tests, the arm will start in a “resting” position. The *resting configuration* or *resting position* corresponds to the configuration $(0, 0, 0, 0, 0, 0)$ where the arm has a neutral shape and is not bent or rotated. The corresponding arm tip position is $(0, 0, 0.45)$ m. Note that the feasible $\mathcal{W}$ -space of our robotic model is in the range of $[-0.30, 0.30]$, $[-0.30, 0.30]$, $[-0.10, 0.45]$ m for the x , y , and z coordinates, respectively.

The first category of tests evaluates obstacle avoidance. In these scenarios, the respective algorithms will be tasked with finding a path from the initial resting configuration to a point that is selected randomly for each scenario and is within the feasible $\mathcal{W}$ -space. In addition, spherical obstacles will be placed randomly in the $\mathcal{W}$ -space. We report the mean computation time needed by each algorithm to produce a solution to each of the trials. Additionally, the mean DynSim-time and its variance σ^2 are reported.

Table 6.1: Summary of Computation Time for Obstacle Avoidance

(For each approach, the mean DynSim-time, the corresponding variance σ^2 , and the mean computation time for 100 random trials in seconds.)

Approach	DynSim-Time [s]	σ^2	Computation-time [s]
C-RRT*	40.98	441.53	607.8
W-RRT*	13.11	43.62	50.61

The second category of tests is for trajectory tracking. A collection of challenging shapes are selected and then represented as a trajectory in the $\mathcal{W}$ -space. Each algorithm was used to produce a path that traces the prescribed trajectory. The computation time needed to produce a solution will be recorded alongside the DynSim-time. Additionally, given the randomness inherent to RRT*, the two RRT*-based approaches will be run 100 times on each trajectory. The mean computation time and DynSim-time will be reported alongside the variance σ^2 of the DynSim-time. Throughout the tables of results, we refer to the Look-up Table approach, Refined-IK approach, $\mathcal{C}$ -space RRT* approach, and $\mathcal{W}$ -space RRT* approach as **LT**, **REF-IK**, **C-RRT***, and **W-RRT***, respectively.

The algorithms discussed above were implemented and tested on a 64-bit Windows 10 PC using an Intel i7-9700K 8 core 3.60GHz CPU with 32 GB of RAM. The Look-up Table and Refined-IK approaches are implemented following the description in Chapter 4.

6.1.8 Category 1: Obstacle Avoidance

A group of 100 random trials was performed in which the algorithms were tasked to produce a path from a random starting configuration (chosen from within the feasible $\mathcal{C}$ -space) to a random target point (from within the feasible $\mathcal{W}$ -space). Obstacles were represented as spheres. Six spheres of random radius [5, 12] were placed randomly in the $\mathcal{W}$ -space. Note that for the RRT*-based approaches, the amount of random sampling dictated by ω is set at 10.

Table 6.1 shows the results from these trials. Note that both approaches successfully produced a solution to each scenario. The mean DynSim-time, the corresponding variance, and the mean computation times are shown. Note that the $\mathcal{W}$ -space RRT* approach has much lower DynSim-times, indicating that these paths are of higher quality. Additionally, the computation time is lower for these cases. Though the $\mathcal{C}$ -space RRT* approach is capable of finding a solution, the undesired ancillary movement introduced by the $\mathcal{C}$ -space exploration causes the quality to be much lower. Additionally, since this ancillary movement can cause the path to be longer, the $\mathcal{C}$ -space RRT* trajectories take a longer amount of computation time to produce.

An example is given in Figure 6.1 of the $\mathcal{W}$ -space and $\mathcal{C}$ -space RRT* approaches performing the obstacle avoidance task. The path produced by the $\mathcal{C}$ -space approach in this example is nearly linear when plotted with respect to time. However, this path, when mapped back to the $\mathcal{W}$ -space, corresponds to undesirable behavior that introduces a lot of unnecessary movement. In fact, many $\mathcal{C}$ -space RRT* simulation results exhibited far worse behavior than the example reported above.

6.1.9 Category 2: Trajectory Tracking

We select a collection of “demonstrative” shapes that are challenging for the arm to trace to demonstrate trajectory tracking. These are circles, figure-8 shapes (the shape resembling the infinity symbol), and cubes that have been rotated. Versions that are both large and small are generated. For example, the “small circle” is located at $z = 0.15$ m and has a radius of 0.1 m, whereas the “large circle” is located

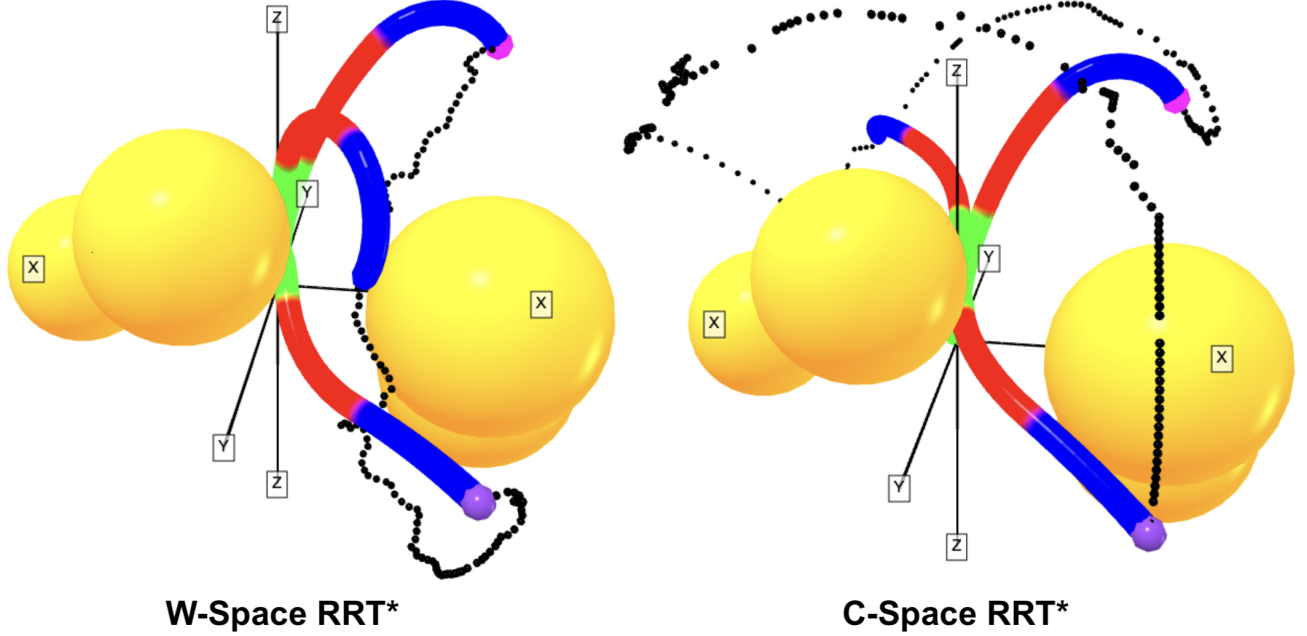


Figure 6.1: A comparison of a trajectory produced by the $\mathcal{W}$ -space RRT* (left) and $\mathcal{C}$ -space RRT* (right) for the obstacle avoidance task. The target point is shown in purple.

Table 6.2: Summary of DynSim-Time Results for small shapes (above line) and large shapes (below line). (All numbers represent the DynSim-time in seconds. Variance is included for the RRT* approaches.)

Approach	LT	REF-IK	C-RRT* σ^2		W-RRT* σ^2	
Circle	13.8	11.0	39.60	1607	9.9	1.1
Figure-8	20.8	17.0	97.40	276.6	16.1	7.1
Rot Cube	110.1	86.5	516.35	4354.97	71.5	106.5
Circle	Failure	57.71	191.70	1339	51.9	5.3
Figure-8	81.61	53.71	503.5	$7.1e^3$	51.0	2.6
Rot Cube	245.34	116.78	941.5	$1.8e^4$	118.3	295

at $z = 0.25$ m and has a radius of 0.345 m. These shapes are passed as trajectories to each of the algorithms. Note that for the RRT*-based approaches, the amount of random sampling dictated by ω is set at 3.

Table 6.2 shows the DynSim-times for each of the approaches on all of the shapes. Table 6.3 shows the computation times for each of the methods for each of the shapes. The DynSim-times and computation times for the two RRT*-based approaches are averages after simulating each scenario 100 times. The average DynSim-time is reported alongside the variance σ^2 . The “Failure” in the table indicates that the Look-up Table approach was not able to complete the task, as some of the cubes did not have any configurations, and therefore a path could not be produced. Figure 6.2 shows a comparison between the approaches tracking a rotated cube.

The Look-up Table, Refined-IK, and $\mathcal{W}$ -space RRT* approaches have similar quality, though our $\mathcal{W}$ -space RRT* approach has the best overall. However, the $\mathcal{W}$ -space RRT* approach substantially improves computation time over the other two methods. The numbers highlighted in the table for the small circle show that $\mathcal{W}$ -space RRT* is a 95% decrease in computation time over the next best

6.2. DYNAMIC $\mathcal{W}$ -SPACE RRT*

Table 6.3: Summary of Computation Time Results for small shapes (above line) and large shapes (below line). (All numbers represent computation time in seconds.)

Approach	LT	REF-IK	C-RRT*	W-RRT*
Circle	549.02	22450	27.31	14.26
Figure-8	255	22809	21.3	19.8
Rot Cube	2414	103914	126.6	98.3
Circle	N/A	79286	171.95	180.6
Figure-8	218.47	52966	66.3	180.8
Rot Cube	923.03	83927	316.6	191.75

result while also having a 26% reduction in DynSim-time relative to Look-up Table. Note also the inferior performance of the $\mathcal{C}$ -space RRT* approach. As discussed, the reorientation required from this approach creates low-quality paths, which is reflected in the DynSim-times. Additionally, the computation time is surprisingly high for $\mathcal{C}$ -space RRT*. This is due to the numerous invocations of IK discussed in Section 6.1.5. Though the $\mathcal{C}$ -space RRT* approach does have a better computation time than the $\mathcal{W}$ -space RRT* approach for the large circle, the difference in DynSim-time between the two approaches is immense, with the $\mathcal{W}$ -space RRT* approach producing a much higher quality path.

6.2 Dynamic $\mathcal{W}$ -space RRT*

In this section, we discuss a $\mathcal{W}$ -space RRT* planner for dynamic environments that we refer to as $WRRT^*D$. The planner grows a sequence of trees in the $\mathcal{W}$ -space, and traces a partial path in each of these trees, beginning from the original starting point p_s and finally arriving at the goal point p_g . During the tree growth, the radius of each obstacle is inflated, so that planning takes place with an implicit buffer around each obstacle. A tree is grown until its growth reaches a certain distance, at which point a partial path in this tree is generated. Once a path is generated, the arm begins to trace this path. While the arm is tracing a path, due to potential collision with obstacles, there may be a need to generate a replacement path. Specifically, if the arm is moving too close to an obstacle or away from the goal, the path needs to be reconstructed. In such a case, a new tree is grown from the arm’s current position. This sequence of growing trees, finding paths, and tracing paths is repeated until the goal is reached or the algorithm fails to find a collision-free path.

6.2.1 The Planner

In dynamic path planning, generally, no information is provided to the planner about the future positions of obstacles. Therefore, the planner must construct a path using the currently observed obstacle state, and then update that path as new obstacle information is acquired during execution. Although path planners for static environments often attempt to compute a global path from the original starting point to the destination point, this is generally not suitable for dynamic planning.

To address these challenges, our planner works in two phases. In the first phase, a partial path is created for the arm to follow. In the second phase, the arm attempts to trace this partial path. During tracing, the currently observed obstacle state may change in a way that makes the present path unsuitable. In that case, a new partial path is created for the arm to follow starting from the arm’s current position.

To avoid unnecessary computation, we use the *horizon planning* approach [120], [121]. Rather than attempting to grow a global path from the original starting point toward the target point, horizon

planning grows a random tree outward from the current source until the depth of nodes reaches a certain threshold. Once this threshold is reached, a path is generated from the tree, whether or not the target has been reached.

A path P is defined as a sequence of configurations corresponding to nodes in $\mathcal{T}$. During execution, this path is traced step-by-step in discrete time, with replanning performed as needed using the currently observed obstacle set.

Despite the use of horizon planning, partial paths are still costly to generate, so it is necessary to limit the frequency of replanning. To achieve this during the path tracing phase, we employ a number of strategies to detect the need for a replan and to limit the frequency of replans. Phase one is shown in Algorithm 7 and phase two is shown in Algorithm 8.

Horizon Planning

Let $h > 0$ be a fixed constant, referred to as the *horizon distance*. To grow a tree $\mathcal{T}$ in horizon planning starting from some node $u \in \mathcal{T}$ with configuration q_u , we proceed as described in Section 6.1. At each iteration, the distance between the tip positions of u and the newly-inserted node n is calculated. There are two termination cases during horizon planning:

- i. Node n reaches the goal point p_g . In this case, growth has been completed and a partial path from u to p_g is generated.
- ii. The distance exceeds the horizon distance h . In this case, node n marks a “crossing of the horizon” and the growth of the partial path is complete. We generate the partial path from u to n .

Tracing

In a dynamic setting, planners only have knowledge of the current and previous positions of obstacles. In phase two of the planner, the planner traces the generated tree path. While the arm is tracing the path, the position of obstacles is updated to reflect the new state of the $\mathcal{W}$ -space.

A *replan* must occur when the current partial path is not suitable for the arm given the current positions of the obstacles. If a replan is initiated, a new path is created using horizon planning and originating from the arm’s current position. This means that a new tree $\mathcal{T}$ with the arm’s current position as the source is grown.

6.2.2 Obstacle Inflation

To avoid collision with obstacles, the continuum arm needs to maintain a “buffer distance” from each obstacle present in the environment. To maintain the buffer distance during planning, the radius of each obstacle is slightly increased during the horizon planning.

6.2.3 Distance Record

At tracing step t , let

$$A(t) = [a_1(t), a_2(t), \dots, a_m(t)]$$

denote a sequence of sampled points along the arm, and let

$$\mathcal{O}(t) = \{o_1(t), o_2(t), \dots, o_k(t)\}$$

denote the currently observed obstacle set in the $\mathcal{W}$ -space. We define the *distance record* at tracing step t to be

$$D(t) = [d_1(t), d_2(t), \dots, d_m(t)],$$

where, for each sampled arm point $a_j(t)$, the entry $d_j(t)$ is the minimum distance from $a_j(t)$ to the obstacles in $\mathcal{O}(t)$.

As the pose of the arm changes during tracing, and as the obstacle positions are updated, the entries of $D(t)$ are recomputed at each tracing step. Let

$$D(t+1) = [d_1(t+1), d_2(t+1), \dots, d_m(t+1)]$$

denote the updated distance record at the next tracing step. If, for some sampled arm point a_j , the new distance entry $d_j(t+1)$ is less than the minimum of its previously recorded distance entry $d_j(t)$ and the critical distance d_{crit} , then a replan occurs. Equivalently, a replan is triggered if there exists an index j such that

$$d_j(t+1) < \min\{d_j(t), d_{\text{crit}}\}.$$

Thus, the distance record provides a reactive mechanism for monitoring whether any portion of the arm is moving too close to the currently observed obstacles as tracing proceeds.

Algorithm 7: Horizon

Input: starting configuration q_{n_s} , goal point p_g
Output: A path P

- 1 create a tree $\mathcal{T}$ using q_{n_s} and p_g ;
- 2 **while** no node has crossed the horizon **do**
- 3 grow $\mathcal{T}$ by inserting a node n ;
- 4 **if** the tip point of n is within an acceptable distance of p_g **then**
- 5 calculate the path P from q_{n_s} to n ;
- 6 **else if** $\text{dist}(n, n_s) \geq h$ **then**
- 7 calculate the path P from q_{n_s} to n ;
- 8 **end**
- 9 **end**
- 10 **return** P ;

6.2.4 Backup Counter

The backup counter is an integer variable, f , that records the number of tracing steps during which the arm moves away from the goal point p_g . Let β be the number of permissible backup steps before a replan is necessary. As the arm traces the path, the distance between the arm's tip point and the goal point p_g is calculated at each tracing step. If this distance increases from one step to the next, then f is incremented by 1. If at any moment $f > \beta$, then a replan is initiated.

6.2.5 Test Environment

In order to evaluate the performance of the dynamic path planner, we compare it to the anticipatory approaches shown in Chapter 5. To summarize, these approaches use previous obstacle locations, a prediction scheme, and a temporal graph to generate paths that “anticipate” the motion of obstacles. We subject each of the tested approaches to three test environments. Each environment uses spherical

Algorithm 8: $\mathcal{W}$ -space RRT* Dynamic

Input: global starting configuration q_{ns} , global goal point p_g , moving obstacles
Output: Global path P_G

```

1 global path  $P_G = \emptyset$ ;
2 initialize tracing step  $t = 0$ ;
3  $P = \text{HORIZON}$ ;
4 current node  $u = P.\text{pop}()$ ;
5 add  $q_u$  to  $P_G$ ;
6 while the tip of the arm has not reached  $p_g$  do
7   observe the current obstacle set  $\mathcal{O}(t)$ ;
8   update the distance record  $D(t)$  and the backup counter;
9   if a replan is required then
10    reset any necessary replanning counters;
11     $P' = \text{HORIZON}$ ;
12    replace the current partial path with  $P'$ ;
13  end
14   $u = P.\text{pop}()$ ;
15  append  $q_u$  to  $P_G$ ;
16  increment tracing step  $t$ ;
17 end
18 return  $P_G$ ;

```

obstacles with a radius of either 2 or 3 cm, with each obstacle assigned a radius randomly. When the pose of the arm is changed during tracing, the change in the obstacle position is roughly 75% of the change of the arm tip. The obstacle inflation radius was selected by experimentation. Let r be the initial radius and r' be the radius after inflation. This change in r was selected such that $r' \leq 3r/2$. Each of the trials begins at the *resting pose* of the arm. This is the pose where none of the PMAs are actuated and corresponds to the configuration (and joint values) $[0, 0, 0, 0, 0, 0]$. The arm is tasked with navigating from the resting pose to a randomly-generated goal point p_g without colliding with any obstacles. Additionally, in all trials, a random goal point p_g is selected such that the point is within 25 cm of the origin. A summary of each of the test environments follows.

Environment 1 - Circular Orbit Motion: In this environment, the obstacles are following a circular orbit around a fixed center. These orbit centers are selected along the straight-line between the initial tip position and the target position. Each orbit lies in a plane that makes an angle $-45^\circ \leq \alpha \leq +45^\circ$ with the xy -plane, where α is chosen at random with uniform probability. Additionally, the direction of the rotation (clockwise, counter-clockwise) is selected randomly with uniform probability.

Environment 2 - Wave Motion: This environment uses sinusoidal waves in order to generate obstacle movement in the $\mathcal{W}$ -space. Obstacles are placed in a line near the target point which forms a 90° angle with the straight-line from the starting point to the target point. Obstacles are placed such that their respective centers are 4 cm apart. The obstacles move with sinusoidal motion that lies on a plane and all obstacles intersect at the midpoint between the starting tip point and the target point. This is, in effect, a “wall” of obstacles advancing towards the arm that moves in a sinusoidal pattern.

Environment 3 - Random Motion: The random motion in this environment consists of random walks. Each of the obstacles is initially placed along the straight-line between the resting pose arm tip and the target point. For each obstacle, a random distance between 1 and 5 cm is selected. Next,

6.2. DYNAMIC W -SPACE RRT*

Table 6.4: Orbit Environment

Obstacles \ Approach	1	2	3	4	5	6
AN2 Succ.	78%	76%	72%	71%	64%	66%
AN2 Time (s)	11.7	15.5	12.4	15.4	17.6	2.4
AN3 Succ.	78%	76%	71%	73%	64%	66%
AN3 Time (s)	11.7	9.6	19.4	23.2	25.3	29.8
WRRT*D Succ.	67%	72%	75%	68%	68%	65%
WRRT*D Time	0.2	20.7	0.7	1.9	0.8	1.6

Table 6.5: Wave Environment

Obstacles \ Approach	1	2	3	4	5	6
AN2 Succ.	53%	31%	26%	25%	23%	12%
AN2 Time	5.6	6.2	9.9	11.2	13.5	12.4
AN3 Succ.	65%	42%	45%	28%	26%	19%
AN3 Time	11.2	13.7	17.0	17.7	20.5	22.6
WRRT*D Succ.	75%	74%	69%	68%	73%	64%
WRRT*D Time	0.3	0.4	0.4	0.5	0.7	0.5

a random direction is selected. The obstacle then travels a random distance in a random direction. When the obstacle has traveled the specified distance, a new distance, and a new direction are selected, and the obstacle moves according to those new parameters. Each of these environments is tested on a range of obstacle counts, with 1, 2, $\dots$, 6 obstacles used. For each number of obstacles, the approaches are tested on the same 100 random trials. The success rate over the 100 trials is reported along with the average computation time that each approach took to generate a solution.

6.2.6 Results

The results of each of the approaches for the test environments are reported in the tables below. We refer to the Temporal-2, Temporal-3, and WRRT*D approaches as $AN2$, $AN3$, $WRRT^*D$ respectively.

Table 6.6: Random Environment

Obstacles \ Approach	1	2	3	4	5	6
AN2 Succ.	48%	25%	17%	3%	4%	1%
AN2 Time	5.1	6.8	7.9	8.1	11.1	11.8
AN3 Succ.	39%	18%	10%	4%	2%	2%
AN3 Time (s)	9.5	10.8	12.9	14.3	16.3	18.8
WRRT*D Succ.	71%	76%	60%	64%	67%	63%
WRRT*D Time	0.25	0.26	0.26	0.257	0.257	0.257

The success rate reported in each table is the percentage of trials that were completed successfully. The columns correspond to the number of obstacles used in each set of trials. The rows contain the success rate and computation time of each of the approaches. Figure 6.3 shows a path generated by WRRT*D for environment 3.

Tables 6.4, 6.5, and 6.6 show the results from the Circular Orbit Motion environment, Wave Motion environment, and Random Motion environment, respectively. In general, it seems that the AN2 and AN3 approaches are very sensitive to the quality of predictions, whereas the WRRT*D approach is more consistent across motions. Random motion, in particular, is challenging for the anticipatory approaches. The low success rate of the WRRT*D approach in certain settings is likely unrelated to the environment, and instead related to the method of calculating configurations.

6.2.7 Continuum Arm Prototype Testing

The 3-section prototype arm shown in Figure 2.1 is tested using the configurations generated in path planning simulations. Configurations are related to length changes of the arm using kinematic relationships [122]. Length changes are mapped to actuation pressure trajectories using the joint space – pressure mapping approach discussed in [123] and fed to the arm via an experimental setup that consists of an 8 *bar* air compressor, 09 proportional pressure regulators (ITV3050-31F3N3, SMC USA) corresponding to 09 PMAs of the arm, and a MATLAB Simulink desktop real-time model interfaced with a data acquisition (DAQ) card (PCI-6703, NI USA). The DAQ card is coupled with pressure regulators and issues voltage signals corresponding to the pressure commands set in MATLAB Simulink to release air pressure.

To replicate the motion of dynamic obstacles, we attached plastic spherical objects to the end effector of a rigid robotic arm (AUBO-i5, AUBO-cobot USA). The obstacle trajectories that were generated relative to the base of the continuum arm are transformed relative to the base of the rigid arm. This helps generate obstacle trajectories by actuating the rigid arm. Figure 6.4 shows a path generated by the WRRT*D for one and two obstacles following circular orbit motion. The arm prototype testing initially validates the path planning algorithm. Due to the operational limitations of the arm, not every path planning algorithm in this chapter was tested. Here we present proof of concept through preliminary experimental results.

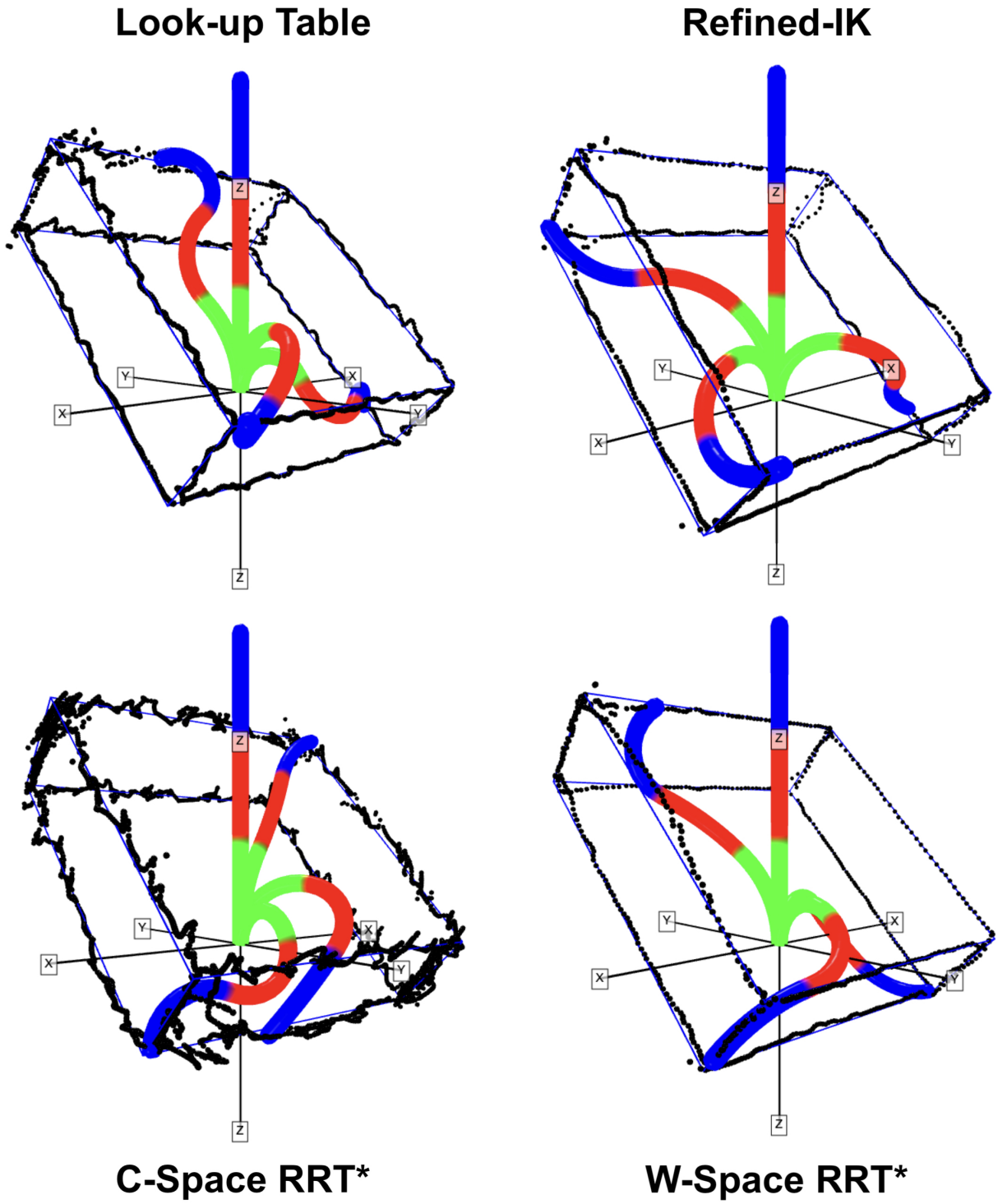


Figure 6.2: A comparison of a trajectory traced by the Look-up Table (top left), the Refined-IK (top right), $\mathcal{C}$ -space RRT* (bottom left), and $\mathcal{W}$ -space RRT* (bottom right).

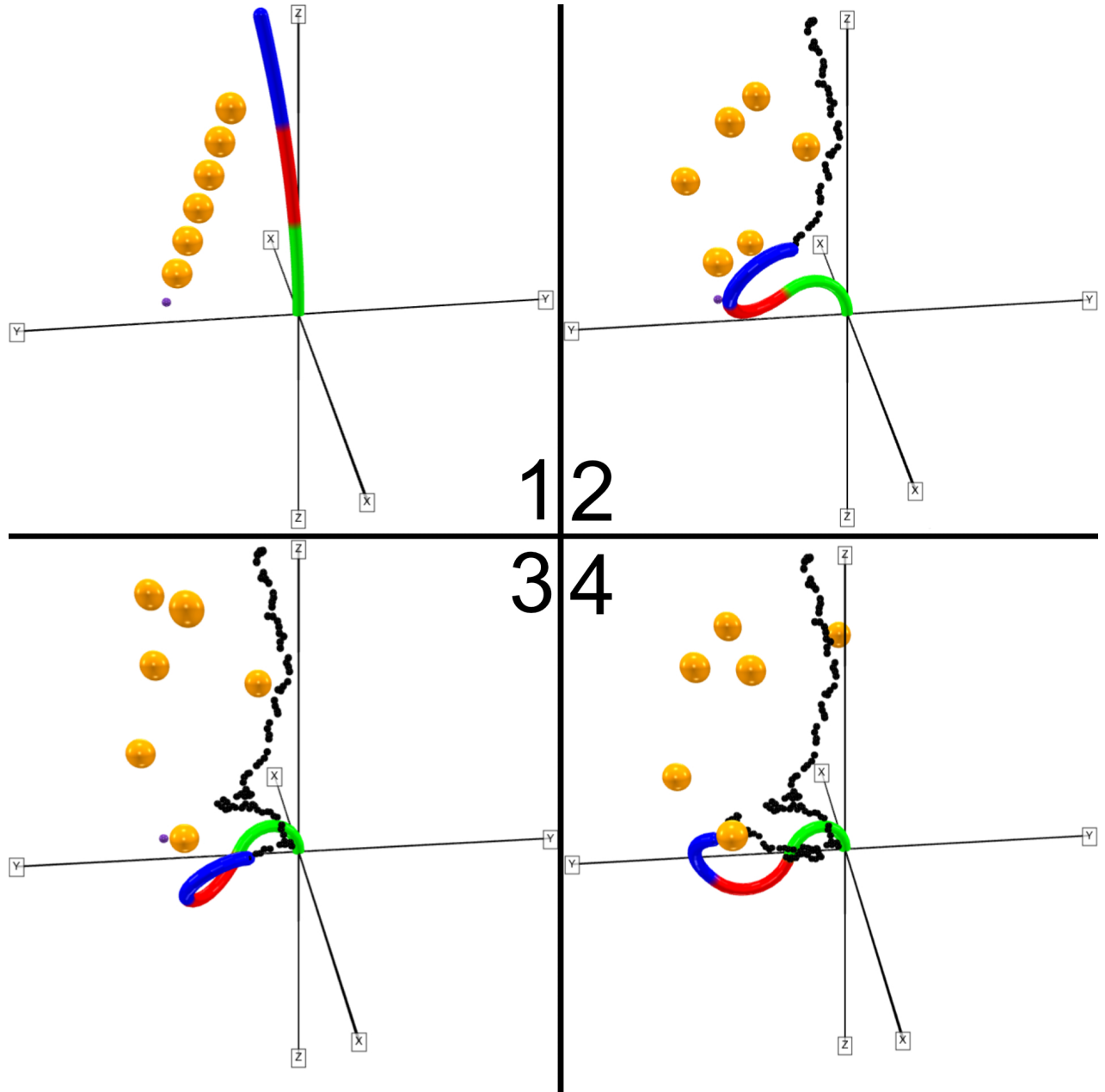


Figure 6.3: A path generated by WRRT*D performed in Environment 3. Frames are shown of the initial position (1), 33% progress (2), 66% progress (3), and the final position (4)

6.2. DYNAMIC W -SPACE RRT*

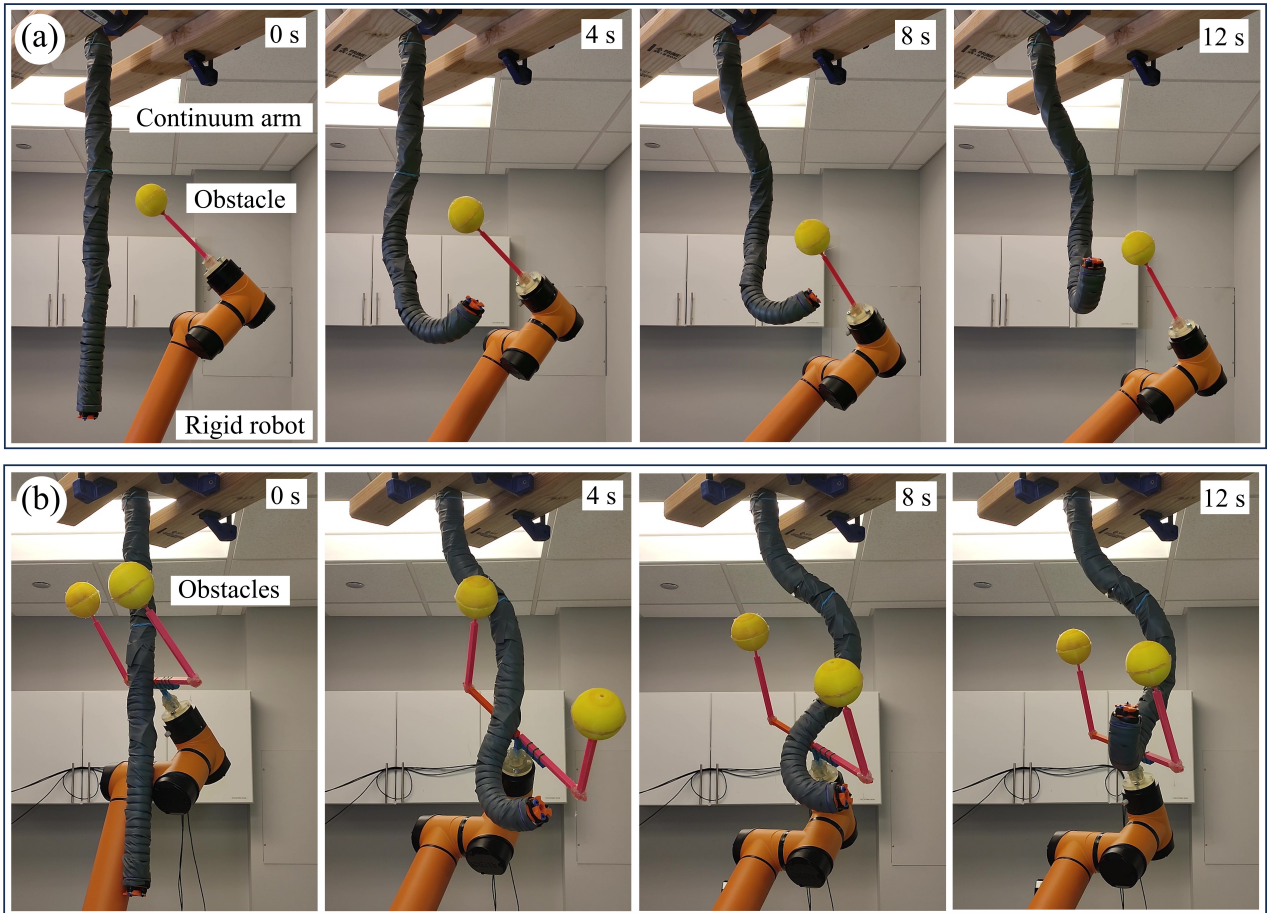


Figure 6.4: Path planning progression of the continuum arm prototype under (a) single and (b) two obstacles.

Chapter 7

Probabilistic Roadmap Path Planning

In this chapter we discuss PRM in the $\mathcal{J}$ -space of the continuum model in question to address the dynamic path planning problem. The PRM and RRT approaches are similar as they allow efficient computation of paths using random sampling. RRT is typically faster than PRM, as its iterative nature limits exploration to only those relevant areas. Though this reduces computation, RRT is less able to broadly explore and often requires constructing a tree from scratch for each query. In contrast, PRM constructs a global roadmap that reflects the connectivity and can be reused across multiple queries. This multi-query property is useful in dynamic settings, where planning is performed from changing starting configurations. In addition, PRM naturally supports multi-goal planning, where a path may terminate at any one of several target configurations, since all targets can be incorporated into the same roadmap without reconstructing the search structure for each goal. By utilizing a precomputed or incrementally constructed roadmap, PRM enables reuse of prior computation while maintaining coverage. Given the broad nature of PRM, it may be less suited for path planning through narrow corridors, though paths that require large changes in state benefit from PRM's broader exploration.

$\mathcal{J}$ -space planning is also advantaged for PRM. Linear trajectories in the $\mathcal{C}$ -space can produce non-smooth and physically unrealistic motions, which are ill-suited for PRM. On the other hand, linear $\mathcal{J}$ -space trajectories more naturally reflect the physical changes taken by the arm during transitions and limit unnecessary posture changes. Consequently, $\mathcal{J}$ -space exploration is preferable to $\mathcal{C}$ -space, especially when making connections to grow a roadmap.

Taken together, these characteristics make PRM well-suited to incorporate temporal graphs for addressing the dynamic path planning problem. Though PRM is less effective for navigating narrow gaps than RRT, the changing environment reduces the need to search these regions and instead emphasizes far-reaching connectivity. Additionally, given the semi-permanent and graph-like nature of the roadmap, temporal graphs can be naturally constructed on top of the roadmap to augment it with time. This chapter presents a temporal $\mathcal{J}$ -space approach that leverages these properties to create a robust planner that demonstrates improved performance over the other presented planners.

7.1 Temporal $\mathcal{J}$ -space PRM

7.1.1 Temporal Local Planner

We discuss a temporal local planner that augments a static $\mathcal{J}$ -space roadmap into a temporal graph over a finite time window and searches for a feasible time-respecting path. Intuitively, the planner iterates forward in time and, at each timestep, instantiates temporal edges corresponding to roadmap motions that remain collision-free over their traversal duration. As temporal edges are created, a

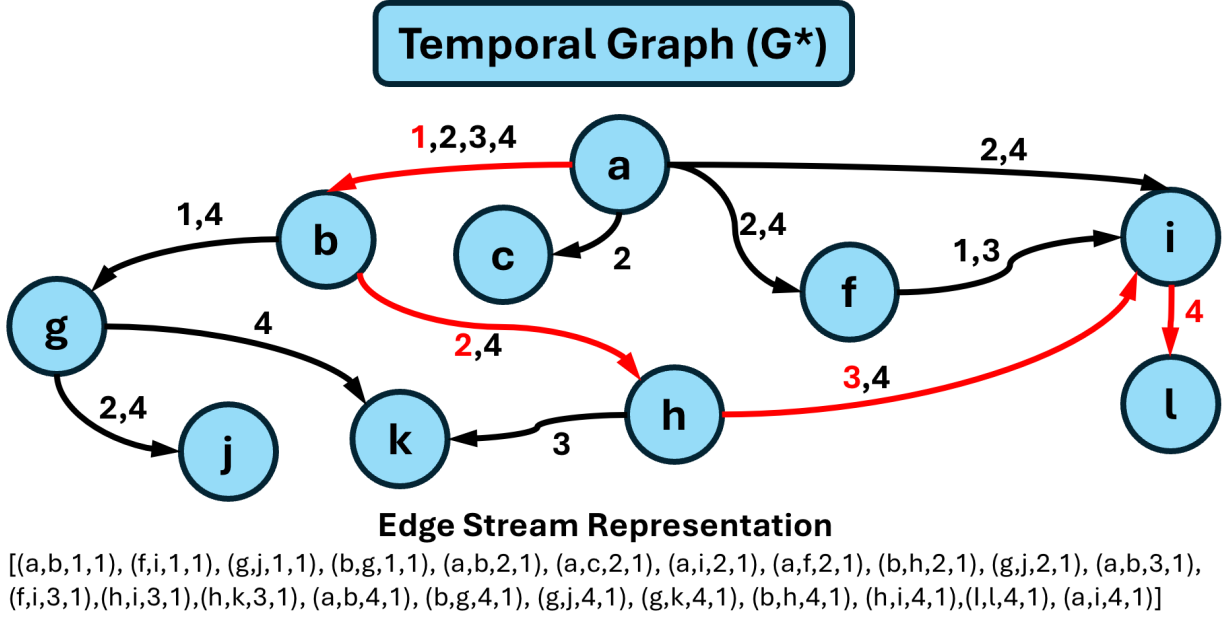


Figure 7.1: An illustration of a temporal graph with all $\lambda = 1$. The edges and times corresponding to an earliest arrival path from vertex (a) to vertex (l) are highlighted in red. The equivalent edge stream representation is also shown.

relaxation scheme propagates earliest arrival times through the graph, allowing the planner to detect if the goal can be reached within the window. Following [17], we use an earliest-arrival formulation, but employ it strictly as a feasibility check rather than for optimality guarantees.

7.1.2 Temporal Graphs for Local Planning

As described in Section 2.4.5, we represent the dynamic planning problem using a temporal graph whose edges encode time-dependent availability with respect to the obstacle sets $\mathcal{O}(t)$. An example of a temporal graph with an edge stream representation and an earliest arrival path is shown in Figure 7.1. Here, we focus on how that temporal graph is constructed and searched in the context of a $\mathcal{J}$ -space roadmap for a continuum arm.

In this work, the temporal graph is instantiated from a $\mathcal{J}$ -space roadmap $G = (V, E)$ representing the motion of a continuum arm. Each vertex $v \in V$ corresponds to a $\mathcal{J}$ -space configuration q , and each edge $(u, v) \in E$ represents a collision-free motion between configurations under static constraints. We augment this static roadmap with time-dependent feasibility over a finite planning window, thereby producing temporal edges only when the corresponding motion remains feasible throughout its traversal.

Let $\gamma(\sigma; u, v)$, with $\sigma \in [0, 1]$, denote the $\mathcal{J}$ -space interpolation between configurations $q(u)$ and $q(v)$ along a roadmap edge $(u, v) \in E$. Let $\mathcal{B}(q) \subset \mathbb{R}^3$ denote the occupied robot body at configuration q , and let $\mathcal{O}(t) \subseteq \mathbb{R}^3$ denote the true obstacle set at time t . Then a temporal edge (u, v, w, λ) is included if and only if the motion from u to v , starting at time w and lasting λ time units, remains collision-free for the entire traversal:

$$\forall \tau \in [w, w + \lambda] : \mathcal{B}(\gamma(\frac{\tau-w}{\lambda}; u, v)) \cap \mathcal{O}(\tau) = \emptyset. \quad (7.1)$$

In this context, we use *configuration*, *state*, *node*, and *vertex* to refer to the same underlying arm instance at different levels of representation: a configuration denotes joint values, a state refers to such a configuration, and a node or vertex indexes that state within the roadmap or temporal graph.

Algorithm 9: Temporal Local Planner

Input: Graph $G = (V, E)$; source v_s ; target v_g ; window $[T_s, T_e]$; edge time $\lambda(u, v)$
Output: A temporal path P

- 1 **Initialize;**
- 2 For each vertex v , set $m_a[v] \leftarrow \infty$ and $m_p[v] \leftarrow \perp$;
- 3 Set $m_a[v_s] \leftarrow T_s$ and $m_p[v_s] \leftarrow v_s$;
- 4 Set found $\leftarrow$ false;
- 5 **Timestep iteration;**
- 6 **for** $w = T_s$ **to** T_e **do**
- 7 **if** found = false **then**
- 8 **foreach** edge $(u, v) \in E$ **do**
- 9 **if** $m_a[u] \leq w$ **then**
- 10 Set $d \leftarrow \lambda(u, v)$ and $w' \leftarrow w + d$;
- 11 **if** (u, v, w, d) is feasible **then**
- 12 **if** $w' \leq T_e$ and $w' < m_a[v]$ **then**
- 13 Set $m_a[v] \leftarrow w'$ and $m_p[v] \leftarrow u$;
- 14 **if** $v = v_g$ **then**
- 15 Set found $\leftarrow$ true;
- 16 **end**
- 17 **end**
- 18 **end**
- 19 **end**
- 20 **end**
- 21 **end**
- 22 **end**
- 23 **if** found = false **then**
- 24 **return** FAILURE;
- 25 **end**
- 26 **Reconstruct path;**
- 27 Set $P \leftarrow []$ and $x \leftarrow v_g$;
- 28 **while** $x \neq v_s$ **do**
- 29 Set $u \leftarrow m_p[x]$;
- 30 Set $d \leftarrow \lambda(u, x)$;
- 31 Set $w' \leftarrow m_a[x]$ and $w \leftarrow w' - d$;
- 32 Prepend $(u \rightarrow x, w, d)$ to P ;
- 33 Set $x \leftarrow u$;
- 34 **end**
- 35 **return** P ;

Temporal Graph Construction

Our temporal local planner receives as input a graph G , a source vertex v_s , a target vertex v_g , and a finite time window $[T_s, T_e]$. The graph is assumed to contain at least one static path from v_s to v_g . As discussed earlier in Section 2.4.5, a temporal path must respect both connectivity and edge timing. In the dynamic planning setting considered here, temporal edge feasibility is evaluated with respect to the obstacle state at each timestep.

The approach begins by initializing an edge stream F , together with two maps m_p and m_a . The map m_p associates each vertex with a predecessor, while m_a stores the earliest time at which each vertex may be reached. These structures are initialized with empty values, except for the source vertex corresponding to the initial configuration, which is assigned an arrival time of T_s and made its own predecessor. Taken together, F , m_p , and m_a represent the evolving temporal graph during planning.

Starting from timestep $w = T_s$, the algorithm iteratively considers the static edges in E at each subsequent timestep and instantiates temporal edges when they are feasible over their full traversal duration. For a given edge (u, v) , if vertex u is reachable by time w , the planner checks whether traversing that edge from time w to time $w + \lambda(u, v)$ satisfies the temporal feasibility condition in (7.1). If so, the algorithm attempts to update the arrival time and predecessor of v . In this way, earliest-arrival information is propagated forward through the evolving temporal graph.

Whenever the target vertex v_g is reached during this process, construction terminates early and a temporal path is recovered by backtracking through the predecessor map. The resulting path is therefore time-respecting by construction, since each predecessor update records a feasible traversal together with its associated arrival time.

Our Approach

Since the earliest-arrival formulation [17] places no constraints on the order in which temporal edges with the same start time are processed, temporal edge construction and relaxation can be performed simultaneously. Each time a temporal edge is created from an edge in G , feasibility is assessed; if valid, predecessor information in m_p and arrival times in m_a are updated. Whenever the goal vertex v_g is encountered and its arrival time improves, a feasible temporal path has been found and the algorithm terminates early. The path is reconstructed by following predecessor relationships in reverse order. A sketch of this planner is shown in Algorithm 9.

7.1.3 Dynamic $\mathcal{J}$ -space Planning

We present a $\mathcal{J}$ -space PRM-based planner for dynamic environments. This approach is a 4-phase plan-replan method combining static PRM planning, temporal-graph validation, and roadmap regrowth. Our planner operates in four phases that alternate between planning and validation. First, batched IK is used to generate a set of valid goal configurations at the goal tip position p_g . Second, a static PRM roadmap is constructed under the assumption that obstacles remain fixed at their current locations. Third, a temporal local planner searches a predicted temporal graph derived from this roadmap to obtain a candidate temporal path. Finally, this path is traced using the true obstacle motion, and replanning is triggered whenever a predicted edge becomes invalid. A successful plan is returned when a goal is reached during tracing. Algorithm 10 summarizes the overall procedure.

Phase 1: Batch Goal Inverse Kinematics

The algorithm begins by generating valid configurations whose tips reach the goal point. A *valid configuration* is one in which the corresponding arm does not collide with itself and does not collide with an obstacle. Any valid configuration within ϵ of the goal tip position p_g is a *valid goal* and can be inserted into the roadmap as a goal state(s).

To accomplish IK solutions, we solve a nonlinear least-squares problem of the form

$$\min_{\mathbf{q} \in \mathbb{R}^6} \left\| \mathbf{p}_{\text{tip}}(\mathbf{q}) - p_g \right\|^2, \quad (7.2)$$

7.1. TEMPORAL $\mathcal{J}$ -SPACE PRM

where $\mathbf{q}$ is a configuration, and $\mathbf{p}_{\text{tip}}(\mathbf{q}) = \text{HTM}(\mathbf{q})_{\text{tip}}$ denotes the forward kinematics of the arm tip. A configuration is accepted as a valid goal if

$$\|\mathbf{p}_{\text{tip}}(\mathbf{q}) - p_g\| \leq \varepsilon.$$

We also enforce the joint constraints given by

$$\frac{2\sqrt{q_{i,1}^2 - (q_{i,1} q_{i,2}) + q_{i,2}^2}}{0.02165} \leq \pi, \quad i = 1, 2, 3 \quad (7.3)$$

to ensure that the robot's joint constraints are met. After generating these configurations, we re-check validity and insert them into the roadmap as goals. We refer to the set of valid goal configurations as $\mathcal{Q}_g$.

Phase 2: PRM-based Static Planning

In phase 2, a roadmap is grown or expanded from the current arm configuration to any of the valid goal states. This plan has no time information in its calculation and is treated like a static planning problem. On first entry, we initialize a PRM roadmap from q_s toward $\mathcal{Q}_g$; later, we grow/expand the existing roadmap from the current configuration q_c . Conventional PRM implementations often rely on a grow roadmap and expand roadmap phase. In the grow phase, random sampling allows the roadmap to explore new areas of the space. In the expand phase, the planner attempts to reach disconnected vertices using tactics like random bounces. In our planner, we introduce a new phase called *sample start* wherein the area around the starting vertex is specifically sampled. This may be thought of as a small version of the grow step, focusing on one region.

Configurations are added to the roadmap if they are valid (Section 7.1.3), meaning that they obey constraints and are collision free. During roadmap construction, we assume obstacles are frozen at the current planning time. That is, if the current configuration q_c is considered at planning time T_i , then we assume all obstacles remain fixed at their positions in the true obstacle set $\mathcal{O}(T_i)$ for the duration of the static planning phase.

Once roadmap growth or expansion occurs, the A* search algorithm is run to find a valid path from the current configuration q_c toward any goal in $\mathcal{Q}_g$. We denote the static free space at the current planning time T_i as

$$\mathcal{J}_{\text{free}}(T_i) = \{q \in \mathcal{J} : \mathcal{B}(q) \cap \mathcal{O}(T_i) = \emptyset\}. \quad (7.4)$$

A roadmap $R = (V, E)$ is constructed with $V \subset \mathcal{J}_{\text{free}}(T_i)$. Two vertices $u, v \in V$ are considered for connection if $v \in \mathcal{N}_k(u)$ (the k nearest neighbors of u in V under a $\mathcal{J}$ -space metric). An edge $(u, v) \in E$ is added only if the interpolated motion is collision-free under frozen obstacles:

$$\forall \sigma \in [0, 1] : \mathcal{B}(\gamma(\sigma; u, v)) \cap \mathcal{O}(T_i) = \emptyset, \quad (7.5)$$

where $\gamma(\sigma; u, v)$ denotes the $\mathcal{J}$ -space interpolation from $q(u)$ to $q(v)$.

Phase 3: Predicted Temporal Local Planning

This phase uses the algorithm described in Section 7.1.1 as a subroutine for finding a temporal path in our roadmap. However, two modifications are needed in the present setting. First, since the planner does not know future obstacle locations exactly, these future obstacle states must be predicted. Second, the planner must accommodate multiple possible goal configurations arising from the batched IK stage.

7.1. TEMPORAL $\mathcal{J}$ -SPACE PRM

In practice, the future true obstacle set $\mathcal{O}(t)$ is not known. We therefore replace it with a predicted obstacle set $\hat{\mathcal{O}}(t)$ constructed from recent observations. Accordingly, the temporal feasibility condition in (7.1) is evaluated using the substituted condition

$$\forall \tau \in [w, w + \lambda] : \mathcal{B}\left(\gamma\left(\frac{\tau-w}{\lambda}; u, v\right)\right) \cap \hat{\mathcal{O}}(\tau) = \emptyset. \quad (7.6)$$

Let $o_k(i) \in \mathbb{R}^3$ denote the observed center of obstacle k at timestep i . Using the two most recent observations, we estimate a constant velocity

$$\hat{v}_k(i) = o_k(i) - o_k(i-1). \quad (7.7)$$

Anticipated future positions are extrapolated for a prediction horizon H via

$$\hat{o}_k(i + \ell) = o_k(i) + \ell \hat{v}_k(i), \quad \ell = 1, \dots, H. \quad (7.8)$$

For $\ell > H$, the obstacle position is held fixed at $\hat{o}_k(i + H)$. A rolling window of the last W observations is maintained per obstacle, though only the two most recent entries are used for velocity estimation. The predicted obstacle set $\hat{\mathcal{O}}(t)$ is constructed from these predicted centers and the corresponding obstacle radii.

Phase 4: Path Tracing and Replanning

After finding a temporal path in the previous steps, our algorithm must now “trace” the temporal path to make sure that it is valid. Since our temporal graph is built on predictions, it may be that a planned path will be invalid due to a bad prediction or the end of the prediction window. To handle this case, our algorithm begins following each edge in the temporal path to ensure that it is valid with the actual obstacle locations. Whenever the temporal path becomes invalid, a replan must occur. During execution, each planned temporal edge (u, v, w, λ) is checked against the true obstacle motion. We declare the edge execution-feasible if

$$\forall \tau \in [w, w + \lambda] : \mathcal{B}\left(\gamma\left(\frac{\tau-w}{\lambda}; u, v\right)\right) \cap \mathcal{O}(\tau) = \emptyset. \quad (7.9)$$

If (7.9) is violated for the next edge, replanning is triggered from the current configuration q_c at the current timestep. After recording the steps already taken, the planner inserts the current position into the static roadmap as the new source and returns to phase 2. If phase 2 and phase 3 are repeated a large number of times, we forgo finding a motion and assume the arm remains motionless for one timestep. This allows the planner to record a new obstacle location and try planning with better predictions. This is repeated until (1) an allotted time is reached, (2) a goal is reached, (3) the arm collides with an obstacle. If the goal is reached during tracing, the recorded path is output as the result.

Algorithm 10: Temporal $\mathcal{J}$ -space PRM (TJPRM)

Input: Start configuration q_s ; goal tip position p_g ; true obstacle sets $\mathcal{O}(t)$; time window $[T_s, T_e]$; calculation time limit T

Output: A successful path P , or FAILURE

- 1 **Phase 1: Bulk Goal IK;**
- 2 Generate candidate set $\mathcal{Q}_c$ by numerical IK;
- 3 Keep configurations with valid joints and no collisions;
- 4 Store remaining configurations as goal set $\mathcal{Q}_g$;
- 5 **Phase 2: Static PRM;**
- 6 **if** *roadmap R is missing* **then**
- 7 Initialize roadmap from q_s ;
- 8 **end**
- 9 **else**
- 10 Sample start, grow roadmap, and expand roadmap;
- 11 **end**
- 12 Sample valid configurations and connect local edges;
- 13 Assume static $\mathcal{O}(t)$ and perform edge checks;
- 14 Run A* to obtain static path toward $\mathcal{Q}_g$;
- 15 **Phase 3: Temporal Local Planning;**
- 16 Build temporal graph $\mathcal{G}^*$ from R and $\hat{\mathcal{O}}(t)$;
- 17 Apply Temporal Planner (Section 7.1.1) to obtain path P^* ;
- 18 **if** *no path is found* **then**
- 19 Return to Phase 2;
- 20 **end**
- 21 **Phase 4: Path Tracing/Replan;**
- 22 Follow P^* using true obstacle motion $\mathcal{O}(t)$;
- 23 **if** *an edge becomes invalid* **then**
- 24 Record progress and return to Phase 2 using the current configuration as the start;
- 25 **end**
- 26 **if** *Phases 2 and 3 are repeated many times* **then**
- 27 Record a motionless timestep;
- 28 **end**
- 29 Continue until the goal is reached, T is exceeded, or a collision is detected;
- 30 **return** successful path P or FAILURE;

7.1.4 Test Environment

To evaluate the performance of the presented dynamic path planner, comparisons are made against the dynamic approaches discussed in Chapters 5 and 6, referred to as ANT and WRT, respectively. ANT employs an offline lookup table, workspace discretization, temporal graphs, and obstacle motion prediction to perform planning. WRT is a workspace RRT* approach that utilizes Jacobian-based configuration generation and trajectory adjustment mechanisms. The parameters for these reference methods are adopted directly from their respective publications.

Each tested approach is evaluated within three distinct environments. Each trial is initiated from the *resting pose* of the arm, corresponding to the configuration $[0, 0, 0, 0, 0, 0]$. From this pose, the arm is required to navigate to a randomly generated goal point p_g while avoiding collisions with all

obstacles. In every trial, the goal point p_g is uniformly selected within a 25 cm radius of the origin. During planning, a maximum allowable joint velocity of 2 cm/s is used to ensure valid transitions between states [3]. For all tests, the obstacle radii are selected randomly to be 2 or 3 cm. Each approach is presented the same randomized trials in each environment. A summary of the three testing environments is provided below.

Environment 1 – Circular Orbit Motion: In this environment, obstacles move along circular orbits with fixed centers. The orbit centers lie along the straight line connecting the arm’s initial tip position and the goal point p_g . Each orbit is contained within a plane tilted by a random angle α relative to the XY -plane, where $-45^\circ \leq \alpha \leq 45^\circ$ and α is drawn from a uniform distribution. The direction of rotation, clockwise or counterclockwise, is chosen uniformly at random.

Environment 2 – Hurricane Motion: This environment consists of two concentric cylindrical layers of obstacles orbiting the arm’s starting position. The cylinders are located at radial distances of 10 cm and 20 cm from the origin in the xy -plane, with obstacle heights uniformly distributed over the range [0,40] cm along the z -axis. The direction of motion, clockwise or counterclockwise, is selected uniformly at random. The goal point p_g is sampled outside the outer cylinder, at heights in the range [0,20] cm, and within the arm’s reachable workspace. Each trial uses the same number of obstacles in each layer, ranging from 1 to 32 per layer, for total obstacle counts ranging from 2 to 64.

Environment 3 – Random Motion: In this environment, obstacles follow random-walk trajectories. Each obstacle is initially positioned along the line segment connecting the arm’s resting tip position and the goal point p_g . A random travel direction and displacement magnitude between 5 cm and 10 cm are selected. After the obstacle completes this displacement, new random values for direction and distance are sampled, and the process repeats.

Each environment is evaluated under varying numbers of obstacles. For each configuration, 100 randomly generated trials are conducted under identical conditions for all tested approaches. The success rate of each approach to generate a feasible trajectory is reported.

7.1.5 Results

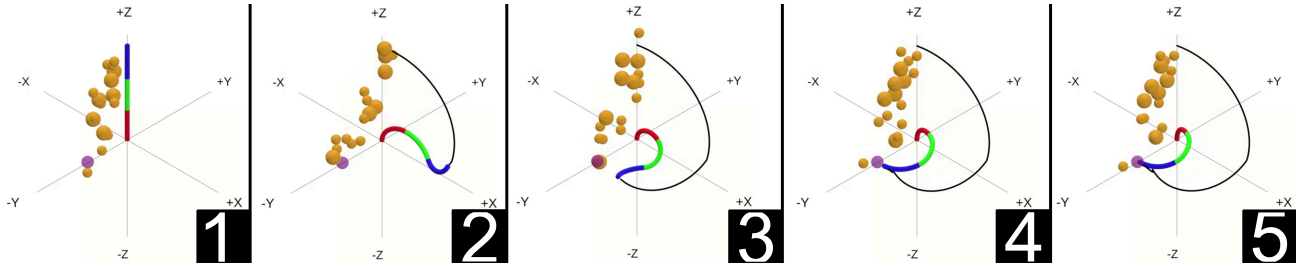


Figure 7.2: An example of the TJPRM approach creating a plan through the Orbit Environment (1). The trajectory is shown at 5 steps of progress, with step 1 and step 5 showing the start and goal states, respectively. In the accompanying video, this is Orbit Trial 2.

7.1. TEMPORAL $\mathcal{J}$ -SPACE PRM

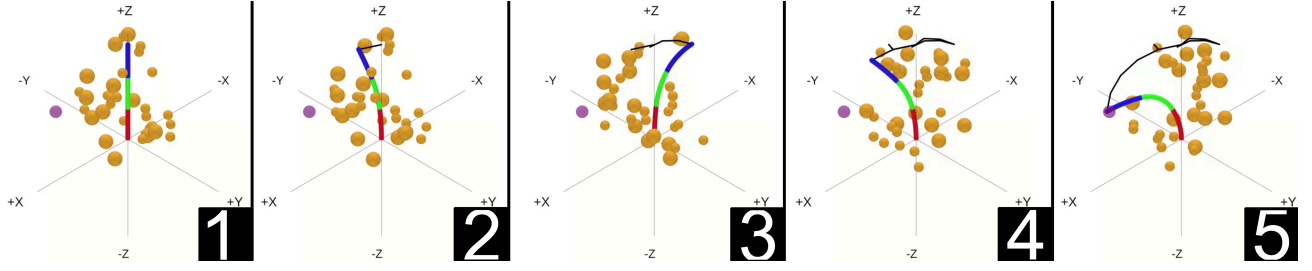


Figure 7.3: An example of the TJPRM approach creating a plan through the Hurricane Environment (2). The trajectory is shown at 5 steps of progress, with step 1 and step 5 showing the start and goal states, respectively. In the accompanying video, this is Hurricane Trial 2.

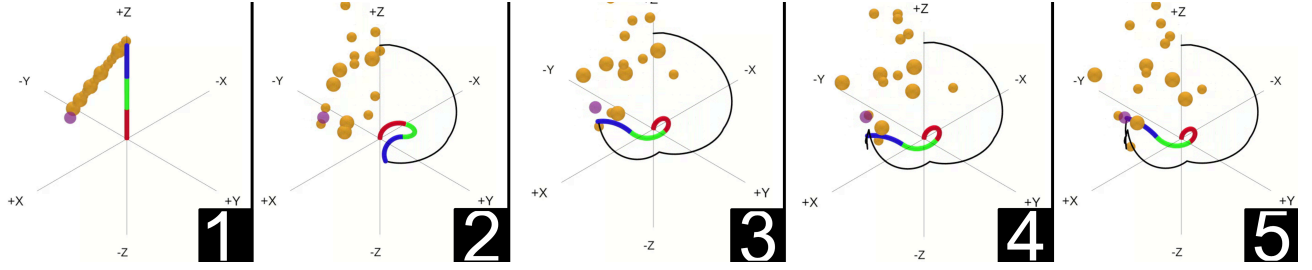


Figure 7.4: An example of the TJPRM approach creating a plan through the Random Environment (3). The trajectory is shown at 5 steps of progress, with step 1 and step 5 showing the start and goal states, respectively. In the accompanying video, this is Random Trial 3.

Table 7.1: Orbit Environment

Obstacles Approach	1	2	4	6	8	16
ANT Succ.	92%	82%	69%	50%	51%	22%
WRT Succ.	55%	36%	27%	22%	18%	5%
TJPRM Succ.	96%	91%	90%	80%	79%	43%

Table 7.2: Hurricane Environment

Total Obstacles Approach	2	4	8	16	32	64
ANT Succ.	79%	68%	46%	23%	1%	0%
WRT Succ.	43%	35%	26%	14%	3%	0%
TJPRM Succ.	99%	93%	74%	35%	7%	1%

Success is reported as the percentage of complete viable trajectories that were created for each trial. Figures 7.2, 7.3, and 7.4 show a sample trajectory in the orbit, hurricane, and random environments, respectively. Additional examples are provided in the accompanying video. Tables 7.1–7.3 summarize results across the three environments.

Overall, the TJPRM approach achieves the highest success rate across the three environments and at every tested obstacle count. In the orbit tests, the planners show the highest parity, though TJPRM still degrades more gradually than the other methods as the number of obstacles increases. For example, in the orbit environment at 16 obstacles, TJPRM retains a 43% success rate, compared with 22% for

7.1. TEMPORAL $\mathcal{J}$ -SPACE PRM

Table 7.3: Random Environment

Obstacles Approach	1	2	4	6	8	16
ANT Succ.	70%	42%	25%	21%	8%	0%
WRT Succ.	61%	50%	40%	23%	24%	8%
TJPRM Succ.	99%	99%	96%	90%	88%	79%

ANT and 5% for WRT. In the hurricane and random environments, the gap between TJPRM and the compared methods becomes substantially larger. Notably, the WRT and ANT planners place greater emphasis on the smoothness of the tip trajectory, which creates quality trajectories with respect to the tip, but limits their viability in more complicated environments.

The random motion and hurricane environments specifically highlight the TJPRM planner’s ability to take advantage of the arm’s high dexterity to navigate complicated environments. In the random environment, TJPRM remains above 88% success through 8 obstacles, whereas ANT falls to single-digit success by 8 obstacles and WRT is at 24% at that same obstacle count. At 16 obstacles in the random environment, TJPRM improves by 71 percentage points over the next-best planner. Similarly, in the hurricane environment, TJPRM maintains the highest success rate at every tested obstacle count, including 74% at 8 obstacles compared with 46% for ANT and 26% for WRT. The TJPRM planner achieves this robust performance by making changes to its trajectory before collision, and weaving through narrow passages between obstacles. Though the WRT approach also shares this behavior, the use of the temporal graph in the TJPRM approach reduces the amount of “wandering” found in the WRT trajectories.

The increased number of configurations available to the TJPRM approach allows it to achieve a wider variety of poses, which means that the tip may not follow a direct path to the goal point p_g , allowing for more complicated motions. Additionally, the TJPRM planner’s lack of tip constraints allows the planner to create trajectories that bypass obstacles entirely.

Chapter 8

Sampling-Based Planning for Dual Continuum Arms

Dual-arm planning is significantly more difficult than single-arm planning. The challenges already present in single-arm continuum planning are compounded by the need to coordinate two manipulators subject to shared constraints. In this chapter, we consider sampling-based planning for the dual-arm continuum model introduced in Section 2.2.3. The system consists of two identical three-section continuum arms whose tips are constrained to remain a fixed distance apart, thereby introducing a closed-chain coupling between the two manipulators. For convenience, this fixed-length coupling is described using the notation L_{rod} . As a result, feasible motion must satisfy the fixed-length and joint constraints while also avoiding self-collision, inter-arm collision, and obstacle collision.

This setting is particularly challenging for sampling-based planning. Algorithms such as RRT* and PRM rely on frequent and inexpensive random samples to populate trees and roadmaps, yet valid states in the dual-arm setting are difficult to obtain and often fragile once found. A random state is unlikely to be feasible, and even a feasible state may become invalid under a very small perturbation in the state space. Thus, planning in this domain requires both a means of generating valid configurations efficiently and a means of preserving validity under small local changes.

To address these issues, we introduce the notion of a *refiner*. A refiner iteratively adjusts a configuration through small, targeted corrections so that an invalid or nearly valid state is pushed toward the feasible set rather than discarded. Consequently, refinement serves two purposes: it provides a local repair mechanism for perturbed states, and it enables valid sampling from random initial states. These refiners are integrated into both RRT* and PRM, allowing the underlying planners to remain largely intact while supplying the additional machinery needed for dual-arm planning.

Unlike other planners in this dissertation, which primarily focus on a single state space representation, the methods in this chapter explore both the $\mathcal{C}$ -space and the $\mathcal{J}$ -space, as well as hybrid formulations that use each representation where it is most effective.

The chapter is organized as follows. We first introduce the general refinement framework and the specific refiner variants. We next present the dual-arm RRT* and PRM formulations, including the IK strategies and planner modifications needed for this setting. Finally, we present the experimental methodology and results for refinement, IK, obstacle avoidance, and shape tracing.

8.1 State Refiners

In the dual-arm setting, valid states are difficult to obtain and may become invalid under small perturbations. To address this issue, we introduce *state refiners*, which iteratively adjust a state so that it moves toward validity. In this way, the refiners support both local repair and practical valid state generation for the planners discussed later in the chapter.

8.1.1 General Refiner Framework

A refiner resolves constraint violations through sequences of small, targeted Jacobian-based corrections. At each iteration, the current state is examined for active violations, and the corresponding correction is applied. When a sampled point on one or both arms violates a constraint, a small adjustment is made so that the state moves toward validity. Depending on the violation, this may involve moving the two tips closer together or farther apart, or moving intermediate points along one or both arms so that they no longer collide with each other or themselves.

These adjustments are described in terms of small displacements δp applied to sampled body points, which are then mapped back into the state space using Jacobian-based methods. By applying these corrections incrementally, a nearly feasible state may often be driven toward validity without large destabilizing changes.

All refiners implement the same operations:

- $\text{Refine}(q, K)$: iteratively update a dual-arm state $q \in Q$, until it becomes feasible or until K iterations have been exhausted;
- $\text{SampleValid}(M)$: repeatedly sample a random state $q \in Q$ from the bounds of the chosen representation and apply Refine , up to M attempts.

These operations are built on top of Jacobian formulations, meaning either the $\mathcal{J}$ -space or $\mathcal{C}$ -space representation may be used, provided a Jacobian can be calculated.

8.1.2 Section Jacobian and Local IK

Each refiner relies on a local Jacobian

$$J(q_a, i, \xi) \in \mathbb{R}^{3 \times 6}, \quad (8.1)$$

where q_a denotes the state of a single arm $a \in \{1, 2\}$. This Jacobian maps a small change in the 6-dimensional state of one arm to the $\mathcal{W}$ -space velocity of the body point located on section i at arc-length fraction $\xi \in [0, 1]$. Writing the position of that point as $p(q_a, i, \xi)$, we have

$$\delta p \approx J(q_a, i, \xi) \delta q_a, \quad (8.2)$$

where δp is a desired displacement of the sampled body point and δq_a is the corresponding state space displacement for arm a . Thus, given a desired target point $\hat{p}$ for the same body point, a single correction is obtained through

$$\delta q_a \approx J(q_a, i, \xi)^\dagger (\hat{p} - p(q_a, i, \xi)), \quad (8.3)$$

and the updated state is

$$q'_a = q_a + \delta q_a. \quad (8.4)$$

When i and ξ correspond to the tip, $p(q_a, i, \xi)$ reduces to the tip position $p(q_a)$ of the single arm, and $J(q_a, i, \xi)$ reduces to the corresponding tip Jacobian $J_{\text{tip}}(q_a)$. A similar strategy is discussed in Section 6.1.1.

Algorithm 11: General Dual-Arm Refinement Loop

Input: State q ; maximum iterations K
Output: Refined state q or failure

```

1 for iter = 1 to  $K$  do
2   if  $q$  is feasible then
3     | return  $q$ ;
4   end
5   if resolve inter-arm collisions then
6     | continue;
7   end
8   if resolve self-collisions on arm 1 then
9     | continue;
10  end
11  if resolve self-collisions on arm 2 then
12    | continue;
13  end
14  if resolve tip-separation violation then
15    | continue;
16  end
17 end
18 if  $q$  is feasible then
19   | return  $q$ ;
20 end
21 return failure;

```

8.1.3 Refinement Loop

Inter-arm and self-collision checks are performed on discretized arm points. When a constraint violation involves both arms, the corresponding correction is applied symmetrically. In particular, when an inter-arm collision is detected, the first colliding pair of sampled points is identified and a unit collision direction is computed between them. Equal-magnitude displacements are then assigned in opposite directions, so that the two arms each absorb half of the total correction. Likewise, when the tip-separation constraint is violated, the correction is applied directly at the two tips, with equal-magnitude displacements assigned in opposite directions so that the tips move symmetrically closer together or farther apart as needed. These $\mathcal{W}$ -space displacements are then mapped into state updates by the selected refiner, either independently or jointly depending on the refinement strategy.

When the violation is local to a single arm, the correction is applied only to that arm. Thus, self-collision is resolved by moving the offending sampled point away from the colliding portion of the same arm. Obstacle handling is not performed by the refiner itself, and is instead left to the surrounding planner mechanics during node generation and edge validation. After each update, the resulting state is returned to the bounds of the current representation. The full refinement loop is provided in Algorithm 11.

8.1.4 Refining Strategies

Though the general strategy revolves around using a Jacobian to produce an arm displacement, there are a number of ways in which the Jacobian can be used to achieve this goal. The refiners discussed in this section differ only in how a desired displacement at a body point is converted into a state space change. The challenge is that moving one part of an arm to eliminate a collision may also move the tip, thereby disturbing the fixed-length tip distance between the two arms. The five refiners described below represent different ways of modifying the arm state while managing the tip positions.

Direct Jacobian Refiner

The simplest strategy applies the local Jacobian update in (8.3) directly. If p_0 denotes the offending point and $\hat{d}$ is the desired push direction, then the target is defined by

$$\hat{p} = p_0 + \delta \hat{d}, \quad (8.5)$$

and the arm state is updated by

$$q'_a = q_a + J(q_a, i, \xi)^\dagger (\hat{p} - p_0). \quad (8.6)$$

Thus, for a non-tip violation, the correction is produced directly from the section Jacobian associated with the offending body point. No explicit effort is made to preserve the tip during this update. Instead, the refiner addresses only the local violation at hand, and any resulting tip-separation error is left to be addressed, if needed, in a subsequent iteration of the general refinement loop. This makes the Direct Jacobian refiner simple and inexpensive, though repeated local corrections may accumulate unwanted tip drift.

Direct Correction Refiner

The Direct Correction refiner extends the Direct Jacobian strategy by adding an explicit tip restoration step. As before, let p_0 denote the offending body point and let $\hat{d}$ be the desired direction of correction. A target point $\hat{p} = p_0 + \delta \hat{d}$ is first defined, and the offending point is moved using the local section Jacobian. Thus, the first update is

$$q_{a,\text{moved}} = q_a + J(q_a, i, \xi)^\dagger (\hat{p} - p_0), Fi \quad (8.7)$$

Because this local correction may disturb the tip of the corrected arm, a second Jacobian update is then applied at that same arm's tip so that its original tip position is restored as closely as possible. If p_{tip} denotes the tip position of arm a before the first update, the corrected state is

$$q_{a,\text{corr}} = q_{a,\text{moved}} + J_{\text{tip}}(q_{a,\text{moved}})^\dagger (p_{\text{tip}} - p(q_{a,\text{moved}})). \quad (8.8)$$

Thus, the Direct Correction refiner uses two Jacobian steps on a single arm: the first resolves the local body-point violation, and the second restores that arm's tip. This tip restoration should be distinguished from the symmetric tip adjustment used in the general refinement loop when the active violation is the fixed tip-separation constraint itself. The method is conceptually simple while still preserving the coupled tip geometry more effectively than the Direct Jacobian refiner.

Null-Space Refiner

The Null-Space refiner follows the same overall goal as the Direct Correction refiner, specifically to resolve a local violation while preserving the tip position of the corrected arm. However, instead of first moving the offending point and then correcting the tip, it attempts to remove the tip-moving component of the first update before that update is applied.

Let p_0 denote the offending body point, and let $\hat{d}$ be the desired direction of correction. A target point $\hat{p} = p_0 + \delta\hat{d}$ is first defined, and a local correction δq_a is computed from the section Jacobian at the offending point. If $J_{\text{tip}}(q_a)$ denotes the tip Jacobian of the corrected arm, then the corresponding null-space projection is

$$N = I - J_{\text{tip}}(q_a)^\dagger J_{\text{tip}}(q_a). \quad (8.9)$$

The local update is then projected into this null space,

$$\delta q_{a,\text{ns}} = N \delta q_a, \quad (8.10)$$

so that the first-order component producing tip motion is removed.

This projected update is then applied to the arm state. Since the null-space projection preserves the tip only to first order, a small nonlinear tip error may still remain after the update. For this reason, a final tip-level Jacobian correction is applied at the tip of the same arm so that the original tip position is restored as closely as possible. As such, the Null-Space refiner first suppresses tip motion through the null-space projection and then uses a single-arm tip correction only to remove the residual error that remains. As above, this tip restoration is distinct from the symmetric tip adjustment used in the general refinement loop when the active violation is the fixed tip-separation constraint. The advantage of this strategy is that it preserves the coupled tip geometry more directly than the Direct Correction refiner, since the tip-moving component is removed before the main local update is applied.

Combined Jacobian Refiner

The Combined Jacobian refiner solves for body-point motion and tip preservation simultaneously. If $J(q_a, i, \xi)$ denotes the local Jacobian at the offending body point and $J_{\text{tip}}(q_a)$ the tip Jacobian, a single linear system is formed:

$$\begin{bmatrix} J(q_a, i, \xi) \\ J_{\text{tip}}(q_a) \end{bmatrix} \delta q = \begin{bmatrix} \delta\hat{d} \\ 0 \end{bmatrix}. \quad (8.11)$$

The upper block moves the offending point, while the lower block enforces zero tip displacement. This produces a unified correction step, though the resulting system may not work when the offending point lies close to the tip.

Gradient Descent Refiner

The preceding refiners follow the general refinement loop, resolving one active violation at a time through a local correction at a sampled body point. The Gradient Descent refiner does not follow this scheme. Instead, it treats refinement as the minimization of a global penalty function over the full dual-arm state, so that all active violations are addressed simultaneously.

To this end, we define a cost

$$C(q) = C_{\text{inter}}(q) + C_{\text{self}}(q) + C_{\text{tip}}(q), \quad (8.12)$$

where the first two terms penalize inter-arm collision and self-collision, respectively, and

$$C_{\text{tip}} = w_{\text{tip}} (\|p_1(q) - p_2(q)\| - L_{\text{rod}})^2. \quad (8.13)$$

8.1. STATE REFINERS

Table 8.1: Refiner benchmark results under the $\mathcal{C}$ -space and $\mathcal{J}$ -space representations.

Representation	Refiner	Refine (%)	Sample (%)	ref ms/t	smp ms/t
$\mathcal{C}$ -space	Direct Jacobian	97.5	49.9	0.0	0.3
	Direct Correction	1.4	5.5	10.0	45.0
	Null-Space	2.2	6.4	2.3	10.7
	Combined Jacobian	1.2	5.1	0.4	2.1
	Gradient Descent	3.1	4.2	0.1	0.5
$\mathcal{J}$ -space	Direct Jacobian	6.2	20.6	1.7	3.1
	Direct Correction	3.7	23.9	2.0	8.9
	Null-Space	3.3	17.2	2.1	9.4
	Combined Jacobian	4.5	16.7	1.7	7.7
	Gradient Descent	1.8	17.3	1.1	19.6

The gradient $\nabla C(q)$ is then computed using the same Jacobian quantities introduced above, and a descent step is taken in the full 12-dimensional state space:

$$q \leftarrow q - \alpha \nabla C(q), \quad (8.14)$$

where $\alpha > 0$ is a step size.

Thus, unlike the other refiners, the Gradient Descent refiner updates the state by descending a global violation cost rather than by applying a local correction within the general refinement loop. This makes the method the most global of the refiners, but also the most iterative and more prone to becoming stuck in local minima.

8.1.5 Test Environment

The five refiners are evaluated under two trial settings, each consisting of 100 trials. In the first, denoted *refine*, a single random dual-arm state is generated per trial by sampling all 12 parameters uniformly from the bounds of the chosen representation. The refiner is then invoked on that state for up to 100 iterations, except for the Gradient Descent refiner, which is allowed 300 iterations. A trial is counted as successful only if the returned state also passes an external validity check.

In the second setting, denoted *sample*, the refiner is used as a valid-state generator. For each trial, up to five random starts are allowed, each followed by a full refinement attempt. As above, a trial is counted as successful only if the returned state passes the external validity check.

Both tests are performed once for each refiner under the $\mathcal{C}$ -space representation and again under the $\mathcal{J}$ -space representation. The reported quantities are the refine success count, the sample success count, and the average time per trial.

8.1.6 Results

Table 8.1 summarizes the performance of each refiner in $\mathcal{C}$ -space and $\mathcal{J}$ -space. Here, *Refine* denotes the percentage of trials in which a random dual-arm state was successfully repaired, and *Sample* denotes the percentage of trials in which the refiner successfully generated a valid state. The columns *ref ms/t* and *smp ms/t* give the average runtime per trial in milliseconds for the refine and sample settings, respectively.

The Direct Jacobian refiner performs best overall. In $\mathcal{C}$ -space, it achieves 97.5% success in the refine setting and 49.9% in the sample setting, far exceeding the remaining methods, which achieve only 1.2%

to 3.1% and 4.2% to 6.4%, respectively. Thus, in $\mathcal{C}$ -space, the Direct Jacobian refiner is clearly the most effective method for both direct repair and valid-state generation. In $\mathcal{J}$ -space, performance is more moderate. Refine success ranges from 1.8% to 6.2%, while Direct Correction attains the highest sample success at 23.9%, followed by Direct Jacobian at 20.6%. However, Direct Correction requires considerably more computation per trial, with an average sample time of 8.9 ms versus 3.1 ms for Direct Jacobian. Thus, although Direct Correction achieves the highest raw sampling success in $\mathcal{J}$ -space, Direct Jacobian provides the strongest practical trade-off between success and runtime.

Overall, the Direct Jacobian refiner is the strongest choice. It is clearly superior in $\mathcal{C}$ -space and remains competitive in $\mathcal{J}$ -space while being substantially faster than the more expensive alternatives.

8.2 Dual-Arm IK Methods

Given a target midpoint $m^* \in \mathcal{W}$, the inverse kinematics problem is to determine a feasible dual-arm state q such that the midpoint induced by the two arm tips is sufficiently close to m^* , while also satisfying the fixed tip-separation constraint. Writing the midpoint as

$$m(q) = \frac{p_1(q) + p_2(q)}{2}, \quad (8.15)$$

the objective is to identify a feasible state satisfying

$$\|m(q) - m^*\| \leq \epsilon_{\text{IK}}, \quad (8.16)$$

together with

$$\|p_1(q) - p_2(q)\| = L_{\text{rod}}. \quad (8.17)$$

Because the feasible region is highly complex, all three IK strategies employ repeated random restarts. Each method begins from a randomly selected initial state in either the $\mathcal{J}$ -space or the $\mathcal{C}$ -space, performs a local numerical optimization, and retains the result only if it satisfies the validity check. The three methods differ in how they distribute the coupling between the two arms during optimization.

8.2.1 Simultaneous IK

The Simultaneous IK strategy retains the coupling between the two arms and solves a single constrained optimization problem in $\mathbb{R}^{12}$. In this case, the objective is simply the midpoint error

$$\min_{q \in \mathbb{R}^{12}} \|m(q) - m^*\|, \quad (8.18)$$

subject to the fixed tip-separation constraint and the representation-specific constraints. No intermediate geometric construction is imposed on either tip individually. Both arms are instead adjusted jointly until the coupled constraints are satisfied. This presents a unified optimization, though it also requires the optimizer to navigate a higher-dimensional constrained landscape.

8.2.2 Sequential IK

The Sequential IK strategy decomposes the 12-DoF problem into two successive 6-DoF subproblems. If the midpoint is to equal m^* and the tip separation is to equal L_{rod} , then

$$p_1(q) + p_2(q) = 2m^*, \quad \|p_1(q) - p_2(q)\| = L_{\text{rod}}. \quad (8.19)$$

8.2. DUAL-ARM IK METHODS

Together, these imply that each tip must lie on the sphere of radius $L_{\text{rod}}/2$ centered at m^* :

$$\|p_1 - m^*\| = \|p_2 - m^*\| = \frac{L_{\text{rod}}}{2}. \quad (8.20)$$

This gives rise to a two-stage construction. In the first stage, arm 1 is adjusted so that its tip reaches some feasible point on the sphere in (8.20). In the second stage, arm 2 is solved while arm 1 is held fixed, so that the midpoint error is minimized while the fixed tip-separation and remaining validity constraints are satisfied. Importantly, this method does not specify a point on the sphere beforehand. It searches for any feasible state satisfying the geometric constraints.

The advantage of Sequential IK is that each local optimization problem is only 6-dimensional. Its disadvantage is that the second stage depends strongly on the first: if arm 1 is placed on an unfavorable portion of the sphere, the second arm may fail even when a feasible 12-dimensional solution exists.

8.2.3 Midpoint-Orientation IK

The Midpoint-Orientation IK strategy removes the coupling from the optimizer by first choosing an orientation for a conceptual line segment of length L_{rod} centered at m^* . If $u \in S^2 \subset \mathbb{R}^3$ is a unit direction, then the corresponding tip locations are

$$e_1 = m^* - \frac{L_{\text{rod}}}{2}u, \quad e_2 = m^* + \frac{L_{\text{rod}}}{2}u. \quad (8.21)$$

These tip points are then converted into the local coordinate frames of the two arms, and two independent 6-DoF IK problems are solved.

This method yields the simplest per-arm IK problems, since the fixed tip separation is handled geometrically before optimization. However, unlike Sequential and Simultaneous IK, it explicitly induces a particular pair of tip targets through the sampled orientation. Poorly chosen orientations may therefore place one or both tips in unreachable or highly colliding regions, in which case the attempt is discarded.

8.2.4 Test Environment

The three IK methods are evaluated under both the $\mathcal{J}$ -space and the $\mathcal{C}$ -space representations. For each representation, 100 random target midpoints are sampled, and each method attempts to generate 1000 candidate states per target, for a total of 100,000 invocations. A returned state is counted in the reported total only if it passes the full validity check and satisfies the midpoint criterion used for success. The reported metrics are the total number of successful returned states, the overall success rate, the average time per target, the average time per candidate state, and the average midpoint error.

8.2.5 Results

In Table 8.2, *TotalValid* denotes the total number of valid returned states across all trials, *Success* denotes the percentage across all trials that produced a state satisfying the IK tolerance, *AvgTime* is the average computation time per target, *ms/pt* is the average computation time per candidate state tested, and *AvgError* is the average midpoint error of the returned states. The IK results in Table 8.2 show a clear difference between the two representations. In the $\mathcal{C}$ -space, none of the methods performs well enough to serve as a practical goal-generation mechanism. Sequential and Simultaneous IK produce no successful states, while Midpoint-Orientation IK yields only negligible success. Thus, unlike the refiner benchmark, the IK results strongly disfavor the $\mathcal{C}$ -space.

8.3. DUAL-ARM RRT*

Table 8.2: Comparison of inverse kinematics methods under the $\mathcal{J}$ -space and $\mathcal{C}$ -space representations.

Method	Repr.	TotalValid	Success (%)	AvgTime (s)	ms/pt	AvgError
Simultaneous	Joint	21964	22.0	10.7260	10.73	0.01166
	Config	0	0.0	14.5309	14.53	0.19583
Sequential	Joint	45383	45.4	3.2048	3.20	0.01975
	Config	0	0.0	0.2502	0.25	0.14912
Midpoint-Orientation	Joint	7046	7.0	12.0412	12.04	0.00181
	Config	46	0.0	11.3551	11.36	0.04000

In the $\mathcal{J}$ -space, all three methods generate successful states, but Sequential IK is the clear best overall. It achieves the highest success rate at 45.4% while also requiring the least time per target among the effective methods. Simultaneous IK is less reliable and substantially more expensive, and Midpoint-Orientation IK, though highly accurate when successful, succeeds infrequently.

8.3 Dual-Arm RRT*

A standard state space RRT* implementation is difficult to apply directly to the dual-arm model. The state space is highly constrained, so valid states and even feasible regions may be difficult to identify, and states that are valid may be destroyed by small perturbations. For this reason, the dual-arm RRT* method discussed here combines tip-based exploration in the $\mathcal{W}$ -space with Jacobian-based configuration generation and local refinement. Consequently, the planner preserves the overall structure of RRT* while adapting its initialization, extension, and validation steps to the dual-arm model.

8.3.1 RRT* Formulation

For the RRT* tree $\mathcal{T}$, each node $n \in \mathcal{T}$ stores a dual-arm joint-space state $q_n \in \mathcal{J}$ together with the corresponding stacked tip vector

$$\tau(q_n) = \begin{bmatrix} p_1(q_n) \\ p_2(q_n) \end{bmatrix} \in \mathbb{R}^6. \quad (8.22)$$

Thus, the tree stores states in the $\mathcal{J}$ -space, while exploration is guided in the 6-dimensional tip space. The corresponding midpoint

$$m(q_n) = \frac{p_1(q_n) + p_2(q_n)}{2} \quad (8.23)$$

is used only for the goal check. The $\mathcal{C}$ -space is used only during Jacobian extension and local refinement, and is never stored in the tree itself.

Before tree growth begins, a goal tip target is constructed. Starting from the tip pair of the resting start state, both tips are translated so that their midpoint coincides with the desired target midpoint m^* . The orientation is therefore inherited from the resting configuration. Although this goal tip pair is only a heuristic and need not be exactly achievable, it provides a 6-dimensional target for goal-biased sampling.

8.3.2 Sampling Strategy

Sampling in Tip Space

At each iteration, a stacked tip target

$$r = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} \in \mathbb{R}^6 \quad (8.24)$$

is sampled directly in tip space. With a fixed probability (0.15), the sample is taken to be the precomputed goal tip target. Otherwise, a random midpoint m_{rand} is selected uniformly in the $\mathcal{W}$ -space, and a random orientation is generated by sampling a unit direction $d \in S^2$, where $S^2 \subset \mathbb{R}^3$ denotes the unit sphere. The two sampled tips are then placed at distance $L_{\text{rod}}/2$ on either side of the midpoint:

$$r_1 = m_{\text{rand}} - \frac{L_{\text{rod}}}{2}d, \quad r_2 = m_{\text{rand}} + \frac{L_{\text{rod}}}{2}d. \quad (8.25)$$

Thus, each random sample is a pair of 3-dimensional tip positions that is exactly one rod length apart, centered at a uniformly sampled $\mathcal{W}$ -space point, and assigned a uniformly random rod orientation.

Nearest Node Selection with Arm Symmetry

Once a tip target has been sampled, the planner identifies the nearest existing node in tip space. Since the two arms are interchangeable with respect to the rod constraint, the sampled target is compared to each node in two ways: first using the original tip ordering, and second using the arm-swapped ordering

$$\tilde{r} = \begin{bmatrix} r_2 \\ r_1 \end{bmatrix}. \quad (8.26)$$

For each node $n \in \mathcal{T}$, the planner computes

$$d(n, r) = \|p_1(q_n) - r_1\| + \|p_2(q_n) - r_2\| \quad (8.27)$$

and

$$d(n, \tilde{r}) = \|p_1(q_n) - r_2\| + \|p_2(q_n) - r_1\|. \quad (8.28)$$

The smaller of these two values is taken to be the effective distance from node n to the sample. The nearest node is then the node that minimizes this effective distance over the tree. If the swapped assignment is better, the sampled target is replaced by $\tilde{r}$ before proceeding. The extension step always uses the tip assignment that best matches the selected node, rather than treating two equivalent arm assignments as artificially distant.

8.3.3 Growth Strategy

Jacobian-Based Extension

After the nearest node $n \in \mathcal{T}$ is identified, steering defines a new tip target $\hat{r}$ by moving from $\tau(q_n)$ toward the sampled target r , while limiting the maximum per-arm tip displacement. Thus, the planner performs a local extension in the sampled direction rather than attempting to connect directly to the full sampled tip pair. Small steps also make the subsequent Jacobian-based extension substantially more stable.

Starting from the nearest node's state $q_n = [q_{n,1}^T \ q_{n,2}^T]^T$, each arm state $q_{n,a}$, $a \in \{1, 2\}$, is converted into the $\mathcal{C}$ -space and updated iteratively using the Jacobian configuration-generation method

introduced in Chapter 6 and a very small tip change. For each arm, up to ten local Jacobian steps are performed, with the arm order randomized to avoid bias. Near-zero parameters are perturbed slightly to avoid singular behavior, and the iteration is terminated early if the target is reached, if progress becomes negligible, or if the local error grows substantially. The resulting state is then mapped back into the $\mathcal{J}$ -space and checked for validity. Though it is possible to perform this extension using the $\mathcal{J}$ -space representation, in practice it is far less stable and leads to more invalid states.

Refinement During Extension

If the Jacobian extension produces an infeasible but near-feasible state, a rescue step is attempted. A refiner is applied (converted to the $\mathcal{C}$ -space if necessary) to the candidate so that body-point violations may be corrected while preserving the tip positions as closely as possible. The refined state is accepted only if it becomes valid, remains sufficiently close to the parent in joint space, and does not drift too far from the intended steered target $\hat{r}$. Thus, the refiner acts only as a local recovery mechanism for extension attempts that are already close to valid.

Rewiring

Once a valid candidate node has been obtained, the RRT* standard parent selection and rewiring occur. The transition cost between two nodes $m, n \in \mathcal{T}$ is defined by the total motion of the two tips,

$$\zeta(m, n) = \|p_1(q_m) - p_1(q_n)\| + \|p_2(q_m) - p_2(q_n)\|, \quad (8.29)$$

and if $P = (n_s = n_0, n_1, \dots, n_k = n)$ is the path from a root node n_s to n , then the path cost of n is

$$v(n) = \sum_{j=0}^{k-1} \zeta(n_j, n_{j+1}). \quad (8.30)$$

Among nearby nodes for which a valid edge can be constructed, the parent yielding the smallest cumulative cost is selected. Edges are validated by linear interpolation in the $\mathcal{J}$ -space, with feasibility checked at each intermediate state. After insertion, nearby nodes are rewired whenever passing through the new node decreases their total path cost.

Goal Check and Path Extraction

A node is treated as a goal candidate once its midpoint satisfies

$$\|m(q_n) - m^*\| \leq \epsilon_{\text{goal}}. \quad (8.31)$$

The implementation used here terminates after the first goal is found rather than continuing to improve the solution. The least-cost goal candidate is then selected, and the corresponding path is extracted by following parent pointers back to its root. The resting/start configuration is added to the resulting sequence so that the final output is a path in the $\mathcal{J}$ -space beginning from the resting configuration.

8.3.4 Multi-Root Seeding

The Jacobian configuration generation method is unstable near the resting/start/zero configuration, so repeated extension from the starting state is unreliable. To address this issue, the planner is initialized with many valid root nodes rather than only one.

Algorithm 12: Dual-Arm RRT*

Input: Target midpoint m^* ; maximum iterations K
Output: Path P of dual-arm states or failure

- 1 construct the heuristic goal tip target r_{goal} from the zero configuration;
- 2 initialize $\mathcal{T}$ with the starting node and the multi-root seed set;
- 3 **for** $k = 1$ **to** K **do**
- 4 sample a stacked tip target r , using r_{goal} with probability 0.15;
- 5 find the nearest node $n \in \mathcal{T}$ in tip space, accounting for arm symmetry;
- 6 steer from $\tau(q_n)$ toward r to obtain a bounded target $\hat{r}$;
- 7 extend from q_n toward $\hat{r}$ using Jacobian-based configuration generation to obtain q_{cand} ;
- 8 **if** q_{cand} *is invalid* **then**
- 9 | attempt local repair with a refiner;
- 10 **end**
- 11 **if** *no valid candidate state is obtained* **then**
- 12 | continue;
- 13 **end**
- 14 choose the least-cost valid parent among nearby nodes;
- 15 insert the new node into $\mathcal{T}$ and perform rewiring when this lowers total path cost;
- 16 **if** $\|m(q_{\text{cand}}) - m^*\| \leq \epsilon_{\text{goal}}$ **then**
- 17 | extract the path to a root, prepend the zero configuration, and return P ;
- 18 **end**
- 19 **end**
- 20 **return** failure;

These seed states are generated through a deterministic sweep in the $\mathcal{C}$ -space. For each arm and for each section i , the curvature parameter is fixed at $\phi_i = 0.1$, while the bending-plane parameter is taken as $\theta_i = \theta$, where θ is swept linearly from $-\pi/2$ to $+\pi/2$ across 2000 values. Geometrically, this produces a group of slightly bent configurations whose tips trace a rotation around the resting pose.

8.3.5 Multi-Step Planning for Tracing

For tracing tasks, the target is not a single midpoint m^* but a sequence of target midpoints $(m_1^*, m_2^*, \dots)$. In this case, the planner is applied segment by segment. The first segment uses the full multi-root seeding strategy described above so that exploration begins from a diverse set of valid shapes. Once a path segment has been found, its final state is used as the starting state for the next segment, and the planner is then run in a single-root form grown from that configuration. The resulting state sequences are concatenated, with the duplicate end/start states removed. As such, tracing is reduced to a sequence of coupled planning problems while still preserving continuity from one segment to the next.

8.4 Dual-Arm PRM

Unlike the dual-arm RRT* approach, which grows a single search tree toward one target midpoint, the dual-arm PRM constructs a roadmap in the $\mathcal{J}$ -space. This is especially useful in the present setting, since the task is specified by a desired midpoint $m^* \in \mathcal{W}$ rather than by a unique configuration. Using

bulk dual-arm IK, valid-state generation, and local refinement, the PRM constructs feasible trajectories while working primarily in the $\mathcal{J}$ -space.

8.4.1 PRM Formulation

Let $R = (V, E)$ denote the roadmap. Each node of R stores a valid dual-arm state $q_n \in \mathcal{J}$, and the valid starting state q_0 is inserted as the start node. The planner then constructs a collection of goal nodes associated with the target midpoint m^* , and attempts to connect the start node to at least one such goal node through local connections in the roadmap.

If the initial roadmap does not connect the start node to a goal node, additional valid states are generated through random sampling and refinement in the $\mathcal{J}$ -space. Each accepted sample is inserted into the roadmap and connected to nearby nodes. Thus, the roadmap grows until the start node lies in the same connected component as a goal node, or until the sample budget is exhausted.

8.4.2 Many-Goal Strategy

The many-goal strategy allows the PRM to exploit the midpoint-based task definition. Since the target is a point $m^* \in \mathcal{W}$ rather than a unique dual-arm state, there may exist many feasible configurations $q \in \mathcal{J}$ satisfying

$$\|m(q) - m^*\| \leq \epsilon_{\text{goal}}.$$

Rather than selecting only one such state, the planner generates a collection of candidate goal states and attempts to incorporate all valid candidates into the roadmap.

Let

$$Q_R = \{ q \in \mathcal{J} : \|m(q) - m^*\| \leq \epsilon_{\text{goal}} \}$$

denote the set of valid dual-arm goal states associated with m^* . In practice, Q_R is not known explicitly, so the planner instead generates a finite set of candidate states using a dual-arm IK method. Each candidate is checked for full validity, and invalid candidates may be repaired through refinement before insertion. The retained candidates are then inserted into the roadmap as goal nodes.

If the number of verified goal states becomes too large, the set is reduced before insertion by retaining a collection of lower-norm states together with a random subset of the remainder. This way, the planner preserves both simple goal states near the resting pose and geometric diversity among the remaining feasible candidates.

8.4.3 Edge Connection with Refinement

The local connection procedure is the central feature of the dual-arm PRM. Given two roadmap nodes $n_i, n_j \in V$ with associated states $q_i, q_j \in \mathcal{J}$, the planner attempts to connect them by advancing along the straight interpolation in the ambient parameter space $\mathbb{R}^{12}$,

$$\bar{q}_{ij}(\lambda) = (1 - \lambda)q_i + \lambda q_j, \quad \lambda \in [0, 1].$$

This interpolation is used only to generate candidate intermediate states; it is not assumed to remain feasible throughout. The segment is traversed in small increments, and each intermediate state is checked for validity. If an interpolated state is valid, it is accepted and the traversal continues. Moreover, whenever the accepted motion since the last inserted roadmap node exceeds a fixed connection threshold, an intermediate node is inserted. Thus, a successful long connection may contribute a chain of additional roadmap nodes rather than only a single edge.

Algorithm 13: Dual-Arm PRM

Input: Valid start state q_0 ; target midpoint m^* ; maximum samples N
Output: Path P of dual-arm states or failure

- 1 initialize the roadmap R with the start node q_0 ;
- 2 generate a collection of candidate goal states for m^* using dual-arm IK;
- 3 validate and refine the candidate goals as needed, then insert the retained goal nodes into R ;
- 4 attempt local connections for nearby node pairs by stepping along their $\mathcal{J}$ -space interpolation;
- 5 insert feasible intermediate states into R ; when an intermediate state is invalid, attempt refinement and continue only if the repaired state is accepted;
- 6 **if** *the start node is connected to a goal node* **then**
- 7 | return P ;
- 8 **end**
- 9 **for** $k = 1$ **to** N **do**
- 10 | generate a valid sample q_{new} in the $\mathcal{J}$ -space;
- 11 | insert q_{new} into R ;
- 12 | attempt local connections between q_{new} and nearby nodes by incremental interpolation;
- 13 | insert feasible intermediate states into R ; when an intermediate state is invalid, attempt refinement and terminate that connection if repair fails;
- 14 **if** *the start node is connected to a goal node* **then**
- 15 | return P ;
- 16 **end**
- 17 **end**
- 18 **return** failure;

If an interpolated state is infeasible, a refinement step is attempted instead of immediately terminating the connection. The invalid state is converted into the $\mathcal{C}$ -space (if necessary), where a refiner is applied, and the repaired state is then mapped back into the $\mathcal{J}$ -space (if necessary). This repaired state is accepted only if it passes the validity check and remains sufficiently close to the most recently accepted roadmap state. Otherwise, the connection attempt is terminated.

Consequently, a connection attempt may succeed only partially: even if the planner does not reach q_j , any valid intermediate states accepted before failure remain in the roadmap. Consequently, an unsuccessful attempt may still contribute useful roadmap structure for later samples and connection attempts. This refinement mechanism is particularly useful in the dual-arm setting, where interpolation in the $\mathcal{J}$ -space may temporarily violate the fixed tip-separation constraint or induce self-collision or inter-arm collision even when the endpoint states are both valid.

8.5 Planner Evaluation

Test Environment - Obstacle Avoidance

To compare the dual-arm RRT* and PRM planners in cluttered environments, randomized obstacle avoidance trials were generated with obstacle counts 1, 2, 4, 6, and 8. For each obstacle count, 100 trials were performed for each planner.

In each trial, a target midpoint was sampled from a bounded workspace region, and a collection of spherical obstacles was placed between the initial midpoint and the target. Obstacle placement was

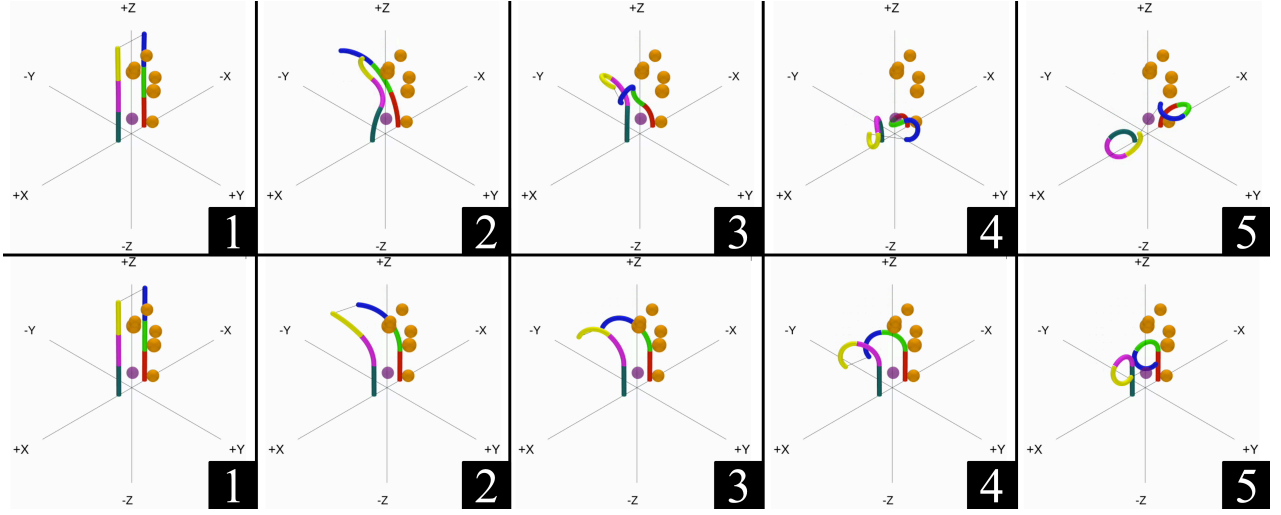


Figure 8.1: A path generated by the Dual-Arm PRM (top) and RRT* (bottom) approaches for a trial with 6 obstacles. The target midpoint is shown in purple. Frames are shown of the initial position (1), 25% progress (2), 50% progress (3), 75% progress (4), and the final position (5).

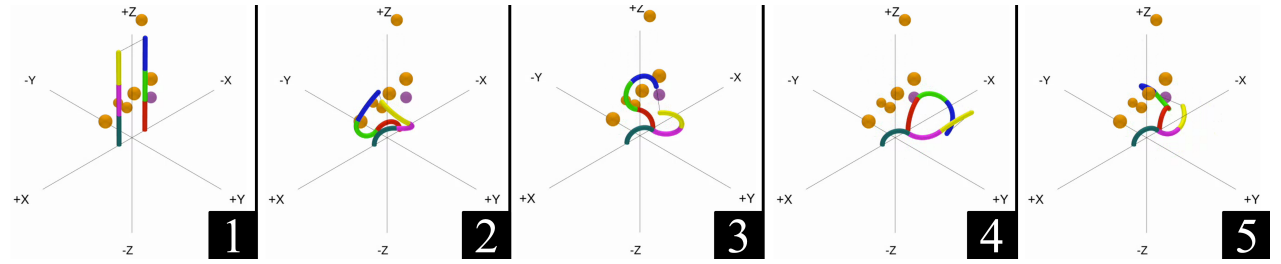


Figure 8.2: A path generated by the Dual-Arm PRM approach for a trial with 6 obstacles. The target midpoint is shown in purple. Frames are shown of the initial position (1), 25% progress (2), 50% progress (3), 75% progress (4), and the final position (5). Note that the Dual-Arm RRT* approach could not produce a path for this trial.

restricted so that the target was not occluded directly and the initial dual-arm state remained valid. Both planners were evaluated on the same obstacle sets.

For each planner and obstacle count, the reported quantities are the number of successful trials, the average computation time per trial in seconds, and the average $\mathcal{C}$ -space path cost over the successful trajectories. The path cost is calculated as the total Euclidean change in the $\mathcal{C}$ -space states constituting the path, with a lower cost indicating a smoother path.

Results

Table 8.3 summarizes the obstacle-avoidance results. In the table, *Success* denotes the percentage of successful trials out of 100, *AvgTime* denotes the average computation time per trial in seconds, and *Cost* denotes the average $\mathcal{C}$ -space path length over the successful trajectories.

The PRM performs more reliably than the RRT* in every obstacle setting. With one obstacle, the PRM succeeds in 65% of trials, compared to 25% for the RRT*. This advantage persists as the number of obstacles increases, with the PRM achieving success rates of 49%, 34%, 22%, and 24% for 2, 4, 6, and 8 obstacles, respectively, while the corresponding RRT* success rates are 16%, 4%, 12%, and 8%. The PRM also requires less computation time on average in each setting, though the gap narrows in the most cluttered cases.

8.5. PLANNER EVALUATION

Table 8.3: Obstacle-avoidance results for the dual-arm planners.
(Success is reported as a percentage, and time is reported in seconds.)

Obs.	Planner	Success (%)	AvgTime (s)	Cost
1	RRT*	25.0	47.8	13.3848
	PRM	65.0	34.8	18.7555
2	RRT*	16.0	68.5	17.3807
	PRM	49.0	45.2	27.6802
4	RRT*	4.0	78.6	13.5714
	PRM	34.0	48.6	27.5996
6	RRT*	12.0	87.6	19.6761
	PRM	22.0	59.7	55.1516
8	RRT*	8.0	62.5	17.5852
	PRM	24.0	61.0	40.0513

The RRT*, however, produces lower cost paths whenever it succeeds. This is consistent across all obstacle counts, indicating that the tree-based exploration often yields more direct trajectories, but does so with substantially lower reliability.

Figure 8.1 shows a trajectory generated by both approaches for the same trial. Figure 8.2 show a PRM trajectory for a trial where RRT* could not produce a result. Subjectively, the RRT* paths look smoother, but suffer from the aforementioned reliability issues. Furthermore, though the arms are able to maintain the tip constraints, their orientations are not amenable to manipulation tasks.

Overall, for the obstacle-avoidance task, the PRM provides the stronger planning strategy, while the RRT* produces smoother successful trajectories at the cost of a much lower success rate. Notably, these trajectories maintain a constant distance between the tips, but there is work to be done in maintaining a consistent orientation.

Test Environment - Shape Tracing

To evaluate multi-step planning performance, the two planners were also tested on a collection of shape-tracing tasks with no obstacles present. In these trials, the midpoint of the dual-arm system was required to visit a prescribed sequence of target midpoints defining a geometric shape. The shapes considered were a circle in the xy -plane, a circle in the xz -plane, a figure-eight in the xy -plane, a figure-eight in the xz -plane, a line in the y -direction, and a line in the x -direction. For each shape, 10 independent trials were performed.

A trial was counted as successful only if the full sequence of target midpoints was completed. For the RRT*, the tracing procedure used the multi-step formulation introduced earlier in the chapter. For the PRM, the trajectory was generated segment by segment, with the terminal state of one segment used as the initial state of the next.

For each planner and shape, the reported quantities are the percentage of successful trials, the average computation time per trial in seconds, and the average $\mathcal{C}$ -space path cost over the successful trajectories. The path cost is calculated as the total Euclidean change in the $\mathcal{C}$ -space states constituting the path, with a lower cost indicating a smoother trajectory.

Results

Table 8.4 summarizes the tracing results. In the table, *Success* denotes the number of successful trials out of 10, *AvgTime* denotes the average computation time per trial in seconds, and *Cost* denotes the

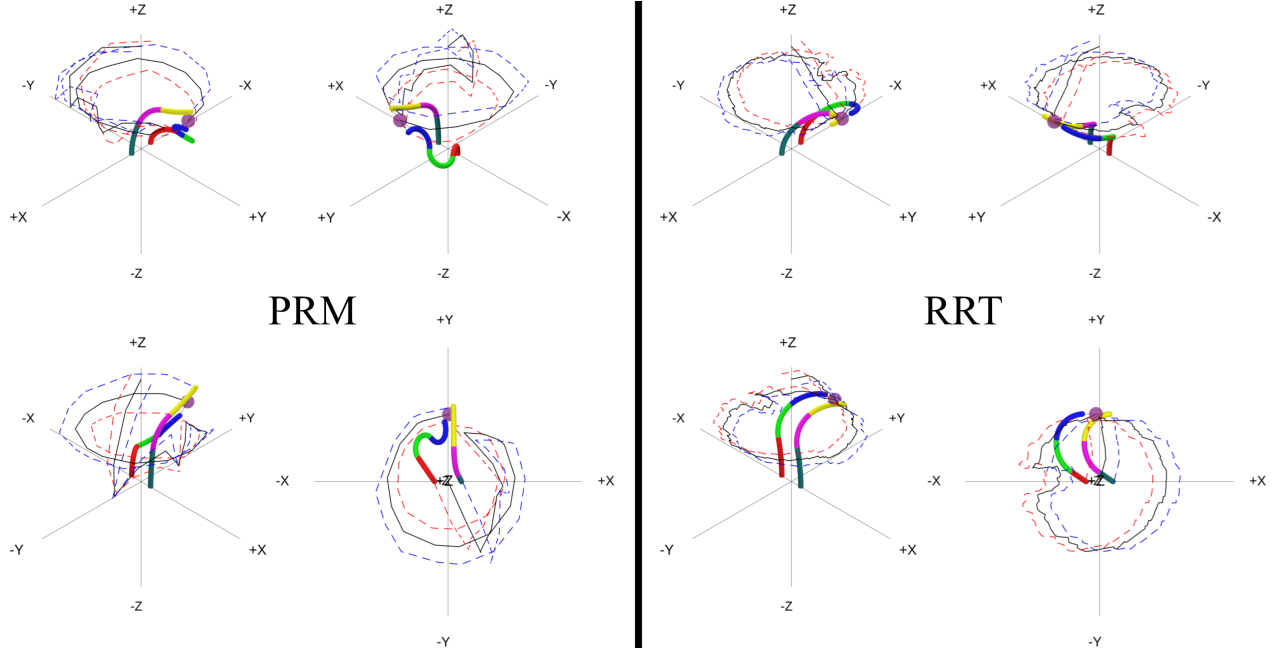


Figure 8.3: An example of a circle on the XY plane traced by the PRM (left) and RRT* (right) planners. The target midpoint is shown in purple. The midpoint trajectory is shown in black, the Arm 1 trajectory is shown in red, and the Arm 2 trajectory is shown in blue.

average $\mathcal{C}$ -space path cost over the successful trajectories. A dash indicates that no successful trial was obtained for that planner and shape.

The tracing results are more mixed than those of obstacle avoidance, but the PRM remains the more versatile planner overall. It succeeds on the circle in the xy -plane, the circle in the xz -plane, the figure-eight in the xy -plane, and both line tasks, whereas the RRT* succeeds on the two circles and the two line tasks but fails on both figure-eight tasks. In particular, the PRM is the only method to complete the figure-eight in the xy -plane. Both planners fail on the figure-eight in the xz -plane, indicating that these tracing tasks remain particularly challenging for the dual-arm system.

The strongest RRT* result appears on the circle in the xz -plane, where it succeeds in 6/10 trials compared to 4/10 for the PRM, and also requires substantially less computation time. More generally, the RRT* tends to be faster whenever it succeeds, but this does not translate into broader reliability. The PRM attains higher success on the circle in the xy -plane and on both line tasks, though at a higher computational cost in most cases. In terms of smoothness, when RRT* produces a trajectory it is typically smoother.

Figure 8.3 and Figure 8.4 show the discussed planners tracing a circle situated on the XY and XZ planes, respectively. Subjectively the RRT* planner seems to produce smoother paths, but the reliability issues mean that the planned trajectory might deviate from the prescribed one.

8.5. PLANNER EVALUATION

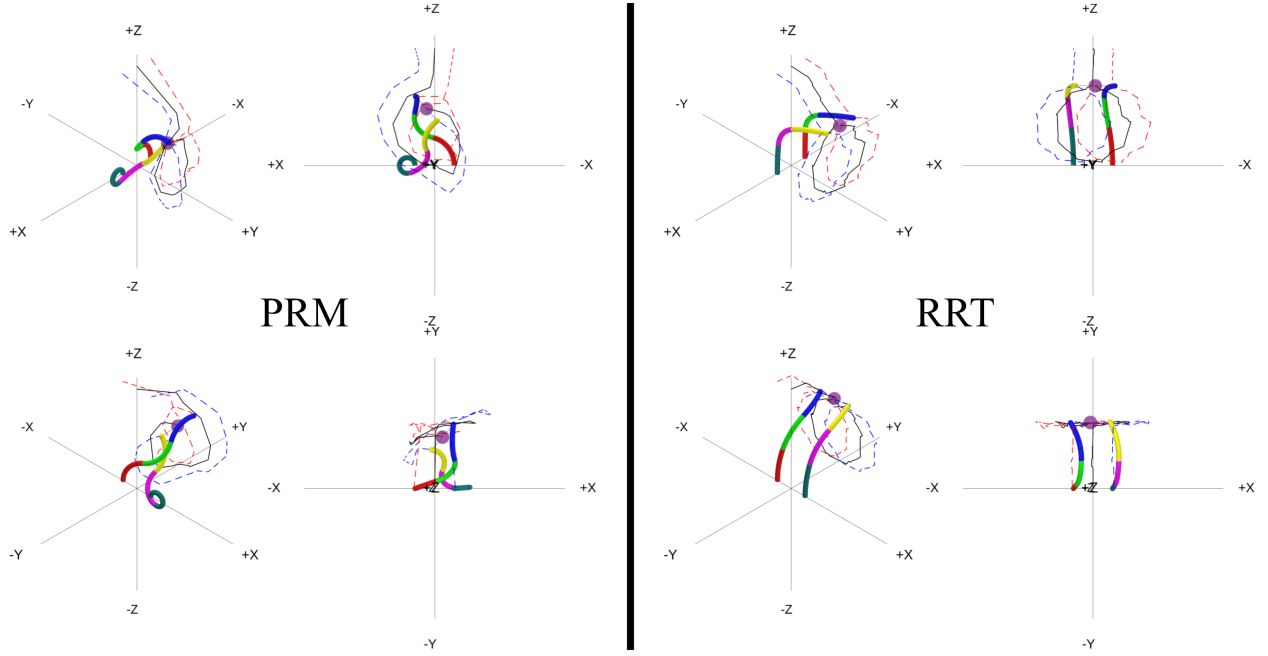


Figure 8.4: An example of a circle on the XZ plane traced by the PRM (left) and RRT* (right) planners. The target midpoint is shown in purple. The midpoint trajectory is shown in black, the Arm 1 trajectory is shown in red, and the Arm 2 trajectory is shown in blue.

Table 8.4: Shape-tracing results for the dual-arm planners.

(Success is reported as a percentage, time is reported in seconds, and — indicates that no successful trial was obtained.)

Shape	Planner	Success (%)	AvgTime (s)	Cost
Circle (XY)	RRT*	20.0	58.0	8.0469
	PRM	60.0	225.7	15.1021
Circle (XZ)	RRT*	60.0	22.4	6.6391
	PRM	40.0	136.9	8.5130
Figure-8 (XY)	RRT*	0.0	84.2	—
	PRM	60.0	267.1	16.2500
Figure-8 (XZ)	RRT*	0.0	65.2	—
	PRM	0.0	124.7	—
Line in X	RRT*	20.0	38.9	4.5153
	PRM	60.0	71.9	8.4670
Line in Y	RRT*	20.0	103.9	11.9526
	PRM	40.0	90.4	5.7432

Chapter 9

Conclusion

9.1 Overview

This dissertation has considered the problem of path planning for multisection continuum arms. These robots offer important advantages in dexterity, compliance, and safety, but their nonlinear kinematics, redundancy, and tendency toward unreliable inverse kinematics behavior make path planning considerably more difficult than for conventional rigid manipulators. Accordingly, the central goal of this work has been the development of planning methods that improve reliability while also producing smooth and practically meaningful trajectories.

To address this problem, the dissertation developed a sequence of planning approaches of increasing scope and capability. The presented methods begin with graph-theoretic planning in discretized workspace models, then incorporate improved inverse kinematic configuration generation, temporal reasoning for dynamic environments, and sampling-based planning in the $\mathcal{W}$ -space, $\mathcal{C}$ -space, and $\mathcal{J}$ -space. This progression culminates in the study of coordinated planning for dual continuum arms, where the difficulty of the problem is compounded by shared geometric constraints and the need to coordinate two manipulators simultaneously.

Taken together, these results show that this problem can be addressed in multiple ways, though each approach offers different advantages. The strongest performance is obtained by selecting the planning representation and strategy according to the structure of the task at hand. In this sense, the dissertation supports a hybrid view of continuum arm planning, in which graph-theoretic, inverse-kinematic, and sampling-based techniques are used in complementary ways to overcome the limitations of any single method.

9.2 Summary of Work

9.2.1 Graph-Theoretic Planning for Continuum Arms

The first planning framework developed in this dissertation employed graph-theoretic tools to avoid the direct dependence on conventional inverse kinematics during online planning. By discretizing the workspace and associating feasible robot configurations with local workspace regions, this approach demonstrated that reliable path planning could be achieved even in the presence of the challenging kinematic behavior characteristic of continuum arms. This result served as an important foundation for the remainder of the dissertation.

This initial framework also highlighted an important tradeoff. While graph-theoretic planning improved reliability, the quality and diversity of the resulting trajectories depended strongly on the

available set of stored configurations. This observation motivated the next stage of the research, which sought to improve configuration generation while preserving the reliability benefits of the graph-based approach.

9.2.2 Improved Configuration Generation via Inverse Kinematics

The second contribution of this dissertation augmented the graph-theoretic framework with improved inverse kinematic configuration generation. Rather than relying exclusively on a fixed offline set of stored configurations, the planner used rapid inverse kinematic routines to enrich the available states used during graph construction and path generation. In doing so, the method retained the reliability benefits of the graph structure while improving pose diversity and trajectory smoothness.

This stage of the work showed that inverse kinematics remains highly useful for continuum arm planning when it is employed selectively and within a planning framework. In particular, IK was used not as a complete planner, but as a local configuration-generation mechanism that strengthened the global search process. This integration improved trajectory quality and broadened the range of tasks that could be handled successfully.

9.2.3 Anticipatory Path Planning in Dynamic Environments

Having established reliable planning in static settings, the dissertation next addressed dynamic path planning through an anticipatory planning framework. This method combined graph-theoretic planning with temporal reasoning and obstacle-motion prediction, allowing the planner to account for future obstacle states rather than reacting only to the present environment. The resulting approach showed that temporal graphs provide a natural mechanism for representing feasible motion when obstacle availability changes over time.

This contribution was significant in two respects. First, it demonstrated that continuum arm planning in dynamic environments benefits from prediction and temporal structure. Second, it showed that planning for such arms may be formulated in a way that explicitly balances collision avoidance and smoothness over time. The anticipatory planner therefore served as an important bridge between the earlier graph-theoretic methods and the more general sampling-based dynamic planners developed later in the dissertation.

9.2.4 $\mathcal{W}$ -space RRT* for Static Obstacle Avoidance and Tracing

The dissertation next considered sampling-based planning through the development of a $\mathcal{W}$ -space RRT* framework. This method explored the workspace directly, while using Jacobian-based projection to determine valid arm states. By planning in $\mathcal{W}$ -space, the approach more directly reflected the geometric objectives of continuum arm motion, especially in obstacle avoidance and trajectory tracking tasks where tip behavior is of primary importance.

The results showed that $\mathcal{W}$ -space exploration yields trajectories of substantially higher quality than direct exploration in configuration representations that do not preserve the desired workspace behavior. In particular, the $\mathcal{W}$ -space RRT* approach reduced unnecessary ancillary motion and produced more natural trajectories for both obstacle avoidance and tracing tasks. This chapter demonstrated that sampling-based planning can be highly effective for continuum manipulators when the exploration space is selected to match the geometric structure of the task.

9.2.5 Dynamic $\mathcal{W}$ -space RRT*

Building on the static $\mathcal{W}$ -space RRT* framework, the dissertation then introduced a reactive dynamic planner capable of handling moving obstacles. This approach incorporated mechanisms such as obstacle inflation, distance records, and backup behavior to enable robust online adaptation as the environment changed. Unlike the anticipatory graph-theoretic method, which relied heavily on prediction, this planner emphasized reactive behavior and continual trajectory adjustment.

This contribution showed that a $\mathcal{W}$ -space sampling-based planner can remain effective even when future obstacle motion is not fully known. In addition, the planner was validated not only in simulation but also on a continuum arm prototype, strengthening the practical relevance of the proposed approach. Accordingly, this chapter provided an important demonstration that the theoretical and algorithmic developments of the dissertation can extend beyond numerical study and support physical execution.

9.2.6 Temporal $\mathcal{J}$ -space PRM for Dynamic Planning

The strongest dynamic planning results in the dissertation were achieved through the temporal $\mathcal{J}$ -space PRM framework. This planner combined batched goal inverse kinematics, static roadmap construction, temporal-graph validation, obstacle prediction, and replanning into a four-phase procedure. The use of $\mathcal{J}$ -space was particularly effective for roadmap construction, since linear interpolation in joint space more naturally reflects the physical transitions executed by the arm than corresponding linear motions in the $\mathcal{C}$ -space.

This method demonstrated the value of combining the multi-query structure of PRM with temporal graphs. The roadmap provided broad connectivity and reuse across replanning steps, while the temporal layer encoded time-dependent feasibility under predicted obstacle motion. As a result, the temporal $\mathcal{J}$ -space PRM achieved the strongest overall performance among the tested single-arm dynamic planners, particularly in the more complicated dynamic environments. This chapter therefore represents a major culmination of the single-arm portion of the dissertation, showing that robust dynamic planning for continuum arms can be achieved by integrating sampling-based planning with temporal reasoning.

9.2.7 Dual-Arm State Refinement and Inverse Kinematics

The final technical portion of the dissertation extended the planning problem from one continuum arm to a dual-arm system subject to a fixed tip-separation constraint. This introduced a substantially more difficult planning problem, since valid states must satisfy both the individual arm constraints and the shared relationship between the two manipulators. To address this, the dissertation developed a general refinement framework together with several specific refinement strategies, as well as multiple inverse kinematic methods for generating valid dual-arm states.

This work showed that representation choice remains critical in the dual-arm setting. Among the tested refiners, the Direct Jacobian method emerged as the strongest overall approach, particularly in $\mathcal{C}$ -space, where it clearly outperformed the alternatives in both repair and valid-state generation. The dual-arm IK work likewise demonstrated that different methods offer different tradeoffs between reliability and practicality. This reinforced that effective planning for continuum arms often depends on selecting the representation and solver most appropriate to the task.

9.2.8 Sampling-Based Planning for Dual Continuum Arms

The dissertation concluded with dual-arm RRT* and PRM planners for obstacle avoidance and shape tracing. These planners extended the sampling-based philosophy developed for single-arm systems to a coordinated two-arm setting, where midpoint goals, fixed tip separation, and shared validity constraints must all be respected. The resulting study demonstrated that coordinated planning for continuum arm pairs is feasible, but significantly more difficult than the corresponding single-arm problem.

The results revealed a meaningful distinction between the two planners. For obstacle avoidance, the PRM was the more reliable planner overall, whereas the RRT* algorithm tended to produce shorter successful paths when it succeeded. For shape tracing, the PRM algorithm again proved more versatile across a broader range of tasks, while the RRT* algorithm produced smoother paths when able. These results show that the extension from one arm to two fundamentally changes the planning problem and places greater emphasis on reliable state generation, constraint adherence, and state space connectivity.

9.3 Future Research Directions

Several natural research directions follow from the work presented in this dissertation. The first is the extension of temporal and dynamic planning ideas to the dual-arm setting. While this dissertation demonstrated coordinated dual-arm planning for obstacle avoidance and tracing, the dynamic case remains open. A temporal multi-arm planning framework that combines prediction, coordinated refinement, and online replanning would be a natural continuation of the present results.

A second direction concerns experimental validation. Although the dissertation includes prototype testing for the dynamic $\mathcal{W}$ -space RRT* planner, most of the presented methods were evaluated in simulation. Extending these methods to physical systems would require addressing sensing uncertainty, model mismatch, actuator limitations, and closed-loop execution. Such work would strengthen the connection between the proposed planning frameworks and the deployment of continuum arms in practical settings.

A third direction is to extend the static dual-arm planning framework toward more general manipulation tasks. Although the dual-arm planners were shown to maintain a constant distance between the arm tips, practical manipulation tasks require additional constraints on arm orientation and on intermediate poses so that a manipulated object is not dropped.

Finally, the dual-arm results suggest several longer-term directions. These include improved methods for state sampling with constraints, more effective local planners for tracing difficult shapes, and the extension from two manipulators to larger multi-arm continuum systems. More broadly, future work should continue to investigate how to take better advantage of the dexterity available to these models. The results of this dissertation suggest that such an approach is both promising and necessary for the continued advancement of continuum arm path planning.

Bibliography

- [1] B. H. Meng, I. S. Godage, and I. Kanj, “RRT*-based path planning for continuum arms,” *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6830–6837, 2022, Presented at IROS 2022 in Kyoto. DOI: [10.1109/LRA.2022.3174257](https://doi.org/10.1109/LRA.2022.3174257).
- [2] B. H. Meng, D. D. K. Arachchige, J. Deng, I. S. Godage, and I. Kanj, “Anticipatory path planning for continuum arms in dynamic environments,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 7815–7820. DOI: [10.1109/ICRA48506.2021.9560952](https://doi.org/10.1109/ICRA48506.2021.9560952).
- [3] B. H. Meng, I. S. Godage, and I. Kanj, “Smooth path planning for continuum arms,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 7809–7814. DOI: [10.1109/ICRA48506.2021.9560982](https://doi.org/10.1109/ICRA48506.2021.9560982).
- [4] B. H. Meng, D. D. K. Arachchige, I. S. Godage, and I. Kanj, “Path planning for continuum arms in dynamic environments,” in *2024 IEEE 7th International Conference on Soft Robotics (RoboSoft)*, 2024, pp. 900–905. DOI: [10.1109/RoboSoft60065.2024.10521950](https://doi.org/10.1109/RoboSoft60065.2024.10521950).
- [5] B. H. Meng, I. S. Godage, and I. Kanj, “Temporal joint-space planning for continuum arms in dynamic environments,” *IEEE Robotics and Automation Letters*, 2026.
- [6] J. Deng, B. H. Meng, I. Kanj, and I. S. Godage, “Near-optimal smooth path planning for multisection continuum arms,” *CoRR*, vol. abs/1812.03615, 2018. arXiv: [1812.03615](https://arxiv.org/abs/1812.03615). [Online]. Available: <http://arxiv.org/abs/1812.03615>.
- [7] I. S. Godage, G. Medrano-Cerda, D. Branson, E. Guglielmino, and D. Caldwell, “Modal kinematics for multisection continuum arms,” *Bioinspiration & biomimetics*, vol. 10, p. 035 002, Jun. 2015. DOI: [10.1088/1748-3190/10/3/035002](https://doi.org/10.1088/1748-3190/10/3/035002). [Online]. Available: <https://api.semanticscholar.org/CorpusID:25269974>.
- [8] I. S. Godage, G. A. Medrano-Cerda, D. T. Branson, E. Guglielmino, and D. G. Caldwell, “Dynamics for variable length multisection continuum arms,” *The International Journal of Robotics Research*, vol. 35, no. 6, pp. 695–722, 2016.
- [9] W. McMahan et al., “Field trials and testing of the octarm continuum manipulator,” in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, IEEE, 2006, pp. 2336–2341.
- [10] I. S. Godage, Y. Chen, and I. D. Walker, “Dynamic control of pneumatic muscle actuators,” *ArXiv*, vol. abs/1811.04991, 2018.
- [11] S. S. Epp, *Discrete Mathematics with Applications*, 4th. USA: Brooks/Cole Publishing Co., 2010.
- [12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd. The MIT Press, 2009.

- [13] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 1959. [Online]. Available: <https://api.semanticscholar.org/CorpusID:123284777>.
- [14] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. DOI: [10.1109/TSSC.1968.300136](https://doi.org/10.1109/TSSC.1968.300136).
- [15] R. Bellman, “On a routing problem,” *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958. Accessed: Mar. 6, 2024. [Online]. Available: <http://www.jstor.org/stable/43634538>.
- [16] D. R. Ford and D. R. Fulkerson, *Flows in Networks*. USA: Princeton University Press, 2010.
- [17] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, “Path problems in temporal graphs,” *Proceedings of the VLDB Endowment*, vol. 7, no. 9, pp. 721–732, 2014.
- [18] V. Kostakos, “Temporal graphs,” *Physica A: Statistical Mechanics and its Applications*, vol. 388, no. 6, pp. 1007–1023, 2009.
- [19] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996. DOI: [10.1109/70.508439](https://doi.org/10.1109/70.508439).
- [20] S. M. LaValle, “Rapidly-exploring random trees : A new tool for path planning,” *The annual research report*, 1998. [Online]. Available: <https://api.semanticscholar.org/CorpusID:14744621>.
- [21] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011. DOI: [10.1177/0278364911406761](https://doi.org/10.1177/0278364911406761). eprint: <https://doi.org/10.1177/0278364911406761>. [Online]. Available: <https://doi.org/10.1177/0278364911406761>.
- [22] G. Robinson and J. Davies, “Continuum robots - a state of the art,” in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 4, 1999, pp. 2849–2854. DOI: [10.1109/ROBOT.1999.774029](https://doi.org/10.1109/ROBOT.1999.774029).
- [23] M. Hannan and I. Walker, “The ‘elephant trunk’ manipulator, design and implementation,” in *2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics. Proceedings (Cat. No.01TH8556)*, vol. 1, 2001, pp. 14–19. DOI: [10.1109/AIM.2001.936423](https://doi.org/10.1109/AIM.2001.936423).
- [24] J. Burgner-Kahrs, D. C. Rucker, and H. Choset, “Continuum robots for medical applications: A survey,” *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1261–1280, 2015. DOI: [10.1109/TR0.2015.2489500](https://doi.org/10.1109/TR0.2015.2489500).
- [25] I. S. Godage, G. A. Medrano-Cerda, D. T. Branson, E. Guglielmino, and D. G. Caldwell, “Dynamics for variable length multisection continuum arms,” *The International Journal of Robotics Research*, vol. 35, no. 6, pp. 695–722, 2016. DOI: [10.1177/0278364915596450](https://doi.org/10.1177/0278364915596450). eprint: <https://doi.org/10.1177/0278364915596450>. [Online]. Available: <https://doi.org/10.1177/0278364915596450>.
- [26] I. Walker, “Continuous backbone “continuum” robot manipulators,” *ISRN Robotics*, vol. 2013, Jan. 2013. DOI: [10.5402/2013/726506](https://doi.org/10.5402/2013/726506).
- [27] K. Salisbury, W. Townsend, B. Ebrman, and D. DiPietro, “Preliminary design of a whole-arm manipulation system (wams),” in *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, 1988, pp. 254–260. DOI: [10.1109/ROBOT.1988.12057](https://doi.org/10.1109/ROBOT.1988.12057).

- [28] D. D. K. Arachchige, Y. Chen, and I. S. Godage, “Soft robotic snake locomotion: Modeling and experimental assessment,” in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, 2021, pp. 805–810. DOI: [10.1109/CASE49439.2021.9551398](https://doi.org/10.1109/CASE49439.2021.9551398).
- [29] M. C. Doerfler, K. Schaffer, and M. M. Coad, “Hybrid soft-rigid continuum robot inspired by spider monkey tail,” in *2023 IEEE International Conference on Soft Robotics (RoboSoft)*, 2023, pp. 1–6. DOI: [10.1109/RoboSoft55895.2023.10122106](https://doi.org/10.1109/RoboSoft55895.2023.10122106).
- [30] E. Guglielmino, N. Tsagarakis, and D. G. Caldwell, “An octopus anatomy-inspired robotic arm,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 3091–3096. DOI: [10.1109/IRoS.2010.5650361](https://doi.org/10.1109/IRoS.2010.5650361).
- [31] P. Rao, C. Pogue, Q. Peyron, E. Diller, and J. Burgner-Kahrs, “Modeling and analysis of tendon-driven continuum robots for rod-based locking,” *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3126–3133, 2023. DOI: [10.1109/LRA.2023.3264869](https://doi.org/10.1109/LRA.2023.3264869).
- [32] R. K. Katzschmann, C. D. Santina, Y. Tashimitsu, A. Bicchi, and D. Rus, “Dynamic motion control of multi-segment soft robots using piecewise constant curvature matched with an augmented rigid body model,” in *2019 2nd IEEE International Conference on Soft Robotics (RoboSoft)*, 2019, pp. 454–461. DOI: [10.1109/ROBOSOFT.2019.8722799](https://doi.org/10.1109/ROBOSOFT.2019.8722799).
- [33] A. D. Marchese, R. K. Katzschmann, and D. Rus, “Whole arm planning for a soft and highly compliant 2d robotic manipulator,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 554–560. DOI: [10.1109/IRoS.2014.6942614](https://doi.org/10.1109/IRoS.2014.6942614).
- [34] E. Amanov, T.-D. Nguyen, and J. Burgner-Kahrs, “Tendon-driven continuum robots with extensible sections—a model-based evaluation of path-following motions,” *The International Journal of Robotics Research*, vol. 40, pp. 7–23, 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:209905263>.
- [35] C. Pogue, P. Rao, Q. Peyron, J. Kim, J. Burgner-Kahrs, and E. Diller, “Multiple curvatures in a tendon-driven continuum robot using a novel magnetic locking mechanism,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 472–479. DOI: [10.1109/IRoS47612.2022.9982193](https://doi.org/10.1109/IRoS47612.2022.9982193).
- [36] H. B. Gilbert, J. Neimat, and R. J. Webster, “Concentric tube robots as steerable needles: Achieving follow-the-leader deployment,” *IEEE transactions on robotics : a publication of the IEEE Robotics and Automation Society*, vol. 31, no. 2, pp. 246–258, Apr. 2015. DOI: [10.1109/tro.2015.2394331](https://doi.org/10.1109/tro.2015.2394331).
- [37] C. Girerd and T. K. Morimoto, “Design and control of a hand-held concentric tube robot for minimally invasive surgery,” *IEEE Transactions on Robotics*, vol. 37, no. 4, pp. 1022–1038, 2021. DOI: [10.1109/TRo.2020.3043668](https://doi.org/10.1109/TRo.2020.3043668).
- [38] V. Falkenhahn, T. Mahl, A. Hildebrandt, R. Neumann, and O. Sawodny, “Dynamic modeling of constant curvature continuum robots using the euler-lagrange formalism,” *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2428–2433, 2014. [Online]. Available: <https://api.semanticscholar.org/CorpusID:17349595>.
- [39] A. B. Clark, V. Mathivannan, and N. Rojas, “A continuum manipulator for open-source surgical robotics research and shared development,” *IEEE Transactions on Medical Robotics and Bionics*, vol. 3, no. 1, pp. 277–280, 2021. DOI: [10.1109/TMRB.2020.3041391](https://doi.org/10.1109/TMRB.2020.3041391).
- [40] M. M. Tonapi, I. S. Godage, and I. D. Walker, “Design, modeling and performance evaluation of a long and slim continuum robotic cable,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2852–2859. DOI: [10.1109/IRoS.2014.6942954](https://doi.org/10.1109/IRoS.2014.6942954).

- [41] I. S. Godage, D. T. Branson, E. Guglielmino, and D. G. Caldwell, "Pneumatic muscle actuated continuum arms: Modelling and experimental assessment," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 4980–4985. DOI: [10.1109/ICRA.2012.6224949](https://doi.org/10.1109/ICRA.2012.6224949).
- [42] S. Satheeshbabu, N. K. Uppalapati, T. Fu, and G. Krishnan, "Continuous control of a soft continuum arm using deep reinforcement learning," *2020 3rd IEEE International Conference on Soft Robotics (RoboSoft)*, pp. 497–503, 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:219858416>.
- [43] Y. Toshimitsu, K. W. Wong, T. Buchner, and R. Katzschmann, "Sopra: Dynamical modeling of a scalable soft continuum robotic arm with integrated proprioceptive sensing," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Sep. 2021. DOI: [10.1109/iros51168.2021.9636539](https://doi.org/10.1109/iros51168.2021.9636539). [Online]. Available: <http://dx.doi.org/10.1109/IROS51168.2021.9636539>.
- [44] Z. Li and R. Du, "Design and analysis of a bio-inspired wire-driven multi-section flexible robot," *International Journal of Advanced Robotic Systems*, vol. 10, no. 4, p. 209, 2013. DOI: [10.5772/56025](https://doi.org/10.5772/56025). eprint: <https://doi.org/10.5772/56025>. [Online]. Available: <https://doi.org/10.5772/56025>.
- [45] W. McMahan, B. Jones, and I. Walker, "Design and implementation of a multi-section continuum robot: Air-octor," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 2578–2585. DOI: [10.1109/IROS.2005.1545487](https://doi.org/10.1109/IROS.2005.1545487).
- [46] M. Russo et al., "Continuum robots: An overview," *Advanced Intelligent Systems*, vol. 5, no. 5, p. 2200367, 2023. DOI: <https://doi.org/10.1002/aisy.202200367>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/aisy.202200367>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202200367>.
- [47] H. B. Gilbert, D. C. Rucker, and R. J. W. III, "Concentric tube robots: The state of the art and future directions," in *International Symposium of Robotics Research*, 2013. [Online]. Available: <https://api.semanticscholar.org/CorpusID:17231879>.
- [48] J. Burgner-Kahrs, P. J. Swaney, R. A. Lathrop, K. D. Weaver, and R. J. Webster, "Debulking from within: A robotic steerable cannula for intracerebral hemorrhage evacuation," *IEEE Transactions on Biomedical Engineering*, vol. 60, pp. 2567–2575, 2013. [Online]. Available: <https://api.semanticscholar.org/CorpusID:17721463>.
- [49] K. Weaver et al., "The use of teleoperated concentric tube robots for transsphenoidal parasellar surgery," *Journal of Neurological Surgery Part B: Skull Base*, vol. 74, Mar. 2013. DOI: [10.1055/s-0033-1336249](https://doi.org/10.1055/s-0033-1336249).
- [50] M. M. Tonapi, I. S. Godage, and I. D. Walker, "Next generation rope-like robot for in-space inspection," in *2014 IEEE Aerospace Conference*, 2014, pp. 1–13. DOI: [10.1109/AERO.2014.6836183](https://doi.org/10.1109/AERO.2014.6836183).
- [51] J. Mehling, M. Diftler, M. Chu, and M. Valvo, "A minimally invasive tendril robot for in-space inspection," in *The First IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics, 2006. BioRob 2006.*, 2006, pp. 690–695. DOI: [10.1109/BIOROB.2006.1639170](https://doi.org/10.1109/BIOROB.2006.1639170).

- [52] C. Sun, L. Chen, J. Liu, J. S. Dai, and R. Kang, "A hybrid continuum robot based on pneumatic muscles with embedded elastic rods," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 234, no. 1, pp. 318–328, 2020. DOI: [10.1177/0954406218822013](https://doi.org/10.1177/0954406218822013). eprint: <https://doi.org/10.1177/0954406218822013>. [Online]. Available: <https://doi.org/10.1177/0954406218822013>.
- [53] J. Colgate, W. Wannasuphoprasit, and M. Peshkin, "Cobots: Robots for collaboration with human operators," English (US), Proceedings of the 1996 ASME International Mechanical Engineering Congress and Exposition ; Conference date: 17-11-1996 Through 22-11-1996, Dec. 1996, pp. 433–439.
- [54] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach," *The International Journal of Robotics Research*, vol. 10, no. 6, pp. 628–649, 1991. DOI: [10.1177/027836499101000604](https://doi.org/10.1177/027836499101000604). eprint: <https://doi.org/10.1177/027836499101000604>. [Online]. Available: <https://doi.org/10.1177/027836499101000604>.
- [55] B. Glavina, "Solving findpath by combination of goal-directed and randomized search," in *Proceedings., IEEE International Conference on Robotics and Automation*, 1990, 1718–1723 vol.3. DOI: [10.1109/ROBOT.1990.126257](https://doi.org/10.1109/ROBOT.1990.126257).
- [56] J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg, "Real-time robot motion planning using rasterizing computer graphics hardware," *SIGGRAPH Comput. Graph.*, vol. 24, no. 4, pp. 327–335, Sep. 1990. DOI: [10.1145/97880.97915](https://doi.org/10.1145/97880.97915). [Online]. Available: <https://doi.org/10.1145/97880.97915>.
- [57] K. Kondo, "Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 267–277, 1991. DOI: [10.1109/70.88136](https://doi.org/10.1109/70.88136).
- [58] L. . and J. CLatombe, "Randomized preprocessing of configuration space for fast path planning," in *IEEE International Conference on Robotics and Automation, San Diego (c SA)*, 1994.
- [59] L. Kavraki and J.-C. Latombe, "Randomized preprocessing of configuration space for path planning: Articulated robots," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, vol. 3, 1994, 1764–1771 vol.3. DOI: [10.1109/IROS.1994.407619](https://doi.org/10.1109/IROS.1994.407619).
- [60] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 1, 1999, 473–479 vol.1. DOI: [10.1109/ROBOT.1999.770022](https://doi.org/10.1109/ROBOT.1999.770022).
- [61] K. Goldberg, "Completeness in robot motion planning," in *Proceedings of the Workshop on Algorithmic Foundations of Robotics*, ser. WAFR, San Francisco, California, USA: A. K. Peters, Ltd., 1995, pp. 419–429.
- [62] L. Zhang, Y. J. Kim, and D. Manocha, "A hybrid approach for complete motion planning," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 7–14. DOI: [10.1109/IROS.2007.4399064](https://doi.org/10.1109/IROS.2007.4399064).
- [63] R. Bohlin and L. Kavraki, "Path planning using lazy prm," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, 2000, 521–528 vol.1. DOI: [10.1109/ROBOT.2000.844107](https://doi.org/10.1109/ROBOT.2000.844107).
- [64] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, 995–1001 vol.2. DOI: [10.1109/ROBOT.2000.844730](https://doi.org/10.1109/ROBOT.2000.844730).

- [65] S. Karaman and E. Frazzoli, *Incremental sampling-based algorithms for optimal motion planning*, 2010. arXiv: [1005.0416 \[cs.R0\]](#).
- [66] T.-Y. Li and Y.-C. Shie, “An incremental learning approach to motion planning with roadmap management,” in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 4, 2002, 3411–3416 vol.4. DOI: [10.1109/ROBOT.2002.1014238](#).
- [67] K. Hauser, “Lazy collision checking in asymptotically-optimal motion planning,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 2951–2957. DOI: [10.1109/ICRA.2015.7139603](#).
- [68] I. A. Şucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, Dec. 2012, <https://ompl.kavrakilab.org>. DOI: [10.1109/MRA.2012.2205651](#).
- [69] O. Arslan and P. Tsiotras, “Use of relaxation methods in sampling-based algorithms for optimal motion planning,” in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 2421–2428. DOI: [10.1109/ICRA.2013.6630906](#).
- [70] M. Otte and E. Frazzoli, “Rrtx: Asymptotically optimal single-query sampling-based motion planning with quick replanning,” *The International Journal of Robotics Research*, vol. 35, no. 7, pp. 797–822, 2016. DOI: [10.1177/0278364915594679](#). eprint: <https://doi.org/10.1177/0278364915594679>. [Online]. Available: <https://doi.org/10.1177/0278364915594679>.
- [71] A. Shkolnik and R. Tedrake, “Path planning in 1000+ dimensions using a task-space voronoi bias,” in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 2061–2067. DOI: [10.1109/ROBOT.2009.5152638](#).
- [72] H. Liu, X. Zhang, J. Wen, R. Wang, and X. Chen, “Goal-biased bidirectional rrt based on curve-smoothing,” *IFAC-PapersOnLine*, vol. 52, no. 24, pp. 255–260, 2019, 5th IFAC Symposium on Telematics Applications TA 2019. DOI: <https://doi.org/10.1016/j.ifacol.2019.12.417>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2405896319323250>.
- [73] P. Xin, X. Wang, X. Liu, Y. Wang, Z. Zhai, and X. Ma, “Improved bidirectional rrt* algorithm for robot path planning,” *Sensors*, vol. 23, no. 2, 2023. DOI: [10.3390/s23021041](#). [Online]. Available: <https://www.mdpi.com/1424-8220/23/2/1041>.
- [74] J. Deng, Z. Li, Y. Zheng, and G. Gu, “An rrt-based motion planning method for hyper-redundant manipulators in confined spaces,” in *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2022, pp. 1067–1073. DOI: [10.1109/ROBIO55434.2022.10011650](#).
- [75] J. Chen, J. Yan, Y. Qiu, H. Fang, J. Chen, and S. S. Cheng, “A cross-entropy motion planning framework for hybrid continuum robots,” *IEEE Robotics and Automation Letters*, vol. 8, no. 12, pp. 8200–8207, 2023. DOI: [10.1109/LRA.2023.3325777](#).
- [76] P. Luo, S. Yao, Y. Yue, J. Wang, H. Yan, and M. Q. -. Meng, *Efficient rrt*-based safety-constrained motion planning for continuum robots in dynamic environments*, 2023. arXiv: [2309.13813 \[cs.R0\]](#).
- [77] K. Leibrandt, C. Bergeles, and G.-Z. Yang, “Concentric tube robots: Rapid, stable path-planning and guidance for surgical use,” *IEEE Robotics & Automation Magazine*, vol. 24, no. 2, pp. 42–53, 2017. DOI: [10.1109/MRA.2017.2680546](#).

- [78] G. Chen, N. Luo, D. Liu, Z. Zhao, and C. Liang, “Path planning for manipulators based on an improved probabilistic roadmap method,” *Robotics and Computer-Integrated Manufacturing*, vol. 72, p. 102196, 2021. DOI: <https://doi.org/10.1016/j.rcim.2021.102196>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S073658452100079X>.
- [79] J. Bruce and M. Veloso, “Real-time randomized path planning for robot navigation,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, 2002, 2383–2388 vol.3. DOI: [10.1109/IRDS.2002.1041624](https://doi.org/10.1109/IRDS.2002.1041624).
- [80] D. Ferguson, N. Kalra, and A. Stentz, “Replanning with rrt,” in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, 2006, pp. 1243–1248. DOI: [10.1109/ROBOT.2006.1641879](https://doi.org/10.1109/ROBOT.2006.1641879).
- [81] M. Zucker, J. Kuffner, and M. Branicky, “Multipartite rrt for rapid replanning in dynamic environments,” in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 1603–1609. DOI: [10.1109/ROBOT.2007.363553](https://doi.org/10.1109/ROBOT.2007.363553).
- [82] R. Gayle, K. R. Klingler, and P. G. Xavier, “Lazy reconfiguration forest (lrf) - an approach for motion planning with multiple tasks in dynamic environments,” in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 1316–1323. DOI: [10.1109/ROBOT.2007.363167](https://doi.org/10.1109/ROBOT.2007.363167).
- [83] X. Huang, G. Soti, H. Zhou, C. Ledermann, B. Hein, and T. Kröger, “Hiro: Heuristics informed robot online path planning using pre-computed deterministic roadmaps,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 8109–8116. DOI: [10.1109/IROS47612.2022.9981740](https://doi.org/10.1109/IROS47612.2022.9981740).
- [84] T. Lai and F. Ramos, “Ltr*: Rapid replanning in executing consecutive tasks with lazy experience graph,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 8784–8790. DOI: [10.1109/IROS47612.2022.9982237](https://doi.org/10.1109/IROS47612.2022.9982237).
- [85] L. Jaillet and T. Simeon, “A prm-based motion planner for dynamically changing environments,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 2, 2004, 1606–1611 vol.2. DOI: [10.1109/IROS.2004.1389625](https://doi.org/10.1109/IROS.2004.1389625).
- [86] M. Hüppi, L. Bartolomei, R. Mascaro, and M. Chli, “T-prm: Temporal probabilistic roadmap for path planning in dynamic environments,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 10320–10327. DOI: [10.1109/IROS47612.2022.9981739](https://doi.org/10.1109/IROS47612.2022.9981739).
- [87] C. I. Vasile and C. Belta, *Sampling-based temporal logic path planning*, 2013. arXiv: [1307.7263 \[cs.R0\]](https://arxiv.org/abs/1307.7263).
- [88] A. Linard, I. Torre, E. Bartoli, A. Sleat, I. Leite, and J. Tumova, “Real-time rrt* with signal temporal logic preferences,” in *2023 IEEE/RSJ international conference on intelligent robots and systems (IROS)* :, QC 20231122, 2023. [Online]. Available: <https://ieee-iros.org/>.
- [89] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, 1985, pp. 500–505. DOI: [10.1109/ROBOT.1985.1087247](https://doi.org/10.1109/ROBOT.1985.1087247).
- [90] Y. Tian, X. Zhu, D. Meng, X. Wang, and B. Liang, “An overall configuration planning method of continuum hyper-redundant manipulators based on improved artificial potential field method,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4867–4874, 2021. DOI: [10.1109/LRA.2021.3067310](https://doi.org/10.1109/LRA.2021.3067310).

- [91] G. Robinson and J. Davies, "Continuum robots - a state of the art," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 4, 1999, pp. 2849–2854 vol.4. DOI: [10.1109/ROBOT.1999.774029](https://doi.org/10.1109/ROBOT.1999.774029).
- [92] S. M. LaValle, *Planning Algorithms*. USA: Cambridge University Press, 2006.
- [93] E. S. Conkur, "Path planning using potential fields for highly redundant manipulators," *Robotics and Autonomous Systems*, vol. 52, no. 2, pp. 209–228, 2005. DOI: <https://doi.org/10.1016/j.robot.2005.03.005>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0921889005000552>.
- [94] I. Soodmand, M. Kebbach, S. Herrmann, R. Bader, and C. Woernle, "An artificial potential field algorithm for path planning of redundant manipulators based on navigation functions," in *Advances in Robot Kinematics 2022*, O. Altuzarra and A. Kecskeméthy, Eds., Cham: Springer International Publishing, 2022, pp. 470–477.
- [95] J. Kim, W. Jie, H. Kim, and M. C. Lee, "Modified configuration control with potential field for inverse kinematic solution of redundant manipulator," *IEEE/ASME Transactions on Mechatronics*, vol. 26, no. 4, pp. 1782–1790, 2021. DOI: [10.1109/TMECH.2021.3077914](https://doi.org/10.1109/TMECH.2021.3077914).
- [96] W. Wang, J. Gu, M. Zhu, Q. Huo, S. He, and Z. Xu, "An obstacle avoidance method for redundant manipulators based on artificial potential field," in *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2018, pp. 2151–2156. DOI: [10.1109/ICMA.2018.8484459](https://doi.org/10.1109/ICMA.2018.8484459).
- [97] S.-O. Park, M. C. Lee, and J. Kim, "Trajectory planning with collision avoidance for redundant robots using jacobian and artificial potential field-based real-time inverse kinematics," *International Journal of Control, Automation and Systems*, vol. 18, Apr. 2020. DOI: [10.1007/s12555-019-0076-7](https://doi.org/10.1007/s12555-019-0076-7).
- [98] W. Wang, M. Zhu, X. Wang, S. He, J. He, and Z. Xu, "An improved artificial potential field method of trajectory planning and obstacle avoidance for redundant manipulators," *International Journal of Advanced Robotic Systems*, vol. 15, no. 5, p. 1 729 881 418 799 562, 2018. DOI: [10.1177/1729881418799562](https://doi.org/10.1177/1729881418799562). eprint: <https://doi.org/10.1177/1729881418799562>. [Online]. Available: <https://doi.org/10.1177/1729881418799562>.
- [99] A. Sepehri and A. M. Moghaddam, "A motion planning algorithm for redundant manipulators using rapidly exploring randomized trees and artificial potential fields," *IEEE Access*, vol. 9, pp. 26 059–26 070, 2021. DOI: [10.1109/ACCESS.2021.3056397](https://doi.org/10.1109/ACCESS.2021.3056397).
- [100] J. Dai, Y. Zhang, and H. Deng, "Novel potential guided bidirectional rrt* with direct connection strategy for path planning of redundant robot manipulators in joint space," *IEEE Transactions on Industrial Electronics*, vol. 71, no. 3, pp. 2737–2747, 2024. DOI: [10.1109/TIE.2023.3269462](https://doi.org/10.1109/TIE.2023.3269462).
- [101] X. Tang and F. Chen, "Robot path planning algorithm based on bi-rrt and potential field," in *2020 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2020, pp. 1251–1256. DOI: [10.1109/ICMA49215.2020.9233539](https://doi.org/10.1109/ICMA49215.2020.9233539).
- [102] J. Park, H.-S. Kim, and J.-B. Song, "Collision-free path planning for a redundant manipulator based on prm and potential field methods," *Journal of Institute of Control, Robotics and Systems*, vol. 17, Apr. 2011. DOI: [10.5302/J.ICROS.2011.17.4.362](https://doi.org/10.5302/J.ICROS.2011.17.4.362).
- [103] I. S. Godage, D. T. Branson, E. Guglielmino, and D. G. Caldwell, "Path planning for multisection continuum arms," in *2012 IEEE International Conference on Mechatronics and Automation*, IEEE, 2012, pp. 1208–1213.

- [104] A. Ataka, P. Qi, H. Liu, and K. Althoefer, “Real-time planner for multi-segment continuum manipulator in dynamic environments,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 4080–4085.
- [105] A. Ataka, P. Qi, H. Liu, and K. Althoefer, “Real-time planner for multi-segment continuum manipulator in dynamic environments,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 4080–4085. DOI: [10.1109/ICRA.2016.7487598](https://doi.org/10.1109/ICRA.2016.7487598).
- [106] T. Greigarn, N. L. Poirot, X. Xu, and M. C. Çavuşoğlu, “Jacobian-based task-space motion planning for mri-actuated continuum robots,” *IEEE Robotics and Automation Letters*, vol. 4, no. 1, pp. 145–152, 2019. DOI: [10.1109/LRA.2018.2881987](https://doi.org/10.1109/LRA.2018.2881987).
- [107] J. Zhang, K. Lu, J. Yuan, J. di, and G. He, “Kinematics modeling and motion planning of tendon driven continuum manipulators,” *Journal of Artificial Intelligence and Technology*, vol. 1, Dec. 2020. DOI: [10.37965/jait.2020.0041](https://doi.org/10.37965/jait.2020.0041).
- [108] J. Xiao and R. Vatcha, “Real-time adaptive motion planning for a continuum manipulator,” in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, IEEE, 2010, pp. 5919–5926.
- [109] N. Developers, “Numpy,” *NumPy Numpy. Scipy Developers*, 2013.
- [110] S. K. Lam, A. Pitrou, and S. Seibert, “Numba: A llvm-based python jit compiler,” in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, ACM, 2015, p. 7.
- [111] G. Csárdi and T. Nepusz, “Igraph reference manual,” URL: <http://igraph.sourceforge.net/documentation.html> (accessed April, vol. 20, 2010).
- [112] F. Lekien and J. Marsden, “Tricubic interpolation in three dimensions,” *Journal of Numerical Methods and Engineering*, vol. 63, pp. 455–471, 2005.
- [113] E. Maeland, “On the comparison of interpolation methods,” *IEEE Transactions on Medical Imaging*, vol. 7, no. 3, pp. 213–217, 1988.
- [114] G. B. Dantzig, “On the shortest route through a network,” *Management Science*, vol. 6, no. 2, pp. 187–190, 1960.
- [115] Y. K. Hwang, N. Ahuja, et al., “A potential field approach to path planning,” *IEEE Transactions on Robotics and Automation*, vol. 8, no. 1, pp. 23–32, 1992.
- [116] E. Jones, T. Oliphant, P. Peterson, et al., *SciPy: Open source scientific tools for Python*, 2001. [Online]. Available: <http://www.scipy.org/>.
- [117] I. C. Jeong and J. Finkelstein, “Introducing contactless blood pressure assessment using a high speed video camera,” *Journal of medical systems*, vol. 40, no. 4, p. 77, 2016.
- [118] M. Vande Weghe, D. Ferguson, and S. S. Srinivasa, “Randomized path planning for redundant manipulators without inverse kinematics,” in *2007 7th IEEE-RAS International Conference on Humanoid Robots*, 2007, pp. 477–482. DOI: [10.1109/ICHR.2007.4813913](https://doi.org/10.1109/ICHR.2007.4813913).
- [119] J. Wan, J. Yao, L. Zhang, and H. Wu, “A weighted gradient projection method for inverse kinematics of redundant manipulators considering multiple performance criteria,” *Strojniški vestnik - Journal of Mechanical Engineering*, vol. 64, no. 7-8, pp. 475–487, 2018. DOI: [10.5545/sv-jme.2017.5182](https://doi.org/10.5545/sv-jme.2017.5182).
- [120] J. Wang et al., “Learning to guide online multi-contact receding horizon planning,” in *IEEE/RSJ Int. Conf. on Intel. Rob. & Sys. (IROS)*, 2022, pp. 12 942–12 949.

BIBLIOGRAPHY

- [121] J. Leu, M. Wang, and M. Tomizuka, “Long-horizon motion planning via sampling and segmented trajectory optimization,” in *European Control Conference (ECC)*, 2022, pp. 538–545. DOI: [10.23919/ECC55457.2022.9838237](https://doi.org/10.23919/ECC55457.2022.9838237).
- [122] D. D. Arachchige and I. S. Godage, “Hybrid soft robots incorporating soft and stiff elements,” in *IEEE Int. Conf. on S. R. (RoboSoft)*, 2022.
- [123] D. D. Arachchige et al., “Soft steps: Exploring quadrupedal locomotion with modular soft robots,” *IEEE Access*, 2023.