

UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Learning-Based Robot Configuration Space Modeling

Permalink

<https://escholarship.org/uc/item/2b50w2pp>

Author

Das, Nikhil

Publication Date

2020

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA SAN DIEGO

Learning-Based Robot Configuration Space Modeling

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering (Signal and Image Processing)

by

Nikhil Das

Committee in charge:

Professor Michael C. Yip, Chair
Professor Nikolay Atanasov
Professor Ken Kreutz-Delgado
Professor Sonia Martinez
Professor Ryan Orosco

2020

Copyright
Nikhil Das, 2020
All rights reserved.

The dissertation of Nikhil Das is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California San Diego

2020

DEDICATION

I dedicate this work to my family:

to my little sister for being a terrific Etsy business partner, for baking me
delicious cookies and brownies, and for always checking up on me,
to my older sister for giving me a place to stay during Bay Area trips, consulting
with me on fonts and color schemes of my papers and presentations, and
providing survival tips for graduate school and the academic field,
to my father for introducing me to the world of science, math, and technology
and for making me feel proud of my skill set,
and to my mother for making me coffee to compensate for my procrastination,
for joining me on weekly trips to Aldi, for ensuring I take care of myself, and for
always believing in my ability to accomplish what I thought I could not.

TABLE OF CONTENTS

Signature Page	iii
Dedication	iv
Table of Contents	v
List of Figures	vii
List of Tables	x
Acknowledgements	xi
Vita	xii
Abstract of the Dissertation	xiii
Chapter 1	
Introduction	1
1.1 Robot Manipulators	1
1.1.1 Kinematics	2
1.1.2 Configuration Space	4
1.2 Motion Planning	5
1.2.1 Potential Field Methods	7
1.2.2 Sampling-Based Motion Planning	7
1.3 Collision Detection	9
1.4 Machine Learning	10
1.4.1 Discriminative Classification	11
1.4.2 Kernel Functions	13
Chapter 2	
Modeling Configuration Space	17
2.1 Approaches to Approximate C-space	18
2.1.1 Grid-Based	18
2.1.2 k-Nearest Neighbors Models	19
2.1.3 Gaussian Mixture Models	20
2.1.4 Support Vector Machines	20
2.1.5 Neural Networks	21
2.2 Fastron	22
2.2.1 Model Update	23
2.2.2 Conditional Bias Parameter	30
2.2.3 Redundant Support Point Removal	32
2.2.4 Active Learning	33
2.2.5 Practical Considerations	34
2.3 Performance of Fastron	36

	2.3.1	Performance in Static Environments	37
	2.3.2	Performance in Changing Environments	44
	2.3.3	Performance in Motion Planning Application	48
	2.4	Further Applications	52
	2.4.1	Pick and Place	52
	2.4.2	Autonomous Surgical Assistant	53
	2.4.3	Optimal Surgical Arm Placement	54
	2.5	Acknowledgments	59
Chapter 3		Forward Kinematics Kernel	60
	3.1	Kernel Functions	61
	3.2	Forward Kinematics Kernel Definition	61
	3.3	Positive Definiteness	65
	3.4	Performance Using Forward Kinematics Kernel	67
	3.4.1	Collision Checking	67
	3.4.2	Motion Planning	72
	3.5	Further Applications	74
	3.5.1	Kernel Principal Component Analysis	74
	3.5.2	Import Vector Machine	78
	3.6	Acknowledgments	81
Chapter 4		Estimating Distance to Collision	82
	4.1	Approaches to Model Distances	83
	4.1.1	Geometric Approaches	84
	4.1.2	Perception-Based Distance Fields	84
	4.1.3	Support Vector Machines	85
	4.1.4	Neural Networks	86
	4.2	Gaussian Process Regression for Distance Estimation	86
	4.3	Confidence-Based Hybrid Model	88
	4.4	Performance of Gaussian Processes	90
	4.4.1	Accuracy and Query Timing	91
	4.4.2	Path Optimization with Clearance Constraint	93
	4.4.3	Path Optimization with Clearance Maximization	96
	4.4.4	Confidence-Based Hybrid Model	97
	4.5	Acknowledgments	99
Chapter 5		Conclusion	100
Appendix A		Additional Work	102
	A.1	GPU Acceleration	102
	A.2	Potential Fields in C-space	104
Bibliography			107

LIST OF FIGURES

Figure 1.1:	A 3 DOF robot and multiple cube obstacles. A C-space representation of the 3 DOF robot is also shown where each surface represents the boundary between C_{obs} and C_{free}	2
Figure 1.2:	A 2 DOF robot and a square obstacle. A toroidal representation of the C-space along with the C_{obs} and C_{free} is shown. A C-space representation where the minimum workspace distance for each configuration is also shown.	3
Figure 1.3:	A motion plan for a 2 DOF robot found using an RRT [1] is represented in both the workspace and C-space.	6
Figure 1.4:	Examples of radial basis function kernels, including the Gaussian kernel function $k_G(\cdot, \cdot)$ and the 2^{nd} and 8^{th} order rational quadratic kernel functions $k_{RQ}(\cdot, \cdot)$	13
Figure 2.1:	Block diagram illustrating the pipeline of the Fastron algorithm.	22
Figure 2.2:	Fastron C-space model of C_{obs} and C_{free} with support points. Workspace representations of C_{free} and C_{obs} support points.	23
Figure 2.3:	TPR and TNR of Fastron for various conditional bias β values for a 2 and 4 DOF robot.	30
Figure 2.4:	Example environments with up to 4 cube obstacles and simple 2 and 4 DOF robots. There are half as many links as there are joints, and each pair of joints is overlapping.	31
Figure 2.5:	TPR, TNR, and model sizes of Fastron for various γ values for a 2 and 4 DOF robot.	40
Figure 2.6:	Accuracy, TPR, and TNR for Fastron, ISVM [2] with active learning [3], and SSVM [4] in a static environment.	41
Figure 2.7:	Example static environments and C-space representations using GJK [5], Fastron [6], incremental support vector machines [2, 3], and sparse support vector machines [4].	43
Figure 2.8:	Baxter robot with cube obstacles used for collision detection and motion planning experiments.	45
Figure 2.9:	Accuracy, TPR, and TNR for Fastron and GJK [5] in a continuously changing environment using the Baxter's right arm and one workspace obstacle as shown in Fig. 2.8.	47
Figure 2.10:	Timings for motion planning for the Baxter robot's right arm with one workspace obstacle using RRT, RRT-Connect, RRT*, and BIT* using Fastron, GJK, and FCL for collision detection.	50
Figure 2.11:	Timings for motion planning for the Baxter robot's 7 DOF right arm with three workspace obstacles using RRT, RRT-Connect, RRT*, and BIT* using Fastron, GJK, and FCL for collision detection.	51

Figure 2.12:	Pick and place task using a UR3 in simulation is shown. An autonomous assistant reaching a target among human-controlled manipulators in da Vinci® simulation is also shown. Optimized placement of da Vinci® PSM base positions is shown.	52
Figure 3.1:	A workspace representation of three configurations of a two-link arm. Examples of the contours of an RBF kernel and an FK kernel are displayed. . . .	62
Figure 3.2:	Four unique configurations of a two-link arm are shown. There are two control points selected along the arm, marked by the square and hollow circle. All four configurations are uniquely identifiable using both control points. .	66
Figure 3.3:	A comparison of the correctness of Fastron FK [7], Fastron RBF [6], and GJK [5] with a single, randomly-sized moving obstacle. Fastron FK has much better accuracy, TPR, and TNR than Fastron RBF and performs similarly to GJK.	69
Figure 3.4:	Typical failure cases using the RBF kernel and the FK kernel, where the configuration is mistakenly labeled as C_{free} but is truly C_{obs} . With the RBF kernel, the robot blatantly intersects with the workspace obstacle. With the FK kernel, the robot slightly intersects with the obstacle.	70
Figure 3.5:	Query times for Fastron FK [7], Fastron RBF [6], GJK [5], and FCL [8] against the number of workspace obstacles.	71
Figure 3.6:	Motion planning results using various collision detection methods: Fastron with FK kernel [7], Fastron with rational quadratic kernel [6], GJK [5], and FCL [8].	73
Figure 3.7:	A 4 DOF planar robot in the workspace is shown. KPCA projections onto first two components using the FK kernel, the Gaussian kernel, and the rational quadratic kernel are also shown.	76
Figure 3.8:	A Baxter robot in the workspace is shown. KPCA projections onto the first three components using the FK kernel, the Gaussian kernel, and the rational quadratic kernel are also shown.	77
Figure 3.9:	Import vector machine [9] estimating the conditional probability $P(\mathbf{y} = 1 \mathbf{x})$ using the rational quadratic kernel and the FK kernel.	79
Figure 4.1:	Different levels of confidence of the GP models for $d_{\mathcal{R}}(\cdot)$ being greater than 0 for a 2 DOF robot are shown.	89
Figure 4.2:	Histogram of absolute errors in distance estimation of $d_{\mathcal{R}}(\cdot)$ when using KR and GP models with the Gaussian and FK kernel. Models with a higher proportion of lower error are better.	93
Figure 4.3:	Examples of end effector paths before and after optimization when using GJK, GP with Gaussian kernel, GP with FK kernel, and KR for distance estimation.	95
Figure 4.4:	Examples of end effector paths before and after optimization when using GJK, GP with Gaussian kernel, GP with FK kernel, and KR for distance estimation.	97

Figure 4.5:	Optimized path and distances when using the proposed hybrid model. . . .	98
Figure A.1:	The timings of performing multiple collision checks using FCL [8] on a CPU, Fastron [6] on a CPU, and Fastron on a GPU are shown. Motion planning timings with each of these methods are also shown.	103
Figure A.2:	Streamline plots of the gradient of a Fastron model and of the gradient of a polyharmonic spline are shown.	105

LIST OF TABLES

Table 2.1:	Performance of Fastron in a static environment for the 2 DOF and 4 DOF robots shown in Fig. 2.4 compared against incremental SVM (ISVM) [2, 3], sparse SVM (SSVM) [4], and the ground truth collision detection method GJK [5].	41
Table 2.2:	Performance in a changing environment when actuating the first 4 DOF, the first 6 DOF, and all 7 DOF of the Baxter robot’s right arm, respectively, shown in Fig. 2.8.	46
Table 2.3:	Performance in a changing environment with three moving obstacles when actuating all 7 DOF of the Baxter robot’s right arm, respectively, shown in Fig. 2.8.	48
Table 3.1:	Performance of Fastron FK and Fastron RBF compared against geometric collision detection methods (GJK and FCL).	68
Table 4.1:	Performance in terms of MSE and query time for noise-free GJK/EPA, KR, GP (Gaussian), and GP (FK).	92
Table 4.2:	Average optimization time, end effector path length, and minimum satisfaction of clearance constraint after optimization of Eq. 4.8 (clearance constraint) when using GJK/EPA [5, 10], GJK/EPA with noise, GP (Gaussian kernel) [11], GP (FK kernel) [11, 7], and KR [12, 13] to estimate $d_{\mathcal{R}}(\cdot)$	94
Table 4.3:	Results for optimal paths using GJK [5, 10], GP (Gaussian kernel) [11], GP (FK kernel) [11, 7], KR [12, 13] to calculate $d_{\mathcal{R}}(\cdot)$ are presented. We include the average times to achieve the optimized plan, the average final cost, and the average worst distance to collision in each optimized path.	96

ACKNOWLEDGEMENTS

I would like to thank Dr. Michael Yip for accepting me as his first PhD student, providing continuous support, promoting my work, and presenting ample opportunity to broaden my skill set.

I appreciate the helpfulness of my doctoral committee, Dr. Nikolay Atanasov, Dr. Ken Kreutz-Delgado, Dr. Sonia Martinez, and Dr. Ryan Orosco.

I would also like to thank ARClab members and other members of the Contextual Robotics Institute with whom I have had the pleasure of working, including (in alphabetical order), Ahmed Qureshi, Andrew Saad, Dimitri Schreiber, Dr. Fei Liu, Florian Richter, Jacob Johnson, James Di, Dr. Jun Zhang, Mayur Bency, Mingwei Xu, Mrinal Verghese, Naman Gupta, Dr. Ryan Orosco, Thai Duong, Dr. Tokio Matsuzaki, Yang Li, and Zhixian Ye for useful and engaging academic discussion.

Chapter 2, in part, is a reprint of material from N. Das, M. Yip, “Learning-Based Proxy Collision Detection for Robot Motion Planning Applications,” in *IEEE Transactions on Robotics*, 2020. The dissertation author is the primary author of this paper.

Chapter 3, in part, is a reprint of material from N. Das, M. C. Yip, “Forward Kinematics Kernel for Improved Proxy Collision Checking,” in *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2349-2356, April 2020. The dissertation author is the primary author of this paper.

Chapter 4, in part, is currently being prepared for submission by Nikhil Das and Michael C. Yip for publication of the material. The dissertation author is the primary author of this paper.

VITA

2014	B.S. in Electrical Engineering, University of California San Diego
2016	M.S. in Electrical Engineering, University of California San Diego
2020	Ph.D. in Electrical Engineering, University of California San Diego

PUBLICATIONS

N. Das, M. C. Yip, “Forward Kinematics Kernel for Improved Proxy Collision Checking,” in *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2349-2356, April 2020.

N. Das, M. Yip, “Learning-Based Proxy Collision Detection for Robot Motion Planning Applications,” in *IEEE Transactions on Robotics*, 2020.

T. Duong, **N. Das**, M. Yip, N. Atanasov, “Autonomous Navigation in Unknown Environments using Sparse Kernel-based Occupancy Mapping”, in *IEEE International Conference on Robotics and Automation*, 2020.

R. Orosco, B. Lurie, T. Matsuzaki, E. Funk, V. Divi, C. Holsinger, S. Hong, F. Richter, **N. Das**, M. Yip, “Compensatory Motion Scaling for Time Delayed Robotic Surgery”, in *Surgical Endoscopy*, 2020.

N. Das, N. Gupta, M. Yip, “Fastron: An Online Learning-Based Model and Active Learning Strategy for Proxy Collision Detection”, in *Conference on Robot Learning*. November 13-15, Mountain View, California, in *Proceedings of Machine Learning Research (PMLR)*, vol. 78, pp. 496-504, 2017.

N. Das, N. Gupta, and M. Yip, “Fastron: A Learning-Based Configuration Space Model for Rapid Collision Detection for Gross Motion Planning in Changing Environments”, *Robotics: Science and Systems, in the workshop: (Empirically) Data-Driven Manipulation*, 2017.

M. Yip, **N. Das**, “Robot Autonomy for Surgery”. *Encyclopedia of Medical Robotics*, World Scientific Publishing, 2017.

ABSTRACT OF THE DISSERTATION

Learning-Based Robot Configuration Space Modeling

by

Nikhil Das

Doctor of Philosophy in Electrical Engineering (Signal and Image Processing)

University of California San Diego, 2020

Professor Michael C. Yip, Chair

Numerous robotics applications, such as path planning, trajectory optimization, and optimal robot placement, involve determining solutions in the robot's configuration space (C-space), the space of all possible configurations the robot may take. For robot manipulators with many joints, the C-space is high-dimensional with typically nontrivial regions of feasibility, such as the set of collision-free configurations. Creating exact models for these regions of C-space becomes unwieldy for even simple robot manipulators. Without a model of the robot's C-space, the most popular approaches to solving problems in C-space involve randomly sampling hundreds to thousands of robot configurations, which unfortunately also requires computationally-expensive verification of the validity of each of these samples. To avoid or alleviate the computational

burden associated with validity checking, we explore the challenge of accurately and efficiently approximating models for a robot’s C-space in this work.

We propose the Fastron algorithm, a learning-based approach to model a robot’s C-space. This algorithm creates a nonlinear model that identifies whether a query robot configuration is either in-collision or collision-free an order of magnitude faster than state-of-the-art validity checking algorithms. To further improve the accuracy of the Fastron model, we develop a new similarity function called the forward kinematics (FK) kernel. With the new FK kernel, model accuracy increased to 95%, a significant improvement over the 75% accuracy when using the original similarity function. Using the Fastron model with the FK kernel for validity checking enabled up to 3 times faster motion planning solutions. As the FK kernel has the properties of a covariance function, we also apply this kernel to Gaussian process (GP) regression to estimate a robot’s distance to collision. The GP model obtains accurate distance estimates 70-80 times faster and allows an order of magnitude faster convergence for trajectory optimization when compared to geometric distance evaluation techniques. With each of these contributions, we demonstrate the advantage of learning-based robot configuration space modeling.

Chapter 1

Introduction

1.1 Robot Manipulators

Robot manipulators are typically actuated kinematic chains, which is a set of linked rigid bodies with controllable actuators. Robot manipulators typically consist of an arm and a tool at the end of the arm called an *end effector*. Fig. 1.1a shows an example of a robot manipulator on a table. As their name suggests, manipulators are designed to safely manipulate objects in their world, performing tasks including warehouse pick and place tasks [14], automotive assembly [15], and robotic surgery [16]. Examples of actuators that move the arms include servomotors [17] and stepper motors [18], and smoother or safer actuation can be achieved with these rigid actuators by using a belt-driven approach [19] or series elastic actuators [20]. In this work, we focus on rigid-link manipulators, but the design and application of curved continuum manipulators, often actuated by multiple tendons or cables [21] or by a series of concentric tubes [22], are ongoing areas of research.

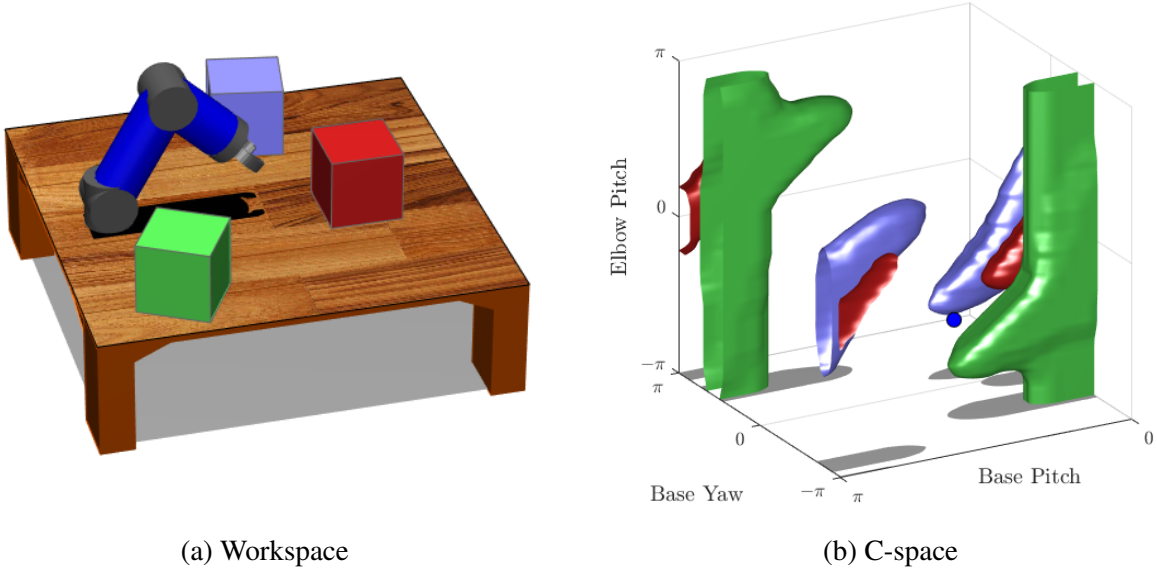


Figure 1.1: (a) A 3 DOF robot and multiple cube obstacles. (b) A C-space representation where each surface represents the boundary between C_{obs} and C_{free} .

1.1.1 Kinematics

Forward kinematics for a kinematic chain determines the pose (position and orientation) of a point on the chain given the joint values. Forward kinematics is often performed to determine the location of the robot's end effector (typically located at the most distal part of the kinematic chain), but it may also be used to determine the location of other points of the arm. When using the standard Denavit-Hartenberg (D-H) convention, the homogenous transformation matrix of the pose of joint i in the frame of joint $i - 1$ is

$${}^{i-1}\mathbf{A}_i = \begin{bmatrix} \cos(\theta_i) & -\cos(\alpha_i)\sin(\theta_i) & \sin(\alpha_i)\sin(\theta_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\alpha_i)\cos(\theta_i) & -\sin(\alpha_i)\cos(\theta_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.1)$$

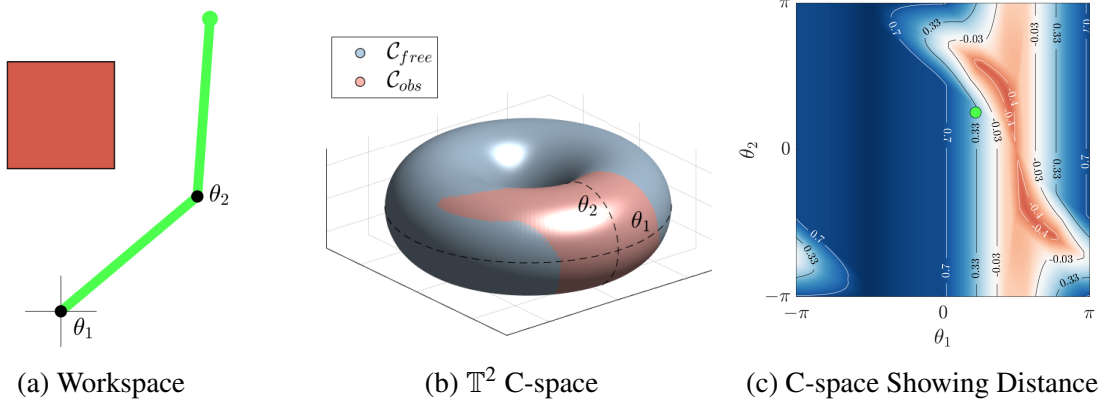


Figure 1.2: (a) A 2 DOF robot and a square obstacle. θ_1 and θ_2 denote the robot’s joint values. (b) A toroidal representation of the C-space. The pink and blue regions represent C_{obs} and C_{free} , respectively. (c) A C-space representation where the minimum workspace distance (or penetration depth for in-collision configurations) for each configuration is shown. The axes represent each DOF of the robot.

where d_i and a_i are the distances between the joints’ origins along the i^{th} x-axis and z-axis, respectively, and θ_i and α_i are the angles between the joints’ x-axes measured along the i^{th} z-axis and vice versa, respectively [23]. θ_i and d_i are the joint variables for revolute and prismatic joints, respectively, and a *configuration* of a robot manipulator with D degrees-of-freedom (DOF) is represented as a D -dimensional vector of these joint variables. Given a robot manipulator’s configuration, the position of the robot manipulator is fully defined [24]. The other D-H parameters are fixed depending on the geometry of the robot. Note that when using the standard D-H convention, the origin of the frame of the i^{th} joint is placed at the distal end of the i^{th} link.

Starting with the base transform relative to a world frame ${}^w\mathbf{A}_0$, the pose of the i^{th} joint is ${}^w\mathbf{A}_i = {}^w\mathbf{A}_0 {}^0\mathbf{A}_1 \dots {}^{i-1}\mathbf{A}_i$, where each transform in the chain is dependent on the robot configuration as described in the previous paragraph. The pose for an arbitrary point on the i^{th} link is found by appending an additional static transformation $\mathbf{T}$ to the pose of the i^{th} joint: ${}^w\mathbf{A}_i \mathbf{T}$.

While forward kinematics determines the pose of a point on the robot manipulator given the joint values, *inverse kinematics* is the task of determining a set of joint values that achieves a desired pose. One use of inverse kinematics is determining how to position the robot arm such

that its end effector can reach a certain location. While forward kinematics equations produce only one pose for each set of joint values, there may exist multiple possible joint configurations that reach a certain pose if there are more DOF in the joint space than constraints in the desired pose. Utilizing this redundancy in the joint space is useful for satisfying certain constraints such as avoiding joint limits [25], and iterative procedures are typically employed to find inverse kinematics solutions as quickly and accurately as possible [26].

1.1.2 Configuration Space

A robot exists and moves in its *workspace*, which is either an $\mathbb{R}^2$ or $\mathbb{R}^3$ space. Fig. 1.1a provides an example workspace, where there is a robot manipulator with 3 cubic workspace obstacles.

A robot's *configuration space* (or C-space) is a space in which each element represents a unique configuration of the robot [24]. A robot's configuration fully defines every point of the robot [24]. The joint space of a robot manipulator is an example C-space, as a set of joint positions may be sufficient to fully define the pose of a robot manipulator in the workspace¹. The dimensionality of the C-space matches the number of independent variables that can uniquely represent a configuration of the robot, which is also known as the degrees of freedom (DOF) of the robot [24].

The workspace may include restrictions on the set of feasible robot configurations. Examples include workspace obstacles (such as the three cubes in Fig. 1.1a restricting the 3 DOF arm), self-collisions, or joint limits. For a given workspace, each configuration in the C-space can belong to one of two regions: C_{free} or C_{obs} . A configuration is in C_{free} if the robot is not in contact with any workspace obstacle when in that corresponding configuration; otherwise, the configuration is in C_{obs} [24, 27]. The robot and obstacles in Fig. 1.1a are represented as the point

¹Using forward kinematics as defined in the Section 1.1.1, the workspace location of any point of the robot manipulator may be identified given the manipulator's joint values

and amorphous bodies in the C-space representation in Fig. 1.1b, respectively. The advantage of planning in C-space is it is easier to plan how to move a point than it is to move bodies with volume [24]. An example workspace and its corresponding C-space for a 3 DOF robot arm are shown in Fig. 1.1. The colors of the cubes in the workspace match the colors of the boundaries of the C_{obs} regions in the C-space.

Note that in Fig. 1.1b, the C-space appears to be $\mathbb{R}^3$ because there are 3 DOF for the robot manipulator: the base yaw joint, the base pitch joint, and the elbow pitch joint. However, each of these joints are a revolute joint, which actually has a unit circle space $\mathbb{S}^1$ [24]. Thus, while Fig. 1.1b appears to be $\mathbb{R}^3$, it is actually a 3-dimensional torus ($\mathbb{T}^3 = \mathbb{S}^1 \times \mathbb{S}^1 \times \mathbb{S}^1$) cut along each dimension between $-\pi$ and π . Considering the 2 DOF robot in Fig. 1.2a, its $\mathbb{T}^2$ C-space is shown in Fig. 1.2b. For ease in plotting, we will flatten 2- and 3-dimensional toroidal C-spaces when displaying them.

The task of representing a robot’s C-space and demarcating the C_{obs} and C_{free} regions are often treated as synonymous. However, the C_{obs} and C_{free} regions are not the only information that can be portrayed in a robot’s C-space. In fact, any configuration-based metric can be represented in C-space, such as discretely labeling which configurations can reach a workspace target, or continuously representing the distance to collision in the workspace. Fig. 1.2 shows the workspace distance for each configuration of a 2 DOF robot. Note that we use a positive value when the robot is not in collision to represent the minimum distance to collision and a negative value to represent penetration depth, which is the minimum distance required to move a robot out of collision.

1.2 Motion Planning

Motion planning, the task of determining a path for a robot from a start to a goal position while satisfying constraints such as obstacle avoidance, is a requirement for almost all applications

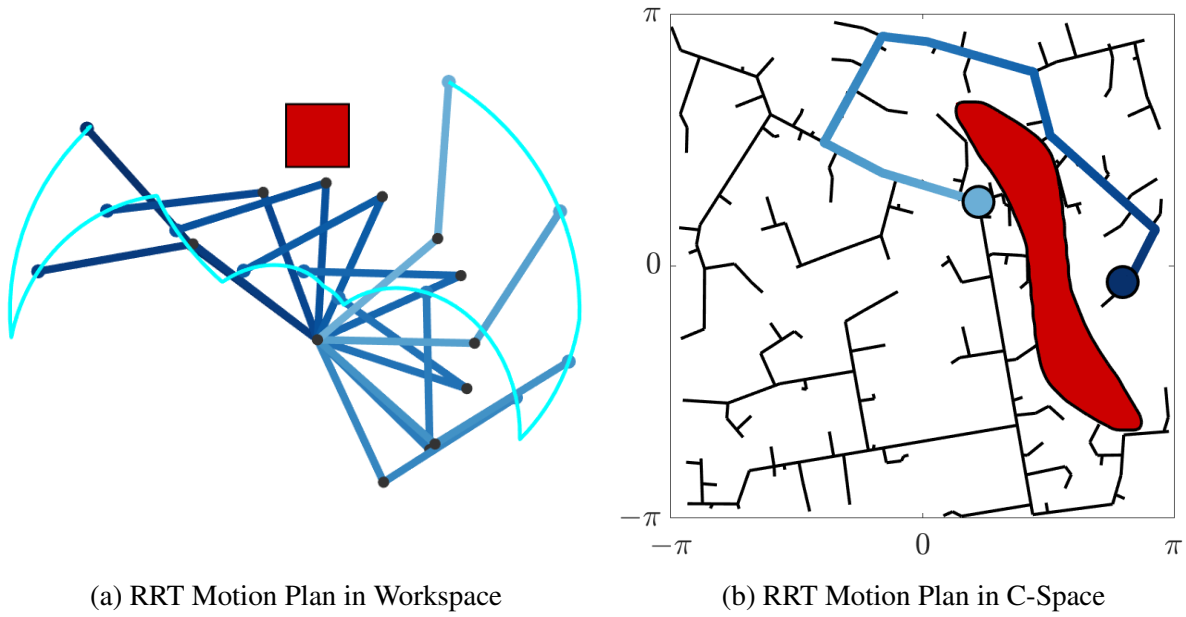


Figure 1.3: A motion plan for a 2 DOF robot found using an RRT [1] is represented in both the (a) workspace and (b) C-space.

in which a robot must move in its environment. Motion planning for robots is often performed in its C-space. As shown in Fig. 1.1b, the robot manipulator in the workspace shown in Fig. 1.1a is represented as a point in C-space. The advantage of planning in C-space is it is easier to plan how to move a point in a space than it is to move bodies with volume [24].

The workspace may include restrictions on the set of feasible robot configurations. Examples include workspace obstacles (such as the three cubes in Fig. 1.1a restricting the 3 DOF arm), self-collisions, or joint limits. Remaining collision-free during a motion is a vital characteristic required for most motion plans. To avoid collisions with workspace obstacles, a continuous path between the starting configuration and the goal configuration that exists entirely in the C_{free} region must be determined. Unfortunately, access to the shapes and locations of these regions in a robot's C-space is generally limited because the mapping of obstacles in the workspace to C_{obs} in the C-space is generally too complex for practical applications [24, 28]. Thus, methods for efficiently determining a continuous path that connects two configurations that circumnavigates C_{obs} regions is still an ongoing topic of research. Popular techniques of motion planning for robot

manipulators include potential field methods and sampling-based methods.

An important property to consider in motion planning is *completeness*, which means the planner finds a path if it exists in asymptotically bound time. A weaker form of completeness is *probabilistic completeness*, which ensures that as the amount of planning time increases, the probability of finding a solution approaches 1 [24].

1.2.1 Potential Field Methods

Potential field methods use potential functions, which are differentiable functions that represent energy in a field. The gradient vector field of the potential function points away from obstacles and toward a goal region, thus dictating the movement of the robot in C-space as if it were a charged particle moving in an electric field. Potential field methods are enticing because they can define natural motions that are reactive to a dynamic environment, such as in the case where a human is working close to the robot [29]. A disadvantage of most potential field methods is the existence of local minima in the field where the repulsive forces away from obstacles and attractive forces toward a goal cancel out and cause the robot to get stuck [24].

In the case of kinematic chains, these “charged particles” can be placed on each link of the manipulator, and the virtual repulsive forces can be calculated based on the distance to obstacles in the workspace. The C_{obs} regions are not represented explicitly when using this method. Instead, Jacobian functions, which relate changes in the position of a point on the robot in the workspace to a change in configuration, are utilized. Once again, this technique is prone to local minima [24], thus not guaranteeing completeness.

1.2.2 Sampling-Based Motion Planning

Sampling-based motion planning involves taking random samples in C-space, assessing the configuration’s validity, and creating a connected graph of the valid configurations. Single-

query planners attempt to determine a path between one pair of start and goal nodes, while multi-query planners develop a roadmap from which different goal configurations may be selected. These sampling-based methods were developed to determine motion plans without explicit representations of the C_{obs} or C_{free} regions in a probabilistically complete way.

A probabilistic roadmap (PRM) is an example of a multi-query planner. A PRM creates a roadmap using random samples by creating collision-free edges between random configurations. During runtime, the start and goal configurations are connected to the roadmap, and then the shortest path along the edges of the roadmap are traversed to achieve the final motion plan.

Among the most popular single-query planners is the Rapidly-Exploring Random Trees (RRT) algorithm, which grows a tree rooted at the start until a leaf node reaches the goal region [1]. Following the goal node up the tree back to the root node achieves the final motion plan. An example motion plan determined for a 2 DOF robot with the RRT algorithm is shown in Fig. 1.3. Fig. 1.3a shows the motion trail of the robot around the square workspace obstacle (the curved line represents the path the end effector takes). The tree shown in Fig. 1.3b shows all the nodes explored before finding the final path, which is the thicker blue curve connecting the start to the goal configuration. Numerous other tree-based motion planners have been developed to improve sample efficiency, such as RRT-Connect [30], or to minimize path length, such as RRT* [31], Informed-RRT* [32], and BIT* [33].

A common component among all sampling-based motion planners is a validity checker that must be called on each sample and each edge between samples. Validity checking typically includes determining the configuration's collision status (i.e., determining whether a query configuration is in C_{obs} or C_{free}) via collision checking, along with any other constraint. Repeated collision checking is computationally expensive and takes up to 90% of the computation time during motion planning [34]. Thus, efficient motion planners are typically those that reduce this computational burden either by reducing the number of samples generated or by delaying sampling until absolutely necessary [24, 35].

1.3 Collision Detection

The task of determining whether an object intersects another object is known as *collision detection* and is a challenge prevalent in many areas including robotics, video games, simulations/animations, and CAD/CAM [36]. In cases where there are a large number of collision checks to be performed, there exists a challenge in performing these checks efficiently and accurately.

In the case of robotics, as described in Section 1.2, collision detection is to verify whether a certain configuration is safe by being collision-free. This involves performing a collision check between each relevant body of the robot (such as a link in the kinematic chain) and each workspace obstacle. For a given robot configuration, if any robot body-obstacle pair is intersecting, further collision checks are not required for that configuration because the robot is already known to be in C_{obs} . Otherwise, each body-obstacle pair must be checked, which has quadratic time complexity as more objects are involved in the environment [36]. For each motion planning query, hundreds to thousands of robot configurations must be considered [37], thereby making efficient collision checking a requirement for effective motion planning [38].

Some body-obstacle pairs may be quickly dismissed as being not in collision depending on their locations, while object pairs that are closer to each other require closer evaluation. Dismissing distant objects and evaluating exact collision statuses between nearby objects is referred to as broad phase and narrow phase collision checking, respectively [36].

Pairwise exact geometric collision checking is expensive, so the goal of broad phase techniques is to restrict these collision evaluations only to objects that are nearby each other [36]. Spatial partitioning methods are an example of broad phase techniques which, as the name implies, divide the space in which the objects reside into multiple regions, and only objects that reside in the same region should be considered for finer resolution narrow phase evaluation. Octrees [39] provide a form of spatial partitioning. Another broad phase technique is bounding

volume hierarchies (BVHs) [36], a technique supported by the popular Flexible Collision Library (FCL) designed for robotic collision queries [8]. Bounding volumes (BVs) are simple shapes, such as axis-aligned bounding boxes, that enclose more complicated shapes. BVHs arrange BVs in a tree structure, where nearby BVs have a common ancestor in the tree. As with spatial partitioning, BVHs allow finer resolution checks to be dismissed if the objects are distant from each other, which in this case means the BVs of the objects are non-overlapping.

For object pairs that cannot be dismissed with broad phase collision checking, narrow phase collision checking must be performed between the objects. Narrow phase techniques typically require convex shapes, which are shapes that fully contain the line that connects two points inside of the shape. To determine whether two convex shapes A and B are intersecting, they must have a point in common. In other words, considering the set of all pairwise differences between the points in A and the points in B (also known as the Minkowski difference of A and B), A and B are intersecting if the origin is included in this set. Efficient narrow phase algorithms, such as the Gilbert-Johnson-Keerthi (GJK) algorithm [5], attempt to determine whether or not the Minkowski difference encloses the origin without computing the entire Minkowski difference. GJK is one of the most popular algorithms for evaluating whether convex shapes are intersecting and is the narrow phase collision checking technique used in FCL [8].

1.4 Machine Learning

Machine learning techniques are those that attempt to create representative models based on data that may be used for prediction or analysis. Handcrafted rules may be developed to represent data, but this method is burdensome, may lack generalization beyond the data for which the rules were designed, and is often inaccurate when exceptions to the rules are encountered. Instead, the machine learning approach is useful to automatically adapt (or *learn*) a set of parameters of some generalizable model to the acquired data [40]. Numerous fields of

research have adopted the machine learning approach to determining models for their applications, including protein structure modeling [41], financial decision making [41], and speech recognition [40]. In each of these applications, handcrafted methods of analysis may be developed, but the data-driven machine learning approach makes the generation of appropriate models efficient.

Supervised machine learning is the case where the model is fit to a training dataset that includes example inputs and their respective target values. *Classification* is one form of supervised learning where a model fit to a training dataset predicts the category to which a query point belongs, such as identifying the correct digit given an image of a handwritten number. Unsupervised learning involves a training dataset without the respective target values and is used to analyze the structure of the data itself, such as the case when clustering together datapoints that are seemingly similar. In this work, we primarily focus on the supervised form of machine learning.

1.4.1 Discriminative Classification

Discriminative classification is the task of assigning class labels to query points. A discriminative classifier is a machine learning model that can produce these class labels given a query point. If there are only two classes, the classifier is called a binary classifier.

The goal of training a binary classifier is to find a model whose output predicts in which of two classes a query point belongs. The parameters and weights chosen for the model are based on a training set $\mathcal{X} = \{\mathcal{X}_1, \dots, \mathcal{X}_N\}$ and its associated training labels $\mathbf{y}$. For binary classification, $\mathbf{y}_i$ can be either ± 1 . To fit a binary classifier to the regions of a manipulator’s C-space, the elements of $\mathcal{X}$ are D -dimensional robot configurations, e.g., the d joint angles defining the manipulator’s position. Considering robot manipulators often have joint limits, the joint angles may be scaled to a region of $\mathbb{R}^D$. For robot manipulators without joint limits (such as revolute joints that may rotate indefinitely), the space for each such joint is a 1-sphere ($\mathbb{S}^1$). For ease in generating data and fitting a model, we can limit $\mathbb{S}^1$ spaces to $(-\pi, \pi]$, which will consequently allow any robot

manipulator's C-space to scale to a region of $\mathbb{R}^D$.

Since a label y can be ± 1 , one possible model for prediction may be $\text{sign}(f(\mathbf{x}))$, where $f : \mathbb{R}^D \rightarrow \mathbb{R}$ is some hypothesis function. We consider the *margin* of $f(\cdot)$ at a point $\mathbf{x}$ to be the product of $f(\mathbf{x})$ and the true label of $\mathbf{x}$, $yf(\mathbf{x})$. Of course, the ideal hypothesis function would have positive margin for all possible inputs. One possible hypothesis function may be $f(\mathbf{x}) = \sum_i \langle \phi(\mathcal{X}_i), \phi(\mathbf{x}) \rangle_{\mathcal{F}} \boldsymbol{\alpha}_i$ where $\phi : \mathbb{R}^d \rightarrow \mathcal{F}$ is a mapping from the input space in which $\mathcal{X}$ exists to some (often higher-dimensional) feature space $\mathcal{F}$ and weight vector $\boldsymbol{\alpha} \in \mathbb{R}^N$ contains a weight for each $\mathcal{X}_i \in \mathcal{X}$. The feature mapping function $\phi(\cdot)$ can be explicit, such as by creating polynomial terms of the elements of the input space, or implicit by computing the inner product in the feature space directly as will be described in the next section. This hypothesis can be interpreted as a weighted sum comparing the query configuration to each training configuration and is used in discriminative models such as perceptrons [42] and support vector machines [43, 2]. $f(\mathbf{x})$ may be written in matrix form for the configurations in $\mathcal{X}$ as $\mathbf{F} = \mathbf{K}\boldsymbol{\alpha}$ where $\mathbf{F}_i = f(\mathcal{X}_i) \forall \mathcal{X}_i \in \mathcal{X}$ and $\mathbf{K}$ is the Gram matrix for $\mathcal{X}$ where $\mathbf{K}_{ij} = \langle \phi(\mathcal{X}_i), \phi(\mathcal{X}_j) \rangle_{\mathcal{F}}$.

There are numerous other hypothesis models that may be used for discriminative classification. Nearest neighbor models [44] assign a label to a query point according to the labels of the points in the training set closest to the query point. With decision trees [45], a set of splitting rules are defined to classify a query point traversing a tree structure. Neural networks [46] form a weighted sums of simple decision rules to create powerful classifiers and are becoming increasingly popular in deep learning where multiple levels of abstraction separate the input data from their targets [47]. Similar to the weighted sum of inner products described above, data in the input space may be mapped to a feature space to assist in fitting a model for each of these other methods.

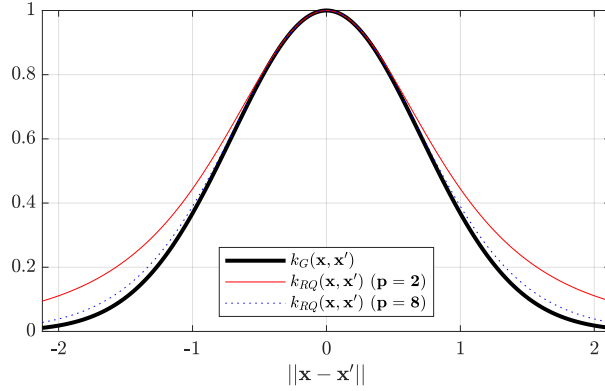


Figure 1.4: Examples of radial basis function kernels, including the Gaussian kernel function $k_G(\cdot, \cdot)$ and the 2^{nd} and 8^{th} order rational quadratic kernel functions $k_{RQ}(\cdot, \cdot)$.

1.4.2 Kernel Functions

A *kernel* function $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{F}}$ compares a configuration $\mathbf{x}$ to $\mathbf{x}'$ by mapping to some feature space $\mathcal{F}$ using $\phi(\cdot)$ and taking an inner product. As a similarity function, $k(\cdot, \cdot)$ should provide a large score for two similar configurations and a low score for dissimilar configurations. The kernel matrix $\mathbf{K} \in \mathbb{R}^{N \times N}$ for a dataset of N points $\mathcal{X}$ where $\mathbf{K}_{ij} = k(\mathcal{X}_i, \mathcal{X}_j)$ is thus a similarity matrix and is a useful tool in machine learning techniques, such as support vector machines [43, 2] and kernel perceptrons [42], where similarity between two samples may suggest they share classification labels. When $\phi(\mathbf{x}) = \mathbf{x}$, the resulting kernel is a linear kernel, which may be used when the input space of the data is sufficient without any mapping to a new feature space. Some examples of nonlinear kernels are given below and are useful when the mapping the input space to another feature space allows easier analysis.

Gaussian Kernel

A popular type of kernel function is a radial basis function (RBF) kernel, which is a function of the distance between the two points. An example of an RBF kernel function is the Gaussian kernel, defined as

$$k_G(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2) \quad (1.2)$$

where $\gamma > 0$ is a parameter dictating the width of the kernel. $k_G(\cdot, \cdot)$, shown as the solid black curve in Fig. 1.4, is often considered the default kernel choice for kernel-based machine learning applications when there is limited prior knowledge of the underlying structure of the data [48].

$k_G(\cdot, \cdot)$ is known to be a *positive definite kernel* [11], which means $\mathbf{K}$ is a positive definite matrix (i.e., $\boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\alpha} \geq 0 \forall \boldsymbol{\alpha} \in \mathbb{R}^N$) when each configuration in $\mathcal{X}$ is unique [49]. According to Mercer's theorem, for each positive definite kernel, there exists a mapping $\phi: \mathbb{R}^d \rightarrow \mathcal{F}$ where $\mathcal{F}$ is a Hilbert space such that $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{F}}$ [50]. Thus, there is no need for explicit mapping to a feature space when utilizing the Gaussian kernel function; computing the inner product without explicitly mapping to a feature space is known as the *kernel trick* [51]. The kernel trick is powerful tool that is useful for learning a classifier that requires richer features than provided in the input space $\mathbb{R}^D$ but explicit mapping to a higher-dimensional may be difficult or time-consuming. This implicit mapping to a richer feature space allows easier separation of two classes when the boundary between the classes is nonlinear and complex, which is typically the case for binary classification problems, including the C_{free} and C_{obs} regions of a robot's C-space.

While the Gaussian kernel function forgoes explicit feature space mapping, an explicit feature space may still be defined. Consider $x, x' \in \mathbb{R}$. Using the Taylor expansion of $\exp(\cdot)$,

$$k_G(x, x') = \exp(-\gamma(x - x')^2) = \exp(-\gamma(x^2 + x'^2 - 2xx')) \quad (1.3)$$

$$= \exp(-\gamma x^2) \exp(-\gamma x'^2) \exp(2\gamma xx') \quad (1.4)$$

$$= \exp(-\gamma x^2) \exp(-\gamma x'^2) \left(1 + \frac{2\gamma xx'}{1!} + \frac{(2\gamma xx')^2}{2!} + \dots \right) \quad (1.5)$$

$$= \exp(-\gamma x^2) \exp(-\gamma x'^2) \left[1, \sqrt{\frac{2\gamma}{1!}}x, \sqrt{\frac{(2\gamma)^2}{2!}}x^2, \dots \right] \left[1, \sqrt{\frac{2\gamma}{1!}}x', \sqrt{\frac{(2\gamma)^2}{2!}}x'^2, \dots \right]^\top. \quad (1.6)$$

By rewriting the Gaussian kernel as a dot product, a feature space mapping function for

the 1-dimensional Gaussian kernel is

$$\phi_G(x) = \exp(-\gamma x^2) \left[1, \sqrt{\frac{2\gamma}{1!}}x, \sqrt{\frac{(2\gamma)^2}{2!}}x^2, \dots \right]^\top. \quad (1.7)$$

For kernels that are products of other kernel functions, the feature space is the Kronecker product of the feature spaces of the kernel factors [52]. The Gaussian kernel for d -dimensional inputs $\mathbf{x}$ and $\mathbf{x}' \in \mathbb{R}^d$ can be written as a product of 1-dimensional Gaussian kernels:

$$k_G(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2) = \exp(-\gamma \mathbf{x}^\top \mathbf{x}) \exp(-\gamma \mathbf{x}'^\top \mathbf{x}') \prod_{i=1}^d \exp(2\gamma \mathbf{x}_i^\top \mathbf{x}'_i) \quad (1.8)$$

where $\mathbf{x}_i$ is the i^{th} element of $\mathbf{x}$. Thus, a feature space mapping for the multidimensional Gaussian kernel is

$$\phi_G(\mathbf{x}) = \exp(-\gamma \mathbf{x}^\top \mathbf{x}) \bigotimes_{i=1}^d \left[1, \sqrt{\frac{2\gamma}{1!}}\mathbf{x}_i, \sqrt{\frac{(2\gamma)^2}{2!}}\mathbf{x}_i^2, \dots \right]^\top \quad (1.9)$$

where $\bigotimes$ denotes the Kronecker product.

Approximations of the explicit mapping of the Gaussian kernel can thus be formed using a finite number of elements of Eq. 1.9.

Rational Quadratic Kernel

While the Gaussian kernel is popular for machine learning applications, exponential evaluations are computationally expensive operations, which can slow down training and classification especially for large N . Exponentiation can be avoided by representing the Gaussian kernel with a limit:

$$k_G(\mathbf{x}, \mathbf{x}') = \lim_{p \rightarrow \infty} \left(1 + \frac{\gamma}{p} \|\mathbf{x} - \mathbf{x}'\|^2 \right)^{-p}. \quad (1.10)$$

For finite values of p , the above kernel is the rational quadratic kernel [11], which we represent as $k_{RQ}(\cdot, \cdot)$. Fig. 1.4 shows examples of the 2^{nd} and 8^{th} order rational quadratic kernel functions. Note that the higher order rational quadratic kernel better matches the Gaussian kernel,

but the lower order version still has roughly the same shape.

The rational quadratic kernel may alternatively be derived as a superposition of Gaussian kernels of varying kernel widths: $k_{RQ}(\mathbf{x}, \mathbf{y}) = \int P(\gamma; p, \theta) k_G(\mathbf{x}, \mathbf{y}) d\gamma$, where $P(\gamma; p, \theta)$ is a gamma distribution [11]. The sum of positive definite kernels is also a positive definite kernel [49]. As a sum of Gaussian kernels, the rational quadratic kernel satisfies the positive definiteness property, showing $k_{RQ}(\mathbf{x}, \mathbf{y})$ provides the result of an inner product in a richer Hilbert space via the kernel trick according to Mercer's theorem and yields positive definite Gram matrices when each configuration in $\mathcal{X}$ is unique. The explicit feature space mapping of the rational quadratic kernel would be a concatenation of the mappings in Eq. 1.9 with γ distributed according to a gamma distribution.

Better approximations to the Gaussian kernel are achieved with higher values of p as seen in Fig. 1.4. When p is a power of 2, efficient implementation of the rational quadratic kernel may take advantage of tetration or repeated exponentiation. With $p = 2$, completely computing a 5000×5000 Gram matrix using the Eigen C++ library takes 257.0 ms using $k_G(\cdot, \cdot)$, while using $k_{RQ}(\cdot, \cdot)$ takes 171.3 ms, illustrating that kernel evaluations can be performed 1.5 times faster with the rational quadratic kernel.

Chapter 2

Modeling Configuration Space

As described in Section 1.1.2, a robot's C-space is a D -dimensional space in which each element represents a unique configuration of the robot with D degrees-of-freedom. Representing a robot's C-space typically involves also representing its C_{obs} and C_{free} regions. The shapes and locations of C_{obs} regions in the C-space are due to the shapes and locations relative to the robot in the workspace; the C_{obs} regions that arise due to a workspace obstacle are sometimes called *C-space obstacles* [28]. Except for some specific cases, creating explicit models of these regions is a difficult task, even with complete knowledge of the location and geometry of the workspace obstacles. Considering the case of a purely translational robot (or a robot whose rotation has no effect on its footprint such as a circular robot), the C-space obstacle is the Minkowski difference between the workspace obstacles (which must be decomposed into convex shapes). For even more slightly complicated robots, modeling the C-space obstacles explicitly is extremely difficult as they would involve using application-specific semi-algebraic models [24, 28]. Problems that require planning in C-space would be practically impossible to solve if the C-space obstacles are exactly represented explicitly in this manner [28].

2.1 Approaches to Approximate C-space

Sampling-based motion planners circumvent the issue of not having a C-space model by taking a guess-and-check approach, where guesses come from random samples in C-space and the checks involve assessing the samples' validity through collision checking. The downside to this approach, as mentioned earlier, is the computational cost of performing numerous collision checks searching in the search of C_{free} samples.

While determining exact models of C-space is often infeasible, in this section, we describe approaches to approximate an explicit model of C-space. The majority of these approaches utilize machine learning techniques to determine a model based on sampled configurations. Querying these approximate C-space models in lieu of the collision checking routine or using these approximate models to facilitate the search for a plan may reduce the computational burden associated with the guess-and-check procedure sampling-based motion planners follow.

2.1.1 Grid-Based

Creating a model for a robot's C-space is difficult for robots that are any more complicated than ones that are purely translational, such as non-circular mobile robots or kinematic chains with many joints. Rather than creating a continuous representation of the C-space obstacles, a dense grid in C-space can be defined where each cell of the grid encodes collision status information. This technique may be useful for visualization for low-dimensional robots, but not for much more such as motion planning unless the resolution of the grid is sufficiently high [24]. As the dimensionality of C-space increases with each additional DOF, exponentially more grid cells will be required for each additional DOF, rendering this grid-based approach to modeling C-space extremely inefficient.

Paden et al. use a 2^n -tree (or generalized quadtree) approach to modeling the C_{free} region of C-space [53]. The center of the C-space is first labeled as either C_{free} , C_{obs} , or "unsure". If the

cell is marked as “unsure”, the cell is divided along each axis, each of those cells is labeled, and the process repeats until no cell is marked as “unsure” or a certain grid resolution is met. The authors define a bound on the size of the neighborhood around a given configuration that shares the same collision status. This bound is based on the distance between the robot and the closest obstacle and is used to determine the collision status of a grid cell given the status of its center point. The authors apply this technique to low-dimensional robot manipulators and determine collision-free motion plans efficiently by connected the centers of cells marked as C_{free} . However, exponentially more grid cells will probably be required as the number of DOF increases.

2.1.2 k-Nearest Neighbors Models

Burns and Brock [54] use a k-nearest neighbors (k-NN) model for C-space. With k-NN, the predicted collision status of a query configuration matches that of the majority of the k nearest configurations. Their k-NN approach is not intended to be used in the case of moving workspace obstacles, so their model cannot easily adapt if the environment changes. Furthermore, their approach was designed to assist with efficient configuration space sampling for motion planning rather than for accurate C-space modeling.

Pan and Manocha [55] also use a k-NN model, accelerated by locality-sensitive hashing. Their method significantly reduces the time required for collision checking for sampling-based motion planners by building a database to use for k-NN queries. Though not implemented in a changing environment, they propose their method can extend to a changing environment by gridding the workspace and only performing collision checks on configurations associated with dynamic cells.

2.1.3 Gaussian Mixture Models

Huh and Lee [56] use Gaussian mixture models (GMMs) to represent C_{free} and C_{obs} , where multivariate Gaussians are fit to samples from each C-space region. The GMMs are used as a discriminative classifier by predicting a query configuration to be in either C_{free} or C_{obs} according to the Mahalanobis distances from the means of the GMMs. Additionally, as generative models, these GMMs may be used to bias the sampling for motion planners toward areas that are likely to be C_{free} .

Their iterative GMM technique allows the model to update when workspace obstacles move to intersect a planned trajectory. A limitation of the GMM approach is the model may not fit irregularly-shaped C_{obs} regions effectively as GMMs use a limited number of Gaussians. Additionally, the underlying generative models are updated to fit new information, which consequently does not guarantee the resulting discriminative classifier immediately fits the new information. The data for this method is generated by a sampling-based motion planning algorithm multiple times. Consistent results for the model are achieved after 3-4 motion plans are generated, which is slow, especially in cases where the environment may change and invalidate a previously-trained model.

2.1.4 Support Vector Machines

Pan et al. [3] use incremental support vector machines (SVMs) to represent an accurate collision boundary between C_{free} and C_{obs} in C-space for a pair of objects and an active learning strategy to iteratively improve the boundary. This method is suitable for moving objects because moving one body relative to another body's frame is simply represented as translating a point in C-space. However, a new model is required for each pair of objects, and each model must be trained offline (which also means the geometries of all workspace obstacles must be known a priori).

In fact, a similar approach is used to create a C-space model by composing multiple subregions of C_{obs} [57]. Multiple SVM models are composed, where each SVM model is trained to model C_{obs} and C_{free} for each link of the robot arm vs. the environmental obstacles. As with the single model approach described in the previous paragraph, the models must be trained offline and the obstacle geometries must be known a priori. Furthermore, this method is only applicable to cases with static (or non-changing) environments.

2.1.5 Neural Networks

Neural networks are becoming increasingly popular in machine learning, but have had limited applications in C-space modeling.

Multilayer perceptrons have been used for modeling C_{free} and C_{obs} . Tran et al. use an encoded (via contractive autoencoder) point cloud representation of the environment. Training a contractive autoencoder involves training an encoder network to form a compressed representation of the input and a decoder network to reconstruct the original inputs [58]. The benefit of an autoencoder is a reduced representation of the original point cloud representation using the encoder network can be formed, which in this case is fed into another neural network for the classification of collision statuses [59]. The accuracy of this method is around 70% [59], showing that this approach models the C-space based on point cloud representations of the workspace somewhat correctly.

ClearanceNet, another neural network-based approach, has been used to estimate distance between a robot arm and its environment [60]. A robot arm's pose is part of the input to the network. When the arm is not in collision, a positive-valued distance to the closest obstacle is the output. When the arm is in collision, a negative-valued penetration depth is the output, which is the minimum distance required to move to get out of collision. As the signs of the outputs denote its collision status, this neural network also models C_{free} and C_{obs} of the C-space. The accuracy of this approach is around 95%. However, this approach has only been trained and tested on

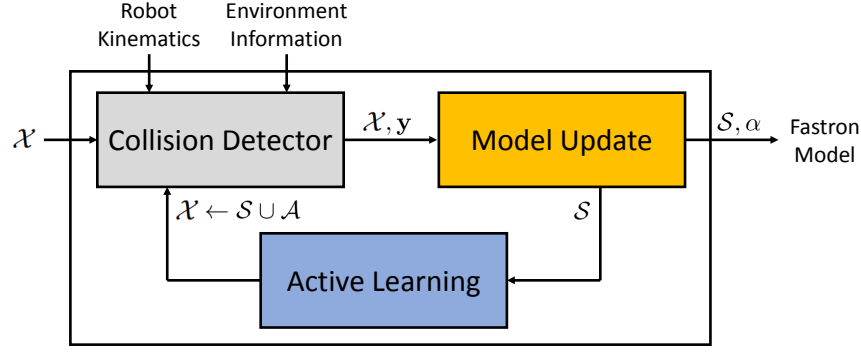


Figure 2.1: Block diagram illustrating the pipeline of the Fastron algorithm.

parameterized representations of the environment.

2.2 Fastron

The Fastron algorithm trains a kernel-based machine learning model to model the $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$ regions of a robot’s C-space given a labeled dataset. The training procedure of Fastron, whose name is a portmanteau of the words “fast” and “perceptron”, was inspired by the kernel perceptron algorithm [42] due to its simplicity in implementation and its ability to model training points in an online manner. Modifications were made to make training faster and allow efficient adaptation to a changing dataset. As obstacles in the workspace move, an active learning procedure collects new samples to augment the dataset, and the model is efficiently updated. The block diagram in Fig. 2.1 shows the sequence of steps for the Fastron algorithm.

In this section, we describe the Fastron algorithm in detail and demonstrate its accuracy in modeling C-space. We also show various applications of Fastron, including its use as a faster collision checker for motion planning and in the optimization of the placement of robot arms for a da Vinci® system.

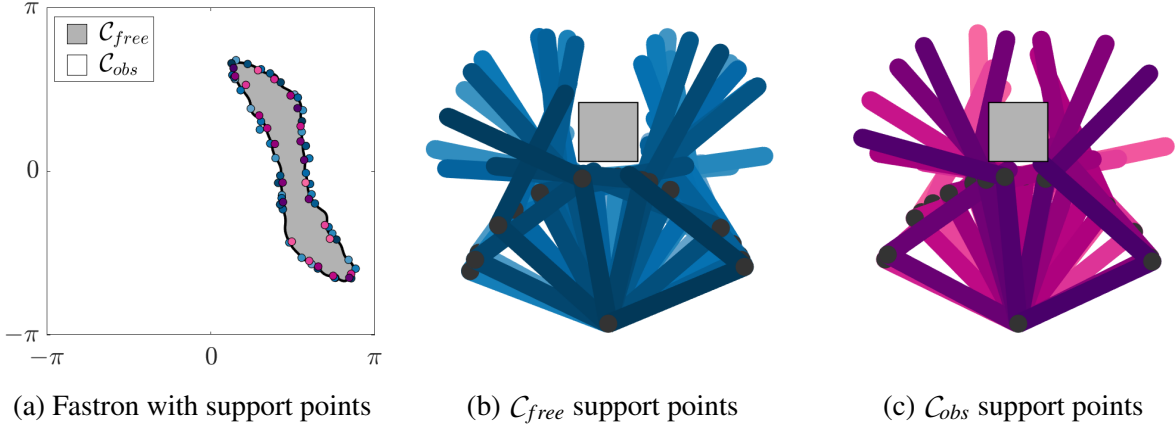


Figure 2.2: (a) Fastron C-space model of C_{obs} and C_{free} with support points. (b) Workspace representation of C_{free} support points. (c) Workspace representation of C_{obs} support points.

2.2.1 Model Update

Derivation of Update Rule

The Fastron algorithm is inspired by the kernel perceptron in the sense that it iteratively adds or adjusts the weights for configurations that are incorrectly classified by the current model [42]. Unlike the perceptron algorithm, the Fastron algorithm prioritizes the configurations in $\mathcal{X}$ with the most negative margin and makes adjustments to the weights such that the configuration is forced to be correctly classified immediately after the update, which is not a guarantee with the standard perceptron algorithm. These changes result in a model with a large number of weights equal to 0 that generally takes fewer updates to converge to a solution compared to the perceptron algorithm. An example of a Fastron C-space model and the points required to represent the model is provided in Fig. 2.2a. Note that, with the update rule described in the following paragraphs, only the samples near the decision boundary are required for the model, thus creating a sparse model. We can also see that this selection of points near the C-space obstacle makes intuitive sense because these configurations are also close to the obstacle in the workspace representation as shown in Fig. 2.2b and 2.2c.

By prioritizing the training configuration with the most negative margin, we guarantee

that the weight we are updating is for a misclassified configuration. In what follows, parenthetical superscripts denote the training iteration upon which the given value depends. If $\mathcal{X}_i$ has the most negative margin of all configurations in $\mathcal{X}$ on iteration n , $\boldsymbol{\alpha}_i^{(n)}$ must be adjusted by $\Delta\boldsymbol{\alpha}_i^{(n)}$ such that $m^{(n+1)}(\mathcal{X}_i) > 0$ (thereby ensuring $\mathcal{X}_i$ is correctly classified immediately after the weight update):

$$y_i \left(\sum_{j \neq i} \mathbf{K}_{ij} \boldsymbol{\alpha}_j^{(n)} + \mathbf{K}_{ii} \left(\boldsymbol{\alpha}_i^{(n)} + \Delta\boldsymbol{\alpha}_i^{(n)} \right) \right) > 0. \quad (2.1)$$

As $\mathbf{K}_{ii} = 1$ when using the rational quadratic kernel, the above condition may be simplified to $m^{(n)}(\mathcal{X}_i) + y_i \Delta\boldsymbol{\alpha}_i^{(n)} > 0$, which may be enforced by setting $\Delta\boldsymbol{\alpha}_i^{(n)} = \mathbf{y}_i - \mathbf{F}_i^{(n)}$. This update rule makes $m^{(n+1)}(\mathcal{X}_i) = 1$. The weight and training hypothesis vector updates are thus:

$$\Delta\boldsymbol{\alpha}_i^{(n)} = \mathbf{y}_i - \mathbf{F}_i^{(n)} \quad (2.2)$$

$$\boldsymbol{\alpha}^{(n+1)} = \boldsymbol{\alpha}^{(n)} + \Delta\boldsymbol{\alpha}_i^{(n)} \mathbf{e}(i) \quad (2.3)$$

$$\mathbf{F}^{(n+1)} = \mathbf{F}^{(n)} + \Delta\boldsymbol{\alpha}_i^{(n)} \mathbf{K} \mathbf{e}(i) \quad (2.4)$$

where $\mathbf{e}(i)$ is the i^{th} standard basis vector.

Note that only one element of $\boldsymbol{\alpha}$ is updated per iteration, while all N elements of $\mathbf{F}$ are updated per iteration.

Alternative Derivation of Update Rule

Representing the problem with a loss function provides an alternative method for deriving the update rule and yields interesting insights. This loss function can be derived using the method of Lagrange multipliers, where we seek to minimize $\|\sum_i \phi(\mathcal{X}_i) \boldsymbol{\alpha}_i\|^2$ (equivalent to maximizing the distance of the projections of training points in the feature space) subject to the constraint

$\mathbf{y}_i f(\mathcal{X}_i) \geq 1 \forall \mathcal{X}_i \in \mathcal{X}$. Consider the maximum margin optimization problem:

$$\begin{aligned} \boldsymbol{\alpha}^* = \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \quad & \frac{1}{2} \left\| \sum_i \boldsymbol{\alpha}_i \phi(\mathcal{X}_i) \right\|^2 \\ \text{subject to } & \mathbf{y}_i \sum_j \boldsymbol{\alpha}_j k(\mathcal{X}_i, \mathcal{X}_j) \geq 1 \quad \forall i \end{aligned} \quad (2.5)$$

$$\begin{aligned} = \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \quad & \frac{1}{2} \boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\alpha} \\ \text{subject to } & \mathbf{1} - \mathbf{Y} \mathbf{K} \boldsymbol{\alpha} \leq \mathbf{0} \end{aligned} \quad (2.6)$$

where $\mathbf{Y} = \operatorname{diag}(\mathbf{y})$.

The Lagrangian is thus defined as

$$\begin{aligned} L(\boldsymbol{\alpha}, \boldsymbol{\lambda}) = \quad & \frac{1}{2} \boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\alpha} + \boldsymbol{\lambda}^\top (\mathbf{1} - \mathbf{Y} \mathbf{K} \boldsymbol{\alpha}) \\ \text{subject to } & \boldsymbol{\lambda}_i \geq 0 \quad \forall i. \end{aligned} \quad (2.7)$$

The dual problem is:

$$\boldsymbol{\lambda}^* = \underset{\boldsymbol{\lambda} \geq \mathbf{0}}{\operatorname{argmax}} \min_{\boldsymbol{\alpha}} L(\boldsymbol{\alpha}, \boldsymbol{\lambda}). \quad (2.8)$$

With no constraints on $\boldsymbol{\alpha}$, the minimum of $L(\boldsymbol{\alpha}, \boldsymbol{\lambda})$ given $\boldsymbol{\lambda}$ may be found by setting its gradient to $\mathbf{0}$:

$$\frac{\partial L(\boldsymbol{\alpha}, \boldsymbol{\lambda})}{\partial \boldsymbol{\alpha}} = \mathbf{K} \boldsymbol{\alpha} - \mathbf{K} \mathbf{Y} \boldsymbol{\lambda} = \mathbf{0} \quad (2.9)$$

$$\Rightarrow \boldsymbol{\alpha}^* = \mathbf{Y} \boldsymbol{\lambda} \quad (2.10)$$

$$\Rightarrow \min_{\boldsymbol{\alpha}} L(\boldsymbol{\alpha}, \boldsymbol{\lambda}) = -\frac{1}{2} \boldsymbol{\lambda}^\top \mathbf{Y} \mathbf{K} \mathbf{Y} \boldsymbol{\lambda} + \mathbf{1}^\top \boldsymbol{\lambda}. \quad (2.11)$$

Thus, the problem can be rewritten as

$$\lambda^* = \underset{\lambda \geq 0}{\operatorname{argmin}} \frac{1}{2} \lambda^\top \mathbf{Y} \mathbf{K} \mathbf{Y} \lambda - \mathbf{1}^\top \lambda \quad (2.12)$$

Replacing $\mathbf{Y} \lambda$ with α and realizing $\mathbf{Y} = \mathbf{Y}^{-1}$, the problem becomes

$$\alpha^* = \underset{\mathbf{Y} \alpha \geq 0}{\operatorname{argmin}} \frac{1}{2} \alpha^\top \mathbf{K} \alpha - \mathbf{1}^\top \mathbf{Y} \alpha \quad (2.13)$$

$$= \underset{\mathbf{Y} \alpha \geq 0}{\operatorname{argmin}} \frac{1}{2} \alpha^\top \mathbf{K} \alpha - \mathbf{y}^\top \alpha. \quad (2.14)$$

Thus, the loss function we should strive to minimize is

$$\begin{aligned} \mathcal{L}(\alpha) &= \frac{1}{2} \alpha^\top \mathbf{K} \alpha - \mathbf{y}^\top \alpha \\ &\text{subject to } \mathbf{y}_i \alpha_i \geq 0 \ \forall i. \end{aligned} \quad (2.15)$$

The loss function in Eq. 2.15 for our definition of the hypothesis function $f(\mathbf{x})$ is similar to that used in support vector machines [43], but the constraint when minimizing Eq. 2.15 is now $\mathbf{y}_i \alpha_i \geq 0 \ \forall i$.

A quadratic programmer may be used to minimize Eq. 2.15, but finding the optimal solution is undesired for computational effort and lack of sparsity. To improve the training time and sparsity of the model, Fastron takes a greedy coordinate descent approach and terminates when $m^{(n)}(\mathcal{X}_i) > 0 \ \forall \mathcal{X}_i \in \mathcal{X}$. Coordinate descent updates one element of $\alpha^{(n)}$ per iteration while leaving all other elements fixed [61]. On iteration n , coordinate descent minimizes $\mathcal{L}(\alpha^{(n)})$ along the i^{th} axis by setting $\alpha_i^{(n+1)}$ to the solution of $\frac{\partial \mathcal{L}(\alpha^{(n)})}{\partial \alpha_i^{(n)}} = 0$:

$$\alpha_i^{(n+1)} = y_i - \sum_{j \neq i} \mathbf{K}_{ij} \alpha_j^{(n)}. \quad (2.16)$$

Replacing $\alpha_i^{(n+1)}$ in Eq. 2.16 with $\alpha_i^{(n)} + \Delta\alpha_i^{(n)}$, we realize $\Delta\alpha_i^{(n)} = \mathbf{y}_i - \mathbf{F}_i^{(n)}$, matching the result in Eq. 2.2. It follows that $\alpha^{(n)}$ and $\mathbf{F}^{(n)}$ will be incremented as shown in Eq. 2.3 and 2.4.

Updating $\alpha_i^{(n)}$ changes $\mathcal{L}(\alpha^{(n)})$ as follows:

$$\mathcal{L}(\alpha^{(n+1)}) = \mathcal{L}(\alpha^{(n)} + \Delta\alpha_i^{(n)} \mathbf{e}(i)) \quad (2.17)$$

$$= \frac{1}{2} \alpha^{(n)\top} \mathbf{K} \alpha^{(n)} - \mathbf{y}^\top \alpha^{(n)} + \frac{1}{2} \Delta\alpha_i^{(n)^2} + \Delta\alpha_i^{(n)} \mathbf{F}_i^{(n)} - \Delta\alpha_i^{(n)} \mathbf{y}_i \quad (2.18)$$

$$= \mathcal{L}(\alpha^{(n)}) - \frac{1}{2} \Delta\alpha_i^{(n)^2}. \quad (2.19)$$

The coordinate descent direction is selected greedily, i.e., the $\alpha_i^{(n)}$ to update should be selected such that there is maximal decrease in loss. Additionally, since the algorithm will terminate when $m^{(n+1)}(\mathcal{X}_i) > 0 \forall \mathcal{X}_i \in \mathcal{X}$, the directions considered are restricted to only include directions where $m^{(n)}(\mathcal{X}_i) \leq 0$. Note that this restriction ensures that if $\alpha^{(n)}$ satisfies $\mathbf{y}_i \alpha_i^{(n)} \geq 0 \forall i$, then $\alpha^{(n+1)}$ satisfies $\mathbf{y}_i \alpha_i^{(n+1)} \geq 0 \forall i$. Initializing $\alpha_i^{(0)} = 0 \forall i$ would allow these conditions to be satisfied during the updates. Maximizing the decrease in loss given the negative margin restriction shows that greedy coordinate descent selects to update $\alpha_i^{(n)}$ according to

$$i = \operatorname{argmax}_{j: m^{(n)}(\mathcal{X}_j) \leq 0} \left(\frac{1}{2} \Delta\alpha_j^{(n)^2} \right) \quad (2.20)$$

$$= \operatorname{argmax}_{j: m^{(n)}(\mathcal{X}_j) \leq 0} \left(\left(\mathbf{y}_j - \mathbf{F}_j^{(n)} \right)^2 \right) \quad (2.21)$$

$$= \operatorname{argmax}_{j: m^{(n)}(\mathcal{X}_j) \leq 0} \left(1 - 2m^{(n)}(\mathcal{X}_j) + (m^{(n)}(\mathcal{X}_j))^2 \right) \quad (2.22)$$

$$= \operatorname{argmin}_j \left(m^{(n)}(\mathcal{X}_j) \right). \quad (2.23)$$

Eq. 2.23 shows that the training point with the most negative margin (intuitively, the

point farthest from the separating hyperplane on the wrong side) is prioritized when updating. The combination of the update rule in Eq. 2.16 and the descent direction in Eq. 2.23 guarantees $\mathcal{L}(\boldsymbol{\alpha}^{(n+1)}) \leq \mathcal{L}(\boldsymbol{\alpha}^{(n)}) - \frac{1}{2}$ because $\Delta \boldsymbol{\alpha}_i^{(n)^2} = (1 - m^{(n)}(\mathcal{X}_i))^2 \geq 1$ whenever there exists a misclassified training point, i.e., $m^{(n)}(\mathcal{X}_i) \leq 0$. In other words, each iteration guarantees a decrease in loss by at least $\frac{1}{2}$ whenever there are still training points with nonpositive margin. This guarantee allows us to show that the algorithm will always eventually yield a model with positive margin for all samples in the dataset.

Claim 1. *Minimization of $\mathcal{L}(\boldsymbol{\alpha})$ with the greedy coordinate descent rule defined in Eq. 2.16 and 2.23 will always eventually yield a hypothesis with positive margin for all samples given nonsingular Gram matrix $\mathbf{K}$.*

Proof. If $\min_i m^{(n)}(\mathcal{X}_i) \leq 0$, the upper bound on the change in loss per descent step is

$$\sup \left(\mathcal{L}(\boldsymbol{\alpha}^{(n+1)}) - \mathcal{L}(\boldsymbol{\alpha}^{(n)}) \right) = \sup \left(-\frac{1}{2}(1 - m^{(n)}(\mathcal{X}_i))^2 \right) = -\frac{1}{2}. \quad (2.24)$$

A lower bound of $\mathcal{L}(\boldsymbol{\alpha})$ is

$$\inf \mathcal{L}(\boldsymbol{\alpha}) = \mathcal{L}(\mathbf{K}^{-1}\mathbf{y}) = -\frac{1}{2}\mathbf{y}^\top \mathbf{K}^{-1}\mathbf{y} \quad (2.25)$$

for nonsingular $\mathbf{K}$. The margin is exactly 1 for all samples when $\boldsymbol{\alpha} = \mathbf{K}^{-1}\mathbf{y}$.

Thus, by combining Eq. 2.24 and 2.25, a loose upper bound on the number of descent steps required to reach $\mathcal{L}(\mathbf{K}^{-1}\mathbf{y})$ from initial loss $\mathcal{L}(\boldsymbol{\alpha}^{(0)})$ is $\frac{\mathcal{L}(\mathbf{K}^{-1}\mathbf{y}) - \mathcal{L}(\boldsymbol{\alpha}^{(0)})}{-\frac{1}{2}} = \mathbf{y}^\top \mathbf{K}^{-1}\mathbf{y} + 2\mathcal{L}(\boldsymbol{\alpha}^{(0)})$.

If $\min_i m^{(n)}(\mathcal{X}_i) > 0$, the hypothesis at iteration n successfully provides a positive margin for all samples. $\square$

Claim 1 means the weight update algorithm can terminate once all training samples have positive margin or will otherwise work toward achieving positive margin for all samples. As established in Section 1.4.2, $\mathbf{K}$ is a positive definite matrix if each configuration in $\mathcal{X}$ is unique

(which is the case when sampling from a continuous space) and using a positive definite kernel (such as the Gaussian or rational quadratic kernel), implying $\mathbf{K}$ is nonsingular in practice.

Assuming all weights start at 0, the upper bound on the number of training iterations is $\mathbf{y}^\top \mathbf{K}^{-1} \mathbf{y}$. Rayleigh quotients are defined as $R(\mathbf{x}) = \frac{\mathbf{x}^\top \mathbf{M} \mathbf{x}}{\mathbf{x}^\top \mathbf{x}}$ for $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{M} \in \mathbb{R}^{N \times N}$. When $\mathbf{M}$ is Hermitian, $R(\mathbf{x})$ is bound by the greatest eigenvalue of $\mathbf{M}$ [62]. The greatest eigenvalue of $\mathbf{K}^{-1}$ is λ_N^{-1} , where λ_N is the smallest eigenvalue of $\mathbf{K}$. Thus, $\mathbf{y}^\top \mathbf{K}^{-1} \mathbf{y} \leq \lambda_N^{-1} \mathbf{y}^\top \mathbf{y}$. Since $y_i = \pm 1$, another upper bound on the number of update iterations is $\lambda_N^{-1} N$. While this bound is looser than the other upper bound, it illustrates the role the eigenvalues of the Gram matrix play in convergence: for a Gram matrix with a larger minimum eigenvalue, the algorithm will converge quicker.

This upper bound makes sense intuitively because with larger eigenvalues, each datapoint is less correlated with other datapoints, thus making each update less likely to influence the value of another datapoint. Consider an extreme case where each datapoint is sufficiently far from each other such that $\mathbf{K}$ is approximately the identity matrix. In this case, $\lambda_N = 1$, which means the algorithm should converge in at most N iterations. This makes sense because in the worst case, each weight in $\boldsymbol{\alpha}$ is updated once to have positive margin.

While the upper bound $\lambda_N^{-1} N$ may be large, the total number of iterations required to satisfy $m(\mathcal{X}_i) > 0 \forall \mathcal{X}_i \in \mathcal{X}$ is typically much smaller, as will be shown empirically by the short training times in Section 2.3. In the case that training takes longer than desired, an iteration limit may be defined for early termination (defined as $iter_{max}$ in Algorithm 1) at the cost of yielding a classifier with lower accuracy. The iteration limit can be selected empirically by determining a limit that allows for convergence in most cases but still limits model update times as desired. We use $iter_{max} = 5000$ in all experiments in Section 2.3 as this number allowed convergence in almost all cases, including initial training where all weights start at 0.

2.2.2 Conditional Bias Parameter

As with the standard kernel perceptron, the Fastron model does not have an additive bias term. On the other hand, the kernel SVM model contains a bias term $b \in \mathbb{R}$ in its hypothesis: $f_{SVM}(\mathbf{x}) = f(\mathbf{x}) + b$. The benefit of including a bias term is that points with the label $\mathbf{y}_i = \text{sign}(b)$ are more likely to be classified correctly because the model is universally biased toward labeling query points as $\text{sign}(b)$.

As false negatives (mistaking C_{obs} for C_{free}) are more costly than false positives (mistaking C_{free} for C_{obs}) in the context of collision detection for motion planning, we would like to bias the model toward the C_{obs} label to err on the side of caution. Rather than trying to learn a bias term, we instead conditionally multiply the target values by a user-selected value during training such that C_{obs} points are more likely to be classified correctly. More specifically, we adjust the rule defined in Eq. 2.2 to

$$\Delta \boldsymbol{\alpha}_i^{(n)} = b_i \mathbf{y}_i - \mathbf{F}_i^{(n)} \quad (2.26)$$

where $b_i = \beta^{0.5(y_i+1)}$ (assuming the labels are ± 1) and $\beta \geq 1$ is a user-selected conditional bias parameter. When $\beta > 1$, weight updates are larger when correcting for a C_{obs} configuration

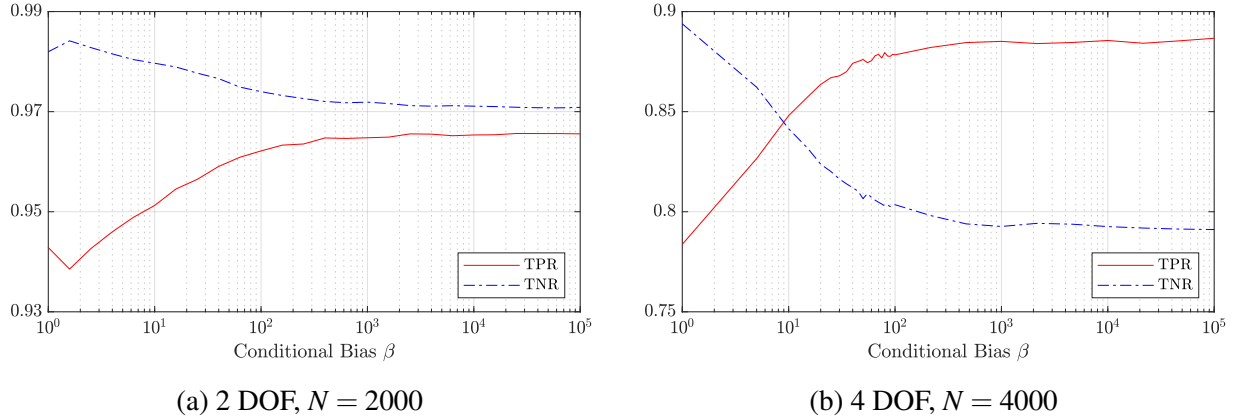


Figure 2.3: TPR and TNR of Fastron for various conditional bias β values for the 2 and 4 DOF robot shown in Fig. 2.4. The ground truth labels are provided using the GJK [5] collision detection method. Increasing the parameter improves TPR at the cost of TNR, but the effects taper off as β gets large.

compared to a C_{free} configuration. Larger weights for C_{obs} points would ultimately influence a larger neighborhood in C-space, thereby padding C-space obstacles and potentially increasing true positive rate.

Fig. 2.3 shows the effect of the conditional bias parameter on true positive rate (TPR) and true negative rate (TNR) (averaged over environments with randomly placed obstacles) for the 2 DOF and 4 DOF robots shown in Fig. 2.4, respectively. Increasing the parameter improves TPR at the cost of TNR, but the effects taper off as β gets large.

Note that an upper bound on the number of iterations still exists when including the conditional bias parameter. As the target values are now $b_i y_i$ instead of y_i , we can consider a modified loss function $\mathcal{L}(\alpha) = \frac{1}{2} \alpha^T K \alpha - y^T B \alpha$, where B is a diagonal matrix containing the b_i value for each y_i . Noting that $\Delta \alpha_i^{(n)^2} \geq 1$ for $\min_i m^{(n)}(X_i) \leq 0$, $\beta \geq 1$, and $\alpha^{(0)} = \mathbf{0}$, the new upper bound on iterations to achieve positive margin for all samples is $y^T B K^{-1} B y$. Once again, in practice, the total number of iterations to satisfy $m(X_i) > 0 \forall X_i \in \mathcal{X}$ is typically significantly smaller.

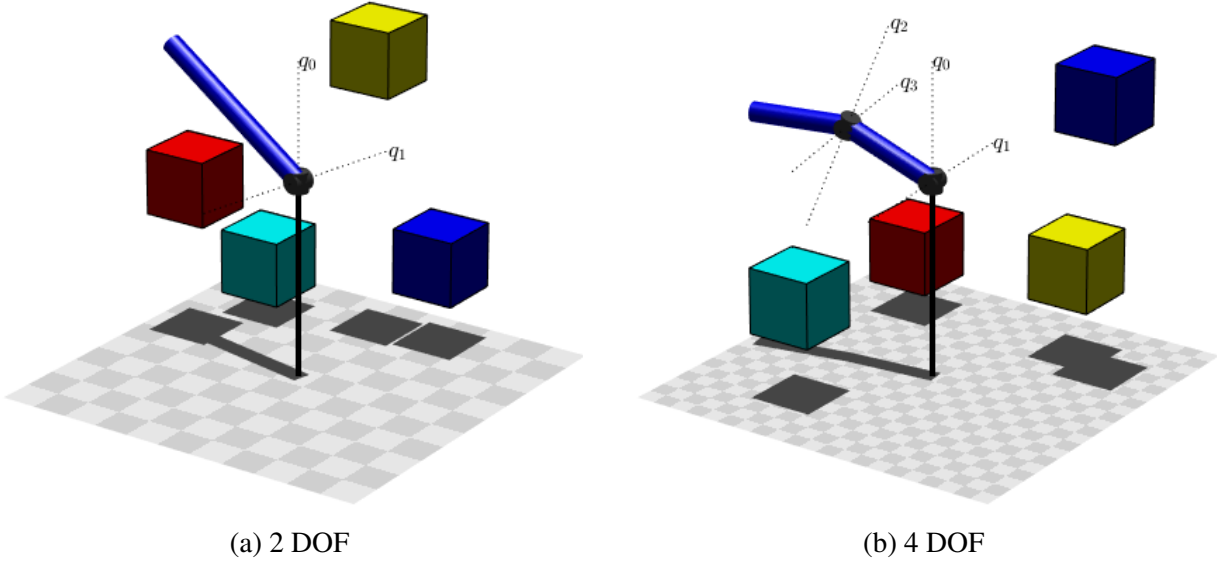


Figure 2.4: Example environments with up to 4 cube obstacles and simple 2 and 4 DOF robots. There are half as many links as there are joints, and each pair of joints is overlapping.

2.2.3 Redundant Support Point Removal

A support point whose margin would be positive even if it were not in the support set is redundant. Redundant support points should be removed from $\mathcal{S}$ (by setting its corresponding weight in $\boldsymbol{\alpha}$ to 0) to promote the sparsity of the model. Redundant support point removal is useful when the workspace obstacles move, causing the collision statuses of the points in $\mathcal{X}$ and the decision boundary to change. Removing outdated support points that no longer contribute to the shifted decision boundary is necessary to reduce the model complexity, allowing computational efficiency in changing environments through model sparsity.

To remove $\mathcal{X}_i$ from $\mathcal{S}$, the weight and training hypothesis vector updates are computed as

$$\boldsymbol{\alpha}^{(n+1)} = \boldsymbol{\alpha}^{(n)} - \alpha_i^{(n)} \mathbf{e}(i) \quad (2.27)$$

$$\mathbf{F}^{(n+1)} = \mathbf{F}^{(n)} - \alpha_i^{(n)} \mathbf{K} \mathbf{e}(i). \quad (2.28)$$

The resultant margin at point $\mathcal{X}_i$ if it were removed from the support set is $m^{(n+1)}(\mathcal{X}_i) = \mathbf{y}_i \sum_{j \neq i} \mathbf{K}_{ij} \boldsymbol{\alpha}_j^{(n)} = m^{(n)}(\mathcal{X}_i) - \mathbf{y}_i \boldsymbol{\alpha}_i^{(n)}$. Considering margin to be a measure of how well a point is classified, points are iteratively removed in decreasing order of positive resultant margin until $m^{(n)}(\mathcal{X}_i) - \mathbf{y}_i \boldsymbol{\alpha}_i^{(n)} < 0 \ \forall \mathcal{X}_i \in \mathcal{S}$, i.e., removing an additional support point will cause it to be misclassified. We perform redundant support point removal only after a hypothesis that yields positive margin for all training points is found, thus allowing Claim 1 to remain valid. Lines 13-17 of Algorithm 1 describe the redundant support point removal step.

Each redundant support point removal step clearly improves the sparsity of the model by 1. However, following similar steps required to find the change in loss in Eq. 2.19, the change in loss when removing support point $\mathcal{X}_i \in \mathcal{S}$ is $\mathcal{L}(\boldsymbol{\alpha}^{(n)} - \alpha_i \mathbf{e}(i)) - \mathcal{L}(\boldsymbol{\alpha}^{(n)}) = \frac{1}{2} \alpha_i^2 + (b_i \mathbf{y}_i - \mathbf{F}_i^{(n)}) \boldsymbol{\alpha}_i$. This change in loss may be positive or negative depending on the values of $\mathbf{y}_i$, $\mathbf{F}_i$, and $\boldsymbol{\alpha}_i$, showing that redundant support point removal can potentially step away from the optimal solution. Furthermore, note that redundant support point removal changes the margin at $\mathcal{X}_j$ when removing $\mathcal{X}_i$ from $\mathcal{S}$

as follows: $m^{(n+1)}(\mathcal{X}_j) = m^{(n)}(\mathcal{X}_j) - \mathbf{y}_j \mathbf{K}_{ij} \boldsymbol{\alpha}_i$. Clearly, the margin decreases for any $\mathcal{X}_j$ where $\mathbf{y}_j \boldsymbol{\alpha}_i > 0$, and the number of misclassified training points becomes nonzero if $m^{(n)}(\mathcal{X}_j) < \mathbf{y}_j \mathbf{K}_{ij} \boldsymbol{\alpha}_i$ for any $\mathcal{X}_j$. If training points become misclassified when a support point is removed, more weight updates are required to correct these points. In practice, a solution is still found quickly as will be shown in the training speed results in Section 2.3. As a safeguard, if the iteration limit is reached and the solution obtained before support point removal has fewer misclassifications than after removal, the more correct solution will be returned as shown on Line 20 in Algorithm 1.

2.2.4 Active Learning

When the environment changes, the true decision boundary will change. The labels for the training set must be updated using the collision checker prior to updating the model. Rather than resampling the entire space, which may result in many unnecessary collision checks, we can exploit our previous model to search for new information. Active learning is a methodology to query an oracle (the KCD routine) for additional information and is intended to reduce the amount of labeling needed to update a model [63].

Our active learning strategy determines a new set of configurations $\mathcal{A}$ to add to the dataset on which to perform collision checks. Collision checks are to be performed on all previous support points $\mathcal{S}$ and $\mathcal{A}$. Because active learning takes place after sparsifying the trained model, the number of collision checks that will be performed for the next model update is always $|\mathcal{S}| + \mathcal{A}_{max}$, where the $\mathcal{A}_{max}$ is the user-specified number of configurations to add to $\mathcal{A}$. As calls to the KCD may be slow, $\mathcal{A}_{max}$ should be selected to reduce label update times.

Fastron’s active learning strategy is conducted in two stages: exploitation and exploration. In the exploitation stage, up to κ Gaussian distributed random samples are generated near each support point and are added to $\mathcal{A}$ as shown on Line 5 in Algorithm 2. For simplicity, we sample using isotropic Gaussians with the variance in each direction $\sigma^2 = (2\gamma)^{-1}$. Up to $\kappa|\mathcal{S}|$ unlabeled configurations are added to $\mathcal{A}$ during the exploitation stage. The idea behind the exploitation stage

is to search for small changes in or improvements to C-space obstacles such as when workspace obstacles move an incremental amount or the C-space obstacle boundary may be defined more precisely.

If the number of points in $\mathcal{A}$ is less than $\mathcal{A}_{max}$, the exploration stage fills the rest of $\mathcal{A}$ with uniformly random samples generated in the C-space as shown on Line 7 in Algorithm 2. For certain robots such as manipulators, newly introduced workspace obstacles can cause new C-space obstacles to materialize in difficult-to-predict locations. The purpose of the random exploration stage is thus to search for these new or drastically different C-space obstacles.

2.2.5 Practical Considerations

Joint Limits

This algorithm is intended to be used with robots with joint limits so that there are bounds within which C-space samples may be generated. When working with isotropic kernels such as the Gaussian or rational quadratic kernels, configurations must be mapped such that the kernels affect the same proportion of each DOF. We choose to map the bounded d -dimensional joint space to a d -dimensional input space $[-1, 1]^d$. We choose $[-1, 1]^d$ as the input space because if the upper and lower limits for any joint are symmetric about 0, a joint position of 0 still maps to 0 in input space. If $\mathbf{q}_u$ and $\mathbf{q}_l$ are vectors containing the upper and lower joint limits, respectively, then the following formula can be applied to map a joint space configuration $\mathbf{q}_{joint}$ to an input space point $\mathbf{q}_{input}$:

$$\mathbf{q}_{input} = (2\mathbf{q}_{joint} - \mathbf{q}_u - \mathbf{q}_l) \oslash (\mathbf{q}_u - \mathbf{q}_l) \quad (2.29)$$

where $\oslash$ performs element-wise division.

Support Point Cap

Whenever a weight update increases the number of nonzero elements in α , another point is added to the support set $\mathcal{S}$. To limit the computation time during classification, a support point limit $\mathcal{S}_{max}$ may be defined to prevent $|\mathcal{S}|$ from growing too large.

During the update steps, if the worst margin occurs at $\mathcal{X}_i \in \mathcal{S}$, the weight update can proceed without consideration of $\mathcal{S}_{max}$ because $|\mathcal{S}|$ will not increase when adjusting the weight of a preexisting support point. On the other hand, if the worst margin occurs at $\mathcal{X}_i \notin \mathcal{S}$, $\mathcal{X}_i$ is added to $\mathcal{S}$ only if $|\mathcal{S}| < \mathcal{S}_{max}$. Otherwise, if $|\mathcal{S}| = \mathcal{S}_{max}$, there is an attempt to remove a redundant support point before continuing to train. If no point can be removed, training is terminated.

Lazy Gram Matrix Evaluation

An advantage of prioritizing the most negative margin rather than sequentially adjusting the weight for each point in $\mathcal{X}$ is that not all values of $\mathbf{K}$ are utilized. The Gram matrix may thus be evaluated lazily. For example, if the most negative margin occurs at point $\mathcal{X}_i$, then only the i^{th} column of $\mathbf{K}$ is required for the update. Once the i^{th} column of $\mathbf{K}$ is computed, it does not need to be recomputed if α_i needs to be adjusted in a later training iteration. `computeGramMatrixColumn()` in Algorithm 1 computes the i^{th} column of the Gram matrix if it has not been computed yet.

Dynamic Data Structures

Some data structures must be retained and updated throughout the lifetime of the algorithm to allow incremental updates between collision status label updates. The training dataset of configurations $\mathcal{X}$ and Gram matrix $\mathbf{K}$ are stored as two-dimensional arrays, while the weight vector α , hypothesis vector $\mathbf{F}$, and true collision status labels $\mathbf{y}$ are stored as one-dimensional arrays. The dimensionality of each of these data structures always depends on how many points are currently in $\mathcal{X}$.

After training is complete, non-support points are discarded from $\mathcal{X}$, and all other data

structures are shrunk accordingly. All sparsification happens on Line 21 in Algorithm 1. After active learning, $\mathcal{A}_{max}$ new points are added to $\mathcal{X}$, and all data structures must be resized accordingly. If repeated deallocation and reallocation of memory slows performance, a fixed amount of memory can be reserved for each data structure based on S_{max} .

Typically the values in $\mathbf{F}$ are determined incrementally. However, elements in $\mathbf{F}$ corresponding to the $\mathcal{A}_{max}$ new points from active learning cannot be set the same way. Instead, the columns in $\mathbf{K}$ that have been partially filled through lazy Gram matrix evaluation should first be completely filled. Next, the uninitialized elements of $\mathbf{F}$ should be calculated directly using the updated Gram matrix and nonzero weights in $\boldsymbol{\alpha}$.

2.3 Performance of Fastron

In this section, we compare the performance of Fastron against other learning-based techniques and geometric collision checkers. We begin with simple (2 DOF and 4 DOF) manipulators in static environments to compare the query times, model update times, model sizes, and model correctness. We then evaluate the same metrics in changing environments using 4, 6, and 7 DOF robot manipulators. The query times and accuracies are measured with a varying number of workspace obstacles to explore how workspace complexity influences performance. Finally, we apply Fastron as the collision checker in an array of motion planning algorithms to assess any improvement to planning time.

Measures of model correctness include the overall classification accuracy, true positive rate (TPR), and true negative rate (TNR), defined as

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + TN + FP} \quad (2.30)$$

$$\text{TPR} = \frac{TP}{TP + FN} \quad (2.31)$$

$$\text{TNR} = \frac{TN}{TN + FP} \quad (2.32)$$

where TP and FP represent the number of samples in the test set that are correctly and incorrectly predicted to be in C_{obs} , respectively, and TN and FN represent the same but for the C_{free} class. TPR and TNR are useful measures of correctness when the volumes of C_{obs} and C_{free} are unbalanced.

As measures of classification performance, we measure the average time required to perform a proxy collision check and the size of the model in terms of the number of support points $|S|$. As proxy checks using each of these models requires a weighted sum of kernel evaluations, proxy check times should be directly related to the number of support points and computational complexity of the type of kernel.

2.3.1 Performance in Static Environments

We compare the performance of Fastron against other machine learning-based methods in static environments. We begin by describing the alternative methods before describing the experimental setup. The quantitative results are supplied at the end. For the 2 DOF robot, numerous examples of workspaces and their corresponding approximated C-space are provided.

The metrics we measure for these static environments include query times, model update times, model sizes, and model correctness. Even though we are working with static environments, we compute training time to determine how long it takes for each algorithm to generate a model. We include training time to assess which algorithms would be suitable for changing environments. This training time includes both the time required to learn the weights for the model and the time required to label each point in the training set using a collision checker.

Descriptions of Alternative Methods

We compare the performance of decision boundary machine learning techniques for static environments. We compare Fastron with two state-of-the-art, kernel-based algorithms: incremental SVM [2] with active learning [3] and sparse SVM [4], which we refer to as ISVM

and SSVM, respectively. As with the Fastron model, we use $[-1, 1]^d$ as the input space and ± 1 as the labels for both ISVM and SSVM. We train and validate all methods with the GJK algorithm [5], a standard for collision detection for convex polyhedra.

Pan et al. use ISVM to create an accurate C-space model [3, 64]. Starting by fitting an SVM model to a small set of labeled configurations, Pan et al. iteratively improve upon this initial model using an active learning approach which randomly selects either exploration sampling or exploitation sampling when seeking new points to add to the training set. Exploration requires generating uniformly random samples, while exploitation generates new points between support points of opposite labels. Note that the purpose for Pan et al.’s active learning strategy is to improve the precision of the model for a static C-space, while Fastron’s active learning strategy is meant to search for changes in C-space when the workspace changes. The workspace is not changing for the experiments in this section, so Fastron’s active learning strategy is not applied here.

While SSVM has not yet been applied to C-space approximation, we include SSVM for comparison because it attempts to minimize the L_0 -norm of the weight vector, i.e., the number of support points. L_0 -norm minimization is advantageous because classification time is directly dependent on the number of support points in the model. SSVM approximates the L_0 -norm by using a weighted L_2 -norm of model weights α , $\alpha^\top \Lambda \alpha$, where Λ is a diagonal matrix. The elements of Λ are set to α_i^{-2} for $|\alpha_i| > \epsilon$ and to ϵ^{-2} otherwise, where ϵ is a small positive value. Including this approximated L_0 -norm into the objective function forces many elements of α to approach 0. Points whose α_i approach 0 are removed from the training set before repeating the optimization.

Description of Experiment

For simplicity, in this set of tests we implemented all algorithms in MATLAB and assume that each algorithm’s standing will remain the same when transferred to a compiled language. For

the training portion of ISVM, we use Diehl et al.’s MATLAB implementation of the algorithm [2]. Furthermore, note that Pan et al. [3] train on pairs of objects, which will require multiple models when working with robot arms. As we are interested in comparing the performance of single models, we instead train one incremental SVM model on collision status labels generated for the entire robot arm. For training the SSVM, we use MATLAB’s `quadprog()` to solve the more efficient dual problem described by Huang et al. [4]. To further improve the training time of SSVM, we terminate training when the number of support points stops decreasing. In our experience, other than training time, no other metric seemed largely affected compared to the stopping rule provided by Huang et al. (i.e., terminate when the change in α is less than 10^{-4}). Our interpretations of the training of SSVM and training and active learning of ISVM are provided in Algorithms 3 and 4, respectively. For Algorithm 4, `incSvmUpdate()` performs the incremental SVM update according to Diehl et al. [2], `rand()` generates a random number between 0 and 1, `randPick()` selects a sample from a set, `colCheck()` performs geometry- and kinematics-based collision checks, and `errRate()` determines the proportion of misclassifications of a given set.

Ground truth collision checking is performed using the GJK algorithm [5]. We work with cube obstacles and fit the tightest possible bounding box around each link of the robot. Robot kinematics and visualization are performed using Peter Corke’s MATLAB Robotics Toolbox [65].

We begin with a simple 2 DOF robot whose body is a single rod whose yaw and pitch may be controlled. Up to 4 obstacles are randomly placed in the environment. Fig. 2.4a shows an example test environment. We repeat the experiments using a 4 DOF robot, which is created by concatenating two of the 2 DOF robots as shown in Fig. 2.4b.

Fig. 2.5 provides TPR, TNR, and model sizes (metrics demonstrating classification accuracy and speed) for Fastron for various γ values in randomly generated environments to motivate our choices for the γ values. We select $\gamma = 30$ for the 2 DOF case and $\gamma = 10$ for the 4 DOF case as these choices seem to provide an adequate balance between TPR/TNR and model

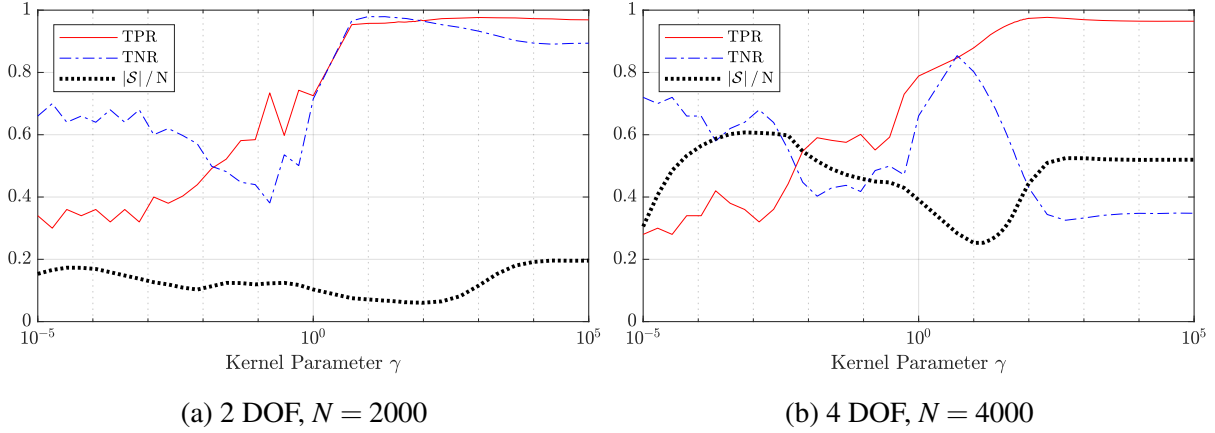


Figure 2.5: TPR, TNR, and model sizes (as a ratio of number of support points to training set size, $|S|/N$) of Fastron for various γ values for the 2 and 4 DOF robots shown in Fig. 2.4. The ground truth labels are provided using the GJK [5] collision detection method. These curves motivate the choices in γ for the 2 DOF and 4 DOF case in which large TPR and TNR are desired while keeping support set sizes as small as possible.

size. A smaller γ for the 4 DOF case makes sense as a wider kernel is needed to account for the larger C-space. These choices of γ values also worked well for ISVM and SSVM.

The rational quadratic kernel is used for Fastron and SSVM, and the Gaussian kernel is used for ISVM as Pan et al. use [3]. Both Fastron and SSVM are trained on a uniformly random set of samples in the input space. The incremental SVM uses the active learning approach to build its training set [3]. Each algorithm is ultimately trained on $N = 2000$ samples for the 2 DOF case, and $N = 4000$ samples for the 4 DOF case. The 4 DOF case is undersampled because larger dataset sizes become computationally intractable for the comparison methods using our MATLAB implementations.

Results for 2 DOF Robot

Fig. 2.6a and Table 2.1a provide comparisons of three machine learning methods trained on an environment with up to 4 obstacles for a 2 DOF case. All methods have comparable accuracy, TPR, and TNR. All methods performed significantly faster than GJK, which took on average $105.1 \mu\text{s}$ per collision check. Of the machine learning based methods, SSVM had the

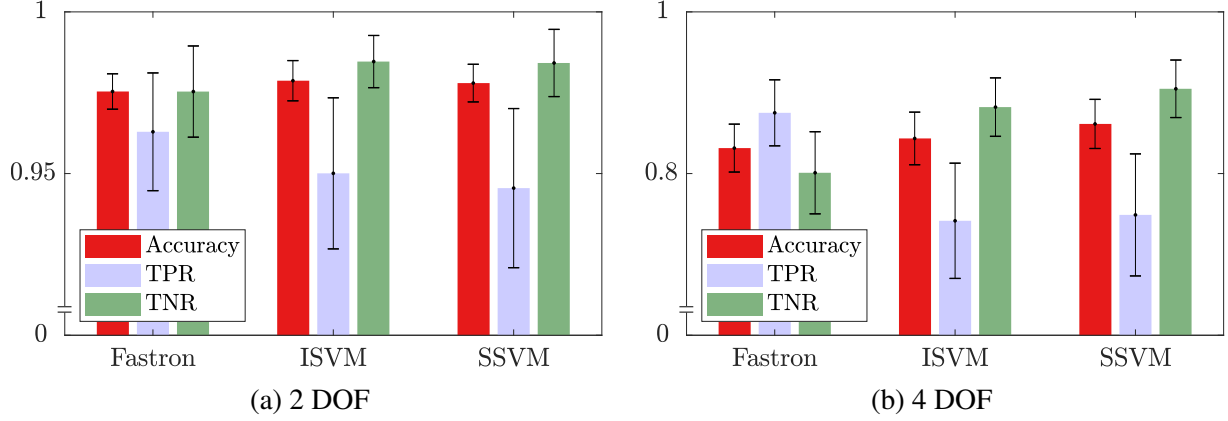


Figure 2.6: Accuracy, TPR, and TNR (higher is better) for Fastron, ISVM [2] with active learning [3], and SSVM [4] in a static environment using the 2 DOF and 4 DOF robots shown in Fig. 2.4. GJK [5] is used as the ground truth collision detection method.

sparsest solution, which results in it providing the fastest proxy collision checks. Furthermore, ISVM has on average more than 6 times the number of support points probably because more support points are placed near the boundary after its active learning procedure. Due to its larger number of support points and slower kernel, ISVM has the slowest proxy collision check timings.

Table 2.1: Performance of Fastron in a static environment for the 2 DOF and 4 DOF robots shown in Fig. 2.4 compared against incremental SVM (ISVM) [2, 3], sparse SVM (SSVM) [4], and the ground truth collision detection method GJK [5]. Lower is better for model size $|S|$, query time, and model training time. The best results are shown in bold.

Method	$ S $	Query Time (μs)	Training Time (s)
Fastron	133.6 ± 32.4	0.42 ± 0.09	0.23 ± 0.04
ISVM [2, 3]	349.1 ± 128.6	1.78 ± 0.62	2.72 ± 1.60
SSVM [4]	52.4 ± 11.3	0.22 ± 0.05	18.8 ± 2.69
FK + GJK	—	105.1 ± 16.5	—
(a) 2 DOF			
Method	$ S $	Query Time (μs)	Training Time (s)
Fastron	1084.1 ± 206.9	3.4 ± 0.6	0.71 ± 0.08
ISVM [2, 3]	1078.1 ± 125.8	5.8 ± 0.7	75.5 ± 20.6
SSVM [4]	509.8 ± 106.3	1.5 ± 0.3	193.7 ± 33.6
FK + GJK	—	166.6 ± 20.0	—
(b) 4 DOF			

SSVM takes nearly 100 times longer than Fastron to train, the fastest among all tested methods. Profiling the code suggests the slowest parts of SSVM and ISVM is having to repeatedly solve an optimization problem. Fastron’s speed in training compared to the SVM methods is due to not having to optimize an objective function completely and not requiring the entire Gram matrix.

Visualization of the C-space is straightforward when working with a 2 DOF robot and allows qualitative comparison. Fig. 2.7b shows example ground truth C-spaces (with axes scaled to the input space) for the workspaces shown in Fig. 2.7a along with approximations provided by the three kernel-based methods in Fig. 2.7c-2.7e. Examining the placement of support points, it is apparent that support points are typically placed closest to the boundary for ISVM except when some C-space obstacles are large. On the other hand, support point placement is much farther from the boundary for SSVM.

Results for 4 DOF Robot

Fig. 2.6b and Table 2.1b provide a comparison of the three machine learning methods trained on an environment with up to 4 obstacles. ISVM and SSVM have comparable accuracy, TPR, and TNR. On the other hand, Fastron has slightly less accuracy than the other methods, but has significantly more TPR and less TNR. Higher TPR is advantageous for collision detection where a more conservative prediction is preferred. We anticipate all methods would improve in terms of accuracy, TPR, and TNR if given more training data.

As with the 2 DOF case, all methods performed significantly faster than GJK, which took on average $166.6 \mu\text{s}$ per collision check. SSVM once again had the sparsest solution, requiring half as many support points as the other methods. SSVM’s sparsest solution also makes it provide the fastest predictions. Fastron and ISVM require approximately the same number of support points, but Fastron classifies faster due to its cheaper kernel.

SSVM takes the longest to train, while Fastron trains at least 200 times faster. Once again,

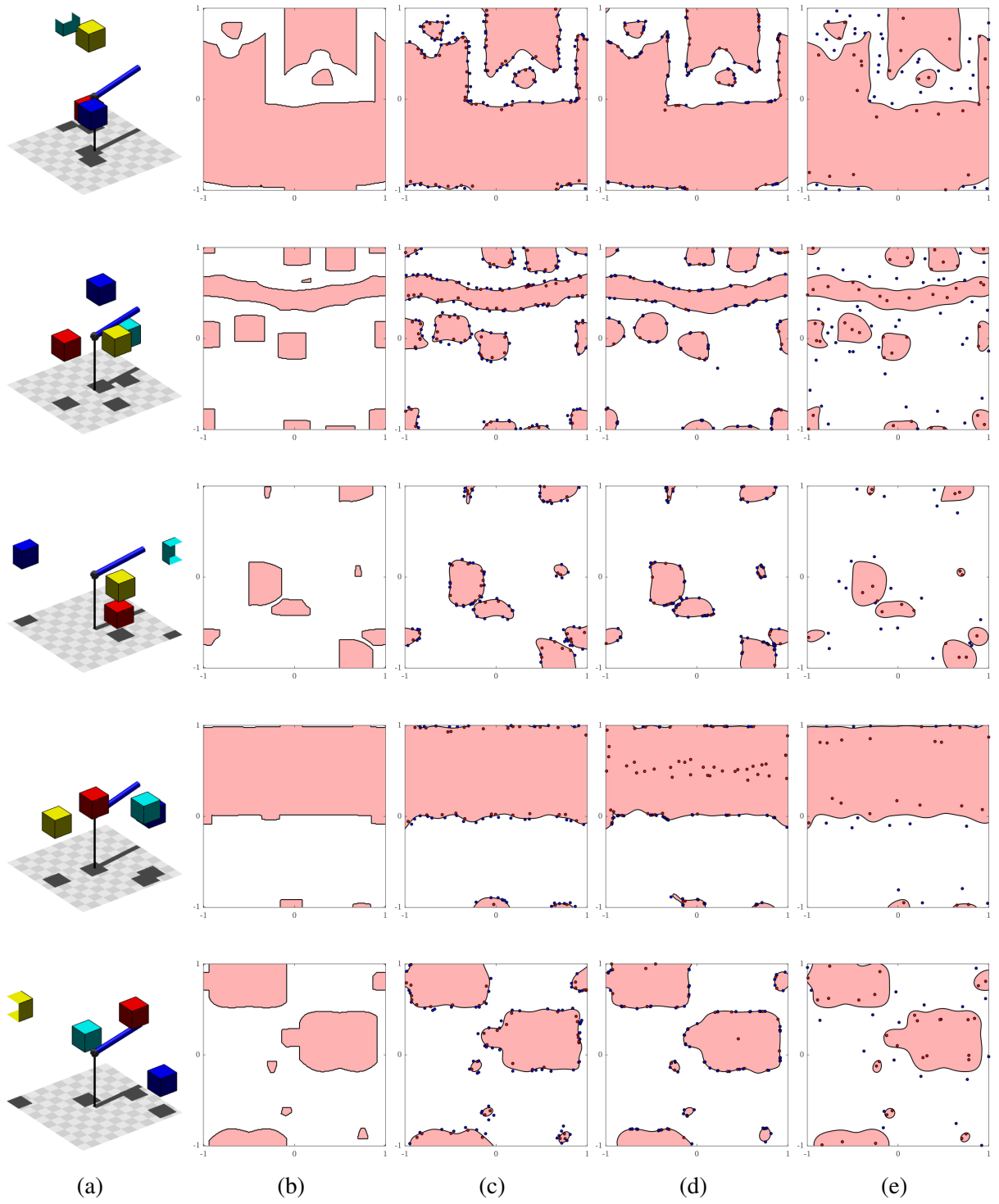


Figure 2.7: (a) Example static environments and C-space representations using (b) GJK collision checker [5], (c) Fastron [6], (d) incremental support vector machines [2, 3], and (e) sparse support vector machines [4] for a simple 2 DOF robot.

Fastron’s training speed is due to not having to optimize an objective function completely and not requiring the entire Gram matrix.

2.3.2 Performance in Changing Environments

Description of Experiment and Metrics

We test the Fastron algorithm in changing environments using C++. As Fastron is the only learning-based method that globally models C-space for changing environments to our knowledge, we do not include comparisons to other machine learning methods. Furthermore, the training timings for ISVM and SSVM from the static case suggest that our implementations of these methods are not suited for changing environments. Instead, we compare Fastron to two collision detection methods: GJK and the Flexible Collision Library (FCL) [8], the default collision library for the MoveIt! motion planning framework [66]. We use FCL as the ground truth labels for these tests, and use GJK [5] as an approximate geometry method. We train Fastron on labels from FCL and GJK and include the results for both variations.

The experiments involve moving boxes around the reachable workspace of a Baxter robot’s right arm. For GJK, we use cylinders as approximations for the geometry of each link in the arm. Robot kinematics and visualization are performed using MoveIt! [66] and rviz [67] in ROS [68]. We anticipate that having only one workspace obstacle (such as the workspace shown on the left in Fig. 2.8) would be the most challenging case for Fastron due to a misbalance of class sizes in C-space. On the other hand, one workspace obstacle would be the easiest case for geometry- and kinematics-based collision detectors whose performance is dependent on the number of objects in the workspace. Thus, most analysis in the following experiments involve one workspace obstacle, but results from multiple obstacle workspaces (such as that shown on the right in Fig. 2.8) are also included.

For the experiments with one obstacle, we perform three versions of the tests: using the

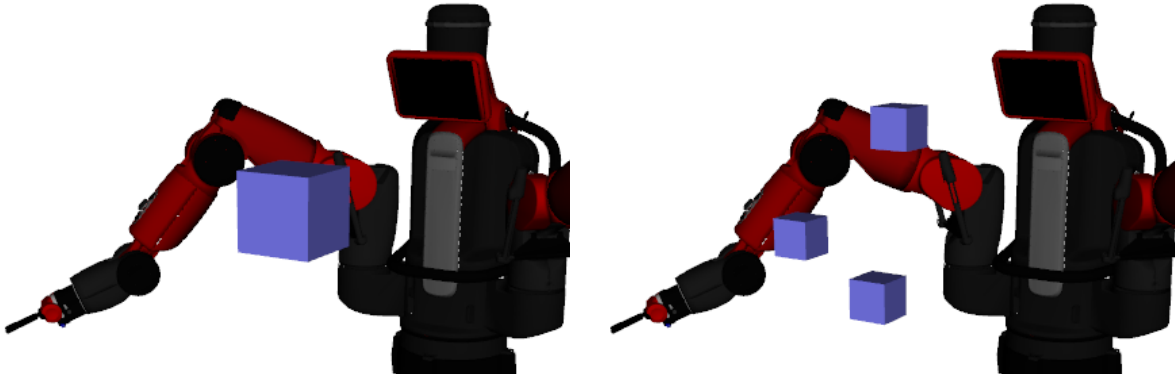


Figure 2.8: Baxter robot with cube obstacles used for collision detection and motion planning experiments.

first 4 DOF of the arm (excluding all wrist motions), the first 6 DOF (excluding gripper rotation), and all 7 DOF. For the 4 DOF and 6 DOF experiments, we perform full collision checking on the arm, but leave the unactuated joints' positions fixed at 0. We only consider the 7 DOF version when using workspaces with multiple obstacles.

We used a coarse grid search to select the parameters for Fastron. We use $\gamma = 10$ for the 4 DOF case and $\gamma = 5$ for the higher DOF cases. The conditional bias parameter is $\beta = 100$ for the 4 DOF case and $\beta = 500$ for the higher DOF cases. We use 10000 points in our initial dataset. Our active learning strategy found and labeled 500 additional points for each case. Despite needing only approximately 1400 update iterations per model update on average, we set $iter_{max} = 5000$ to allow sufficient time for convergence in case some datasets are more difficult to model.

As with the static case, we compute accuracy, TPR, and TNR to measure the correctness against FCL of each approximation method. Additionally, we include model size for Fastron and the query timing of all methods. FK is included in the query timing for both GJK and FCL. Finally, we include the time required to update the Fastron model given the previously trained model.

Table 2.2: Performance in a changing environment when actuating the first 4 DOF, the first 6 DOF, and all 7 DOF of the Baxter robot’s right arm, respectively, shown in Fig. 2.8. Lower is better for model size $|S|$, query time, and model update time. The best results are shown in bold.

Method	$ S $	Query Time (μs)	Update Time (ms)
Fastron (FCL)	638.6 ± 139.4	2.1 ± 0.5	42.0 ± 5.8
Fastron (GJK)	558.1 ± 151.1	1.9 ± 0.5	15.8 ± 3.8
FK + GJK [5]	—	9.1 ± 0.2	—
FK + FCL [8]	—	29.2 ± 0.5	—
(a) 4 DOF			
Method	$ S $	Query Time (μs)	Update Time (ms)
Fastron (FCL)	1060.5 ± 217.9	4.1 ± 0.8	62.4 ± 10.4
Fastron (GJK)	886.8 ± 215.5	3.5 ± 0.9	24.2 ± 6.4
FK + GJK [5]	—	9.5 ± 0.2	—
FK + FCL [8]	—	30.5 ± 0.5	—
(b) 6 DOF			
Method	$ S $	Query Time (μs)	Update Time (ms)
Fastron (FCL)	1306.0 ± 291.7	5.5 ± 1.2	74.7 ± 15.7
Fastron (GJK)	1151.3 ± 283.9	4.9 ± 1.2	32.6 ± 7.5
FK + GJK [5]	—	9.6 ± 0.4	—
FK + FCL [8]	—	29.3 ± 0.5	—
(c) 7 DOF			

Results for 4, 6, and 7 DOF Manipulators

Fig. 2.9 and Table 2.2 provide comparisons of Fastron (trained on FCL and GJK) to the geometry-based collision detectors for the 4, 6, and 7 DOF cases, respectively, when using one workspace obstacle. In all cases, GJK had the highest accuracy and TNR. The TPR of Fastron trained on FCL almost meets that of GJK, but only beats GJK for the 4 DOF case. Accuracy and TNR are approximately the same and TPR is lower than the other metrics for these cases because C_{obs} only occupies approximately 5-7% of the C-space in these experiments.

Comparing Fastron trained on FCL to Fastron trained on GJK, we can see a noticeable improvement in update time. The update rate increases from around 24 to 63 Hz for the 4 DOF

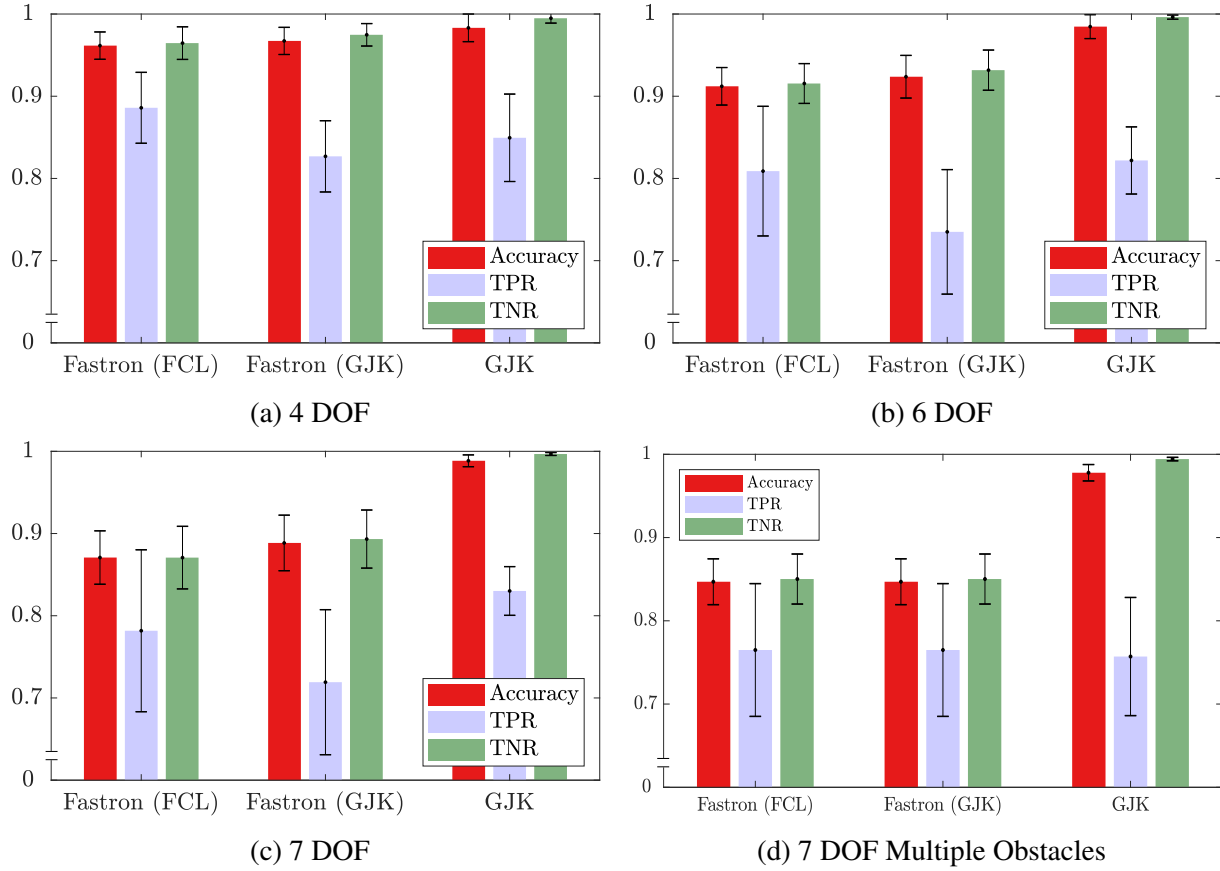


Figure 2.9: (a-c) Accuracy, TPR, and TNR (higher is better) for Fastron and GJK [5] in a continuously changing environment using the Baxter’s right arm and one workspace obstacle as shown in Fig. 2.8. Fastron is trained on both FCL [8] and GJK. FCL is used as the ground truth collision detection method. The three cases correspond to actuating (a) the first 4 DOF, (b) the first 6 DOF, and (c) all 7 DOF of the arm, respectively. Note that the GJK results do not change significantly because the entire arm is always used for geometry-based collision detection while the dimensionality of the space that Fastron models equals the number of DOF. (d) Accuracy, TPR, and TNR in a continuously changing environment using one of the Baxter’s 7 DOF arms and three workspace obstacles as shown in Fig. 2.8.

case, from 16 to 41 Hz for the 6 DOF case, and from 13 to 30 Hz for the 7 DOF case. The reason for the drastic decrease in update time is obvious: as a large portion of time to update the model is spent on performing collision checks, using a cheaper collision detector for labeling would decrease the update time. On the other hand, we also see that the TPR of Fastron (GJK) is lower compared to that of Fastron (FCL), probably because training on the approximate collision detection method causes some missed detections that would only be detected with a more precise

Table 2.3: Performance in a changing environment with three moving obstacles when actuating all 7 DOF of the Baxter robot’s right arm, respectively, shown in Fig. 2.8. Lower is better for model size $|S|$, query time, and model update time. The best results are shown in bold.

Method	$ S $	Query Time (μs)	Update Time (ms)
Fastron (FCL)	1637.5 ± 305.9	6.7 ± 1.2	89.2 ± 17.4
Fastron (GJK)	1582.9 ± 321.1	6.6 ± 1.4	56.3 ± 14.5
FK + GJK [5]	—	15.3 ± 1.6	—
FK + FCL [8]	—	29.1 ± 0.8	—

detector. These missed detections may also explain why Fastron (GJK) requires fewer support points on average than Fastron (FCL).

As the degrees of freedom increase, we notice the accuracies of Fastron decrease while model sizes, query times, and update times increase. This is due to the fact that higher dimensional spaces requires more data to correctly model. GJK and FCL are unaffected by the change in DOF because we always need to use the entire arm for collision detections.

Table 2.3 provides model sizes, query times, and update times for the full 7 DOF arm when there are three workspace obstacles. Comparing Table 2.3 to Table 2.2c, we can see that Fastron query times increase marginally when the number of workspace obstacles increase because the number of support points required to model the corresponding C-space is higher. We can see that GJK suffers when the number of workspace obstacles increase while FCL does not significantly change. Consequently, while update times increase for both versions of Fastron with more workspace obstacles, the update times increase more for the GJK-based Fastron than for the FCL-based Fastron.

2.3.3 Performance in Motion Planning Application

Description of Experiment

As motion planning is one application that requires frequent collision checks, we apply Fastron to motion planning for one of the Baxter arms to see the effect on the computation time

required to generate a feasible plan. We use the same environments methods as used in Section 2.3.2. As proxy collision detection timings were roughly the same for both Fastron (FCL) and Fastron (GJK), we only provide results for Fastron (FCL) in this section.

The experiments involve using the various collision detection methods in standard motion planners. Each trial involved incrementally moving the obstacles in the robot’s reachable workspace and generating a new motion plan (using each collision detection method) from scratch each time the obstacle is in a new position. For each collision checking method, we perform 50 trials, where a trial consists of up to 30 random movements of the workspace obstacle, and the selected motion planner generates 10 plans for each obstacle position. The start and goal configurations are randomly generated such that the arm must move from one side of the workspace to the other, and are only regenerated after a motion plan has been generated using each collision detection method. The OMPL library [69] is used to handle the motion planning, and we fill in the state validity checking routine with one of the three collision checking methods. We select RRT [1], RRT-Connect [30], RRT* [31], and BIT* [33] to demonstrate the performance of each collision detection method. RRT and its bidirectional variant RRT-Connect are probabilistically complete motion planners [24] that terminate once a path is found. RRT* and BIT* are optimizing planners that usually continue to search C-space for a shorter path after an initial feasible path has been found. As we are interested more in generating feasible plans than optimal plans, we terminate RRT* and BIT* once a feasible plan is found, which means the resulting path may not be close to optimal. We include these results to show how quickly the initial path can be found upon which RRT* and BIT* may improve.

Since Fastron and GJK are approximations to true collision detection, we anticipate that the plans that are generated with these approximate methods may actually include some C_{obs} waypoints. Thus, for Fastron- and GJK-based motion planning, we include verification and repair steps to correct any plan that includes invalid waypoints. In the verification step, we check each waypoint of the motion plan using FCL. In the repair step, invalid segments are first excised out

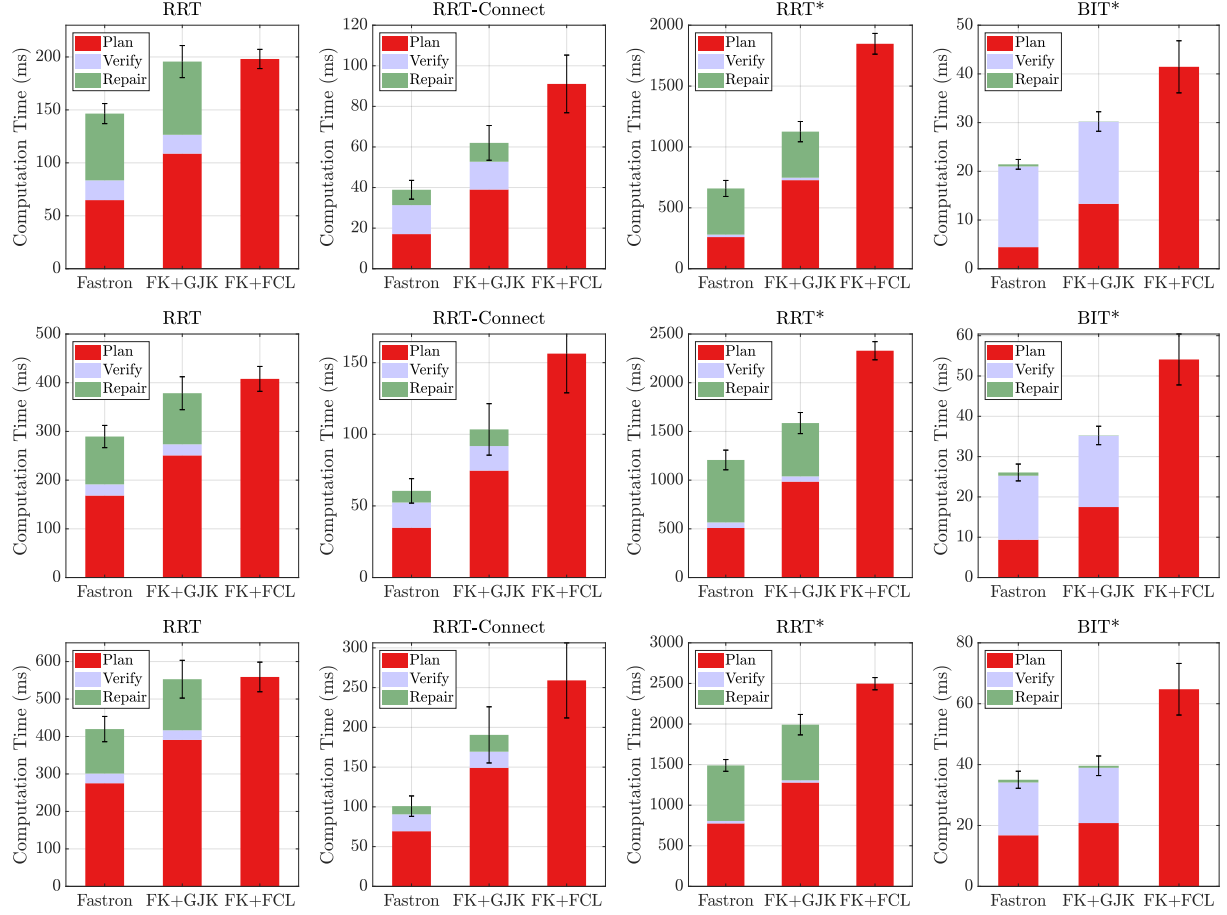


Figure 2.10: Timings for motion planning for the Baxter robot’s right arm with one workspace obstacle using RRT, RRT-Connect, RRT*, and BIT* using Fastron, GJK, and FCL for collision detection. The three rows correspond to planning using only the first 4 DOF, only the first 6 DOF, and all 7 DOF of the arm, respectively. Forward kinematics (FK) are included in the timings for the geometry- and kinematics-based collision detectors. The error bars denote the bounds of the 95% confidence intervals for the mean motion plan times.

of the motion plan, and then the same motion planner (with FCL as the collision detector) is called again to bridge the excised part of the motion plan. The combination of the verify and repair steps ensures that the final plan is truly collision-free, assuming a motion plan is found.

Results for Planning with 4, 6, and 7 DOF

Fig. 2.10 shows the timing results for the various motion planners and different number of actuated DOF. It is clear that Fastron yields the fastest motion plans (up to 4 times faster for the 7

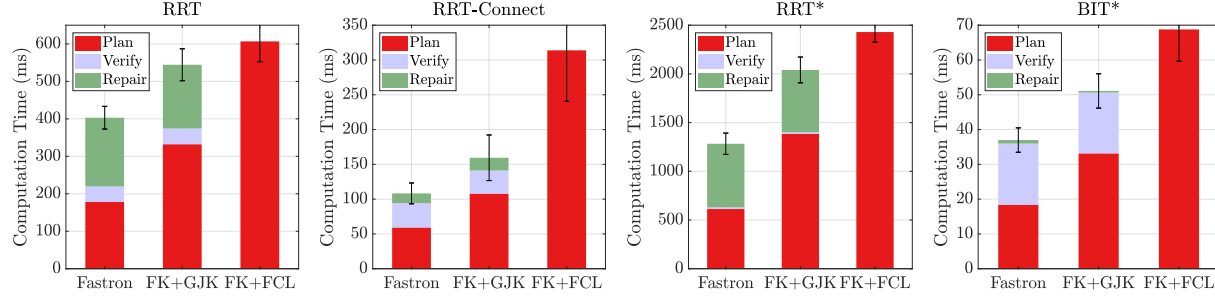


Figure 2.11: Timings for motion planning for the Baxter robot’s 7 DOF right arm with three workspace obstacles using RRT, RRT-Connect, RRT*, and BIT* using Fastron, GJK, and FCL for collision detection. Forward kinematics (FK) are included in the timings for the geometry- and kinematics-based collision detectors. The error bars denote the bounds of the 95% confidence intervals for the mean motion plan times.

DOF case), and GJK provides the second fastest. Even with the verify and repair steps, Fastron provides the fastest solutions. The verification and repair steps take approximately the same time for both Fastron- and GJK-based motion planning in all cases, so the speed improvement of Fastron over GJK is due to its initial plan. Using GJK in place of FCL did not seem to significantly improve the timing for RRT when including the verify and repair steps, but the timings did improve for the other motion planning methods.

Since RRT* typically took a long time to generate feasible plan using any of the motion planning methods, the verification step took a negligible amount of time compared to the planning and repairing time for both Fastron- and GJK-based planning. On the other hand, planning and repairing is fast when using BIT*, so verification appears to take a larger proportion of time for Fastron- and GJK-based planning.

The motion planning times with three workspace obstacles are shown in Fig. 2.11. Once again, even with the verify and repair steps, using Fastron for collision checking yields the fastest collision-free motion plans. The speed improvement of Fastron is due to its initial planning time, which is up to 6 times faster compared to when using FCL.

As the total amount of planning time is lowest for Fastron for all tested motion planning algorithms, we anticipate that using Fastron will improve planning times for other sampling-based

motion planners as well and may provide a fast initial path for optimizing planners.

2.4 Further Applications

2.4.1 Pick and Place

We apply the Fastron algorithm for a pick and place task using a Universal Robots UR3 robot and a shelf. This robotic system is designed for tabletop or workbench usage, possibly in the presence of humans, thereby causing a changing environment. In our this case, we have the end effector reach from a lower shelf to a higher shelf (without hitting the shelf), as shown in Fig. 2.12a. We do not simulate the manipulation part of the task and thus no gripper is shown.

We generate motion plans for the 6 DOF robot using an RRT. In a MATLAB environment, solving an RRT took on average 334.7 ms when using GJK for collision detections. When using Fastron for collision checking, solving an RRT took on average 119.7 ms, which includes verification and repairing where needed. These timings show that Fastron provides almost a threefold speed improvement for this application of motion planning.

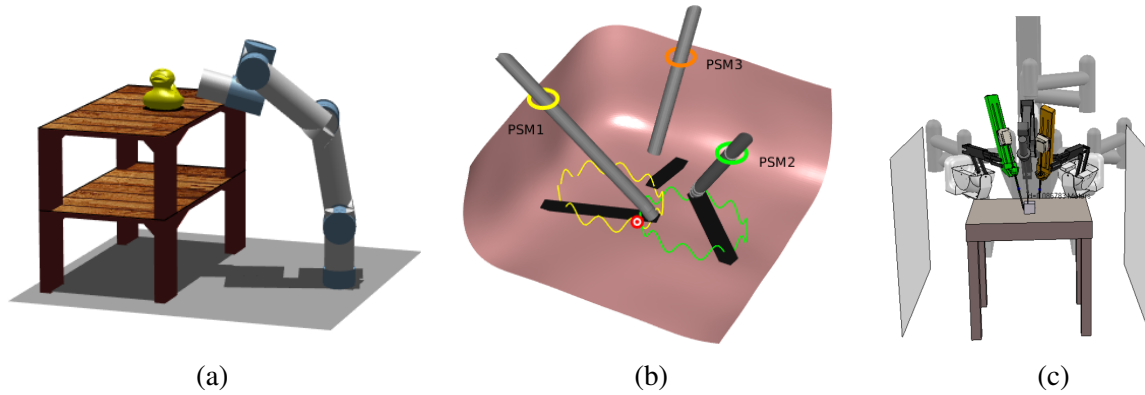


Figure 2.12: (a) Pick and place task using a UR3 in simulation. RRTs using Fastron for collision detection are used for motion planning. (b) An autonomous assistant (PSM3) reaching a target among human-controlled manipulators (PSM1 and PSM2) in da Vinci® Surgical System simulation. RRTs using Fastron for collision detection are used for motion planning. (c) Optimized placement of da Vinci® PSM base positions.

2.4.2 Autonomous Surgical Assistant

We apply the Fastron algorithm in the automation of an autonomous surgical assistant in the context of robot-assisted surgery using a da Vinci® Surgical System. For this simulation, we assume that two of the three patient-side manipulators (PSMs) are controlled by a human surgeon, while the third PSM must autonomously perform a task without colliding with the surgeon’s movements. Example tasks that this type of assistant may perform autonomously or under supervision of a human include suction, irrigation [70], cautery [71], and ultrasound transducer placement [72].

For our scenario, we let PSM1 and PSM2 execute volatile motions that sometimes impede PSM3 from reaching a target configuration. Fig. 2.12b shows an example where PSM1 and PSM2 follow the circular paths while PSM3 attempts to reach the target between the paths. All PSMs are constrained by a remote center of motion, represented as a ring constraint (yellow, green, and orange). PSM3 is disallowed from colliding with the surgeon-controlled arms and any surrounding tissues. In a real system (such as a first-generation da Vinci® Surgical System with da Vinci Research Kit boxes [73]), joint positions of all PSMs will be known and locations of surrounding tissues may be known via preoperative or intraoperative imaging. When excluding wristed motions, PSM3’s controllable DOF include its pitch, yaw, and insertion, meaning we are working with 3-dimensional Fastron models. Excluding the wristed motion is permissible when controlling PSM3 because only three DOF are required to reach the target. As Fastron is based on snapshots of workspace, we use a padded version of PSM3 to improve its responsiveness to PSM1 and PSM2’s motions.

While numerous motion planning methods may be chosen to dictate PSM3’s motions, we are able to repeatedly solve an RRT from scratch at a 56 Hz rate (ignoring cases where PSM3 is already at its target and no planning is required) when using Fastron for collision detection in a MATLAB environment. In comparison, RRTs can be solved at a 10 Hz rate when using GJK for collision detection, demonstrating the Fastron enables a more responsive autonomous assistant.

2.4.3 Optimal Surgical Arm Placement

Fastron is also used in the acceleration of determining the optimal placement of da Vinci® PSMs. While robotic laparoscopic surgery offers important advantages to the primary surgeon over manual laparoscopic surgery including improved ergonomics and dexterity, the sizable robot occupies a large portion of the sterile part of the operating room [74]. Key elements of successful design for surgical robotic systems include sufficient workability for the surgeon inside the patient and adaptability to any operating room [75]. While the da Vinci Surgical System can satisfy these requirements, trained staff are currently required to enforce these requirements by deciding and implementing the arm placement during preoperative setup procedures. Preoperative setup contributes to one of the limitations of robotic procedures, which is the time required to set up the system [76, 77]. The considerations the surgical staff take during arm placement including minimizing collisions of the arms with other arms, the endoscopic camera, and other obstacles in the operating room while still allowing the primary surgeon a wide range of motion inside the patient, often results in arm placement taking half of the da Vinci setup time [78]. However, even with careful manual arm placement, instruments colliding during the procedure is still reported [79].

We investigated methods to analyze and minimize potential collision of da Vinci arms with other arms and operating room obstacles while maximizing the surgeon’s workspace manipulability. The output of our proposed method is a suggested set of patient-side manipulator (PSM) base setup positions for the surgical staff to carry out, along with a score representing the optimality of the suggested solution. In order to minimize self-collisions and minimize environment collisions and maximize the surgeon’s reachability in the operative field, we define scoring functions for each of these requirements. We consider C-space of the base of each PSM, which is its translational position and its orientation (X, Y, θ) , assuming the height of the base Z is constrained by the patient/geometry. Note that this representation of the base of the PSM only uses three values, while the position of each actual PSM instrument position can vary during the

operation and involves 6 DOF (excluding the gripper).

Thus, in our definition of the scoring functions, evaluation of a 3 DOF base position involves taking an aggregation over multiple 6 DOF instrument positions from which proportions for each of the metrics are estimated. The scoring functions for reachability, self-collisions, and environmental collisions are defined as follows:

$$score_{reach} = \frac{\# \text{ reachable targets}}{\# \text{ targets}} \quad (2.33)$$

$$score_{self} = \frac{\# \text{ self-collision-free positions}}{\# \text{ instrument positions}} \quad (2.34)$$

$$score_{env} = \frac{\# \text{ environment-collision-free positions}}{\# \text{ instrument positions}} \quad (2.35)$$

$score_{reach}$ involves performing inverse kinematics and determining the proportion of targets the instrument can reach from a given base position. $score_{self}$ and $score_{env}$ involve determining the proportion of instrument positions that are collision-free in a given base position. The scores are considered “better” if they are closer to 1. Fig. 2.12c shows an instance of a set of PSM base positions with a maximized sum of scores.

To get an appropriate estimate of the proportion for $score_{self}$ and $score_{env}$, 1000 random instrument positions are sampled for each base position. As with motion planning, this collision checking is computationally burdensome, especially because numerous base positions must be evaluated when searching for the optimal set of base positions. Supplanting the geometric collision checker with Fastron-based collision checks enabled approximately 2.4 times faster evaluations for $score_{self}$ and $score_{env}$.

Algorithm 1: Fastron Model Updating

Input: X : training dataset of configurations; y : collision status labels
Parameters: $iter_{max}$: maximum number of iterations; S_{max} : maximum number of support points; β : conditional bias parameter; γ : kernel parameter for Gram matrix
Output: α : updated weights; S : support set of configurations

// Get weights, hypothesis, and Gram matrix from previous update

```
1  $\alpha, F, K \leftarrow \text{loadPreviousModel}()$ 
  // Back up previous  $\alpha$  and  $F$ 
2  $\alpha_{before} \leftarrow \alpha, F_{before} \leftarrow F$ 
3 for  $iter = 1$  to  $iter_{max}$  do
  // Check for misclassifications
4   if  $\min yF \leq 0$  then
5      $i \leftarrow \text{argmin} yF$ 
6      $\text{computeGramMatrixColumn}(i)$ 
  // Add/adjust support point
7     if  $\text{count}(\alpha \neq 0) < S_{max}$  OR  $\alpha_i \neq 0$  then
8        $\Delta\alpha \leftarrow \beta^{0.5(y_i+1)} y_i - F_i$ 
9        $\alpha_i \leftarrow \alpha_i + \Delta\alpha$ 
10       $F \leftarrow F + \Delta\alpha Ke(i)$ 
11      continue
  // Back up  $\alpha$  and  $F$ 
12   $\alpha_{before} \leftarrow \alpha, F_{before} \leftarrow F$ 
  // Remove redundant support points
13  if  $\max y(F - \alpha) > 0$  subject to  $\alpha \neq 0$  then
14     $i \leftarrow \text{argmax} y(F - \alpha)$  subject to  $\alpha \neq 0$ 
15     $F \leftarrow F - \alpha_i Ke(i)$ 
16     $\alpha_i \leftarrow 0$ 
17    continue
18  break
  // Revert solution if prior was better
19 if  $\text{count}(yF_{before} \leq 0) < \text{count}(yF \leq 0)$  then
20    $\alpha \leftarrow \alpha_{before}, F \leftarrow F_{before}$ 
  // Remove all elements corresponding to non-support points
21  $\alpha, S, F, K \leftarrow \text{removeNonsupportPoints}()$ 
22 return  $\alpha, S$ 
```

Algorithm 2: Fastron Active Learning

Input: $\mathcal{S}$: support set of configurations

Parameters: $\mathcal{A}_{max}$: number of configurations to add to dataset; κ : number of exploitation samples near each support point; σ^2 : exploitation Gaussian variance

Output: $\mathcal{A}$: unlabeled set of configurations to append to dataset

```
1  $\mathcal{A} \leftarrow \emptyset$ 
  // Exploitation
2 for  $k = 1$  to  $\kappa$  do
3   foreach  $x$  in  $\mathcal{S}$  do
4     if  $|\mathcal{A}| == \mathcal{A}_{max}$  then break
5      $\mathcal{A} \leftarrow \mathcal{A} \cup \{\mathbf{x}_{new} \sim \mathcal{N}(x, \sigma^2 I)\}$ 
  // Exploration
6 for  $i = 1$  to  $\mathcal{A}_{max} - \kappa|\mathcal{S}|$  do
7    $\mathcal{A} \leftarrow \mathcal{A} \cup \{\mathbf{x}_{new} \sim U(-1, 1)\}$ 
8 return  $\mathcal{A}$ 
```

Algorithm 3: Sparse SVM Method (Adapted from Original [4])

Input: $\mathcal{X}$: training dataset of configurations; $\mathbf{y}$: collision status labels

Parameters: γ : kernel parameter for SVM; C : regularization parameter for SVM;
 $iter_{max}$: maximum number of iterations

Output: $\boldsymbol{\alpha}$: $\mathbb{R}^{N+1}$ weight vector (where first element is model bias)

// Initialize weights and Gram matrix

```
1  $\boldsymbol{\alpha} \leftarrow \mathbf{1}$ 
2  $\mathbf{K}_{ij} \leftarrow k(\mathcal{X}_i, \mathcal{X}_j) \forall \mathcal{X}_i, \mathcal{X}_j \in \mathcal{X}$ 
3  $\bar{\mathbf{K}} \leftarrow [\mathbf{1} \ \mathbf{K}]^T \text{diag}(\mathbf{y})$ 
4 for  $iter = 1$  to  $iter_{max}$  do
  // Perform optimization
5    $\mathbf{A} \leftarrow \text{diag}(\boldsymbol{\alpha}), \mathbf{H} \leftarrow \bar{\mathbf{K}}^T \mathbf{A}^2 \bar{\mathbf{K}}$ 
6    $\boldsymbol{\beta} \leftarrow \text{argmin}_{\mathbf{b}} \frac{1}{2} \mathbf{b}^T \mathbf{H} \mathbf{b} - \mathbf{1}^T \mathbf{b} \text{ s.t. } 0 \leq \mathbf{b}_i \leq C \forall i$ 
  // Obtain new weights  $\boldsymbol{\alpha}$ 
7    $\boldsymbol{\alpha}_{prev} \leftarrow \boldsymbol{\alpha}$ 
8    $\boldsymbol{\alpha} \leftarrow \mathbf{A}^2 \bar{\mathbf{K}} \boldsymbol{\beta}, \boldsymbol{\alpha}_i \leftarrow 0 \forall |\boldsymbol{\alpha}_i| < \epsilon$ 
  // Exit if no change in model size
9   if  $\|\boldsymbol{\alpha}\|_0 == \|\boldsymbol{\alpha}_{prev}\|_0$  then
10    break
11 return  $\boldsymbol{\alpha}$ 
```

Algorithm 4: Incremental SVM with Active Learning Method (Adapted from Original [3])

Input: $\mathcal{X}_{init}$: initial training dataset of configurations; $\mathbf{y}_{init}$: initial collision status labels

Parameters: γ : kernel parameter for SVM; C : regularization parameter for SVM; $\Delta P_{explore}$: amount to change exploitation/exploration probability threshold; $N_{\mathcal{A}}$: number of collision checks to perform per active learning update

Output: α : $\mathbb{R}^{N+1}$ weight vector (where first element is model bias); $\mathcal{X}$: updated dataset

```

// Train initial SVM model
1  $\mathcal{X} \leftarrow \mathcal{X}_{init}$ 
2  $\alpha \leftarrow \text{incSvmUpdate}(\mathcal{X}_{init}, \mathbf{y}_{init})$ 
// Generate up to  $\mathcal{A}_{max}$  new samples
3 while  $|\mathcal{X} \setminus \mathcal{X}_{init}| < \mathcal{A}_{max}$  do
    // Active Learning
4      $\mathcal{A} \leftarrow \emptyset$ 
5      $explore \leftarrow \text{rand}() < P_{explore}$ 
6     if  $explore$  then
7         for  $i = 1$  to  $N_{\mathcal{A}}$  do
8              $\mathcal{A} \leftarrow \mathcal{A} \cup \{\mathbf{x}_{new} \sim U(-1, 1)\}$ 
9     else
10        for  $i = 1$  to  $N_{\mathcal{A}}$  do
11             $\mathcal{X}_{sel} \leftarrow \text{randPick}(\mathcal{S})$ 
12             $\mathcal{X}_{near} \leftarrow \text{argmin}_{\mathcal{X}_j} |\mathcal{X}_{sel} - \mathcal{X}_j|$     s.t.  $\mathcal{X}_j \in \mathcal{S} \wedge \mathbf{y}_j \neq \mathbf{y}_{sel}$ 
13             $\mathbf{x}_{new} \leftarrow \text{argmax}_{\mathbf{x}} k(\mathbf{x}, \mathcal{X}_{sel}) + k(\mathbf{x}, \mathcal{X}_{near})$ 
14             $\mathcal{A} \leftarrow \mathcal{A} \cup \{\mathbf{x}_{new}\}$ 
15     $\mathcal{X} \leftarrow \mathcal{X} \cup \mathcal{A}$ 
16     $\mathbf{y}_{\mathcal{A}} \leftarrow \text{colCheck}(\mathcal{A})$ 
    // Update SVM with new information
17     $\alpha \leftarrow \text{incSvmUpdate}(\mathcal{A}, \mathbf{y}_{\mathcal{A}})$ 
    // Increase  $P_{explore}$  if exploration samples are poorly classified
18    if  $explore$  AND  $\text{errRate}(\mathcal{A}) > \text{err}_{thres}$  then
19         $P_{explore} += \Delta P_{explore}$ 
20    else
21         $P_{explore} -= \Delta P_{explore}$ 
    // Exit if all new samples are correctly classified
22    if  $\text{errRate}(\mathcal{X} \setminus \mathcal{X}_{init}) == 0$  then
23        break
24 return  $\alpha, \mathcal{X}$ 

```

2.5 Acknowledgments

Chapter 2, in part, is a reprint of material from N. Das, M. Yip, “Learning-Based Proxy Collision Detection for Robot Motion Planning Applications,” in *IEEE Transactions on Robotics*, 2020. The dissertation author is the primary author of this paper.

©2020 IEEE.

Chapter 3

Forward Kinematics Kernel

The kernel-based techniques mentioned in Section 2 used radial basis function (RBF) kernels to represent the regions of C-space. While these techniques can successfully model the C-space given a sufficient dataset of labeled robot configurations, the robot’s workspace and C-space are treated as completely separate entities and the link that relates the two spaces is more or less neglected. More specifically, the Fastron model described in Section 2 considers the joint configurations of a robot at face value, where each joint is considered to have equal contribution in determining a configuration’s collision status. While the Fastron model is fit to samples in the C-space, the collisions occur in the workspace. Thus, a better way to compare C-space samples should also represent some information from the workspace, which will involve forward kinematics (FK).

Kernel functions receive considerable attention in machine learning because not only do they measure similarity between two objects (the basis of learning from preexisting data), but also they represent implicit higher-dimensional feature spaces that easily allow nonlinear models [49]. As kernels represent useful feature spaces implicitly, creating new kernels that represent relevant feature spaces is a worthwhile effort for enhancing machine learning performance [80]. In this section, we aim to improve the link between a robot’s C-space and workspace by defining

a new kernel function based on FK.

3.1 Kernel Functions

A kernel function $K : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$ is a similarity function that should provide a large score for similar configurations and a low score for dissimilar configurations. RBF kernels, which are functions of the distance between two points, such as the Gaussian kernel (Eq. 1.2) or rational quadratic kernel, are popular. In this section, we use the second-order rational quadratic kernel [11] for all rational quadratic kernels, defined as

$$K_{RQ}(\mathbf{x}, \mathbf{x}') = \left(1 + \frac{\gamma}{2} \|\mathbf{x} - \mathbf{x}'\|^2\right)^{-2} \quad (3.1)$$

where $\gamma > 0$ is a parameter dictating the width of the kernel. Each element of a kernel matrix $\mathbf{K}$ is defined elementwise for each pair of points in a dataset $\mathcal{X}$: $\mathbf{K}_{ij} = K(\mathcal{X}_i, \mathcal{X}_j)$, where $\mathcal{X}_i \in \mathbb{R}^D$ represents a configuration of the robot with D DOF.

3.2 Forward Kinematics Kernel Definition

The RBF kernels defined above compute similarity metrics according to the configurations of a robot, such as the robot manipulator's joint angles. While RBF kernels are common in practice in machine learning, the isotropic nature of the RBF kernels assumes each joint has equal contribution in evaluating the similarity of one configuration to another.

Kernel functions should be selected to capture meaningful similarities between objects [81], so we strive to create a more intuitive similarity metric by computing distances in the workspace rather than in the joint space. Computing distances directly between joint configurations (as is done by applying the RBF kernel directly to the elements of $\mathcal{X}$) does not take into consideration the robot's position in the workspace. Consider the blue two-link arm shown in Fig.

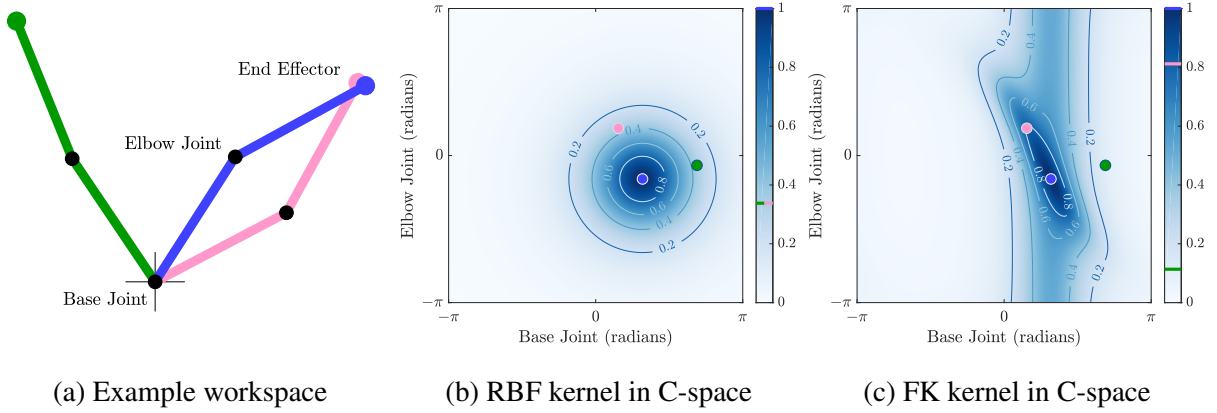


Figure 3.1: (a) A workspace representation of one configuration of a two-link arm shown in blue along with two alternate configurations of the same robot shown in green and pink. (b) A visualization of the RBF kernel in C-space centered on the blue configuration. According to the RBF kernel, the blue configuration is as similar to the pink configuration as it is the green configuration, as they reside on the same isocontour. (c) A visualization of the FK kernel in C-space centered on the blue configuration. According to the FK kernel, the blue configuration is more similar to the pink configuration than to the green, which is more representative of the proximities of the blue and pink configurations in the workspace representation.

3.1a. Comparing the pink and green configurations to the blue, the pink configuration appears much closer to the blue than the green does. However, the distance in the C-space between the blue and pink configurations (represented as colored dots) is equal to that of the blue and green configurations as can be seen in Fig. 3.1b and 3.1c. Consequently, according to the RBF kernel, the blue configuration is as similar to the pink configuration as it is the green configuration as shown in Fig. 3.1b.

Rather than comparing the joint values of two configurations directly, we use FK and compare distances in the workspace between *control points* on the arm. Control points are a set of hand-selected points placed on a robot and their locations may be used to indicate the position of a rigid-body robot or a kinematic chain manipulator. Control points should be chosen to constrain the configuration of the robot, and the minimum number of these points depends on the number of DOF of the robot [24]. FK provides the positions of these control points for manipulators given a set of joint values as defined in Section 1.1.1, and we consider FK to be the link that relates a robot's position in C-space to its position in the workspace.

The concept of control points has been used in various works related to planning or modeling for robot manipulators. Artificial potential fields can be used for motion planning in a robot’s C-space, where the sum of attractive and repulsive forces drive the robot away from C-space obstacles and toward a goal [82]. Potential fields can be applied to kinematic chain manipulators using control points placed along the kinematic chain [24].

Another example that utilizes control points is RelaxedIK, a motion-synthesis method for manipulators, uses a neural network to estimate the risk of self-collision for an input robot configuration [25]. Control points are placed at each joint and their locations in the workspace are included in the input to the network. The use of control points allows estimation of the self-collision costs two orders of magnitude faster than a distance-based method, and optimizing over these self-collision costs allows finding configurations that are free from self-collisions.

In the context of kernel functions, we utilize control points to create a similarity function between manipulator configurations, and we leverage how FK relates a manipulator’s C-space to its workspace for more accurate proxy collision checking.

The equations for FK are defined in Section 1.1.1. Let $\text{pos}(\cdot)$ extract the position from a homogeneous transformation matrix, i.e., the first three elements of the last column. We can thus represent the position of the m^{th} control point for configuration $\mathbf{x} \in \mathbb{R}^D$ as

$$FK_m(\mathbf{x}) = \text{pos}({}^w\mathbf{A}_{j_m}\mathbf{T}_m) \quad (3.2)$$

where $\mathbf{T}_m$ is the static transform applied to the the pose of the joint with index j_m .

We can thus define the FK kernel as a sum of rational quadratic kernels evaluated between each corresponding control point location:

$$K_{FK}(\mathbf{x}, \mathbf{x}') = \frac{1}{M} \sum_{m=1}^M K_{RQ}(FK_m(\mathbf{x}), FK_m(\mathbf{x}')) \quad (3.3)$$

where $FK_m(\cdot)$ provides the position of the m^{th} control point and M is the number of control points

representing the kinematic chain.

In our implementation, we only place control points at joint frame origins (using the standard D-H convention) by setting $\mathbf{T}_m = \mathbf{I}$ for each control point. In other words, a control point is placed at the distal end of each link. However, control points may be placed at arbitrary locations along the manipulator if desired, as long as the control points constrain the positions of the links. If the origins of the frames of two subsequent joints in the kinematic chain are coincident, we only place one control point at the overlapping origin. In other words, using the standard D-H convention, if a revolute joint has both d_i and a_i equal to 0, a control point is not placed for this joint as its origin overlaps with the preceding joint origin. Ultimately, one control point is placed for each link, which also matches the suggested number of control points for the potential fields method for manipulators [24].

Fig. 3.1c provides a visualization of the FK kernel for the blue two-link arm in Fig. 3.1a, where control points are placed at the location labeled “Elbow Joint” and “End Effector”. According to the FK kernel, the similarity score between the blue and pink configurations is about 0.8, while the similarity between the blue and green is about 0.1. The similarity scores produced by the FK kernel better represent the differences we perceive in the positions of the robots in the workspace.

To achieve an intuitive understanding of the shape of the FK kernel, we can consider a rough approximation of the kernel by replacing each rational quadratic function in Eq. 3.3 with a Gaussian kernel and considering a first-order approximation of the norm that occurs in each Gaussian:

$$\|FK_m(\mathbf{x}) - FK_m(\mathbf{x}')\|^2 \approx \|J_m(\mathbf{x}')(\mathbf{x} - \mathbf{x}')\|^2 = (\mathbf{x} - \mathbf{x}')^\top J_m(\mathbf{x}')^\top J_m(\mathbf{x}')(\mathbf{x} - \mathbf{x}') \quad (3.4)$$

where $J_m(\mathbf{x})$ is the $\mathbb{R}^{3 \times D}$ translational manipulator Jacobian relating joint velocities to the $\mathbb{R}^3$ translational workspace velocities of the m^{th} control point. It should be evident from Eq. 3.4 that

the shape will be approximately proportional to that of a Gaussian with a covariance proportional to $(J_m(\mathbf{x}')^\top J_m(\mathbf{x}'))^{-1}$. Note that changes in joint values more distal than the m^{th} control point will have no influence on the m^{th} control point's workspace velocity, so the corresponding columns of $J_m(\cdot)$ will be 0. Ultimately, this makes the m^{th} rational quadratic kernel in Eq. 3.3 approximately a multivariate Gaussian with as many dimensions as there are joints that influence the location of control point m . This should be evident in Fig. 3.1c, where the first control point on the elbow joint yields an approximate 1-dimensional Gaussian (creating the vertical column shape when plotted in 2D) and the second control point on the end effector yields an approximate 2-dimensional skewed Gaussian shape.

3.3 Positive Definiteness

$\mathbf{K}$ is called positive definite (PD) if $\boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\alpha} \geq 0 \ \forall \boldsymbol{\alpha} \in \mathbb{R}^N$, and equality occurs only for $\boldsymbol{\alpha} = \mathbf{0}$. $\mathbf{K}$ is considered to be positive semidefinite (PSD) if equality may occur for $\boldsymbol{\alpha} \neq \mathbf{0}$.

A PD kernel is one that yields a PD kernel matrix $\mathbf{K}$ when each configuration in set $\mathcal{X}$ is unique [49]. Positive definiteness is an important property for a kernel function because it guarantees nonsingularity for the kernel matrix and represents an implicit mapping to a higher dimensional feature space according to Mercer's theorem [50]. PSD kernels and matrices do not have the same guarantees their PD counterparts have. $K_{RQ}(\cdot, \cdot)$ is an example of a PD kernel [11]. Summing PD kernels or multiplying PD kernels with a scalar also yields a PD kernel [49].

The FK kernel matrix based on the FK kernel defined in Eq. 3.3 is a sum of RBF kernel matrices evaluated on M control points: $\mathbf{K} = \frac{1}{M} \sum_i^M \mathbf{K}_m$. We henceforth refer to the RBF kernel matrices that comprise the FK kernel matrix as *summand kernel matrices*. The m^{th} summand kernel matrix $\mathbf{K}_m$ is PD as long as the locations of the m^{th} control point (determined by applying $FK_m(\cdot)$ to the configurations in $\mathcal{X}$) are unique. However, depending on the choices of control points or kinematic redundancy, it is possible for two unique configurations in $\mathcal{X}$ to yield the

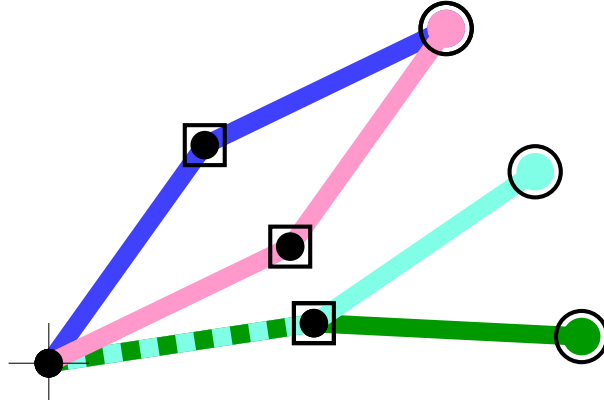


Figure 3.2: Four unique configurations of a two-link arm. There are two control points selected along the arm, marked by the square and hollow circle. Overlapping links are marked with a dashed line. Despite being unique configurations, some control points overlap, causing summand kernel matrices to be positive semidefinite. However, the FK kernel is positive definite as long as the configurations with coincident control points are distinguishable based on the other control points.

same control point location. For example, Fig. 3.2 shows four unique configurations of a two-link arm, but there are only three unique locations for each control point (represented as a hollow square and circle on each configuration). With coincident control point locations, the summand kernel matrices are demoted to PSD.

Despite possibly having PSD summand kernels, the *FK kernel matrix is PD* if the intersection of the nullspaces of the summand matrices is the zero vector.

Claim 2. The FK kernel matrix $\mathbf{K} = \frac{1}{M} \sum_{m=1}^M \mathbf{K}_m$ is PD if the nullspaces of each summand kernel matrix $\mathbf{K}_m$ share only the zero vector, $\bigcap_{m=1}^M N(\mathbf{K}_m) = \mathbf{0}$.

Proof. Assume $\mathbf{v} \in \bigcap_{m=1}^M N(\mathbf{K}_m)$. Thus, $\mathbf{K}_1 \mathbf{v} + \dots + \mathbf{K}_M \mathbf{v} = M \mathbf{K} \mathbf{v} = \mathbf{0} \Rightarrow \bigcap_{m=1}^M N(\mathbf{K}_m) \subseteq N(\mathbf{K})$. For $\mathbf{K}$ to be PD, it must also be nonsingular, which means $N(\mathbf{K}) = \mathbf{0}$. Thus, $\bigcap_{m=1}^M N(\mathbf{K}_m) = \mathbf{0}$. $\square$

What Claim 2 means intuitively is that even if some control point locations coincide for two unique configurations, the FK kernel is PD if these configurations are distinguishable using the locations of the other control points. For example, in Fig. 3.2, the locations of the control points marked with hollow circles differentiate the configurations with coincident control points

marked with squares, and vice versa. Note that in practice, if the configurations in $\mathcal{X}$ are generated via random sampling, control point locations should almost surely be unique.

3.4 Performance Using Forward Kinematics Kernel

In this section, we evaluate the performance of the FK kernel by examining how it improves proxy collision checking and motion planning. We use the 7 DOF arms of the Baxter robot for these experiments. We use Fastron with the proposed FK kernel and with the original rational quadratic kernel, denoted as Fastron FK and Fastron RBF, respectively. Via grid search, we select $\gamma = 100$ for Fastron FK and $\gamma = 10$ for Fastron RBF and $\beta = 100$ for both models, and both models are initialized with 5000 training samples. We select the maximum number of support points and maximum number of iterations for model updates to be large enough to allow full convergence. The geometric collision detectors compared are GJK [5] and FCL [8]. GJK can determine whether two convex shapes are intersecting, and FCL is the collision detection library used in the MoveIt! motion planning framework [66]. We use GJK as a faster, approximate geometric method and use FCL as the ground truth collision method.

3.4.1 Collision Checking

To compare collision checking performance, we compare model correctness, query times, model sizes, and model update times. Measures of model correctness include the overall classification accuracy, true positive rate (TPR), and true negative rate (TNR), defined as

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + TN + FP} \quad (3.5)$$

$$\text{TPR} = \frac{TP}{TP + FN} \quad (3.6)$$

$$\text{TNR} = \frac{TN}{TN + FP} \quad (3.7)$$

Table 3.1: Performance of Fastron FK and Fastron RBF compared against geometric collision detection methods (GJK and FCL). Lower is better for query time, model size $|S|$, and model update time.

Method	Query Time	$ S $	Update Time
Fastron FK	$2.6 \pm 0.5 \mu s$	329.7 ± 120.6	$56.8 \pm 5.5 ms$
Fastron RBF	$4.8 \pm 1.2 \mu s$	2505.2 ± 616.7	$134.5 \pm 32.9 ms$
GJK	$14.5 \pm 0.1 \mu s$	—	—
FCL	$22.8 \pm 0.3 \mu s$	—	—

where TP and FP represent the number of samples in the test set that are correctly and incorrectly predicted to be in C_{obs} , respectively, and TN and FN represent the same but for the C_{free} class. TPR and TNR are useful measures of correctness when the volumes of C_{obs} and C_{free} are unbalanced. Model size $|S|$ is the count of support points required to represent the model for the proxy collision check methods. The results of the experiments are averaged over 200 trials, where each trial uses one randomly-sized obstacle that moves up to 30 times in a random direction, and the models are updated between each obstacle movement.

The query timings, model sizes, and model update timings are tabulated in Table 3.1. Query timings refer to the time per collision check, model sizes refer to the number of support points required for the Fastron models, and model update timings refer to the time required to update a trained Fastron model when obstacles move. The initial model training (i.e., labeling 5000 random configurations using FCL and training the model starting from $\alpha = 0$) took on average 133.9 ms and 164.6 ms for Fastron FK and Fastron RBF, respectively. The majority of this initial training time (approximately 115 ms) for both methods is dedicated toward labeling the dataset using FCL. While Fastron FK has 7.6 times fewer support points than Fastron RBF, the FK kernel is more expensive to compute than the RBF kernel, causing Fastron FK to be only 1.8 times faster than Fastron RBF for query times. Fastron FK beats GJK and FCL by a factor of 5.6 and 8.8, respectively, for query times. Since there are fewer support points in the Fastron FK model, the update time (largely throttled by labeling new data via FCL) is about 2.4 times faster

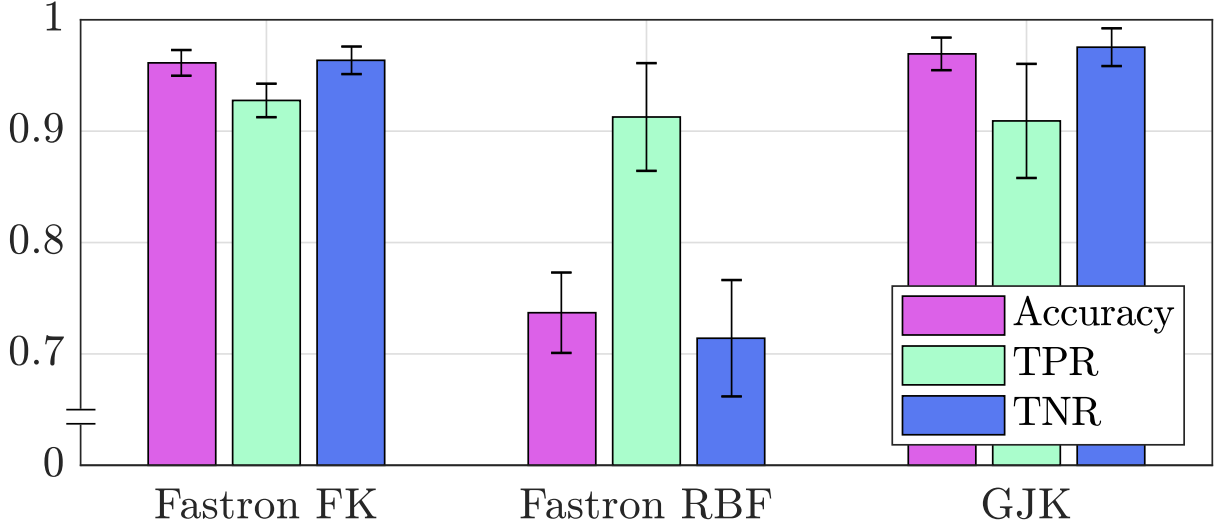


Figure 3.3: A comparison of the correctness of Fastron FK [7], Fastron RBF [6], and GJK [5] with a single, randomly-sized moving obstacle. Fastron FK has much better accuracy, TPR, and TNR than Fastron RBF and performs similarly to GJK.

for Fastron FK than for Fastron RBF.

The correctness of the approximate collision methods are shown in Fig. 3.3, illustrating the advantages of Fastron FK over Fastron RBF. The accuracy, TPR, and TNR of Fastron FK are much higher than those of Fastron RBF. In fact, Fastron FK performs similarly in terms of correctness to GJK, but is 5.6 times faster. Fastron FK achieves an accuracy of 96.1%, a significant improvement over Fastron RBF’s accuracy of 73.7%. While the accuracy, TPR, and TNR are roughly balanced for Fastron FK, Fastron RBF’s TPR is much higher than its accuracy and TNR because it prioritizes the C_{obs} label over the C_{free} label. While biasing toward the C_{obs} label yields a more conservative model compared to not biasing toward C_{obs} , it also reduces the volume of the valid region of C-space and increases search time during sampling-based motion planning, which we will see is the case for Fastron RBF in the next section.

Fig. 3.4 shows typical false negatives when using Fastron FK and Fastron RBF, where the true collision status is C_{obs} but the Fastron models incorrectly predict C_{free} . Using the RBF kernel, the robot significantly intersects with the workspace obstacle. The false negatives have an average (over 500 random environments) penetration depth of 3.3 cm, and 22.1% of them

have more than 5 cm of penetration. On the other hand, using the FK kernel, the robot only slightly intersects with the workspace obstacles. The false negatives have an average penetration depth of 1.6 cm, and 4.1% of them have a penetration greater than 5 cm. A lower penetration depth for false negatives shows that Fastron FK has less severe mistakes than Fastron RBF in the workspace.

In an additional experiment, we increase the number of obstacles to see the effect on query times. Fig. 3.5 shows the query times for each method with respect to the number of workspace obstacles for up to 50 randomly-placed obstacles. We can see that the geometry-based methods, GJK and FCL, increase in query times as the number of obstacles increase. On the other hand, Fastron-based query times first increase and then decrease because fewer support points are required when one collision status is more prevalent than the other. Fewer support points are required by both Fastron FK and Fastron RBF when C_{obs} dominates the C-space, and because each FK kernel (Eq. 3.3) evaluation is a sum of RBF kernels (Eq. 3.1), Fastron RBF inevitably becomes faster than Fastron FK when both models require fewer support points to represent the C-space. Nonetheless, the accuracy of Fastron FK is consistently high (between 94-98%), while the accuracy of Fastron RBF is lower (between 71-94%). Fastron RBF is only able to achieve

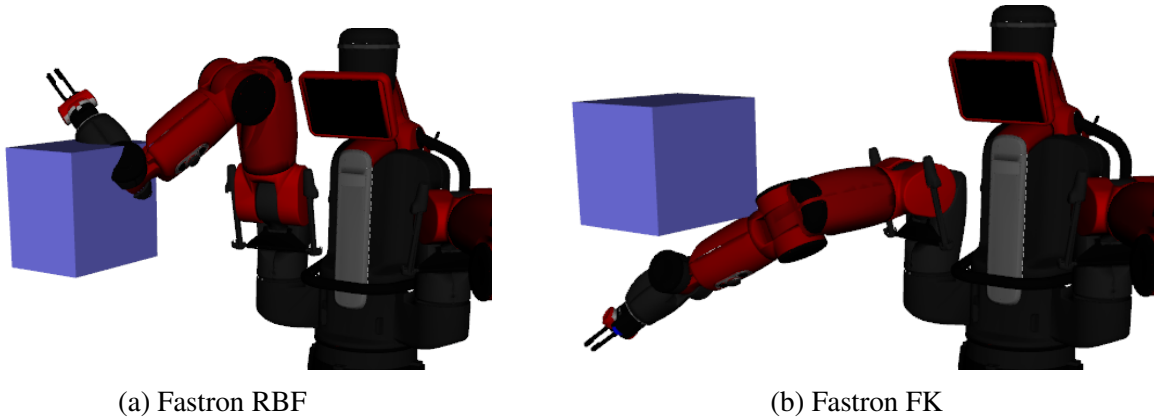
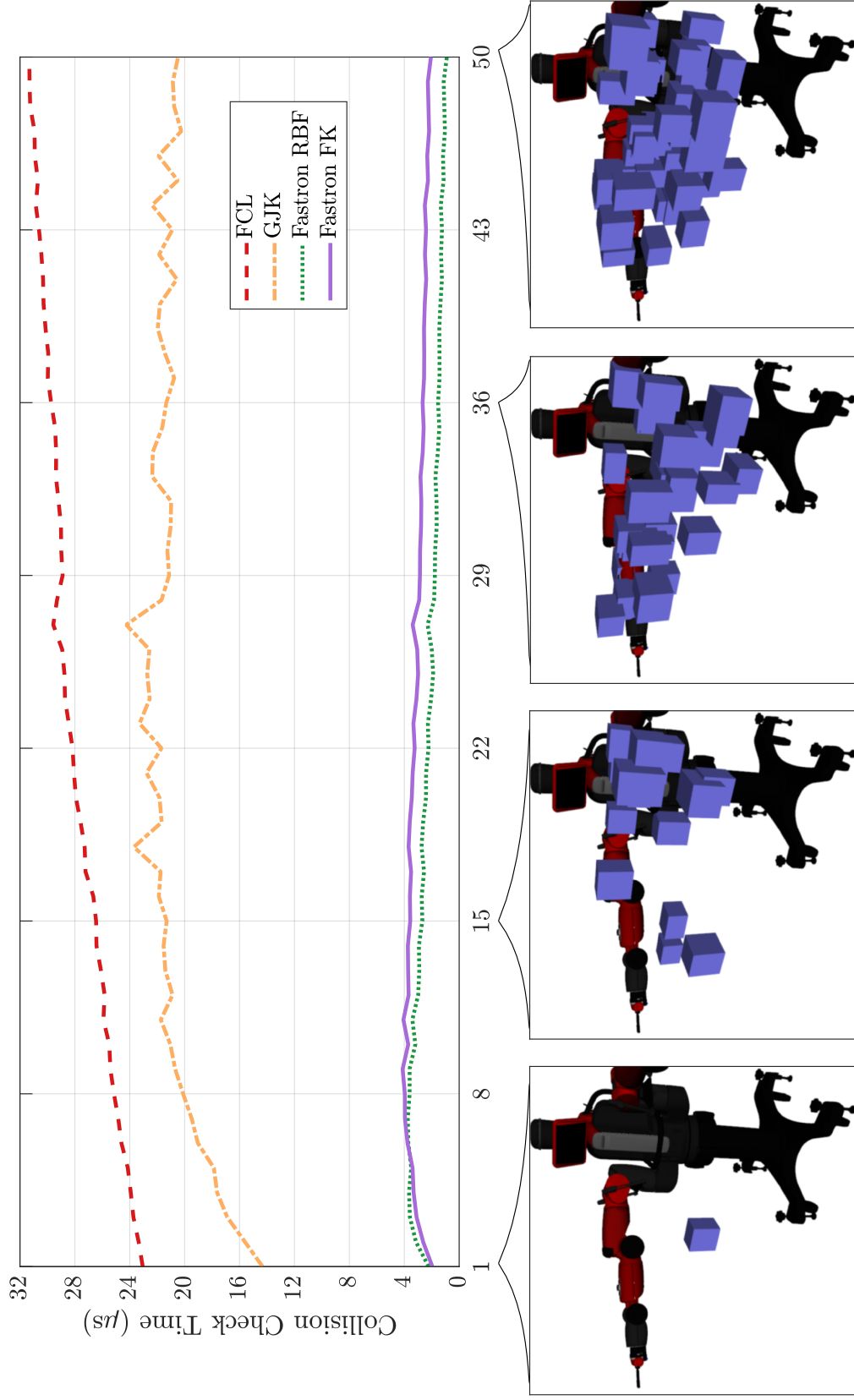


Figure 3.4: Typical failure cases using the RBF kernel and the FK kernel, where the configuration is mistakenly labeled as C_{free} but is truly C_{obs} . With the RBF kernel, the robot blatantly intersects with the workspace obstacle. With the FK kernel, the robot slightly intersects with the obstacle.



Number of Workspace Obstacles

Figure 3.5: Query times for Fastron FK [7], Fastron RBF [6], GJK [5], and FCL [8] against the number of workspace obstacles. Both GJK and FCL increase in query times as the number of obstacles increase, while both Fastron methods increase slightly before decreasing because the number of support points required to represent the space are the most when the class sizes are more balanced.

94% accuracy when there are 50 obstacles because almost all configurations are considered to be in C_{obs} at that point.

3.4.2 Motion Planning

We apply Fastron FK to a broad collection of the sampling-based motion planning algorithms implemented in OMPL [69]. We select RRT [1], RRT-Connect [30], SBL [83], RRT* [31], FMT* [84], and Informed-RRT* [32] to demonstrate each collision checking method’s performance. RRT, its bidirectional variant RRT-Connect, and SBL are probabilistically complete motion planners [24] that terminate once a path between start and goal is found. SBL includes lazy collision checking, which only calls the collision check routine when absolutely necessary. The other planners are optimizing planners that attempt to determine the shortest collision-free path between start and goal. Rather than letting the optimizing planners run until timeout (which would cause all planning times to be equal), we stop these planners once an initial feasible path has been found. While these paths may not be close to optimal, we include the average lengths of these initial paths in the plot of results.

We use similar environments as used in Section 3.4.1, but the obstacle is sized such that the arm must always move around it; these environments are more challenging than in Section 2.3.3 because the volume of C_{free} is smaller due to a larger workspace obstacle. The start and goal configurations are selected to be on opposite sides of the workspace obstacle. Each collision checker is applied to each motion planner for 50 trials, where a trial consists of up to 30 random movements of the workspace obstacle, and the motion planner is repeated 10 times for each obstacle position. When using the approximate collision checkers, we verify and repair (if necessary) each plan using FCL by replanning with FCL on the invalid part of the plan. The resulting motion plan is thus guaranteed to be collision-free according to FCL.

The average motion planning timings are shown in Fig. 3.6. Even with the verify and repair steps, Fastron FK provides the fastest motion plans, providing up to 3 times faster collision-

free motion plans compared to when using FCL. On the other hand, Fastron RBF often struggles to find a feasible path because it overestimates the C_{obs} region, causing its planning time to often be large or hit the time limit. Furthermore, the repair times for Fastron RBF are significantly larger than those for Fastron FK and GJK, illustrating that Fastron RBF often yields invalid paths.

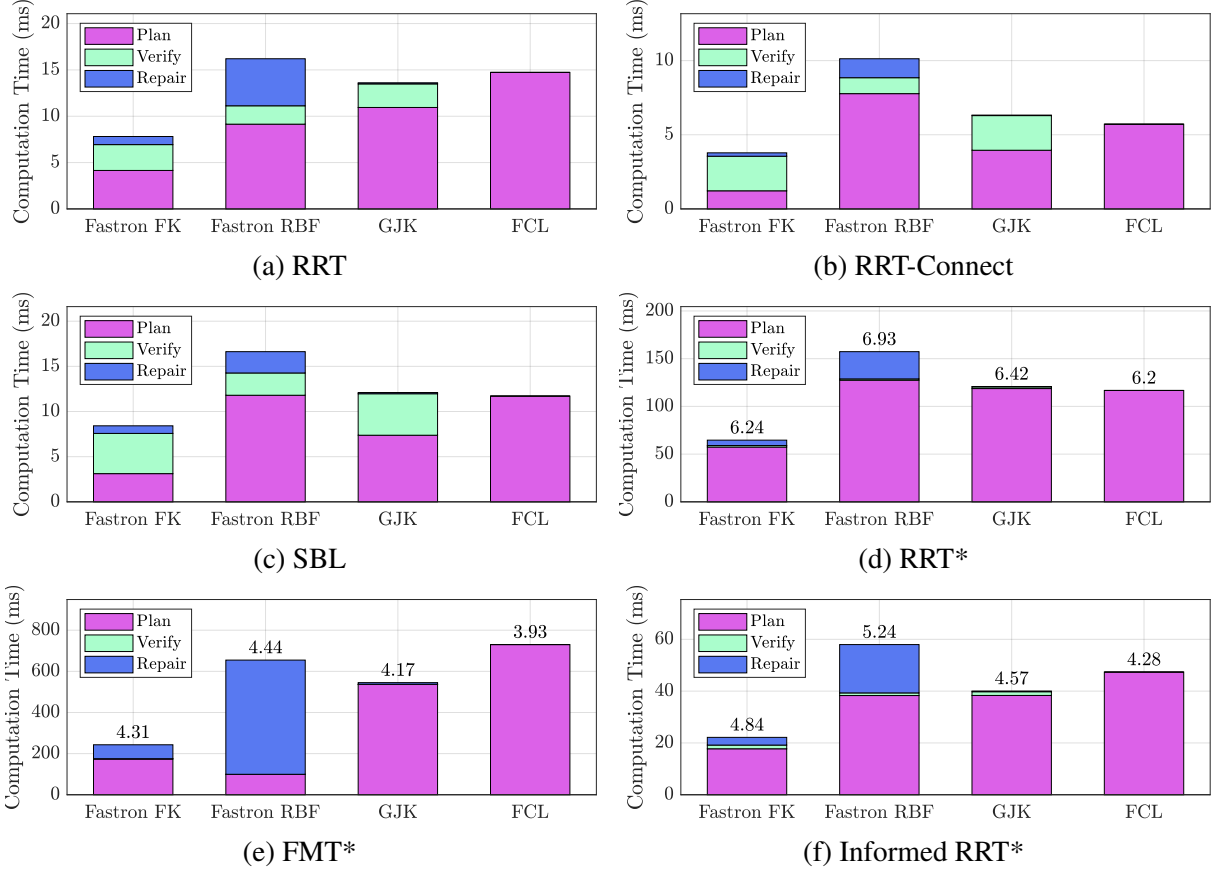


Figure 3.6: Motion planning results using various collision detection methods: Fastron with FK kernel [7], Fastron with rational quadratic kernel [6], GJK [5], and FCL [8] (which serves as the ground truth collision detector). The obstacle in the workspace is repeatedly moved. For each approximated collision detection method, the path is validated and fixed using FCL in the Verify and Repair steps. The numbers above each optimizing planner’s bars are the average lengths of the initial motion plans.

3.5 Further Applications

The FK kernel is a positive definite kernel, and Fastron is only one kernel-based machine learning technique that may utilize such a kernel. In fact, there are multiple other kernel-based algorithms that may utilize the FK kernel, such as kernel principal components analysis [85] or import vector machines [9].

3.5.1 Kernel Principal Component Analysis

Principal component analysis (PCA) is a dimensionality reduction technique that projects a dataset onto the largest eigenvectors of the covariance matrix, which correspond to the directions of maximum variance in the data. Kernel PCA (KPCA) is another dimensionality reduction technique that utilizes the kernel matrix [85].

Note that a kernel function implicitly maps points into a feature space, which often does not have a finite explicit mapping. Thus, finding eigenvectors in the feature space is difficult. Instead, the projection onto the eigenvectors is done directly by assuming that the principal components are a weighted sum of feature mappings of the dataset. Let $\mathbf{V}^n = \sum_{i=1}^N \alpha_i^n \phi(\mathcal{X}_i)$ be the n^{th} principal component, where $\phi(\mathcal{X}_i)$ is a datapoint in the feature space, and α^n is the set of weights on each feature space point to create the n^{th} principal component. The projection of a query point $\mathbf{x}$ in the feature space onto the n^{th} principal component is thus $\mathbf{V}^n \cdot \phi(\mathbf{x}) = \sum_{i=1}^N \alpha_i^n K(\mathcal{X}_i, \mathbf{x})$. α^n is determined by solving the eigenvalue problem $N\lambda\alpha = \mathbf{K}\alpha$ and determining the eigenvectors corresponding to the top eigenvalues. Each principal component $\mathbf{V}^n$ should also be normalized to have norm 1, which is achieved by normalizing α^n according to $1 = \mathbf{V}^n \cdot \mathbf{V}^n = \alpha^{n\top} \mathbf{K} \alpha^n = \lambda^n \alpha^{n\top} \alpha^n$ [85].

PCA centers the data prior to determining the covariance matrix. Similarly, KPCA first centers the kernel matrix before computing any of the eigenvalues by creating a centered version

of the kernel matrix

$$\tilde{\mathbf{K}} = \mathbf{K} - \mathbf{1}_N^T \mathbf{K} - \mathbf{K} \mathbf{1}_N + \mathbf{1}_N^T \mathbf{K} \mathbf{1}_N \quad (3.8)$$

where $\mathbf{1}_N$ is a $N \times N$ matrix with each entry set to $\frac{1}{N}$.

We apply KPCA using various kernels on data acquired for a planar 4 DOF robot arm (Fig. 3.7a) and for a 7 DOF arm of a Baxter robot (Fig. 3.8a). Three obstacles are placed in the robot's environment. Each configuration is labeled as either collision-free or in-collision with a specific obstacle. The weights for the top principal components in the feature space are determined using a dataset of 2000 and 5000 points for the 4 DOF and 7 DOF robot, respectively. A test dataset is then projected onto the top 2 and 3 principal components for the 4 DOF and 7 DOF robot, respectively. The projections are shown in Fig. 3.7 and Fig. 3.8, where the red points correspond to collision-free configurations and the blue, green, and purple colors correspond to configurations that are in collision with the workspace obstacles with these respective colors. We can see a much clearer distinction in the locations of the projections of different colored points when using the FK kernel, but the points are significantly overlapping when using the Gaussian kernel or rational quadratic kernel. This clear distinction when using the FK kernel illustrates the power of its feature space over the RBF kernels. The reason the FK kernel has better intrinsic separability of the colors is the FK kernel's reliance on workspace locations probably better retains the distinction between the workspace obstacle locations.

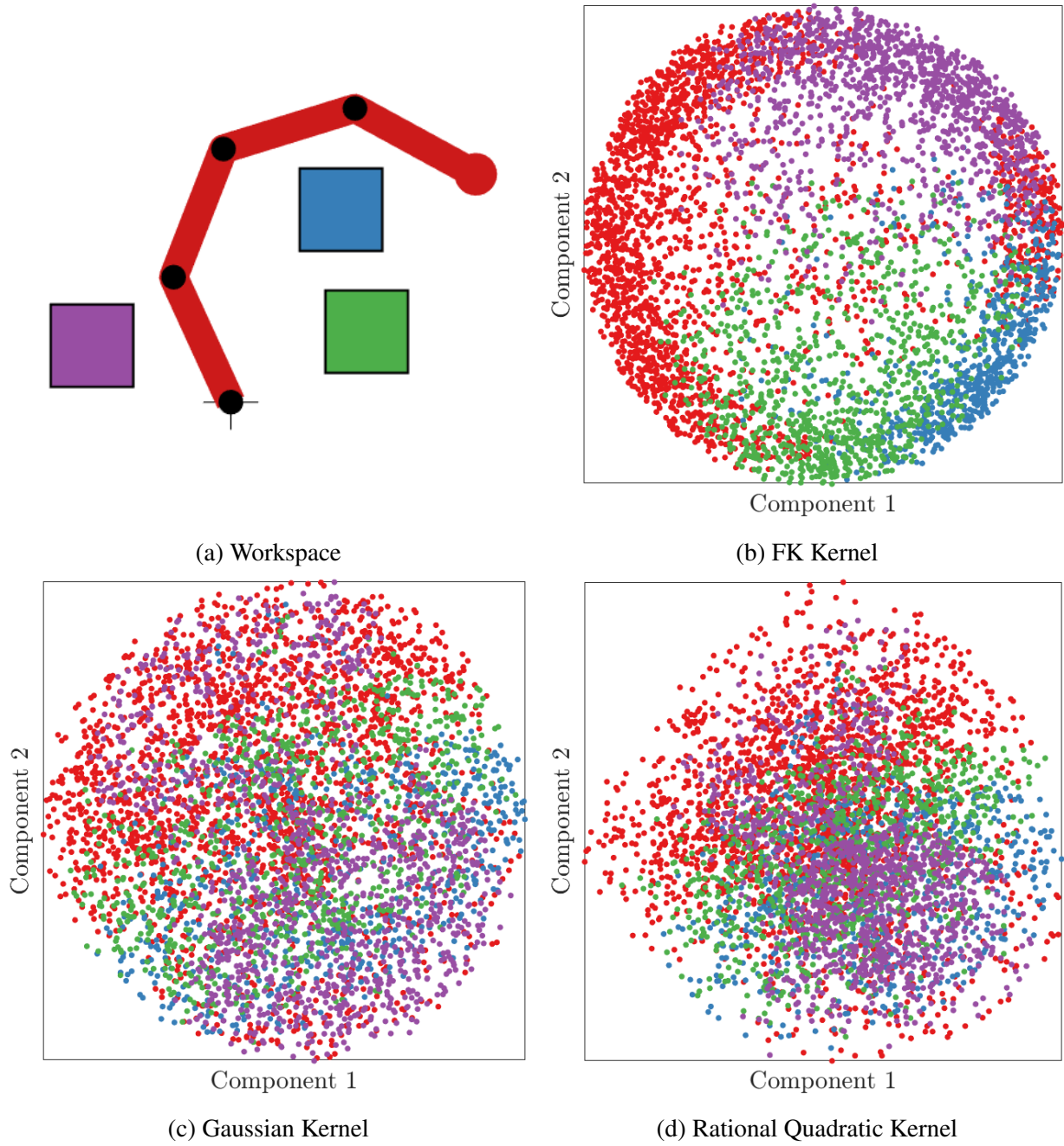


Figure 3.7: (a) 4 DOF planar robot in workspace. KPCA projections onto the first two components using (b) the FK kernel, (c) the Gaussian kernel, and (d) the rational quadratic kernel. The red points correspond to collision-free configurations while the other colors match the obstacle the robot touches.

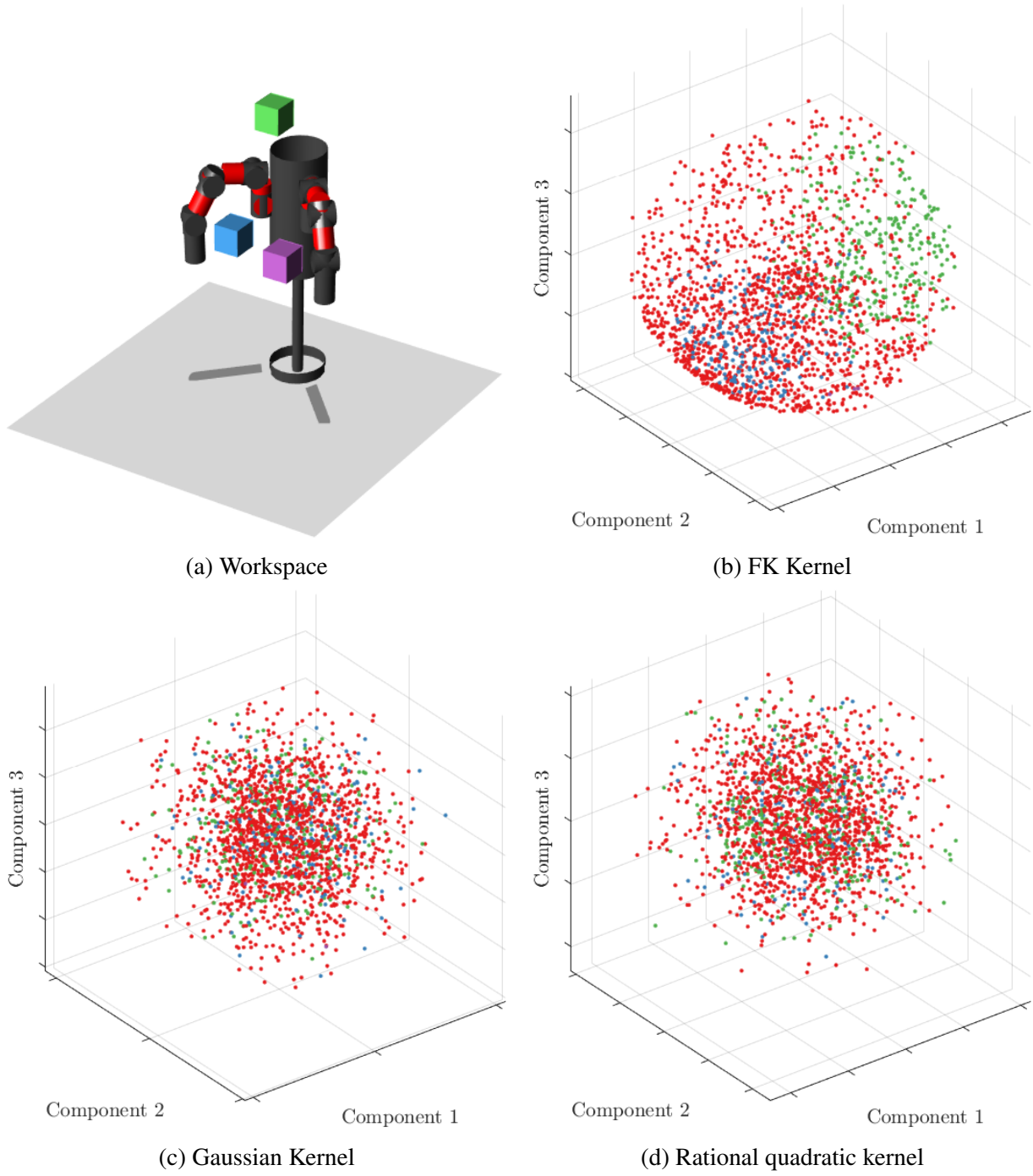


Figure 3.8: (a) Baxter robot in workspace. KPCA projection onto the first three components using (b) the FK kernel, (c) the Gaussian kernel, and (d) the rational quadratic kernel. The red points correspond to collision-free configurations while the other colors match the obstacle the robot touches.

3.5.2 Import Vector Machine

Binary classifiers such as SVMs [43] assign a query point to one of two class labels. On the other hand, a probability classifier estimates a probability that a query point is in a certain class. Such a classifier will estimate the probability that a given query point $\mathbf{x}$ has the label $\mathbf{y} = 1$ for example¹. The import vector machine (IVM) [9] is an example of a probability classifier, which is similar in structure to the SVM.

IVM attempts to solve the same optimization problem that is solved in kernel logistic regression:

$$\min_{\alpha} -\mathbf{y}^T \mathbf{K} \alpha + \mathbf{1}^T \log(1 + \exp(\mathbf{K} \alpha)) + \frac{\lambda}{2} \alpha^T \mathbf{K} \alpha \quad (3.9)$$

where λ is a regularization parameter. A disadvantage of kernel logistic regression is that all points in the dataset are required to make the model. With IVM, only a subset of the dataset is required.

The values of α are determined iteratively by taking a weighted least squares step. For the k^{th} iteration, the step is:

$$\alpha^{(k)} = \left(\mathbf{K}_a^T \mathbf{W} \mathbf{K}_a + \lambda \mathbf{K}_q \right)^{-1} \mathbf{K}_a^T \mathbf{W} \mathbf{z} \quad (3.10)$$

where $\mathbf{K}_a$ is the kernel matrix between all of the data $\mathcal{X}$ and the support set so far, $\mathbf{K}_q$ is the kernel matrix for just the support set so far, $\mathbf{W} = \text{diag}(p(\mathcal{X}_i)(1 - p(\mathcal{X}_i)))$, $\mathbf{z} = \mathbf{K}_a \alpha^{(k-1)} + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p})$, and $p(\mathcal{X}_i) = (1 + \exp(-f(\mathcal{X}_i)))^{-1}$.

The support set $\mathcal{S}$ begins as the empty set, and on each iteration, the step in Eq. 3.10 is computed for each point in $\mathcal{X} \setminus \mathcal{S}$. The point in $\mathcal{X} \setminus \mathcal{S}$ that minimizes the 3.9 the most is added to $\mathcal{S}$, and the process repeats until the loss function converges. Algorithm 5 summarizes the training procedure.

Note that any positive definite kernel may be used to create the kernel matrices for IVM. In this case, we compare the use of the rational quadratic kernel to the FK kernel for a 2 DOF

¹In this section, $\mathbf{y}_i = 1$ for the in-collision class and $\mathbf{y}_i = 0$ for the collision-free class

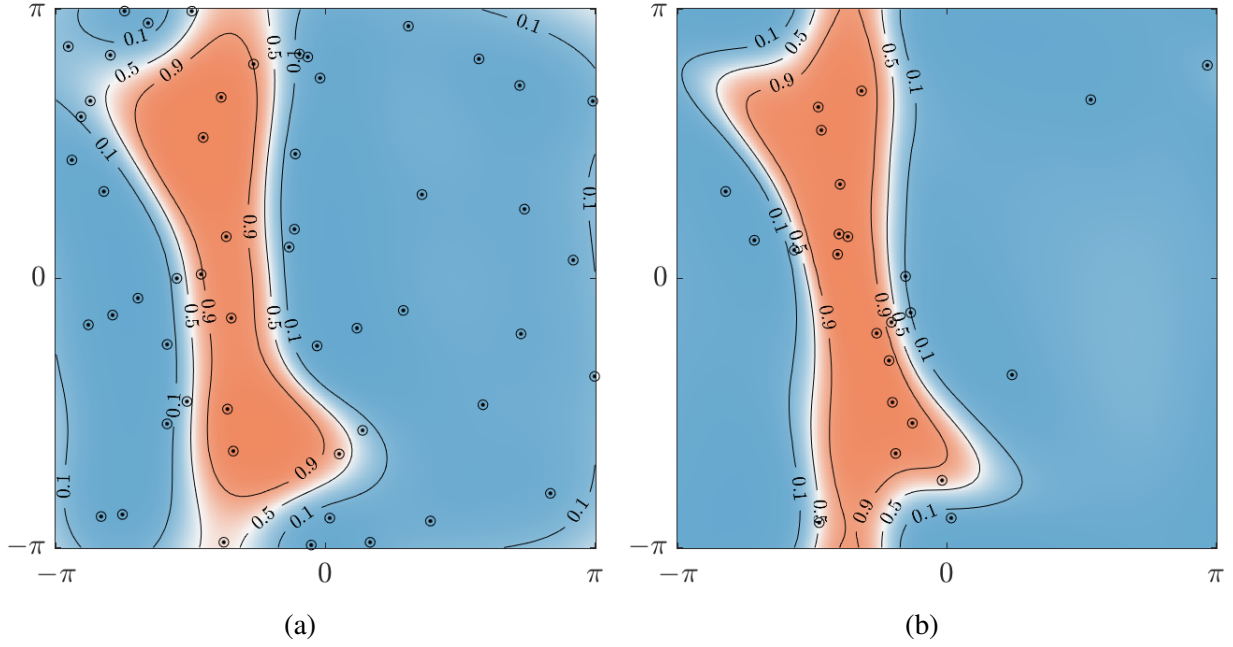


Figure 3.9: Import vector machine [9] estimating the conditional probability $P(\mathbf{y} = 1|\mathbf{x})$ using (a) the rational quadratic kernel and (b) the FK kernel.

planar robot. Fig. 3.9a and 3.9b shows two IVM models using the rational quadratic and the FK kernel, respectively. In both of these cases, the probability that a point $\mathbf{x}$ has label $\mathbf{y} = 1$ is shown. We can see a few advantages of the FK kernel over the rational quadratic kernel. First, we can see that the rational quadratic kernel required significantly more support points to represent the model. Second, we can also observe that the edges of the space are not represented well with the rational quadratic kernel, because we can see that the $P(\mathbf{y} = 1|\mathbf{x}) = 0.1$ contour exists near the edges of the sampling space. On the other hand, with the FK kernel, the model does not have issues with the edges of the sampling space. This is most likely due to the fact that the FK kernel represents differences in the workspace while the rational quadratic kernel only accounts for the joint values themselves.

The benefit of IVM is that class probabilities may be predicted, which may assist in avoiding low confidence regions (i.e., where $P(\mathbf{y} = 1|\mathbf{x}) \approx P(\mathbf{y} = 0|\mathbf{x})$) or suggest areas where denser sampling is required. In both of these models, $P(\mathbf{y} = 1|\mathbf{x}) = 0.5$ is roughly where the

boundary between $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$ should be. Thus, these IVM models can still be used for binary classification if the predicted probability is thresholded at 0.5.

Algorithm 5: Import Vector Machine [9]

Input: $\mathcal{X}$: training dataset of configurations; $\mathbf{y}$: collision status labels; $\mathbf{K}$: kernel matrix for $\mathcal{X}$
Parameters: λ : regularization parameter; $iter_{max}$: maximum number of iterations
Output: $\boldsymbol{\alpha}$: $\mathbb{R}^N$ weight vector; $\mathcal{S}$: support set
// Initialize elements
1 $\mathcal{S} \leftarrow \emptyset, \mathcal{R} \leftarrow \mathcal{X}$
// Compute loss function step
2 **while** $H(\mathcal{X})$ is not converged **OR** $iter_{max}$ reached **do**
3 **for each** $\mathcal{X}_l \in \mathcal{R}$ **do**
4 $H(\mathcal{X}_l) = -\mathbf{y}^\top \mathbf{K}_a^l \boldsymbol{\alpha}^l + \mathbf{1}^\top \log(1 + \exp(\mathbf{K}_a^l \boldsymbol{\alpha}^l)) + \frac{\lambda}{2} \boldsymbol{\alpha}^{l\top} \mathbf{K}_q^l \boldsymbol{\alpha}^l$
5 where $\boldsymbol{\alpha}^l$ is as specified in Eq. 3.10,
6 $\mathbf{K}_a^l$ is the kernel matrix between $\mathcal{X}$ and $\mathcal{S} \cup \{\mathcal{X}_l\}$,
7 and $\mathbf{K}_q^l$ is the kernel matrix for $\mathcal{S} \cup \{\mathcal{X}_l\}$ itself
8 $\mathcal{X}_{l*} = \min_{\mathcal{X} \in \mathcal{R}} H(\mathcal{X})$
9 $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathcal{X}_{l*}\}, \mathcal{R} \leftarrow \mathcal{R} \setminus \{\mathcal{X}_{l*}\}$
10 **return** $\boldsymbol{\alpha}, \mathcal{S}$

3.6 Acknowledgments

Chapter 3, in part, is a reprint of material from N. Das, M. C. Yip, “Forward Kinematics Kernel for Improved Proxy Collision Checking,” in *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2349-2356, April 2020. The dissertation author is the primary author of this paper.

©2020 IEEE.

Chapter 4

Estimating Distance to Collision

Evaluating whether a robot manipulator in a given configuration is either in $\mathcal{C}_{obs}$ or in $\mathcal{C}_{free}$ typically involves binary collision detection. However, there are situations where determining the distance to collision in the workspace is useful, as closer proximities may suggest the robot is more at risk to colliding with an obstacle. This distance to collision is useful in defining robot control rules for safe robot motion in potentially unpredictable environments [29], such as in cases where the robots are working in close proximity with humans [86]. In addition to collision avoidance for robots [29], other applications requiring accurate representation of the distance and penetration depth between two objects include contact resolution in simulation (where objects must respond to collisions realistically) [64] and force feedback in haptic rendering (where forces are generated for a human manipulating a virtual or teleoperated tool) [87].

A robot's distance to collision is typically defined to be the minimum distance between all of the robot's links and all of the workspace obstacles [29, 57]. Assuming that the i^{th} link of a robot in configuration $\mathbf{x}$ is represented as $\mathcal{R}_i(\mathbf{x})$ and the j^{th} workspace obstacle is o_j , we can define the distance to collision for a given environment as

$$d_{\mathcal{R}}(\mathbf{x}) = \min_{i,j} dist(\mathcal{R}_i(\mathbf{x}), o_j) \quad (4.1)$$

where $dist(\cdot, \cdot)$ provides a scalar distance between two non-intersecting bodies. When the two bodies are in collision, $dist(\cdot, \cdot)$ provides the penetration depth (the minimum amount intersecting bodies must move to be out of collision [87]) as a negative value. Note that $sign(d_{\mathcal{R}}(\mathbf{x}))$ is positive when no robot links are in collision and negative when at least one link is in collision. Thus, determining $d_{\mathcal{R}}(\mathbf{x})$ also efficiently provides the collision status for a certain configuration.

As with collision detection, distance estimation is a time consuming operation, with time complexity as high as $O(m^6n^6)$ for generalized distance calculations, where m and n are the number of vertices in two different bodies [8]. Furthermore, reliable collision avoidance often requires the extensive use of external sensors [29]. The sensors that provide obstacle information, such as cameras or depth sensors, are a source of uncertainty in the distance measurements [88]. Thus, modeling or accounting for this uncertainty is a necessity in distance estimation. As there are many applications that utilize distances, in this chapter, we seek to create a model to efficiently and accurately estimate $d_{\mathcal{R}}(\cdot)$ while accounting for potential uncertainty in the distance estimations.

4.1 Approaches to Model Distances

The standard approach to calculating distances involves first using sensors to obtain the locations of obstacles, and then approximating the robot geometry and workspace obstacles to actually compute the distance, which is a considerably expensive process if there are many pairwise checks to perform [57, 89]. Some techniques attempt to reduce this computational effort through approximate methods, such as by using values from a depth sensor to approximate a distance or by creating a proxy machine learning model using exemplar data. This section describes some geometric, depth perception-based, and machine learning techniques.

4.1.1 Geometric Approaches

A straightforward approach to calculating distance to collision is to approximate the objects and use a geometric distance method between them. Han et al. [57] calculate $d_{\mathcal{R}}(\cdot)$ by taking an approximate representation of both the robot links and the workspace obstacles and using the GJK [5] algorithm to compute the distance. GJK requires all objects to be a convex shape, which are shapes that fully contain the line that connects two points inside of the shape. A Baxter robot is used for their simulations, where each link is approximated as a cylinder. Each workspace obstacle is approximated using convex shapes. In their implementation, the authors represent obstacles by converting point clouds acquired from a depth camera into multiple small cubes. While their technique can accurately represent distances for a robot and obstacles observed with a depth camera, breaking obstacles into numerous small polyhedra increases the potential number of pairwise checks that need to be performed, and there is no measure of uncertainty with this approach.

4.1.2 Perception-Based Distance Fields

Flacco et al. create a method to estimate distance to collision for robot manipulators based on depth sensors [29]. For any given point $\mathbf{P}$ in the workspace, distance equations are defined using a depth map, which is an image with depth values evaluated for each pixel. Depending on whether the depth of an obstacle point $\mathbf{O}$ is greater than or less than that of $\mathbf{P}$, the authors either estimate the distance between $\mathbf{P}$ and $\mathbf{O}$ or the distance between $\mathbf{P}$ and the occluded region behind $\mathbf{O}$ (assuming the occluded region is an obstacle), respectively. As the minimum distance to an obstacle can be determined for any point in the workspace, this technique creates a distance field from a depth sensor.

Control points are placed along the robot manipulator (similar to the potential fields method [24]), and the locations of these control points in the distance field are used to create

artificial repulsive forces to drive the robot manipulator away from obstacles. The authors claim that the distance values computed with their technique are satisfactory for the purpose of collision avoidance but are not to be considered as actual Cartesian distances. This limitation precludes usage of this technique beyond the purpose of defining repulsive forces.

Furthermore, their distance estimation technique has no measure of sensor or model noise. However, as the primary purpose of their technique is to define repulsive forces away from obstacles, the authors take an average over all obstacles' repulsive fields to reduce any potential noise originating from the depth sensor.

4.1.3 Support Vector Machines

Pan et al. use an incremental SVM to model the C_{obs} region that exists between two bodies. Binary collision checks are performed on random configurations of one object relative to the other to form a precise decision boundary. By determining the distance between a query configuration and the decision boundary, their SVM model is used to estimate the $d_{\mathcal{R}}(\cdot)$. There is no closed form solution for the distance to the decision boundary for nonlinear SVM, so the authors employ sequential quadratic programming, a constrained nonlinear optimization technique, to find the closest configuration on the boundary. This closest point is used to calculate the distance [64]. As distances in the C-space do not generally equate to distances in the workspace, a different distance metric is used (the displacement distance metric), which involves determining application-specific weighting terms [64]. Additionally, this approach requires a unique SVM model between each pair of obstacles, which may make applying this technique to robot manipulators or environments with multiple obstacles more computationally expensive. Finally, this technique assumes the state of each object is perfectly known, which means there is no measure of uncertainty in this model.

4.1.4 Neural Networks

RelaxedIK estimates self-collision costs by using line segments to represent the links. Distances between the links are used to approximate the distances. These distance computations are further approximated using a neural network [25]. Their method successfully estimates these self-collision costs which enables usage in optimization problems. However, as with many neural network approaches, this technique requires a large amount of labeled data and a significant amount of time to train, precluding this approach from being applied to instances beyond distance to self-collision.

ClearanceNet is another neural network-based approach to estimate $d_{\mathcal{R}}(\cdot)$. The input to this network is the robot configuration and a parameterized representation of the environment. Their approach achieves over 90% accuracy if the network’s output is used to predict the collision status [60]. Once again, the neural network required a large amount of data to train, and since the locations of the workspace obstacle are parameterized, generalizing the implementation to new environments may require even more data and possibly a restructuring of the neural network implementation.

4.2 Gaussian Process Regression for Distance Estimation

Regression models can produce a continuous scalar output given a possibly multi-dimensional input. A common perspective of regression analysis is to view the true output as a sum of its input-dependent expected value and some error term, and using the expected value as a predictor of the output for new queries [90]. In the context of distance evaluation, we can assume that the measured, sensor-based distance is a sum of the true distance and a noise term: $\tilde{d}_{\mathcal{R}}(\mathbf{x}) = d_{\mathcal{R}}(\mathbf{x}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \eta^2)$ and η^2 is the measurement noise variance.

Nonparametric models such as kernel regression (KR) and Gaussian processes (GP) allow distance maps to scale to the complexity of the environment and to account for changing

environments over time. This is the ideal scenario as many environments a robot must work in are nonstationary and may include other agents.

Gaussian processes (GPs) can be used for regression problems. GPs are kernel-based models that fit a Gaussian distribution over the functions that can potentially fit a set of training data [11].

A GP is completely defined by a mean and covariance (or kernel) function. The prior distribution considers the space of all possible functions that can be defined using a prior mean (which is sometimes, but not always, 0) and a positive definite kernel function $k(\cdot, \cdot)$. A posterior distribution is defined given training data to condition the prior distribution. Assuming that $\mathcal{X}$ is a training dataset, $\mathbf{y}$ is a vector of corresponding training outputs, the posterior distribution for the regression output y for a query $\mathbf{x}$ is

$$P(y|\mathcal{X}, \mathbf{y}, \mathbf{x}) = \mathcal{N}(k(\mathbf{x}, \mathcal{X})[k(\mathcal{X}, \mathcal{X}) + \eta^2 \mathbf{I}]^{-1} \mathbf{y}, k(\mathbf{x}, \mathbf{x}) - k(\mathbf{x}, \mathcal{X})[k(\mathcal{X}, \mathcal{X}) + \eta^2 \mathbf{I}]^{-1} k(\mathcal{X}, \mathbf{x})) \quad (4.2)$$

where η^2 is the measurement noise variance in the measurements of $\mathbf{y}$ [11]. This noise term can either be selected prior to model fitting if it is known or estimated in advance, or it can be treated as a hyperparameter that can be estimated based on the training dataset [11].

For a single query $\mathbf{x}$, the posterior mean according to Eq. 4.2, $\mu(\mathbf{x})$, can serve as the regression output. A benefit of GPs is some measure of confidence may be obtained with the variance of the posterior distribution in Eq. 4.2, represented now as $\sigma^2(\mathbf{x})$. A confidence intervals (CI) is a set of bounds containing the true output with some probability. The bounds of a two-sided CI are $\mu(\mathbf{x}) \pm z\sigma(\mathbf{x})$, where z corresponds to a number of standard deviations from the mean. z can be selected to contain a certain proportion of the area under a standard Gaussian distribution and may be determined from Gaussian distribution tables [91] or statistics software. For example, if $z = 1.96$, the two-sided CI contains 95% of the area, which intuitively means the CI has a probability of 95% of containing the true regression output [91]. Selecting larger z values to

increase the probability of containing the true output will have a tradeoff of looser bounds on the CI.

The inversion that occurs in Eq. 4.2 may be costly for larger training datasets. This issue may be addressed by reduced rank approximations or using subsets of the training dataset [11], which are handled by many GP software libraries, and for which robot-specific GP modeling strategies have been proposed [92].

We propose to fit a GP model to a training set of configurations $\mathcal{X}$ because of a GP’s flexibility in modeling arbitrary datasets and its capability in accounting for uncertainty. We further propose the use of the FK kernel in the GP due to the boost in performance this kernel has proven to provide in binary proxy collision checking applications [7].

Uniformly randomly sampled robot configurations are used as the training set $\mathcal{X}$. Each configuration in the set is labeled with a noisy distance measurement (calculated using the GJK/EPA [5, 10] geometric methods in our implementation), and the labels are stored in vector $\mathbf{y}$. To simulate noisy, sensor-based measurements, each label in $\mathbf{y}$ is created by taking a sum of the true distance and Gaussian noise: $\tilde{d}_{\mathcal{R}}(\mathbf{x}) = d_{\mathcal{R}}(\mathbf{x}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \eta^2)$.

All methods were written in MATLAB. In the implementation of the FK kernel, a control point is placed at the distal end of each link in the manipulator as was done in [7].

4.3 Confidence-Based Hybrid Model

As described in Section 4.2, two-sided CIs may be defined based on the posterior distribution standard deviation. Similarly, a one-sided CI may be defined, e.g., $[\mu(\mathbf{x}) - z\sigma(\mathbf{x}), \infty)$ is the CI with a lower bound on what the true regression output is for configuration $\mathbf{x}$ with some degree of confidence. In this case, $z = 1.64$ means that there is 95% probability that the true regression output is greater than $\mu(\mathbf{x}) - 1.64\sigma(\mathbf{x})$. A one-sided lower-bounded CI may be used when a bound tighter than in the two-sided CI case is desired and an upper bound on the true

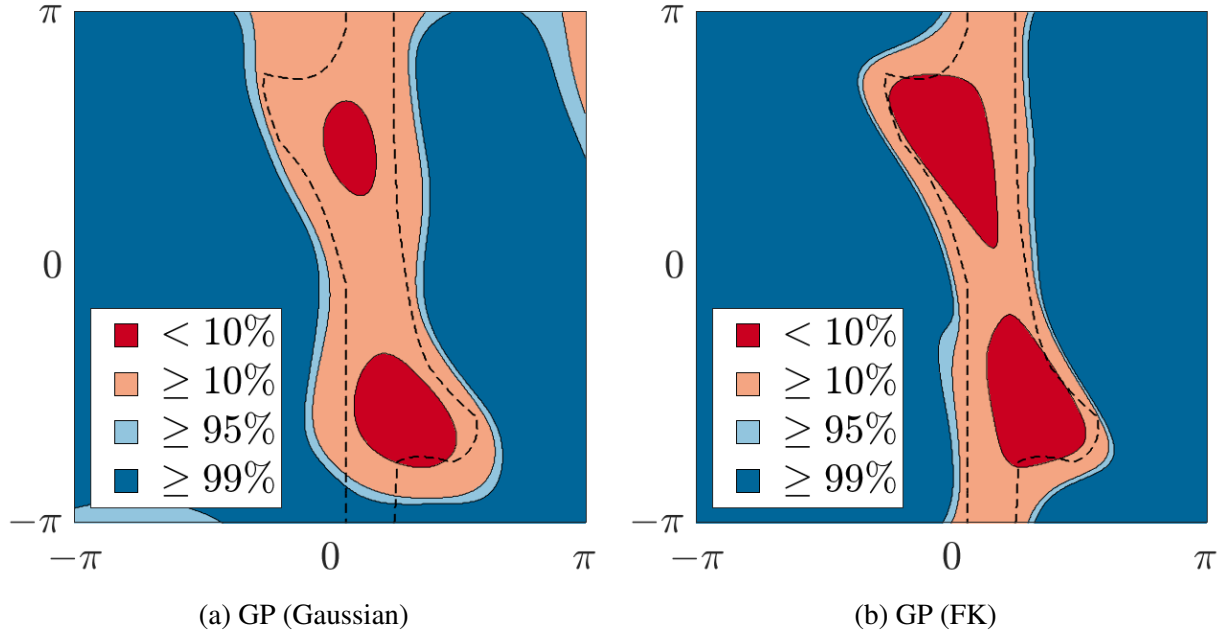


Figure 4.1: The colored regions represent different levels of confidence of the GP models for $d_{\mathcal{R}}(\cdot)$ being greater than 0 for a 2 DOF robot, where each axis represents one DOF. The ground truth is shown in Fig. 1.2. The dashed black curve is where $d_{\mathcal{R}}(\cdot) = 0$ using GJK. The GPs use (a) the Gaussian kernel and (b) the FK kernel.

value is not as important as a lower bound.

In Fig. 4.1, we visualize the levels of confidence for a 2 DOF robot having $d_{\mathcal{R}}(\mathbf{x}) \geq 0$ when using either GP (Gaussian) or GP (FK). The dashed black curve is where $d_{\mathcal{R}}(\mathbf{x}) = 0$. We can see that the confidence regions for GP (FK) closely align with the C-space obstacle contour, while GP (Gaussian) does not match as closely. In fact, GP (Gaussian) has pockets of inappropriate confidence, such as decreased confidence in the corners of the shown C-space or high confidence at the bottom of the C-space obstacle. On the other hand, GP (FK) does not have these issues and has increasing confidence of $d_{\mathcal{R}}(\mathbf{x})$ truly being greater than 0 for configurations farther away from the C-space obstacle. GP (FK)’s more appropriate assignment of confidence suggests that we can create a hybrid model that switches distance estimation methods depending on the GP’s level of confidence.

Using the lower-bounded CI, we define a hybrid distance estimation technique that uses

the GP mean as the predicted distance in areas with a high confidence of being collision-free and a more expensive method in other areas:

$$\mu^{(P)}(\mathbf{x}) = \begin{cases} \mu(\mathbf{x}), & \text{if } \mu(\mathbf{x}) - z\sigma(\mathbf{x}) \geq 0 \\ \frac{1}{N_{\tilde{d}}} \sum_{i=1}^{N_{\tilde{d}}} \tilde{d}_{\mathcal{R}}(\mathbf{x}), & \text{otherwise} \end{cases} \quad (4.3)$$

where P denotes the desired level of confidence associated with the number of standard deviations z (e.g., $P = 95\%$ for $z = 1.64$), and $N_{\tilde{d}}$ is the number of samples to collect from $\tilde{d}_{\mathcal{R}}(\cdot)$ to potentially reduce the effects of sensor noise. The lower bound in Eq. 4.3 may of course be nonzero if a more conservative model is desired. This hybrid model may be useful when cheaper distance evaluations from the GP may be used when working far from the obstacle, but more a more meticulous method may be required when in close proximity with obstacles.

4.4 Performance of Gaussian Processes

We evaluate the performance of our proposed GP-based distance estimation technique against three other approaches: the geometric approach GJK/EPA (ground truth), a GP with a typical Gaussian kernel, and KR.

Kernel regression (KR) [12, 13] is one example of a regression technique, where a probability distribution is estimated and used to compute an expected value. A training set of N input samples $\mathcal{X}$ and their corresponding outputs $\mathbf{y}$ can be used to estimate a probability distribution of the outputs conditioned on the inputs. This probability distribution may then be used to calculate the expected value of query inputs by taking a weighted sum of all training datapoints to predict the output of a query point. The equation for prediction for a single query point $\mathbf{x}$ is

$$\mu_{KR}(\mathbf{x}) = \frac{\sum_{i=1}^N k(\mathcal{X}_i, \mathbf{x}) \mathbf{y}_i}{\sum_{i=1}^N k(\mathcal{X}_i, \mathbf{x})} \quad (4.4)$$

where $k(\cdot, \cdot)$ is some kernel function. KR does not need to learn any weights to estimate the expected value of $d_{\mathcal{R}}(\cdot)$, so a training procedure is not required. KR typically uses a kernel that has equal influence in each direction, such as the Gaussian kernel [11]. The FK kernel is not a function of the distance between its inputs and changes shape depending on where the kernel is evaluated, so it is not well suited for the KR formulation in Eq. 4.4.

We begin by comparing the accuracy and timing of each technique before applying these techniques to path optimization. In each experiment, a 7 DOF planar robot manipulator is modeled with unit length link and a rectangular link profile, and a polygonal workspace obstacle is randomly placed in the manipulator’s reachable workspace. We set the measurement noise to be $\eta = 0.05$ when creating each dataset. To get an idea of the level of this noise, for an obstacle one link length away from the robot (i.e., $d_{\mathcal{R}}(\mathbf{x}) = 1$), 99% of the measured values could be anywhere in the range $[0.85, 1.15]$. We assume the scale of the robot and environments are arbitrary, so units for any measurements of length are omitted.

4.4.1 Accuracy and Query Timing

In this section, we evaluate the GP model’s performance in terms of its accuracy and computation timing. Metrics that we utilize to evaluate performance include mean squared error (MSE) and model query time. MSE is defined as

$$MSE = \frac{1}{\hat{N}} \sum_{i=1}^{\hat{N}} \left| \hat{\mathbf{y}}_i - \mu(\hat{\mathcal{X}}_i) \right|^2 \quad (4.5)$$

where the test set $\hat{\mathcal{X}}$ has $\hat{N}$ elements, $\hat{\mathbf{y}}$ is the set of true labels for $\hat{\mathcal{X}}$, and $\mu(\cdot)$ provides the predicted distance using one of the approximation techniques. Note that while each model is trained on noisy distance evaluations (i.e., $\tilde{d}_{\mathcal{R}}(\cdot)$), we use noise-free distance evaluations (i.e., $d_{\mathcal{R}}(\cdot)$) to obtain $\hat{\mathbf{y}}$ when computing the MSE.

Because the volumes of the in-collision and collision-free regions in the C-space may

Table 4.1: Performance in terms of MSE and query time for noise-free GJK/EPA, KR, GP (Gaussian), and GP (FK).

Method	MSE	TPMSE	TNMSE	Query Time (μ s)
GJK/EPA	—	—	—	466.9
KR	0.78	0.71	2.76	10.4
GP (Gaussian)	0.50	0.41	2.92	1.8
GP (FK)	0.06	0.05	0.12	6.7

not be equal, the overall MSE may misrepresent the MSE within each region. Thus, we also evaluate MSE on subsets of the test data for which $\hat{\mathbf{y}}$ is either positive (collision-free) or negative (in-collision), which we refer to as true positive MSE (TPMSE) and true negative MSE (TNMSE), respectively. TPMSE and TNMSE are defined as

$$TPMSE = \frac{1}{\hat{N}_+} \sum_{i: \hat{\mathbf{y}}_i > 0} \left| \hat{\mathbf{y}}_i - \mu(\hat{\mathcal{X}}_i) \right|^2 \quad (4.6)$$

$$TNMSE = \frac{1}{\hat{N}_-} \sum_{i: \hat{\mathbf{y}}_i < 0} \left| \hat{\mathbf{y}}_i - \mu(\hat{\mathcal{X}}_i) \right|^2 \quad (4.7)$$

where $\hat{N}_+$ and $\hat{N}_-$ denote the sizes of the subsets for TPMSE and TNMSE, respectively.

Table 4.1 shows the performance of the models in terms of MSE and query time of each of the methods. Note that the average distance to collision is 1.7. We can see the MSEs when using the GP method with the FK kernel are significantly lower (i.e., closer to the GJK/EPA ground truth values) than the other two approximation methods. The FK kernel gives a large boost in modeling accuracy compared to the Gaussian kernel, as was expected [7], due to the FK kernel’s strong relation with locations in the workspace. The KR method performs worse than GP (Gaussian) in terms of MSE, probably because all points in $\mathcal{X}$ have equal importance with the KR method while GPs assign weights to each point. Fig. 4.2 shows histograms of absolute errors in estimation of $d_{\mathcal{R}}(\cdot)$ for GP (Gaussian), GP (FK), and KR. We can see that GP (FK) has a larger proportion of lower errors compared to the other methods.

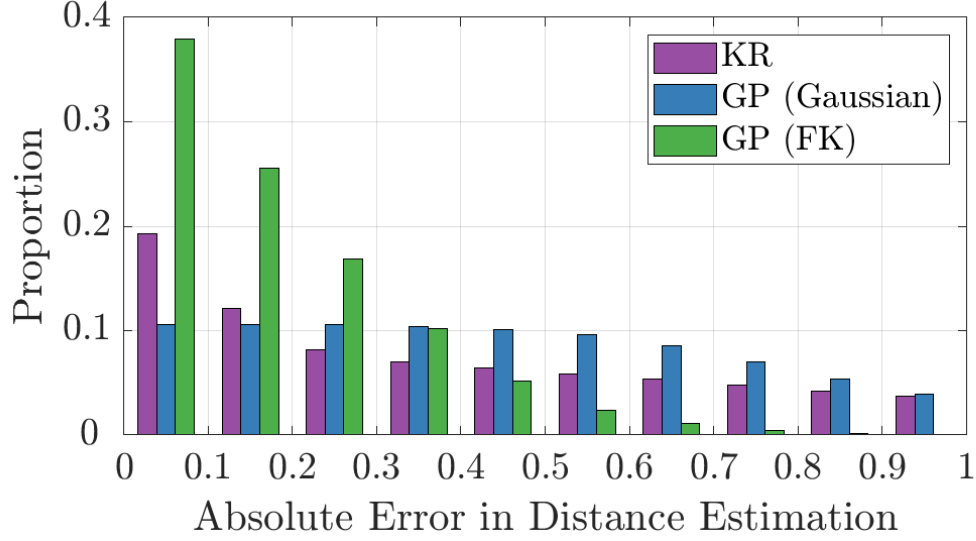


Figure 4.2: Histogram of absolute errors in distance estimation of $d_{\mathcal{R}}(\cdot)$ when using KR and GP models with the Gaussian and FK kernel. Models with a higher proportion of lower error are better.

4.4.2 Path Optimization with Clearance Constraint

In this section, we utilize the distance estimation techniques described above in a path optimization application. One purpose of path optimization is to minimize a cost, such as path length, while satisfying constraints, such as keeping a certain distance from obstacles, given a motion plan generated by some other method [93]. In our experiment, we structure our optimization problem similar to the trajectory optimization formulation in TrajOpt [93]. More specifically, the cost is defined as a sum of distances between waypoints under the inequality constraint of distance to collision.

In our implementation, we minimize the manipulator’s end effector trajectory rather than the C-space trajectory in order to avoid trivial solutions such as rotating the manipulator the other way around an obstacle. In this section, we represent the Cartesian position of the end effector in configuration $\mathbf{x}$ as $FK_{EE}(\mathbf{x})$. Our optimization problem to find a sequence of T configurations

$\Theta = [\theta_1, \dots, \theta_T]$ where $\theta_t \in \mathbb{R}^D$ is thus

$$\begin{aligned}
\Theta^* = \underset{\Theta}{\operatorname{argmin}} \quad & L(\Theta) \\
\text{subject to} \quad & \theta_1 = \theta_{start}, \\
& \theta_T = \theta_{goal}, \\
& \|\theta_{t+1} - \theta_t\| \leq \Delta\theta \quad \forall t \in [1, \dots, T-1], \\
& \min_{t \in [1, \dots, T]} d_{\mathcal{R}}(\theta_t) \geq d_{min}
\end{aligned} \tag{4.8}$$

where Θ^* is the optimized motion plan, $L(\Theta) = \sum_{t=1}^{T-1} \|FK_{EE}(\theta_{t+1}) - FK_{EE}(\theta_t)\|$ is the distance traveled by the end effector when executing motion plan Θ , θ_{start} and θ_{goal} are desired start and goal configurations, $\Delta\theta$ prevents large joint changes, and d_{min} is a desired amount of clearance to maintain from a workspace obstacle. $d_{\mathcal{R}}(\theta_t)$ in the constraint is replaced by each model's estimate.

The constrained optimization problem solver `fmincon()` in MATLAB is used to solve Eq. 4.8. θ_{start} and θ_{goal} are randomized on either side of the obstacle to require obstacle avoidance in the motion plan. The initial motion plan is acquired using a Rapidly-Exploring Random Tree motion planner [1], which is then used as the initial seed for the optimization. d_{min} is set to be 0.2 in these experiments.

Table 4.2: Average optimization time, end effector path length, and minimum satisfaction of clearance constraint after optimization of Eq. 4.8 (clearance constraint) when using GJK/EPA [5, 10], GJK/EPA with noise, GP (Gaussian kernel) [11], GP (FK kernel) [11, 7], and KR [12, 13] to estimate $d_{\mathcal{R}}(\cdot)$.

Method	Optim. Time (s)	$L(\Theta)$	$\min d_{\mathcal{R}}(\Theta) - d_{min}$
GJK/EPA	864.1	14.4	0.16
GJK/EPA (Noisy)	240.4	16.5	0.23
KR	29.1	26.9	0.29
GP (Gaussian)	49.2	16.1	-0.12
GP (FK)	106.8	13.7	0.06

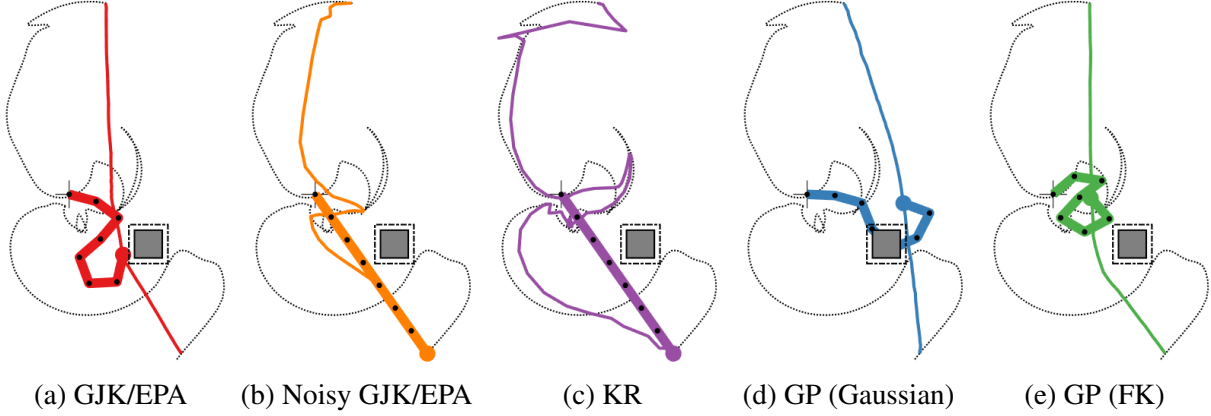


Figure 4.3: Examples of end effector paths before (black curves) and after (colored curves) optimization of Eq. 4.8 (clearance constraint of 0.2) when using (a) GJK/EPA, (b) GJK/EPA with noise, (c) KR, (d) GP with Gaussian kernel, (e) GP with FK kernel for distance estimation. The configuration along the path with the worst distance to collision and the amount of clearance required from the obstacle are also shown.

Table 4.2 shows optimization times, optimized path lengths, and the worst satisfaction of the clearance constraint $\min d_{\mathcal{R}}(\Theta) - d_{min}$. Note that $\min d_{\mathcal{R}}(\Theta) - d_{min}$ is positive if the constraint is satisfied. While optimization with GP (Gaussian) and KR is faster than when optimizing with GP (FK), the optimized result is often either in collision or has a long end effector path length. All learning-based methods are significantly faster than GJK/EPA, but using the GP (FK) provides the safest results.

Fig. 4.3 shows various examples of the optimized paths using the various methods to approximate $d_{\mathcal{R}}(\cdot)$ in Eq. 4.8, including a case where the noisy distance measurements are used directly. The original unoptimized end effector path is shown as the dotted curve while the optimized paths are the thicker, colored curves. The configuration along each optimized path with the lowest distance to collision (according to GJK) is displayed, and the amount of desired clearance is represented as the dotted box around the workspace obstacle. It is evident that the optimized trajectories when using the GP (FK) is fairly similar to those when using GJK/EPA, while the other methods yield less optimal paths such as ones with collisions or with longer path lengths.

4.4.3 Path Optimization with Clearance Maximization

In this section, we once again apply the distance estimation techniques to path optimization. However, rather than utilizing the distance to collision as a constraint, we include distance in the objective function to increase a trajectory’s clearance. We weight the path length objective function used in Eq. 4.8 with an exponentiated distance, which changes the optimization problem to

$$\begin{aligned}
\Theta^* &= \underset{\Theta}{\operatorname{argmin}} \quad C(\Theta)L(\Theta) \\
&\text{subject to} \quad \theta_1 = \theta_{start}, \\
&\quad \theta_T = \theta_{goal}, \\
&\quad \|\theta_{t+1} - \theta_t\| \leq \Delta\theta \quad \forall t \in [1, \dots, T-1]
\end{aligned} \tag{4.9}$$

where $C(\Theta) = \exp(-\min d_{\mathcal{R}}(\Theta))$. $d_{\mathcal{R}}(\Theta)$ in $C(\Theta)$ is replaced by each distance method’s estimate. The exponentiation makes $C(\Theta)$ large for trajectories causing egregious penetration into an obstacle and small for trajectories with larger clearances.

Table 4.3 shows average optimization times, path lengths, and minimum distances to collision for each method. GP (FK) has a higher amount of clearance than GP (Gaussian), KR, and noisy GJK/EPA methods, and in fact achieves results closer to the noise-free geometric method both in terms of clearance and path length despite being trained on noisy measurements. The noisy version of GJK/EPA yielded suboptimal solutions with large end effector path lengths

Table 4.3: Average optimization time, end effector path length, and minimum distance to collision after optimization of Eq. 4.9 (clearance maximization) when using GJK/EPA, GJK/EPA with noise, GP (Gaussian kernel), GP (FK kernel), and KR to estimate $d_{\mathcal{R}}(\cdot)$.

Method	Optim. Time (s)	$L(\Theta)$	$\min d_{\mathcal{R}}(\Theta)$
GJK/EPA	481.9	4.3	1.73
GJK/EPA (Noisy)	56.4	32.3	0.47
KR	28.7	9.3	0.03
GP (Gaussian)	15.7	4.5	-0.25
GP (FK)	51.0	4.9	1.24

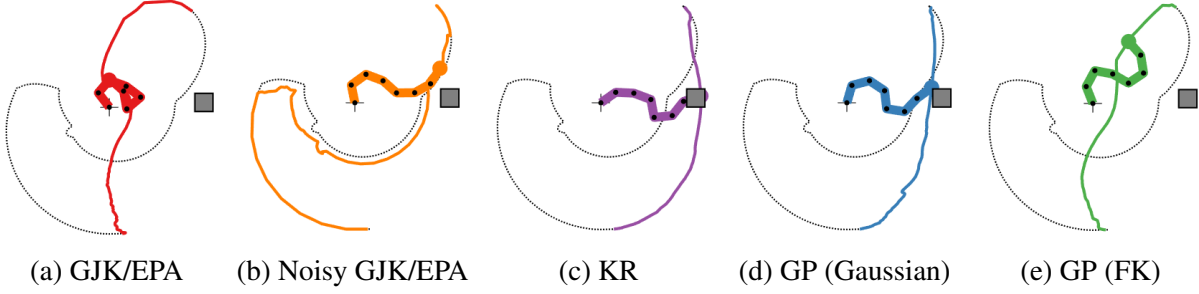


Figure 4.4: Examples of end effector paths before (black curves) and after (colored curves) optimization of Eq. 4.9 (clearance maximization) when using (a) GJK/EPA, (b) GJK/EPA with noise, (c) KR, (d) GP with Gaussian kernel, (e) GP with FK kernel for distance estimation. The configuration along the path with the worst distance to collision is also shown.

and smaller clearances than GP (FK). GP (Gaussian) and KR have shorter optimization times because the optimization solver often quickly converges to an invalid solution (as can be seen from the lower minimum distances to collision). The paths in Fig. 4.4 exemplify these results.

4.4.4 Confidence-Based Hybrid Model

We apply our proposed hybrid model to a scenario involving a narrow passage, where the manipulator must reach between two nearby obstacles. Inaccuracy in the distance estimation model may inadvertently lead to collision between these obstacles, so the hybrid model's predictions can potentially handle this scenario by switching to the geometric method if the GP model is not confident.

Fig. 4.5a shows an example optimized trajectory achieved with the hybrid model $\mu_{FK}^{(95)}(\cdot)$. The plotted robot configuration denotes the transition point where the hybrid model switches from GP-based predictions to noisy, sensor-based distance evaluations $\tilde{d}_{\mathcal{R}}(\cdot)$. We used $N_{\tilde{d}} = 5$ evaluations of $\tilde{d}_{\mathcal{R}}(\cdot)$ per configuration when the hybrid model did not use a GP-based prediction. The colors of the path denote the confidence level of the GP that the true distance to collision is greater than 0, which we determine via MATLAB's `normcdf()`. We can observe that the GP model is fairly confident when far from the obstacles, but the confidence level rapidly drops when the manipulator needs to move into the narrow passage.

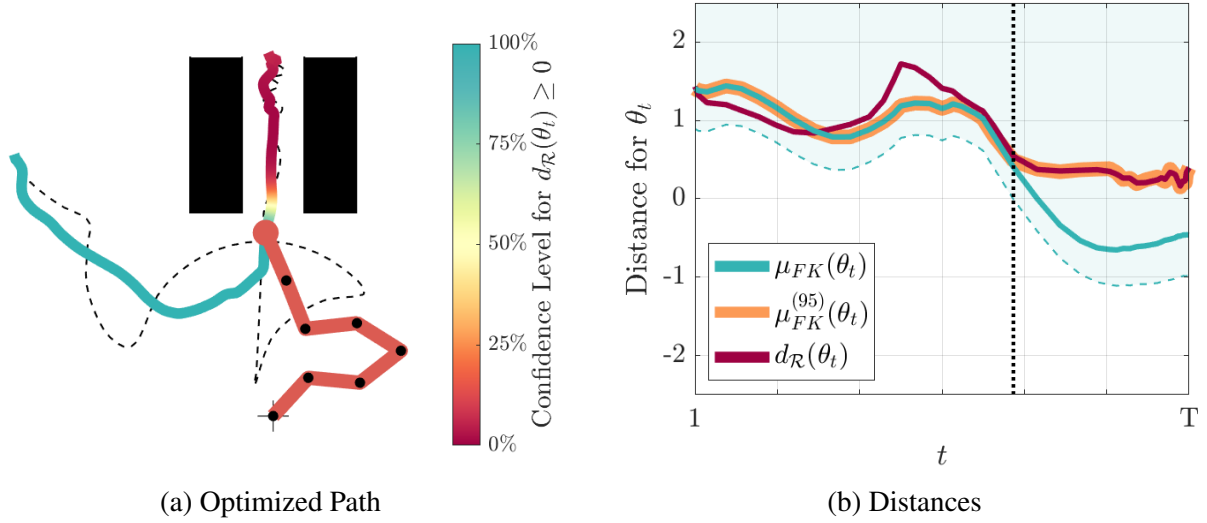


Figure 4.5: (a) The optimized path determined using the hybrid model $\mu_{FK}^{(95)}(\cdot)$ is shown, where the GP’s confidence level that the true distance to collision is greater than 0 is represented by the color of the optimized path. The plotted robot configuration marks the point where the hybrid model switches from the GP-based predictions to the sensor-based method. (b) The distances based on the GP model with the FK kernel ($\mu_{FK}(\cdot)$), the hybrid model ($\mu_{FK}^{(95)}(\cdot)$), and the noise-free geometric method $d_{\mathcal{R}}(\cdot)$ for the trajectory $\theta_{1:T}$ are shown. The shaded region represents the lower-bounded CI for the GP model, and the vertical line shows where the hybrid model switches from GP-based to sensor-based predictions.

Fig. 4.5b shows the GP-based distance estimates $\mu_{FK}(\Theta)$, the hybrid estimates $\mu_{FK}^{(95)}(\Theta)$, and the true distance to collision $d_{\mathcal{R}}(\Theta)$. The shaded region represents the lower-bounded CI for the GP model. We can see that $\mu_{FK}(\Theta)$ deviates from $d_{\mathcal{R}}(\Theta)$ as the distance decreases, but the hybrid model $\mu_{FK}^{(95)}(\Theta)$ manages to produce accurate measurements of distance by switching to the noisy distance measurements when the GP loses confidence.

4.5 Acknowledgments

Chapter 4, in part, is currently being prepared by Nikhil Das and Michael C. Yip for submission for publication of the material. The dissertation author is the primary author of this paper.

Chapter 5

Conclusion

While a robot manipulator exists, moves, and acts in its workspace, problems involving robot motion or positioning are typically solved in its C-space, the space of all possible configurations the robot may take [24]. Constraints in the workspace, such as the presence of obstacles, are mapped to regions of infeasible configurations in the C-space, so a valid motion plan in the C-space would be a curve that circumnavigates these regions of infeasibility. The primary disadvantage of the concept of C-space is the mappings from workspace constraints to C-space constraints are typically impractical to determine exactly even for simple cases [24, 28]. Sampling-based algorithms successfully avoid having to create explicit models to determine solutions in C-space, but validity checks must be performed on thousands of configurations [37], causing a computational bottleneck for these algorithms [38, 34].

In this work, we investigate creating approximate explicit models of C-space to reduce the computational burden that arises from these validity checks. Our approach involves using machine learning techniques to efficiently create models for these spaces. We developed the Fastron algorithm which models the C_{obs} and C_{free} regions of a robot’s C-space, and we demonstrate that the Fastron model may be used as a fast proxy collision checker, such as to accelerate motion planning. To improve the accuracy of the Fastron model, we developed the forward kinematics

(FK) kernel, which provides a better similarity measure than isotropic kernels applied in C-space. With the new FK kernel, model accuracy increased to 95%, a significant improvement over the 75% accuracy achieved when using the original kernel function. The FK kernel has the positive definiteness property, allowing it to be considered a valid covariance function. Thus, we also successfully apply the FK kernel in Gaussian process regression to efficiently predict a robot's distance to collision 70-80 times faster than a state-of-the-art distance evaluation technique.

While the methods we proposed are steps toward tackling the challenge of a robot's C-space, there is still room for future work. For example, utilization of parallelization in training multiple models should enable faster query times or allow the modeling of more complex environments. Methods to determine the validity of continuous curves in C-space (rather than discrete configurations) should be investigated, which may accelerate the validation of motions in motion planning. Finally, techniques for smarter sampling to improve sample efficiency, either for training the C-space models or for more efficient sampling-based motion planning, should be explored.

Appendix A

Additional Work

A.1 GPU Acceleration

Graphics processing units (GPUs) are originally intended for quick image rendering. Their computing power surpasses that of computers’ typical central processing units (CPUs) due to the GPUs high parallel processing capabilities. Recently, GPUs’ computational power has been a great benefit to machine and deep learning [94]. In sampling-based motion planning, collision checking is a computational burden, taking up to 90% of the planning time. Of the collision checking time, the majority of time is spent evaluating the validity of edges, or the local path between C_{free} nodes [35]. There are primarily two classes of methods for evaluating the validity of an edge in graph-based motion planning: discretization and swept volume methods, both of which are computationally expensive [35]. In this section, we apply the computational power of GPUs to the edge collision checking challenge.

Realizing that numerous collision checks are performed per edge during motion planning, we implemented Fastron’s proxy collision checking on a GPU to parallelize numerous collision checks. We first implemented Fastron’s proxy collision checking in CUDA C++ and cuBLAS. This involved creating custom CUDA kernel functions so that the NVIDIA RTX 2070 GPU can

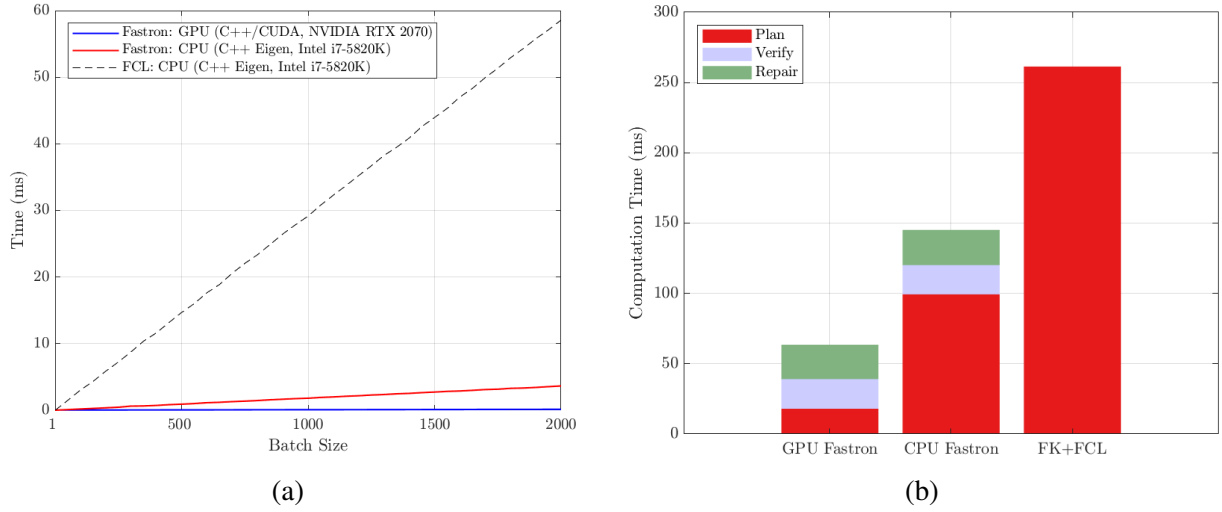


Figure A.1: (a) The timing of performing multiple collision checks using FCL [8] on a CPU, Fastron [6] on a CPU, and Fastron on a GPU. (b) Motion planning timing when using FCL on the CPU, Fastron on the CPU, or Fastron on the GPU for collision checks.

be utilized. All operations other than the parallelized proxy collision checks were performed on the CPU (Intel i7-5820K). Single-precision numbers were used to reduce the amount of data transfer between CPU and GPU.

CUDA allows code to be parallelized in a grid of thread blocks. We used blocks with 256 threads, where each thread evaluated one element of a $N \times N_q$ query kernel matrix, where N and N_q are the sizes of the support set and query set, respectively. The size of the grid is selected to fit the entire query kernel matrix. The rational quadratic kernel was implemented in these experiments.

We first compare the speed performance of the parallelized Fastron against the CPU version of Fastron and also against FCL [8], the default collision checker used in the MoveIt motion planning framework [66]. To perform a batch of 2000 collision checks, FCL took approximately 58.6 ms while Fastron took about 3.6 ms with the CPU. Fastron on the GPU took 0.14 ms for a batch of 2000 collision checks, clearly a significant improvement over the CPU-based techniques. Fig. A.1a shows the timing of performing a batch of collision checks with and without the GPU.

Next, we utilize parallelization during the edge collision checks during motion planning. We define our own edge validity checking method within the OMPL motion planning library [69] that performs collision checks along the edge in parallel on the GPU. Fig. A.1b shows the time to perform motion planning with an RRT when using the GPU to parallelize the edge checking. A collision-free path is determined about 5 times faster compared to using FCL on the CPU and about 3 times faster compared to using Fastron on the CPU.

Note that the verification and repairing stages are performed in the CPU for both GPU Fastron and CPU Fastron, and thus there is no change in timing for each of these stages. However, the actual planning time for GPU Fastron is about 6 times faster than for CPU Fastron and about 15 times faster than for FCL. As employing the edge checks on the GPU rather than the CPU should not change the accuracy of the predictions of the Fastron model, performing proxy collision checks in parallel on a GPU is an effective method to accelerate motion planning performance even further than CPU-based proxy collision checking.

A.2 Potential Fields in C-space

As described in Section 1.2 potential field methods use potential functions whose gradients dictate motion away from obstacles and toward a goal configuration. For the case of robot manipulators, potential fields methods are applied by utilizing control points placed along the manipulator in the workspace and using the distance between the obstacles and these control points to calculate the forces [24]. A common issue attributed to potential field methods is the existence of local minima, which may cause a robot to get stuck in certain configurations.

As an alternative, we can apply the potential fields method in a robot’s C-space. This would involve obtaining a C-space model with a gradient that directs the robot away from C-space obstacles and toward a goal.

Our algorithm Fastron [6] can provide a fairly accurate C-space model, and the structure

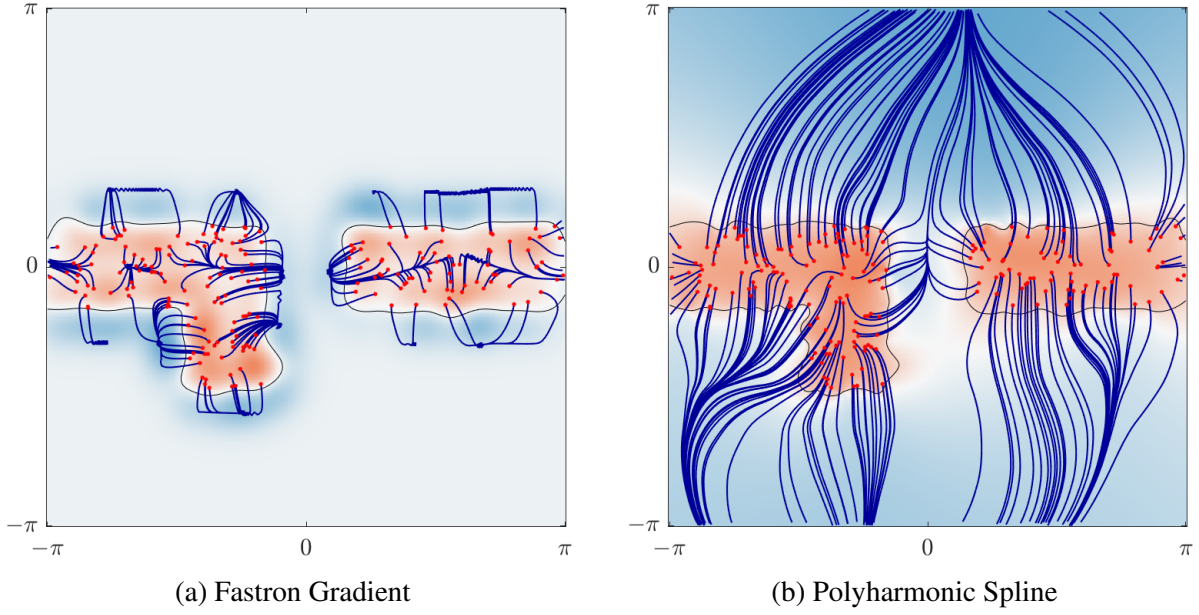


Figure A.2: (a) A streamline plot of the gradient of a Fastron model. (b) A streamline plot of the gradient of a polyharmonic spline.

of the model is differentiable prior to applying the threshold needed for binary classification. As seen in the streamline plot in Fig. A.2a, however, we can see that this model is already full of local minima even without any attractive forces. In some cases, following the gradient can lead a point out of the red region, but in other cases, following the gradient just leads the point deeper into the red.

Polyharmonic splines are smoothing functions of the form

$$s(\mathbf{x}) = p(\mathbf{x}) + \sum_{k=1}^N \mathbf{w}_k \phi_p(|\mathbf{x} - \mathbf{X}_i|) \quad (\text{A.1})$$

where $\mathbf{x} \in \mathbb{R}^d$, $p(\mathbf{x})$ is a low-dimensional polynomial, $\mathbf{w}$ is a set of weights, and $\phi_p(\cdot)$ is a radial basis function for interpolation [95]. This radial basis function is defined as

$$\phi_p(r) = \begin{cases} r^n, & n = 1, 3, \dots \\ r^n \log(r), & n = 2, 4, \dots \end{cases} \quad (\text{A.2})$$

where n is the order of the function.

Assuming $p(\mathbf{x})$ is a linear function (with weight vector $\mathbf{v} \in \mathbb{R}^{d+1}$), Eq. A.1 can be written in matrix form for the points in $\mathcal{X}$ as:

$$\mathbf{S} = \Phi \mathbf{w} + \mathbf{D} \mathbf{v} \quad (\text{A.3})$$

assuming each row of $\mathcal{X}$ is a datapoint, $\Phi_{ij} = \phi_p(|X_i - X_j|)$, and $\mathbf{D} = \begin{bmatrix} \mathbf{1} & \mathcal{X} \end{bmatrix}$. When considering the orthogonality conditions [95], $\mathbf{D}^\top \mathbf{v} = \mathbf{0}$, the weights for the polyharmonic spline can thus be solved by solving:

$$\begin{bmatrix} \mathbf{S} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \Phi & \mathbf{D} \\ \mathbf{D}^\top & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \mathbf{v} \end{bmatrix} \quad (\text{A.4})$$

Polyharmonic splines are solutions to the polyharmonic equation $\Delta^n f = 0$, where Δ is the Laplace operator. Recall that the Laplace operator is the divergence of the gradient of a function. Thus, an intuitive explanation for the case when $n = 1$ is for any neighborhood, the net flux is 0. This idea is advantageous for potential fields because it means there will be no areas where the gradient vector field converges. Fig. A.2b shows a streamline plot of the gradient of a polyharmonic spline. It is evident that, unlike in Fig. A.2a, there are no regions where a point will get stuck if it follows the gradient regardless of the origin of the point.

Harmonic functions have been used before for potential fields for mobile robots [96]. However, these functions have not been applied to C-space to the best of our knowledge. Using polyharmonic splines will allow for proxy collision checking (by checking the sign of the function) in C-space for high-dimensional robot, but also allow for efficient error correction or movements away from C-space obstacles.

Bibliography

- [1] S. M. LaValle, “Rapidly-Exploring Random Trees: A New Tool for Path Planning,” *Animal Genetics*, vol. 39, no. 5, pp. 561–563, 2008.
- [2] C. P. Diehl and G. Cauwenberghs, “SVM Incremental Learning, Adaptation and Optimization,” in *Proceedings of the International Joint Conference on Neural Networks*, pp. 2685–2690, 2003.
- [3] J. Pan and D. Manocha, “Efficient Configuration Space Construction and Optimization for Motion Planning,” *Engineering*, vol. 1, no. 1, pp. 046–057, 2015.
- [4] K. Huang, D. Zheng, J. Sun, Y. Hotta, K. Fujimoto, and S. Naoi, “Sparse learning for support vector classification,” *Pattern Recognition Letters*, vol. 31, no. 13, pp. 1944–1951, 2010.
- [5] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, “A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space,” *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.
- [6] N. Das and M. Yip, “Learning-Based Proxy Collision Detection for Robot Motion Planning Applications,” *IEEE Transactions on Robotics*, pp. 1–19, 2020.
- [7] N. Das and M. C. Yip, “Forward Kinematics Kernel for Improved Proxy Collision Checking,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2349–2356, 2020.
- [8] J. Pan, S. Chitta, and D. Manocha, “FCL: A general purpose library for collision and proximity queries,” *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3859–3866, 2012.
- [9] J. Zhu and T. Hastie, “Kernel logistic regression and the import vector machine,” *Advances in Neural Information Processing Systems*, 2002.
- [10] G. van den Bergen, “Proximity queries and penetration depth computation on 3d game objects,” *Game developers conference*, 2001.
- [11] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

- [12] E. Nadaraya, “On Estimating Regression,” *Theory of Probability and Its Applications*, vol. 9, no. 1, pp. 157–159, 1964.
- [13] G. S. Watson, “Smooth Regression Analysis,” *Sankhya: The Indian Journal of Statistics*, vol. 26, no. 4, pp. 359–372, 1964.
- [14] A. Zeng, S. Song, K.-t. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo, N. Fazeli, F. Alet, N. C. Daffe, R. Holladay, I. Morona, P. Q. Nair, D. Green, I. Taylor, W. Liu, T. Funkhouser, and A. Rodriguez, “Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching,” *The International Journal of Robotics Research*, pp. 1–16, 2019.
- [15] D. Meike, M. Pellicciari, G. Berselli, A. Vergnano, and L. Ribickis, “Increasing the energy efficiency of multi-robot production lines in the automotive industry,” in *IEEE International Conference on Automation Science and Engineering*, pp. 700–705, 2012.
- [16] M. Yip and N. Das, “Robot Autonomy for Surgery,” in *Encyclopedia of Medical Robotics*, pp. 281–313, 2018.
- [17] B. van Beek and B. de Jager, “RRR-robot design: basic outlines, servo sizing, and control,” in *IEEE International Conference on Control Applications*, pp. 36–41, 1997.
- [18] A. Rubaai, M. J. Castro-Sitiriche, M. Garuba, and L. Burge, “Implementation of Artificial Neural Network-Based Tracking Controller for High-Performance Stepper Motor Drives,” *IEEE Transactions on Industrial Electronics*, vol. 54, no. 1, pp. 218–227, 2007.
- [19] D. A. Schreiber, D. B. Shak, A. M. Norbash, and M. C. Yip, “An Open-Source 7-Axis, Robotic Platform to Enable Dexterous Procedures within CT Scanners,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019.
- [20] G. A. Pratt and M. M. Williamson, “Series Elastic Actuators,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 399–406, 1995.
- [21] M. Verghese, F. Richter, A. Gunn, P. Weissbrod, and M. Yip, “Model-free Visual Control for Continuum Robot Manipulators via Orientation Adaptation,” in *International Symposium on Robotics Research*, pp. 1–12, 2019.
- [22] H. Su, D. C. Cardona, W. Shang, A. Camilo, G. A. Cole, D. C. Rucker, R. J. Webster, and G. S. Fischer, “A MRI-guided concentric tube continuum robot with piezoelectric actuation: A feasibility study,” in *IEEE International Conference on Robotics and Automation*, pp. 1939–1945, 2012.
- [23] C. R. A. Rocha, C. P. Tonetto, and A. Dias, “Robotics and Computer-Integrated Manufacturing A comparison between the Denavit – Hartenberg and the screw-based methods used in kinematic modeling of robot manipulators,” *Robotics and Computer Integrated Manufacturing*, vol. 27, no. 4, pp. 723–728, 2011.

- [24] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. The MIT Press, 2005.
- [25] D. Rakita, B. Mutlu, and M. Gleicher, “RelaxedIK: Real-time Synthesis of Accurate and Feasible Robot Arm Motion,” in *Robotics: Science and Systems*, 2018.
- [26] S. R. Buss, “Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares Methods,” *IEEE Journal of Robotics and Automation*, vol. 17, pp. 1–19, 2004.
- [27] T. Lozano-Perez, “Spatial Planning : A Configuration Space Approach,” *IEEE Transactions on Computers*, vol. 32, no. 2, 1983.
- [28] S. M. LaValle, *Planning Algorithms*. Cambridge, MA: Cambridge University Press, 2006.
- [29] F. Flacco and A. D. Luca, “A Depth Space Approach to Human-Robot Collision Avoidance,” in *IEEE International Conference on Robotics and Automation*, 2012.
- [30] J. J. Kuffner and S. M. Lavalle, “RRT-Connect : An Efficient Approach to Single-Query Path Planning,” in *International Conference on Robotics and Automation*, no. April, pp. 995–1001, 2000.
- [31] S. Karaman and E. Frazzoli, “Sampling-based Algorithms for Optimal Motion Planning,” *International Journal of Robotics Research*, vol. 30, no. 7, pp. 1–76, 2011.
- [32] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, “Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic,” *IEEE International Conference on Intelligent Robots and Systems*, no. Iros, pp. 2997–3004, 2014.
- [33] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, “Batch Informed Trees (BIT*): Sampling-based Optimal Planning via the Heuristically Guided Search of Implicit Random Geometric Graphs,” in *IEEE International Conference on Robotics and Automation*, pp. 3067–3074, 2015.
- [34] M. Elbanhawi and M. Simic, “Sampling-Based Robot Motion Planning: A Review,” *IEEE Access*, vol. 2, pp. 56–77, 2014.
- [35] K. Hauser, “Lazy collision checking in asymptotically-optimal motion planning,” *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2015-June, no. June, pp. 2951–2957, 2015.
- [36] C. Ericson, *Real-Time Collision Detection*. CRC Press, 2004.
- [37] N. Vahrenkamp, T. Asfour, and R. Dillmann, “Efficient Motion Planning for Humanoid Robots using Lazy Collision Checking and Enlarged Robot Models,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.

- [38] J. Bialkowski, S. Karaman, M. Otte, and E. Frazzoli, “Efficient Collision Checking in Sampling-Based Motion Planning,” in *Algorithmic Foundations of Robotics X*, pp. 365–380, Springer, 2013.
- [39] D. Meagher, “Octree Encoding: A New Technique for the Representation , Manipulation and Display of Arbitrary 3-D Objects by Computer,” tech. rep., Rensselaer Polytechnic Institute, 1980.
- [40] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.
- [41] P. Langley and H. A. Simon, “Applications of Machine Learning and Rule Induction,” *Communications of the ACM*, vol. 38, no. 11, 1995.
- [42] Y. Freund and R. E. Schapire, “Large Margin Classification Using the Perceptron Algorithm,” *Machine Learning*, vol. 37, no. 3, pp. 277–296, 1999.
- [43] C. Cortes and V. Vapnik, “Support-Vector Networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [44] K. Fukunaga and P. M. Narendra, “A Branch and Bound Algorithm for Computing k-Nearest Neighbors,” *Transactions on Computers*, pp. 750–753, 1975.
- [45] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.
- [46] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, 2009.
- [47] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [48] A. J. Smola, B. Scholkopf, and K. R. Muller, “The connection between regularization operators and support vector kernels,” *Neural Networks*, vol. 11, no. 4, pp. 637–649, 1998.
- [49] T. Hofmann, B. Schölkopf, and A. J. Smola, “Kernel methods in machine learning,” *Annals of Statistics*, vol. 36, no. 3, pp. 1171–1220, 2008.
- [50] H. Q. Minh, P. Niyogi, and Y. Yao, “Mercer’s Theorem, Feature Maps, and Smoothing,” in *International Conference on Computational Learning Theory*, pp. 154–168, 2006.
- [51] B. Scholkopf, “The Kernel Trick for Distances,” in *Advances in Neural Information Processing Systems*, pp. 301–307, 2001.
- [52] O. Chum, “Low Dimensional Explicit Feature Maps,” in *IEEE International Conference on Computer Vision (ICCV)*, pp. 4077–4085, 2015.
- [53] B. Paden, A. Mees, and M. Fisher, “Path planning using a Jacobian-based freespace generation algorithm,” in *IEEE International Conference on Robotics and Automation*, pp. 1732–1737, 1989.

- [54] B. Burns and O. Brock, “Toward Optimal Configuration Space Sampling,” in *Robotics: Science and Systems*, (Cambridge, MA), 2005.
- [55] J. Pan and D. Manocha, “Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing,” *International Journal of Robotics Research*, vol. 35, no. 12, pp. 1477–1496, 2016.
- [56] J. Huh and D. D. Lee, “Learning high-dimensional Mixture Models for fast collision detection in Rapidly-Exploring Random Trees,” *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 63–69, 2016.
- [57] Y. Han, W. Zhao, J. Pan, Z. Ye, R. Yi, and Y.-j. Liu, “A Configuration-Space Decomposition Scheme for Learning-based Collision Checking,” *arXiv preprint arXiv:1911.08581 [cs]*, 2019.
- [58] S. Rifai, Pascal Vincent, X. Muller, X. Glorot, and Y. Bengio, “Contractive Auto-Encoders: Explicit Invariance During Feature Extraction,” in *International Conference on Machine Learning*, no. 1, 2011.
- [59] T. Tran, J. Denny, and C. Ekenna, “Predicting Collision Detection with a Two-Layer Neural Network,”
- [60] J. C. Kew, B. Ichter, and T.-w. E. Lee, “Neural Collision Clearance Estimator for Fast Robot Motion Planning,” *arXiv preprint arXiv:1910.05917 [cs.RO]*, 2019.
- [61] S. J. Wright, “Coordinate descent algorithms,” *Mathematical Programming*, vol. 151, no. 1, pp. 3–34, 2015.
- [62] R. A. Horn and C. R. Johnson, *Matrix Analysis*. 1985.
- [63] G. Schohn and D. Cohn, “Less is More: Active Learning with Support Vector Machines,” in *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 839–846, 2000.
- [64] J. Pan, X. Zhang, and D. Manocha, “Efficient Penetration Depth Approximation using Active Learning,” *ACM Transactions on Graphics*, vol. 32, no. 6, 2013.
- [65] P. I. Corke, *Robotics, Vision & Control*. Springer, 2017.
- [66] I. A. Sucan and S. Chitta, “MoveIt!.”
- [67] D. Hershberger, D. Gossow, and J. Faust, “3D visualization tool for ROS,” 2018.
- [68] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, “ROS: an open-source Robot Operating System,” in *ICRA Workshop on Open Source Software*, 2009.

- [69] I. A. Şucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” *IEEE Robotics & Automation Magazine*, no. December, pp. 72–82, 2012.
- [70] B. Yuh, “The Bedside Assistant in Robotic Surgery - Keys to Success,” *Urologic Nursing*, vol. 33, no. 1, pp. 29–32, 2013.
- [71] R. M. Kuntz, S. Ahyai, K. Lehrich, and A. M. R. Fayad, “Transurethral Holmium Laser Enucleation Of The Prostate Versus Transurethral Electrocautery Resection Of The Prostate: A Randomized Prospective Trial In 200 Patients,” *The Journal of Urology*, vol. 172, no. September, pp. 1012–1016, 2004.
- [72] O. Mohareri and S. E. Salcudean, “da Vinci Auxiliary Arm as a Robotic Surgical Assistant for SemiAutonomous Ultrasound Guidance During Robot-Assisted Laparoscopic Surgery,” in *The Hamlyn Symposium on Medical Robotics*, no. July, 2014.
- [73] Z. Chen, A. Deguet, R. Taylor, S. Dimaio, G. Fischer, and P. Kazanzides, “An Open-Source Hardware and Software Platform for Telesurgical Robotics Research,” *The MIDAS Journal*, pp. 1–11, 2013.
- [74] Y. P. Tan, P. Liverneaux, and J. K. Wong, “Current Limitations of Surgical Robotics in Reconstructive Plastic Microsurgery,” *Frontiers in Surgery*, vol. 5, no. March, pp. 1–9, 2018.
- [75] I. Georgilas, G. Dagnino, P. Tarassoli, R. Atkins, and S. Dogramadzi, “Robot-Assisted Fracture Surgery: Surgical Requirements and System Design,” *Annals of Biomedical Engineering*, vol. 46, no. 10, pp. 1637–1649, 2018.
- [76] C. Taleb, E. Nectoux, and P. Liverneaux, “Limb Replantation With Two Robots: A Feasibility Study In A Pig Model,” *Microsurgery*, vol. 29, pp. 504–506, 2009.
- [77] R. D. Katz, J. A. Taylor, G. D. Rosson, P. R. Brown, and N. K. Singh, “Robotics in plastic and reconstructive surgery: Use of a telemanipulator slave robot to perform microvascular anastomoses,” *Journal of Reconstructive Microsurgery*, vol. 22, no. 1, pp. 53–57, 2006.
- [78] P. Iranmanesh, P. Morel, O. J. Wagner, I. Inan, F. Pugin, and M. E. Hagen, “Set-up and docking of the da Vinci® surgical system: prospective analysis of initial experience,” *The International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 6, no. 1, pp. 57–60, 2010.
- [79] J. D. Garisto, R. Bertolo, and J. Kaouk, “Technique for Docking and Port Placement Using a Purpose-built Robotic System (SP1098) in Human Cadaver,” *Urology*, vol. 119, no. June, pp. 91–96, 2018.
- [80] M. G. Genton, “Classes of Kernels for Machine Learning : A Statistics Perspective,” *Journal of Machine Learning Research*, vol. 2, pp. 299–312, 2001.
- [81] L. A. Belanche Munoz, “Developments in Kernel Design,” in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2013.

- [82] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” in *IEEE International Conference on Robotics and Automation*, pp. 500–505, 1985.
- [83] G. Sanchez and J.-C. Latombe, “A Single-Query Bi-Directional Probabilistic Roadmap,” in *The Tenth International Symposium on Robotics Research*, pp. 403–417, 2003.
- [84] J. A. Starek, J. V. Gomez, E. Schmerling, L. Janson, L. Moreno, and M. Pavone, “An asymptotically-optimal sampling-based algorithm for Bi-directional motion planning,” *IEEE International Conference on Intelligent Robots and Systems*, vol. 2015-Decem, pp. 2072–2078, 2015.
- [85] B. Scholkopf, A. Smola, and K.-R. Muller, “Nonlinear Component Analysis as a Kernel Eigenvalue Problem,” *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998.
- [86] R. Schiavi, F. Flacco, and A. Bicchi, “Integration of Active and Passive Compliance Control for Safe Human-Robot Coexistence,” in *IEEE International Conference on Robotics and Automation*, 2009.
- [87] M. C. Lin and D. Manocha, “Collision and Proximity Queries,” in *Handbook of Discrete and Computational Geometry*, pp. 1029–1056, CRC Press, 3rd ed., 2003.
- [88] K. J. Kyriakopoulos and G. N. Saridis, “Minimum Distance Estimation and Prediction Under Uncertainty for Online Robotic Motion Planning,” *IFAC Proceedings Volumes*, vol. 23, no. 8, pp. 93–98, 1990.
- [89] M. Safeea, N. Mendes, and P. Neto, “Minimum Distance Calculation for Safe Human Robot Interaction,” *Procedia Manufacturing*, vol. 11, no. June, pp. 99–106, 2017.
- [90] L. Fahrmeir, T. Kneib, S. Lang, and B. Marx, *Regression Models, Methods and Applications*. Springer, 2013.
- [91] D. G. Altman and M. J. Gardner, “Calculating confidence intervals for regression,” *British Medical Journal*, vol. 296, no. April, pp. 1238–1242, 1988.
- [92] B. Wilcox and M. C. Yip, “SOLAR-GP: Sparse Online Locally Adaptive Regression Using Gaussian Processes for Bayesian Robot Model Learning and Control,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2832–2839, 2020.
- [93] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, “Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization,” in *Robotics: Science and Systems*, 2013.
- [94] N. Lopes and B. Ribeiro, “GPUMLib: An efficient open-source GPU machine learning library,” *International Journal of Computer Information Systems and Industrial Management Applications*, vol. 3, pp. 355–362, 2011.

- [95] R. K. Beatson, J. B. Cherrie, and D. L. Ragozin, “Fast Evaluation of Radial Basis Functions: Methods for Four-Dimensional Polyharmonic Splines,” *SIAM Journal of Mathematical Analysis*, 2001.
- [96] S. Panati, B. Baasandorj, and K. T. Chong, “Autonomous Mobile Robot Navigation Using Harmonic Potential Field Autonomous Mobile Robot Navigation Using Harmonic Potential Field,” in *IOP Conference Series: Materials Science and Engineering*, 2015.