Trajectory Planning for Manipulators Performing Complex Tasks

by

Ariyan M Kabir

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Mechanical Engineering)

December 2019

Copyright 2020 Ariyan M Kabir

To my parents

Acknowledgements

I have been fortunate to get the opportunity to have Professor Satyandra K. Gupta as my doctoral advisor. I am forever grateful to him for his patience, support, and guidance to make this journey meaningful and exciting. The lessons I have learned working with him will continue to be strong influence for me many years into the future.

I would like to thank the members of my committee, Professor Nora Ayanian and Professor Steve R. Nutt, for their valuable insights, feedback, and suggestions for this work. Additional thanks to the National Science Foundation (NSF grant 1634431) and National Institute of Standards and Technology (NIST Cooperative Agreement 70NANB15H250) for their generous financial support of the work contained in this document under.

I would also like to thank the many great people who I have had the pleasure of working with over the past years. Thank you to all my colleagues, especially Joshua Langsfeld, Brual Shah, Shaurya Shriyam, Nithyananda Kumbla, Pradeep Rajendran, Di Zheng, Vinai Rachakonda, Rishi Malhan, Aniruddha Shembekar, Alec Kanyuck, Shantanu Thakar, Prahar Bhatt, Jason Gregory, Yeo Jung Yoon, and Professor Krishna Kaipa. Their support and feedback was crucial for successfully completing this dissertation.

I would like to thank all the wonderful faculties and staffs from University of Southern California and University of Maryland, College Park for making my journey memorable. I am grateful to all the wonderful faculties and staffs from Bangladesh University of Engineering and Technology, Notre Dame College, Dhaka, and Engineering University Higher Secondary School for nurturing me into the person I am today.

I specially thank all my friends, who are located all around the globe, for their endless encouragement and inspiration.

Finally, I could not have done this without my family. My parents, Ruma and Alamgir, who instilled in me a lifelong love of learning and the values required for higher education. My sister, Alija, who always had words of support for my work. My grandparents, uncles, aunts, and cousins, for truly making me believe I can accomplish anything that I set my mind to. My wife, Sarah, who has never been anything but a supportive and loving partner. I must specially thank her for this wonderful journey together, for taking everything in stride and with a sharp wit. My family's endless love and patience is what kept me going.

Table of Contents

List Of Tables

List Of Figures

Abstract

Traditionally industrial manipulators are programmed manually for executing repeated motions. Examples include robotic welding and painting. Recent advances in hardware capabilities, computation power, and control algorithms have physically enabled manipulators to perform highly complex tasks. Representative examples can be laundry folding, composite sheet layup, liquid pouring, mobile manipulation, surgery, etc. However, it is not feasible to manually program manipulators for such complex tasks as it will take a significant amount of time, effort, and cost. Therefore, manipulators are not being used in "high-mix, low-volume" productions or service applications. We could use manipulators in complex service and manufacturing applications, where the task description and the environment is frequently changing, if manipulators could plan their own trajectories.

Trajectory planning for manipulators performing complex tasks is a problem with different facets. It requires avoiding obstacles present in the robot's workspace, assigning appropriate tasks to the degrees of freedoms in the robotic systems, respecting the kinematic and dynamic limitations of the manipulators, and identifying the appropriate trajectory and process parameters for achieving the desired task performance. This dissertation develops algorithmic foundations to solve this problem and presents three major contributions. Firstly, this dissertation presents a unified framework to address the problem of trajectory planning by combining three traditionally diverse technologies: discrete state-space search, non-linear parametric optimization, and self-directed learning. Secondly, this dissertation presents novel contributions in each of these three technologies to overcome the limitations of existing approaches. The work presents a

context-dependent search strategy switching algorithm to navigate the discrete state-space search towards promising directions. The work proposes successive refinement strategies to find feasible solutions for non-linear parametric optimization problems with conflicting constraints. The work presents a self-directed learning method that simultaneously identifies optimal parameters and optimizes resources using probabilistic decision making. Finally, the dissertation presents novel representations to be used with the unified framework for improving computational efficiency. The developed technologies and the unified framework have been verified through practical applications, including robotic finishing. It is anticipated that the research contribution from this dissertation will enable the use of manipulators in new applications.

Chapter 1

Introduction

1.1 Motivation

Traditionally manipulators are deployed in large volume productions (e.g., automotive, electronics) where they repeat pre-programmed motions to perform a predefined sequence of actions. These routine tasks are performed in structured environments that require weeks to build. There is a growing interest to use manipulators for complex tasks that are traditionally performed by humans in manufacturing, service, maintenance, and other applications. The recent advancements in hardware capabilities and computation processing power has enabled robots to be used in such challenging tasks. Examples include pouring liquids, cleaning deformable parts, composite sheet layup, additive manufacturing, service, household applications, mobile manipulation, etc. This requires the robot to plan its own trajectory on the fly based on the task requirement.

To understand the complexity of trajectory planning, let us consider cleaning as a representative task. Humans often use two hands for cleaning a large or geometrically complex part. They use one hand for holding the part and the other for cleaning. Humans reposition and reorient the part with the holding hand to maximize the reachability on the part's surface for the cleaning hand. Ergonomic part placement and continuously changing the part pose enable them to apply the required amount of force and perform smooth cleaning passes. They generate safe motions of

the arms, without damaging the part or hurting themselves, for moving the part and executing the cleaning passes. Their whole body assists the motions of the arms to facilitate better reach and comfort. They also adjust the force, velocity, oscillation frequency, and direction of cleaning to remove hard stains efficiently. Given a new part and different kind of stain, humans can instantly adapt to complete the task efficiently.

Manually programming robots (or manipulators) to perform such complex tasks is time-consuming and challenging. It is not feasible to manually program a robot every time the environment or the task description changes between different instances of the task. Sometimes the time and cost of manual programming are significantly larger compared to the expected time and cost reduction by using robots. Moreover, the trajectory and operation parameters (e.g., velocity, force, stiffness, tool speed, etc.) need to be adjusted to ensure the desired task performance before robots can be used for such tasks. Therefore, for many service and manufacturing applications, it is time, effort, and cost-intensive to use manipulators due to the challenges in programming them. It becomes even more difficult when these tasks require using multiple manipulators or mobile-manipulators to overcome the kinematic and dynamic limitations of a single manipulator.

We need robots to be able to program themselves for using them in complex tasks. They need to be able to plan their own trajectories in real-time based on high-level task descriptions. They also need to safely learn and sufficiently estimate trajectory parameters on the fly, with minimal use of resources to satisfy the task requirements. Therefore, methods that can significantly reduce the time, cost, and effort for programming robots for new tasks (or task instances) have the potential to open up opportunities for robots to become widespread, thereby creating significant additional value for society at large.

However, automated planning of manipulator trajectories suffer from the following challenges.

- Manipulators can not add value to an application if the trajectory planing time is long. For all practical applications, high degrees of freedom (DOF) manipulators need to work

in complex and confined environments where obstacles of different sizes and geometries are present. The obstacles are defined in the workspace of the manipulator. The manipulator need to navigate through these obstacles to move the end-effector from one point in the workspace to another. It is not practical to map the obstacles into configuration space or joint space of the manipulator. This leads to a very complex trajectory generation problem in the configuration space of the manipulator. Figure 1.1 illustrates an example. It is challenging to plan optimal trajectories for manipulators working in obstacle rich environments in a computationally efficient manner.

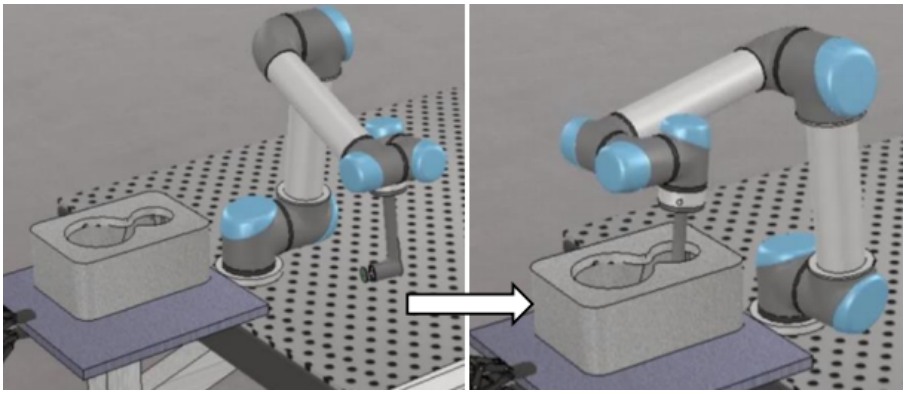


Figure 1.1: Let us assume that a 6-DOF manipulator needs to move from the initial configuration to the final configuration to polish the internal surfaces of the part. The part geometry creates a constrained workspace for the robot to guide the tool inside the part.

- Many tasks are specified through the relative motion constraints among the objects to be manipulated. Finishing [3, 4], painting [5], inspection [6], fiber placement [7, 8, 9, 10, 11, 12, 13, 14, 15, 16], sheet forming and handling [17, 18, 19], transportation [20], wire harnessing [21], etc are some of the operations where relative motion of objects can be encoded as path-constrains. Many of these complex tasks require the use of manipulators or mobile manipulators working together (agent-ensembles). Therefore, we need to generate trajectories of high-DOF robotic systems such that the motion constraints in the task description are satisfied. Figure 1.2 illustrates an example. It is computationally challenging to generate

feasible and safe trajectories of manipulators or agent-ensembles to satisfy complex motion constraints of the objects being manipulated.

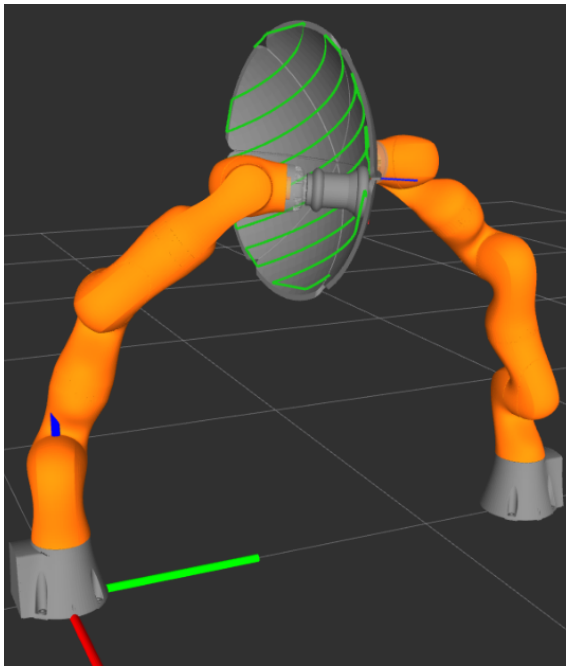


Figure 1.2: Example of a path-constrained trajectory generation problem for a 14-DOF robotic system. Joint trajectories for the manipulators need to be generated to manipulate the tool and the part for executing the buffing operation

- Manipulators have limited reachability and manipulability due to complex kinematics and dynamics. The range of orientations that the end-effector can achieve is very limited for every position it can reach in the Cartesian-workspace. The amount of force the end-effector can apply or the velocity at which the end-effector can move is not uniform across the workspace of the manipulator. Therefore, the relative pose between a workpiece and a manipulator need to be carefully selected for the successful completion of a given task. Figure 1.3 illustrates an example. Moreover, complex operations require the successful completion of multiple tasks. An ensemble of robotic agents might be required to perform these tasks. Let us consider a shop floor as an example. There can be multiple manipulators working on different processes, mobile robots transferring parts to different stations, single

or multi-arm mobile-manipulators tending machines, multiple mobile-manipulators working collaboratively, etc. Depending upon the task, agents might collaborate (e.g., two arms lifting an object together) or work independently (e.g., one arm doing the polishing task and the second arm doing the sensing task). Figure 1.4 illustrates an example. It is challenging to find a near-optimal solution for the robot/part placement problem or the task-agent assignment problem in a computationally efficient manner.

Figure 1.3: A 7-DOF manipulator is tasked to sand a bike-fender made of composite material. The figure on the left illustrates a combination of five setups to complete the sanding task. The green region is reachable by the robot in each setup to carry out the desired operation. The figure on the right illustrates the execution of the sanding trajectory for setup 1.

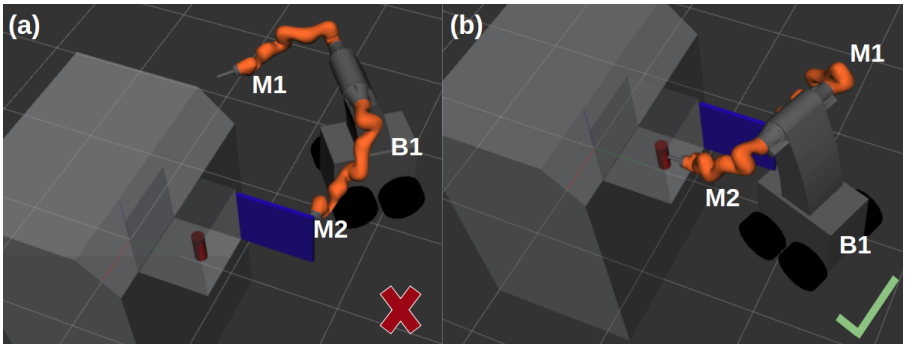


Figure 1.4: The three agents (M_1, M_2, B_1) need to cooperate to extract the part out of the machine and transfer it to the desired location. The door of the machine is *spring-loaded*. (a) An infeasible assignment of agents. There is no feasible configuration for the agents such that M_2 can hold the door while M_1 extracts the part. (b) A feasible assignment of agents. M_1 can hold the door while M_2 extracts the part.

- Manipulators are traditionally used in assembly [22, 23], machine tending [24, 25, 26, 27], and material handling [28, 29, 30] tasks. There is an increasing interest in using manipulators for process applications. In these applications, the manipulator is required to make changes to the part being processed. Robotic surface finishing tasks such as cleaning, polishing, sanding, machining, and painting are some examples of process applications. The underlying physics-based model of the process application is known in some cases. Analytical optimization methods or simulation-based approaches can be used in these cases to identify and optimize operation parameters to achieve desired task performance. However, physics-based models are not known a priori for many applications. This is often the case when new material, part, or tool is under consideration. Trajectory parameters (e.g., force, velocity, stiffness, etc.) play a significant role in the successful completion of a given task. For example, without selecting the appropriate parameters, a manipulator will not be able to satisfactorily complete a finishing task, even if the end-effector moves along the same nominal tool-path. Figure 1.5 illustrates two examples. In many cases, the physics-based models of the tasks are not available. Therefore, the trajectory parameters need to be identified through physical experiments in a way such that the overall process time is minimized for a task. It is challenging to identify the optimal trajectory or process parameters without conducting a large number of experiments, which is not time and cost-efficient.

1.2 Goal and Scope

The goal of this dissertation is to develop algorithmic foundations of trajectory planning for manipulators performing complex tasks. This dissertation presents a unified framework to combine three traditionally diverse technologies- discrete state-space search, non-linear parametric optimization, and self-directed learning. It presents novel contributions in each of these three technologies to overcome the limitations of existing approaches and improve computational efficiency. The work

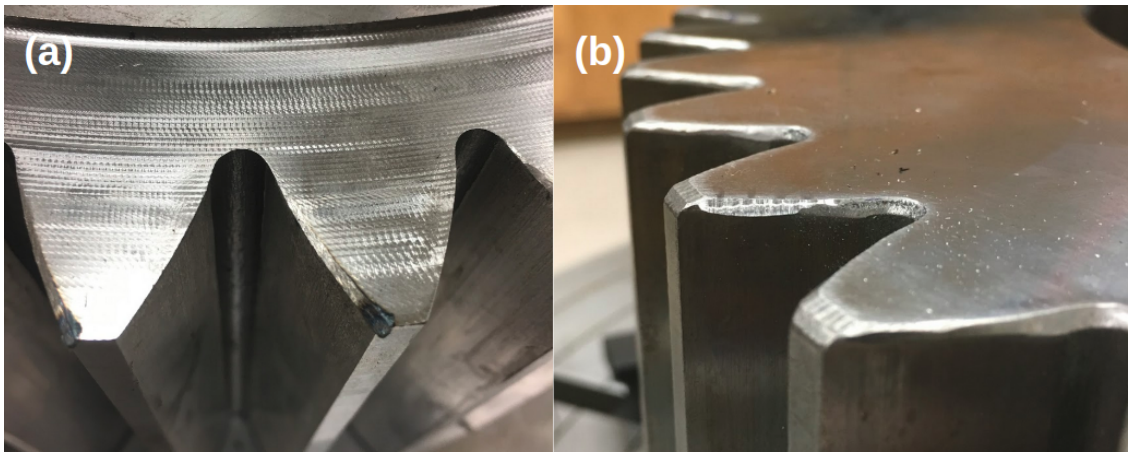


Figure 1.5: Examples of failed robotic chamfering operations due to poor trajectory parameter selection. (a) Burnt edges due to low velocity and excessive force. (b) Non-uniform material removal due to improper selection of stiffness parameters.

presents a context-dependent search strategy switching algorithm to enable automated heuristic selection for guiding the search towards promising directions. The work introduces a successive refinement strategy to identify feasible solution using non-linear optimization in high dimensional parameter space with conflicting constraints. The work presents a self-directed learning approach to identify optimal parameters by focusing on promising regions in the parameter-space without learning the complete model. Moreover, the work presents novel representations to be used with the unified framework for improving computational efficiency.

The main research issues addressed in this work are summarized below:

- *Point-to-Point Trajectory Planning for Manipulators Operating in Confined Workspaces:* A manipulator needs to move its end-effector from a start point in its workspace to a goal point for different tasks. Applications for such motion includes machine tending, assembly, bin-picking, kitting, etc. A safe and feasible trajectory needs to be generated in the manipulator's configuration space (or joint space) for such motion. Trajectories of this kind will be referred to as point-to-point trajectories in this dissertation. The work addresses the research issues of generating real-time near-optimal manipulator trajectories for point-to-point motions in confined workspaces.

- *Generation of Synchronized Configuration Space Trajectories with Workspace Path Constraints for Multi-Robot Systems:* For any given physical task, the relative motions between the objects are defined in the workspace or task-space of the manipulators. For example, a tool or workpiece may need to be moved through a specified workspace path for certain tasks. Both position and orientation for the tool and the workpiece can be specified (with acceptable tolerance) for each point on the path. Application examples can be cleaning, painting, inspection, assembly, transporting large parts using multiple robots, shaping deformable objects, etc. Configuration space trajectories for robots need to be generated to perform such tasks. The work considers the relative motion constraints among the objects to be manipulated as constraints for the trajectory planner and addresses the research issues for generating path-constrained trajectories for manipulators.

- *Bringing Setup Planning Considerations in Trajectory Planning:* It may not be feasible to plan manipulator trajectories to complete a task when the relative pose between the manipulator and the workpiece is fixed. Depending on the workpiece geometry and reachability of the manipulator, the workpiece needs to be presented to the manipulator in a sequence of poses such that the end-effector can reach different regions of the workpiece and complete the task. For example, to clean a large or geometrically complex part, the part needs to be repositioned and reoriented so that the manipulator can clean the entire surface using a tool. The dual problem will be having the workpiece at a fixed location and moving the manipulator base through a sequence of locations to complete the task. Inspecting a large helicopter wing using a mobile-manipulator equipped with imaging sensor is an example of such application. The work addresses the research issues in setup planning algorithms to determine the optimal sequence of relative poses between the robot and workpiece such that the given task is completed in a time-optimal manner.

- *Bringing Task-Agent Assignment Consideration in Trajectory Planning:* Multiple manipulators or multi-arm mobile-manipulator systems can be viewed as an ensemble of robotic agents. Depending upon the task, agents might collaborate or work independently. The capabilities of the robots are different in a heterogeneous ensemble of robotic agents. Moreover, the capabilities of homogeneous agents change with the constraints posed by the work environment. Sometimes, an agent needs to assist one or more other agents for a given task. Invoking motion planners to generate motion for an infeasible task-agent assignment incurs computation cost. Therefore, task-agent assignment considerations need to be brought during trajectory planning. The work addresses the research issues to identify the right task-agent assignment to enable successful completion of the given tasks in a complex task network.

- *Identification of Trajectory and Process Parameters using Physical Experiments:* Trajectory planning algorithms enable robots to generate required motions on their own for complex tasks. However, certain tasks require adjusting process and trajectory parameters to ensure desired performance. Let us consider robotic finishing as an example. The finishing performance depends on the tool rotation speed, tool-tip velocity, force applied to the surface, stiffness of the manipulator's joints, etc. The process models are often unknown or partially known for these complex tasks. The input-output relationship can be revealed by conducting physical experiments. However, conducting a large number of physical experiments and building an accurate model for performance constraints can be expensive, especially for cases where task description or environment is frequently changing. Robots need to quickly and safely identify the optimal trajectory and process parameters and adjust their trajectories on the fly to ensure desired performance. The work addresses the research issues to identify the trajectory and process parameters for the successful completion of complex tasks in a time-efficient manner.

1.3 Overview

This section briefly summarizes the chapters of this dissertation.

Chapter 2 presents a literature review on point-to-point and path-constrained trajectory planning, reachability considerations in trajectory planning, task and motion planning, and trajectory parameter identification for manipulators.

Chapter 3 presents a branch and bound heuristic search algorithm for point-to-point trajectory planning. The algorithm is capable of switching search strategies depending on the context, to eliminate branches with low promise and reducing the depth of the search tree, enabling faster computation. The main focuses of this work are computational foundations for efficiently searching in different regions of the search space, which includes- context identification based on neighborhood and history analysis, adaptive selection of heuristics for different regions in the search space, adaptive switching between work-space and configuration-space motion primitives, adaptive scaling of the motion primitives, pruning nodes in bottleneck regions, and back-tracking and repairing trajectories.

Chapter 4 presents an optimization-based approach to generate path-constrained synchronous motion of multiple robots (e.g., manipulators, mobile manipulators). It represents the motion constraints as path-constraints and poses the problem of path-constrained synchronous trajectory generation as a non-linear optimization problem. The method formulates the problem as a discrete parameter optimization problem. It uses successive refinement techniques to solve the optimization problem in high-dimensional space with conflicting constraints. The method adaptively selects the parametric representation of the configuration variables for a given scenario. It also generates an approximate solution as the starting point for the successive refinement stages to reduce the computation time.

Chapter 5 presents a setup planning algorithm using sampling, search, and optimization methods to identify the optimal sequence of setups in the context of process applications where the

manipulator will execute both point-to-point and path-constrained trajectories. The research issues addressed include how a discrete set of candidate setups can be generated from the six-dimensional continuous space of part pose using uniform vs random sampling, how the sampling schemes can be sufficient to identify the candidate setups that will lead to minimum setup solution, how to improve the reachability for the candidate setups, and how branch guiding and branch pruning heuristics influence the computation time to generate the optimal sequence of setups using branch and bound depth-first search.

Chapter 6 introduces a novel architecture for integrating trajectory planning with task-agent assignment in a computationally efficient manner. The method utilizes a spatial constraint checking layer between task-agent assignment and trajectory planning layers to reduce the computation time. The spatial constraint checking layer quickly identifies infeasible task-agent assignments, thus reducing calls to the motion planners. It enables auto-selection of point-to-point or path-constrained trajectory planners for the agents based on task-description and spatial constraint checking. Moreover, the method uses a multi-tree representation to enable efficient caching and avoiding repeated computations.

Chapter 7 formulates the parameter identification problem as a constrained optimization problem with the goal of identifying the optimal parameters by reducing the overall process time. The black-box or unknown constraints are task performance constraints, such as surface roughness. The known objective function is task objective, such as minimizing time and force. The presented method for solving this problem uses the data from exploratory experiments to build surrogate models for black-box task constraints. By estimating the probability of success, parameters are selected iteratively during refining experiments to optimize task-objective and satisfy task-constraints. The focus of this approach is not to learn the complete model of the task constraints. Rather the goal is to quickly and accurately learn the regions of interest in the process model to identify optimal parameters.

Chapter 8 presents a case study on robotic finishing. It combines the developed technologies in this dissertation with an intuitive user interface to develop an advanced robotic finishing system. The human operator can import the CAD model of a part, select desired operation and regions of interest, populate available tools, and specify other process preferences through the user interface. The system then generates the instructions for the robot assistants using the developed algorithms presented in this work. The robot trajectory is executed under a feedback loop, and the human operator is warned when a contingency arises. The user interface allows the human operator to monitor the task or provide help to the robot assistants as needed. If required, the robot assistants make a call to the human operator to provide help in the workcell physically.

Chapter 9 presents the intellectual contributions of this dissertation. It discusses the merits of the unified framework and novel representations that combine discrete state-space search, non-linear parametric optimization, and self-directed learning technologies. It summarizes the technological advancement that enabled trajectory planning for manipulators performing complex tasks. Moreover, this chapter talks about how the developed technologies can help the industry and summarizes potential future research directions.

Chapter 2

Literature Review

2.1 Overview

This chapter reviews the existing work regarding manipulator trajectory planning for point-to-point motions, path-constrained trajectory planning for a single manipulator, trajectory generation for high-DOF robotic systems, reachability analysis of manipulators, workspace estimation for manipulators, setup planning, task assignment and motion planning, and trajectory and process parameter optimization. Moreover, this chapter summarizes the limitations of the existing approaches in the context of trajectory planning for manipulators performing complex tasks.

2.2 Trajectory Planning for Point-To-Point Motions

Point-to-point trajectory planning for manipulators has been studied for decades. Researchers have focused on solving the problem using potential field-based [31], sampling-based [32, 33, 34, 35, 36, 37, 24, 25], continuous optimization-based [38, 39], and heuristic search-based [40, 41, 42, 43, 44, 45, 46, 47, 48, 49] methods.

Potential field-based methods [31] rely on describing the obstacles in the configuration space of the robot. It considers the obstacles as sources of negative potential and the goal state as a source of positive potential. If the potential field is designed accurately, then the goal configuration can

be achieved by flowing through the field from any start state in the configuration space. However, describing the obstacles in the configuration space of the robot and explicit construction of the potential field is not feasible for robots with high degrees of freedom (DOF), such as 6 DOF or more.

Researchers have extensively explored sampling-based algorithms for trajectory planning of high DOF robots. PRM [32] and RRT [34, 33] are two major sampling-based algorithms used for motion planning. They sample the configuration space for obstacle-free nodes and find collision-free edges between a pair of nodes. Karaman et al. proposed RRT* [35] to improve the path and start approaching asymptotic optimality. Several different approaches have been proposed to improve the computation time of RRT* [50, 36]. However, the solution quality is not predictable due to the stochastic nature of the sampling-based algorithms. The methods also do not perform well when a path needs to be constructed through narrow corridors of the configuration space.

Researchers have also studied optimization-based methods for trajectory planning. Two major methods developed for manipulator trajectory planning are CHOMP [39] and STOMP [38]. CHOMP iteratively improves an initial seed trajectory by using functional gradient techniques. STOMP generates multiple candidate trajectories by sampling around an initial seed, and iteratively improves them through stochastic optimization using the estimated gradient.

Graph search-based algorithms can guarantee resolution optimality for a solution. Researchers have developed different variations of A* for faster convergence of heuristic-based search in manipulator trajectory planning applications. Cohen et al. [42] used anytime A* algorithm with informative heuristics and motion primitives to plan manipulator trajectory. The authors considered motion primitives that lead to small actions. The authors extended their work [44] and presented an approach that uses adaptive motion primitives for manipulator trajectory planning. This method dynamically constructs the graph using static, inverse kinematics (IK)-based, and orientation solver-based motion primitives. The authors considered a weighted sum of metrics in position and orientation space as the heuristic for the search. The position metric was calculated

using Dijkstra search in xyz space, which essentially gives the least cost path from any xyz point in the workspace to the goal xyz. The metric in orientation was the angle of rotation about a fixed axis between the orientation of a state and the goal orientation.

Gochev et al. [43] proposed an algorithm to speed up the high dimensional trajectory planning by adaptively searching in lower-dimensional state spaces. The authors extended their work to develop a motion planning algorithm for manipulators with independent wrist joints [45]. They divided the high dimensional planning problem into two lower dimensional problems. The algorithm focuses on lower-dimensional planning to bring the wrist closer to the goal region first. Then the algorithm uses IK-based tracking to reach the goal pose using high dimensional planning.

The heuristics are often problem-specific and carefully crafted in search-based algorithms. Researchers have focused on using multiple heuristics simultaneously to reduce the effort on heuristic design and search convergence time. Aine et al. developed Multi-Heuristic A* (MHA*) [49] algorithm that combines the advantage of each of the heuristics. It iterates through multiple searches in a structured way using separate heuristic and maintains a separate priority queue for each search. Islam et al. [46] developed a method to generate heuristics for MHA* search dynamically. It monitors the progress of all the baseline heuristics and dynamically generates a new heuristic when the baseline heuristics are stuck in local minima. Islam et. al. [48] developed a bounded, suboptimal, bidirectional search (A*-connect). This heuristic search algorithm behaves like the RRT-connect as it simultaneously searches from both start and goal nodes. The algorithm uses an approximate front-to-front heuristic to lead the forward and backward searches towards each other. The authors used improved MHA* [51] with one guiding heuristic towards the opposite root and one towards the opposite frontier. They used a *"connect"* heuristic, which guides towards the most promising last expanded state of the opposite search, instead of scanning through the complete frontier.

2.3 Path-Constrained Trajectory Generation for Manipulators

Path-constrained trajectories require manipulators to move their end-effectors through a set of waypoints in a continuous motion. This problem has been mostly studied for single manipulator systems. This section summarizes the existing methods for path-constrained trajectory generation for a single manipulator.

Researchers have developed methods [52, 53] to optimize predefined trajectories or generate trajectories from predefined configuration space paths by minimizing time, jerk, and effort under joint position, velocity, and torque constraints. The solution for each joint angle for the complete trajectory is found either as a functional using optimal control [54] or as a parametric curve using discrete parameter optimization [55, 56]. Pfeiffer et al. [55] developed a dynamic programming-based procedure to solve the problem in minimum time by representing the curve in the robot's workspace using arc-length parameterization. Constantinescu et al. [56] solved the parameter optimization problem using flexible tolerance method. Piazzi et al. [57] represented the motion of each joint using cubic spline and used a minimax algorithm and interval analysis to solve minimum jerk trajectory generation problem. Chettibi et al. [58] solved the problem with cubic spline representation using sequential quadratic programming (SQP). Gasparetto et al. [59] combined jerk and time in their objective function and used B-splines [60] for approximating the trajectories. However, these approaches can only generate trajectories from a given initial configuration space path.

To generate path-constrained trajectories, we can generate configuration paths from given workspace paths and then convert it to configuration space trajectories. Kieffer [61] addressed the problem by generating path and trajectory in the robot's joint space from a given end-effector's path in the robot's workspace. The work solves the pathfinding problem in joint-space by representing each joint motion as a parametric curve and then converts it into the trajectory.

Ordinary singularities of manipulators were defined as the dead points on the curve where the end-effector needs to pause, but the manipulator joints can keep on moving. The algorithm is based on the predictor-corrector method and can generate trajectories to pass through ordinary singularities by finding alternate joint configurations at the point of singularity.

Martin et al. [62, 63] considered a parametric representation of each joint motion using B-Splines and formulated the minimum-effort trajectory generation problem as an SQP. The objective function was to minimize torque over the complete trajectory. The work presented a recursive algorithm for computing the objective function and the gradients of objective and constraint functions. It used lie algebra and group theory to convert the optimization problem into an SQP with optimization variable being the control points of the B-Spline curves. Joint limits, joint velocity, and end-effector pose were presented as constraints over the duration of motion. The quality of the trajectory and success of these methods depend on a good initial seed and an appropriate parametric representation. Generating an initial solution or seed and identifying the appropriate parametric representation of trajectories for a given problem can be very challenging.

The path of the end-effector is often given in curve tracing applications. Conkur [64] presented an approach for a different kind of path following problem for hyper-redundant manipulators. For a given path, represented using B-Spline curve, the problem was to guide the links of the robot along the path such that eventually the robot takes the shape of the specified path or curve. The work presented a numerical approach to keep the links of the manipulator tangent to the curve and within a tight tolerance around the curve as they maneuvered along the path. However, this approach did not consider time-varying path-constraints.

Search-based algorithms can be used for path constrained trajectory generation. We can find the joint configuration for manipulators at the waypoints on the workspace path by solving inverse kinematics (IK). There can be multiple valid joint configurations for each waypoint or end-effector's pose. In that case, a graph is constructed considering all the IK solutions for each waypoint. A graph search algorithm can then be used to find the configuration space path as a

sequence of joint configurations to move the end-effector through the waypoints by minimizing time or effort. Descartes [65] is an open-source library developed by ROS-Industrial Consortium that uses graph search on inverse kinematics (IK) solutions at sampled points on the workspace path for path constrained trajectory generation. Search-based approaches can be computationally expensive as branching factor in the graph increases exponentially with the number of joints of the manipulator, and the graph depth increases linearly with the number of waypoints used to approximate the workspace path.

Researchers have explored sampling-based approaches and genetic algorithms to generate constrained trajectories for manipulators. Cefalo et al. [66] presented a sampling-based planner that produces cyclic, collision-free paths in configuration space and guarantees continuous satisfaction of the task constraints. Cyclic motions require the robot's end-effector to continue tracing the same curve in cycles. The proposed algorithm relied on bidirectional search and loop closure in the task-constrained configuration space. The fitness function was designed as a combination of pose error and effort. Tarokh et al. [67] also approached the problem using a genetic algorithm. The population diversity and fitness were used to adjust the probabilities of the operators for the next generation.

In [68], a NURBS (Non-uniform rational B-spline) [69] representation of the joints are used to achieve various optimization objectives like time, smoothness, and energy optimality. In [70], a fifth-order B-Spline is used instead. In both approaches, a genetic algorithm is employed to minimize cost functions with joint limits. Similarly, in [71], the trajectory is discretized, and at each step, a search is performed for a new position of the end effector in the workspace to reach the final position. Because of the redundancy, this position can be achieved by an infinite number of configurations in the joint space. Thus, this property is used to find the best configuration that allows for avoiding obstacles and singularities of the robot. The proposed method is based on a bi-level optimization formulation of the problem and bi-genetic algorithm to solve it. In order to avoid obstacles, the constraints of the problem are managed dynamically. It can be tough to

identify a feasible region where all the constraints are satisfied using sampling in high dimensional parameter space. The genetic algorithm-based approaches can be computationally expensive and may fail to find a feasible solution for complex motion synchronization problems with a high-DOF system and conflicting constraints.

2.4 Trajectory Generation for High-DOF Robotic Systems

Researchers have also studied trajectory generation problems for high-DOF robotic systems such as multiple manipulators, mobile manipulators, and humanoid robots. Some of the methods developed for high-DOF robotic systems consider path-constrained motion generation. Trajectory generation for high-DOF systems has been studied for bi-manual manipulation [72], mobile manipulation [73, 74, 75, 26, 27, 76], and manipulation using humanoids [77, 78, 79]. Convex optimization [80, 81] and Quadratic Programming (QP) [82, 83, 84] have been studied to generate trajectory for high-DOF systems in dynamic environments. Joint velocity control-based manipulator trajectory generation approaches [85, 86, 87] use Jacobian approximations for generating joint configurations to minimize the time to execute a path-constrained trajectory. Although the higher-order Jacobian approximation is close to the actual value resulting in a reduced error in the end-effector trajectory, it is necessary to give a valid initial joint configuration. For a redundant manipulator, the quality of the solution is highly dependent on this initial joint configuration.

Non-linear programming [88, 85], Constrained Quadratic Programming (CQP) [89] and Quadratic Programming (QP) [86] have been used to generate time-optimal trajectory by minimizing pose error (by estimating pose using Jacobian). Joint limit and velocity have been considered as constraints. Collision, path smoothness, and manipulability have been explored as part of the objective function. Stilman [73, 74] proposed an approach for high-DOF manipulation generation by combining sampling-based and Jacobian control-based methods. The method can generate

a motion plan for task-constrained manipulation by defining and incorporating the constraint matrix in the configuration space and workspace velocity relation based on Jacobian.

Berenson et al. [90, 91] presented sampling-based planners to generate robot trajectories to satisfy the constrained motion of objects. The work studied point-to-point motion planners that can satisfy task-specific constraints on the object's motion. Some of these constraints can be represented as path-constraint on the object (e.g., opening door, sliding a part). Dalibard et al. [77] proposed an approach where task constrained Jacobian-based local motions were generating to connect nodes in the RRT-Connect trees for generating whole-body motions in humanoids. Shankar et al. [92] presented QP-based approach to solving local motion planning for whole-body manipulation of a high-DOF robotic system (humanoid or mobile manipulator). Their formulation can take collision avoidance and other geometric constraints as constraints to the QP or as part of the objective function. Giftthaler et al. [83] approached the trajectory planning (applications include curve tracing and mobile manipulation) problem using sequential linear-quadratic optimal control. Escande et al. [84] presented an approach for trajectory planning of humanoids using hierarchical quadratic programming. The core idea of their approach is to solve the QP for one constraint at a time. The method finds solutions for latter constraints in the null space of the prior constraints. It is often challenging to represent many constraints (e.g., mesh-to-mesh collision) in a quadratic format. Moreover, jacobian-control based approaches may not converge without proper initialization. They may not be well suited for complex synchronization problems of high-DOF multi-robot systems.

Dynamic environments require robots to make local adjustments to their global motion plans. Lehner et al. [80] presented an incremental motion generation method for mobile manipulators in unknown and dynamic environments. Alonso-Mora et al. [81] formulated a convex optimization problem for the task of multi-robot deformable object manipulation. The method used a global motion planner for high-level trajectory generation for the robots and a receding horizon local motion planner that ensures collision avoidance and shape maintenance for the deformable object.

Salehian et al. [72] proposed a QP-based IK-solver for coordinated multi-arm motion generation in a dynamic environment. The method used SVM for data-driven self-collision prediction in the optimization routine. Buml et al. [82] demonstrated how SQP-based solvers could be used for dynamic manipulations tasks, such as catching flying objects, using humanoids. The object tracker was operating at 25Hz, and the optimizer was producing solutions at 60ms intervals. These approaches are designed for target tracking applications where motion is generated for a small time window at a time and not suitable for path-constrained motion generation applications.

Dietrich et al. [78] integrated impedance controller with whole-body motion generation to enable safe and compliant manipulation. Leidner et al. [79] integrated optimal base-placement, whole-body compliant motion generation, and object-centered hybrid reasoning to plan task-constrained whole-body motion for humanoids. The core contribution of this work was the integration of high-level symbolic and geometric reasoning with low-level controllers. The paper demonstrated the performance of the method on whole-body manipulation for surface cleaning tasks. Many manipulation tasks often require high DOF robots to satisfy multiple constraints. Their method solves the inverse kinematics solutions sequentially for the given path-constraint and then execute the trajectory bypassing the IK-solutions through a controller. Generating trajectory by solving IK solutions sequentially is not suitable for tasks where complete path-constraint is given. This is because the robots may end-up in singular configurations or other infeasible configurations while sequentially tracing the path if the complete path-constraint is not considered during motion generation.

2.5 Reachability Analysis, Workspace Estimation, and Setup Planning

We can use inverse kinematics(IK) to determine if a point in the workspace (SE(3)) of the manipulator is reachable by the end-effector. Inverse kinematics is the mathematical method to derive

configuration space or joint space states from a given workspace state of a robot. Meaning, given the pose (position and orientation) of the end-effector, inverse kinematics allows us to generate the joint angles for a manipulator that can achieve the end-effector pose. Analytical solutions [93, 94] for inverse kinematics (IK) can be derived for non-redundant (6-DOF or less) manipulators. Inverse kinematics (IK) can not be solved analytically for redundant manipulators. Numerical methods exist for solving inverse kinematics for redundant manipulators using Jacobian-based methods. Buss [95] has summarized different methods for computing inverse kinematics using manipulator Jacobian. There are few popular libraries [96, 97] for numerically solving inverse kinematics. TracIK [97] is a library for generating fast IK solutions using Jacobian approximation and numerical optimization using Sequential Quadratic Programming (SQP).

For most manipulation tasks, end-effector of a manipulator needs to reach a point in space both at the desired position and orientation. Solving inverse kinematics is computationally expensive, especially for redundant manipulators. Motion planning for manipulators can become faster if we could explicitly construct the reachable workspace of robots and use them through look up tables. Explicit construction of the complete position and orientation workspace of a manipulator is not memory efficient and not possible on standard computers. Researchers have worked on constructing approximate workspace of manipulators [98, 99].

Zacharias et al. [98] developed a method to approximate the workspace by representing the positional space as voxel-grid. They considered a sphere in each voxel and represented orientation as surface normals on the sphere. They sampled a certain number of surface normals and solved inverse kinematics of the robot to score the reachability of each voxel. They stored this representation and inverse kinematics solution as a capability map. For certain manipulation tasks, they used this map to position the redundant robot's base such that the desired end-effector pose for a certain part is enclosed within the map. Sprecher et al. [99] developed similar voxel representation for robot workspace and used it for estimating the minimum time required for a robot to reach a

point in its workspace. They produced the reachability-grid by sweeping each joint of the robot at certain discrete steps.

Workspace analysis can also be used for collision detection. Taubig et al. [100] developed a real-time self-collision detection algorithm for high DOF robotic systems using swept volume computation. Their method considers the joint motion within a bound to generate a convex hull of the swept volume.

Setup planning refers to the problem of generating a sequence of workpiece poses to optimize task-objective. Approach taken by the machining community to address the setup planning problem includes- heuristic search algorithms [101], constraint satisfaction [102], neural network [103], genetic algorithm[104], simulated annealing[104], fuzzy set theory [105], and particle swarm optimization[106]. The task constraints for setup planning included accurate part registration, restraint by fixture, tolerance on part deformation, quality of task performance, collision avoidance, etc. Most of these approaches considered generating a solution with a minimum number of setups. However, a minimum number of setup does not guarantee time optimality for task completion. The trajectory execution time needs to be incorporated in the objective function to generate a true time-optimal solution for task completion.

2.6 Task Assignment and Motion Planning

This work focuses on task-agent assignment and motion planning for an ensemble of manipulators or multi-arm mobile manipulators. Significant work has been done in the combined task and motion planning of such robotic systems. In [107], an interface between the task planner and the motion planner is developed, which generates errors giving feedback to the task planner in terms of logical predicates when motion plans fail. Off-the-shelf task and motion planners are used along with geometric information, making them efficient. Many specialized search-based approaches for solving similar task and motion planning problems have been implemented [108, 109, 110,

111]. These approaches incorporate robot manipulation needs and object rearrangements. These planners take into account complex sequential tasks. Geometric backtracking like in [112] is an important feature necessary in combined task and motion planning. This enables the robot to backtrack into the task planning domain if motions are not feasible for a particular task sequence. In [113, 114], the inefficiency of backtracking is highlighted, and an approach using culprit detection mechanisms is introduced to search in the symbolic task space using information extracted from the geometric space. Various heuristics have been suggested for combined task and motion planning methods [115, 116, 117]. Heuristics guide the search towards feasible motions, thus reducing the number of motion plan attempts.

Task and agent assignment and scheduling for robotic systems have been studied in detail in [118, 119, 120]. Multi-robot task assignment has been studied from swarm robotics perspective in [121, 122, 123]. Here, each robot is an independent agent, and although their motions are constrained due to tasks, they are free to move independently. However, the motions of the mobile base and each of the arm are coupled in systems like bi-manual mobile manipulators.

Task and manipulator motion planning has been studied in [124] in which the geometric conditions of manipulators are integrated with a symbolic description. This enables performing manipulation tasks with several robots and objects. Combined task and motion planning is studied for a bi-manual humanoid setup in [125] focusing on perception and plan execution. Task-oriented motion planning for multi-arm robotic systems is implemented in [126]. In this, a workpiece oriented formulation of cooperative tasks is proposed where a user gives instructions for the motion of the workpiece. The motion of the single stationary arms is generated using kinematic transformations. Mobile manipulator task and motion planning have been implemented in [127]. Here, hierarchical planning is presented where kinematic solutions are determined for task-level problems to determine optimal solutions for abstracted problems. Here, mobile base positioning for performing multiple tasks while optimizing for time is considered. Mobile base positioning is

especially important in bi-manual setups, where the two arms can perform distinct tasks if the mobile base is located at the appropriate position [128].

2.7 Trajectory and Process Parameter Optimization

The problem of trajectory and process parameter optimization has been addressed for many manufacturing processing applications like polishing [129], grinding [130], milling [131], cutting [132], spray coating [133], welding [134, 135], cleaning [136, 137, 138], and assembly [139, 140, 141]. In all these processes, there is significant interaction between the tooltip and the work-piece. Usually, a large number of process parameters have varying degrees of influence on task performance. Model-driven optimization methods can perform well in the simulation when the physical model of the system is available. However, a common challenge faced in all these problems is the lack of physics-based analytical models to derive functional relationships between process parameters and input-output behavior [132]. Only the input-output behavior of the system can be known through physical experiments. Therefore, most researchers in the field resorted to data-driven approaches for parameter optimization in different process applications. These data-driven approaches can be broadly classified into (1) Design of experiments (DOE) approaches [142] (e.g., Taguchi method, factorial design, and response surface design methodology), (2) Surrogate model for task performance-based approaches, (3) Meta-heuristic search approaches (e.g., genetic algorithms, simulated annealing, Tabu search), and (4) constrained optimization-based methods (e.g., COBYLA, NOMAD).

The Design Of Experiments (DOE or experimental design) is a method where a task is designed to identify the influence of certain variables on the outcome of an event. We can tune them to achieve the desired outcome by understanding the underlying influence of different parameters. This approach relies on conducting experiments by combinatorially selecting the sets of different parameter values. This method is applicable in scenarios where the outcome of the event is

measurable from the experimental data. On the other hand, surrogate model-based approaches are useful when the outcome of an event is not directly measurable by conducting experiments or it is not feasible to conduct required experiments. In this approach, a surrogate model, that can approximate the outcome of the real event, is used for experimentation and decision making. The surrogate model needs to be much cheaper to evaluate compared to the real event. It also needs to follow the input-output behavior of the actual model as closely as possible. Meta-heuristic approaches are heuristics designed to guide the search process in an optimization problem. These methods can find a sufficiently good solution without having sufficient knowledge about the entire search space. Meta-heuristics algorithms focus on efficiently exploring the search space to find a near-optimal solution without heavy computational complexity. In contrast with DOE and surrogate model-based approaches, meta-heuristic algorithms are not problem-specific.

Design of experiments (DOE) is a statistical method to determine which parameters are essential in a process and the values of these parameters that give rise to optimal performance [142]. It involves fractional factorial experiments to identify influential parameters, full factorial experiments to find the optimal parameter set, and evaluation of the distribution of the objective metrics by running a number of experiments using the optimized parameter set.

Many researchers applied DOE approaches to grinding and polishing tasks [143, 129, 144, 145]. The objective in these tasks was to minimize surface roughness or maximize material removal rate. Different process parameter sets were considered: (1) Abrasive size, contact force, linear belt velocity, and feed rate, (2) Depth of cut, wheel speed, and work speed, (3) Grinding time, rotating velocity, the belt material, and workpiece material. Other machining processes where DOE was applied include cryogenic machining of stainless steel [146], machining of KDP crystals [147], and electrochemical polishing [148, 149, 150].

DOE was applied in different works [140, 151, 152] for robotic assembly of a transmission torque converter. The goal was to optimize assembly cycle time and first-time-through-rate. Process parameters consisted of a starting point, assembly stage, insertion distance, timeout

limit, max number of trials, searching force, rotation speed, rotation angle, force amplitude, and force period and so on. These process-related parameters were converted into robot force control parameters and corresponding robot motions to perform a particular assembly task.

Elangovan et al. [153] applied Taguchi's method, which offers a distribution-free and orthogonal array design [154], to ultrasonic welding. The goal was to maximize weld strength as a function of parameters like welding pressure, weld time, and amplitude of the vibration. DOE was also applied to spot welding [155] and laser sintering of metallic powder [156].

If the grid resolution becomes finer in the parameter space or the number of parameters increases, then DOE-based methods become intractable. This is the limitation of complete DOE-based approach.

Surrogate model-based approaches have been used by Cheng and Chen [141] to optimize parameters in force control based robotic assembly. Their online optimization algorithm (GPRBOA) is based on surrogate models using a Gaussian Process (GP) [157] and Bayesian Optimization Algorithm (BOA) [158]. Their approach was presented to be more effective compared to DOE-based approaches. A tight peg-in-hole insertion problem ($40\mu m$ clearance) was considered as the assembly process. Insertion force, search force, and search speed were the trajectory parameters under consideration. The GPRBOA algorithm was later improved by Wu et al. [159] using Orthogonal Exploration.

Another approach to surrogate models are metamodels [160] that approximate computationally expensive simulation response functions. The surrogate models presented in this paper are similar to local metamodels that are used within an iterative optimization strategy and updated as the optimization progresses. The metamodels are obtained by conducting experiments at different parameter settings and fitting parametric probability models to observed values of the system response.

Genetic Algorithms (GA) have been used in multi-constraint optimization of grinding Ni-based alloys [130] and force-based robotic assembly [161]. Non-traditional optimization algorithms like

artificial bee colony (ABC), particle swarm optimization (PSO), and simulated annealing (SA) have been used for the optimization of multi-pass milling operations [131]. Teaching-learning based optimization [162, 163] has been used for machining processes like abrasive water jet machining process, grinding, milling, and turning.

Many applications require optimization of a task objective with constraints on the task performance. The physics-based models are not available for task performances. The problems can be framed as an optimization problem under black-box constraints. Two established optimization methods that focus on minimizing function evaluation are briefly described below.

The first method is known as NOMAD [1]. This method is based on the Mesh Adaptive Direct Search (MADS) algorithm, which was developed by Abramson, Audet, and Dennis [164, 165]. The objective of NOMAD is to find the best feasible solution by conducting a small number of experiments. The authors used Kriging-based surrogate models for unknown constraint functions in one version of NOMAD.

The second method utilizes a stochastic Radial Basis Function (RBF) based algorithm. This was developed by Regis [2] for large scale optimization problems. His algorithm considers both objective and constraint functions to be black-box. His method was tested on a number of synthetic test problems and was shown to outperform several other algorithms including a pattern search algorithm, a genetic algorithm, a derivative-free trust region-based algorithm (COBYLA), a sequential quadratic programming algorithm, and NOMAD.

2.8 Summary

The existing technologies have limitations to solve the problem of trajectory planning for manipulators performing complex tasks. There is a significant limitation in the fast computation of near-optimal manipulator trajectories in environments with challenging obstacle distribution. The existing methods have computational limitations in generating safe and feasible trajectories

for high-DOF robotic systems (e.g., multiple manipulators, mobile-manipulators) for performing tasks that require complex synchronization. Complex tasks require fast computation of setup plans and task-agent assignments, which is not possible using the current approaches. Moreover, traditional approaches do not consider simultaneous trajectory and process parameter identification and resource optimization for high-mix, low-volume operations.

Chapter 3

Point-to-Point Trajectory Planning for Manipulators Operating in Confined Workspaces

3.1 Introduction

This chapter[1] presents an approach to solve the point-to-point trajectory planning problem for manipulators. For this kind of trajectories, the robot needs to avoid all unwanted contacts and collisions. However, the workspace of the manipulator can be highly constrained with geometrically-complex obstacles, which in turn makes the trajectory planning problem very challenging. Fig. 3.1 illustrates a representative example. Let us assume that a six degrees of freedom (DOF) manipulator needs to move from the initial configuration (Fig. 3.1(d)) to the final configuration (Fig. 3.1(e)) to polish the internal surfaces of the part (Fig. 3.1(a)). The part geometry creates a very constrained workspace for the robot to guide the tool inside. Five possible entry regions for the tool are marked in the figure. Due to the tool and part dimensional constraints, only region 1 and 5 can be used as valid insertion regions. Regions 2, 3, and 4 are too narrow to be valid insertion regions. Moreover, no internal maneuver of the tool is possible through region 4. If the robot inserts the tool through region 5, then it cannot reach region 3 by moving the tool internally through region 4. The robot can take the tool to region 3 only by inserting it through

[1]The work in this chapter is derived from the work published in [166]

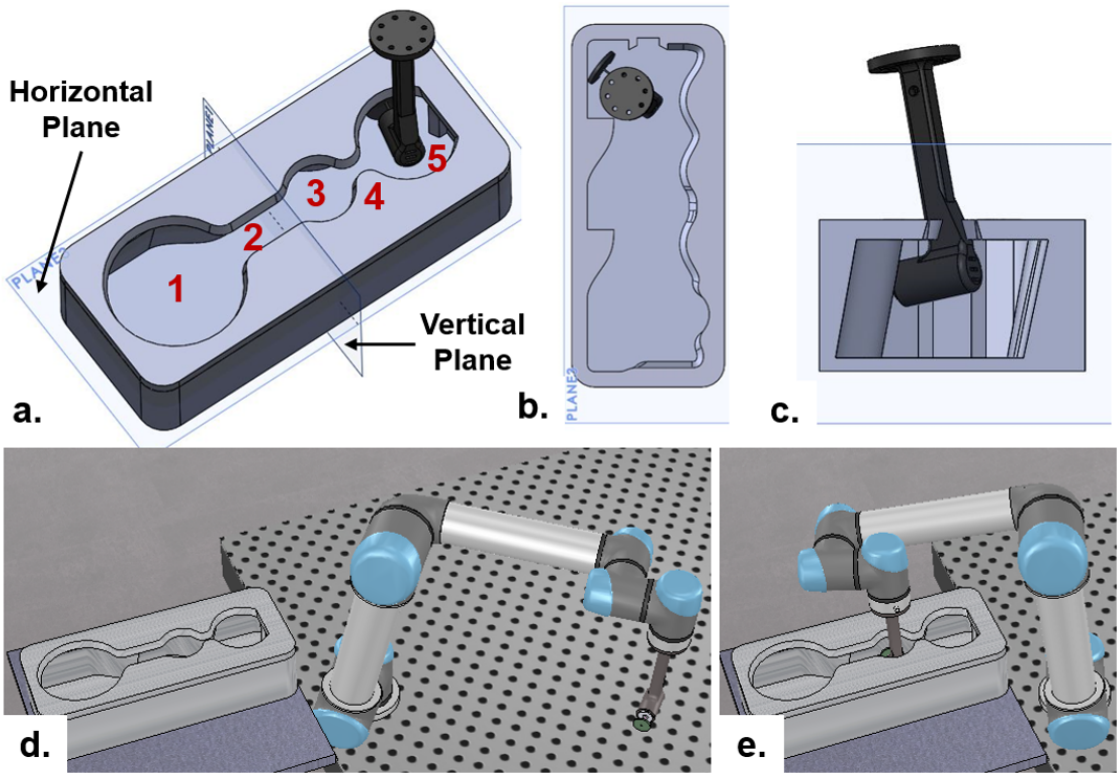


Figure 3.1: (a) A part with complex internal geometry, (b) Projection of horizontal plane section, (c) Projection of vertical plane section, (d) Robot in the initial configuration, (e) Robot in the goal configuration.

region 1 and moving it through region 2. The complex internal geometry (Fig. 3.1(b,c)) highly constraints the maneuvers possible by the robot.

The work seeks to generate a robot trajectory that can be executed in a time-optimal manner. However, search-based trajectory planning for manipulators with time-optimal execution cost can be intractable. Most applications with non-repetitive tasks require fast computation of near-optimal, feasible trajectories. The previous work has shown that the use of multiple different heuristics and adaptive primitives can improve search performance in manipulator trajectory planning applications. The work builds upon these findings. The work hypothesizes that the understanding of the contexts is important in designing the right heuristics and primitives. The work presents a COntext DEpendent Search Strategy Switching (CODES3) algorithm. The algorithm presents (1) context-dependent search strategies for searching different regions of the search space

efficiently for manipulators operating in confined workspaces and (2) identifying context based on the neighborhood analysis of the state space. This enables switching search strategies depending upon the context and improves search performance.

3.2 Problem Formulation

Let $\Theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$ be the joint configuration of a serial link manipulator with n joints. Each joint is bounded within its specific range ($\theta_i^{min} \leq \theta_i \leq \theta_i^{max}$). We call $\mathscr{C}$ to be the configuration space of the robot which contains all possible joint angle combinations of the robot. Therefore, $\mathscr{C} = \cup\Theta$, for all possible Θ.

Let $\mathscr{W}$ be a subset of 3D Euclidean space and the robots work space is contained in $\mathscr{W}$. Each link of the robot is modeled as a rigid solid. We represent the robot at configuration Θ as a collection of rigid solids $\mathscr{R}(\Theta) \subset \mathscr{W}$. We assume that the geometric, kinematic, and dynamic models ($\mathscr{M}$) of the robot are well defined and available to us. Let $\mathscr{O} \subset \mathscr{W}$ be the set of obstacles in the robot workspace. We represent the set of joint configurations that lead to collision as $\mathscr{C}_{obs}$ (obstacle space). Therefore, $\mathscr{C}_{obs} = \{\Theta \in \mathscr{C} : \mathscr{R}(\Theta) \cap \mathscr{O} \neq \emptyset\}$. Let $\mathscr{C}_{free}$ represent the collision-free configuration space, i.e., $\mathscr{C}_{free} = \{\Theta \in \mathscr{C} : \mathscr{R}(\Theta) \cap \mathscr{O} = \emptyset\}$.

Let T be the homogeneous transformation matrix representing the pose (position and orientation) of the robot's end-effector in $\mathscr{W}$. For a given joint configuration Θ, we can find T by applying forward kinematics (FK). Similarly, given an end-effector's pose, we can find the joint configuration by computing inverse kinematics (IK).

In this work, we are interested in the trajectory planning problem of manipulator where the robot needs to move from one configuration to another in the configuration space. Let the start and goal configurations of the trajectory be Θ_{start} and Θ_{goal}, respectively ($\Theta_{start}, \Theta_{goal} \in \mathscr{C}_{free}$).

We define Θ_G to be a set of acceptable goal configurations for the trajectory in the neighborhood around Θ_{goal}. $\Theta_G = \{\tilde{\Theta}_{goal} \in \mathscr{C}_{free} : ||\tilde{\Theta}_{goal} - \Theta_{goal}|| < \epsilon\}$.

Let $\mathscr{P}(s)$ be a feasible path (collision-free and continuous) in the configuration space parameterized by $s \in [0,1]$. $\mathscr{P}(0) = \Theta_{start}$ and $\mathscr{P}(1) \in \Theta_G$. We can convert $\mathscr{P}(s)$ into a trajectory $\tau(s)$ by assigning joint velocities for the robot in configuration space. In this work, we aim to minimize the trajectory execution time. Therefore, between two waypoints on $\mathscr{P}(s)$, we assign maximum joint velocity to the joint that needs to go through maximum displacement. We assign appropriate velocities to the rest of the joints such that all the joints start and finish their motion at the same time.

We consider collision-free and continuous trajectories as feasible trajectories. Let $\mathscr{T}$ be the set of all feasible trajectories from Θ_{start} to Θ_{goal}. Let $L(\tau): \tau \rightarrow (0, \infty)$ be the cost of a trajectory τ. Here, $L(\tau) = \infty$ corresponds to infeasible trajectories. Therefore, the optimal trajectory to move from Θ_{start} to Θ_{goal} will be, $\tau^* = \text{argmin}_{\tau \in \mathscr{T}} L(\tau)$.

Given a robot model ($\mathscr{M}$), workspace obstacles ($\mathscr{O} \subset \mathscr{W}$), start and end configurations ($\Theta_{start}, \Theta_{goal}$), an acceptable boundary around the end configuration (ϵ), and computation time bound (t_{max}), our objective is to device a branch and bound heuristic search algorithm to find a near-optimal trajectory, $\tilde{\tau}^* \in \mathscr{T}$.

3.3 Approach

3.3.1 Overview of Approach

Trajectory planning problems are PSPACE-hard [167] and it is computationally intractable to generate an optimal solution for the problems in high dimensional search spaces. The computational efficiency and the quality of the computed trajectory depend on the branching factor and the depth of the search tree.

The branching factor of the tree is directly proportional to the number of available motion primitives used during the expansion of each node. Moreover, it is inefficient to use the same set of primitives in the complete workspace around the robot. The algorithm presented in this work

Algorithm 1 CODES3 Algorithm($\Theta_{start}, \Theta_{goal}, \vec{GH}, \vec{MP}$, t_{max})

$InitializeDataStructs(\Theta_{start}, \Theta_{goal}, \vec{GH}, \vec{MP})$ {$\vec{GH}$ is the list of all the Guiding Heuristic Functions. $\vec{MP}$ is the list of all the motion primitives. $\vec{PQ}$ is the list of all the priority queues. The size of $\vec{PQ}$ is the same as $\vec{GH}$, and each queue $\vec{PQ}_i \in \vec{PQ}$ sorts the nodes according to the cost function $\vec{GH}_i$. $\vec{HIST}$ is the database containing the list of the last n expanded nodes. $\vec{CLOSED}$ stores the list of all the expanded nodes.}
while $t_{elapsed} < t_{max}$ **do**
  $gh\_id \leftarrow SelectGH(\vec{GH}, \vec{HIST}, \vec{COL})$
  $\vec{n}_{curr} \leftarrow GetNextBestNode(\vec{PQ}, \vec{HIST}, gh\_id)$
  if $ReachedGoal(\vec{n}_{curr})$ **then**
    $\tau \leftarrow ReconstructPath(\vec{n}_{curr})$
    return τ
  end if
  $\vec{HIST} \leftarrow \vec{HIST} \cup \vec{n}_{curr}$
  if $\vec{n}_{curr} \in \vec{CLOSED}$ **then**
    continue
  else
    $\vec{CLOSED} \leftarrow \vec{CLOSED} \cup \vec{n}_{curr}$
  end if
  $\vec{COL} \leftarrow RandomSampling(\vec{n}_{curr})$
  $\vec{P}(\vec{n}_{curr}) \leftarrow SelectMP(\vec{n}_{curr}, \vec{MP}, \vec{HIST}, \vec{COL})$
  $n_{children} \leftarrow Expand(\vec{n}_{curr}, \vec{P}(\vec{n}_{curr}), \vec{COL})$
  for n_{child} *in* $n_{children}$ **do**
    if $n_{child} \in \vec{CLOSED}$ **then**
      continue
    end if
    $\vec{n}_{child}.parent \leftarrow GetParentByBT(\vec{n}_{curr})$
    $\vec{n}_{child}.g \leftarrow GCost(\vec{n}_{curr}, \vec{n}_{start})$
    for $i \leftarrow \{1, |\vec{GH}|\}$ **do**
      $\vec{n}_{child}.h_i \leftarrow HCost(\vec{n}_{curr}, \vec{n}_{goal}, \vec{GH}_i)$
      $PQ_i.push(\vec{n}_{child}.h_i, \vec{n}_{child})$
    end for
  end for
  $flag_{hmn} \leftarrow IsHumanIPReq(\vec{HIST}, \vec{COL})$
  if $flag_{hmn}$ **then**
    $\tau \leftarrow ComputeTrajUsingHumanIP(\cdots)$
    return τ
  end if
end while

uses a list of a distinct set of motion primitives and switches between them depending upon the estimated configuration space obstacle and the progress of the search (see Section 3.3.8). On the other hand, the depth of the tree is dependent upon the length of the motion primitives. This requires the algorithm to vary the length of the primitives depending upon the estimate of the configuration space obstacles around each expanded node (see Section 3.3.8). Further, the set of motion primitives that moves one joint at a time increases the depth of the search tree and the execution time. We have introduced a feature of backtracking that combines the motion of several nodes that reduces the trajectory execution time (see Section 3.3.9).

The computational efficiency of the search relies on the selection of promising nodes that are used to expand the frontier of the search tree towards the goal. The selection of the promising nodes of the search tree relies on the heuristics used during the search. Well-designed search-guiding heuristics are required to solve the problem in a computationally-efficient manner. If a heuristic reflects the true cost (or cost-to-go) to reach the goal from a node, then it correlates to the promise that the node will be lying on the trajectory leading towards the goal. A heuristic may start misleading the search when the correlation is lost. A guiding heuristic that does not take search context into account may not work well throughout the entire search space. However, different guiding heuristics can be designed that can perform well in different regions of the space (see Section 3.3.4).

An efficient algorithm needs to switch heuristics to perform well in different regions. We have developed an algorithm that can make informed decisions to switch between these guiding heuristics. This algorithm is called *COntext DEpendent Search Strategy Switching* (CODES3). It makes the decision based on the progress of the search and the estimate of obstacles in the configuration space. We describe details of our search guiding heuristics for manipulator trajectory planning in Section 3.3.5. Finally, in the scenario where designed heuristic functions are not able to guide the search, the human operator can provide a subgoal to the search problem and divide the search into segments. The available heuristics may work well for the subsegments and accelerate

the overall search. This is often beneficial in practical applications where a feasible and fast solution is desired over cost-optimal but computation time-inefficient solutions. We discuss more on human input for guiding in section 3.3.7.

During the search, the nodes expanded in the complex and confined configuration spaces may have a marginally different cost. The search algorithm will invest a large number of computational resources evaluating all of these nodes with marginal improvement in the progress of the search tree. In this work, we dynamically measure the progress of the search and the estimated free configuration space to select nodes that are diverse in nature. The diversity of a node can help the search algorithm to prune branches and direct the resources towards the nodes that improve the progress of the search towards goal. (see Section 3.3.5).

3.3.2 Description of Algorithm

We present our approach in Alg. 11. Line 1 states the initialization routine that initializes all the data structures and flags. The function *SelectGH* in Line 3 is used to select an appropriate heuristic function that can be used by the search (see Section 3.3.4 and 3.3.5). Line 4 of the algorithm uses a function $GetNextBestNode$ that uses the history of nodes and evaluates the diversity condition to get the next best node for the provided heuristic cost function $\vec{GH}_{gh\_id}$ (see Section 3.3.6). Lines 10-14 is used to ensuring that each node is expanded just once. Line 15 gets the collision data around the current node from the Sampling Routine that is running parallel to the search. Lines 16-17 selects the appropriate motion primitive set that will be used to determine children for the current node (see Section 3.3.8). Line 22 includes a function $GetParentByBT$ that iterates through the previous n nodes in the search from the current node and gets the best new parent for the child node (see Section 3.3.9). Lines 24-27 compute the heuristic value using all the defined heuristic functions and pushes the child node in the priority queue using an appropriate cost function. Lines 29-33 determine the need for human input, invoke a function

that will reset all the data structures, and start two new instances of Alg. 11 with modified start and goal using an intermediate configuration provided by the human (see Section 3.3.7).

3.3.3 Search

The state space used for search consists of a discrete version of the continuous configuration space $\mathscr{C}$, denoted by $\mathscr{C}_d$. Each state in the state space consists of a discrete joint configuration of a serial link manipulator with n joint values denoted by $\Theta_d = \{\theta_{d,1}, \ldots, \theta_{d,n}\} \in \mathscr{C}_d$. The graph used for the search consists of nodes that correspond to discrete states in $\mathscr{C}_d$, and each node is denoted by $\vec{n}$. The nodes corresponding to the start Θ_{start} and goal Θ_{goal} states are represented by $\vec{n}_{start}$ and $\vec{n}_{goal}$ respectively. The transition between the current node $\vec{n}_{curr}$ to child node $\vec{n}_{child}$ in the graph is achieved by a set of motion primitives $\vec{P}(\vec{n}_{curr}) \in \vec{MP}$, where $\vec{MP}$ is the list of all motion primitive sets.

In this work, we use heuristic functions for the selection of promising nodes to guide the search towards the goal node $\vec{n}_{goal}$. The heuristic function used for expanding nodes in the least cost fashion is given by $h(\vec{n}) = w_1 g(\vec{n}) + w_2 k^c(\vec{n}) + w_3 k^w(\vec{n}) + w_4 l(\vec{n})$. Here, the functions $k^c(\vec{n})$, and $k^w(\vec{n})$ are the guiding heuristics in configuration space, and workspace, respectively (see Section 3.3.4). Finally, the function $l(\vec{n})$ is termed as the diversity function, which is used to evaluate the change in the robot configuration (or state) with respect to previously expanded nodes during the search. This cost is dynamically varied as the search progresses (see Section 3.3.5). The cost function used to evaluate the quality of the solution is given by $g(\vec{n})$, where $g(\vec{n})$ is the cost-to-come to node $\vec{n}$ from the start node $\vec{n}_{start}$.

3.3.4 Heuristic Function

The graph search-based algorithm uses a heuristic function to approximate the cost-to-go from the current node to the goal node in the configuration space. However, that may not be sufficient to guide the search in the complex workspace environment. This is primarily due to the fact that

we do not have complete knowledge of the configuration space obstacles, especially for robots with high DOF. Therefore, we have introduced several new heuristic functions that will help to guide the search in conjunction with the traditional heuristic function $k^c(\vec{n})$ in configuration space. The new guiding function $k^w(\vec{n})$ that is introduced as a part of the heuristic function $h(\vec{n})$ will approximate the search progress in the workspace. Also, $k^w(\vec{n})$ can leverage the complete knowledge of the workspace obstacles and navigate the search in the regions that may be closer to the workspace representation of the goal configuration.

Let the 6D workspace representation of the tool tip of each node $\vec{n}$ be represented by $\vec{n}^{\vec{x}} = \{x, y, z\}$ (position representation), and $\vec{n}^{\vec{q}} = \{q_x, q_y, q_z, q_w\}$ (orientation representation using quaternion) of the tool tip. Also, let the joint space configuration of $\vec{n}$ be represented by $\vec{n}^{\Theta} = \{\theta_1, \ldots, \theta_n\}$.

In our current implementation, we use Fast Marching Method (FMM) [168] for estimation of the cost-to-go in 3D (x, y, z) workspace. For a given goal node $\vec{n}_{goal}$, we precompute 3D FMM of the entire workspace. Let $FMM(\vec{n})$ be the shortest path distance from the node $\vec{n}$ to $\vec{n}_{goal}$ in 3D workspace. Also, let $Q(\vec{n}) = 2cos^{-1}(|\vec{n}^{\vec{q}} \cdot \vec{n}^{\vec{q}}_{goal}|/(|\vec{n}^{\vec{q}}||\vec{n}^{\vec{q}}_{goal}|))$ be the difference in the orientation of the tool tip between the current $\vec{n}$ and the goal $\vec{n}_{goal}$ node. Now, the workspace guiding heuristics can be calculated as $k^w(\vec{n}) = w_1^{k,w} FMM(\vec{n}) + w_2^{k,w} Q(\vec{n})$, where the weights $w_1^{k,w}$, and $w_2^{k,w}$ are used to normalize the cost function.

During the computation of guiding heuristics for configuration space $k^c(\vec{n})$, we consider the time cost to move from the joint angle configuration of a node $\vec{n}$ to goal $\vec{n}_{goal}$. Let v be the maximum attainable joint speed for the joint that needs to rotate most. Now, the heuristic function $k^c(\vec{n})$ can be computed as $k^c(\vec{n}) = (||\vec{n}^{\Theta} - \vec{n}^{\Theta}_{goal}||_\infty)/v$. Finally, the cost-to-come to node $\vec{n}$ from start $\vec{n}_{start}$ can be calculated as $g(\vec{n}) = g(\vec{n}.parent) + (||\vec{n}^{\Theta} - \vec{n}^{\Theta}.parent||_\infty)/v$, where $\vec{n}.parent$ is the parent node of the node $\vec{n}$ in the search tree. If $\vec{n}.parent = \emptyset$, then the value of $g(\vec{n}) = 0$. The cost-to-come $g(\vec{n})$ is used as a part of the heuristic function and also to compute the execution cost of the computed trajectory.

3.3.5 Identification of Promising Nodes

In the graph search-based algorithm, each node $\vec{n}$ is pushed into a priority queue and selected to expand based upon the estimated promise computed by a single heuristic function $h(\vec{n})$. However, a single function is unable to guide the search throughout the entire workspace. Our algorithm dynamically updates the evaluation function that is used to select the frontier node.

We define a set of n heuristic functions $h_i(\vec{n}) \in \vec{GH}$, where $i \in \{1, n\}$, and $\vec{GH}$ is the set of all heuristic functions. Each heuristic function $h_i(\vec{n})$ is generated by varying weights w_1, w_2, and w_3 of the cost function $h(\vec{n})$ as described in Section 3.3.3. We also define a set of n priority queues $PQ_i \in \vec{PQ}$ each sorting the nodes with respect to heuristic function $h_i(\vec{n}) \in \vec{GH}$.

The selection of heuristic for different regions in the workspace is achieved by maximizing the effectiveness of each heuristic with respect to congestion in the configuration space (line 3 Alg. 11, *SelectGH*). The search starts with an initial estimation of configuration space around the start node and selects to use heuristics *gh_id* that might be best for the regions around the start node. Here, *gh_id* is just the index of the heuristic. For example, if the start node is lying inside the concavity of the hollow part (e.g., Part 2 in Section 6.4), then it is beneficial to use a heuristic function that puts higher weights on $k^w(\vec{n})$ as compared to $k^c(\vec{n})$. This is under the assumption that the FMM values will be able to navigate the tooltip inside the part without getting stuck in local concavities. Further, the heuristics are also switched for the next best heuristic if the selected heuristic *gh_id* is not making progress towards the goal.

In our implementation, the progress towards the goal is measured by calculating the mean change in the heuristic cost over the last n nodes ($HIST$ in Alg. 11). If the mean change in heuristic cost is below a threshold $\epsilon_{h,i}$, then the heuristic is switched for the next best heuristic function. Also, the estimation of congestion in configuration space helps to select the right heuristic. This is achieved by sampling the configuration space, selecting a set of sampled points around the vicinity of the previous node $\vec{n}_{prev}$, and performing principal component analysis (PCA) on

the configurations (samples) that are colliding and collision-free. The sampling of the configuration space is performed in parallel to the search algorithm. From the principle components, both in configuration space and workspace, we can locally identify which direction in the configuration space is relatively free, and by what scale. During the search, each node queries in the sampling thread for the estimated congestion values around that node (see line 15 in Alg. 11).

3.3.6 Pruning of Nodes in Bottleneck Region

Despite using several heuristics, there will be areas of the workspace where the search progress towards the goal is minuscule. However, given enough time, the search will converge and reach the goal. These regions of the workspace where the progress of the search is hindered are referred to as bottleneck regions. In these regions, the search tree has a large number of nodes with a marginal change in heuristic cost as compared to the previously expanded nodes. The search algorithm spends a large amount of computational resources by sequentially evaluating the next best nodes and eventually expanding nodes that can lead the search out of the bottleneck region. In order to improve the computational performance of the algorithm, we have introduced a cost $l(\vec{n})$ in the heuristic function $h(\vec{n})$ (see Section 3.3.3) that estimates the diversity of the expanded node. The diversity of a node is used to prune branches of the search tree that have joint configurations that are marginally different from the current node. This cost is added to the cost function $h(\vec{n}_{curr})$ after the node is popped out of the priority queue PQ_{gh_id} and the new cost is evaluated to determine if the node is diverse and should be expanded.

In our implementation, the region is considered to be the bottleneck region if the maximum of the mean change in the heuristic cost over all the heuristics in $\vec{GH}$ is less than ϵ_l. Further, the cost $l(\vec{n}_{curr})$ is computed by taking the normalized distance to the previously expanded node $\vec{n}_{prev}$. The weight w_4 associated with the cost function $l(\vec{n})$ in heuristic cost function $h(\vec{n})$ is kept to be negative. Thus, it discounts the cost $h(\vec{n}_{curr})$, compares it with the heuristic cost $h(\vec{n}_{prev})$

of the previously expanded node, and if $h(\vec{n}_{curr}) < h(\vec{n}_{prev})$, then the node is evaluated. The cost $l(\vec{n}_{curr}) = 0$ in the areas other than the bottleneck regions.

3.3.7 Human Aid to Guide Search

The algorithm prompts the human for an input when the designed heuristic functions are unable to guide the search. The algorithm computes the mean change in heuristic value for all the heuristic functions $\vec{GH}$. If it drops below a constant threshold $\epsilon_{human,h}$, and the change in the joint configuration of the selected nodes drops below $\epsilon_{human,j}$. Then the human is notified. The user evaluates the progress of the search and determines the constriction region in the workspace. The user is expected to select an intermediate waypoint $\vec{n}^{\Theta}_{interim}$ to circumnavigate the constriction region. These intermediate waypoints are used as $\vec{n}'_{start}$ and $\vec{n}'_{goal}$ for two new instance calls to Alg. 11 which compute the paths independently and combine them to get the complete path from start to goal node.

Table 3.1: Performance comparison between the Bi-directional RRT, ARA*, BFS, and the developed algorithm on the test parts

Scenario	Computation Time (in sec)					Expanded States					Trajectory Execution Time (in sec)				
	Bi-RRT	ARA	BFS	CODES3	% Reduction in CODES3 with BFS	Bi-RRT	ARA	BFS	CODES3	% Reduction in CODES3 with BFS	Bi-RRT	ARA	BFS	CODES3	% Reduction in CODES3 with BFS
1a	51.7	131.5	38.8	8.1	79.1	46	13475	4007	201	95.0	3.3	N/S	5.0	3.4	N/S
1b	27999.0	90.2	64.3	5.4	91.5	31645	10636	5917	128	97.8	3.2	5.2	5.4	3.4	34.8
2a	6642.7	18.1	16.6	5.5	67.2	1291	2136	1782	251	85.9	2.8	3.7	3.8	1.9	48.0
2b	1524.0	15.4	9.8	9.5	3.1	434	1528	687	261	62.0	2.1	2.2	2.3	2.4	-9.8
2c	5539.9	34.7	27.4	9.0	67.3	6514	3380	2344	237	89.9	2.8	2.5	3.3	2.4	6.3
3a	5053.2	62.5	29.0	10.6	63.4	6420	8132	2856	385	86.5	4.7	N/S	4.7	2.3	N/S
3b	2.1	31.5	35.8	0.5	98.6	12	3175	1110	11	99.0	2.7	3.0	3.0	2.6	11.1
3c	6.0	6.4	3.6	5.5	-54.4	48	788	388	148	61.9	1.1	1.2	1.2	1.0	15.0
4d	3.2	5.2	8.7	14.9	-70.4	16	463	490	381	22.2	1.4	1.5	1.5	2.0	-31.1
3e	N/S	N/S	294.6	439.3	-49.1	N/S	N/S	11752	8397	28.5	N/S	N/S	11.4	4.9	N/S
4a	4.4	10.6	11.7	0.4	96.8	7	1636	1636	3	99.8	2.2	2.9	2.9	2.2	26.2
4b	33.6	52.7	55.1	29.9	45.8	21	7417	7074	813	88.5	7.7	12.5	12.6	4.6	63.2
4c	N/S	N/S	N/S	17.8	N/S	N/S	N/S	N/S	416	N/S	N/S	N/S	N/S	0.5	N/S
4d	N/S	N/S	N/S	19.6	N/S	N/S	N/S	N/S	435	N/S	N/S	N/S	N/S	2.2	N/S
4e	N/S	N/S	N/S	N/S	N/S	N/S	N/S	N/S	N/S	N/S	N/S	N/S	N/S	N/S	N/S
5a	N/S	N/S	14065.2	19.1	99.9	N/S	N/S	125760	610	99.5	N/S	N/S	1.7	1.1	N/S
6a	16.4	67.0	23.4	5.3	77.5	37	5575	2655	141	94.7	3.2	3.9	4.0	3.5	9.1

3.3.8 Context Dependent Selection and Scaling of Primitives

In our current implementation, we generate two distinct sets of motion primitives. The first set contains motion primitives that move one joint of the robot at a time in the configuration space and is termed as $\vec{P}^c \in \vec{MP}$. The second set contains motion primitives that move the tooltip of the robot in 6D workspace $(x, y, z, \alpha, \beta, \gamma)$ and termed as $\vec{P}^w$. We take the IK solution to convert the workspace motion primitives into joint configurations.

The search starts with an initial estimation of the congestion around the start node and selects the appropriate set of motion primitives corresponding the congestion value. The congestion value is obtained by querying the sampling thread for the estimated congestion value around the current node $\vec{n}_{curr}$. Similar to the selection of heuristic cost function, we perform a PCA on the sampled collision-free states around the current node $\vec{n}_{curr}$ and evaluate the free space in different directions of the workspace and configuration space. The principle components in the workspace can be used to identify which directions of the workspace are locally free. Similarly, the principal components in configuration space are used to identify which joints have more room for movement in the local neighborhood. Further, we select the primitive set that has the most free space around the current node. This method of selection of primitive might not be advantageous in scenarios where the generated child node does not minimize the selected heuristic function $h(\vec{n})$. Thus, we also consider the history ($HIST$) of the previously expanded nodes by each set of primitives and determine the effectiveness of each set.

In the experiments presented in Section 6.4, it was observed that the workspace-based motion primitives $\vec{P}^w$ works better in the interior regions of the parts as compared to the configuration-based heuristic $\vec{P}^c$. However, when the search reaches the bottleneck regions (see Section 3.3.6) it is beneficial to search in configuration space primitives $\vec{P}^c$.

In this work, we not only use different sets of motion primitives but also dynamically scale the length of the primitives by considering the congestion around the current node $\vec{n}_{curr}$. The length

of the motion primitives is determined by performing a binary search in the bounds provided for the selected dimension. For example, the motion primitives that are generated by varying the x component of the pose would be $(x \pm \delta_x, y, z, \alpha, \beta, \gamma)$. Here, the value of the δ_x is determined by performing binary search within bound $[\delta^{\vec{n},x}_{min}, \delta^{\vec{n},x}_{max}]$. The bounds $\delta^{\vec{n},x}_{min}$, and $\delta^{\vec{n},x}_{max}$ are determined by the congestion values around the node $\vec{n}$.

3.3.9 Backtracking and Trajectory Repair

In this work, we introduce the backtracking feature that directly connect nodes in the search tree by combining several motion primitives. This is implemented by evaluating collision free path between the new child node, $\vec{n}_{child}$ and n-step parent nodes of the current node, $\vec{n}_{curr}$ (see line 22, alg. 11). We define the i^{th} parent node of $\vec{n}_{curr}$ as $\vec{n}_{curr}.parent_i, i = 1, \ldots, n$, where $\vec{n}_{curr}.parent_1 = \vec{n}_{curr}.parent$, $\vec{n}_{curr}.parent_2 = (\vec{n}_{curr}.parent).parent$, and so on. We start from the furthest parent ($i = n$) to evaluate collision free path and keep on reducing i until $i = 1$. If the combined motion from the i^{th} parent node to the child node is collision free, then the child $\vec{n}_{child}$ is directly connected to the $\vec{n}_{curr}.parent_i$. Thus, the time cost $g'(\vec{n}_{child}) = g_{n_{curr}.parent_i} + (||\vec{n}^{\Theta}_{child} - \vec{n}^{\Theta}.parent_i||_\infty)/v$ and is guaranteed to be less than or equal to the $g(\vec{n}_{child})$. The higher the value of n lower is the trajectory execution cost, but higher is the computation time of the search.

Table 3.2: Performance comparison on the test parts by activating/deactivating different features of CODES3 algorithm

Scenario	Computation Time (in sec)					States Expanded					Trajectory Execution Time (in sec)				
	M1	M2	M3	M4	% Reduction in M4 over M1	M1	M2	M3	M4	% Reduction in M4 over M1	M1	M2	M3	M4	% Reduction in M4 over M1
1a	5.7	5.6	5.5	8.1	-41.7	201	201	201	201	0.0	6.2	6.2	6.2	3.4	44.9
1b	3.6	3.7	3.5	4.4	-22.6	128	128	128	128	0.0	6.2	6.2	6.2	3.4	45.6
2a	831.9	4.0	3.9	5.5	99.3	24599	251	251	251	99.0	18.8	3.8	3.8	1.9	48.6
2b	109.7	10.8	7.6	9.5	91.4	2902	350	261	261	91.0	4.9	3.7	3.4	2.4	30.2
2c	8.8	7.2	7.0	8.9	-0.6	251	237	237	237	5.6	3.9	3.9	3.9	2.4	38.3
3a	262.5	11.6	7.6	10.6	96.0	7216	502	385	385	94.7	10.0	5.6	4.5	2.3	49.2
3b	0.4	0.4	0.4	0.5	-5.3	11	11	11	11	0.0	2.7	2.7	2.7	2.6	1.3
3c	75.1	10.9	4.3	5.5	92.7	2065	313	148	148	92.8	2.9	2.0	1.8	0.9	49.8
4d	2924.0	20.0	11.8	14.9	99.5	65312	580	381	381	99.4	9.4	4.0	4.1	1.9	54.0
3e	2420.9	1985.7	365.1	439.3	81.9	49280	49602	8397	8397	83.0	9.7	10.4	7.2	4.9	31.3
4a	0.3	0.3	0.3	0.4	-10.5	3	3	3	3	0.0	2.5	2.5	2.5	2.2	12.7
4b	N/S	10.3	10.5	29.9	N/S	N/S	813	813	813	N/S	14.2	13.4	13.4	4.6	65.8
4c	295.5	15.6	14.2	17.8	94.0	1514	426	416	416	72.5	1.3	1.3	1.9	0.5	73.4
4d	20.2	15.8	15.8	19.6	3.4	462	435	435	435	5.8	10.8	4.2	4.2	2.2	46.7
4e	N/S	N/S	N/S	N/S	N/S	N/S	N/S	N/S	N/S	N/S	N/S	N/S	N/S	N/S	N/S
5a	N/S	14.0	14.8	19.1	N/S	N/S	610	610	610	N/S	N/S	2.7	2.7	1.1	N/S
6a	310.2	3.6	4.2	5.3	98.6	6489	122	141	141	97.8	3.8	4.4	4.6	3.5	23.4

3.4 Results

We have considered the application of robotic surface finishing of complex internal regions for our experiments. The robot needs to guide a tool inside a complex and constrained geometry to finish internal surfaces. Fig. 8.10 illustrates the 3D models of six parts that we used in our simulation experiments to test the performance of CODES3. The complexity of the test parts increases from 1-4. Parts 5-6 were used for physical experiments. We considered multiple start-end configurations of the robot for each part. The test scenarios also included cases where both the start and end configuration required the tool to be inside the part. For simplicity, we will be referring the test scene for part 1, case a as scenario $1a$ and so on. We have tested our algorithm in simulation (using UR5 robot) and physical experiments (using UR5 & KUKA iiwa7 robots). The video demonstrating the robots executing the trajectories in simulation and physical experiments is shown in [169].

The algorithm was implemented using MATLAB on a computer with an Intel Xeon 3.50GHz processor and 32GB of RAM. We considered the grid resolution of 1^o for configuration space and 0.01 meters for workspace. The maximum planing time for each scenario was kept to be $t_{max} = 5$ hours.

We have benchmarked our method against RRT-Connect, Best First Search (BFS), and Any-time A* (ARA*) in the simulation experiments. We have selected ARA* instead of A* for comparison due to the poor computation performance of A* in its vanilla implementation. The cost function used for ARA* is $f(\vec{n}) = g(\vec{n}) + \epsilon k^c(\vec{n})$, where $\epsilon = 5$ was used as inflation factor. The cost function used for the BFS was $f(\vec{n}) = k^c(\vec{n})$. The motion primitives set used for ARA*, and BFS consists of $\pm 1^o$ motion for each joint of the robot. We can see that ARA* has better computation time in few scenarios as compared to BFS. This is mainly due to the skewed cost function of ARA* which involves g and inflated h terms as compared to just the h term in the case of BFS. Moreover, the results presented for RRT-Connect, BFS, and ARA* in Table 3.1 can

be further be improved by tuning the planning parameters for each part. The purpose of these comparisons was to assess the relative merit of each method across parts. Therefore we did not attempt to tune parameters to improve performance on individual parts.

In our experiments, the range of scalable configuration space primitives $P^c(\vec{n})$ was $\pm[1^o, 20^o]$ and the range of workspace primitives $P^w(\vec{n})$ was $\pm[0.01, 0.15]$ meters for $[x, y, z]$, and $\pm[1^o, 20^o]$ for $[\alpha, \beta, \gamma]$ (Euler angles). The joint velocity used by the cost function $g(\vec{n})$ and $k^c(\vec{n})$ was kept to be 1 rad/sec. The weights $\epsilon_{h,i}$ (see Section 3.3.5) for both the heuristic function were kept constant at 0.01. We just used immediate parent ($n = 1$) for backtracking. Finally, we analytically computed the FK and IK for UR5 robot.

Table 3.1 shows the comparisons among the methods in computation time, node expansion, and trajectory execution time. The cells marked with 'N/S' represents the cases where the planners were not able to generate solution within time-bound t_{max}. We can see that our method performed equivalent on the parts with simple geometries and significantly better in the cases where the part have complex geometries. None of the methods, except our method, was able to find a solution for scenarios 4*c*-4*e*. Further, our algorithm was not able to compute the trajectory for scenario 4*e*.

We have also compared the search performance (see Table 3.2) by introducing each feature and compared its performance with the baseline. The baseline method 1 (M1) presented in the table searches using scalable workspace primitives $P^w(\vec{n})$ and cost function $h(\vec{n})$ with weights $[w_1, w_2, w_3, w_4] = [0, 0, 1, 0]$ (see Section 3.3.3). Each feature we add in methods 2-4 is an addition to the previous method. Method 2 (M2) uses a hybrid set of primitives $P^c(\vec{n})$ and $P^w(\vec{n})$ along with weights of the cost functions $[w_1, w_2, w_3, w_4] = [0, 1, 1, 0]$. Method 3 (M3) uses a hybrid set of primitives along with pruning nodes using diversity condition and the weights of the cost functions $[w_1, w_2, w_3, w_4] = [0, 1, 1, 1]$. Finally, Method 4 (M4) is an extension to M3 with the backtracking feature.

In the cases with simple parts (e.g., part 2) the computation time of all the methods is similar. However, the M4 is able to generate trajectories with low execution cost and a minor increase in computational time. This is primarily due to the backtracking feature, which combines motions associated with multiple nodes into a single motion that leads to lower execution cost. Further, there are a large number of cases where M1 have significantly large computation time as compared to M2. This is because the workspace cost function is not able to guide the orientation of the tool. Also, M1 is unable to compute a path for scenario 5*b*. This is because the tool reaches the goal configuration in position. However, the robot requires a large flip in one of the joint angles in order to achieve the goal orientation and the cost function $k^w(\vec{n})$ is unable to guide the search. This is improved in M2 which solves the same case (5*b*) in just 10 sec.

The method M3 performs the same as M2 in several scenarios where there were no bottleneck regions encountered (see Table 3.2). However, in the few scenarios like 3*c*-3*e* there were bottleneck regions encountered between the start to the goal state. These scenarios were mainly going from large hole region of part 3 (see Fig. 8.10) to small hole region (3*c*,3*d*), and coming out of the part from the small hole region (3*e*). In such scenarios, the algorithm invests a lot of time expanding the nodes with a minor change in joint configuration, therefore the diversity condition helps to reduce the number of expanded nodes and reduces the computation time by approximately $50 - 60\%$ with respect to M2 in scenarios 3*c*-3*e*, and 4*b*. Finally, method 4 (M4) on an average reduces the trajectory execution time by approximately 40%. However, due to extra collision checks to determine connectivity with the parent, there is a marginal increase in computation time.

Among the test scenarios, scenario 4*e* was unsolved, and the computation time of scenario 3*e* was high. In these scenarios, we activate the human input feature, where the human provides an intermediate configuration between the start and the goal node. Fig. 3.4 shows the start, goal, and intermediate configuration of the robot for scenario 4*e*. Part 4 is designed such that the tool cannot pass through the channel between the start and goal states (see Section 6.1). In this example, the heuristic functions will misguide the search, and the planner will not converge

Table 3.3: Performance comparison on the scenarios 3e and 4e between BFS and the developed planner with human input

Scenario	Computation Time (in sec)		Trajectory Execution Time	
	BFS	CODES3	BFS	CODES3
3e	20.9	12.2	5.3	3.5
4e	2084.8	104.5	12.5	7.0

to a solution. However, when the planner gets additional input in the form of an intermediate configuration, then the heuristic function is able to guide the search easily from the start to goal via the intermediate node. Table 3.3 shows the computation time and trajectory execution time for BFS and CODES3 with human input. We can see that the computation time and execution time have reduced significantly for scenario 3e.

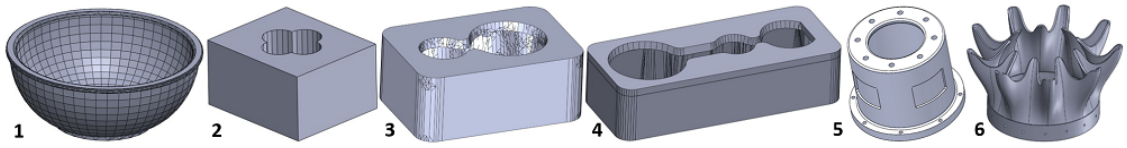


Figure 3.2: Six test parts used in our experiments

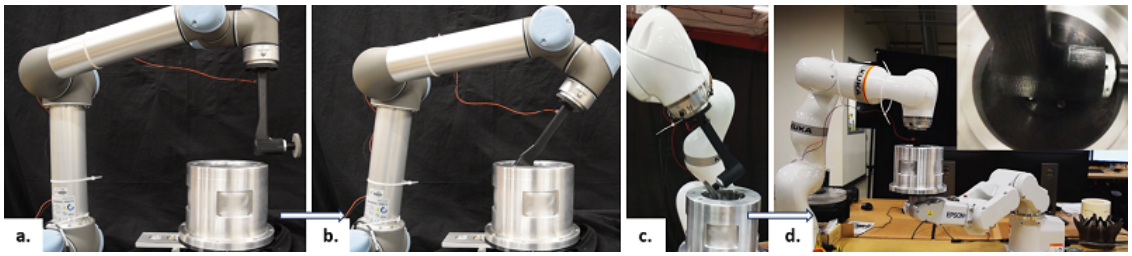


Figure 3.3: A 6-DOF (a,b) and a 7-DOF (c,d) manipulator are executing trajectories to take an L-shape tool inside a constrained workspace.

We have performed physical experiments on part 5 and 6 using UR5 (6DOF) and KUKA iiwa7 (7DOF) robots. These two robots are significantly different by design. The design on UR5 separate the wrist joints (last three) from the base, shoulder, and elbow joints (first three). This separation allows the planning algorithm to decouple the two sets of joints and plan independently. On the other hand, all the joints of iiwa7 are equally coupled. Therefore, the specialized algorithms developed for robots with independent wrist joints cannot be used for iiwa7. The context-dependent

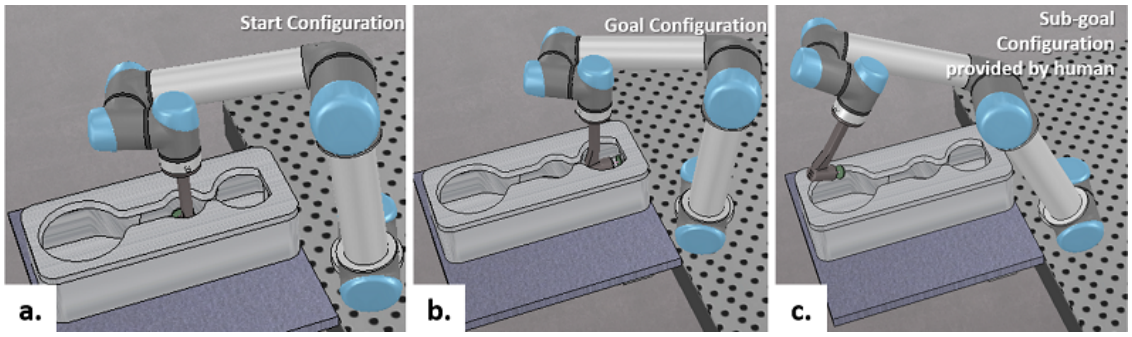


Figure 3.4: (a) Start configuration, (b) Goal configuration, (c) Human provided sub-goal configuration

search strategy switching enables our method to be independent of robot geometry. Fig. 3.3 shows the images of UR5 and iiwa7 executing computed trajectories. The computation time for iiwa7 in scenario 5*a* is 1300.6 sec, however out of which 1247.2 sec is taken up by numerical IK solver in MATLAB robotics toolbox. Therefore, the planning time comes down to 53.4 sec. Similarly, the computation time for scenario 6*a* is 167.6 sec out of which 133.5 sec for computation of numerical IK. In the future, we plan to integrate TracIK[170] for faster computation of numerical IK. Our preliminary experiments indicate that it will reduce the computation time by 10 to 100 folds.

3.5 Summary

The work demonstrates that context-dependent search strategy switching can significantly improve performance of manipulator trajectory planning in confined workspaces. The algorithm described in this chapter (1) uses context-dependent search strategies for searching different regions of the search space efficiently and (2) identifies context based on the neighborhood analysis of the state space to apply search strategies.

Chapter 4

Generation of Synchronized Configuration Space Trajectories with Workspace Path Constraints for Multi-Robot Systems

4.1 Introduction

The relative motion among the objects to carry out certain tasks can be defined as path-constraints. Figure 4.1 illustrates an example. These path-constraints or the motion of the objects can be generated using traditional motion planners [33, 171]. However, if we want to automate these tasks using robots, then we need to generate trajectories for robots in addition to generating the motion plans for the objects. For example, let us consider a buffing operation of a reflective mirror, as illustrated in Figure 4.1. The tool needs to stay normal to the surface during the operation. We can define the tool path (or the path-constraint between the tool and the part) in the robot's workspace as a parametric curve. We can attach the tool to the end-effector of a robot, as shown in Figure 4.2, to automate the task. We can generate the robot's trajectory for performing the task using existing approaches [73, 172]. However, the complete surface area of the large part is not reachable with the desired tool orientation by a single robot (shown in Figure 4.2).

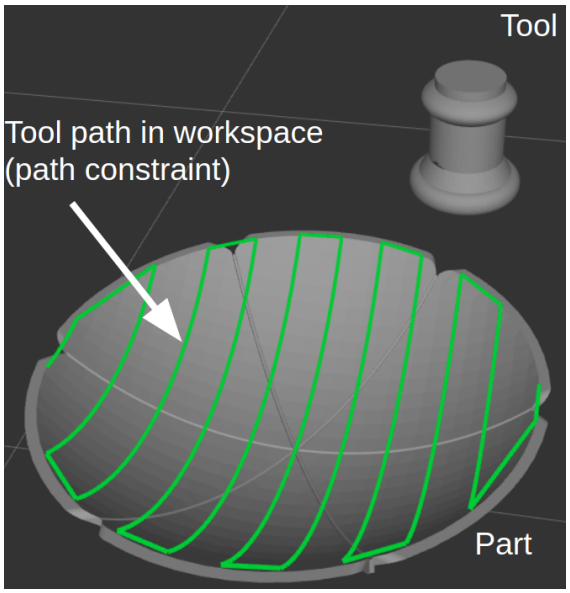


Figure 4.1: Illustration of a part to be buffed and a sanding tool. The tool needs to follow the path projected on the part's surface to complete the sanding operation. The position and orientation that the tool needs to maintain are encoded in the tool path, and it is considered as a path-constraint for the tool. We will represent the tool path or path-constraints as parametric curves ($c(s) \subset SE(3)$, $s \in [0, 1]$) in this work.

Multi-robot systems can benefit us by expanding the operational workspace, increasing payload capacity, and reducing process time. We can consider a bi-manual system to complete the buffing task, as shown in Figure 4.3. A 7-DOF manipulator is operating the buffing tool, and another 7-DOF manipulator is holding the large part. In this bi-manual setup, one robot can operate the tool, and the other can move the part simultaneously. The synchronous motion of the robots enables buffing the complete surface area of the part. Theoretically, we can double the speed of the process by moving both the manipulators at their maximum speed. We want to generate feasible and time-optimal configuration space trajectories (e.g., joint trajectories) of the robots to accomplish the task. We refer to the robot trajectories as path-constrained trajectories since the motion of the robots need to satisfy the path-constraint between the objects (e.g., tool and part) under consideration.

The bi-manual setup under discussion is a redundant system. For each waypoint on the workspace path (path-constraint), there can be an infinite number of joint configurations of the

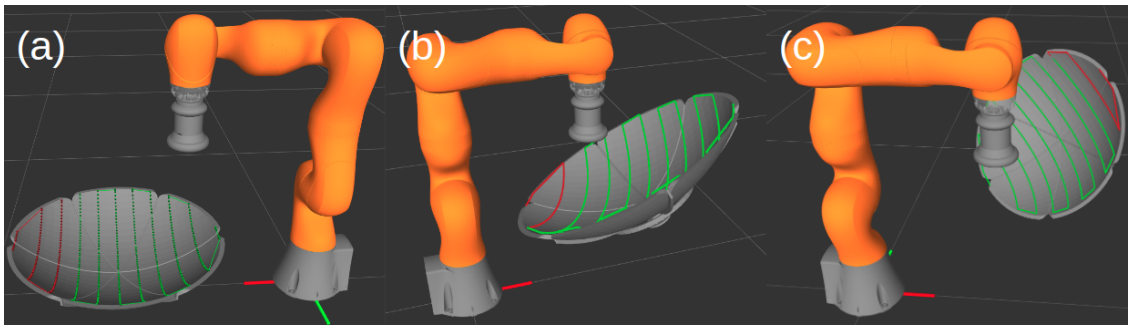


Figure 4.2: The part under consideration is large for the robot under consideration. The entire surface to be buffed is not reachable by a single robot. (a), (b), and (c) shows three different poses of the part and highlights the reachable and unreachable portions of the tool path. The red portion of the tool path is unreachable.

robots that bring the tool in contact with the waypoint. It is crucial to generate the complete path-constrained trajectory for the robots in configuration space as a whole. For example, if we take a sequential approach to find configuration space or joint space solutions for each of the waypoints, then we may not be able to guarantee a single continuous trajectory. The robots may even get stuck into singular positions or collide while tracing the constrained-path sequentially (an example is shown in Figure 4.4). It is also critical to choose the right parametric representation if the configuration space trajectories are generated as parametric curves. An inaccurate representation may not be able to generate a feasible solution. An example is shown in Figure 4.5, where each configuration variable is modeled with a third-order B-Spline curve with four control points. Figure 4.6 shows a case where an appropriate parametric representation (a combination of third-order B-spline curves, each with nineteen control points) is chosen for this example, and the solution is generated for the complete trajectory as a whole.

We can formulate the path-constrained trajectory generation for synchronous motion among robots as an optimization problem. The objective of this work is to minimize the processing time (trajectory execution time) and generate a safe trajectory (collision-free) that satisfies physical and application-specific constraints. This optimization problem is very challenging to solve

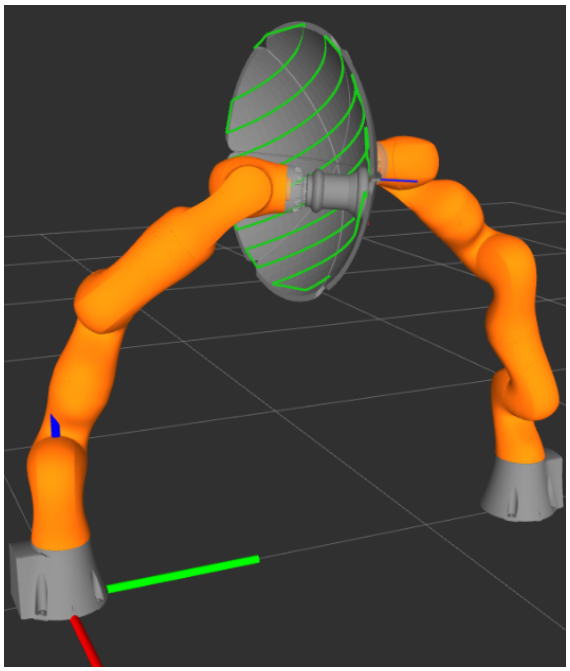


Figure 4.3: Illustration of a bi-manual buffing operation. It is possible to buff the entire surface by moving two robots in a synchronous manner. One robot can hold the part while the other operates the buffing tool.

as the motion constraints are highly non-linear (e.g., path-following, collision avoidance, physical constraints of the robots, application-specific constraints). Moreover, it is computationally challenging to generate trajectories in high-dimensional configuration space.

This chapter[1] presents the limitations of single-stage optimization and discusses how successive refinement strategies are effective to find feasible solutions. Moreover, this work presents algorithms to generate the trajectories by adaptively representing the configuration variables using multiple spline segments in a computationally efficient manner.

This work presents a method for solving the trajectory generation problem with the following functional capabilities.

- Selecting an appropriate representation of the high dimensional configuration space trajectories so that complex synchronous motion can be generated in the robots' workspace.

[1]The work in this chapter is derived from the work published in [173]

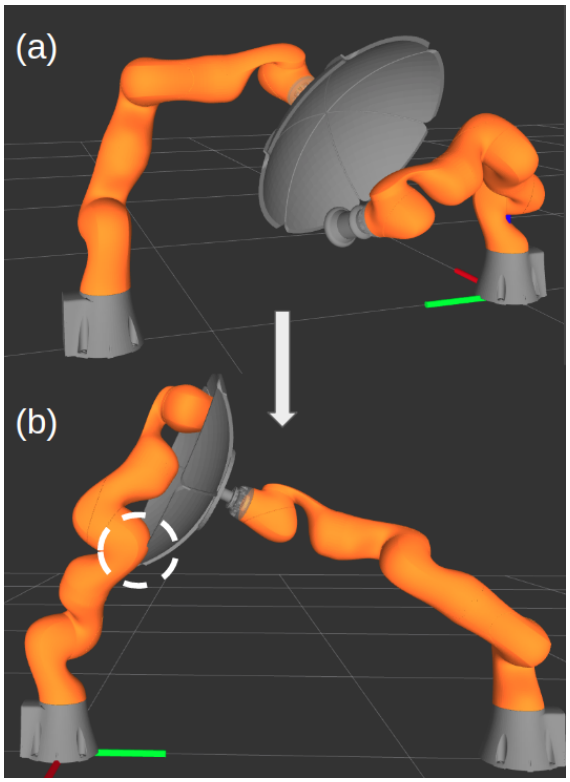


Figure 4.4: Illustration of the limitations of generating a path-constrained trajectory for the multi-robot system in a sequential manner. (a) shows the robot configurations generated by solving IK for the first waypoint on the path-constraint. (b) shows one robot getting stuck into a singular configuration as a result of sequentially generating robot configurations, starting from the initial configuration shown in (a), for the waypoints on the path-constraint. In this example, no feasible trajectory was possible using a sequential method as it led to a collision. The trajectories of the robots need to be generated as a whole for the complete path-constraint to avoid such incidents.

- Implicitly generating initial configurations of the robots to complete the constrained motion among the objects. This indirectly incorporates features of robot placement problem. It will be impossible to complete the motion through the constrained workspace paths without proper initial configuration of the robots.

- Exploiting all the DOF of the robots during execution to magnify relative speed and expand the operational workspace. This enables manipulating large objects and reducing the processing time.

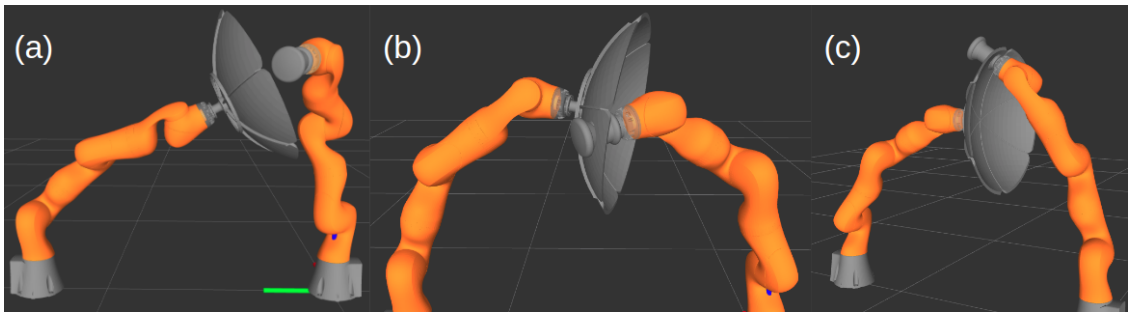


Figure 4.5: Illustrations of an infeasible trajectory with pose error and collisions. This can occur when the representation for configuration variables are not chosen properly for trajectory generation.

4.2 Problem Formulation

Let, $\mathscr{R} = \{R_1, R_2, \ldots, R_{n_R}\}$ be the set of robots in a multi-robot system. In this work, we will use the term robot to refer to a serial link manipulator or a mobile robot. Let, $\mathscr{O} = \{O_1, O_2, \ldots, O_n\}$ be the set of objects (tools, parts, etc.) present in the system. The objects are attached to the robots. We represent the ensemble of given objects and robots in a tree structure ($\mathscr{T}$). We consider the robots and objects as nodes in the tree. The tree starts with the ground or world frame (W) as the root and ends with the objects as the leaves. There are coordinate frames attached to each of the objects and each link of the robots. The robots have degrees of freedom, and their motion gets propagated through the edges of the tree. The motion of the robots in the tree will create relative motion among the objects. The degrees of freedom of the robots are the control variables in the system. We refer to them as configuration variables ($\Theta = \{\theta_1, \theta_2, \ldots, \theta_m\}$). They are functionals of time ($\theta_j(t)$). The robots will move for a given time-signal as the control input of the configuration variables. We refer to them as configuration variables ($\Theta = \{\theta_1, \theta_2, \ldots, \theta_m\}$), which are functionals of time ($\theta_j(t)$).

Different tasks introduce different relative motion constraints among objects. We are interested in problems where the objects in the tree have rigid motion constraints with respect to each other or with respect to W. These relative motion constraints are specified as workspace path constraints. We represent these paths as parametric curves,

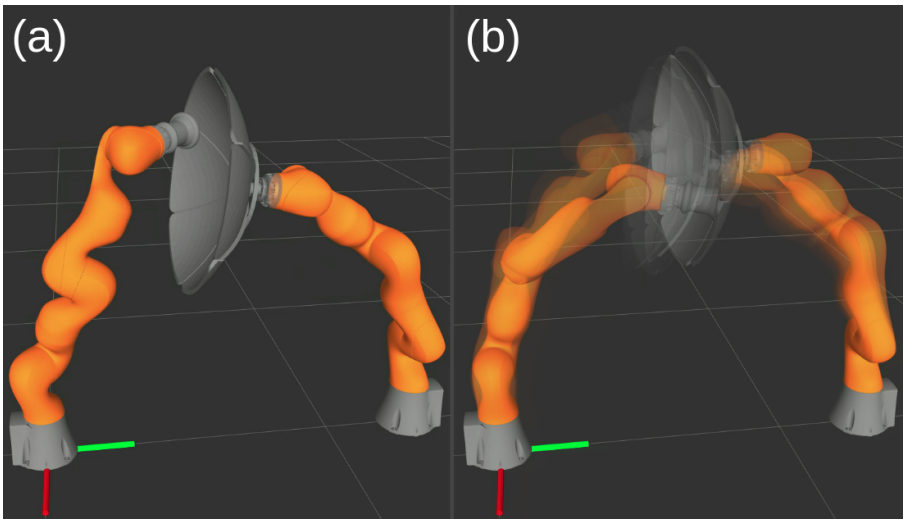


Figure 4.6: Illustration of a feasible trajectory for buffing the whole part. The complete synchronous robot trajectory for the entire path-constraint need to be generated as a whole such that all the process requirements are maintained, physical constraints of the robots are not violated, and there is no collision. (a) shows the robot configurations at the starting point and (b) shows the overlayed frames for the complete path.

$$\mathscr{C}(s) = \{c_i(s) \subset SE(3) : i = 1, 2, \ldots, n\} s \in [0, 1] \tag{4.1}$$

s is the arc-length parameter. We want to generate trajectories for the configuration variables $(\theta_j(t))$ such that the path-constraints for the objects are satisfied. Let us assume that the robots start executing the trajectory at time t_i and finishes at time t_f. Therefore, a map exists between time (t) and arc-length parameter (s) for the curves in $\mathscr{C}(s)$ such that at $t = t_i$, $s = 0$ and at $t = t_f$, $s = 1$.

Figure 4.7 illustrates an example of an ensemble of objects and robots. This is the bi-manual buffing task we discussed in section 6.1. Figure 4.8 illustrates the tree representation of this ensemble. A relative motion constraint is defined between a tool (O_1) and a part (O_2). The motion constraint is the tool path which we consider as a workspace path constraint $(c(s))$. The tool should follow a path $(c(s))$, which is defined with respect to a static frame located on O_2. This

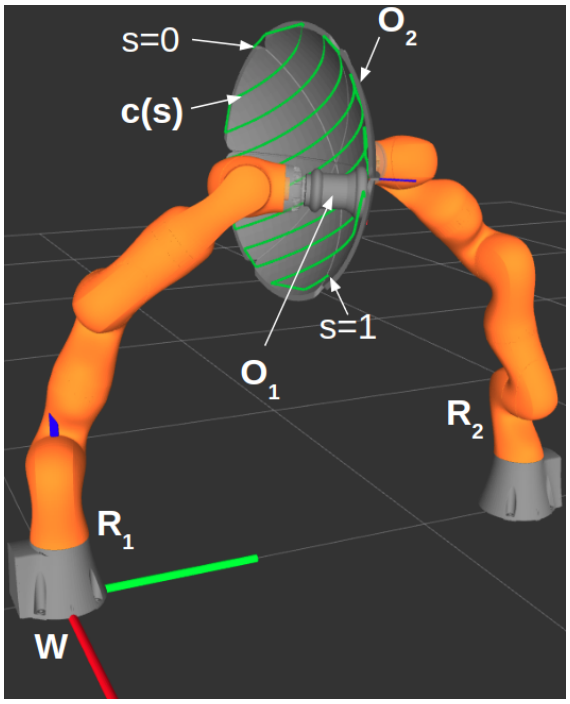


Figure 4.7: Example of an ensemble of robots and objects. The tool (object O_1) needs to follow a path defined on the surface of the part (object (O_2)). The tool path is considered as a path-constraint ($c(s)$). The path constraint can be represented as a collection of waypoints for the tool, which in turn can be represented as a parametric curve. ($c(s) \subset SE(3)$, where $c(s)$ is parameterized with arc-length parameter s). The first robot (R_1) is holding the part and the second robot (R_2) is holding the tool.

relative motion performs a buffing task. The part and the tool are attached to the end-effectors of two separate 7-degrees of freedom (DOF) manipulators (R_1 and R_2). Both manipulators are attached to the world frame or ground through fixed joints. There are 14 configuration variables representing the joint angles of the robots.

Figure 4.9 illustrates a different example of ensemble of objects and robots. This example shows a transportation task by a bi-manual mobile manipulator. Even though the transportation problem is different than the buffing problem, Figure 4.10 illustrates how we can capture the tree representation of this ensemble under the same framework. There are three robots, a mobile-base (R_1) and two manipulators (R_2 and R_3). The mobile-base has 3-DOF motion (translation and rotation on the plane) with respect to the world frame. An object (O_1) needs to be transported through a narrow door. The motion constraint on O_1 is defined as a path ($c_1(s)$) for a coordinate

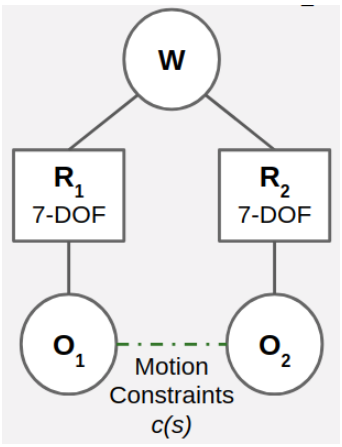


Figure 4.8: Example of tree representation for ensembles of robots and objects illustrated in Figure 4.7. Tree representation eliminates cycles. Solid edges in the tree imply rigid connections. Additional motion constraints are added with the dotted lines. This can be path constraint or zero relative motion.

frame attached to O_1. This path is defined with respect to the static world frame (W). The end-effector of R_2 is rigidly attached to O_1. The end-effector of R_3 is constrained by a rigid transformation with respect to the end-effector of R_2. This constraint generates the motion constraint $c_2(s)$ for R_3's end-effector. This constraint ensures that both R_2 and R_3 will be holding O_1 at the desired locations during the transportation task. There are 17 configuration variables in this example. 14 configuration variables represent the joint angles of R_2 and R_3. And out of the remaining three configuration variables, two represent the position, and one represents the orientation of the mobile-base R_1.

We want to generate trajectories for the robots such that the objects are in motion and pairwise constraints between the objects' motions are satisfied. Therefore, we model the desired relative motion among the objects in terms of the robots' motion. Moreover, the configuration space trajectories should optimize the process or application objective, avoid collisions, satisfy the physical constraints of the robots, and satisfy other process constraints.

We formulate the trajectory generation problem for high-DOF robotic systems as a constrained optimization problem. Given the ensemble as a tree structure ($\mathscr{T}$), the relative motion constraints

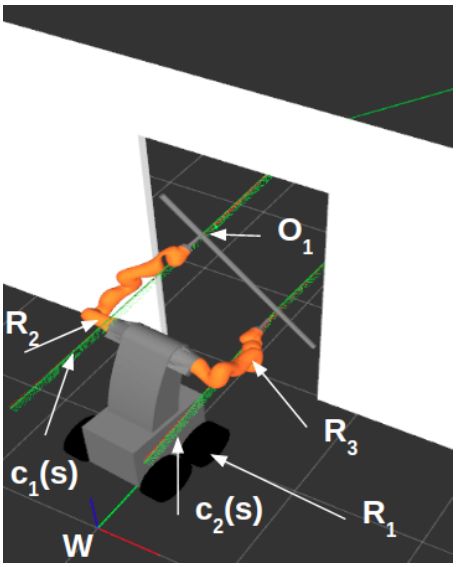


Figure 4.9: Example of an ensemble of robots and objects. The object (O_1) needs to be transported by following a path defined in the workspace. The object is attached to two manipulators (R_1 and R_2). This ensemble creates two path-constraints ($c_1(s)$ and $c_2(s)$).

among the objects as paths ($\mathscr{C}$), and the physical description of the robots ($\mathscr{R}$) and their operational workspace ($\mathscr{W}$), we want to find the time-optimal trajectory of the configuration variables ($\Theta^*(t)$) in a computationally efficient manner such that, for $t_i \leq t \leq t_f$,

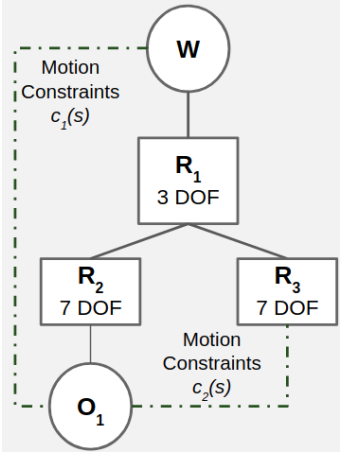


Figure 4.10: Examples of tree representations for ensembles of robots and objects illustrated in Figure 4.9. Tree representation eliminates cycles. Solid edges in the tree imply rigid connections. Additional motion constraints are added with the dotted lines. This can be path constraint or zero relative motion. ($c_1(s)$ and $c_2(s)$ represent path constraints as parametric curves of desired poses. $c_1(s), c_2(s) \subset SE(3)$, s is arc-length parameter.)

$$\Theta^*(t) = \arg\min_{\Theta(t)} \int_{t_i}^{t_f} dt \tag{4.2}$$

$$s.t.$$

$$\underline{\Theta} \leq \Theta(t) \leq \overline{\Theta}, \tag{4.3}$$

$$\underline{\dot{\Theta}} \leq \dot{\Theta}(t) \leq \overline{\dot{\Theta}} \tag{4.4}$$

Relative Motion Constraint among objects,

$$g_{curve}(\mathscr{C}(s(t)), \Theta(t)) = 0 \tag{4.5}$$

$$\text{Constraint on Collision Avoidance}, g_{coll}(\Theta(t)) \leq 0 \tag{4.6}$$

Application Specific Constraints,

$$g_j(\mathscr{C}(s(t)), \Theta(t)) \leq 0, j = 0, 1, \ldots, l \tag{4.7}$$

In this work, we will approximate each of the configuration variable trajectories as a third order B-Spline. Representation of the j^{th} configuration variable, θ_j, will be,

$$\theta_j(s, x^j) = \sum_{i=1}^{N_{cp}} R_{i,k}(s) x_i^j, s \in [0, 1] \tag{4.8}$$

The map between the arc-length parameter (s) and time (t) is the following, $s = 0$ when $t = t_i$ and $s = 1$ when $t = t_f$. N_{cp} is the number of control points for the one dimensional spline curve representing θ_j, x^j is the vector of control points for θ_j, and $R_{i,k}(s)$ are the basis functions parametrized with arc-length parameter s of the workspace curves $\mathscr{C}(s)$.

Given the spline representation, the optimization problem is to find the optimal set of control points (X) for the configuration variables. X is a vector created by vertically stacking the vectors $x^j, \forall j$. For a robotic system with N^d degrees of freedom, the length of the vector X will be $N^d \times N_{cp}$. Details on creating the optimization problem instance is discussed in section 4.3.3.

4.3 Approach

4.3.1 Overview of Approach

Generating path-constrained trajectories is computationally challenging for high-DOF robotic systems. This is due to the large number of highly non-linear constraints present in the relative motion of objects being manipulated. The configuration space trajectories need to be generated as functionals of time for the robots. We convert the trajectory generation problem into a discrete parameter optimization problem. We need to use the right parametric curve representation to approximate these trajectories, determine how to evaluate the objective and constraint function, and appropriately select representative points along the workspace path to evaluate the objective and constraints. Section 4.3.3 discusses the details on creating optimization problem instances. If we initiate the optimization routine with a random seed, then we may not be able to find a feasible solution within a reasonable computation time-bound. We may land on an infeasible

solution, as well as highly sub-optimal solution, owing to the numerous local minima in our class of problem. Section 4.3.4 discusses our method to generate an approximate solution. As discussed in section 4.3.2, solving the optimization problem with all the constraints under consideration may take a large number of iterations. Sometimes it may fail to find a feasible solution as well. Instead, we can solve the problem successively by introducing different subsets of the constraints in different stages. Identifying the right sequence of adding constraints is necessary to improve the computation speed. Section 4.3.5 discusses the details on our approach to solve the problem efficiently. A high-level overview of the three stages in our approach is illustrated in Figure 4.16. Algorithm 2, which makes calls to algorithm 3, 4, and 5, summarizes our method.

4.3.2 Selection of Optimization Strategy

4.3.2.1 Limitation of Single Stage Optimization

Solving the optimization problem in a single stage with all the constraints applied can be computationally intensive. It may not converge to a feasible solution either. In this work, we have investigated the nature of constraints and interaction among them for path-constrained trajectory generation problems.

Let us consider an example problem illustrated in Figure 4.11. This is a synchronous trajectory generation problem for a planar mobile manipulator. This planar robotic system has five degrees of freedom. The mobile-base can translate in x and y direction independently, and the three joints of the manipulator can rotate in the plane. We want to generate trajectories in the five-dimensional configuration space such that the end-effector of the robot follows the given path at the desired velocity. The trajectory should be collision-free. This example problem is a simplified representation of the class of problems we are interested in solving in this work.

We solve this illustrative problem by representing each of the configuration variables with third-order B-spline curve with 4 control points. The goal is to identify these control points. Therefore,

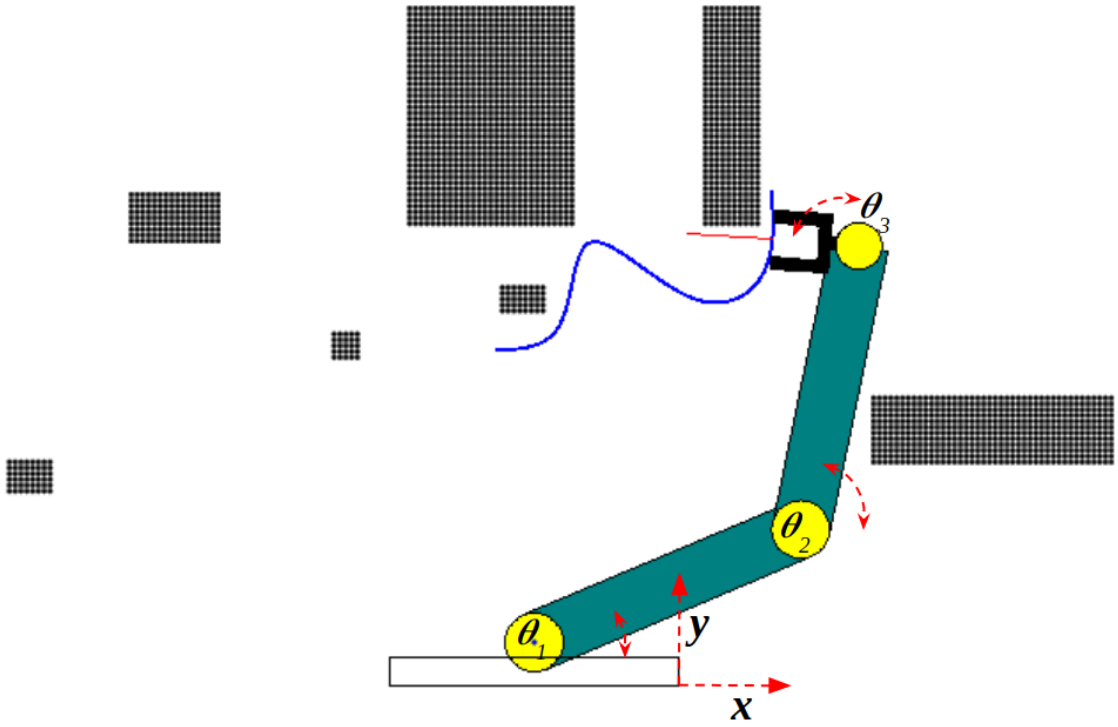


Figure 4.11: An illustrative example with a planar mobile manipulator. A 3 DOF manipulator is mounted on a 2 DOF base that can translate on the xy plane. The end-effector is required to trace the dark-blue path at desired velocity by maintaining an orientation normal to the tangent along the curve. The black rectangular objects are obstacles. A collision-free trajectory in five-dimensional configuration space $(x, y, \theta_1, \theta_2, \theta_3)$ needs to be generated to perform the task.

we have a 20-dimensional parameter space for this problem. We want to find a solution for the optimal set of parameters using optimization.

The objective of the optimization problem is to generate a time-optimal trajectory. We have considered four constraints that should be satisfied along the path. They are - (1) position constraint of the end-effector, (2) orientation constraint of the end-effector, (3) minimum velocity constraint of the end-effector, and (4) collision avoidance.

We sampled the 20-dimensional parameter space to pick a random initial solution. In this work, we will refer to this initial solution as a seed. We passed this seed through a single stage of optimization. The optimization routine failed to find a feasible solution.

We will now use abstract examples to illustrate why single stage optimization can fail to find a feasible solution for problems with multiple constraints. Let us consider the case of n_c different

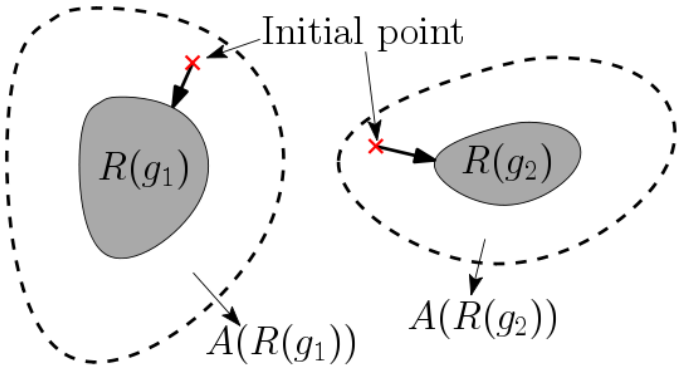


Figure 4.12: Illustration of region of constraint satisfaction and attractor basin.

constraints. A constraint g can have multiple disjoint regions in the parameter space where it is satisfied. Let us consider a region $R(g)$ in the parameter space on which g is satisfied. Also, let us consider a randomly generated initial solution (or seed for the optimization routine). An optimization routine takes the seed and uses gradient-based methods to iteratively move the seed through the parameter space to identify the point where all the constraints are satisfied, and the objective function is optimized. In this problem, at every iteration of the optimization routine, the solution will move through the parameter space when constraint violation function is used to update this solution to reduce the constraint violation through gradient descent. If the initial solution is in the vicinity $R(g)$, then the optimization process will iteratively converge to a solution in $R(g)$. The region in parameter space $A(R(g))$ that attracts initial solutions to $R(g)$ is called the attractor basin for $R(g)$. $R(g) \subseteq A(R(g))$. Figure 4.12 shows an abstract graphical illustration of these definition using two-dimensional parameter space. The probability that a given initial solution will move to a given region $R(g)$ depends on the size of $A(R(g))$.

If a constraint has a large number of disjoint feasible regions, then the basin size associated with each feasible region will tend to be small and associated probabilities of converging to optima will be small. Figure 4.13 illustrates one such abstract example. In this abstract example, constraints g_1 and g_2 have single regions of satisfaction (or feasibility) and attractor basins. However, constraint g_3 has multiple disjoint regions of satisfaction and attractor basins. There is only one

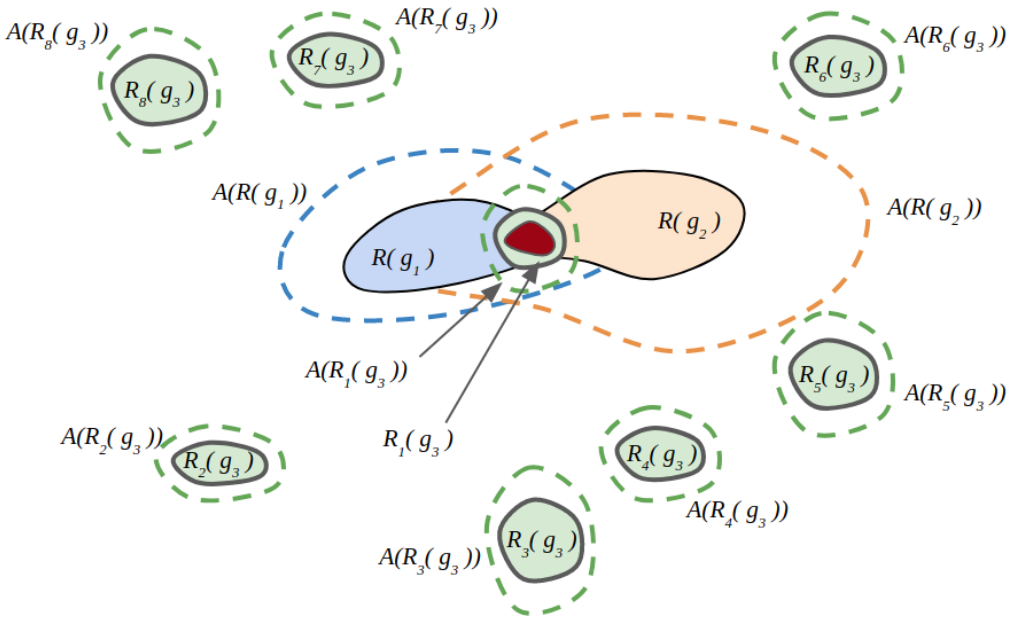


Figure 4.13: Illustration of region of constraint satisfaction and attractor basins for multiple disjoint constraints. Regions with blue, orange, and green boundaries show feasibility regions for each constraint. The region in red represents the feasibility region for all three constraints combined.

attractor basin of g_3 that overlaps with the basins of g_1 and g_2. Therefore, during a single-stage optimization, if the seed moves to any of the attractor basins of g_3 except $A(R_1(g_3))$, then it will struggle to come out of it and move to the common overlapped regions of attraction of g_1, g_2, and g_3 without having a random restart.

Constraint violation functions are used to guide the initial solutions to the feasible regions. A typical constraint violation function behaves well near a feasible region and monotonically increases with respect to the distance from the feasible region boundary. Well-behaved constraint violation function can guide the initial solution to the feasible region. However, when there are multiple feasible regions nearby, the constraint violation function does not behave well away from each of the feasible regions as each feasible region tends to attract the solution towards itself. This leads the constraint violation function to behave erratically and does not provide meaningful guidance. The interaction among different regions reduces the region over which the constraint

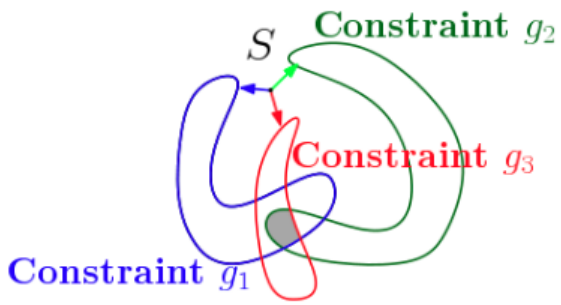


Figure 4.14: Illustration of an abstract example of conflicting constraints at a sampled point in the parameter space.

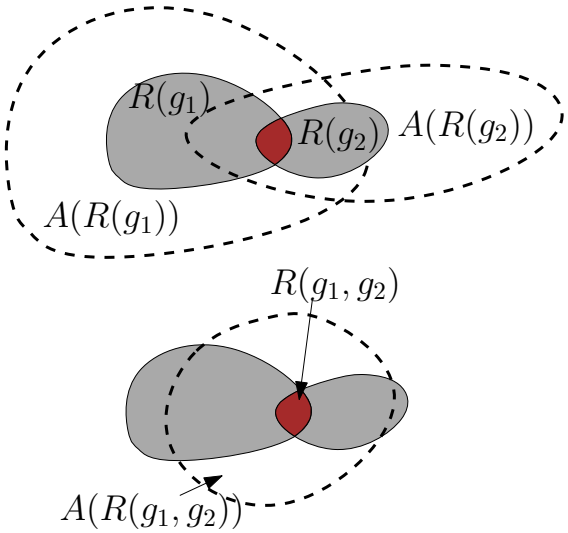


Figure 4.15: Illustration of basin size of independent constraints vs combined constraints.

violation function behaves well. The basin size is strongly correlated to the region over which the constraint violation functions behave well.

Let us consider the effect of applying multiple constraints together. Figure 4.14 illustrates an abstract example. There are three constraints in this abstract example problem. Regions with blue, red, and green boundaries show feasibility regions for each constraint. A grey area denotes the region where all constraints are satisfied. Let, s be a random starting point in the parameter space. All three constraints are violated at s. The goal for the optimization routine is to iteratively move s through the parameter space and bring it inside the grey area where all the constraints are satisfied. Attempt to minimize constraint violations from s leads to different

desired gradient direction for each of the constraints. The combined constraints have a different feasibility region (and associated attractor basin) compared to each of the individual constraints. If constraint violation functions associated with individual constituent constraints are producing conflicting information, then the basin size for the combined constraints can be very small and hence may not be able to produce solutions in a reasonable amount of time.

4.3.2.2 Successive Refinement Strategies for Handling Conflicting Constraints

Let us consider an abstract optimization problem with two constraints. The attractor basins of the constraints may or may not completely overlap with each other. If one of the attractor basins completely overlaps the other one, then the combined attractor basin will behave similarly to the union of the two. If there is a very small overlap between the attractor basin, then the combined attractor basin will behave similarly to the intersection of the two. Let us consider an abstract example shown in Figure 4.15. The attractor basin size for constraint g_1 is large compared to the basin size for the constraints g_1 and g_2 combined. In this case, we will get a solution much faster if we use g_1 first. This will bring the solution within the combined basin of g_1 and g_2 with a high probability. After using g_1, we can use g_1 and g_2 to guide the solution to the region where both constraints are satisfied.

Selecting the right sequence for adding constraint components to the optimization procedure can help to solve this problem efficiently. Our exploratory experiment on a planar robot confirmed this hypothesis. Selecting the right sequence requires us to understand the basin sizes of various constraints and how interactions among constraints can affect the basin sizes. Some constraints will be highly correlated, and hence using them together will not negatively impact the basin sizes. In this work, we have considered numerical analysis to determine the order in which different sets of constraints will be applied for synchronous trajectory generation of multi-robot systems. The number of parameters will be large in this case. Hence estimating basin sizes and interaction

Table 4.1: Ten different sequences of successive refinement considered to study the planar mobile manipulation problem. (t: trajectory execution time, p: position constraint, o: orientation constraint, v: velocity constraint, c: constraint on collision avoidance. Section 4.3.3 discusses the formulation of these constraints in details)

Successive Refinement Sequence #	Constraints Active in Optimization Stage 1	Constraints Active in Optimization Stage 2	Constraints Active in Optimization Stage 3	Constraints Active in Optimization Stage 4	Success Rate	Avg Computation time (s)
1	t+p+o+v+c	N/A	N/A	N/A	0.00	N/A
2	t+p	t+p+o	t+p+o+v	t+p+o+v+c	69.23	101.17
3	t+p+o	t+p+o+v	t+p+o+v+c	N/A	69.23	80.31
4	t+p+o	t+p+o+c	t+p+o+v+c	N/A	61.54	174.85
5	t+p+o	t+p+o+v+c	N/A	N/A	69.23	124.70
6	p+o+c	p+o+c+t+v	N/A	N/A	0.00	N/A
7	p+o+v	p+o+v	t+c	N/A	0.00	N/A
8	p+o	p+o+c	p+o+c+t+v	N/A	69.23	201.74
9	p+o	p+o+v+c	p+o+v+c+t	N/A	69.23	233.18
10	p+c	p+c+o	p+c+o+t	p+c+o+t+v	15.38	250.35

among basin sizes by the exhaustive search will not be feasible. We have used a sampling-based approach to estimate basin sizes and interactions among them.

Table 4.2: Mean and standard deviation of survival rate (%) (for 1000 random initial seeds) in four different sequences of successive refinement on the illustrative example.

Successive Refinement Sequence 1		**Constraints Active in Optimization stage 1**				**Successive Refinement Sequence 3**		**Constraints Active in Optimization stage 1**	**Constraints Active in Optimization stage 2**	**Constraints Active in Optimization stage 3**	
		t+p+o+v+c						**t+p+o**	**t+p+o + v**	**t+p+o+v+c**	
Strict Filtering	Mean	0.00				Strict Filtering	Mean	97.23	68.61	45.50	
	Std. Dev.	0.00					Std. Dev.	1.24	3.21	3.21	
Gross Filtering	Mean	0.00				Gross Filtering	Mean	97.23	70.53	66.13	
	Std. Dev.	0.00					Std. Dev.	1.24	3.15	2.97	
Successive Refinement Sequence 2		**Constraints Active in Optimization stage 1**	**Constraints Active in Optimization stage 2**	**Constraints Active in Optimization stage 3**	**Constraints Active in Optimization stage 4**	**Successive Refinement Sequence 10**		**Constraints Active in Optimization stage 1**	**Constraints Active in Optimization stage 2**	**Constraints Active in Optimization stage 3**	**Constraints Active in Optimization stage 4**
		t+p	**t+p+o**	**t+p+o+v**	**t+p+o+v+c**			**p+c**	**p+c+o**	**p+c+o+t**	**p+c+o+t+v**
Strict Filtering	Mean	99.85	39.66	21.45	11.21	Strict Filtering	Mean	30.61	2.57	0.25	0.02
	Std. Dev.	0.27	3.58	2.86	2.31		Std. Dev.	2.99	1.09	0.35	0.09
Gross Filtering	Mean	99.85	39.71	53.64	52.03	Gross Filtering	Mean	30.61	8.16	8.45	1.54
	Std. Dev.	0.27	3.55	2.86	2.31		Std. Dev.	2.99	1.94	1.79	0.89

For solving the optimization problem using successive refinement, we have conducted exploratory experiments and performed numerical analysis on the illustrative example (4.11) to identify a promising sequence. We sampled the 20-dimensional parameter space to pick 1000 random initial solution (or seed) and passed them through different sequences of the successive refinement for multi-stage optimization. The 10 different sequences we considered for successive refinement are listed in Table 4.1. It is important to note that orientation, velocity, and collision constraint were never selected to be applied without position constraint. Applying collision constraint without position constraint is equivalent of generating a random collision-free initial solution, which may not be remotely close to the attractor basin of the combined constraints. A similar argument is also applicable for applying velocity and orientation constraint without position constraint. This is because the desired orientation and velocity might be achievable at many different regions in the parameter space, which can be far away from the attractor basin of the position constraint. Considering position constraint at the earlier stage narrows down the volume in the parameter space where the combined feasible attractor basin is located.

The success rate column in Table 4.1 indicates what fraction of the initial seeds satisfied all the constraints at the end of the successive refinement. From these initial results, we considered four different sequences to closely study the behavior of the attraction region for each constraint. We picked sequence 2 and 3 for their high success rate and low computation time. We picked sequence 1 and 10 to keep them as benchmarks. We picked sequence 1 as all the constraints were applied together for a single-stage optimization in this case, even though the success rate was 0. We picked sequence 10 for its average performance. We ignored sequence 4,5,8, and 9 for further study as their computation time is higher even though their success rate is comparable to sequence 2 and 3. We can see that collision constraint was applied in multiple stages of successive refinement for sequence 4, 8, and 9. Which, in turn, led to higher computation time. Sequence 6 and 7 were ignored as they never produced a feasible solution.

For the thorough numerical analysis, we considered 1000 batches of samples and passed them through the four different sequences of successive refinement. We chose the population size for each batch to be 200. This was chosen as the mean, and standard deviation of success rate was consistent beyond this population size. We logged how many samples survived or satisfied respective constraints after each stage of refinement. We considered two different filtering approach. In *Strict Filtering*, only the samples that satisfied constraints in the last stage were allowed to be passed to the next stage of refinement. In *Gross Filtering*, all the samples were passed through the complete sequence of refinement. The results are summarized Table 4.2. The column of each stage tells us the probability of success for the sub-sequences until that stage. The column of the last stage of strict filtering tells us the probability of success for the complete sequences. We can see that success rate or probability of success of sequence 3 is highest. These numerical results can be explained with the following hypotheses.

The position constraint is well behaved and has a single attractor basin in the parameter space. There can be sub-regions of level sets inside this region. Orientation constraint, on the other hand, can have multiple attractor basins and large level fields. This is because the same end-effector orientation can be achieved by keeping the end-effector in different positions. The size of the set of configurations that satisfy position constraint is much smaller compared to the set of configurations that satisfy orientation constraint. Therefore, the intersection of position and orientation constraints, i.e., the attractor region of position and orientation constraint together, is located close to the position constraint's attractor region in the parameter space. Also, the combined landscape of position and orientation constraints will be similar to the landscape of position constraint alone.

The landscape of velocity constraint will have scattered attractor regions all over the parameter space. This is because many different configurations of the robot can satisfy the same velocity constraint for the end-effector. For this example problem, the collision constraint will also have small attractor regions scattered over the parameter space. This is because the obstacles are

scattered in the robot's workspace, and there are very few locations in the parameter space that will produce a collision-free trajectory for the entire robot motion.

From the previous two paragraphs, we can understand that applying collision or velocity constraint in the earlier stages of successive refinement may guide the initial seed to an attractor region which might become infeasible in the next stage. On the other hand, applying position and orientation constraints first will bring the seed to an attractor region that will guide it to the basin of the next combination of constraints.

The numerical analysis of the example planar-robot problem shows us that multi-stage optimization with successive refinement can be a promising direction to solve the path-constrained trajectory generation problems. We used our understanding to design a successive refinement strategy for solving this class of problems efficiently. Moreover, our approach adaptively generates an approximate solution in the beginning to improve the success rate of the successive refinement stages.

Algorithm 2 TrajPlan Algorithm($\mathscr{C}, \mathscr{R}, \mathscr{W}$)

$Flag \leftarrow True$
$\delta, X_0, N_{seg}, N_{cp}, k \leftarrow GetSplineSeed(\mathscr{C}, \mathscr{R}, \mathscr{W})$
$N_{cp}^{nominal} \leftarrow N_{cp}$
if $N_{seg} == 1$ **then**
  $Flag, \Theta \leftarrow GetSingleSplineTraj(X_0, \delta, N_{cp}, k, \mathscr{C}, \mathscr{R}, \mathscr{W}, \{\}, \{\})$
end if
if $N_{seg} > 1$ OR $!Flag$ **then**
  $Flag, \Theta \leftarrow GetMultiSplineTraj(X_0, \delta, N_{cp}, k, N_{seg}, \mathscr{C}, \mathscr{R}, \mathscr{W})$
end if
$iter \leftarrow 0, \delta \leftarrow \delta^{MIN}$
while $!Flag$ AND $iter < iter^{MAX}$ **do**
  $N_{cp} \leftarrow N_{cp} + 1$
  if $N_{cp} > N_{cp}^{MAX}$ **then**
    $N_{cp} \leftarrow N_{cp}^{nominal}$
    $N_{seg} \leftarrow N_{seg} + 1$
  end if
  $Flag, \Theta \leftarrow GetMultiSplineTraj(X_0, \delta, N_{cp}, k, N_{seg}, \mathscr{C}, \mathscr{R}, \mathscr{W})$
  $iter \leftarrow iter + 1$
end while
Return Θ

Algorithm 3 GetSplineSeed($\mathscr{C}, \mathscr{R}, \mathscr{W}$)

$k \leftarrow 3$ {order of spline}
$\delta \leftarrow EstimateSamplingResolution(\mathscr{C})$
$\mathbb{P} \leftarrow SamplePointsOnWorkspacePath(\delta, \mathscr{C})$
$\Theta_{ik} \leftarrow FindIKSolution(\mathbb{P}, \mathscr{R}, \mathscr{W})$
$N_{seg} \leftarrow EstimateRequiredSplineSegments(\Theta_{ik})$
$N_{cp} \leftarrow EstimateRequiredControlPoints(\Theta_{ik})$
$error \leftarrow \infty; N_{cp}^{nominal} \leftarrow N_{cp}, iter \leftarrow 0$
while $error > \epsilon$ AND $iter < iter^{MAX}$ **do**
    $X_0, error \leftarrow FitSplineThroughIKSolutions(\Theta_{ik}, N_{seg}, N_{cp})$
    $N_{cp} \leftarrow N_{cp} + 1$
    if $N_{cp} > N_{cp}^{MAX}$ **then**
        $N_{cp} \leftarrow N_{cp}^{nominal}$
        $N_{seg} \leftarrow N_{seg} + 1$
    end if
    $iter \leftarrow iter + 1$
end while
if $error > \epsilon$ **then**
    $x_0 \leftarrow GetRandomSplineSeed(\mathscr{C}, \mathscr{R}, \mathscr{W})$
    $N_{seg} \leftarrow 1$
    $N_{cp} \leftarrow N_{cp}^{MIN}$
    $X_0 \leftarrow \{x_0\}$
end if
Return $\delta, X_0, N_{seg}, N_{cp}, k$

Algorithm 4 GetSingleSplineTraj($X_0, \delta, N_{cp}, k, \mathscr{C}, \mathscr{R}, \mathscr{W}, \Theta_{pre}, \Theta_{post}$)

$X \leftarrow_X ObjectiveFunction(X_0, \delta, N_{cp}, k, \mathscr{C}, \mathscr{R}, \mathscr{W}, \Theta_{pre}, \Theta_{post})$
$\eta \leftarrow GetErrorComponents(X_0, \delta, N_{cp}, k, \mathscr{C}, \mathscr{R}, \mathscr{W}, \Theta_{pre}, \Theta_{post})$
$Flag \leftarrow AreConstraintsViolated(\eta)$
$\Theta(s) \leftarrow GetParametricTrajectory(X)$
Return $Flag, \Theta(s)$

Algorithm 5 GetMultiSplineTraj($X_0, \delta, N_{cp}, k, N_{seg}, \mathscr{C}, \mathscr{R}, \mathscr{W}$)

$\Theta \leftarrow \{\}$, $Flag \leftarrow False$, $q \leftarrow 1$,$\Theta^q_{pre} \leftarrow \{\}$, $\Theta^q_{post} \leftarrow \{\}$
for $q <= N_{seg}$ **do**
  $X^q_0 \leftarrow GetSeedForSegment(X_0, q)$
  $\mathscr{C}^q \leftarrow GetCurveForSegment(\mathscr{C}, q)$
  $Flag, \Theta^q, X^q \leftarrow GetSingleSplineTraj(X^q_0, \delta, N_{cp}, k, \mathscr{C}^q, \mathscr{R}, \mathscr{W}, \Theta^q_{pre}, \Theta^q_{post})$
  if $!Flag$ **then**
    Return $Flag, \Theta$
  end if
  $\Theta \leftarrow \Theta \cup \Theta^q$
  $q \leftarrow q + 1$
end for
$E \leftarrow \{(1,2),(2,3),\ldots,(N_{seg}-1, N_{seg})\}$
$A \leftarrow \{\}$
for $(q, q+1) \in E$ **do**
  $A \leftarrow A \cup DetermineAffinityScore(\Theta(1, X^q), \Theta(0, X^{q+1}))$
end for
Sort (MAX to MIN) entries of E w.r.t. Affinity Score value (corresponding values in A)
$iter \leftarrow 0$
while $|E| > 0$ **do**
  $iter \leftarrow iter + 1$
  $(q, q+1) \leftarrow E.pop()$
  $\Theta^q_{pre} \leftarrow \Theta(1, X^{q-1})$
  $\Theta^q_{post} \leftarrow \Theta(1, X^{q+1})$
  $Flag, \Theta^q, X^q \leftarrow GetSingleSplineTraj(X^q_0, \delta, N_{cp}, k, \mathscr{C}^q, \mathscr{R}, \mathscr{W}, \Theta^q_{pre}, \Theta^q_{post})$
  if $!Flag$ **then**
    $E.push((q, q+1), end)$
    Continue
  end if
  Update Θ with Θ^q
  for $(q, q+1) \in E$ **do**
    $A \leftarrow A \cup DetermineAffinityScore(\Theta(1, X^q), \Theta(0, X^{q+1}))$
  end for
  if $iter > N_{seg} \times (N_{seg}+1)/2$ **then**
    $Flag \leftarrow False$
    Break
  end if
end while
Return $Flag, \Theta$

4.3.3 Creating Optimization Problem Instance

We considered polynomial [174], B-Spline [60], and NURBS [69] as candidates for parametric curve approximation of the configuration variables. Polynomial models may overfit or underfit a curve due to their poor interpolation properties. NURBS, on the other hand, can approximate complex trajectories. B-Spline has a limitation in approximating curves belonging to the conics family. However, uniform weight vector across the basis functions leads to significantly less number of optimization variables for B-Spline compared to NURBS. A single B-Spline with high order can approximate a complex trajectory at the expense of overfitting. A single B-spline with low order may need a large number of control points to approximate the same trajectory, leading to a higher number of optimization variables. For computational efficiency, we can break the underlying trajectory into multiple segments, approximate each segment with a low order B-Spline with a relatively small number of control points, and enforce pair-wise continuity among the segments. Figure 4.17 illustrates an abstract example of approximating a 2D curve with different parametric representations. Figure 4.17(a) shows the actual curve we are trying to approximate. Figure 4.17(b) shows a bad approximation with a single spline with low number of control points. Figure 4.17(c) shows a good approximation with large number of control points. Figure 4.17(d) shows a multi-segment spline approximation where each segment needs low number of control points.

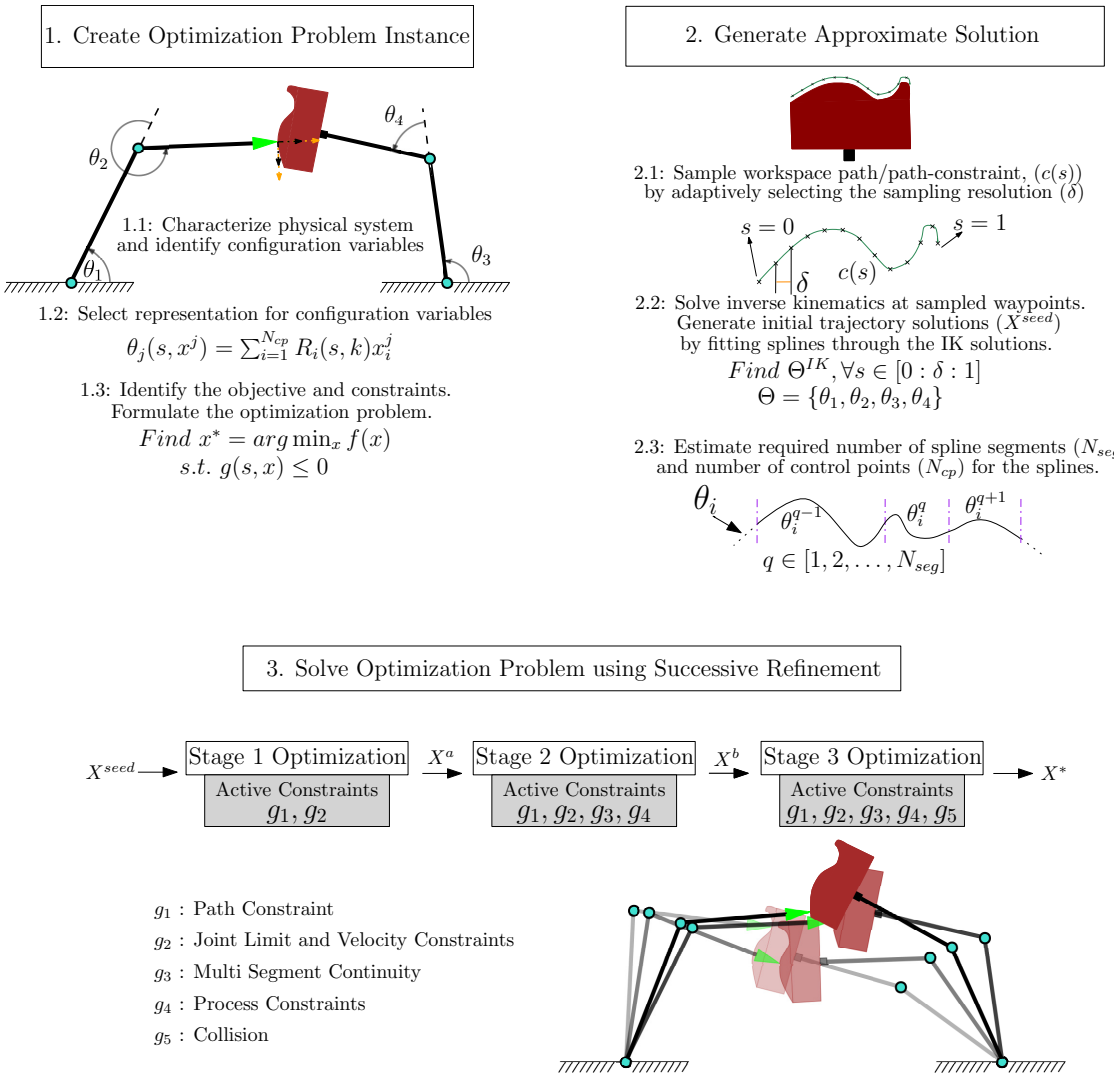


Figure 4.16: The three stages in our approach for solving the path-constrained trajectory generation problem. The figures illustrate a 2D representative example. The task is to move the green tool (O_1) along the path ($c(s)$) defined on the surface of the red part (O_2). The tool needs to maintain a specified position and orientation during this motion. The tool is attached to a 2-DOF planar-manipulator, and the part is attached to another manipulator. Our goal is to generate the trajectories for the 4 joints ($\theta_1 - \theta_4$) such that the relative motion between O_1 and O_2 is completed as fast as possible. During the first stage of our approach, we represent the configuration variables (joints) as splines and formulate our problem as a discrete parameter optimization problem. We want to find x, which is a vector of the control points for the splines. During the second stage, we sample the workspace path ($c(s)$) and solve inverse kinematics at the sampled points. We estimate the number of control points and the number of spline segments required to represent the configuration variables and generate an approximate solution. At the third stage, we successively solve the optimization problem using the estimates.

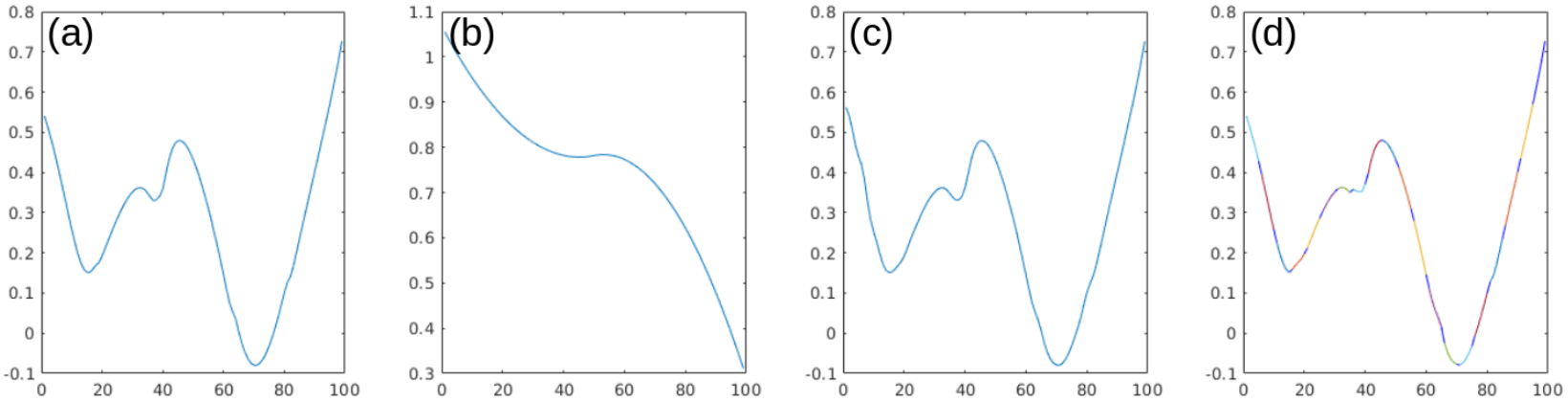


Figure 4.17: (a) Ideal Trajectory, (b) Trajectory approximated with a single third order B-Spline with 4 control points, (c) Trajectory approximated with a single third order B-Spline with 46 control points, (d) Trajectory approximated with Multi-Segment third order B-Splines (20 segments), each segment has 4 control points.

As described in section 6.2, in this work, we have adopted third-order B-Spline in our approach to approximate the trajectories of the configuration variables (Equation 4.8). We have considered multi-segment B-spline approximation for a trajectory when a single B-spline fails to approximate it. The goal for the discrete parameter optimization problem is to identify the spline control points.

We consider a sequence of spline segments to approximate a trajectory when a single spline segment can not do it. If N_{seg} number of spline segments are needed to approximate θ_j, then we will be considering intermediate arc-length parameters for each segments. For the q^{th} spline segment, we represent the intermediate arc-length parameter as s^q. The map between s and s^q will be- $s^q = 0$ when $s = (q-1)/N_{seg}$ and $s_q = 1$, when $s = q/N_{seg}$. For the q^{th}segment of θ_j, it will be represented using x^q as the control points, $\theta_j(s^q, x^q)$. Therefore, for N_{seg} segments, we will need to find $X_q, q = 1, \ldots, N_{seg}$ by solving at least N_{seg} number of optimization problems. We consider first order continuity between two consecutive spline segments for the test cases discussed in section 6.4.

We present the objective and constraint functions to create an optimization instance in equation (4.9) and (4.10). The goal is to find the optimal spline control points (X^*) for minimum-time trajectory. We consider the position and velocity limits for the configuration variables as independent constraints. We add all the other constraints (path-constraint, continuity among spline segments, collision, physics, and process) as a weighted sum in the objective function. We also include the trajectory execution time in the objective function. In equation (4.9), $\mathcal{S}$ is a set of sampled values of arc-length parameter $s \in [0, 1]$ where we will be evaluating constraint violations.

$$X^* = \arg\min_X \sum_{s \in \mathcal{S}} (\ w_1.ExecutionTime(\Theta(s,X))$$
$$+w_2.PoseError(\Theta(s,X))$$
$$+w_3.MultiSegmentContinuityError(\Theta(s,X)))$$
$$+w_4.ProcessConstraintsViolation(\Theta(s,X))$$
$$+w_5.CollisionScore(\Theta(s,X)\ ) \tag{4.9}$$
$$\underline{\Theta} \leq \Theta(s,X) \leq \overline{\Theta},\ \underline{\dot{\Theta}} \leq \dot{\Theta}(s,X) \leq \overline{\dot{\Theta}} \tag{4.10}$$

Pose Error: We define the pose error as a weighted sum of position error and orientation error. Some applications may require zero-error in position and orientation. Some applications may allow tolerance in position or orientation. Depending on the application, we define position error in one of the two following ways- (a) absolute position error (equation 4.12) and (b) position error with tolerance (equation 4.13). In our method, we have considered three different representations for orientation error to give the flexibility to different applications. They are (a) complete orientation match (equation 4.14), (b) orientation match along one axis, free rotation is allowed about this axis (equation 4.15), and (c) orientation match along one axis with tolerance (equation 4.16). Position and orientation of the coordinate frames of interest are computed using forward kinematics for the tree ($\mathcal{T}$) of robots and objects. Figure 4.18 illustrates examples of the different position and orientation match requirements.

In equation (4.12 - 4.16), the subscript $_{O_i}$ represents the coordinate frame attached to the point-of-interest on object O_i. The subscript $_t$ represents the target frame on the workspace path. R is the rotation matrix. v can be either $x, y, or\ z$ axis of the desired coordinate frame, about which a margin of tolerance for rotation is available. β and ψ represent the tolerance margin in position and orientation, respectively. Position and orientation of the frame on O_i are computed

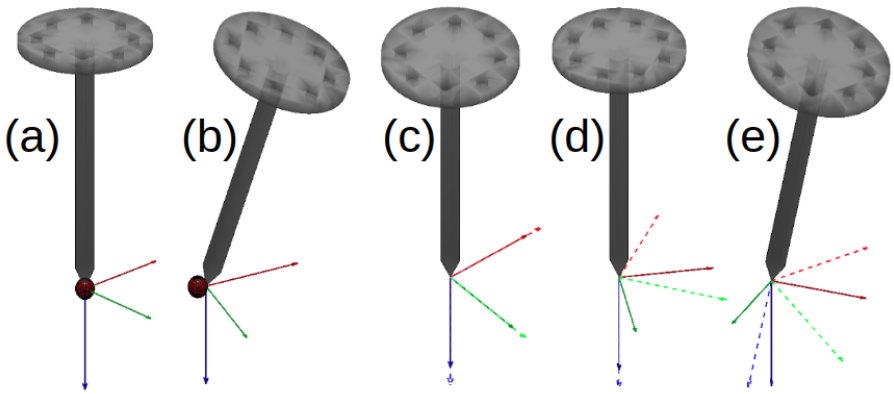


Figure 4.18: Examples of different position and orientation match requirements. In (a) and (b), the center of the red ball is the target position for the tool-tip. The radius of the ball represents tolerance in position. In (c), (d), and (e) the coordinate frame with solid lines represent the target frame. The coordinate frame with the dashed lines represents the coordinate frame attached to the tool-tip. Red, Green, and Blue represents x, y, and z-axis respectively.
(a) Absolute position match. The tool-tip aligns with the center of the ball. (b) Position match with tolerance. The tool-tip is contained in the ball. (c) Complete orientation match. The coordinate frame at the tool-tip matches completely with the target coordinate frame. (d) Orientation match along one axis. The z-axes of the coordinate frames align. (e) Orientation match along one axis with tolerance. The angle between the z-axes of the coordinate frames are below a specified tolerance.

using forward kinematics (FK) of the i^{th} serial-link manipulator. We designed the position and orientation error functions with tolerance as decaying exponents instead of conditional statements so that the optimization routine can estimate the gradients.

$$PoseError = w_1 PositionError + w_2 OrientationError \tag{4.11}$$

$$\|(x_{O_i}, y_{O_i}, z_{O_i}) - (x_t, y_t, z_t)\|_2 \tag{4.12}$$

$$exp^{-\gamma(\beta - \|(x_{O_i}, y_{O_i}, z_{O_i}) - (x_t, y_t, z_t)\|_2)} \tag{4.13}$$

$$\left\|I - R_{O_i}^T R_t\right\|_2 \tag{4.14}$$

$$1 - v_{O_i}^T v_t \tag{4.15}$$

$$exp^{-(\psi - |v_{O_i}^T v_t|)} \tag{4.16}$$

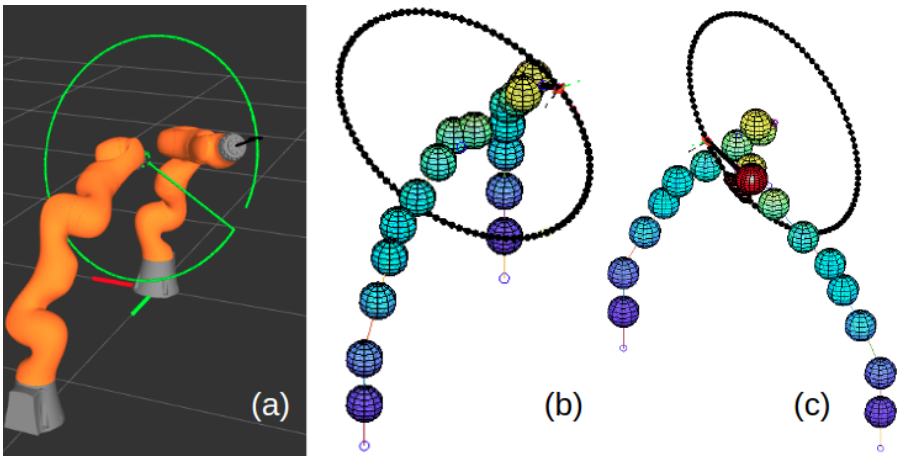


Figure 4.19: Illustration of spherical representations of robots, tool, and part for collision detection. (a) CAD representation, (b) Spherical representation, (c) An example of collision detection using the spherical representations. The inflated red spheres are in collision.

Collision Score: We need to generate collision-free trajectories for the robots. There can be three kinds of collision- robot-robot collision, robot's self-collision, and robot-object collision. The collision evaluation function will be called for every sampled waypoint of the path-constraint during each iteration of the optimization routine. Therefore a fast collision detection function is required. We can consider one of the two following methods for collision detection- (a) Mesh-to-Mesh collision detection function that returns signed penetration depth, (b) Approximate collision detection using a spherical representation of robots, parts, and obstacles. Mesh-to-Mesh collision detection [175] will produce accurate collision score at the expense of high computation time. On the other hand, approximate collision detection using spherical representation can be computationally efficient. In our approach, we represented the rigid bodies as a collection of spheres and evaluated collision by computing penetration depth of the sphere pairs for fast collision detection [176]. Figure 4.19 illustrates an example of collision detection using the spherical representation. The collision score using spherical representation can closely approximate the mesh-based collision detection if the spheres are properly designed and placed to represent the rigid bodies. The formulation for approximate collision score using spherical representation in our approach is given in equation (4.17). (c_i, c_j) and (r_i, r_j) represents the coordinates of the center and radii of the

two spheres under consideration respectively. The pair-wise collision check can be omitted for the spheres belonging to the same link and the spheres physically adjacent to each other across a joint during self-collision check for the robots.

$$exp(-\sum_{i,j,i\neq j} \|c_i - c_j\|_2 - (r_i + r_j)) \tag{4.17}$$

Trajectory Execution time: We have considered a surrogate function to estimate the trajectory execution time given in equation (4.18). The idea is that we will be moving the configuration variable with maximum displacement between two waypoints at its maximum velocity and regulate velocities of other configuration variables such that they all complete motion at the same time. Therefore, equation (4.18) will govern the trajectory execution time.

$$\sum_{i,i\in\{1,2,\ldots,|\mathcal{S}|-1\}} \|\Theta(\mathcal{S}[i]) - \Theta(\mathcal{S}[i+1])\|_\infty / \dot{\theta}_{max} \tag{4.18}$$

Process or Application Specific Constraint Violation: The process constraints can be minimum force requirement at the end-effector, joint-torque constraint, non-holonomic motion constraint for a mobile base, special motion requirements, etc. Below are two examples of process constraints that we considered in our experiments.

Most mobile robot platforms are non-holonomic by design. We can compute the score for Non-holonomic constraint violation for mobile base using equation (4.19).

$$|\dot{x}cos(\phi) - \dot{y}sin(\phi)| \tag{4.19}$$

Some applications may require cyclic motions for the robots. We can compute the error of cyclic-motion constraint using equation (4.20).

$$\|\Theta(0) - \Theta(1)\|_2 \tag{4.20}$$

Multi segment continuity error: Our method generates a trajectory that is combination of multiple splines if the given workspace path can not be traced with a single spline. In such scenarios, we want first order continuity among the spline segments. The error of First order continuity among spline segments constraint can be computed using equation (4.21).

$$\sum_{i=2}^{N_{seg}-1} \left\|\Theta^{i-1}(1) - \Theta^{i}(0)\right\|_2 + \left\|\Theta^{i}(1) - \Theta^{i+1}(0)\right\|_2 \tag{4.21}$$

Similarly, we can consider second and third-order continuity between each pair of segments if that is required by the process or application.

4.3.4 Generating Approximate Solution

Determining Resolution for Sampling Points on Workspace Path: It is not feasible to evaluate the functions in equations (4.9, 4.10) at every point on the continuous workspace paths or path-constraints ($\mathscr{C}$). Sampling the workspace paths at a coarse resolution and evaluating the objective and constraint functions at the sampled points will lead to a poor approximation. A sampling at very fine resolution will lead to higher computation time. We analyze the workspace path-constraints to estimate a feasible sampling resolution that can sufficiently capture the variations in the workspace paths. We determine the curvature ($\mathscr{K}_i$) of the workspace paths ($c_i \in \mathscr{C}$) and take their derivatives ($\dot{\mathscr{K}}_i$). We then subtract the linear trend (best straight-line fit) from $\dot{\mathscr{K}}_i$. Let this augmented signal be h and let us define ζ as

$$\zeta = (max(h) - min(h))/4 \tag{4.22}$$

We find the peaks in $\dot{\mathscr{K}}_i$ such that each peak has a vertical ascend or descend greater than ζ on both directions without encountering another larger peak or the boundaries of the path-constraint. Let the number of peaks be N_{peaks}. We then select an arc-length sampling resolution $\delta \propto 1/N_{peaks}$.

Estimating number of spline segments (N_{seg}) required for the path-constrained trajectory: We numerically find inverse kinematics (IK) solutions at the sampled points along the workspace paths. We add collision avoidance, physical constraints, and process constraints as constraints to the numerical optimization routine to find feasible solutions. We estimate the number of spline segments required to represent the complete trajectory by analyzing the IK solutions. Let, $\Theta^{IK} = \{\theta_1^{IK}, \theta_2^{IK}, \dots \theta_{N_{DOF}}^{IK}\}$ represent the IK solutions ($\theta_j^{IK} = \{\theta_j^{IK}(s) : s \in \mathcal{S}\}$). N_{DOF} is the total degrees of freedom in the robotic system. This is also the dimension of the configuration space. We remove the linear trend from all $\theta_j^{IK}, j \in \{1, \dots, N_{DOF}\}$ and identify the number of peaks in it. Each θ_j^{IK} may have a different number of peaks present in it. We take the mode of the number of peaks present in all the θ_j^{IK} and consider this number as the initial estimate on the number of spline segments.

Estimating number of control points (N_{cp}) required for spline segments: For estimating the number of control points, we analyze $\theta_j^{IK}, \dot{\theta}_j^{IK}, \ddot{\theta}_j^{IK}, \dddot{\theta}_j^{IK}$, $j \in \{1, \dots, N_{DOF}\}$, remove the linear trend from them, and estimate the number of peaks. For each configuration variable (θ_j^{IK}), we take,

$$N_{cp} = \max_j(mode(peaks(\theta_j^{IK}), peaks(\dot{\theta}_j^{IK}), \\ peaks(\ddot{\theta}_j^{IK}), peaks(\dddot{\theta}_j^{IK}))), \tag{4.23}$$

as the initial estimate on the number of control points needed to represent the trajectories as splines.

Generating seed control points (approximate solution) by fitting spline through the IK-solutions: Using the estimated N_{cp} and N_{seg}, we fit splines for each configuration variable through the IK solutions (θ_j^{IK}) by solving the following optimization problem,

$$x_j^0 =_x \sum \left\| \theta_j(s,x) - \theta_j^{IK}(s) \right\| \tag{4.24}$$

$$s \in \mathcal{S}, X^0 = \{x_1^0, x_2^0, \ldots, x_{N_{DOF}}^0\} \tag{4.25}$$

We use interior point method [177] for solving this optimization problem. We then use X^0 as an approximate solution or seed control points in the third stage of our approach (section 4.3.5). For multiple spline segments, we consider the continuity among segments (equation 4.26, 4.27) while solving the optimization problems. If the seed generation fails, then we consider a random seed for the final optimization stage (algorithm 3).

$$\theta_j(1, x^{q-1}) = \theta_j(0, x^q) \tag{4.26}$$

$$\theta_j(1, x^q) = \theta_j(0, x^{q+1}) \tag{4.27}$$

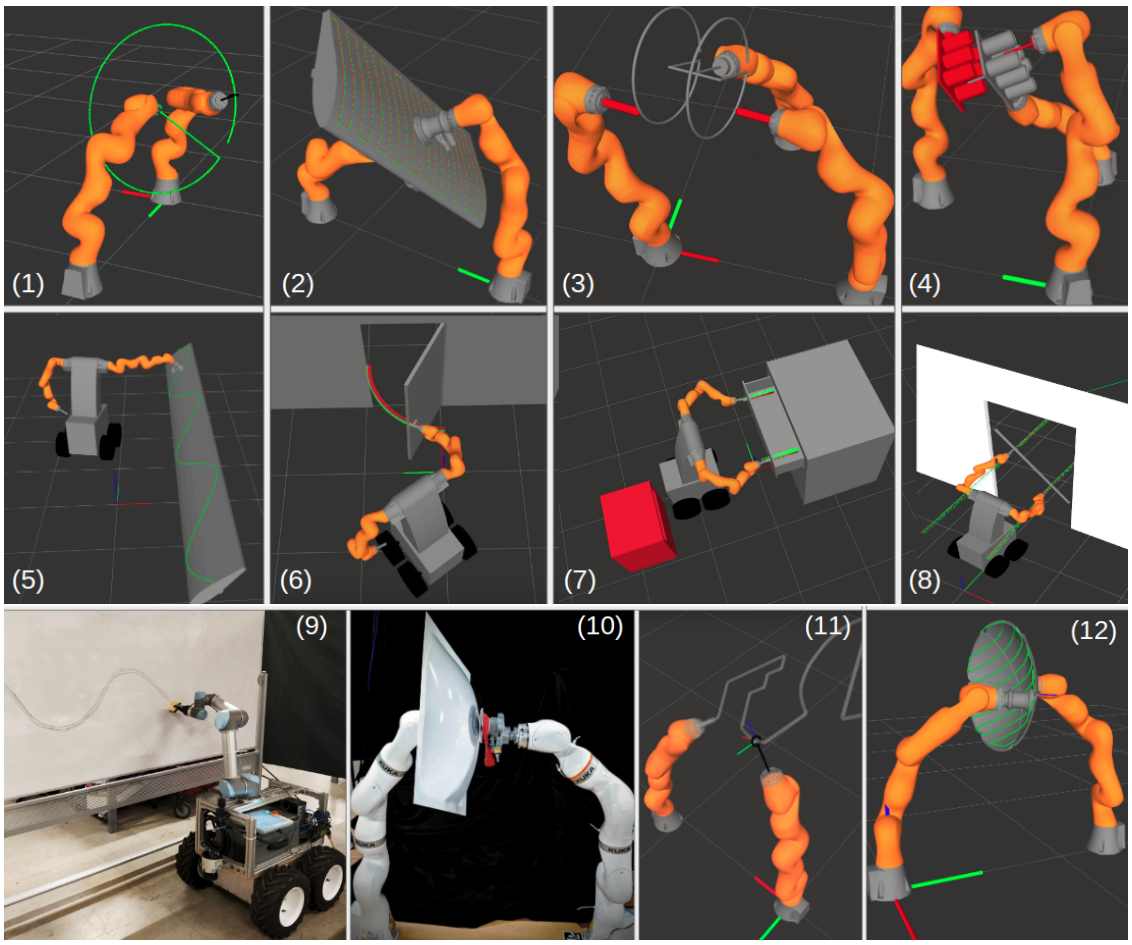


Figure 4.20: Snapshots from the simulation and physical test cases

Table 4.3: Comparison of solution quality.

Test Case	Average Position Error at sampled waypoints (m)			Average Orientation Error at sampled waypoints			Collision Score (Accumulated penetration depth (m) throughout the trajectory)		
	Linear Fit through IK Solutions	Spline Fit through IK Solutions	Spline fit through Successive Refinement	Linear Fit through IK Solutions	Spline Fit through IK Solutions	Spline fit through Successive Refinement	Linear Fit through IK Solutions	Spline Fit through IK Solutions	Spline fit through Successive Refinement
1	7.51E-03	9.01E-02	5.32E-04	8.51E-04	1.88E-02	1.53E-05	7.05E-01	1.29E+00	0.00E+00
2	6.04E-01	5.41E-01	1.40E-03	3.82E-02	2.21E-02	4.16E-03	0.00E+00	0.00E+00	0.00E+00
3	1.48E-02	7.88E-02	1.91E-03	8.51E-04	9.38E-04	2.92E-04	0.00E+00	0.00E+00	0.00E+00
4	1.15E-02	3.48E-02	1.50E-03	6.66E-04	6.72E-04	7.25E-04	1.83E-03	1.82E-03	0.00E+00
5	2.09E+00	2.06E+00	1.99E-03	1.28E-02	1.30E-02	3.12E-04	0.00E+00	0.00E+00	0.00E+00
6	1.09E-03	3.76E-03	9.92E-04	1.65E-03	1.71E-03	1.11E-03	9.67E-01	9.67E-01	0.00E+00
7	1.06E-03	2.42E-03	8.43E-05	1.62E-03	1.62E-03	1.65E-03	0.00E+00	0.00E+00	0.00E+00
8	3.10E-02	6.56E-01	1.70E-03	3.00E-03	1.60E-02	6.45E-03	2.25E-04	2.65E-04	0.00E+00
9	1.16E-01	1.33E+00	1.85E-03	1.82E-02	8.01E-03	2.14E-04	1.75E-01	1.07E+00	0.00E+00
10	8.24E-01	1.67E+00	2.46E-04	3.26E-02	1.19E-02	1.75E-03	9.07E-01	3.99E-01	0.00E+00
11	9.59E-02	1.14E+00	5.11E-04	2.91E-02	1.63E-02	2.29E-03	4.64E-01	2.78E-01	0.00E+00
12	3.05E-01	2.86E-01	1.82E-03	3.12E-02	2.73E-02	2.52E-03	1.20E-01	1.31E-01	0.00E+00

Table 4.4: Properties and parametric representations of the test cases

Test Case	Workspace Path Length (m)	DOF	Sampling Resolution of Arc-Length Parameter for Workspace Path	Number of Spline Segments	Number of Control Points for Splines	Trajectory Execution Time (s)
1	2.83	14	0.1	1	15	4.56
2	11.91	14	0.01	10	19	39.19
3	1.26	21	0.1	1	15	2.10
4	0.89	21	0.05	2	19	2.41
5	6.97	10	0.02	3	11	11.22
6	1.44	10	0.05	1	22	5.37
7	0.50	17	0.1	1	13	0.95
8	7.01	17	0.02	1	17	11.53
9	1.65	9	0.02	2	10	8.97
10	3.71	14	0.01	3	14	5.6
11	2.74	14	0.01	6	8	30.62
12	6.06	14	0.01	6	19	29.37

Table 4.5: Computation time taken by different stages and number of calls to forward kinematics function for the test cases.

Test Case	Computation Time (s)						Number of FK Calls		
	Estimating Sampling Resolution	Solving Inverse Kinematics	Estimating Number of Segments and Number of Control Points	Fitting Spline through IK Solutions	Generating Splines through Hierarchical Optimization	Total Computation Time	Number of FK Calls for Solving IK	Number of FK Calls during Hierarchical Optimization	Total FK Calls
1	1.02	8.02	0.63	4.77	0.15	14.60	26323	1806	28129
2	0.58	27.95	1.28	6.38	23.45	59.63	130879	697992	828871
3	0.57	8.92	0.63	8.03	0.32	18.46	28444	2653	31097
4	0.06	15.72	0.67	21.67	0.63	38.76	52666	9383	62049
5	0.64	19.68	0.48	0.30	56.39	77.48	78412	4141673	4220085
6	0.60	12.83	0.51	12.34	13.86	40.14	30436	249854	280290
7	0.05	9.65	0.33	8.01	11.81	29.84	22887	88156	111043
8	0.10	40.75	0.36	62.10	142.23	245.53	103902	8511500	8615402
9	0.53	17.26	0.42	3.68	34.26	56.15	34284	1900859	1935143
10	0.47	29.97	0.66	0.514	39.74	71.354	116036	1474373	1590409
11	0.62	32.65	0.95	0.23	68.44	102.89	16502	2419043	2435545
12	0.59	51.89	1.12	6.34	28.59	88.53	155645	3386173	3541818

Table 4.6: Comparison of computation time(s) taken by single stage optimization and optimization with successive refinement.

Test Case	Single Stage Optimization	Single Stage Optimization with Simplified Kinematics Evaluation	Successive Refinement and Adaptive Collision Checking with Simplified Kinematics Evaluation
1	2,921.30	2,404.90	14.60
2	timeout	timeout	53.63
3	timeout	timeout	18.46
4	timeout	timeout	38.76
5	6,128.80	4,948.30	77.48
6	172.49	162.34	40.14
7	7,978.40	1,998.10	29.84
8	32,248.00	31,088.00	245.53
9	1917.26	1,652.84	56.15
10	timeout	timeout	71.35
11	timeout	timeout	102.89
12	timeout	timeout	88.53

4.3.5 Solving Optimization Problem Efficiently

We find the path-constrained trajectory by solving the constrained optimization problem given in equation (4.9,4.10) using the approximate solution. If the estimated $N_{seg} = 1$, then we first attempt to represent the trajectory using a single spline (algorithm 4). If the optimization routine fails to find a solution to the single spline generation problem or an estimated $N_{seg} > 1$, then we generate the trajectory using multiple spline segments.

Solving Optimization Problem using Multiple Spline Segments: It may not be feasible to represent the trajectory of a configuration variable using a single spline of a low number of control points. Using a large number of control points increases computation time. We consider representing the trajectories of configuration variables with multiple splines each of them consisting of a small number of control points. This is computationally efficient as each of the segments are represented with a small number of control points, and each segment deal with a fraction of the sampled waypoints on the workspace path. Therefore the dimension of parameter space for each

segment is smaller, and the number of the objective function and constraint function evaluations is also smaller.

To generate trajectories with multiple spline segments, we solve the optimization problem given in equation (4.9,4.10) for each of the segments and store the nominal solution. Then we calculate pair-wise affinity score among all the consecutive segments. We define the affinity score between i^{th} and $i+1^{th}$ segment using equation (4.28). ϵ is a small number in equation (4.28).

$$1/(\left\|\Theta^{i}(1) - \Theta^{i+1}(0)\right\|_2 + \epsilon) \tag{4.28}$$

We rank the consecutive segment pairs based on their affinity score and pick the pair with the maximum score. Then we attempt to generate spline for one of the segment in the pair considering continuity constraint with the other. If a feasible solution is found, then we re-calculate the affinity scores and repeat the process for the remaining pairs. Otherwise, we attempt spline generation for the pair with next best affinity score. Essentially, solving the multi-segment spline trajectory generation problem can be viewed as a branch-and-bound search with an affinity score as the branch guiding heuristic (algorithm 5).

We iteratively increase the number of control points and the number of segments from their initial estimates until a complete solution is found for the path-constrained trajectories. Using StoneWeierstrass theorem [178] and analysis of polynomial approximation for joint trajectories of serial-chain mechanisms, it can be proved that, the method will converge to find a solution using multiple spline segments.

Optimization with Successive Refinement: Solving the full-blown problem considering all the constraints together can be computationally slow. It may converge to an infeasible local-minima too as there can be a large number of small basins in highly non-linear problems. Selecting the right sequence for adding constraint components to the optimization procedure can help to converge quickly in the feasible basins. We conducted experiments on different sequences

and used numerical analysis to identify the promising sequence that worked best across the test cases described in section 6.4. In our approach, we start by solving the problem of minimizing $ExecutionTime$ and $PoseError$ with constraints on position and velocity limits. This is done by taking $w_1, w_2 > 0$ and $w_2 = w_3 = w_4 = w_5 = 0$ in equation (4.9). We take the solution of this stage and use it as a seed for the next optimization stage. In the second stage we add $MultiSegmentContinuityError$ and $ProcessConstraintsViolation$ by setting $w_3 > 0, w_4 > 0$ in equation (4.9). The $CollisionScore$ is considered in third stage by setting $w_5 > 0$ in equation (4.9). We conducted experiments across the test-cases described in section 6.4 and used an active-learning inspired adaptive search [179, 180] to identify the weights (w_1-w_5) such that the computation time is minimized without compromising solution quality. This reduces the overhead of identifying appropriate Lagrange-multipliers from the optimization routine.

Adaptive Collision Checking: We perform adaptive collision checking to reduce computation time. To guarantee a collision-free trajectory, we need to check collision at densely sampled points on the workspace path. However, this will significantly increase the computation time. Instead, we start by checking collision at coarsely sampled points along the workspace path. We successively refine the sampling resolution in the neighborhood of the points that are either in a collision or very close to collision boundary. Successive refinement and adaptively checking collision significantly reduces computation time, as shown in section 6.4.

Simplified Kinematics Evaluation: In our approach, we calculate the pose error using the forward kinematics (FK) function. It is also used to transform each link for collision checking. FK is computed at every sample of the workspace path during each iteration. This is the bottleneck function during the computation as it is called for the maximum number of times by the algorithm, and it is relatively expensive to compute. Therefore, if we can speed up the computation time to evaluate this function, then our overall computation will become faster. In a vanilla implementation, transformation matrices will be computed for each joint and link and then multiplied sequentially. We performed algebraic simplification of the symbolic expression in

FK calculation. This speeded up the FK function evaluation by a factor of 300 as compared to basic implementation (from $0.5ms$ to $1.6\mu s$).

4.4 Results

We have conducted path-constrained trajectory generation experiments on 12 test cases. The test cases included 9, 10, 14, 17, and 21 DOF robotic systems. Figure 4.20 illustrates the test scenarios used in the simulation and physical experiments. We have used KUKA iiwa 7 and UR5 as the manipulators, and Inspector-bot (Mega Bot) as the mobile robot in our experiments. We conducted physical experiments on test case 9 and 10. Details of these problems and complete trajectories are available in video format for better understanding. The video can be accessed at `https://www.youtube.com/watch?v=bD7FItjOWY0`.

The workspace tool paths for test case 1,3, and 11 were generated from the skeletal geometry of the respective part. Tool paths for test case 2, 10, and 12 were generated using area coverage algorithm to cover the surface of the respective part. Sine-wave patterns were overlayed on the part's surface for generating the workspace path of test case 5 and 9. For test case 4, a continuous tool path was generated around the Fanta cans. For test case 6 and 7, we considered the articulated motion of the door and drawer mechanisms and considered the path traced by the knobs to generate the workspace path. For test case 8, we generated a 6-D motion plan for the object to be transferred using a search-based algorithm. This motion was constrained such that the grasping points on the part are reachable by the robot. Brief description of these test cases is as follows. Test case 1: (14 DOF) One robot is moving a large ring, and the other robot is moving out a smaller ring through it avoiding collisions. Test case 2: (14 DOF) A bi-manual robotic system sanding surface of a large part. Test case 3: (21 DOF) One robot is moving a part, and two other robots are tracing cyclic path-constrained trajectory with attached tools. Test case 4: (21 DOF) One robot is moving a tool through a path defined around Fanta cans attached to

a tray. This tray is attached to the end-effector of the second robot. Another tool is attached to this tray. This second tool is moving through a path defined around Fanta cans on a second tray. The second tray is attached to the third robot. This problem was inspired by ABB Fanta Can Challenge. Test case 5: (10 DOF) A mobile-manipulator is sanding a large part. Test case 6: (10 DOF) A mobile-manipulator is opening a door. Test case 7: (17 DOF) A bi-manual mobile-manipulator is opening a drawer. Test case 8: (17 DOF) A bi-manual mobile-manipulator is transporting a long part through a narrow corridor. Test case 9: (10 DOF) A mobile-manipulator is cleaning a whiteboard. Test case 10: (14 DOF) One robot is moving a part, and the other robot is polishing it. Test case 11: (14 DOF) One robot is moving a large wire-frame, and the other robot is moving out a ring through it avoiding collisions. Test case 12: (14 DOF) A bi-manual robotic system buffing the surface of a large part.

We benchmarked our approach against the following two methods- (1) Solving IK at sampled points on the workspace path and linearly interpolating configuration variables through the IK solutions and (2) Solving IK at sampled points on the workspace path and fitting a spline through the IK solutions. The first benchmark method is similar to Jacobian control-based approaches. For the second benchmark method, we used the same number of control points as we used in our method.

Workspace paths are continuous. Therefore, it is not computationally feasible to solve IK at every point of the workspace paths. For a complex workspace path, we need to have a large number of samples (very fine resolution sampling) to accurately generate the robot trajectory by linearly interpolating the IK solutions. Sequentially solving IK or inverse dynamics for a large number of samples may not be computationally feasible. On the other hand, Coarsely sampling the workspace path, finding IK solutions at the samples, and linearly interpolating between them to generate the robot trajectory may lead to a collision, error in pose constraint, violation of physical constraints, etc. These are the limitations of the first benchmark approach.

For the second benchmark approach, if we do not solve the IK at densely sampled points on the workspace path, then there is a chance of over-fitting or under-fitting the spline. This will lead to a collision and other constraint violations. On the other hand, densely sampling points on the workspace path, solving IK on them, and fitting spline through them will be computationally expensive.

As compared to our approach, the average accumulated pose error over the trajectories were 150 and 187 times higher for benchmark approach 1 and 2, respectively. Moreover, there were collisions in the trajectories for 8 out of 12 test cases generated by these benchmark approaches. The position error, orientation error, and collision score for the benchmark approaches, and our method are summarized in Table 4.3.

Table 4.4 and 4.5 summarizes the results of our approach on the 12 test cases. Table 4.4 summarizes the length of the workspace path, DOF of the multi-robot system, sampling resolution, number of spline segments, number of control points for splines, and the trajectory execution time for the test cases. Table 4.5 summarizes the computation time taken by different steps of our approach. It also summarizes the number of calls to the forward kinematics function for the test cases. The algorithm was implemented using MATLAB on a computer with an Intel Xeon 3.50GHz processor and 32GB of RAM. *Interior Point* algorithm was used as the optimization method. Seven out of twelve test cases took less than a minute to find the complete solution. The slowest computation time was for test case 8, which took about 3.5 minutes to find a solution. Our approach was able to generate trajectories under 10mm tolerance across the workspace paths.

We have compared computation time, and the number of FK function calls among the following three optimization runs for the simulation experiments- (a) Single-stage optimization, (b) Single-stage optimization with algebraically simplified FK function, and (c) Optimization with successive refinement and adaptive collision checking with algebraically simplified FK function. The maximum computation time for each scenario was set to $t_{max} = 10$ hours. We can see in Table 4.6 that algebraic simplification of the bottleneck function (FK) can reduce computation

time. Furthermore, optimization with successive refinement and adaptive collision checking helps to reduce the computation time significantly.

4.5 Summary

This chapter presented an approach to generate path-constrained trajectories for synchronous motion among multiple robots. The method adaptively selects the parametric representation of configuration variables and formulates the trajectory generation problem as a discrete parameter optimization problem. It generates an approximate solution and solves the complex non-linear optimization problem using successive refinement in a computationally efficient manner.

Chapter 5

Bringing Setup Planning Considerations in Trajectory Planning

5.1 Introduction

This chapter[1] describes setup planning algorithms to generate a sequence of poses for a part to be processed. Some regions on the part's surface may require multiple passes and the part may need to be repositioned and/or reoriented multiple times. This chapter will proceed with cleaning as a representative task and assume that the stain intensity determines the number of required cleaning passes. We used a cleaning tool with an abrasive surface. The experimental setup involves two robot arms. The first arm immobilizes the object. The second arm moves the cleaning tool. Figure 5.1 shows the experimental setup used to implement the results of the planning algorithm. The algorithm analyzes the stain and determines the sequence of poses (positions and orientations) needed to clean the part based on the kinematic constraints of the robot arm. Each pose is referred to as a setup in this work. The algorithm generates multi-pass trajectories for the cleaning tool to follow.

[1]The work in this chapter is derived from the work published in [181, 182]

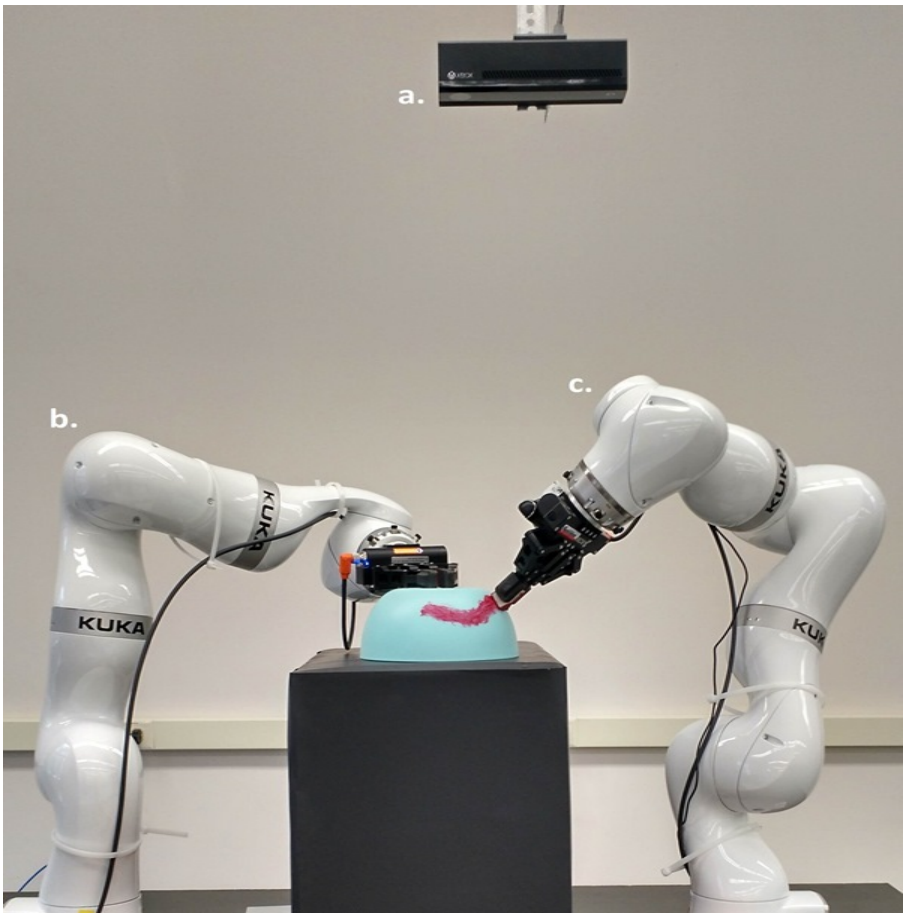


Figure 5.1: Robotic cleaning setup built with two KUKA iiwa robots and Microsoft Kinect: (a) Kinect. (b) Holding robot arm. (c) Cleaning robot arm.

5.2 Problem Formulation

We define the task for the robot as cleaning stains on an arbitrary curved surface $\Gamma \in \mathbb{R}^3$. Let $\ell \in \mathbb{R}^6 = \{x, y, z, \alpha, \beta, \gamma\}$ represent a general pose where (x, y, z) and (α, β, γ) represent the position and orientation, respectively in 3D. Let $\Gamma(\ell)$ represent the target surface oriented in an arbitrary pose ℓ. We approximate the stain on the surface as a set of small discrete stain patches $\mathscr{P} = \{p_i : i = 1, 2, \ldots, n\}$. Each patch p_i is a small planar triangle with an area $a_i \leq a_m$, where a_m is the surface area of the tip of the cleaning tool. Figure 5.2(a) demonstrates an example of a surface after triangulation. The red region represents $\mathscr{P}$.

We assume that the stain intensity is not uniform across the surface and that a single pass may not be able to clean the entire stain region completely. Let N_i represent the number of cleaning passes required to remove the stain from patch p_i. The number of passes is determined using image processing. We restrict the robot's motion such that its tool axis aligns to the surface normal and the sweeping motions are orthogonal to the surface normal. The robot may fail to satisfy these conditions for some segments of $\mathscr{P}$, for some $\Gamma(\ell)$. For each $\Gamma(\ell)$, we can test how many patches can be reached by the robot by solving its inverse kinematics. This reachability problem can be solved by changing $\Gamma(\ell)$ in steps such that all the subsets of the target surface fall in the robot's reachability space at least once. Therefore, we formulate our cleaning problem as a multi-setup, multi-pass, cleaning task with setup planning for the target surface and trajectory planning for the cleaning robot.

We define a set of candidate setups $\mathscr{S} = \{s_j; j = 1, 2, \ldots, m\}$, where $s_j = \{p_j^i : i = 1, 2, \ldots, k\} \subseteq \mathscr{P}$, $k \leq n$ and the conditions on robot motion is satisfied $\forall p_j^i \in s_j$. Figure 5.2(b,c) demonstrates two candidate setups to clean $\mathscr{P}$. Each setup s_j corresponds to a distinct pose $\Gamma(\ell_j)$. The maximum number of passes to cover s_j is given by $N_j^{\max} = \max_{i=1}^{|s_j|} N_j^i$, where N_j^i is the number of passes to clean p_j^i. Let t_j^i be the time required for the i^{th} cleaning pass for setup s_j. Let, t_j

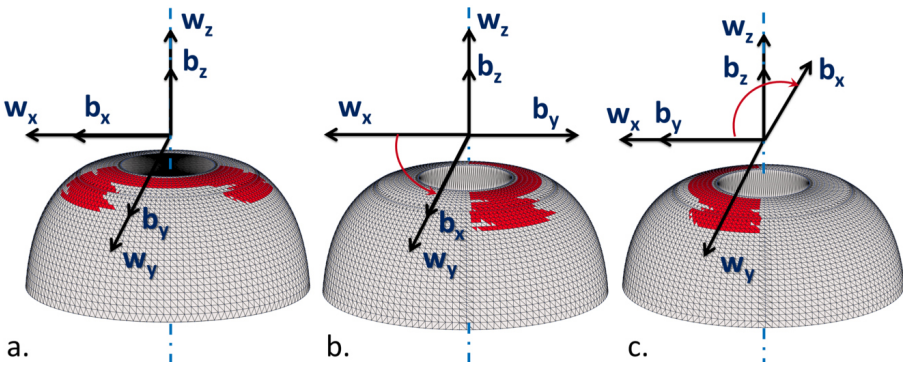


Figure 5.2: (a) Representation of an initial setup, i.e., the object's coordinate frame aligned with world coordinate frame. (w_x, w_y, w_z) and (b_x, b_y, b_z) represent the world coordinate frame and the bowl's coordinate frame respectively. The red region ($\mathscr{P}$) is the target region to clean. (b) (c) Two sample candidate setups. The setup configurations (b) and (c) are achieved by applying $+90^o$ and -90^o rotation about w_z $axis$ to the initial setup.

be the time to clean setup s_j, i.e., time to complete N_j^{max} cleaning passes for setup s_j. Then $t_j = \sum_{i=1}^{N_j^{\max}} t_j^i$.

Let, t_j^s be the setup time, defined as the time to change the pose of the object from setup s_j to setup s_{j+1}. Let, a_j^i is the surface area of the patch p_j^i in setup s_j. We define the cleaning rate for setup s_j as $R_j = (\sum_{i=1}^{|s_j|} a_j^i)/(t_j + t_j^s)$.

We can generate different ordered setup sequences by permuting $s_i \in \mathscr{S}$. We define a valid setup plan $\mathbb{S} = (s_1, s_2, \ldots, s_q)$, where $q \leq |\mathscr{S}|$, as an ordered sequence of setups that cleans the entire region. The path planner generates a trajectory τ_j for each $s_j \in \mathbb{S}$. The trajectory τ_j comprises N_j^{max} cleaning passes. The robot may need to reposition the tool to cover the disjointed patches in a setup. Both the cleaning motions and repositioning motions are captured by τ_j. The sampling based method for generating $\mathscr{S}$ and the algorithm to find setup sequence solutions are described in Section 5.3.2 and 5.3.3.1, respectively. Our method to generate tool-path is described in Section 5.3.4.

For each setup, s_j, there is an execution time $t^e(\tau_j) = t^c(\tau_j) + t^r(\tau_j)$, where $t^c(\tau_j) = t_j$ is the cleaning time when the tool is in contact with $\mathscr{P}$ while following τ_j and $t^r(\tau_j)$ is the repositioning

time when the cleaning robot moves between disjoint patches. We define the total cleaning time for a valid setup plan $\mathbb{S}$ as below:

$$\mathbb{T}(\mathbb{S}) = \sum_{j=1}^{|\mathbb{S}|} (t^e(\tau_j) + t^s_j) \tag{5.1}$$

The problem is formally stated as follows: *Find a setup plan* $\mathbb{S}^* = (s^*_1, s^*_2, \ldots, s^*_k), k \leq |\mathcal{S}|$, *where* $s^*_1, s^*_2, \ldots, s^*_k \in \mathcal{S}$, *such that* $\mathscr{P}$ *is completely clean and* $\mathbb{T}(\mathbb{S}^*)$ *is minimized.*

5.3 Approach

5.3.1 Overview of Approach

We approach the problem of setup planning in two stages. In the first stage, we generate candidate setups using sampling and optimization-based approaches. In the second stage, we identify a subset of the candidate setups and convert it into a sequence using branch and bound search such that, the selected setup sequence is the optimal setup sequence.

5.3.2 Generating candidate setups

A sampling-based approach is used to generate the initial set of candidate setups. For each candidate setup s_i, we determine the number of stain patches that the robot can reach in the desired orientation. This is achieved by solving the inverse kinematics of the manipulator for all the vertices of each patch $p_i \in \mathscr{P}$. We consider a stained patch to be reachable when all three vertices of that patch are reachable by the robot. We then assign that patch to the candidate setup s_i under consideration. While triangulating the surface, we pose the constraint that the surface area of the patch needs to be smaller than the surface area of the cleaning tooltip.

Our sampling starts with a coarse resolution over a wide range of configuration space parameters. We find a narrower feasible sampling range by eliminating setups that do not cover any

patch. We experimented with the following three sampling approaches in the narrower sampling range of configuration parameters: (1) Fine resolution uniform sampling, (2) Hierarchical uniform sampling with gradient descent, and (3) Random sampling with gradient descent. These sampling approaches are described in section 5.3.2.1, 5.3.2.1, and 5.3.2.1 respectively. The notion of non-dominated setups has been used to describe these approaches. We consider a setup s to be dominated if there exists another setup s' that contains all the patches covered by s.

5.3.2.1 Sampling Approaches

Fine resolution uniform sampling

(i) Initialize an empty set ψ of setups.

(ii) Set a fine resolution for each axis of the configuration space to perform uniform sampling over the narrower sampling range of configuration parameters.

(iii) Generate setup samples using the resolution from Step 2 and add them to ψ.

(iv) Eliminate all dominated setups in ψ.

(v) Send non-dominated setups as input to setup planner.

Hierarchical uniform sampling with gradient descent

(i) Initialize an empty set ψ of setups.

(ii) Set a coarse resolution for each axis of the configuration space to perform uniform sampling over the narrower sampling range of configuration parameters.

(iii) Generate samples of setups using above resolution.

(iv) Pick setups in the generated set which do not belong to ψ and refine them by using gradient descent over the configuration parameters to optimize the area covered by each setup. Add the refined setups to ψ.

Table 5.1: Gradient descent variants explored to improve area covered by a setup. Scoring methods: Type I - Conservative; Type II - Two rounds (Round 1 is conservative and round 2 is absolute)

Abbreviation	**Scoring method**	**Gradient descent type**
C_XY	Type I	Along x and y axes for fixed α
CA_XY	Type II	
C_XY_α	Type I	Along x and y axes keeping α fixed and then along α axis alone
CA_XY_α	Type II	
C_XYα	Type I	Along x, y, and α axes
CA_XYα	Type II	

(v) Eliminate all dominated setups in ψ.

(vi) If the non-dominated setups in ψ do not cover all $p_i \in \mathscr{P}$, then refine sampling resolution and go to step (3).

(vii) If the non-dominated setups cover all the patches in $\mathscr{P}$, then send them as input to the setup planner.

Random sampling

(i) Initialize an empty set ψ of setups.

(ii) Generate a random setup sample from the narrower sampling range of configuration parameters.

(iii) Refine this setup by using gradient descent to optimize the area covered by this setup.

(iv) If refined setup does not belong to ψ, then add it to ψ.

(v) Repeat steps 2-4 until setups in ψ cover all $p_i \in \mathscr{P}$.

(vi) Eliminate all dominated configurations in ψ.

(vii) Send non-dominated setups in ψ as input to the planner.

5.3.2.2 Scoring scheme for gradient descent to improve area coverage

We use gradient descent in the configuration space to optimize the area covered by a setup. Let $P_1 \in \mathscr{P}$ be the set of patches that were reachable by the robot for a starting configuration. We define $|P_1|$ as the score for the starting configuration. Suppose one step was taken by gradient descent and it is at a new setup configuration. Let $P_2 \in \mathscr{P}$ be the set of patches that are reachable by the robot for this new configuration. We use two score evaluation schemes:

Conservative scoring: In this scheme, we enforce a constraint to ensure that the improved configuration is able to cover all the patches that were present in the starting configuration. Therefore, if $P_1 \subseteq P_2$, then the score of the new configuration is $|P_2|$. Else, it is $|P_1 \cap P_2|$.

Absolute scoring: We do not enforce any constraint. The score for the new configuration is evaluated as $|P_2|$.

We explore different gradient descent variants by performing the search over the entire configuration space, over the subspaces by batch covering the entire configuration space, with a conservative scoring scheme, and with both conservative and absolute scoring schemes (refer to Table 5.1).

5.3.3 Setup Planning

The sampling based method described in Section 5.3.2 leads to a set of refined setups. The setup planner finds a setup plan from this set of candidate setups $\mathscr{S}$.

5.3.3.1 Setup Planner

Our algorithm uses a *depth-first branch-and-bound (DFBnB)* search with computational time bound $\mathscr{T}_{max}$ to generate setup plans. DFBnB [183] is an efficient search algorithm. We have adopted this algorithm to solve our setup planning problem. Let $s^i_j \in \mathscr{S}$ represent a setup, where j is the node index in the i^{th} solution (not necessarily a sequence of setups that completes cleaning the part). Therefore, the i^{th} solution is given by $\mathbb{S}^i = (s^i_1, s^i_2, \ldots, s^i_k), k \leq |\mathscr{S}|$. The

planner implementation consists of the main routine FindSetupPlan($\mathscr{P}, \mathscr{S}$) (Algorithm 1) that calls the routine AddSetUp($\mathscr{P}_r, \mathscr{S}_u, \mathbb{S}_{curr}$) (Algorithm 2). Cleaning rate is used as a branch-guiding heuristic. At the first instance of a call to Algorithm 14, we compute the cleaning rate (defined in Section 5.2) for all the setups and select the setup $s_1^0 \in \mathscr{S}$, which has the maximum cleaning rate. This results in the remaining stain area $\mathscr{P}_r = \mathscr{P} - \{p_i \in s_1^0\}$ to be covered and the set of un-used setups $\mathscr{S}_u = \mathscr{S} - s_1^0$. Then, in the second instance, we recompute the cleaning rate for all setups in $\mathscr{S}_u$, and select the setup $s_2^0 \in \mathscr{S}_u$, which has the maximum cleaning rate. This cycle repeats by making recursive calls to Algorithm 14 to find other setups s_j^0, one at a time, until all the stain area is covered ($\mathscr{P}_r = \emptyset$). Once the initial solution $\mathbb{S}^0$ is found, we set the best solution $\mathbb{S}^* = \mathbb{S}^0$ and the cleaning time for the best solution $\mathbb{T}^* = \mathbb{T}(\mathbb{S}^0)$, which is the sum of the setup and execution times for all the setups in the solution sequence. Then we keep branching to find better solutions until $\mathscr{T}_{max}$ is exceeded.

We use the following branch-pruning heuristic for faster convergence. Let $\mathbb{S}_{curr}$ represent the set of setups in the current partial solution. We consider $\mathbb{T}(\mathbb{S}_{curr})$ as the cost of the current partial solution. Next, a lower bound on future cost is defined as the lower bound on the execution and setup time for the remaining stain region $\mathscr{P}_r$:

$$\mathbb{T}^{lb}(\mathscr{P}_r) = \min_j (t_j^s + t^e(\tau_j)) \tag{5.2}$$

where $s_j^i \in \mathbb{S} - \mathbb{S}_{curr}$. If $\mathbb{T}(\mathbb{S}_{curr}) + \mathbb{T}^{lb}(\mathscr{P}_r) \geq \mathbb{T}^*$, then we prune that branch from the search tree since it is sub-optimal.

Algorithm 6 FindSetupPlan ($\mathscr{P}, \mathscr{S}$)

1: Initialize $\mathbb{S}^* = \emptyset$
2: Initialize $\mathbb{T}^* = \infty$
3: Call AddSetUp ($\mathscr{P}, \mathscr{S}, \emptyset$)
4: Return $\mathbb{S}^*$

Algorithm 7 AddSetUp($\mathscr{P}_r, \mathcal{S}_u, \mathbb{S}_{curr}$)

If computation time exceeds $\mathcal{T}_{max}$ then abort search.
If $\mathscr{P}_r = \emptyset$ and $\mathbb{T}(\mathbb{S}_{curr}) > \mathbb{T}^*$ then Return
If $\mathscr{P}_r = \emptyset$ and $\mathbb{T}(\mathbb{S}_{curr}) \leq \mathbb{T}^*$, then
  update $\mathbb{S}^* = \mathbb{S}_{curr}$, $\mathbb{T}^* = \mathbb{T}(\mathbb{S}_{curr})$ and Return
If $\mathbb{T}(\mathbb{S}_{curr}) + \mathbb{T}^{lb}(\mathscr{P}_r) \geq \mathbb{T}^*$, then Return
Otherwise,
  If $\exists p \in \mathscr{P}_r$ associated with only one $s \in \mathcal{S}_u$, then
  Find $\mathscr{P}(s)$ by patches that are present in s
  Call AddSetUp ($\mathscr{P}_r - \mathscr{P}(s)$, $\mathcal{S}_u - s$, $\mathbb{S}_{curr} \bigcup s$)
  Otherwise,
    Sort $\mathcal{S}_u$ by highest to lowest cleaning rate.
    For every s in $\mathcal{S}_u$ in decreasing order of
    cleaning rate
      Find $\mathscr{P}(s)$ by patches that are present in s
      Call AddSetUp ($\mathscr{P}_r - \mathscr{P}(s), \mathcal{S}_u - s, \mathcal{S}_{curr} \bigcup s$)

Algorithm 8 FindSetupPlan ($\mathscr{P}, \mathcal{S}$)

Initialize $\mathbb{S}^* = \emptyset$
Initialize $\mathbb{T}^* = \infty$
IESS $\leftarrow$ InitialEstimateOfSetupSize($\mathscr{P}, \mathcal{S}$)
Call AddSetUp ($\mathscr{P}, \mathcal{S}, \emptyset$)
Return $\mathbb{S}^*$

Algorithm 9 AddSetUp($\mathscr{P}_r, \mathcal{S}_u, \mathbb{S}_{curr}$)

If computation time exceeds $\mathcal{T}_{max}$ then abort search.
If $|\mathbb{S}_{curr}| >$ IESS then Return
If $\mathscr{P}_r = \emptyset$ and $\mathbb{T}(\mathbb{S}_{curr}) > \mathbb{T}^*$ then Return
If $\mathscr{P}_r = \emptyset$ and $\mathbb{T}(\mathbb{S}_{curr}) \leq \mathbb{T}^*$, then
  update $\mathbb{S}^* = \mathbb{S}_{curr}$, $\mathbb{T}^* = \mathbb{T}(\mathbb{S}_{curr})$ and Return
If $\mathbb{T}(\mathbb{S}_{curr}) + \mathbb{T}^b(\mathscr{P}_r) + \mathbb{T}_{decay} \geq \mathbb{T}^*$, then Return
Otherwise, If $\exists p \in \mathscr{P}_r$ associated with only one $s \in \mathcal{S}_u$, then
  Find $\mathscr{P}(s)$ by patches that are present in s
  Call AddSetUp ($\mathscr{P}_r - \mathscr{P}(s)$, $\mathcal{S}_u - s$, $\mathbb{S}_{curr} \bigcup s$)
  Otherwise,
    Sort $\mathcal{S}_u$ by highest to lowest cleaning rate.
    For every s in $\mathcal{S}_u$ in decreasing order of
    cleaning rate
      Find $\mathscr{P}(s)$ by patches that are present in s
      Call AddSetUp ($\mathscr{P}_r - \mathscr{P}(s), \mathcal{S}_u - s, \mathcal{S}_{curr} \bigcup s$)

5.3.3.2 Setup Planner using initial estimate on setup size

In Section 5.2, we defined setup time as the time taken to change setups. Note that no cleaning happens during this time. Therefore, a solution with a large number of setups will lead to low cleaning rate. This implies that the initial solution produced by the setup planner may be sub-optimal. The number of setups in the optimal solution may be much lower than that in the initial solution. Since the algorithm considers the total time of this sub-optimal solution as an initial bound and keeps updating when a better solution is found, it will take a long time to converge. However, the convergence will be faster if we could use a better initial bound. If we can estimate the setup size in the optimal solution (i.e., the number of setups in the optimal solution or the size of the minimum set cover[2]) before running the AddSetUp algorithm, then we could use it as a branch-pruning heuristic to converge faster towards the optimal solution.

For this purpose, we modified Algorithms 13 and 14 as Algorithms 15 and 9, respectively. Algorithm 15 calls InitialEstimateOfSetupSize($\mathscr{P}, \mathscr{S}$) (Algorithm 10) in step 3, which gives us an initial estimate of the setup size (IESS) of the optimal solution. Step 2 of Algorithm 9 prunes the branches in the search tree when the solution size exceeds the initial estimate (IESS). Therefore, it makes the search converge faster towards the solution compared to Algorithm 14.

Algorithms 2 and 4 are depth-first branch-and-bound search algorithms. They are constructed in a recursive manner. Their computational complexity is exponential in time with an order of $|\mathscr{S}|$, where $\mathscr{S}$ is the set of candidate setups. The exponent is the number of nodes in the first solution branch. In Algorithm 5, N_{repeat} is the number of times the random sampling process is repeated. Algorithm 5 is constructed as two nested loops. The outer loop runs N_{repeat} times. The inner loop picks one setup from the set of candidate setups ($\mathscr{S}$) in each iteration until $|\mathbb{P}|$ is empty. Therefore the inner loop runs $|\mathscr{S}|$ times or lower. However, as explained in section 5.4.2, N_{repeat} needs to be significantly larger than $|\mathscr{S}|$. Therefore, the computational complexity of Algorithm 5 is linear in time with N_{repeat}.

[2]Sub-collection of the setups which will cover all stain patches

Algorithm 10 InitialEstimateOfSetupSize($\mathscr{P}, \mathcal{S}$)

```
1:  Initialize ψ = ∅
2:  Initialize N = [ ]
3:  For i in range(N_repeat)
4:    Initialize ℙ = 𝒫
5:    While( |ℙ| > 0 )
6:        Randomly pick s ∈ 𝒮 − ψ
7:        ψ ← ψ ∪ s
8:        Find 𝒫(s) by patches that are present in s
9:        ℙ ← ℙ − 𝒫(s)
10:   N.append(|ψ|)
11: Return min(N)
```

Initial estimate on setup size Algorithm 10 generates the initial estimate on number of setups by finding the set cover through random sampling. It repeats this process for N_{repeat} times. The probability of finding the minimum set cover at least once in N_{repeat} trials can be be analytically derived as follows.

Let $\mathbb{S}_c = \{\mathcal{S}_c^i \subseteq \mathcal{S}\}$ represent the set of all set-covers. The minimum set-cover $\mathcal{S}_c^* \in \mathbb{S}_c$ is given by

$$\mathcal{S}_c^* = \{s_1^*, \ldots, s_q^* \in \mathcal{S} \mid \bigcup_{i=1}^{q} \mathscr{P}(s_i^*) = \mathscr{P}\} \tag{5.3}$$

$$\text{where, } q = arg\min |\mathcal{S}_i|,\ \mathcal{S}_i \in \mathbb{S}_c \tag{5.4}$$

Note that if $|\mathcal{S}_c^*| = |\mathcal{S}|$, then the size of minimum set-cover can be found using Algorithm 10 with probability one.

Next, assume that the size of the minimum set-cover $q = |\mathcal{S}_c^*| < |\mathcal{S}|$ and that $\mathcal{S}_c^*$ is unique. The probability of finding this unique set cover is given by the following theorem.

Theorem: If there is a unique minimum set cover $\mathcal{S}_c^* \subset \mathcal{S}$ such that $q = |\mathcal{S}_c^*| < m = |\mathcal{S}|$, then the probability of finding $\mathcal{S}_c^*$ using Algorithm 10 is given by

$$1 - \left(1 - \frac{q! \times (m-q)!}{m!}\right)^{N_{repeat}} \tag{5.5}$$

Proof: Algorithm 10 (step 5−9) finds a set cover by randomly sampling one setup at a time from $\mathcal{S}$.

Since $q < m$, the probability of finding the minimum set-cover in one trial is given byf

$$\begin{aligned} p &= \left(\frac{q}{m}\right)\left(\frac{q-1}{m-1}\right)\cdots\left(\frac{q-(q-1)}{m-(q-1)}\right) \\ &= \frac{q! \times (m-q)!}{m!} \end{aligned} \tag{5.6}$$

Hence, the probability of not finding the minimum set cover in one trial is $(1-p)$. This implies that the probability of not finding the minimum set cover by repeating the random sampling process for N_{repeat} times is $(1-p)^{N_{repeat}}$.

Therefore, the probability of finding minimum set cover at-least once in N_{repeat} trials is given by

$$1-(1-p)^{N_{repeat}} = 1-\left(1-\frac{q! \times (m-q)!}{m!}\right)^{N_{repeat}}$$

The above theorem considered the case when the minimum set-cover is unique. However, there could be multiple minimum set-covers of the same size. In these cases, the probability will be much higher, and (5.5) gives us a lower bound on the probability of finding a minimum set cover using Algorithm 10. Note that, we cannot predict q beforehand. We can numerically evaluate the probability given by (5.5) for $q = 1, 2, \ldots, m-1$ with different N_{repeat}. This can guide us to select the best N_{repeat} for different m. In Section 5.4.2, we discuss how to choose N_{repeat} such that Algorithm 10 can guarantee the size of optimal solution with high probability.

The heuristic-based on IESS does not prune branches with an optimal solution if the setup size of the true optimal solution size is smaller than IESS. This is because it only prunes branches with solution size higher than IESS. Suppose the size of the optimal solution is IESS. There might be multiple solutions of the same size but with different cleaning rates. Since branches with solution size $\leq$ IESS are not pruned, the search will still yield a solution with optimal cleaning rate.

The effectiveness of the branch-pruning heuristic in step 4 of Algorithm 14 depends on the setup time. It may generate a solution with a large number of setups if the setup time is too low. The initial estimate on the setup bound heuristic makes Algorithm 9 robust against variations in setup time. It will guarantee a minimum setup size with a good probability even if it fails to guarantee a solution with true optimal cleaning rate.

Different bounds on future cost Step 4 of Algorithm 14 and step 5 of Algorithm 9 are branch-pruning heuristics that prune branches based on future cost of covering remaining patches. In Section 5.3.3.1, the lower bound on future cost was given by (5.2). If we can guarantee with high enough probability that Algorithm 10 can estimate the solution size of the minimum set cover, then we can modify (5.2) as follows:

$$\mathbb{T}^b(\mathscr{P}_r) = \max_i(t_i^s + t^e(\tau_i)),\ where\ s_i \notin \mathbb{S}_{curr} \tag{5.7}$$

This will enable the search to converge faster. However, this will guarantee an optimal solution with a probability ≤ 1.0.

The algorithm may find the least setup-size solution very fast (might not be optimal in the cleaning rate). However, it may still take a long time to prune all the remaining branches when the number of non-dominated setups or setup time is high. In step 5 of Algorithm 9, the term $\mathbb{T}_{decay}$ on the left side of the inequality, can be an exponential function of the number of nodes expanded in the search tree. This will accelerate convergence (similar to simulated annealing Algorithm) for practical purposes when the true optimal solution might not be absolutely desired. In our experiments, we set $\mathbb{T}_{decay} = 0$ and consider the setup time to be fixed for all setups.

5.3.4 Tool Path Planning

A cleaning trajectory is generated for each pass of each setup by creating a spline through the connected triangles in the setup. Therefore, the cleaning tool path consists of multiple splines for a single setup. A setup $s_j^* \in \mathbb{S}^*$ may require $N_j^{\max}$ passes to complete cleaning. Correspondingly, there is a total $N_j^{\max}$ number of trajectories generated for the setup s_j^*. For example, if s_j=$\{p_1^j, p_2^j, p_3^j\}$ and number of passes required for p_1^j, p_2^j and p_3^j are 1, 2, and 3, respectively, then the first tool path will pass through all the patches p_1^j, p_2^j, and p_3^j. The second tool path will pass through p_2^j and p_3^j, and the third tool path will cover p_3^j alone. This strategy utilizes the advantage of continuous motion and saves cleaning time. We find most stains to be locally continuous in reality. Therefore, all the stain patches in a locally continuous stain region may require the same number of cleaning passes. It will take more time if the robot stops at each stain patch to complete all the required cleaning passes for it and then moves to the next patch.

The target surface is intersected with equally spaced parallel planes to generate the continuous trajectory [184, 185]. We sample the curves resulting from the intersections to get way-points. From the way-points, we create spline curves as nominal trajectories to be followed by the cleaning tool.

5.4 Results

5.4.1 Synthetic test cases

We conducted experiments in simulation with a Puma 560 robot to evaluate our setup planner. We considered four synthetic test objects, as illustrated in Fig. 5.3. They are represented as triangulated meshes. The area of each triangular face is less than 1.5 cm^2. The red regions represent the stain and consist of about 4,000 to 5,000 stain patches on each object. Stain intensity was user-defined such that not more than three cleaning passes would be required. In

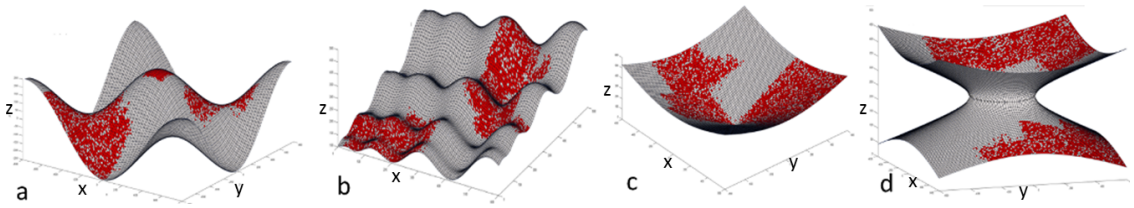


Figure 5.3: Four curved surfaces used as synthetic test objects: (a) Sine function (1,600mm x 1,600mm x 500mm), (b) Schwefel function (600mm x 600mm x 350mm), (C) Concave bowl (1,200mm x 1,200mm x 530mm), and (d) Hyperboloid of one sheet (1,200mm x 1,200mm x 412mm).

our experiments, we restricted ourselves to a 3D configuration space (x, y, and α, where α is the rotation about z) for the objects' pose (i.e., we did not consider roll, pitch, and translation about z-axis). The narrowed down sampling region in this space was [0, 1 m] $\times$ [0, 1 m] $\times$ $[0, 2\pi]$.

5.4.1.1 Baseline results

We conducted fine resolution sampling on the configuration space, removed dominated setups, and ran the setup planner on the non-dominated setups to find the number of lowest possible setups for each synthetic test object. We did not use gradient descent in these baseline experiments. We found out that the largest sampling resolution corresponding to the solution with a minimum number of setups was 50mm $\times$ 50 mm $\times$ 10^o for all the test cases. This leads to 16,317 setup configurations to start with. Table 5.2 summarizes the baseline results. We have a reachability test function that solves the inverse kinematics of the robot for each stain patch for a given setup. Each instance of this function takes a significant amount of computational time. Let, n_{rfc} be the number of calls made to the reachability test function. For example, let's assume there are 5,000 stain patches on a target surface. For each pose of the object, we need to test if the robot can reach all the three vertices of each of the stain patches. For each pose of the object, this will involve finding the inverse kinematics solution of the robot for 15,000 times. If there are 12 candidate setups, then we will need to solve inverse kinematics for 180,000 times. Therefore, we use the number of calls to the reachability test function (n_{rfc}) as a performance metric of the

algorithms. Lower n_{rfc} with optimal setup plan indicates better performance of the algorithm. For all the cases in Table 5.2, n_{rfc} = 16,317.

5.4.1.2 Comparison between uniform and random sampling approaches with and without gradient descent

As mentioned in Sections 5.3.2.1 and 5.3.2.1, we experimented with hierarchical uniform sampling and random sampling approaches. Different gradient descent variants (Section 5.3.2) were added on top of these sampling schemes, and their performance was evaluated. Different sampling resolutions were used at different hierarchy levels for the hierarchical sampling approach. Sampling resolution was scaled by half along one axis at a time at the different hierarchies. Table 5.3 summarizes the results of hierarchical uniform sampling. Optimal setup solution was found for all four cases by refining the set of candidate setups with the CA_XY variant of gradient descent (refer to Table 5.1).

We evaluated the random sampling approach by repeating the experiment 100 times for each test case with each type of gradient descent. Table 5.4 shows the likelihood of finding the optimal solution for synthetic objects. The CA_XY variant worked reasonably well for all four test cases. The number of function calls for the Sine-object, Schwefel-object, Concave bowl, and Hyperboloid-object is illustrated in Fig. 5.4. We can see that sampling without gradient descent, and sampling with C_XY and CA_XY gradient descent have relatively low variation in n_{rfc}.

From the experiments on synthetic cases, we found that hierarchical uniform sampling with the CA_XY gradient descent variant guarantees an optimal solution. The random sampling-based approach has a significant reduction of n_{rfc}. However, it guarantees an optimal solution with a probability of less than 0.5.

Figure 5.5 represents the landscape of area coverage by absolute scoring on a uniformly sampled grid on the $x-y$ plane for a fixed α. For any (x, y) location of the object, the $z-axis$ corresponds to the number of triangles reachable by the robot. The sampling resolution was 50 mm in both x

Table 5.2: Baseline results for the four test cases in simulation

Case	No. of setups	Non Dominated Setups
Sine	4	36
Schwefel	1	1
Conic	5	531
Hyperboloid	4	751

and y directions. The black-star represents the randomly selected initial seed (x, y configuration) and a number of triangles covered by this configuration. The green-star and the red-star represent points indicating the number of triangles covered by the configurations, after running a gradient descent using conservative and absolute scoring schemes, respectively. We can see that both scoring schemes give a significant improvement in area coverage for gradient descent on the $x - y$ plane keeping α fixed.

5.4.1.3 Performance of CA_XY type gradient descent

Intuitively, we can understand that changing orientation can make an abrupt change in the number of reachable stain patches. Whereas, changing position makes a gradual change in the number of reachable stain patches. This phenomenon is reflected in the landscapes of Fig. 5.5, which explains why CA_XY type gradient descent performed well for both hierarchical uniform sampling and random sampling approaches.

5.4.1.4 Comparison of heuristics for setup planner

We used a branch-guiding heuristic and a branch-pruning heuristic in our depth-first-branch-and-bound-search. The branch-guiding heuristic is based on the cleaning rate. In step 5 of Algorithm 14 and step 6 of Algorithm 9, the candidate setups are sorted based on the cleaning rate. The setup with relatively high cleaning rate is chosen as a potential optimal solution branch. The branch-pruning heuristic is based on bound on future cost (future cleaning time). Step 4 in Algorithm 14 and step 5 in Algorithm 9 prune branches based on the estimated bound on future cost.

Table 5.3: Results of Hierarchical Uniform sampling for four synthetic objects

Case	**Gradient Descent Type**	**No. of setups in solution**	**Resolution level to find set-cover**	n_{rfc}
Schwefel	W/o Grad. Dsc.	1	1	36
	CA_XY	1	1	264
	CA_XY_α	1	1	811
	CA_XYα	1	1	426
Sine	W/o Grad. Dsc.	5	1	36
	CA_XY	4	1	536
	CA_XY_α	5	1	1074
	CA_XYα	5	1	659
Concave bowl	W/o Grad. Dsc.	6	1	36
	CA_XY	5	1	574
	CA_XY_α	5	1	1179
	CA_XYα	7	2	1209
Hyperboloid	W/o Grad. Dsc.	5	1	36
	CA_XY	4	1	457
	CA_XY_α	4	1	1089
	CA_XYα	5	1	671

Table 5.4: Likelihood of finding the optimal solution by random sampling

Grad. Dsc. Type	**Sine**	**Schwefel**	**Concave bowl**	**Hyperboloid**
Without Gradient Descent	0	0.13	0.01	0.05
C_XY	0.1	0.35	0.14	0.16
CA_XY	0.19	0.35	0.21	0.32
C_XY_α	0.02	0.51	0.02	0
CA_XY_α	0.07	0.54	0.04	0.01
C_XYα	0.03	0.41	0	0
CA_XYα	0.05	0.43	0.05	0.08

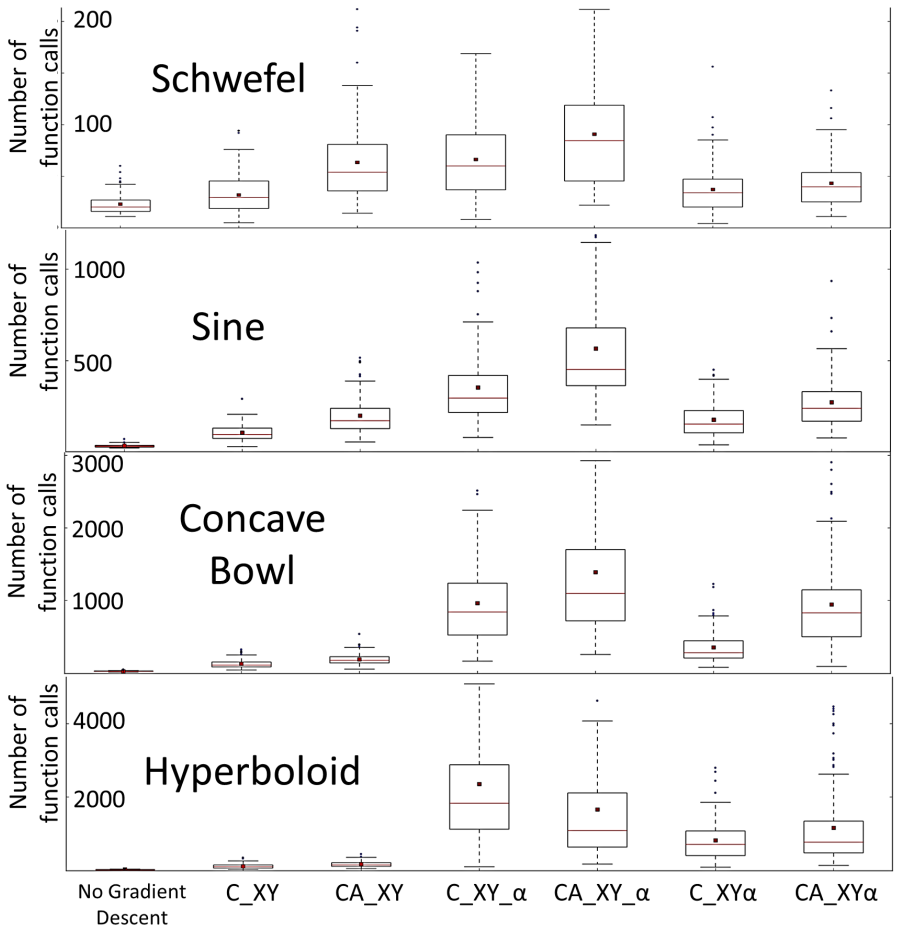


Figure 5.4: Variation in n_{rfc} for different kinds of gradient descent in the random sampling based approach

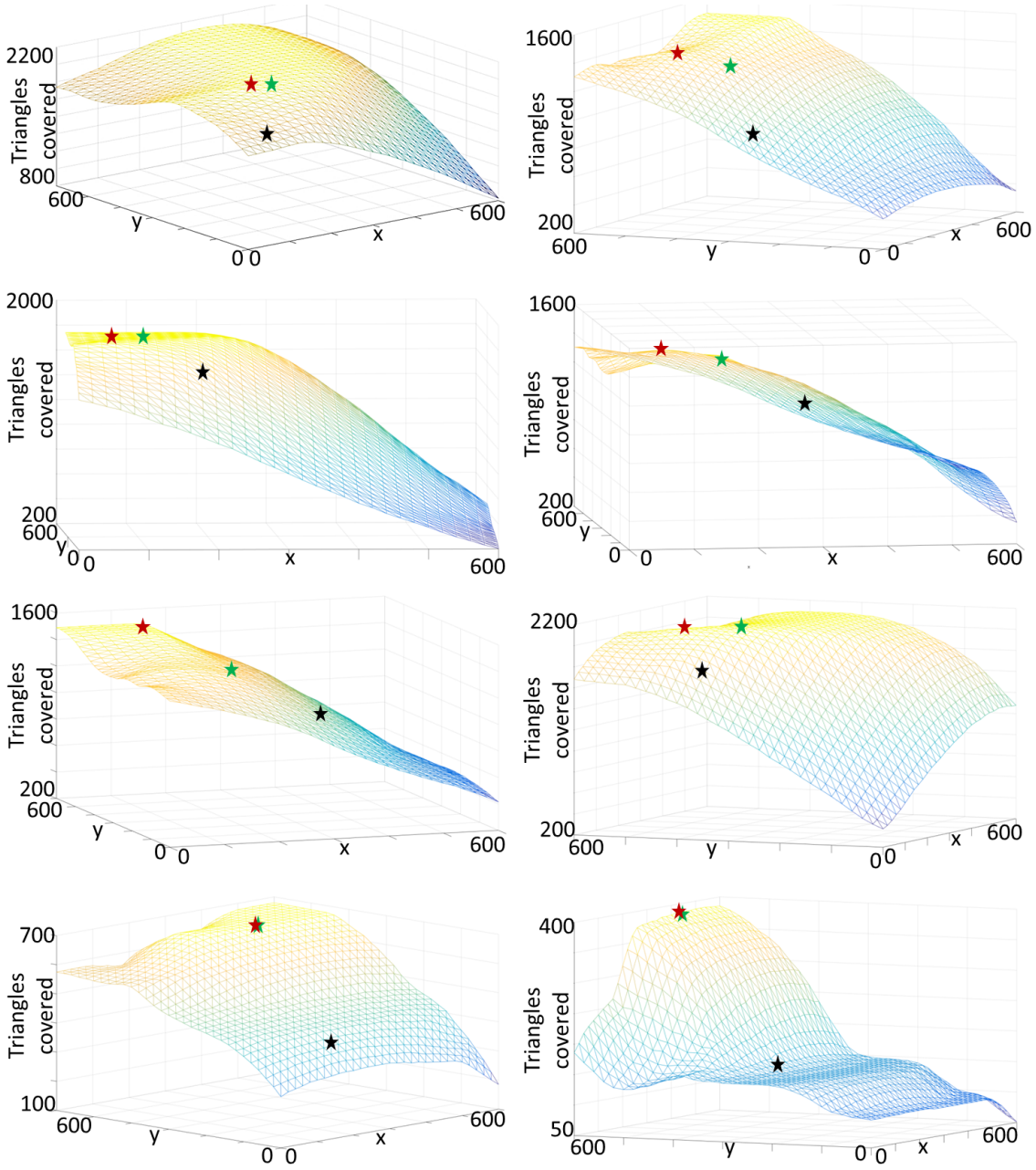


Figure 5.5: Eight examples of gradient descent along x & y keeping α fixed (CA_XY type gradient descent) for the convex bowl used in physical test. In these figures, x and y axis represent the (x,y) configuration of the part. The z axis represents number of reachable stain patches on the part by the robot, at the corresponding (x,y) location of the part. Therefore the mesh is representing the landscape of the number of reachable stain patches for different (x,y) configuration of the part. The number of stain patches reachable by the robot at the initial (x,y) location of part is represented with black-star. The green-star and red-star represent the number of stain patches reachable by the robot at the new (x,y) location after applying gradient descent using conservative and absolute scoring, respectively.

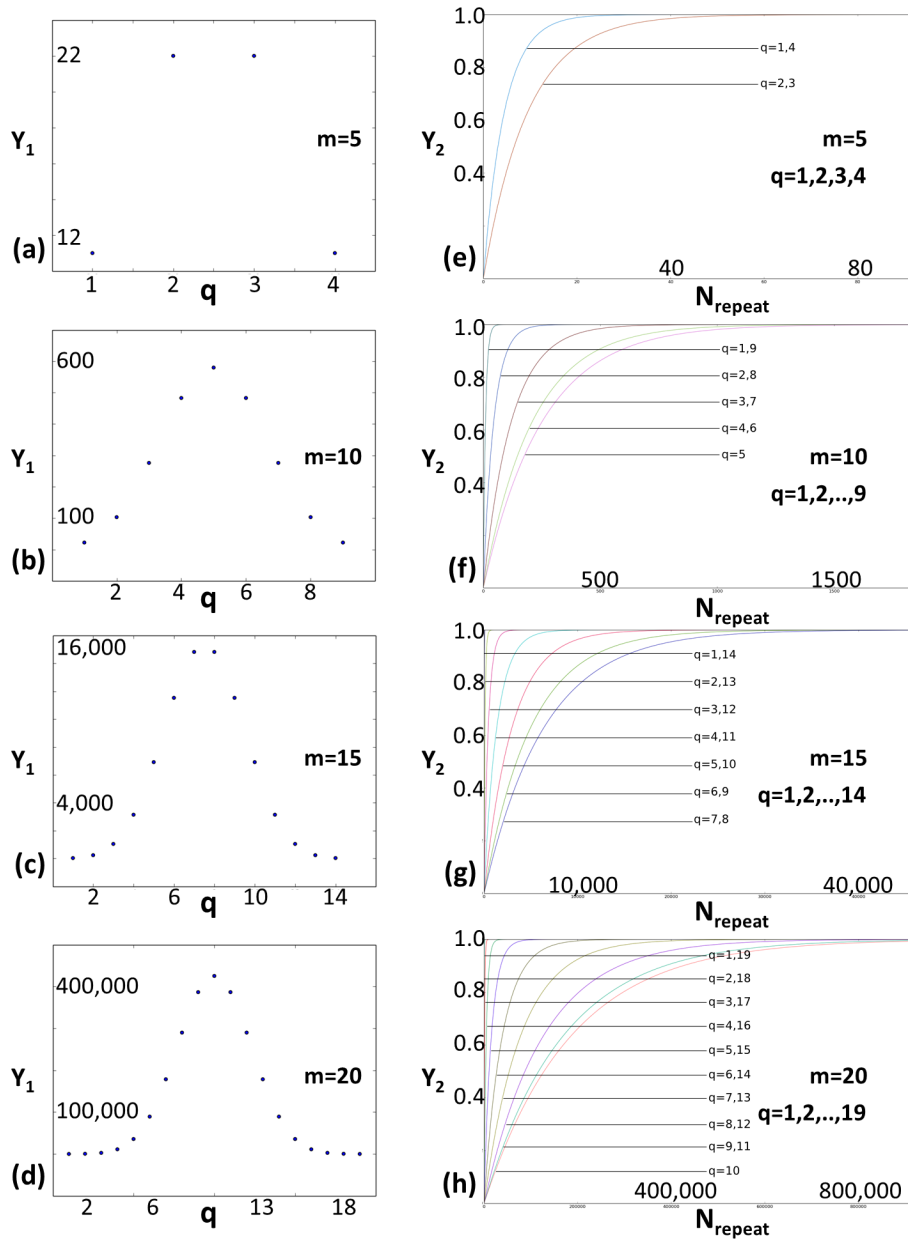


Figure 5.6: (a,b,c,d): (Y_1) vs q, where $Y_1 = N_{repeat}$ to find minimum set cover size with probability 0.90. (e,f,g,h): (Y_2) vs N_{repeat}, where Y_2 = Probability of finding minimum set cover size. $m = |\mathcal{S}|$ =Cardinality of the set of candidate setups, for q =possible minimum set cover size = $1, 2, \ldots, m-1$. In (e,f,g,h) each curve represent different q

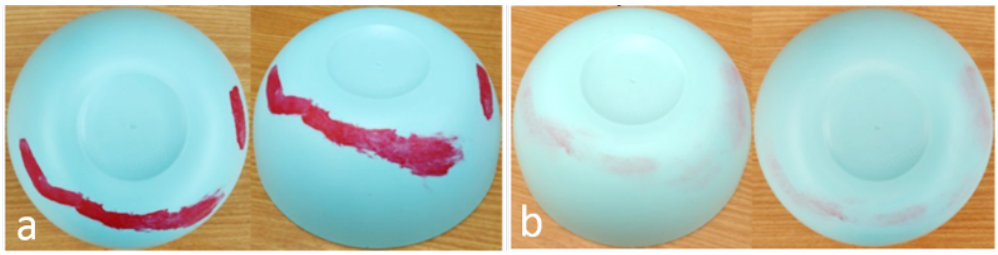
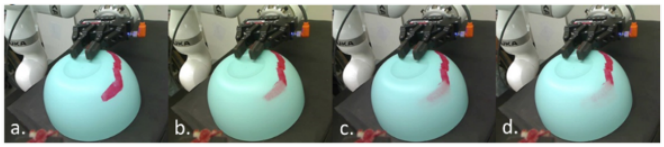
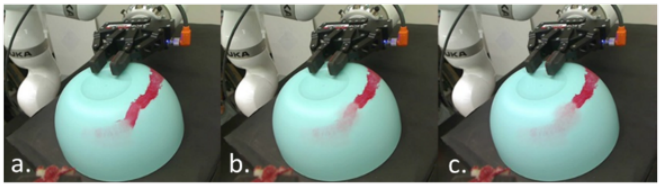


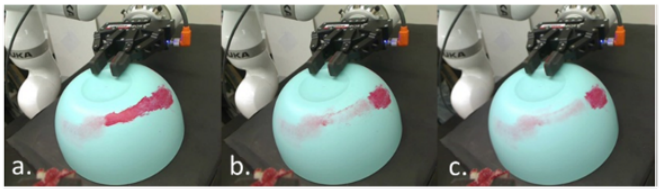
Figure 5.7: The bowl's surface (a) before cleaning and (b) after cleaning. Images taken from two different angles.



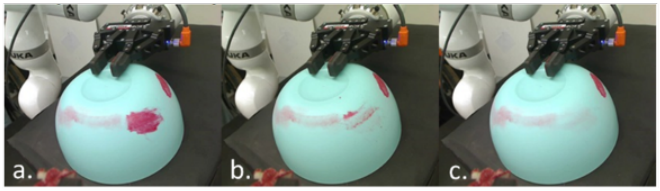
Setup 1: (a) before cleaning, (b) after first cleaning pass, (c) after second pass, (d) after third pass.



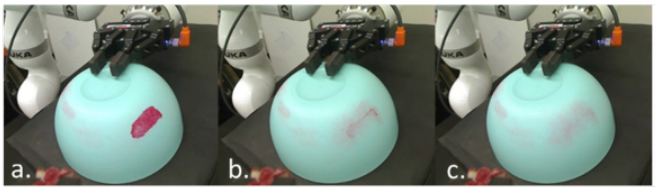
Setup 2: (a) before cleaning, (b, c) after first I & II passes.



Setup 3: (a) before cleaning, (b, c) after first I & II passes.



Setup 4: (a) before cleaning, (b, c) after first I & II passes.



Setup 5: (a) before cleaning, (b, c) after first I & II passes.

Figure 5.8: Snapshots from a video showing execution of cleaning on the plastic bowl according to setup plan comprising five setups. The part's surface is mostly convex.

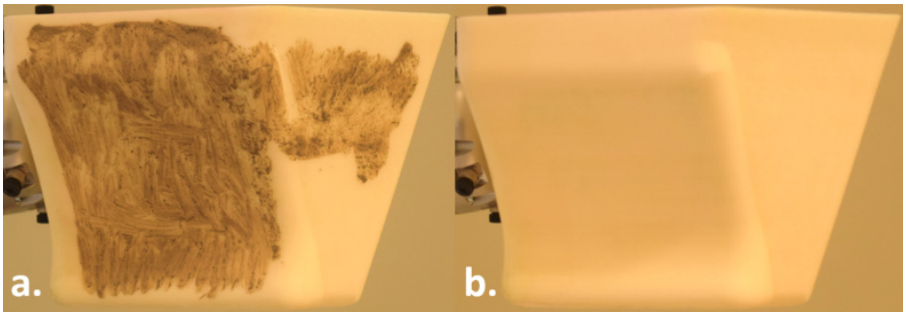


Figure 5.9: The 3D printed ship hull model's surface (a) before cleaning and (b) after cleaning.

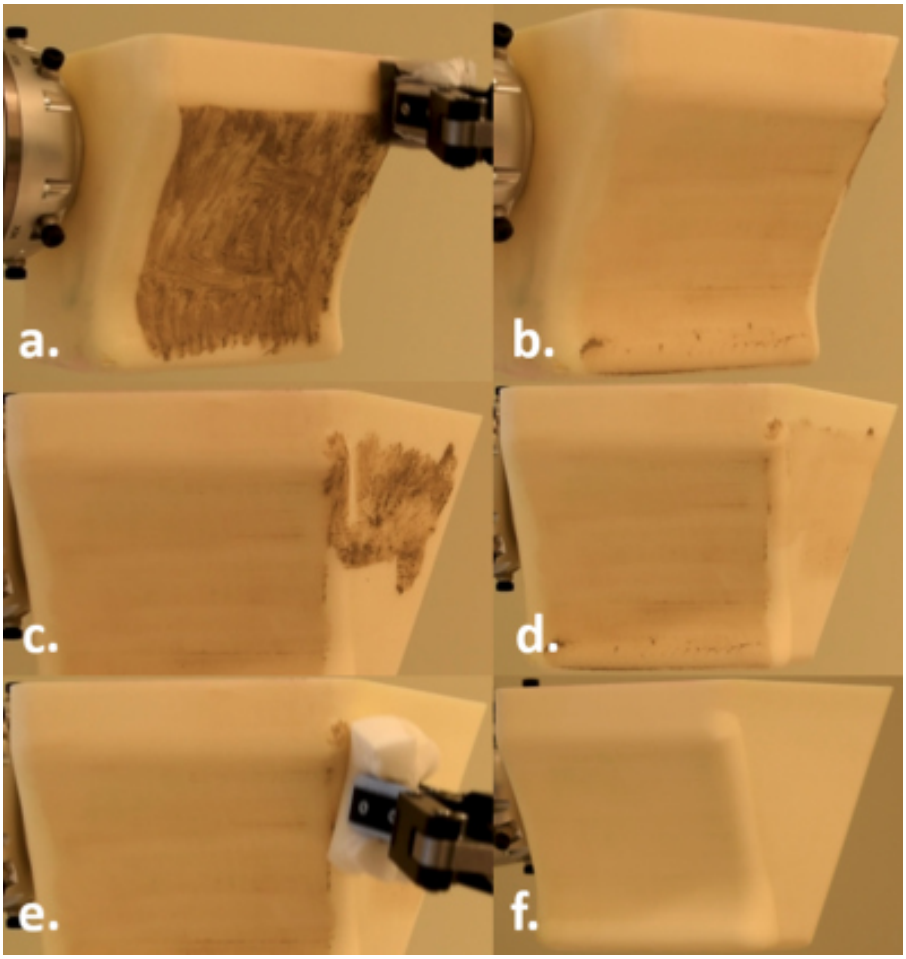


Figure 5.10: Snapshots from a video showing execution of cleaning on the 3D printed model of ship hull. This part has convex, concave, and flat regions on the surface. The setup planner generated a two setups solution. (a) On going cleaning on setup 1, (b) Setup 1 after cleaning by scrubbing, (c) Setup 2 before cleaning, (d) Setup 2 after cleaning by scrubbing, (e) Robot wiping off the dust from the surface, (f) The cleaned surface

We experimented by keeping either of these heuristics ON or OFF to see their impact on the convergence rate of the search. Table 5.5 summarizes the results for three synthetic objects. In these experiments, there were 36 candidate setups. They were uniformly sampled without refinement using gradient descent. There were 7, 12, and 13 non-dominated setups for Hyperboloid, Sine, and Concave bowl objects, respectively. The number of setups in the optimal solution was 5, 5, and 7 for Hyperboloid, Sine, and Concave bowl objects, respectively. The branch pruning heuristic leads to faster computation time by terminating infeasible branches in the search tree. The impact of the branch pruning heuristic can be seen in Table 5.5. It significantly reduced the number of node expansions in the search tree. On the other hand, the branch guiding heuristic is crafted to help the search algorithm find setups with higher cleaning rates first. This helps to find a feasible branch faster. Keeping both the heuristics active leads to quickest convergence in search.

5.4.2 Comparison of Algorithm 2 and Algorithm 4 with two different bounds on future cost

Table 5.6 summarizes the performances of Algorithms 14 and 9 under different setup times with two different bounds on future costs. In these experiments, candidate setups were generated using uniform sampling without gradient descent. The computation was performed on a single core of a machine with an Intel Xeon E3-1241 3.50 GHz processor and 8GB RAM. We see that larger setup time leads to higher node expansions, hence slower convergence. This is because when setup time is much greater than the execution time, then the future cost based branch-pruning heuristic becomes less effective. As described in Section 5.3.3.2, we also see that the performance of Algorithm 14 varies with setup time. We see that it is more likely to get sub-optimal solutions with lower setup time. Also, setting $\mathbb{T}^b(\mathscr{P}_r)$ using (5.7) leads to faster convergence overusing (5.2). This can guarantee optimality with probability ≤ 1.0 for proper choice of N_{repeat}. We can

see that the combination of Algorithm 9 and (5.7) can lead us to an optimal solution with a good probability at a much faster rate.

Figure 5.6 guides us in selecting N_{repeat} such that Algorithm 10 guarantees optimality with probability ≥ 0.90. Since we do not know the size of minimum set cover q beforehand, we need to compute the probability for different q for different N_{repeat}. If the cardinality of the set of candidate setups $m = |\mathcal{S}|$, then we need to numerically test the probability for $q = 1, 2, \ldots, m-1$ for different values of N_{repeat}.

Suppose we want Algorithm 10 to give us a correct estimate with probability 0.90. Then for each m and q, we can evaluate the probability using (5.5) by gradually increasing N_{repeat} and stopping when probability $\geq$0.90 is reached. We should pick the highest N_{repeat} that we found among different q for a fixed m. From Fig. 5.6(a, b, c, d), we can see that N_{repeat} vs q forms a bell curve and N_{repeat} is highest for q close to $m/2$. From Fig. 5.6(e, f, g, h), we can see for what values of N_{repeat} the probability converges to 1.0. The fastest rising curves in Fig. 5.6(e, f, g, h) are for $q = 1$ *and* $m - 1$, and the slowest rising curves are for $q = \frac{m}{2}$. Therefore, for a given m we can evaluate (5.5) for $q = \lceil \frac{m}{2}$ by gradually increasing N_{repeat} and stopping when our desired probability threshold is reached.

Table 5.5: Comparison of heuristics for setup planner

Object	**Branch Pruning Heuristic**	**Branch Guiding Heuristic**	**Node expansion in search tree before convergence**	**Percentage of node expansion w.r.t. OFF-OFF**
Hyperboloid	OFF	OFF	3138	100 %
	OFF	ON	3104	98.92 %
	ON	OFF	679	21.64 %
	ON	ON	569	18.13 %
Sine	OFF	OFF	1188175	100 %
	OFF	ON	1176109	98.98 %
	ON	OFF	39822	3.35 %
	ON	ON	20286	1.71 %
Concave Bowl	OFF	OFF	16260590	100 %
	OFF	ON	16242862	99.89 %
	ON	OFF	362208	2.23 %
	ON	ON	353630	2.17 %

Table 5.6: Comparison of Algorithm 2 and Algorithm 4 with two different bounds on future cost

Bound on Future Cleaning Time			$\mathbb{T}^{lb}(\mathscr{P}_r)$			$\mathbb{T}^{b}(\mathscr{P}_r)$			$\mathbb{T}^{b}(\mathscr{P}_r)$		
AddSetUp Algorithm			Algorithm 14			Algorithm 14			Algorithm 9		
	Non Dominated Setups	Setup Time (s)	Node Expansion	Solution Size	Time (h:m:s)	Node Expansion	Solution Size	Time (h:m:s)	Node Expansion	Solution Size	Time (h:m:s)
Hyperboloid	7	0	513	5	00:00:40	132	7	00:00:26	67	5	00:00:22
		60	569	5	00:00:40	168	6	00:00:27	234	5	00:00:25
		3000	1327	5	00:01:02	1067	5	00:00:53	1101	5	00:00:54
Sine	12	0	12531	9	00:07:55	229	11	00:00:30	2377	5	00:00:57
		60	20286	6	00:12:53	2200	5	00:01:07	2799	5	00:01:10
		3000	97124	6	00:47:57	85795	5	00:41:51	85795	5	00:42:33
Concave Bowl	13	0	353630	7	03:04:05	108	10	00:00:31	2157	7	00:01:19
		60	484371	7	03:35:15	3110	7	00:02:49	25677	7	00:09:12
		3000	1283388	7	06:19:42	1002333	7	04:37:58	1135870	7	04:12:10

Table 5.7: Results from physical experiments

Part	**Solution Size**	**Node Expansion**	n_{rfc}	**Avg. Setup Time(s)**	**Setup Planning Time(s)**
Bowl	5	52	360	59.31	58
Hull	2	8	192	46.12	32

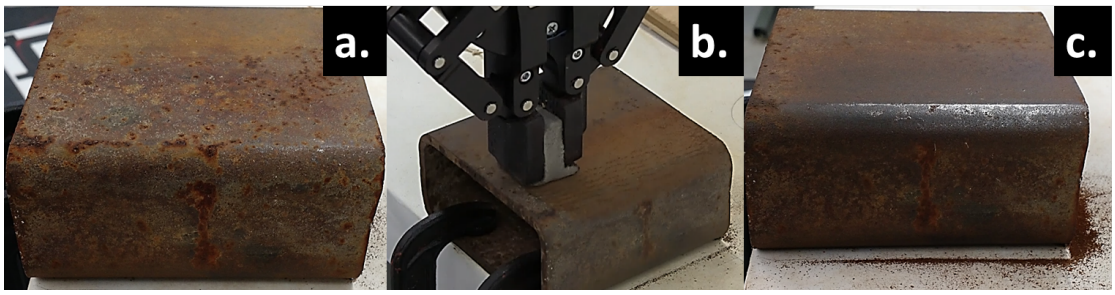


Figure 5.11: Robotic cleaning of a rusty surface. (a) Before cleaning. (b) An ongoing cleaning pass. (c) After seven cleaning passes.

5.4.3 Physical Test Results

We conducted robotic cleaning experiments with two Kuka iiwa 7 manipulators. One robot immobilized or held the part, and the other robot manipulated a cleaning tool. We used a plastic bowl and a 3D printed model of ship hull as curved surfaces. The bowl had a convex surface. The ship hull model had convex, concave, and flat regions on the surface. A human operator performed the setup changes manually for the bowl. The ship hull model was mounted on one robot arm to perform setup change automatically. We applied acrylic paint and mud on them as a surrogate for stains, as shown in Fig. 5.7 and 5.9 respectively. We applied different layers of stain on different regions to obtain nonuniform stain intensity. The bowl and hull surfaces were represented as a mesh of 15,644 and 22,764 triangles. The stain region consisted of 1,000 and 5,709 triangles. Each triangle had a surface area of less than 10 mm^2. The dimensions of the bowl were $21.5mm \times 21.5mm \times 9.5mm$. The dimensions of the hull model were $20.1mm \times 18.0mm \times 14.0mm$. For the bowl, we used the following values for the robot motion parameters: (1) f_x=7 Hz (2) κ_x, κ_y=5000 N/m, (3) A_x, A_y=30 N, (4) v=0.3 (where maximum joint velocity v_{max}=1), and (5) f_n=10 N. For the hull, we used the following values for the robot motion parameters: (1) f_x=6

Hz (2) κ_x, κ_y=5000 N/m, (3) A_x, A_y=35 N, (4) v=0.1 (where maximum joint velocity v_{max} =1), and (5) f_n=20 N. These parameters were found to be optimal by using the method described in chapter 7. In the learning method, we used C_p^{th}=0.60 and P^{th}=0.10 for the bowl. For the ship hull model, we used C_p^{th}=0.90 and P^{th}=0.30. For the ship hull, we considered a two pass cleaning, the first pass for dry cleaning by scrubbing; the second pass with a wet sponge to remove the dust particles.

The candidate setups were generated by uniformly sampling the 3D configuration space (x, y, α, where α is the angle about z axis). We used CA_XY type gradient descent on top of hierarchical uniform sampling as they performed well in the synthetic tests. We used our setup planner described in Algorithm 14 to generate a solution. Table 5.7 summarizes computational data of the setup planner from the physical experiments. For the bowl, it took three and two cleaning passes for setup 1 and all other setups, respectively. For the hull, it took a single cleaning pass for all the setups.

Fig. 5.7 and Fig. 5.9 illustrate the before-and-after cleaning states of the bowl and hull respectively. Fig. 5.8 and Fig. 5.10 illustrate snapshots from a video of execution of the cleaning experiments. We also tested our system on cleaning rust on a metal block. The target region to be cleaned was user-defined in this experiment. Figure 5.11 illustrates the cleaning performance on a setup. The metal piece started to shine after seven cleaning passes. The operation parameter selection method ensures the selection of the right set of operating parameters to achieve the desired cleaning performance for each cleaning pass. In this work, we referred to the required number of cleaning passes as N_c. In our current implementation N_c is user-defined. If the estimation of N_c is correct, then there will be no residue stain. If the estimation of N_c is lower than what is needed, then there will be stain residue. The perception system tells the robots to stop cleaning when there is no more stain residue, even if N_c was over-estimated initially. A supervised learning method can be applied to estimate N_c accurately from the experimental data. However, in our current implementation, we evaluate the cleaning performance on the entire part

once the robot completes cleaning the part. If there is an error in estimating N_c, then we correct for it and run our setup planner again for the stain residues and iterate the cleaning process until there is no more stain left.

5.5 Summary

This chapter presented algorithms to bring setup planning considerations in manipulator trajectory planning. The presented heuristics enable the algorithm to provide a sequence of setups for the workpiece in a computationally efficient manner, such that the manipulator trajectory can be generated to complete the given task in a time-optimal manner.

Chapter 6

Bringing Task-Agent Assignment Consideration in Trajectory Planning

6.1 Introduction

Robotic agents need to be assigned appropriately, and trajectories need to be generated to carry out different tasks. The symbolic task-agent assignment without accounting for feasible motion of agents will lead to failure in task execution. Figure 6.1 shows an example. In this example, the bi-manual mobile-manipulator is an ensemble of three robotic agents (two manipulators and a mobile-base).

Task and Motion Planning (TAMP) algorithms [114] can generate a sequence of tasks to be performed by an agent and generate its configuration space trajectories. Most TAMP algorithms invoke the motion planning routine for each task assignment to an agent to validate the assignment. This approach is computationally viable when the query to the motion planning routine is not computationally intensive. However, in the case of agent assignments where multiple agents need to work collaboratively, and therefore motion planning problem has high degrees of freedom (DOF), each query to the motion planning routine becomes computationally expensive. It becomes challenging to find a motion plan for all the tasks with a small time budget.

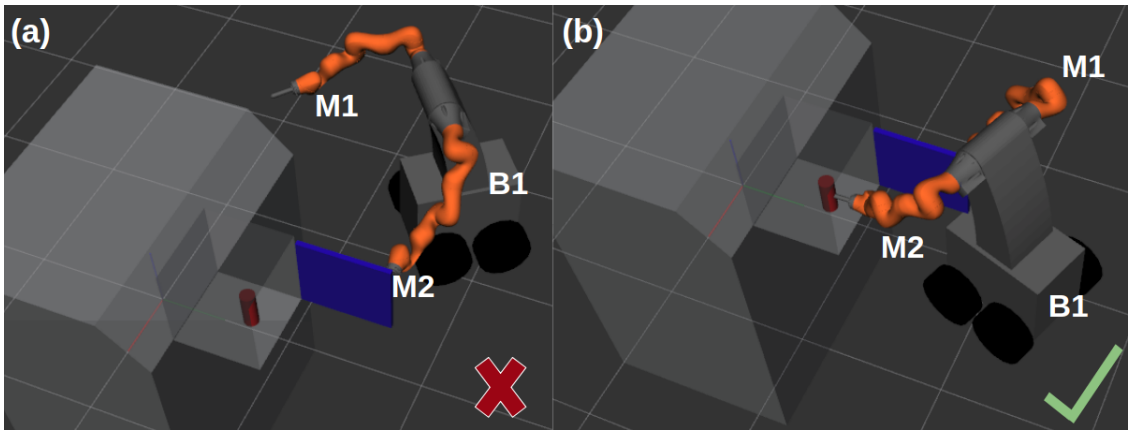


Figure 6.1: The three agents (M_1, M_2, B_1) need to cooperate to extract the part out of the machine and transfer it to the desired location. The door of the machine is *spring-loaded*. Therefore an agent needs to keep holding the door while another agent can extract the part. We need to assign appropriate agents and generate their configuration space trajectories to complete the operation. (a) An infeasible assignments of agents. There is no feasible configuration for the agents such that M_2 can hold the door while M_1 extracts the part. (b) A feasible assignment of agents. M_1 can hold the door while M_2 extracts the part.

This chapter proposes a three-layer architecture to integrate motion planning with task-agent assignments for operations that require high-DOF robotic agent ensembles. It assumes that a task network has been generated and given. The first layer assigns agents to each task and sequences the tasks for each agent for the given task network. The third layer generates motion for the agents. The work introduces a middle layer to check the spatial constraints quickly. This layer can efficiently check the feasibility of task-agent assignment without invoking the motion planner. The developed method uses infeasible assignment knowledge during reassignment to achieve high chances of feasibility. It adaptively selects the DOF of the mobile-manipulation system required to complete the assigned task, which further improves the computation time required for motion planning. Moreover, the algorithm uses a multi-tree representation for efficient caching during the search to store and reuse explored branches in the search trees. This enables the method to avoid repeated computations.

6.2 Problem Formulation

6.2.1 Definitions

Object: Let, O_i be an object. In this work, we will assume that O_i is a rigid body with a coordinate frame attached to it. We assume that the *state* of O_i is defined as its the pose ($p = x, y, z, q_x, q_y, q_z, q_w$ or $p = x, y, z, \alpha, \beta, \gamma$) with respect to the world coordinate frame (W).

Agent: Let, E be an ensemble of agents composed of a mobile-base (B_l) and one or more serial-link manipulators (M_j). We represent the configuration of B_l with its position (x, y) and orientation (ϕ) on the ground. We represent the configuration of M_j with its joint angles ($\theta_1, \ldots, \theta_n$, here n is the degrees of freedom (DOF)). In this work, we will consider each of the mobile-bases and serial-link manipulators as individual agents. We assume that the agents are equipped with appropriate end-effectors to manipulate objects.

Task: Let, T_k be a task. A task is defined by the state transition of an object. T_k, is associated with the pre-conditions (p_{pre}) and post-condition (p_{post}) of the state of one object. We assume that the associated object is available at its pre-condition state before the task starts. We assume that the type and number of agents required by the task are specified. One or more agents will carry out the task based on the requirement. The object will be transferred to the post-condition state after the agent/s have completed the task. We assume that the pre/post-conditions are given as sets of poses (i.e., $p_{pre} \in P_{pre}$, $p_{post} \in P_{post}$, $|P_{pre}| \geq 1$,$|P_{post}| \geq 1$). During task execution, we assume that a fixed-joint formation is required between an agent and an object whenever the agent operates on the object.

6.2.2 Task Network Modeling

Let, $\mathscr{T}$ be a task-network. There are one or more connected tasks in $\mathscr{T}$ that completes an *operation*. We assume that any two tasks (T_A, T_B) in $\mathscr{T}$ can have one of the following four interdependencies-

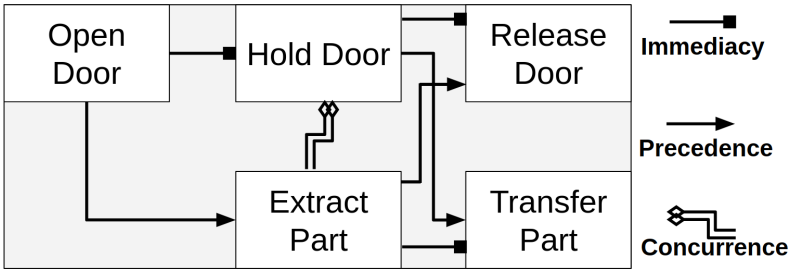


Figure 6.2: Task network for the example shown in Figure 6.1. We want to assign each task to an agent such that the operation is completed in minimum time. The double-diamond head in the concurrence edge indicates the container task.

(1) No interdependency

(2) Precedence: If T_A has precedence over T_B, then T_B cannot start until T_A has been finished. Based on the availability, we can assign the same or different agent for T_A and T_B.

(3) Immediacy: If an immediacy connection comes from T_A to T_B, then T_B has to start immediately after T_A. The state of the object at the end of T_A will be the state of the object at the start of T_B. Based on the object's state transition requirement, it may not be possible to assign different agents for T_A and T_B.

(4) Concurrence: If two tasks have concurrency, then they will have some overlap in the time domain during execution. We define *container task* and *embedded task* to explain concurrence more clearly. We define *container task* as the task that will have a longer duration and *embedded task* as the task that is overlapped by the *container task*. Let, T_A and T_B two concurrent tasks. Let, T_A be the *container task* task and T_B be the *embedded task*. By definition, T_B cannot start until T_A has started and T_A cannot be finished until T_B has finished. Therefore, the same agent cannot be assigned for the *embedded task* of a *container task*.

We consider each of the task given in $\mathcal{T}$ as a *core task*. We assume that there can be *supporting tasks* required to facilitate the core task. For example, a mobile-base agent (B_l) may need to assist a manipulator agent (M_j), whenever M_j is working on a task (T_k). Here, T_k is a *core task* and the assisting task for B_l is a *supporting task*.

Let us consider the example shown in Figure 6.1. A bi-manual mobile manipulator, comprising three agents (M_1, M_2, B_1), is available for carrying out the operation. The associated task-network is illustrated in Figure 6.2. There are five tasks in this task network. We want to assign agents to carry out these tasks and generate trajectories for the agents. As an example, we will summarize the attributes/requirements of the tasks below.

- OpenDoor:
  - Associated Object: Door.
  - Required Agent Type: Manipulator.
  - Pre-condition: $p_{pre} = p^{door}_{closed\ state}$.
  - Post-condition: $p_{post} \in P^{door}_{open\ states}$.
- HoldDoor:
  - Associated Object: Door.
  - Required Agent Type: Manipulator.
  - Pre-condition: $p_{pre} \in P^{door}_{open\ states}$
  - Post-condition: $p_{post} \in P^{door}_{open\ states}$
- ReleaseDoor:
  - Associated Object: Door.
  - Required Agent Type: Manipulator.
  - Pre-condition: $p_{pre} \in P_{pre} = P^{door}_{open\ states}$
  - Post-condition: $p_{post} = p^{door}_{closed\ state}$
- ExtractPart:
  - Associated Object: Part.

- Required Agent Type: Manipulator.
- Pre-condition: $p_{pre} = p^{part}_{inside\ machine}$
- Post-condition: $p_{post} \in P^{part}_{outside\ machine\ states}$

- TransferPart:
  - Associated Object: Part.
  - Required Agent Type: Manipulator.
  - Pre-condition: $p_{pre} \in P^{part}_{outside\ machine\ states}$
  - Post-condition: $p_{post} = p^{part}_{on\ table}$

From our definitions of *precedence*, *immediacy*, and *concurrence*, and the task requirements, we can see that HoldDoor (HD) has to begin immediately after OpenDoor (OD). ExtractPart (EP) can not start until OD has finished and HD has started. HD cannot finish until EP is finished. ReleaseDoor (RD) and TransferPart (TP) have to start as soon as EP and HD are finished. TP and RD can not start until HD and EP have finished.

We can see that the agent (M_1) that opened the door will need to keep holding the door. A different agent (M_2) will need to be assigned for extracting the part while M_1 keeps holding the door. After extracting the part, M_1 can release the door and M_2 can transfer the part. During the whole process, B_1 will need to move such that it can facilitate the motion requirements for M_1 and M_2. B_1 will also need to bring M_1 and M_2 near the machine at the beginning and take them to the next destination after completing the *operation*.

6.2.3 Problem Statement

For a given task network, we need to sequence the tasks to satisfy the constraints in the network, assign agents to the tasks, and generate their trajectories to carry out the tasks. The trajectories for the agents need to be generated such that the pre and post-conditions of each task in $\mathcal{T}$ are

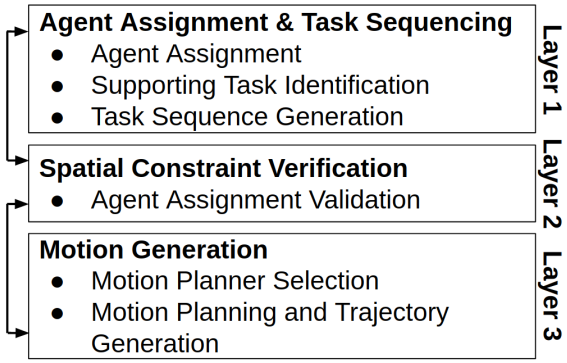


Figure 6.3: The three-layer architecture used by our approach.

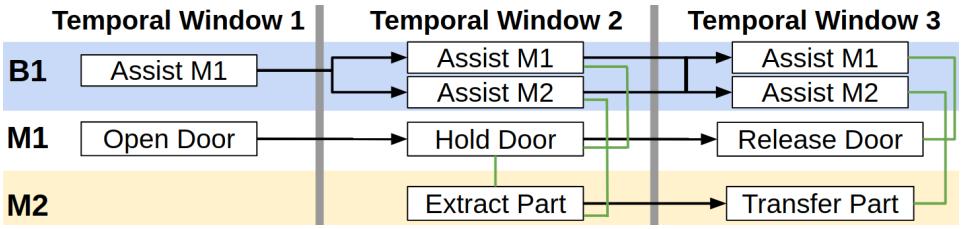


Figure 6.4: Example of a linear temporal task sequence generated by the first layer for the task network shown in Figure 6.2. The linear task sequence contains all the core tasks for M_1 and M_2, and potential supporting tasks for B_1. The black arrows represent the sequence of tasks for each agent. The green lines connect the tasks that will occur concurrently.

satisfied. Therefore, we need to generate continuous trajectories for the agents such that there is a smooth transition between the end of one task and start of the next. For certain tasks, it might be required to generate synchronous trajectories for the agents to complete the task in a feasible and time-efficient manner. For other tasks, it might be feasible to generate an independent trajectory for an agent without affecting the operation completion time.

Formally, given a task network $\mathcal{T}$ and an ensemble of agents E, our objective is to assign tasks to the agents, sequence the tasks, and generate continuous configuration space trajectories for each agent such that there is no conflict in the assignment and the time span of the operation is minimized.

6.3 Approach

Algorithm 11 AgentAssignmentAndMotionPlanning($\mathscr{T}$)

$\mathscr{T} \leftarrow$ RefineTaskNetworkbyAddingSupportingTasks($\mathscr{T}$)
$E \leftarrow$ Ensemble of Agents
$H \leftarrow$ Set of Heuristics
$L_n \leftarrow$ Number of Layers in Architecture
$\Theta^* \leftarrow empty$ \\Global container for Best Solution
$\Theta^*.cost \leftarrow \infty$
$t_{max} \leftarrow$ Maximum computation time
$W \leftarrow$ IdentifyTemporalWindows($\mathscr{T}$)
$T \leftarrow \{\}$ \\Container for Search Trees
for $i \in \{1, \ldots, |W|\}$ **do**
  for $j \in \{1, \ldots, L_n\}$ **do**
    $t_{i,j} \leftarrow$ InitializeTree$(i, j, \mathscr{T}, E, H, W)$
    $T \leftarrow T \cup t_{i,j}$
  end for
end for
ExpandTree$(t_{1,1}, \mathscr{T}, E, H, W, L_n, T, t_{max})$
return Θ^*

Algorithm 12 ExpandTree$(t_{i,j}, \mathscr{T}, E, H, W, L_n, T, t_{max})$

$H_g \leftarrow GetBranchGuidingHeuristics(t_{i,j}, H)$
$H_p \leftarrow GetBranchPruningHeuristics(t_{i,j}, H)$
while $ElapsedTime() \leq t_{max}$ **do**
  $node \leftarrow t_{i,j}$.PQ.pop()
  if !$node$.hasChildren() **then**
    break
  end if
  $children \leftarrow$ PopulateChildren$(node, \mathscr{T}, E)$
  $children \leftarrow$ SortChildren$(children, H_g)$
  for $child \in children$ **do**
    if isPruned$(child, H_p, \mathscr{T})$ **then**
      continue
    end if
    $t_{i,j}$.PQ.push($child$, $child$.cost)
  end for
end while
$T.update(t_{i,j})$
$flag, t_{i,j} \leftarrow$ IsBranchComplete$(node, t_{i,j})$
if !$flag$ **and** $i == 1$ **and**$j == 1$ **then**
  return
end if
if $flag$ **and** $j == L_n$ **and** $i == |W|$ **then**
  UpdateSolution(T, Θ^*)
end if
$nextTree \leftarrow$ SelectNextTree$(flag, t_{i,j}, L_n, T)$
ExpandTree$(nextTree, \mathscr{T}, E, H, W, L_n, T, t_{max})$

6.3.1 Overview of Approach

We have developed a method to solve the task assignment and motion planning problem for a given task network. Our method uses a three-layer architecture where each layer passes messages to the previous and/or next layer. The three layers in our approach are illustrated in Figure 6.3.

In the first layer, the method assigns agents to tasks and sequences the tasks. Initially, our method identifies if any supporting task might be required by the given core tasks in the task network. Then the method sequences all the tasks and groups them into temporal windows using a search. All the tasks in a temporal window have to be finished before any task in the next temporal window can start. In this work, we assume that all the core tasks specified in the task network will be performed by manipulators, and the supporting tasks will be for mobile-bases.

The method initiates a search (section 6.3.2) for agent assignment after grouping the tasks into temporal windows. An agent is selected for a task based on its availability and the task requirement. The *precedence*, *immediacy*, and *concurrence* relationships are used for branch guiding and pruning during agent assignment. Our method assigns agents to each task and creates a linear temporal task sequence for each agent. This task sequence is used to validate the agent assignment in layer 2 and generate trajectories for the agents in layer 3. Figure 6.4 illustrates an example of a linear temporal task sequence generated by our method.

In the second layer, our method validates the agent assignment using spatial constraint checking. This layer does not generate motion plans for the agents. Instead, it focuses on quickly eliminating an infeasible task-agent assignment and moving back to the first layer for re-assignment. In this layer, our method samples some static key-frames based on the pre-condition and post-condition requirements on the state of the objects associated with the task. The static key-frames are essentially sampled feasible states for the objects from the task description. The method determines if the assigned agent can carry out the task by evaluating the reachability of the agents

to identify feasible configurations for the agent at the key-frames. Section 6.3.3 discusses the details about the middle layer of our approach.

The third layer generates motion for the agents to carry out the tasks. Our method analyzes the feasible configurations found for the key-frames in layer 2, identifies agent interaction, and determines what kind of motion planner is required to generate the trajectory of the agents. If a motion planner fails to generate a feasible plan, then this layer moves back to the second layer to identify a different combination of agents to generate motion for, or it moves to the first layer to re-assign agents. Section 6.3.4 discusses the details about the motion generation layer of our approach.

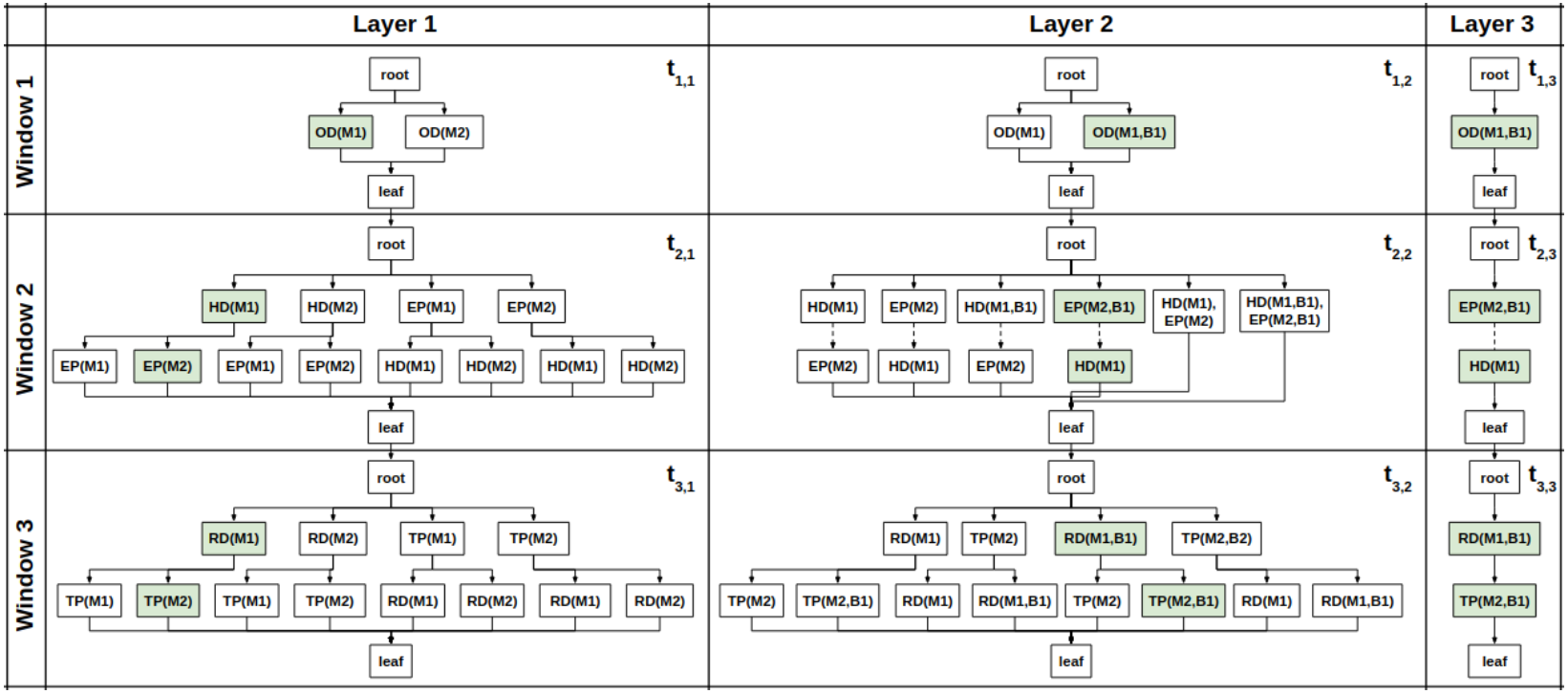


Figure 6.5: Example of completely expanded search trees at different temporal windows of the three layers for the example in Figure 6.2.

6.3.2 Description of Algorithms

We have developed an anytime algorithm to solve the task-agent assignment and motion planning problem using the three-layer architecture shown in Figure 6.3. In each layer of the architecture, our method creates a search tree for each temporal window. An example is shown in Figure 6.5 with all the search trees in their completely expanded states.

Each search tree ($t_{i,j}$) is labeled by its window index (i) and layer index (j). Our method generates feasible branches for each tree using a depth-first search. The branch guiding and branch pruning heuristics are domain-specific and different for each layer of the architecture. Moreover, the search is constrained to ensure that a smooth transition is possible between the end (or leaf node) of a tree and the start (or root node) of the next window's tree in the same layer. The search starts with the first temporal window of the first layer and continues through the last temporal window of the last layer. Having individual trees for each temporal window in each layer enables the method to cache previously explored branches and reuse them. This, in turn, enables to avoid repeated computation and unnecessary exploration. We will describe the idea of using the example shown in Figure 6.5.

Figure 6.5 illustrates the completely expanded search trees for each of the temporal windows. The subtasks are not shown in $t_{1,1}, t_{2,1}, t_{3,1}$ since there is only one possible agent assignment for them (i.e., the mobile-base). The green highlights illustrate an example of a feasible search result. The root and leaf nodes in these figures are virtual nodes, which directly connect to the selected nodes in the first and last stage of each tree, respectively.

In the beginning, the search starts with the first window of the first layer, $t_{1,1}$. Once a feasible branch is found for $t_{1,1}$, the method moves to find a feasible branch for $t_{2,1}$. Agent assignment is complete after finding a feasible branch for $t_{3,1}$. Then the method moves to the second layer to verify the spatial constraints for the task assignments. Let us assume that M_2 was selected for

the task OD in $t_{1,1}$. No feasible branch will be found in $t_{1,2}$ if M_2 is not suitable for the task. Then the method will directly move back to $t_{1,1}$ to find an alternative assignment (M_1) for OD.

Now, let us assume that the search trees are complete in the second layer and currently, $t_{2,3}$ is being explored. The feasible branch selected in $t_{2,2}$ determined that motion needs to be generated for M_2 and B_1 first to carry out the task EP. Then, the motion will be generated for M_1 to carry out the container task HD, considering the motion of M_2 and B_1 as constraints. Let us assume that this selected sequence for motion generation failed in $t_{2,3}$. The method will then directly move back to the tree in the corresponding window in layer 2 ($t_{2,2}$) and look for alternative coupling between agents for motion generation. Once an alternative coupling is found in $t_{2,2}$ (e.g., synchronous motion generation for M_1, M_2, B_1 for EP and HD tasks), the method will move back to $t_{2,3}$ to run motion planner and generate motion. In this example, the method did not have to explore $t_{3,2}$ and $t_{1,3}$ again since the new outcome of $t_{2,2}$ does not have any effect on them.

Algorithm 11 takes the task network ($\mathscr{T}$) as input, initializes the search trees, and makes calls to algorithm 12 to find feasible agent assignment and generate the trajectories (Θ^*) for the agents. In algorithm 11, line 1 refines $\mathscr{T}$ by identifying subtasks and adding the subtasks to $\mathscr{T}$. Line 2-7 initializes agents, heuristics, and other variables. Line 8 identifies the temporal windows based on the task network. Line 9-15 initializes the search trees for each window of each layer. Line 16 starts the search procedure.

In algorithm 12, line 1-2 selects the domain-specific branch guiding and pruning heuristics for the current search tree. Line 3-16 performs a depth-first search using these heuristics and a priority queue to identify a feasible branch. Line 17 updates or caches the results and the current states. Line 18 checks if the new branch found by the search is complete and feasible. The search immediately terminates (line 19-21) if no feasible branch is found for $t_{1,1}$ (i.e., no feasible agent assignment is possible for the tasks in temporal window 1). If the search has reached the last window of the last layer, then a complete solution is generated, and the best solution is updated in

line 22-24. Line 25-26 identifies the appropriate search tree to transition to and starts expanding it.

6.3.3 Spatial Constraint Checking

Given a task-agent assignment, a planner needs to verify the feasibility of the motions that need to be performed by the agents to achieve the assigned task. One of the methods to achieve this is by computing a complete motion plan for the multi-agent system as a whole and evaluate the feasibility of the current task-agent assignment. However, making a motion planning query for each task-agent assignment is computationally expensive for high-DOF systems, especially when agents are interacting. We utilize the middle layer of our architecture as a filter to quickly identify infeasible assignments and reduce queries to the motion planners for validation of task-agent assignment.

In order for executing any assigned task, each agent has to execute a continuous motion. During this motion, an agent have to move the interacting object through the key-states, which will enable the agent to complete the motion successfully. For example, let us consider the OpenDoor task as described in section 6.2. The pre-condition is a static pose of the door in the closed state. The post-condition is a pose from a set of open state poses of the door. To be able to open the door, the assigned agent/s at-least need to have a few feasible configurations such that it can be in contact with the door at its closed state and some feasible open states.

The developed planner uses the domain-specific knowledge to identify these key-states and checks for kinematic feasibility. If these key-states belonging to the agents' motion are not kinematically feasible, then the chances of successful motion plan for achieving the current task is quite low. Thus, it enables the elimination of a large number of queries to the motion planning layer.

Our method samples a few key-states of the object from the task description. We refer to these sampled object states as key-frames for which the second layer needs to identify feasible

configurations of the agents. Our method picks all the static object states from the pre/post-conditions. It samples multiple states when the pre/post-condition is a range of states or an element from a set of feasible states. If the postcondition of open door requires the yaw angle of the door to be between $10 - 120$ degrees, then our method will sample multiple yaw angles as key-frames within this range. It will check the reachability of the assigned agent for each of these sampled key-frames. If a task requires moving an object from one static pose to another, then our method generates a rough motion plan for the object alone by running a sampling-based planner [33] or by linearly interpolating the object's poses between start and goal. The key-frames are then taken as some sampled waypoints from the object's motion.

If multiple tasks are being done by different agents concurrently, then our method performs the reachability check with a different combination of the task-agent pairs to verify assignment feasibility. Feasible configurations for the first set of agents is considered as a spatial constraint while finding the configuration for the next set of agents. In other words, the feasible configurations for the next set of agents need to be collision-free with the first set.

The method verifies if help from the supporting agent/s is required by the core agent/s for a given task. The method starts by evaluating the reachability of the core agent/s. If the reachability fails, then the method evaluates the reachability for the core agent/s and supporting agent/s together. Essentially, the method starts the reachability analysis using a low-DOF system, and incrementally increases the DOF until feasible configurations of the agents are found to accomplish the task. If none of the combinations of core and supporting agents can perform a task, then the method moves back to layer 1 for reassignment.

As shown in layer 2 of Figure 6.5, the search algorithm will consider satisfying reachability for the key-frames using the assigned manipulator or the combination of base and manipulator/s. The computation time of motion planning increases with the DOF of the robotic system. Assigning a lower DOF to perform a task reduces the motion planning time. Therefore, we consider the total DOF of the agents for a task to be the branch guiding heuristic for the search trees in the

second layer. For illustration, the nodes in the search trees in layer 2 of Figure 6.5 are sorted (left-to-right) using this heuristic.

The tasks that are independent of each other can be ordered in any sequence. The method identifies a feasible sequence from the different permutations and produces a feasible branch in the search tree. However, by our definition of *concurrence*, the *container task* can not start after or end before the *embedded task*. Therefore, we cannot generate the motion of the agents for a container task before generating the motion for the embedded tasks. For example, in $t_{2,2}$ of Figure 6.5, branches emerging from $HD(M_1)$ and $HD(M_1, B_1)$ will be pruned by this policy. After generating the motion of the agents assigned to the *embedded task*, we can generate the motion of the agent for the *container task*. The motion of the agents associated with the *embedded task* will be considered as a constraint during motion planning for the *container task*. For example, in $t_{2,2}$ of Figure 6.5, the third and fourth branch from the left has this situation. For either of these branches, there will be two motion planners invoked in the corresponding tree ($t_{2,3}$) of the third layer. Each of them will be generating motion for relatively lower DOF system as compared to the fifth and sixth branch.

Verification through Capability Map: We use capability map to perform collision-free reachability test if a single redundant manipulator is assigned to perform the task [186]. A capability map is a pre-computed voxel grid, where a single query to each voxel returns if the centroid of the voxel is reachable by the robot in position and within a range of orientations. A single query on a pre-computed capability map is significantly faster than solving IK numerically.

Verification through Inverse Kinematics: We use analytical IK-solver for non-redundant manipulators and numerical IK-solver for redundant systems (i.e., mobile-base and arm/s combination) to test the collision-free reachability at key-frames. If a task has to be performed by multiple manipulators, each attached to a different mobile-base, then we solve the IK for one manipulator/s and mobile-base combination at a time by keeping the configuration of others constant.

6.3.4 Motion Generation

In the third layer, our method adaptively selects the motion planners and generates motion for the agents. Each node in each of the search tree in this layer is composed of task-agents assignments. Based on the pre/post conditions of the task, our method determines whether a point-to-point ($p2p$) or constrained motion generation is required. For example, opening a door requires constrained motion generation and extracting/transferring a part requires point-to-point motion. The method selects the motion planners from a set of available planners. The method identifies the required combination of agents to carry out a task by analyzing the feasible configurations from the key-frames and selects the motion planner appropriately. In addition to generating motion of the agents at each node, our method generates motion for the transition between the end of a tree in one window and the beginning of a tree in the next.

In this work, we have used a search-based algorithm similar to [166] to generate $p2p$ motion of a single manipulator. The algorithm adaptively selects heuristics and motion primitives by analyzing the context. We have used sampling-based planners [33, 187] for the point-to-point motion of high-DOF systems, such as mobile base and arm combinations. We used an optimization-based method to generate constrained trajectories. We pose the constrained trajectory generation problem as a discrete parameter optimization problem and solve it using successive refinement [173]. We set the objective to be minimizing the trajectory execution time and constraints to be- (1) maintaining the constraints imposed by the task, (2) satisfying the kinematic and dynamic constraints of the robot, and (3) avoiding collisions.

Many tasks require complex motion synchronization among multiple agents. For high-DOF systems, we generate the synchronous trajectory of multiple agents together using the optimization-based approach. However, it can be computationally efficient to generate motion of an agent independently while keeping the previously generated motion of other agents as constraints. For

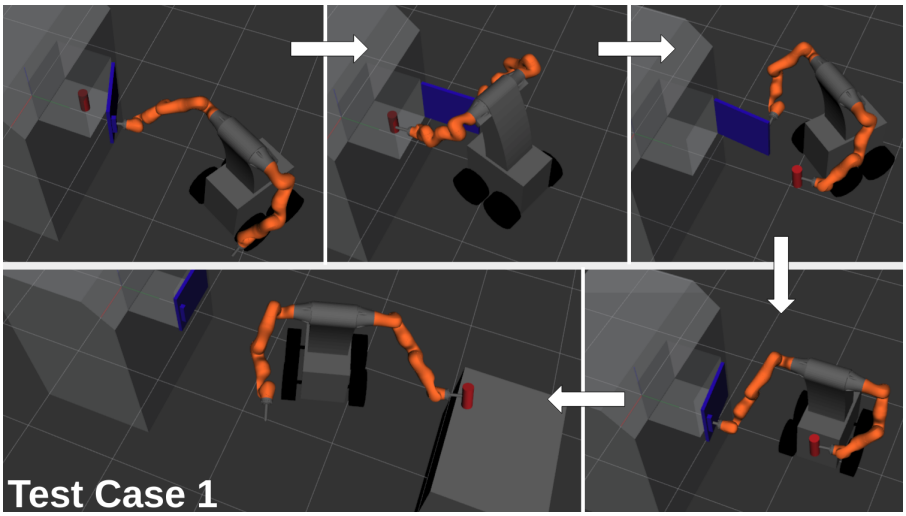


Figure 6.6: Snapshots from the test-case one and four. The mobile manipulator/s is/are tasked to get the part out of the machine and transfer it to a target location

Table 6.1: Impact of different features on computation time (s)

Spatial Verification	Caching	Test Case 1	Test Case 2	Test Case 3	Test Case 4
Inactive	Inactive	358.75	357.75	596.23	1189.25
Inactive	Active	323.69	322.75	516.98	1046.54
Active	Inactive	52.78	56.58	110.31	202.17
Active	Active	28.13	10.15	36.18	94.63

example, we can consider the fourth branch of $t_{2,3}$ in Figure 6.5. We can generate the point-to-point motion for the task EP using sampling-based planner for one manipulator and the mobile base. Once this motion is generated, we can consider it as a spatiotemporal constraint and generate motion for the other manipulator for holding the door.

6.4 Results

We have tested our method using bi-manual mobile manipulators on four challenging test cases. Three of the test cases used one bi-manual mobile manipulator (17-DOF system), and one of the test cases used a team of two bi-manual mobile manipulators (34-DOF system). There were 5, 4,

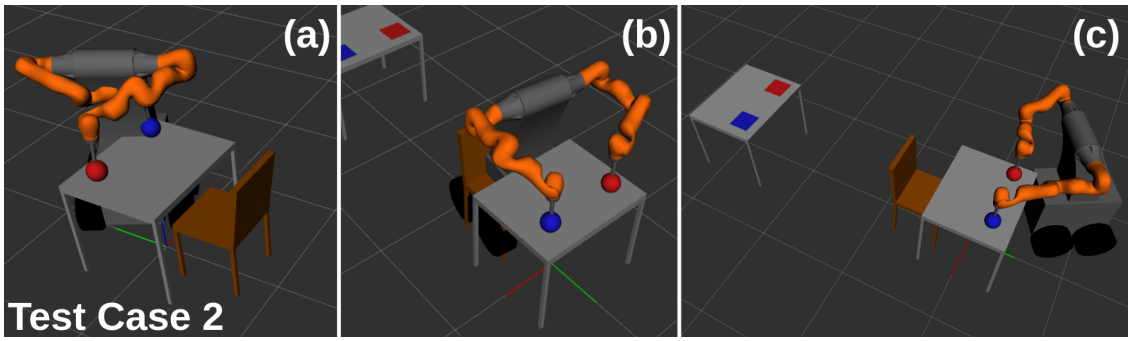


Figure 6.7: The mobile manipulator is tasked to pick-up the two balls and transfer them to goal locations. The makespan will be minimized if the agents pick up and transfer both the balls together. (a) and (b) shows some failed attempts of motion planner during pick up due to infeasible agent assignment. There is no collision-free pick-up configuration for this assignment. (c) Shows a feasible assignment.

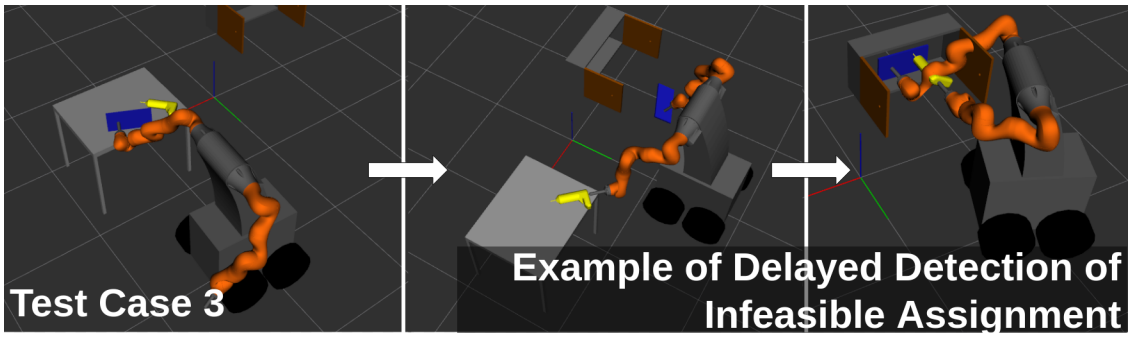


Figure 6.8: The mobile manipulator is tasked to open the panel and attach a new component on the board. This example shows that multiple motion plans can be generated for certain tasks before the infeasible agent assignment is detected during the motion planning of a task later in time. The spatial constraint checking layer reduces computation time by avoiding such cases.

9, and 16 tasks in the task network of test case 1, 2, 3, and 4, respectively. Some of the tasks in the task networks were possible to execute with a single agent, while others required synchronous motion of multiple agents. We have used KUKA iiwa 7 as the manipulator and Inspectorbot Mega Bot as the mobile-base in our experiments. The mobile-base was assumed to be holonomic. Figure 6.6 and 6.9 illustrates some screenshots from the trajectories generated by our method for test case 1 and 4 respectively. The details of the task networks and the generated trajectories are available in the attached video.

Figure 6.7 and 6.8 illustrates some infeasible assignments from test case 2 and 3 respectively. Multiple motion planners timed out before declaring that these assignments are infeasible when

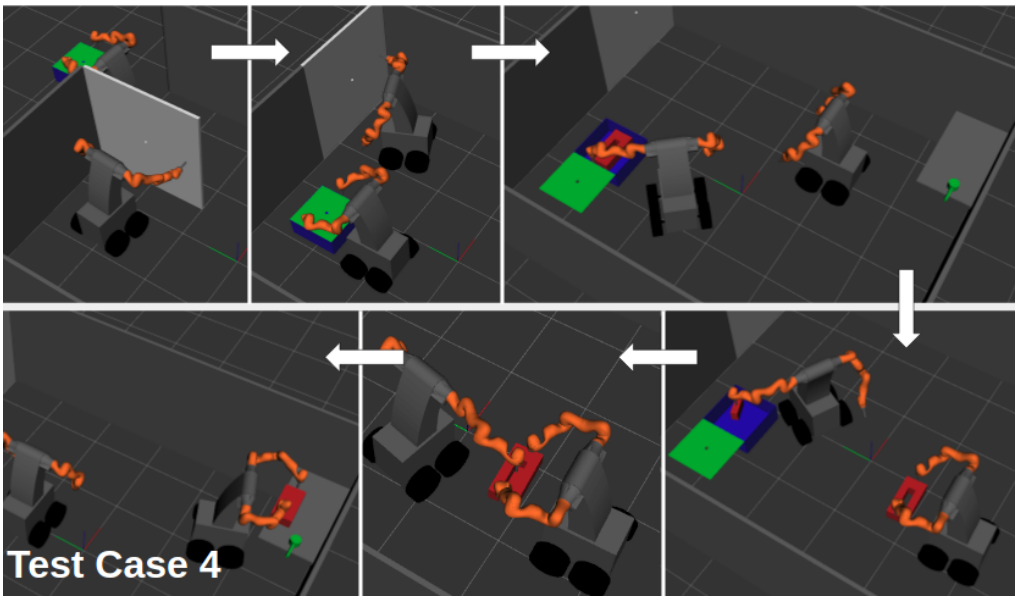


Figure 6.9: Snapshots from the test-case one and four. The mobile manipulator/s is/are tasked to collect and unpack a package of parts and collaborate to assemble them.

the spatial constraint checking was inactive. Using the spatial constraint checking layer, our method was able to avoid invoking motion planners for such assignments.

The algorithm was implemented using Matlab on a computer with an Intel Xeon 3.50GHz processor and 32GB of RAM. We set the total time limit for the agent assignment and motion planning algorithm to be 30 mins. For motion planners, we set the time limit to be 30s, 60s, and 90s for motions involving one, two, and three agents, respectively.

We have studied the performance of our method by switching on/off the spatial constraint checking layer and by switching on/off the caching scheme. The results are summarized in table 6.1. This table shows the computation time taken by the method to find the least-cost solution within the 30 mins time limit. We can see that spatial constraint checking reduced the computation time by 83.49% on average. Having the caching scheme with spatial constraint checking reduced computation time by a factor of 93.82% on average, as compared to the baseline approach.

6.5 Summary

This chapter presents a three-layer architecture for feasible agent assignment and motion planning for a given task network. The work shows that a spatial-constraint checking layer, between the agent assignment and motion generation layers, reduces computation times by reducing the number of calls to motion planners. Moreover, the method presents a multi-tree representation that enables an efficient caching scheme which prevents the search algorithm from exploring previously explored branches, reducing the computation time further.

Chapter 7

Identification of Trajectory and Process Parameters using Physical Experiments

7.1 Introduction

This chapter[1] focuses on process applications where the underlying physics-based process models are not known a priori. A large number of physical experiments need to be conducted to build a complete physics-based model. This can be done for large volume production. Building a complete model first and then using it for parameter optimization is not practical for low volume production or one of a kind job. Moreover, the complete model might not be reused due to the non-repetitive nature of tasks in small-volume manufacturing. This work devises a method that can identify the optimal operation and trajectory parameters while minimizing task completion time. This approach will have to focus on identifying optimal operation parameters instead of building an accurate physics-based model.

In process applications, the tasks are performed by robots manipulating a tool, which is in physical contact with a work-piece. The motion parameters (direction, speed, etc.) and the tool interaction parameters (stiffness, deflection, force, etc.) govern the trajectory followed by the

[1]The work in this chapter is derived from the work published in [179, 180, 188]

robot. The work is focused on identifying the optimal set of parameters for the robot trajectory using physical experimentation results.

There are three main considerations in processing tasks. The task objective is generally the first consideration. Most tasks require minimizing time and/or cost. We can express the task completion time and costs as a function of trajectory parameters. The task performance constraints are usually the second consideration. For example, a specific tolerance for matching operations, certain surface roughness, or minimum level of cleaning might need to be achieved. Usually, the mapping between the trajectory parameters and task performance variables is very complex. In this work, we assume that such a model is not known in advance. The third consideration is the constraint on part damage. It comes from the fact that the robot will be applying force on the part using a tool. For example, we want to prevent breaking or to scratch the part during the cleaning or polishing process. We can mathematically transform these three considerations into a constrained optimization problem with a known objective function and black-box constraint functions.

The work represents the black-box performance constraints using surrogate models based on Gaussian Processes. The uncertainties in the surrogate models are taken into account to identify the optimal parameters. This enables calculating the probability of candidate points to be optimal and predicting the parameter values that will have the maximum probability to be optimal. The algorithm iteratively proceeds by reducing uncertainties near the predicted optimal point. This allows it to converge quickly to the optimal point without the need for conducting an extensive number of experiments at other portions of the search space.

7.2 Problem Formulation

We want to find a solution to the constrained optimization problem by conducting a small number of experiments. We follow the following steps to formulate the problem:

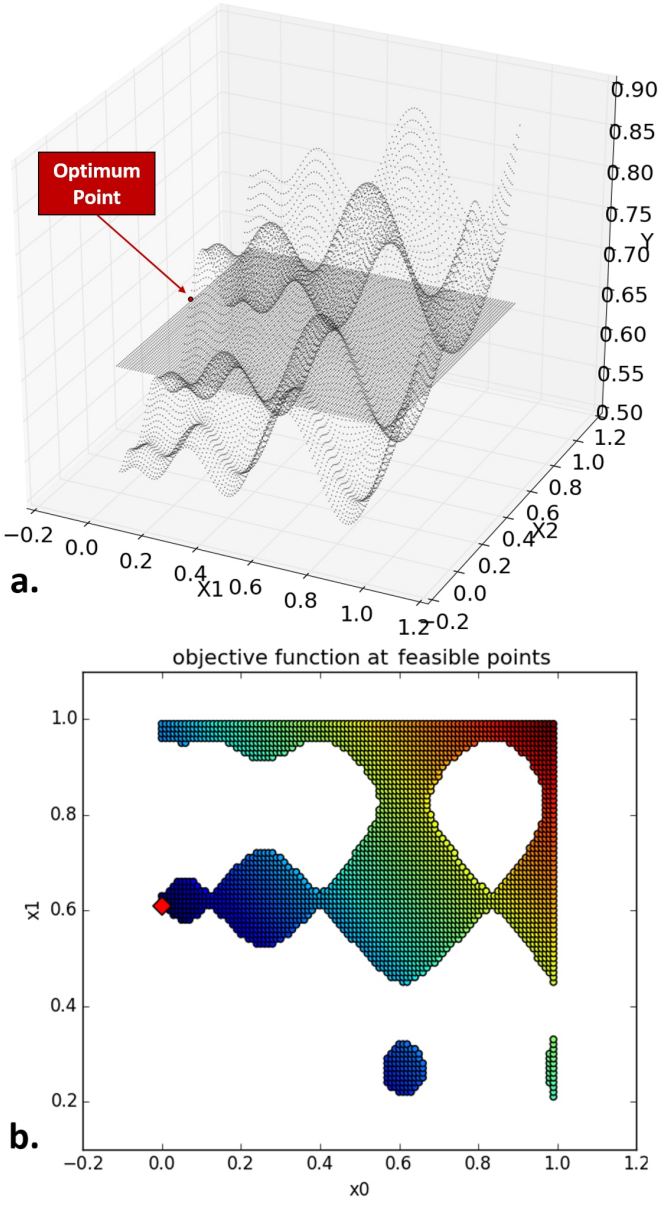


Figure 7.1: A 2D example of parameter optimization problem under black-box constraints. Here the objective is to minimize $f(x) = x_1 + x_2$. The black-box constraint function, $g(x)$, is a normalized 2D Schwefel function with different bias along each axis. The constraint is $g(x) \geq 0.65$. (a.) The 2D curved surface represents the constraint function $g(x)$. The flat plane represents the boundary $g(x) = 0.65$. Points on the surface of Schwefel function above the plane are feasible points. The optimal point is pointed with an arrow. (b.) Feasible points on a 2D plane. The values of the objective function at these points are represented with color. Darker blue represents the lower values of the objective function. The red marker indicates the optimal point.

1. Define the desired task performance targets. These will be expressed as constraints in our approach. For example, we may want surface roughness to meet or go below a certain given threshold.

2. Identify robot trajectory parameters (operation parameters) of interest. The task performances are unknown functions of these parameters.

3. Define the limits of the operation parameters. The robot capabilities and safety considerations will dictate the range of the robot trajectory parameters.

4. Formulate the task objective function in terms of the robot trajectory parameters. For example, we may be interested in performing the task at the minimum force and the highest possible speed. We express the task objective as a single function. Various sub-objectives are combined in that single function with proper weights.

The problem is mathematically formulated in (7.1). The task objective function, $f(x)$ is known. The task performance functions or constraint functions, $g_i(x)$, are black-box, i.e. they are known by their input-output behavior only.

$$
\begin{gathered}
min\ f(x) \\
s.t.\ x \in \mathbb{R}^d,\ a_j \leq x_j \leq b_j \\
a, b \in \mathbb{R}^d, j = 1, 2, \ldots, d \\
g_i(x) \geq 0,\ i = 1, 2, \ldots, n \\
f(x):\ known,\ g_i(x):\ black-box
\end{gathered}
\tag{7.1}
$$

A two-dimensional synthetic example of the problem is illustrated in Fig. 7.1. We are assuming that the unknown or black-box performance constraint ($g(x)$) is a normalized 2D Schwefel function with non-uniform bias along the axes. This function is represented by the curved surface

in Fig. 7.1(a.). We want to find a point which satisfies the task performance constraint, $g(x) \geq 0.65$ and minimizes the task objective function $f(x) = x_1 + x_2$. The points on the curved surface, which are above the flat plane are the feasible points. We want to locate the point among the feasible points that have a minimum value for $f(x) = x_1 + x_2$. Fig. 7.1(b.) shows the feasible points on 2D plane. The value of the objective function at these points is represented by colors. Darker blue represents the lower value of the objective function. The optimal point is indicated by the red marker.

7.3 Approach

7.3.1 Overview of Approach

Our method can be summarized in the following steps:

1. Select points in the task parameter space for initial exploratory experiments and conduct experiments at these points. These points can be selected by uniformly sampling the parameter space with a coarse resolution.

2. Fit surrogate models for the task performance constraints using the experimental data.

3. Identify candidate points from which the next point for the experiment will be selected.

4. Evaluate the objective function at the candidate points.

5. Estimate the probability of meeting the task performance constraints for the candidate points using the surrogate models.

6. For each of the candidate points, calculate its probability to be the optimal point among all the candidate points.

7. Select the point from the candidate points with the maximum probability to be optimum.

8. Conduct an experiment at the selected point. Generate new candidate points in the neighborhood of the picked point and remove this point from the list of candidate points. Use a finer sampling resolution for the new candidate points than the resolution that was associated with the experimented point.

9. Update the surrogate models with the performance data from the new experiment.

10. Go to step 4

Initial exploratory experiments: Our method identifies optimal parameters to meet the desired task performance constraints and optimizes the given task objective function. We use the uncertainty in the task performance surrogate models to select points in the parameter space to conduct an experiment. The initial points for exploratory experiments are uniformly sampled from the parameter space at a coarse resolution.

Let $\mathbb{X}_0$ and $\mathbb{X}$ be the set of the initial exploration points and set of tested points respectively. We add a point to $\mathbb{X}$ after conducting an experiment on it. $\mathbb{X}$ is updated as, $\mathbb{X} \leftarrow \{\emptyset\} \cup \mathbb{X}_0$ after completing the initial exploratory experiments. We keep track of the associated sampling resolution of each point. We select the initial sampling resolution to achieve full factorial design [189] of the exploratory experiments. Let, d be the dimension of the parameter space. The coarse sampling resolution, $r_{initial}$, is selected to have 3^d or $2^d + 1$ full factorial design for the initial exploratory experiments. The values we used for $r_{initial}$ in our experiments are summarized in table 7.1. In 2^d and 3^d full factorial designs the combinations of [‘low', ‘$high'$] and [‘low', ‘$medium'$, ‘$high'$] values along each axis are considered respectively. We are calling $2^d + 1$ full factorial design as the design where combinations of [‘low', ‘$high'$] values along each axis are considered and in addition, the midpoint of the space is considered. For example, if $d = 2$ then the points in $2^d + 1$ full factorial design will be the followings [‘low', ‘low'], [‘low', ‘$high'$], [‘$high'$, ‘low'], [‘$high'$, ‘$high'$], [‘$medium'$, ‘$medium'$]. 2^d full factorial design only covers the boundary

Table 7.1: Parameters values used in our design of exploratory experiments

Dimension of parameter space, d	n_c	$r_{initial}$	Design of exploratory experiments
$d < 5$	3	0.5	$3^d + 1$ full factorial
$5 \leq d < 9$	2	1.0	$2^{d-m} + 1$ fractional factorial $(d - m = 5)$
$d \geq 9$	1	1.0	$2^{d-m} + 1$ fractional factorial $(d - m = 5)$

points of the space. We combined the midpoint of the space with 2^d full factorial design to training the surrogate models with a better picture of the landscape.

For parameter space with dimensions 6, 7, 8, 9, and 10, the 2^d full factorial design will lead to 64, 128, 256, 512, and 1024 initial exploratory points respectively. We want to find the optimal solution by conducting a small number of experiments. Conducting exploratory experiments at every sampled point becomes infeasible for higher-dimensional parameter spaces. In such cases, we pursue $2^{d-m}+1$ fractional factorial design. We consider randomized fractional factorial design, i.e.; we select the 2^{d-m} points as a random subset of 2^d points from the full factorial design. We also include the midpoint of the entire space. In the experiments presented in this work, we decided not to conduct more than 33 initial exploratory experiments. Therefore, for $d \geq 5$, we selected m such that $d - m = 5$.

Candidate points for refining experiments after the exploratory experiments: We refer to the experiments after the exploratory experiments as refining experiments. This is because we refine our surrogate models at each experiment and actively select the optimal point in the expected sense for the next experiment. At each round of refining experiments, we select one point from the set of candidate points $\mathbb{C}_x$. The candidate points are uniformly sampled from the neighborhood of points in $\mathbb{X}$ (set of already tested or experimented points). Along each axis around the tested point, we consider n_c neighbor points in both positive and negative direction of the axis at a

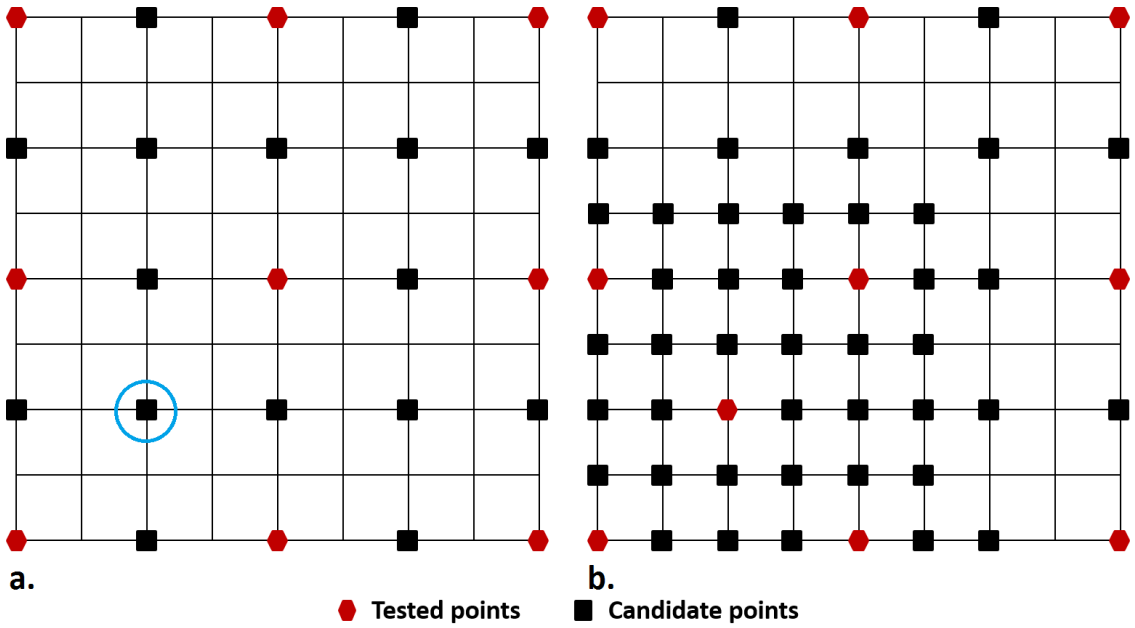


Figure 7.2: Tested and candidate points in a 2D parameter space. Here $n_c = 3$. Left: Tested points and candidate points after exploratory experiments. Suppose the candidate point marked by the ring was selected for the first refining experiment. Right: Tested points and candidate points after the first refining experiment.

finer resolution. Table 7.1 lists values of n_c and $r_{initial}$ we used in our experiments. Suppose we conducted an experiment at a point, x, which was originally sampled at a resolution of $r_x = 0.5$. Then we will sample points in the neighborhood of this tested point at a resolution of $r_x/2.0 = 0.25$ and add them to the set of candidate points, $\mathbb{C}_x$. If we conduct an experiment at one of these new candidate points, then we will use a sampling resolution of 0.125 to generate candidate points around the new experimented point. Fig. 7.2 illustrates examples of tested points and candidate points in 2D parameter space. The red points are the tested points, and the black points are the candidate points. Fig. 7.2(a) shows tested and candidate points after the initial explorations. Suppose, the candidate point marked by the blue ring was selected for the first refining experiment. Candidate points will be now generated around this point once an experiment is conducted. Fig. 7.2(b) shows updated candidate points and tested points after the first refining experiment.

Our algorithm determines the point to conduct a refining experiment based on its probability to be optimum among the candidates. The probability computation follows through Equations 7.2, 7.3, and 7.4. It relies on the uncertainty of outcome at the candidate points. Therefore by design, our algorithm avoids regions of the space where the uncertainty of the outcome is high. Now, if the uncertainty around the global optimum is high initially, the algorithm will take a larger number of iterations to converge. However, having uniformly sampled initial exploration points and generating candidate points around every tested point helps the algorithm to come out of a local optimum and go towards the global optimum as the surrogate models are refined in every iteration. At every iteration, we identify the point $x_{best} \in \mathbb{X}$, which minimizes the objective function and meets all the constraints. When x_{best} is updated at an iteration, the algorithm generates candidate points at a finer resolution around the new x_{best}. This helps to find a better solution locally.

Selecting point for refining experiments: We build surrogate models for the black-box performance constraint functions from the experimental data. We are using Gaussian Processes (GP) for the surrogate models. A distinct GP is trained for each performance constraint. Using GPR (Gaussian Process Regression) [157], we estimate the means and standard deviations in performances for the candidate points. From the mean and standard deviation we can calculate the probability of meeting the performance constraints, $\forall x \in \mathbb{C}_x$, using the following equations.

$$P(g_i(x) \geq 0) = \tfrac{1}{2}[1 - erf(\tfrac{-\mu_i(x)}{\sigma_i(x)\sqrt{2}})], \ where$$
$$erf(x) = \tfrac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt \tag{7.2}$$

$\mu_i(x)$ is the mean of predicted $g_i(x)$ and $\sigma_i(x)$ is the corresponding standard deviation

We compute the probability of meeting all the constraints $\forall x \in \mathbb{C}_x$. For example, if we have n constraints, then the probability of x_i meeting all the constraints,

$$\begin{aligned} P_i^c =& P(g_1(x_i) \geq 0) \times P(g_2(x_i) \geq 0) \times \\ & \ldots \times P(g_{n-1}(x_i) \geq 0) \times P(g_n(x_i) \geq 0) \end{aligned} \tag{7.3}$$

Let, $\mathbb{C}_x = \{x_1, x_2, \ldots, x_m\}$, where m is the number of candidate points. We keep the points in $\mathbb{C}_x$ sorted (low to high) in terms of the value of the objective function, i.e. $f(x_1) \leq f(x_2) \leq \ldots \leq f(x_m)$. If any subset of $\mathbb{C}_x$ happens to have the same value of objective function, then we sort them (high to low) based on their probability of meeting all the constraints. For example, if $f(x_i) \leq f(x_j) = f(x_k) = f(x_l) \leq f(x_m)$ and $P_k^c \geq P_j^c \geq P_l^c$, then the points get sorted as $\{x_i, x_k, x_j, x_l, x_m\}$. Sorting in this way is important because there might be regions in the parameter space where the objective function is a level set, but the constraint functions may vary in the same regions.

We determine the probability of every element of $\mathbb{C}_x$ to be optimum among elements of $\mathbb{C}_x$. Since the points in $\mathbb{C}_x$ are sorted in terms of the value of the objective function, the probability of x_1 to be optimum is same as its probability to meet all the constraints. For x_2, the probability to be optimum will be the product of its probability to meet the constraints and probability that x_1 is not optimum.

Let, P_i be the probability of $x_i \in \mathbb{C}_x$ to be optimum. Then,

$$P_1 = P_1^c$$

$$P_2 = P_2^c \times (1 - P_1)$$

$$P_3 = P_3^c \times (1 - P_1) \times (1 - P_2)$$

$$\dots$$

$$P_i = P_i^c \times \prod_{j=1}^{i-1}(1 - P_j)$$

$$\dots$$

$$P_m = P_m^c \times \prod_{j=1}^{m-1}(1 - P_j) \tag{7.4}$$

At each iteration we pick the point that has the maximum probability to be the optimal point and conduct one refining experiment. We repeat the process until we converge to a point that meets all performance constraints and optimizes the objective functions. After each refining experiment we update the surrogate models and set of candidate points. We remove a point from the set of candidate points after conducting a refining experiment on it.

We have described our constrained optimization method in Algorithm 13. At each iteration, Algorithm 13 makes call to Algorithm 14 and 15. Algorithm 14 performs uncertainty-based decision making to select the point for the refining experiment from the set of candidate points. We keep on accumulating more and more candidate points in each iteration. The computation time increases with the number of candidate points. Algorithm 15 prunes candidate points based on probabilistic decision making to reduce computation time.

Algorithm 13 takes the following inputs: (1) Dimension of the parameter space, d, (2) Lower and upper bounds on the parameter values, a and b, where $a, b \in \mathbb{R}^d$ s.t. $a_i \leq x_i \leq b_i$, $\forall x \in S$, $i = 1, 2, \dots, d$. $S \subset \mathbb{R}^d$ is the parameter space, (3) Sampling resolution for initial exploration points, $r_{initial}$. Algorithm 14 takes the following inputs: (1) Set of candidate points, $\mathbb{C}_x$, (2)

Array of probabilities for each candidate point to meet each constraint, G_{C_x}, and (3) Array of objective function values for each candidate point, F_{C_x}. Algorithm 15 takes the following inputs: (1) Set of candidate points, $\mathbb{C}_x$ and (2) Array of probabilities for each candidate point to meet all the constraints, P_c.

Computational Complexity: Let, M be the number of candidate points, and N be the number of tested points. The computational complexity of the probability computation for the candidate points is $O(M)$. We have used Gaussian Process Regression (GPR) for uncertainty estimation of the outcome at each iteration. GPR has $O(N^3)$ time complexity and $O(N^2)$ memory complexity, where N is the size of the training sample. Therefore the computational complexity of our method at each iteration is $O(n \times N^3)$, where n is the number of constraints.

Optimality in continuous space vs. discrete space: We have designed our method to find the optimal solution up to the resolution of the finest grid in the discrete search space. This solution may not coincide with the true global optimum in the continuous space. The user can set the resolution of the finest grid in the discrete search space. Therefore, our method can provide the optimum solution up to the grid resolution required for a specific application.

7.3.2 Algorithms

7.3.3 Theoretical analysis of convergence

We show that the strategy used in our algorithm to solve the problem (7.1) requires the minimum number of experiments in the expected sense.

Theorem 1. *Let $\mathbb{C}_x = \{x_1, x_2, \ldots, x_m\}$ be the set of m available candidate test points at an instance. Let P_i be the probability of $x_i \in \mathbb{C}_x$ to be optimum, where $\sum_{i=1}^{m} P_i = 1$. Let $x^* \in \mathbb{C}_x$ be the candidate point with the highest probability of being the optimum. That is, $x^* = x_j$ with $P_j \geq P_i \ \forall i \neq j$, where $i, j \in \{1, 2, \ldots, m\}$. Now, to solve problem (7.1) at any instance, if x^* is*

Algorithm 13 ParameterOptimization $(d, a, b, r_{initial})$

1: Initialize $\mathbb{X} = \{\emptyset\}$
2: $\mathbb{X}_0 :=$ {Set of uniformly sampled points in the parameter space}. The sampling resolution is $r_{initial}$.
3: $\mathbb{X} \leftarrow \mathbb{X} \cup \mathbb{X}_0$. Also, $\forall x \in \mathbb{X}$, keep record of associated sampling resolution.
4: $\mathbb{C}_x :=$ {Set of uniformly sampled points in the neighborhood of each $x \in \mathbb{X}$}. The sampling resolution is $r_{initial}/2$.
5: Evaluate $f(x)$ and $g_i(x)$, by conducting experiments $\forall x \in \mathbb{X}$. $i = 1, 2, \ldots, n$.
6: Find $x_{best} \in \mathbb{X}$, s.t. $g_i(x_{best}) \geq 0$, $i = 1, 2, \ldots, n$ and $f(x_{best}) = minf(x), \forall x \in \mathbb{X}$. If such x_{best} is found, then generate more candidate points at finer resolution around x_{best}. Use $r_{initial}/2.0$ as sampling resolution.
7: Train n GP models, one for each of the constraint functions, using values of $g_i(x), \forall x \in \mathbb{X}$. $i = 1, 2, \ldots, n$; where n is the number of constraints.
8: Evaluate $f(x), \forall x \in \mathbb{C}_x$. Let F_{C_x} be the array that stores these evaluated values. Each element of F_{C_x} is associated with a specific element of $\mathbb{C}_x$.
9: Using GPR, estimate the mean $(\mu_i(x))$ and standard deviation $(\sigma_i(x))$ of $g_i(x)$, $\forall x \in \mathbb{C}_x$. $i = 1, 2, \ldots, n$
10: Determine probability of meeting each constraint using (7.2), $\forall x \in \mathbb{C}_x$.
11: Let G_i be the array that stores values of $P(g_i(x) \geq 0), \forall x \in \mathbb{C}_x$, $i = 1, 2, \ldots, n$. Each element of G_i is associated with a specific element of $\mathbb{C}_x$. Let G_{C_x} be the array that holds $[G_1, G_2, \ldots, G_n]$.
12: $x_{test}, P_c \leftarrow PickTestPoint(\mathbb{C}_x, G_{C_x}, F_{C_x})$
13: Generate new candidate points at the neighborhood of x_{test}. The sampling resolution will be $r_{x_{test}}/2$, where $r_{x_{test}}$ is the resolution associated with x_{test}. Add these new candidate points to $\mathbb{C}_x$ and remove x_{test} from $\mathbb{C}_x$.
14: $\mathbb{X} \leftarrow \mathbb{X} \cup \{x_{test}\}$
15: $\mathbb{C}_x \leftarrow$ PruneCandidates$(\mathbb{C}_x, P_c)$
16: Evaluate $f(x_{test})$ and $g_i(x_{test})$, $i = 1, 2, \ldots, n$, by conducting experiment at x_{test}.
17: If $g_i(x_{test}) \geq 0$, $i = 1, 2, \ldots, n$ and $f(x_{test}) \leq f(x_{best})$, then $x_{best} \leftarrow x_{test}$. Generate more candidate points at finer resolution around x_{best}. Use $r_{x_{test}}/4$ as sampling resolution.
18: Terminate if x_{best} has converged, else go to step 8.

Algorithm 14 PickTestPoint$(\mathbb{C}_x, G_{C_x}, F_{C_x})$

1: $\forall x \in \mathbb{C}_x$, Compute the probability of meeting all the constraints. For example, the probability of x_i meeting all the constraints, $P_i^c = G_1(x_i) \times G_2(x_i) \times \ldots G_n(x_i)$. We described in algorithm 13 that $G_{C_x} = [G_1, G_2, \ldots, G_n]$.
2: Let, $P_c = [P_1^c, P_2^c, \ldots, P_i^c, \ldots, P_m^c]$ be the array of probability of candidate points meeting all the constraints.
3: Sort F_{C_x} (low to high).
4: Rearrange the elements of $\mathbb{C}_x$ to match their index with associated element in F_{C_x}. Let, the rearranged set be $\mathbb{C}_x = \{x_1, x_2, \ldots, x_m\}$, where $m = |\mathbb{C}_x|$. If a subset of $\mathbb{C}_x$ has the same value for the objective function, then sort (high to low) the points in that subset in terms of their probability to meet all the constraints (P_i^c).
5: Calculate the probability of each element of $\mathbb{C}_x$ to be optimum among all the elements of $\mathbb{C}_x$ using (7.4). Let, P_i be the probability of x_i to be optimum.
6: Let, $P_{max} = max\{P_1, P_2, \ldots, P_m\}$, where $m = |\mathbb{C}_x|$. Let, $x^* \in \mathbb{C}_x$ correspond to P_{max}.
7: Return x^*, P_c

Algorithm 15 PruneCandidates($\mathbb{C}_x, P_c$)

```
Initialize ϵ (Threshold on probability of meeting all the constraints)
Initialize μ (Threshold on fraction of candidate points to prune)
Initialize constants k1, k2
Find T_x s.t. ∀x ∈ T_x, P_x < ϵ, where P_x is the probability of meeting all the constraints for x.
  For x = x_i, P_x = P_c[i].
WHILE (|T_x| / |C_x| > μ):
    IF (ϵ < k1):
      μ ← μ/2.0
    ELSE:
      ϵ ← ϵ/2.0
    Find T_x s.t. ∀x ∈ T_x, P_x < ϵ
    IF (μ < k2):
      BREAK
∀x ∈ T_x, remove x from C_x
Return C_x
```

chosen for the experiment (to evaluate the black-box function), then the expected number of trials required for convergence to solution will be less than or equal to any scheme where $x(\neq x^*) \in \mathbb{C}_x$ is chosen for conducting the experiment.

Proof. Let us sort the points in the set $\mathbb{C}_x$ from high to low w.r.t. their probability of being optimum:

$$\text{If } P_1 \geq P_2 \geq \ldots \geq P_m \tag{7.5}$$

then the sorted sequence of points is

$$\{x_1, x_2, \ldots, x_m\}. \tag{7.6}$$

Let T be the number of trials required till convergence for the sequence in (7.6). If we use the selection scheme described in theorem 1, then the expected number of trials $E(T)$ to converge to solution at an instance is

$$E(T) = \sum_{i=1}^{m} i \times P_i \tag{7.7}$$

Let us consider the sequence $\{x_2, x_1, x_3, \ldots, x_m\}$. Let T' be the number of trials required till convergence for this sequence. We can come up with expected number of trials for this sequence $E(T')$ as following,

$$\begin{aligned} E(T') &= 1 \times P_2 + 2 \times P_1 + \sum_{i=3}^{m} i \times P_i \\ &= (2-1) \times P_2 + (1+1) \times P_1 + \sum_{i=3}^{m} i \times P_i \\ &= \sum_{i=1}^{m} i \times P_i + P_1 - P_2 \\ &= E(T) + P_1 - P_2 \\ &\geq E(T), \quad \because P_1 \geq P_2 \text{ from (7.5)} \end{aligned} \tag{7.8}$$

Let us consider another sequence where any two elements of the sequence in (7.6) is swapped. The expected number of trials $E(T')$ is given by

$$\begin{aligned} E(T') =& 1 \times P_1 + 2 \times P_2 + \ldots + q \times P_r \\ &+ \ldots + r \times P_q + \ldots + m \times P_m \end{aligned} \tag{7.9}$$

Now, $r > q$ and $P_q \geq P_r$ (from (7.5)). Therefore,

$$\begin{aligned} & q \times P_r + r \times P_q \\ &= (r - (r-q)) \times P_r + ((r-q)+q) \times P_q \\ &= r \times P_r + q \times P_q + (r-q) \times (P_q - P_r) \end{aligned} \tag{7.10}$$

Using (7.10) in (7.9),

$$\begin{aligned} E(T') &= E(T) + (r-q) \times (P_q - P_r) \\ &\geq E(T) \end{aligned} \tag{7.11}$$

Therefore, from (7.8) and (7.11), for any sequence $\{x'_1, x'_2, \ldots x'_m\} \neq \{x_1, x_2, \ldots, x_m\}$ we have $E(T') \geq E(T)$. Hence, theorem 1 is proved. □

In our implementation, we have used the probability estimate described in (7.4). If a better design for probability estimate is available, then it can be incorporated in our algorithm.

Now, the probability map over the set of candidate points will be updated after each black-box evaluation. We have proven theorem 1 for each instance or trial. Therefore, in the expected sense, we will always converge faster compared to any other method if we conduct an experiment at the candidate point with the highest probability to be the optimum at each iteration.

7.4 Results

7.4.1 Benchmarking on synthetic problems

Physical experiments can be expensive. We conducted experiments on synthetic test problems before physical experiments to validate our method. We benchmarked our method against two other established methods for solving optimization problems under black-box constraints.

We have selected some well established synthetic test problems for benchmarking from [190, 191, 192, 193, 194, 195, 196]. We replaced the equality constraints with inequality constraints. We considered the objective function to be known and the constraint functions to be black-box in all the problems. The description of our selected problems is in the Appendix. The dimensions of the parameter space and number of constraints for the synthetic test problems are summarized in Table 7.2.

7.4.1.1 Optimization methods for benchmarking

We have chosen methods for benchmarking that try to solve the problems of optimization under black-box constraints with a reduced number of experiments. We considered the NOMAD algorithm [1] and the Stochastic RBF (Radial Basis Function) based algorithm developed by Regis [2] for benchmarking our method. The NOMAD algorithm considers mesh adaptive direct search for optimization. This algorithm does not require to start from a feasible point.

RBF was used as surrogate models for the black-box functions in Regis' algorithm. His algorithm requires at least one feasible point in the set of initial experimentation points. Regis benchmarked his method against six optimization methods, including NOMAD [1]. He considered the objective and constraint functions to be black-box and tested his method on synthetic problems. His method converged to the solution with fewer iterations compared to the other methods. However, none of these methods involve decision making based on uncertainty in the surrogate model.

We modified Regis' method to solve for known objective functions to compare with our algorithm. For a fair comparison, we considered the same initial exploration points for all the three methods.

7.4.1.2 Results on synthetic test problems

Comparison of our method against Regis' algorithm and NOMAD algorithm is illustrated in Fig. 7.3 and 7.4. The figures illustrate the results of synthetic test problems. From the plots, we can see the convergence of the mean of objective function values at the best feasible point in each iteration. The plots only show the points after the initial exploratory experiments.

These problems were of dimension 2 to 10. There were $2^5+1 = 33$ initial exploration points for TP7-TP16. For TP1-TP3 there were $3^2 = 9$, for TP4 $2^3+1 = 9$ and for TP5-TP6 $2^4+1 = 17$ initial exploration points. The method of selecting points for exploratory experiments was described in section 8.2. We can see from the plots that our method took a lower number of experiments to converge to solution compared to the other methods on the synthetic test problems.

At every iteration, Regis' method looks for points for future experiments in a local neighborhood of the current best feasible solution. Also, neither NOMAD nor Regis' method consider uncertainty in the surrogate models while selecting points for the experiment. Our algorithm stores candidate points in the neighborhood of all the tested or experimented points. This helps our algorithm to select points from a set of candidates sampled over the entire parameter space.

This also helps to come out of a local optimum as the surrogate models keep refining themselves. Also, we consider the uncertainty in the models and select the points for experiments based on their expected probability to be optimum among all the candidate points. All these factors enable our algorithm to convergence faster compared to the other two methods.

7.4.2 Results on dense uniform grid for Schwefel function

To illustrate the progression of our algorithm, we conducted experiments on uniformly sampled fine resolution grid for the synthetic problem of 2D Schwefel function (1). Here we considered all the points on the grid to be candidate points, instead of hierarchically generating candidate points as described in algorithm 13.

Figures 7.5, 7.6, 7.7, and 7.8 illustrate the value of objective function, estimated constraint function, probability of meeting constraints, and estimated probability to be optimum at the grid points.

The objective function is known. Therefore it remains constant throughout the refining experiments. The estimated constraint function, probability of meeting constraints, and estimated probability of being optimum is updated at every refining experiment.

We have also plotted the tested points, next test point, best feasible point among Tested Points, and the true optimum on top of the color maps, to illustrate the progression of the algorithm at different iterations of refining experiments.

Our method does not focus on improving the probability map uniformly on the entire parameter space. From these figures, we can see that our method improves the probability map around the promising regions where it believes the true optimum is located. This approach helps to find the optimum with a low number of experiments.

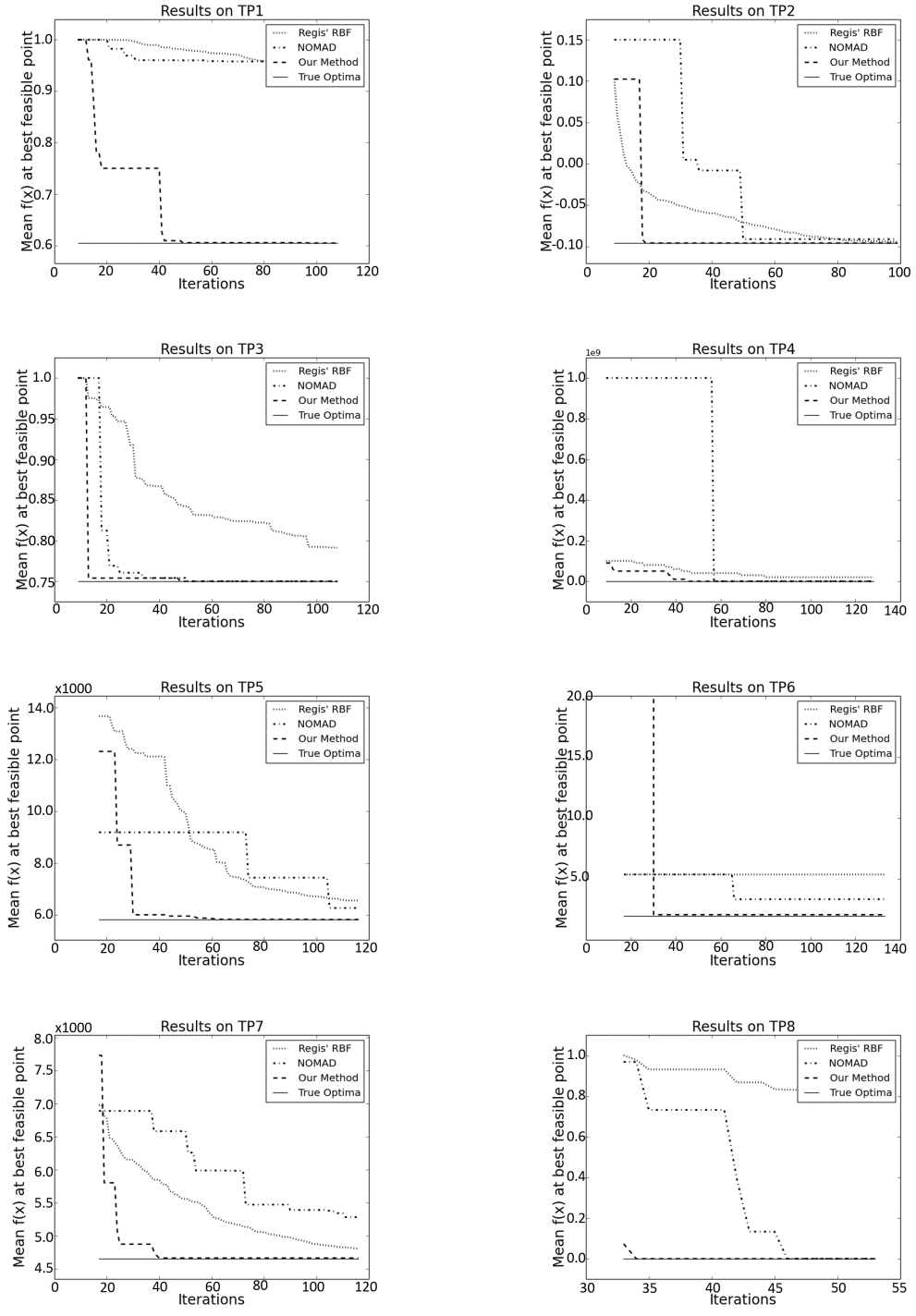


Figure 7.3: Comparison of our method with [1] and [2] on synthetic test problems (TP1-TP8). The plots show the mean objective function value at best feasible point at every iteration after the exploratory experiments. Our method, Regis' method, and NOMAD are denoted by green, red, and blue lines respectively.

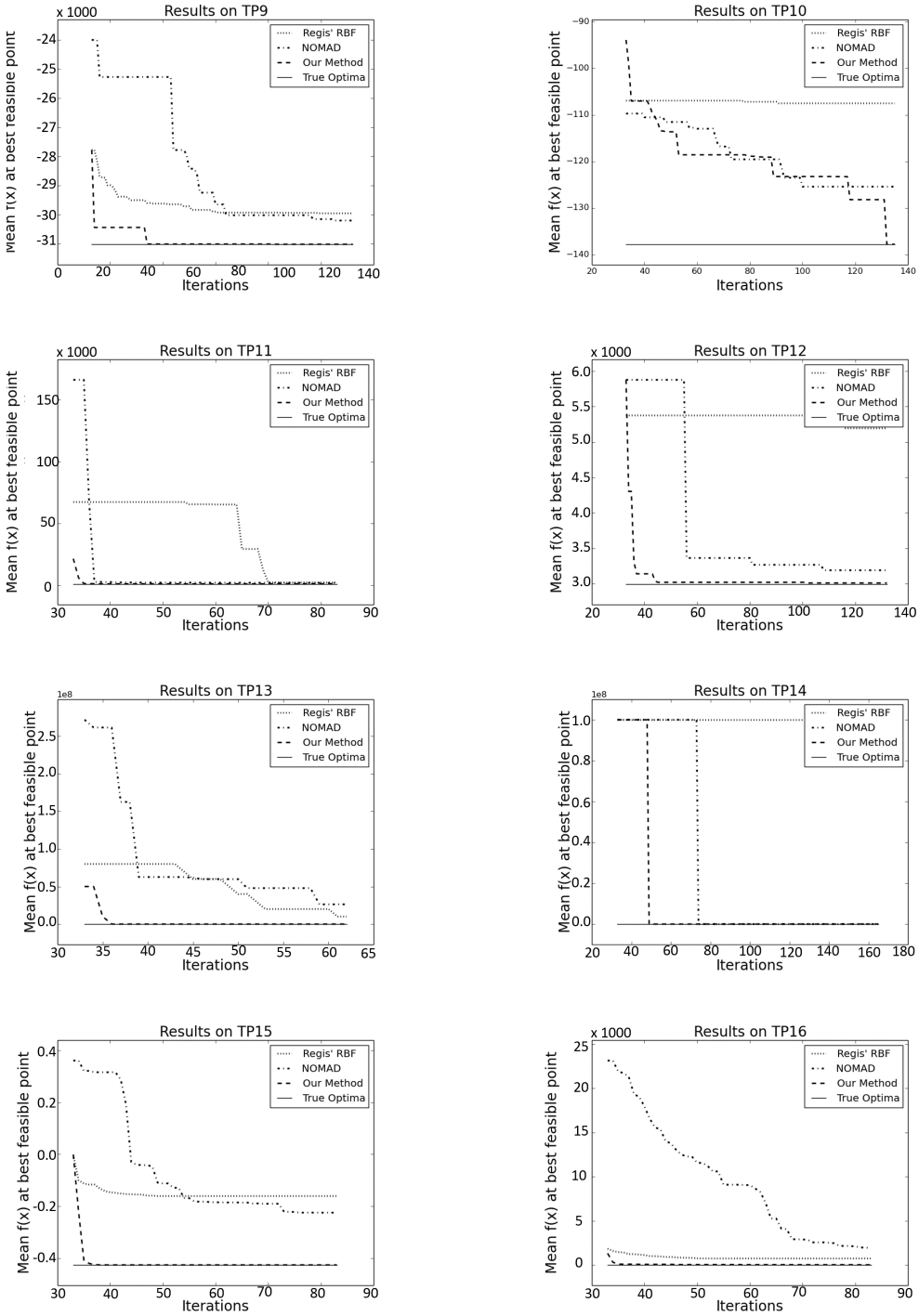


Figure 7.4: Comparison of our method with [1] and [2] on synthetic test problems (TP9-TP16). The plots show the mean objective function value at best feasible point at every iteration after the exploratory experiments. Our method, Regis' method, and NOMAD are denoted by green, red, and blue lines respectively.

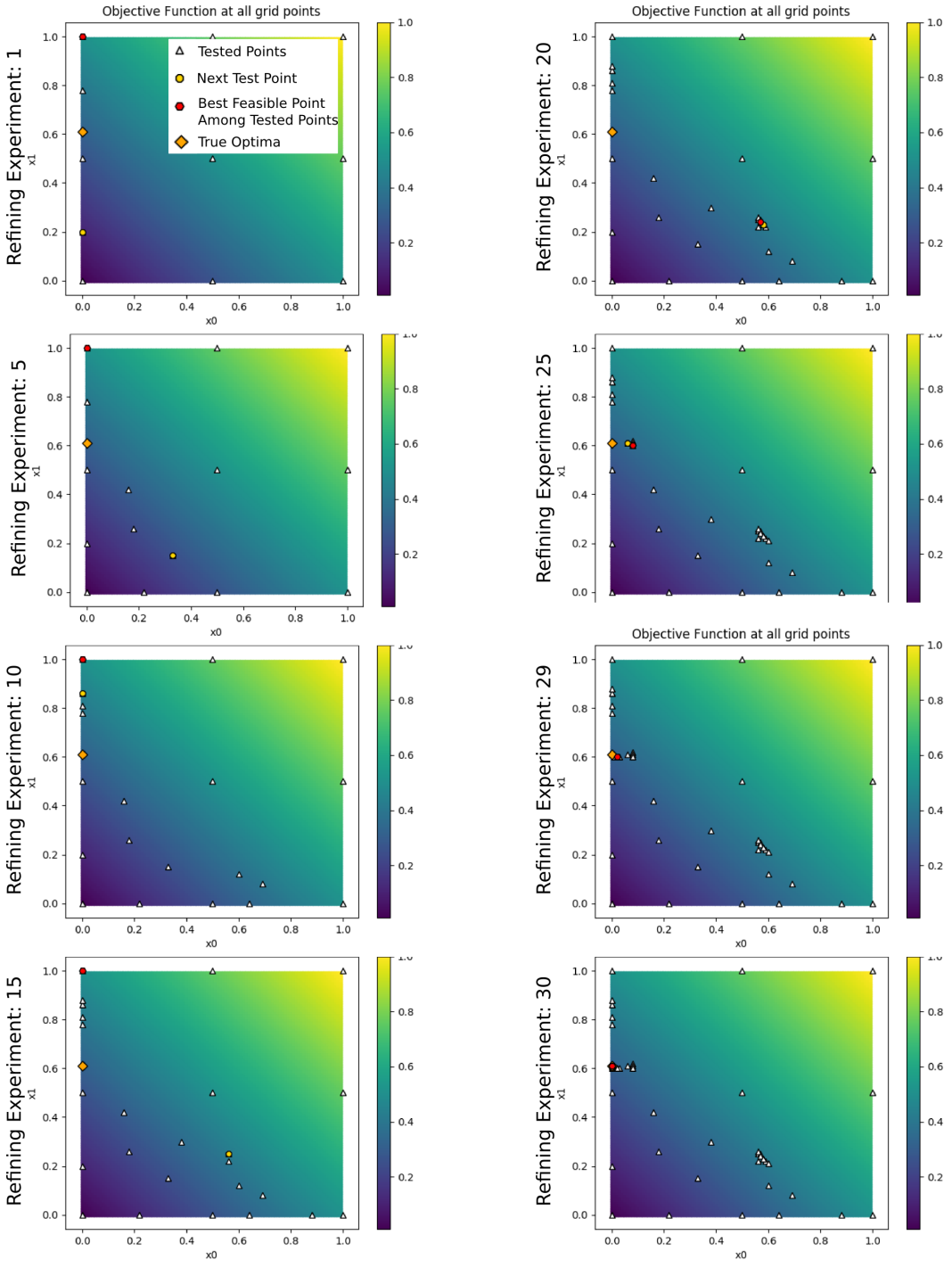


Figure 7.5: Results on dense uniform grid for Schwefel function. The color map illustrates value of objective function at every grid point. The Tested Points, Next Test Point, Best feasible Point among Tested Points, and True Optimum are plotted on top of the color map, at different iterations of refining experiments.

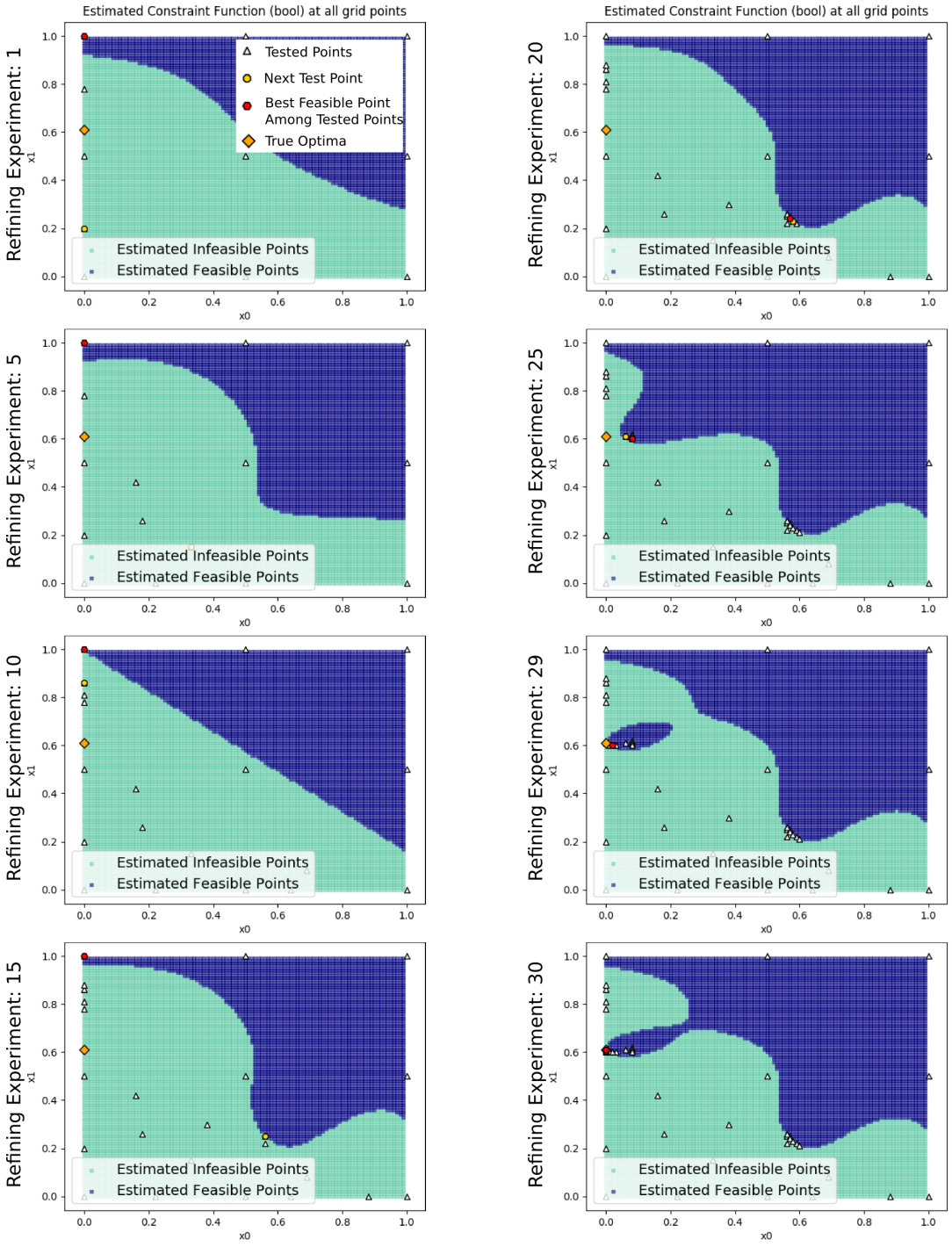


Figure 7.6: Results on the dense uniform grid for Schwefel function. The color map illustrates the value of estimated constraint function at every grid point, at different iterations of refining experiments. The Tested Points, Next Test Point, Best feasible Point among Tested Points, and True Optimum are plotted on top of the color map at different iterations of refining experiments.

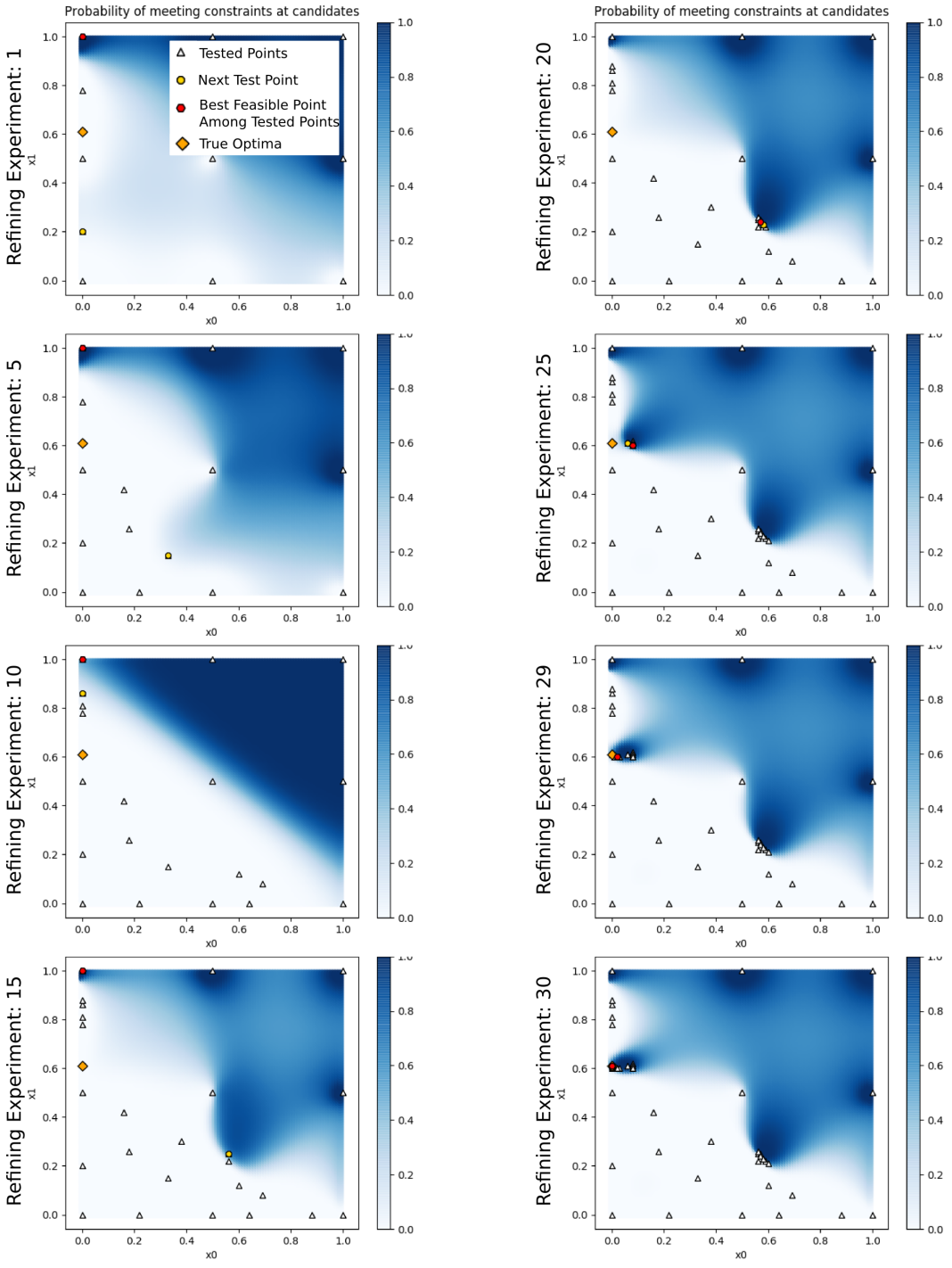


Figure 7.7: Results on the dense uniform grid for Schwefel function. The color map illustrates the value of the probability of meeting constraints at every grid point, at different iterations of refining experiments. The Tested Points, Next Test Point, Best feasible Point among Tested Points, and True optimum are plotted on top of the color map, at different iterations of refining experiments.

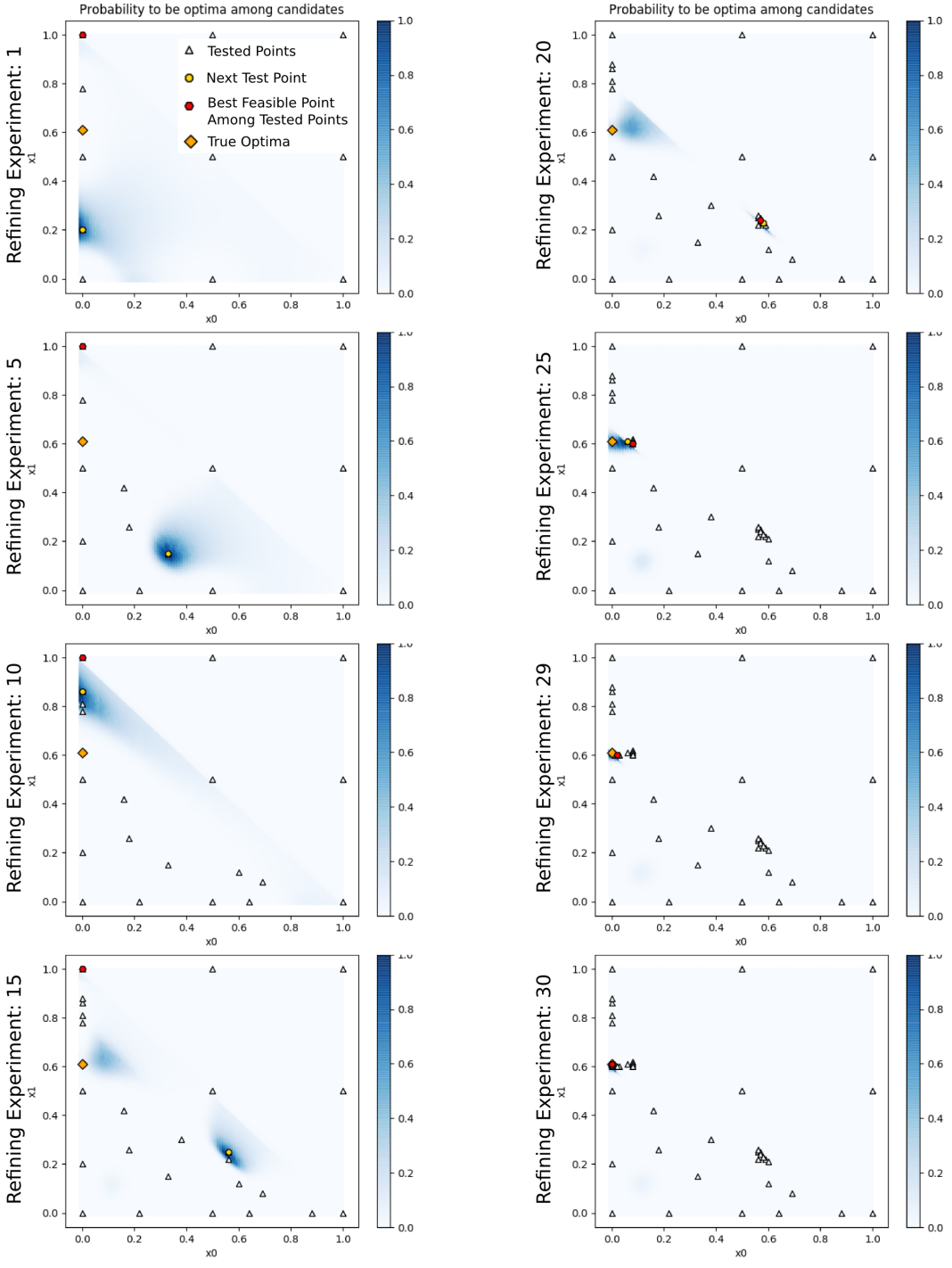


Figure 7.8: Results on the dense uniform grid for Schwefel function. The color map illustrates the value of estimated probability to be optimum at every grid point, at different iterations of refining experiments. The Tested Points, Next Test Point, Best feasible Point among Tested Points, and True Optimum are plotted on top of the color map, at different iterations of refining experiments.

7.4.3 Results on physical experiments

We conducted robotic cleaning and sanding experiments to validate our method. The details on the experiments are described in the subsections.

7.4.3.1 Robotic Scrubbing

In practice, the foreign particles are separated from any target surface by applying a cleaning fluid or abrasive actions. Both methods are required in some cases. Our focus was on cleaning hard stains that require scrubbing.

We explored effects of the following robot trajectory parameters on cleaning performance: (1) Tool speed v, (2) Force f_n applied by the tool in the normal direction to the surface, (3) Stiffnesses κ_x and κ_y in the x and y directions of the tool reference frame, (4) Frequency f_x and f_y of the forced oscillation in the x and y direction with respect to the tool reference frame, and (5) Amplitude of overlaid force A_x and A_y for force oscillation. Changes in these parameters change the scrubbing trajectory from the nominal trajectory. Figure 7.9(a-c) illustrates an example of a nominal trajectory and two actual trajectories generated by two different settings of trajectory parameters.

During robotic scrubbing, the applied force on the surface needs to be low to avoid damaging the part. The user may also want the cleaning process to complete quickly. In our experiments, we defined the objective functions as a weighted sum of normal force on the surface and tool speed. We defined the constraint to be the desired cleaning performance. It is very difficult to model the cleaning performance as a function of the trajectory parameters. Therefore we considered the cleaning performance to be a black-box function of the trajectory parameters.

Let $Y(\theta)$ represent the cleaning performance and Y_{th} be the performance constraint, where $\theta = [v, f_n, \kappa_x, f_x, A_x]$. The optimization problem in robotic cleaning can be formulated as following:

Find θ^* s.t. $Y(\theta^*) \geq Y_{th}$, $\theta^* = \underset{\theta}{\operatorname{argmin}}\ f(\theta)$, where $f(\theta) = -(w_1 \times v + w_2 \times (1 - f_n))$ and w_1 and w_2 are positive constant weights.

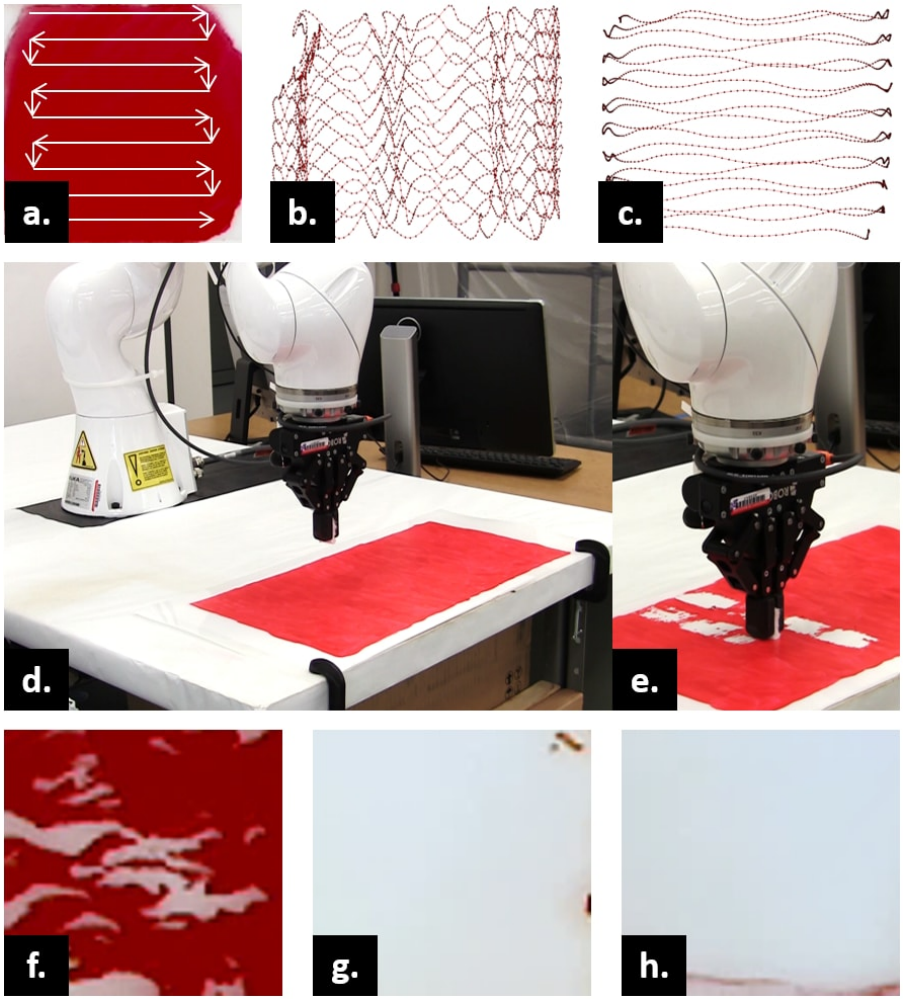


Figure 7.9: (a) Example of a nominal trajectory, (b,c) Actual trajectory originating from the nominal trajectory for different set of robotic operation parameters, (d) The experimental setup for robotic cleaning by scrubbing, (e) An on-going cleaning experiment, (f) An infeasible cleaning performance, (g) A feasible cleaning performance with high objective function value, (h) The feasible cleaning performance with the best found objective function value

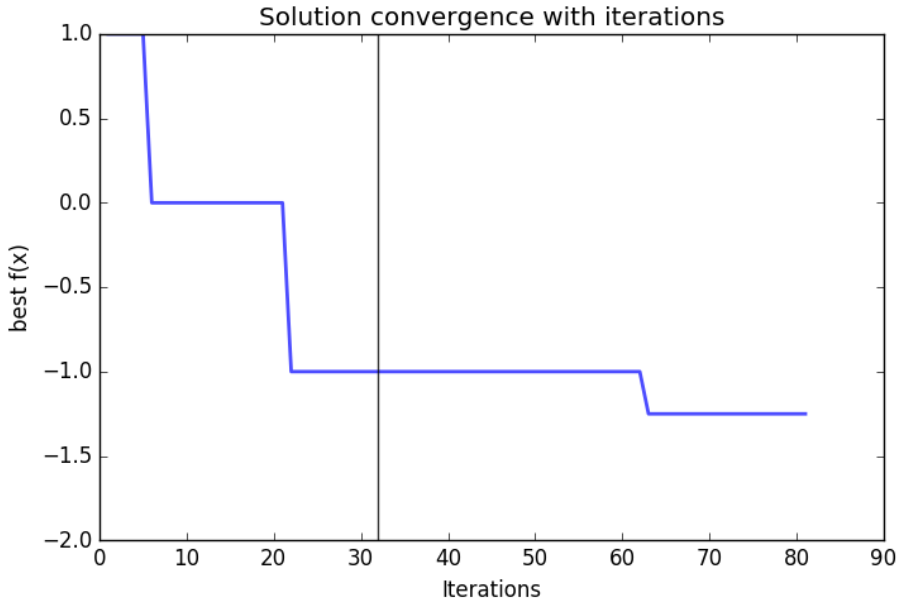


Figure 7.10: Convergence to solution in robotic cleaning. Iteration 1-33 are exploration experiments. Iteration 34 to 80 are refining experiments.

Experimental setup: A KUKA iiwa 7 manipulator was used in our robotic scrubbing experiments. We operated the arm in impedance control mode. Acrylic paint was used as a surrogate for stain, and a plastic board was used as the surface to clean. The plastic board was subdivided into smaller blocks of stain. The robot performed cleaning with different sets of trajectory parameters in different blocks.

The experimental setup and an ongoing experiment are illustrated in Fig. 7.9(d,e). In our earlier work on parameter optimization, we had used small plastic tiles for our robotic cleaning experiments. Changing the slides and painting them individually for each experiment was time-consuming. In our current setup, the robot performed cleaning on 16 cells before the board needed to be rotated. Cleaning the entire surface from a fixed relative pose between robot and board was not possible due to kinematic constraints of the robot. This approach saved both time and effort

as we had to paint only 3 boards and rotate or change the board for 5 times. In our previous setup, we had to change the tile at each iteration accumulating a large setup change time.

Perception: The k-means clustering [197, 198] algorithm was used in our experiments for stain detection. The metric for cleaning performance was the difference in the number of stain pixels before and after cleaning.

Result: The cleaning performance threshold was set to be $Y_{th} = 0.96$ in our robotic cleaning experiments (we wanted 96% of the stain pixels to be removed). A total of 80 experiments were conducted, including 33 initial exploratory experiments. We found the best feasible solution at the 63^{rd} experiment. Examples of some cleaning results from the refining experiments are shown in Fig. 7.9(f-h). We found the optimal parameters to be: $v = 0.3, f_n = 11.25N, \kappa_x = 3500N/m, f_x = 3Hz, A_x = 5N$. The graph of the best feasible objective function value is illustrated in Fig. 7.10.

7.4.3.2 Bi-manual robotic sanding

Sanding is a process application under the category of surface finishing. For our robotic sanding experiments, we have used a rotary tool (Dremel 3000). Our objective was to achieve the desired surface roughness(Ra) value.

We explored effects of the following robot trajectory parameters on sanding performance: (1) Tool speed v, (2) Force f_n applied by the tool in the normal direction to the surface, (3) Stiffnesses κ_z in the normal direction of the tool reference frame, and (4) Stiffnesses κ_x and κ_y in the x and y directions of the tool reference frame. In our experiments we considered $\kappa_x = \kappa_y$

Our focus was to minimize force and maximize velocity. Also, we wanted to reduce the Cartesian stiffness in the z-direction to aid tool manipulation on a curved surface. We defined the objective functions as a weighted sum of normal force on the surface, tool speed, and stiffness in the normal direction of the tool. We defined the constraint to be the desired surface roughness. Developing a physics-based model for surface roughness in terms of robot trajectory parameters is not feasible. Therefore surface roughness was the black-box function in these experiments.

Let $Y(\theta)$ represent the surface roughness and Y_{th} be the performance constraint, where $\theta = [v, f_n, \kappa_z, \kappa_x]$. The optimization problem in robotic sanding can be formulated as following:

Find θ^* s.t. $Y(\theta^*) \geq Y_{th}$, $\theta^* = \underset{\theta}{\operatorname{argmin}}\ f(\theta)$, where $f(\theta) = -w_1 \times v + w_2 \times f_n + w_3 \times \kappa_z$; w_1, w_2, and w_3 are positive constant weights.

Experimental setup: We have used one KUKA iiwa 7 manipulator and one Epson C3 manipulator in the robotic sanding experiments. In the bi-manual operation, the iiwa arm manipulated the tool while the C3 arm held the part. The holding arm changed the pose of the part so that the robot with the tool can reach all the regions on the surface. The advantage of bi-manual operation of such kind is that no human intervention is needed to change the setup. Also, this eliminates the need of a fixed fixture.

We operated the iiwa arm in impedance control mode. Additively manufactured ABS plastic parts were used in the sanding experiments. The surface roughness of the raw part was $90\mu m$. The tool manipulating robot performed sanding with different sets of trajectory parameters in different regions on the part surface. We considered multiple passes (six passes) of sanding over the surface. Sandpapers of grit size 800 and 1200 were used in the first three passes and last three passes respectively. This selection was based on the opinion of an expert human user. The experimental setup and an ongoing experiment are illustrated in Fig. 7.11.

Perception: We have used a surface roughness measurement device (Mitutoyo SJ-410) to evaluate the sanding performance at each iteration. Fig. 7.12 illustrates the surface roughness measurement process.

Result: The surface roughness threshold was set to be $Y_{th} = 10\mu m$ in our robotic sanding experiments. A total of 124 experiments were conducted, including 17 initial exploratory experiments. We found the best feasible solution at the 44^{th} experiment. Examples of some sanding results from the refining experiments are shown in Fig. 7.12. We found the optimal parameters to be: $v = 0.01, f_n = 0.5N, \kappa_x = \kappa_y = 800N/m, \kappa_z = 400N/m$. The graph of the best feasible objective function value is illustrated in Fig. 7.13.

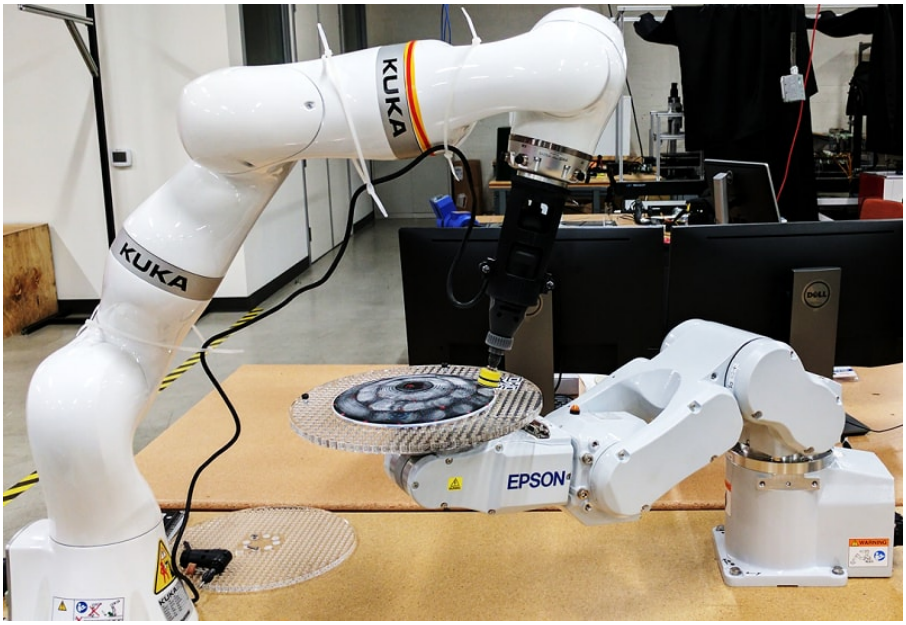


Figure 7.11: An ongoing robotic sanding experiment

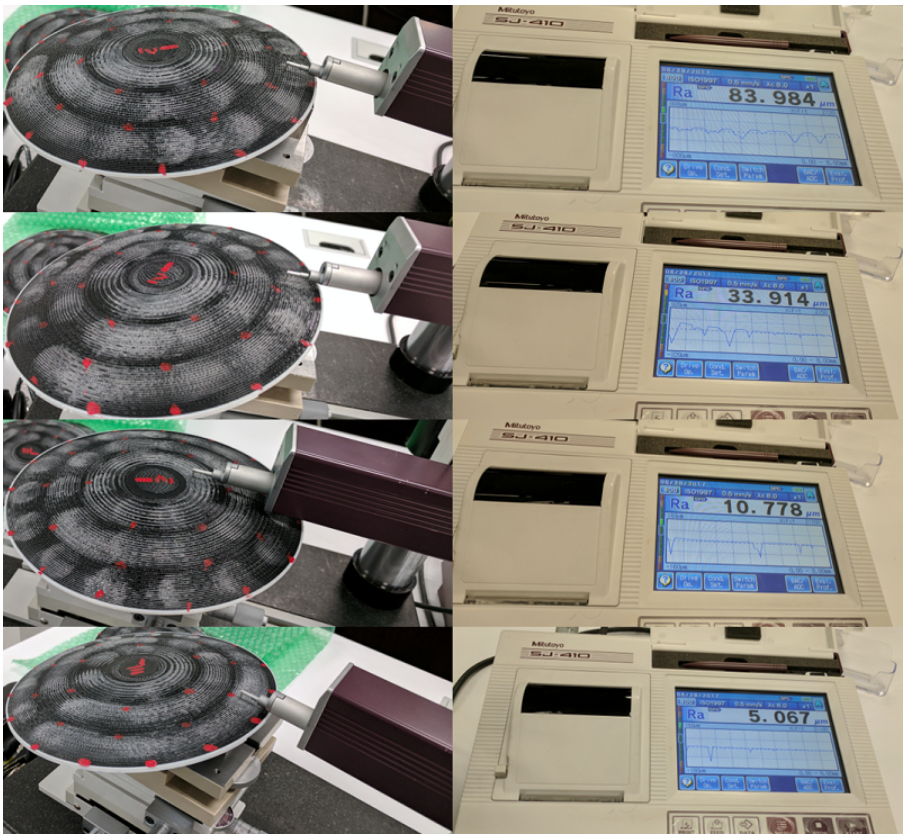


Figure 7.12: Some sanding performance measurement from refining experiments

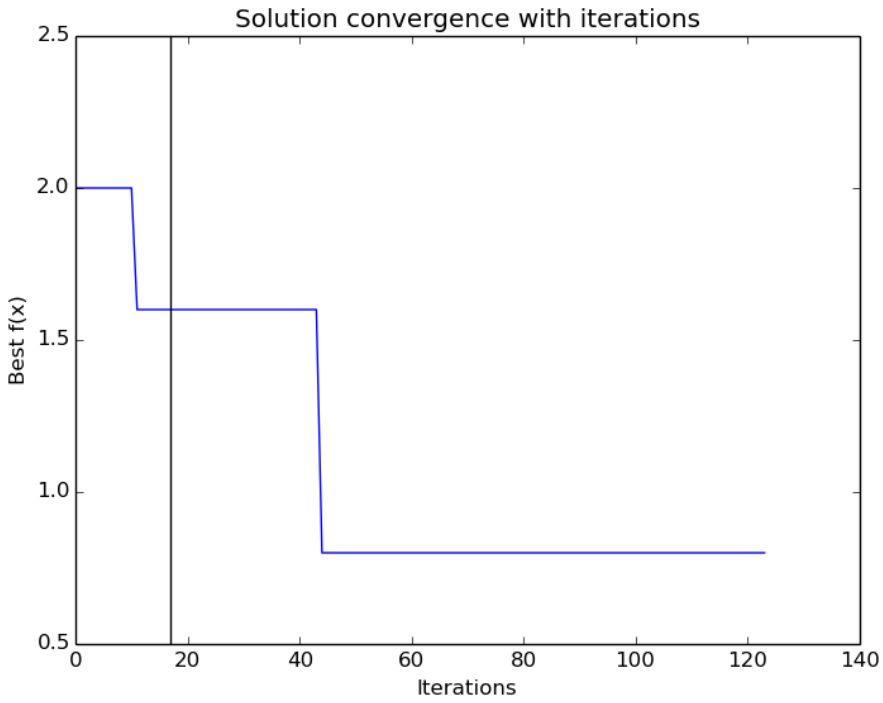


Figure 7.13: Convergence to solution in robotic sanding. Iteration 1-17 are exploration experiments. Iteration 18 to 124 are refining experiments.

Table 7.2: Summary of synthetic test problems

Test Problem	**Dimension of Parameter Space**	**Inequality Constraints**	**Region defined by Parameter Space Boundary**
TP1	2	2	$[0, 500] \times [0, 500]$
TP2	2	2	$[13, 100] \times [0, 100]$
TP3	2	1	$[-1, 1]^2$
TP4	3	1	$[0, 10]^3$
TP5	4	3	$[0, 1]^2 \times [0, 50] \times [0, 240]$
TP6	4	7	$[0.125, 10] \times [0.1, 10]^3$
TP7	4	5	$[0, 1200]^2 \times [-0.55, 0.55]^2$
TP8	5	3	$[-2.3, 2.3]^2 \times [-3.2, 3.2]^3$
TP9	5	6	$[78, 102] \times [33, 45] \times [27, 45]^3]$
TP10	6	6	$[0, 5] \times [0, 4] \times [1, 5] \times [0, 6] \times [1, 5] \times [0, 10]$
TP11	7	4	$[-10, 10]^7$
TP12	7	11	$[2.6, 3.6] \times [0.7, 0.8] \times [17, 28] \times [7.3, 8.3]^2 \times [2.9, 3.9] \times [5, 5.5]$
TP13	8	6	$[10^2, 10^4] \times [10^3, 10^4]^2 \times [10, 10^3]^5$
TP14	8	6	$[10^2, 10^4] \times [10^3, 10^4]^2 \times [10, 10^3]^5$
TP15	10	2	$[0, 10]^{10}$
TP16	10	8	$[-10, 10]^{10}$

7.4.3.3 Experimentation Time

Robotic finishing of the entire surface of a part with fairly complex geometry or large surface area usually requires hours to complete. The total time to determine the optimal trajectory parameters needs to be small. We have designed our method such that each experiment will be done on a very small surface patch. Finishing a small surface patch can be done within a few seconds. In our cleaning and sanding experiments, each trial took less than 5 seconds to complete. Therefore the entire process of optimization was less than 10 minutes in both cases.

7.5 Summary

The work describes an approach to identify optimal trajectory parameters for robots. The method optimizes an objective task function and satisfies the task performance constraints. The iterative algorithm makes decisions based on the uncertainty in the parameter space and reduced task completion time by minimizing the number of experiments.

Table 7.3: List of Symbols

Symbol	**Meaning**
d	Dimension of parameter space
S	Parameter Space, $S \subset \mathbb{R}^d$
x	A point in the parameter space (d-dimensional vector)
a,b	Lower and upper bounds for the parameter space. $a, b \in \mathbb{R}^d$, $a^i \leq x^i \leq b^i, i = 1, \ldots, d$
$f(x)$	Known objective function
$g(x)$	Black-box constraint
$\mathbb{X}_0$	Set of initial exploration points
$\mathbb{X}$	Set of tested/evaluated points
$\mathbb{C}_x$	Set of candidate points
n_c	Number of candidate points at (a fixed resolution) in the neighborhood of a tested point
$r_{initial}$	Coarse sampling resolution for initial experiments
r_x	Sampling resolution in the neighborhood of $x \in \mathbb{X}$
x_{best}	$x_{best} \in \mathbb{X} : g_i(x_{best}) \geq 0, i = 1, \ldots, n$ and $x_{best} = argmin f(x), x \in \mathbb{X}$
P_i^c	Probability of $x_i \in \mathbb{C}_x$ to satisfy all the constraints
P_i	Probability of $x_i \in \mathbb{C}_x$ to be the optimum in $\mathbb{C}_x$
$G_{\mathbb{C}_x}$	Array of probabilities for each candidate point to meet each constraint
$F_{\mathbb{C}_x}$	Array of objective function value for each candidate
P_c	Array of probabilities for each candidate point to meet all constraints
Y	Finishing Performance
Y^{th}	Desired Finishing Performance or Threshold on finishing performance
θ	Trajectory Parameters (1 $\times$ d vector)
v	Tool speed
f_n	Normal force applied by the tool on the surface
κ_x	Stiffness of manipulator along $x - axis$ of tool
κ_y	Stiffness of manipulator along $y - axis$ of tool
κ_z	Stiffness of manipulator along $z - axis$ of tool
A_x	Amplitude of overlaid force-oscillation along $x - axis$ of tool
A_y	Amplitude of overlaid force-oscillation along $y - axis$ of tool
w	Weight vector for designing objective function in the physical experiments

Chapter 8

Case Study: Robotic Finishing of Geometrically Complex Parts

8.1 Introduction

This chapter[1] presents a case study on automated robotic finishing. It shows the benefits of the developed trajectory planning, setup planning, task-agent assignment, and parameter identification technologies in robotic finishing applications.

Finishing processes are one of the four basic classes of manufacturing. It is required in many different manufacturing applications. Surface finishing includes tasks such as cleaning, sanding, polishing, superfinishing, deburring, and paint-stripping. These tasks account up to 25% of the manufacturing cost. Automating these tasks can improve efficiency, and significantly reduce manufacturing costs.

Finishing processes are often highly non-repetitive in small and medium volume productions. Many parts with intricate interior regions require finishing operations. The finishing process automation needs to adapt to the changes in individual part geometry, scale, surface, and tools. A unique tool trajectory is required to be generated for every new part. Experiments need to be

[1]The work in this chapter is derived from the work published in [3]

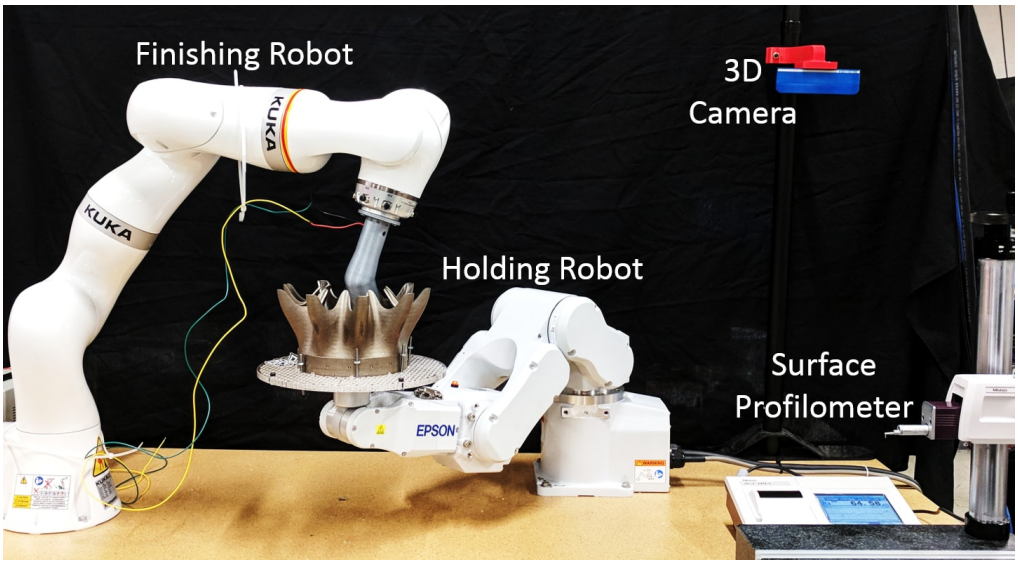


Figure 8.1: An on-going surfcace finishing in our robotic finishing cell

carried out to set the right operation parameters (e.g., force, velocity, stiffness, oscillation) for the desired finish performance.

Traditionally, manipulators have been used in manufacturing for repetitive material handling and assembly tasks. Employing robots on non-repetitive finishing tasks is difficult, as the manipulator cannot use preprogrammed motions. An advanced impedance controller needs to be used instead of a position controller to ensure that the robot can deal with part position uncertainty caused by registration errors. In addition, processing the entire complex target surface from one relative pose between the robot and the part may not be possible. Often, the part must be moved and re-grasped to ensure access to the entire surface. Moreover, in small production volume runs, it is economically not feasible to build custom fixtures to hold the parts during finishing processes. In order to finish a part without fixtures, we need to dynamically generate motion plans for a robot or mechanism to manipulate the part in space, while a second robot performs the finishing task. Parts may also be delicate and cannot withstand arbitrarily large forces, requiring that the applied force be monitored and controlled to ensure that the part being processed is not damaged. Finally, finishing tasks require continual autonomous monitoring and

assessment of the surface. As a result of these complexities, tedious and time-consuming finishing tasks of geometrically complex parts with interior features are traditionally performed by hand. Automating the finishing process on such parts is expected to provide the following benefits:

1. Performing finishing operations on complex parts is often an ergonomically challenging task. Automating the finishing operation can reduce the potential for risk to human health.

2. Automated finishing process can provide consistent quality and reduce the risk of part damage caused by human error.

3. Finishing is often a time-consuming operation. Automated finishing can increase productivity by reducing the touch labor.

This chapter presents a system for non-repetitive finishing of user-specified parts. The automated system is capable of finishing complex parts. The system is applicable to non-repetitive finishing tasks in small and medium volume productions, where manually programming robots is not feasible. The system automates the mundane tasks of finishing to improve productivity. The system architecture of the presented method is sufficiently general to permit scaling to meet the requirements of different industries where finishing is an important part of manufacturing processes.

8.2 Requirements for Robotic Finishing System

An automated robotic finishing system will need to have certain hardware and software components to interact with the part and the human operator.

8.2.1 Hardware

We are listing down the hardware components below with brief descriptions.

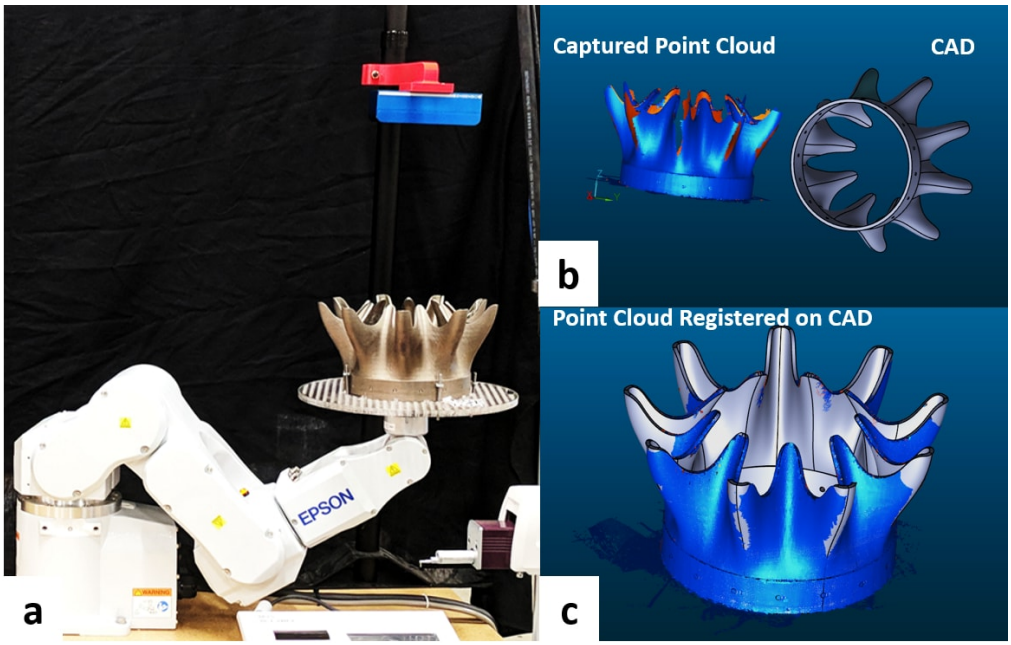


Figure 8.2: (a) Registering the part on the holding robot using vision system, (b) Captured Point Cloud and CAD before registration, (c) Captured Point Cloud overlaid on CAD after registration

1. *Finishing Robot:* A robotic finishing system will need to have a robotic manipulator to operate the finishing tool on the part.

2. *Holding Mechanism (or Robot):* For a fixed pose of the part, the finishing robot will not be able to reach all the regions on it due to kinematic and dynamic constraints. Therefore, the system will need a mechanism (or a second robot) to hold the part and change its pose (setup) relative to the finishing robot.

3. *Part Fixture:* Most of the surface finishing operations are non-repetitive in nature. It is economically not feasible to build custom fixtures for each part in low and medium volume production. An automated system for robotic finishing needs to have a reconfigurable fixture to mount a large variety of parts on the holding mechanism.

4. *Finishing Tools and Tool Changer:* Different geometry and material requires different kinds of finishing tools. The system needs to have a set of finishing tools available in the work

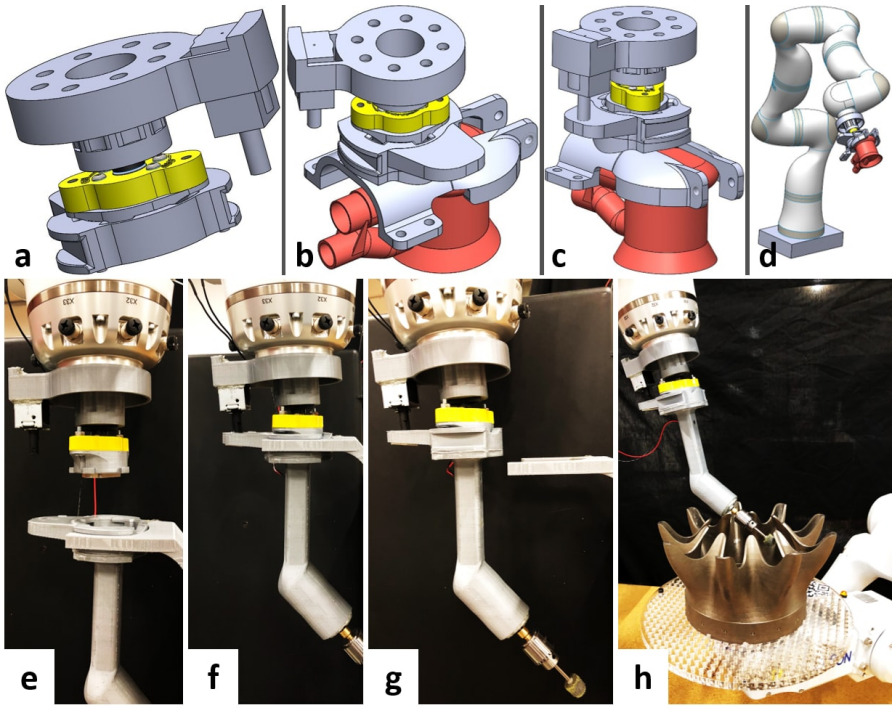


Figure 8.3: (a-d) The tool changing mechanism in CAD environment. (e-h) The tool being mounted on the finishing robot using the designed tool changer.

cell to work on different parts. A tool changing mechanism is required to change the tools between processes automatically.

5. *Sensors for Localization:* An appropriate sensor system needs to be integrated with the system to localize the part with respect to the robots.

6. *Sensors for Performance Evaluation:* An appropriate sensor system needs to be integrated with the system for performance evaluation and feedback.

8.2.2 Software

We are listing down the required software components below with brief descriptions.

1. *Tool Selection Module:* Depending on part geometry, it may not be possible to use the same finishing tool on the entire surface of a part. This is due to the constraints posed by part

and tool geometry. The robotic finishing system needs to have a tool selection module that will determine the right tools to use for different regions of the part.

2. *Task-Agent Assignment Module:* The system needs to have a task-agent assignment module to identify which robot/agent to use for operating finishing tool and which robot/agent to use for holding the part.

3. *Trajectory Planner for Finishing Robot:* The system needs to have a trajectory planner that is capable of generating collision-free insertion, finishing, and retraction trajectories in real-time for the finishing robot. The trajectories need to meet the process constraints (e.g., constraint on finishing time).

4. *Setup Planner for Holding Robot:* We call the pose of the part relative to the finishing robot to be a setup. The part needs to be placed in a certain setup based on the reachable workspace of the finishing robot. The system needs to have the capability of determining the setup to maximize reachability and minimize finishing time.

5. *Process Parameter Selection Module:* The process parameters influence the performance of robotic finishing. These parameters include velocity, tool speed, normal and shear force, stiffness and damping parameters, etc. The system needs to be capable of selecting the optimal process parameters.

6. *Process Monitoring and Contingency Handling Module:* An automated performance measurement module need to be integrated with the system for evaluating finishing performance and monitoring progress. The system also needs to be able to detect contingencies that may arise. Based on performance feedback and contingencies, the system needs to be capable of auto-adjustment. It should seek human help when the auto-adjustments are not enough to resolve contingencies.

7. *User Interface:* The robotic finishing system needs an intuitive user interface through which the human operator can interact with the system and provide high-level instructions. He/she should be able to perform the following operations through the interface.

   (a) Import Computer-Aided Design (CAD) model of the part

   (b) Select desired regions on the surface to finish

   (c) Select process parameters

   (d) Initiate trajectory generation for the finishing robot

   (e) Verify the system generated a trajectory for the finishing robot

   (f) Provide seeds for the part setup

   (g) Verify the optimized setup by the system

   (h) Execute robot motion

   (i) Remotely monitor the process

8.3 System Design

We have developed an automated finishing system to complete non-repetitive finishing tasks for parts with complex geometries, especially concave interiors. Our system takes high-level commands from the human operator and transforms them into low-level instructions for the robot. The inputs to our system are a CAD model, task specifications, desired finishing performance constraints, and process parameter bounds. The output of the system is a finished part with desired performance.

1. *Design of Workcell:*

   (a) *Robots:* We designed the robotic finishing work cell with two robots (see Fig. 8.1). One of the two robots is a 7 degree-of-freedom (DOF) KUKA iiwa7 that is used as a

finishing robot which manipulates the finishing tool required for the finishing task. The second robot in the work cell is a 6 DOF Epson C3 which is used for manipulation of the part that needs to be finished. Ideally, any mechanism that can change the parts' position and orientation in space can be used as the holding mechanism.

(b) *Finishing Tools:* The finishing robot is equipped with custom tools to finish a wide range of geometries. The housings of the custom tools are additively manufactured. DC motors with appropriate ratings are attached to the housing to create tools with desired specifications. The speed controllers of the motors are operated directly from a host computer. We selected the tool heads based on application requirements.

(c) *Finishing Tool Changer:* We have designed and integrated a custom tool changer in our system. It helps to install and remove different tools for having different applications quickly. A permanent magnet and a linear actuator are used to hold the tool in position securely. All the parts used in the assembly are additively manufactured with Polylactic Acid (PLA) material. To restrict the linear motion of the tool or the direction from where tool can fall from the attached part, a mechanism using a permanent magnet and a linear-pull solenoid actuator is used. Fig. 8.3(a-d) illustrates the tool latching mechanism in a CAD environment, the Fig. 8.3(e-f) shows the robot grasping the tool in the physical environment.

(d) *Reconfigurable Fixture:* We attached a platform (made out of acrylic sheet) for part placement on the holding robot. This platform was designed to be a flexible universal fixture. Having flexible fixtures can facilitate small volume manufacturing, where tasks are highly non-repetitive in nature. Parts of a wide variety of geometry can be mounted using our simple clamping mechanisms.

(e) *Perception:* We have integrated visual and tactile perception modules in our system to automate part registration and performance measurements.

i. *Visual:* We have used the Ensenso N10 camera to acquire the point cloud of the part during part registration. We have used the Iterative Closest Point (ICP) algorithm in our system to register the point cloud of the part to the CAD model. The human operator provided input to improve the accuracy of the part registration and reduce the computation time required to compute the parts' pose. After the part registration, the system knows the pose of the part with respect to the finishing robot as the holding mechanism changes the pose of its end-effector. The part needs to be registered every time it is placed at a different location on the platform. Fig. 8.2 illustrates an example of part registration.

ii. *Tactile:* We have integrated a Mitutoyo SJ-410 surface profilometer to measure surface roughness accurately when needed. The holding robot in our system takes the target surface under the probe of this device to take measurements. Fig. 8.8 illustrates some images taken from footage of automated performance measurement. Due to complex part geometries, it is not always possible to automate the process of taking measurements. In our experiments, the human operator manually measured the performance of the parts with complex internal regions using the surface profilometer. Moreover, the operator used an external CMM to take approximate measurements for regions where a surface profilometer could not reach.

2. *Software Architecture:* We have developed our own software architecture to connect all the hardware components and user-interface together. The communication to the robots and measurement devices were done using TCP/IP and serial communication. A central computer acts as the master (or host) device, and all the hardware components act as slave devices. The master (or host) device takes user input from the user interface, runs all the algorithms (e.g., trajectory planning, parameter optimization, etc.) in the background, and transfers the instructions to the robots, measurement devices, sensors, and motor drives.

We used the ROS framework as part of our architecture. It is mainly used for simulation and visualization environments.

3. *User Interface:* We developed a Qt-based application for the user-interface. The human operator can import the CAD model through this interface and select regions on the surface that needs to be finished. The selection can be made either by drawing lines on the surface or by selecting surface patches. For line selections, the user only selects the start and endpoints. We used a geodesic curve generation algorithm to generate the complete scribe line as the shortest path on the surface. Fig. 8.5 illustrates an example of drawing scribe lines for trajectory generation. We have modified the open-source application Meshlab to create the interface for surface patch selection. Fig. 8.6 illustrates an example of selecting surface patches for trajectory generation.

4. *User Interaction with the System:* Once the CAD of the part is imported, and the desired surface area has been selected, the user goes through the following steps to interact with the system.

   (a) *Task-Assignment:* At the beginning, the system can suggest the operator which robots to use for the given tasks. This is done using the task-agent assignment algorithm. The operator can choose to go with the system provided suggestion or make his/her own choice. The system will then validate if the operator selected robots or agents are capable of performing the task. It allows the operator to proceed when a feasible task-agent assignment is complete.

   (b) *Part placement with the aid of Mixed Reality:* We have integrated a Microsoft Hololens in our system that can guide an inexperienced user in mixed reality. It projects a hologram of the part on the platform during part placement and is capable of guiding the user through the clamping sequence. This way, the user will not mistakenly place the wrong part on the platform. Our current mixed reality system uses markers for

detecting different objects. It is also restricted by the viewing angle of the Hololens camera. Fig. 8.9(a) illustrates an example of the mixed reality interface.

(c) *Tool Selection:* After placing the part on the holding robot, the operator initiates the tool selection module through the user interface. We use a search-based algorithm to determine which tools will be optimal to use on different regions of the part's surface. The search-based algorithm uses the tool geometry as a part of the cost function to minimize the finishing time and to detect collision with part features. The operator verifies the system generated a selection of tools and makes modifications if needed. In our current system, we are only able to accommodate the tools with cylindrical geometry. The operator manually selects the regions for the tools with complex geometry in the scenarios where the automated algorithm fails to provide a solution.

(d) *Setup Planning:* In section 8.2, we defined the setup to be a pose of the part with respect to the finishing robot. The pose of the part is six-dimensional with three translation and three rotation components. It is not feasible to analytically determine the optimal setup for a part to be finished in six-dimensional space by considering the kinematic and dynamic constraints of a 7-DOF manipulator. Therefore, we designed our system to take some sample seeds for the setups from the human operator. The operator can input multiple setups (or pose of the part) by articulating the holding robot through the user interface. Our setup planner [181] run search and optimization algorithms on the user-provided seeds to find the optimal setup or sequence of setups.

(e) *Trajectory Planning:* After setup planning, the user can initiate the trajectory generation module from the interface. The trajectory generation module generates insertion, process, and retraction trajectories for the finishing robot. It also generates repositioning and setup changing trajectories for the holding robot. The trajectories are generated based on the inputs provided by the user. Fig. 8.7 and 8.8 illustrates some

images taken from footage of an insertion trajectory execution by the finishing robot and repositioning trajectory execution by the holding robot, respectively.

(f) *Trajectory Verification and Execution:* Once the trajectory is generated, the operator can simulate and visualize the motion of the robots in the user interface. The operator can modify waypoints on the trajectory if needed. We integrated the RViz visualization environment with our user interface. Once verified, the operator can then deploy the trajectories on the physical system and initiate execution. Our interface provides an option to monitor the physical system remotely. It imitates the live motion of the robot and tool in a virtual environment so that the operator can monitor the process without being present close to the finishing cell.

(g) *Process Parameter Selection:* We have integrated a module for process parameter selection within the RViz environment. Fig. 8.4 illustrates the RViz environment. The user can manually tune the process parameters through this interface and observe their effects.

(h) *Performance Measurements and Feedback-based Adjustments:* The user can specify a schedule for taking performance measurements. To measure the performance of the finishing process, we have developed a module that can support and take feedback from laser scanners, probe-based CMMs, and surface profilometers. The user can decide the type of measurement device, the required feedback, and the interval to take performance measurement. Based on the feedback, the operator can adjust process parameters and trajectories through the interface. We used surface profilometer for performance measurement in our experiments. Human assistance might be required to take performance measurements manually for very complex internal geometries. A mixed reality system can be used to help the operator by highlighting the specific regions on the part to take the measurements at the right locations.

(i) *Contingency Handling and Smart Notifications:* Depending on the task profile, the finishing operation can take a long time to complete. Contingencies may arise during the process, and human inputs are sometimes crucial to resolve them quickly. However, it is not effective for the human operator to stand next to the robotic finishing cell the entire time. We have developed a smart notification module in our system. It can send text messages to the operator's phone or wearable devices (e.g., smartwatch) when the robot needs help from the operator or when any other human input is required by the system. Fig. 8.9(b) illustrates an example of a notification sent to the wearable device of the operator.

8.4 Results

We have conducted experiments of robotic finishing on a set of representative parts with complex geometries. These parts require finishing of internal regions for functionality. The names and brief description of the test parts are given below.

1. Housing for supporting the load-bearing shaft
2. Exhaust flow mixer for mixing exhaust gases with cold air in turbofan engines
3. Miniature turbine structure with epicyclic gear loading for greater torque output
4. Nozzle for fuel injection in gas turbines
5. Impeller for centrifugal pumps
6. Gear bearing for improved performance over traditional ball bearings
7. Bracket for jet engines

Fig. 8.10 illustrates the representative parts we used in our experiments. The surface roughness measurements(before and after finishing) for these parts are summarized in table 8.1. Fig. 8.11

Table 8.1: Robotic Finishing Performance on Different Parts

Part	**Ra Value** (μm)	
	Before	**After**
Shaft Housing	1.29	0.52
Lobbed Flow Mixer	12.75	1.62
Miniature Turbine with Epicyclic Gear Train	2.87	1.64
Fuel Injection Nozzle	7.95	1.64
Pump Impeller	4.70	0.44
Gear Bearing	5.88	3.28
Jet Engine Bracket	13.81	3.08

and 8.12 visually illustrate the surface roughness difference on the impeller blade and flow mixer respectively. A short video of robotic finishing process for the housing and flow mixer can be viewed at [199].

The internal surface area of the load-bearing shaft housing and exhaust flow mixer was 0.0903 m^2 and 0.0784 m^2 respectively. Our robotic finishing system took 24 $mins.$ to finish 0.0326 m^2 of the internal surface of the housing part and 58 $mins.$ to finish 0.0094 m^2 of the internal surface of the flow mixer. However, the surface area of the tool used for flow mixer was significantly smaller compared to the tool used for the housing. The finishing time is governed by the geometry of the part and tool. It is also constrained by the optimum velocity required to achieve the desired surface finish. We determined the velocity requirement using our process parameter selection module.

8.5 Summary

This chapter has shown that the collaborative system involving the human operators and the robotic assistants for complex finishing operations can be realized. The developed planning algorithm takes the CAD model of the part and high-level user input and generates the motion plans for the finishing robot. This significantly reduces the programming time and improves the efficiency of the system. Secondly, the developed parameter optimization routine iteratively selects the process parameters until the system reaches the satisfactory finishing performance. The

system has been evaluated on a representative set of parts with complex geometries. The system was able to perform finishing operations on the test parts satisfactorily.

Manual finishing operations are tedious and time-consuming. The system helps the human operators by reducing their workload and eliminating tasks that are ergonomically challenging. It enables them to utilize their time more effectively and increase their productivity.

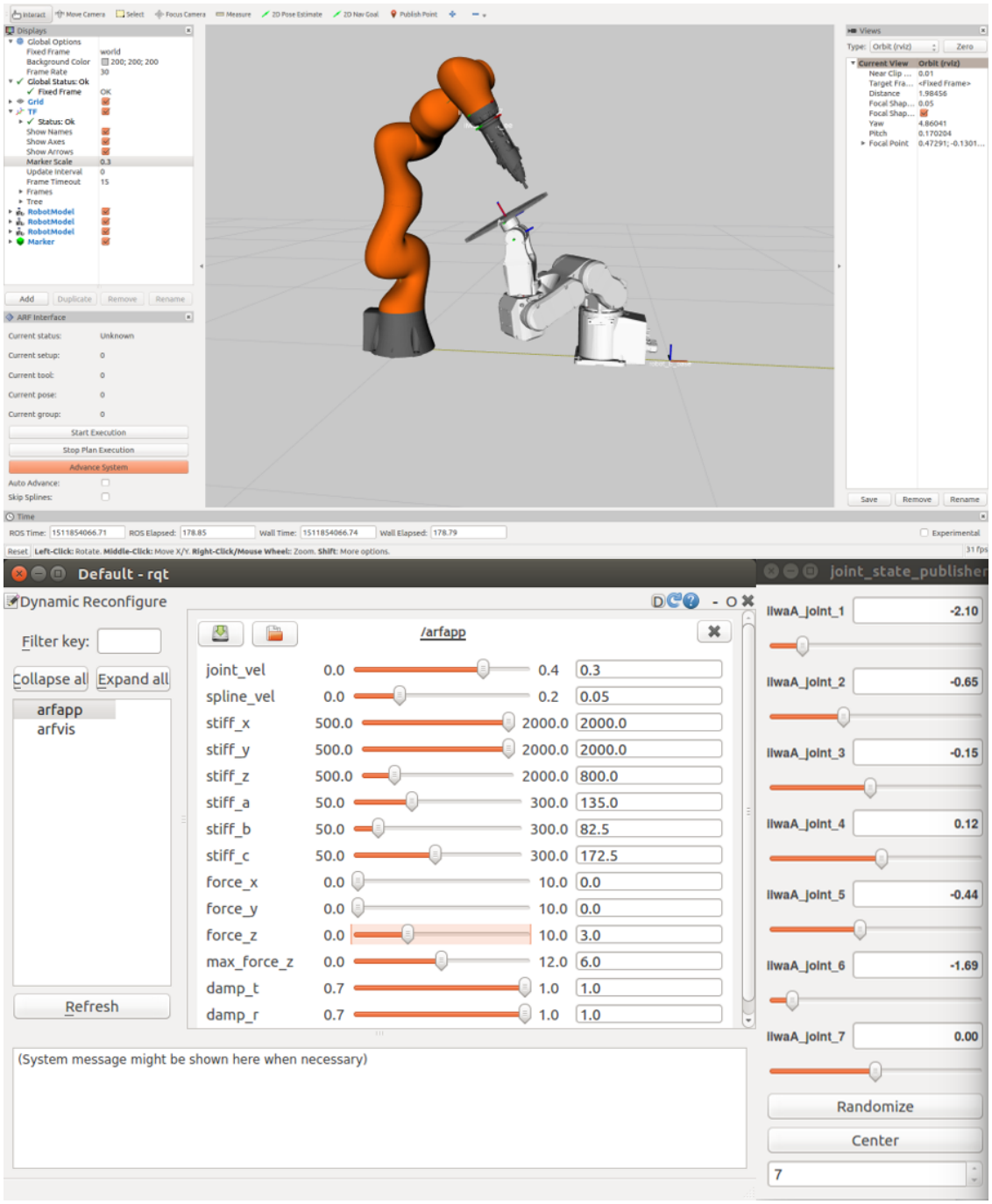


Figure 8.4: RViz-based virtual environment and process parameter selection module in the user interface.

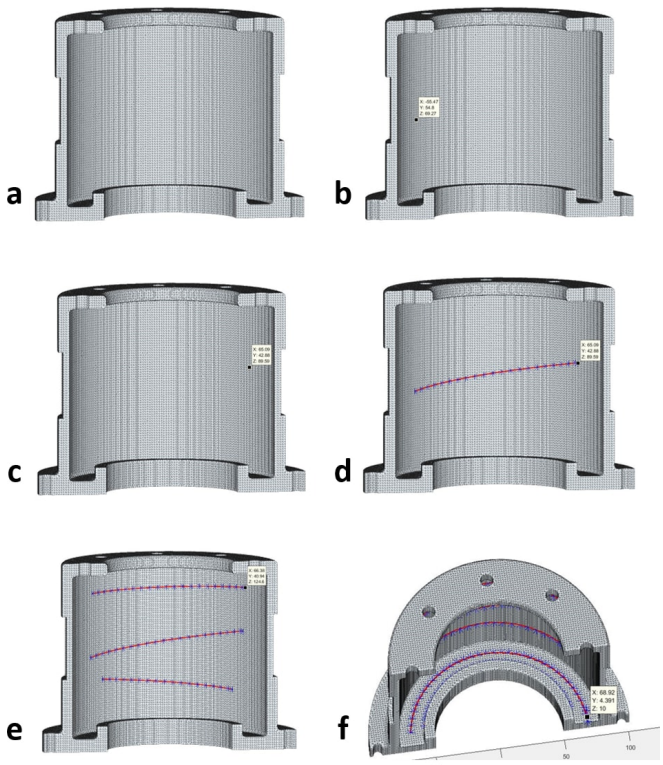


Figure 8.5: Example of trajectory generation by drawing scribe lines. (a) Imported CAD model with cross sectional view in the user interface, (b) Operator selected start point of a scribe line, (c) Operator selected end point of the same scribe line, (d) System generated scribe line as a geodesic curve, (e,f) More scribe lines created by operator.

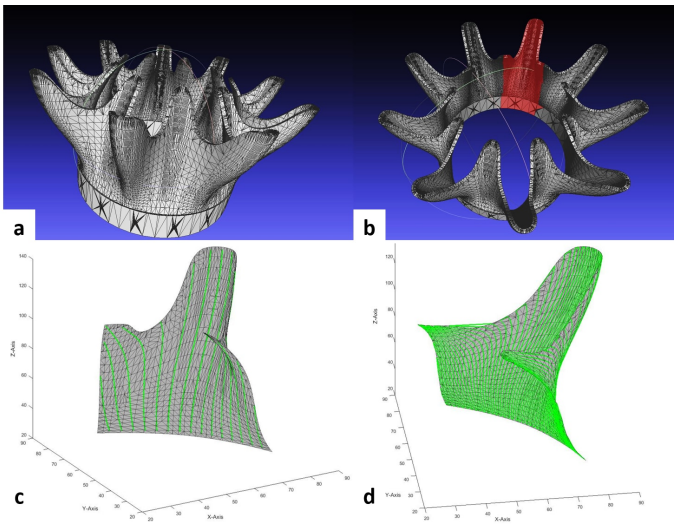


Figure 8.6: Example of trajectory generation by selecting surface patches. (a) Imported CAD model in the user interface, (b) Selected patches to finish, (c) System generated coarse resolution trajectory, (d) System generated fine resolution trajectory

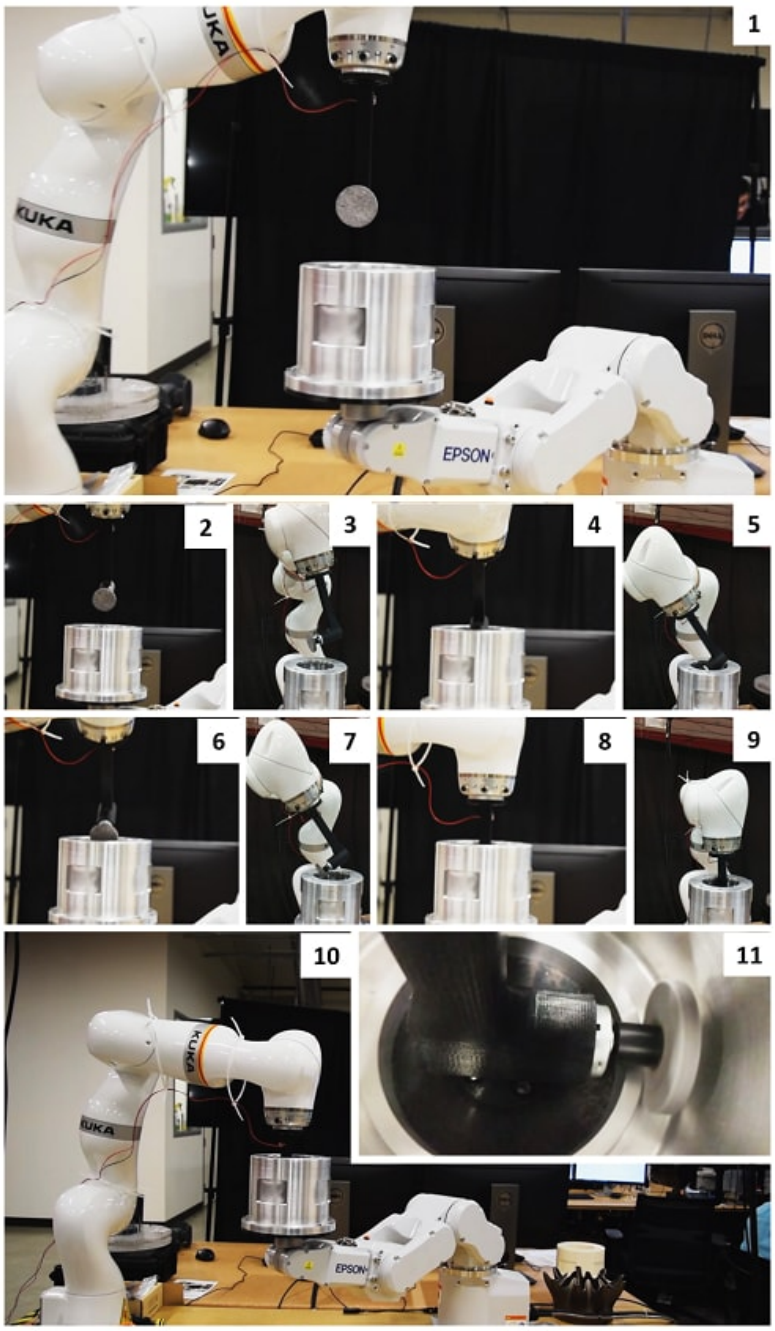


Figure 8.7: (1-10) Snaps from a footage of insertion trajectory on a test part. (11) Execution of finishing trajectory has started.

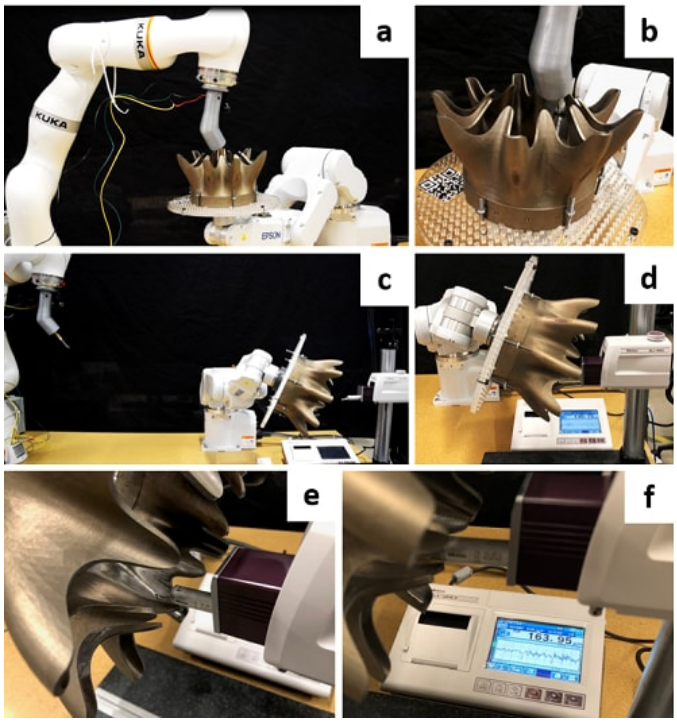


Figure 8.8: Snapshots from a footage of performance measurement using surface profilometer. (a,b) Ongoing robotic finishing of a part, (c,d) Holding robot takes the part to the surface profilometer to measure surface roughness, (e,f) Ongoing surface roughness measurement. The result is send back to the system as feedback.

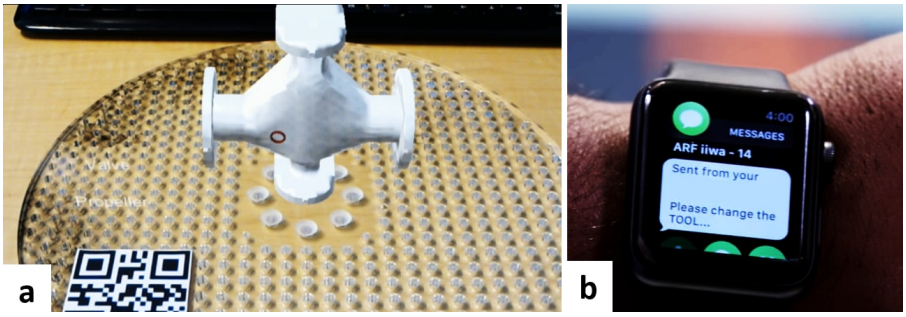


Figure 8.9: (a) A view through Hololens during part placement. Hologram of the part guides the operator while mounting the part. (b) A notification sent to the operator for contingency handling.

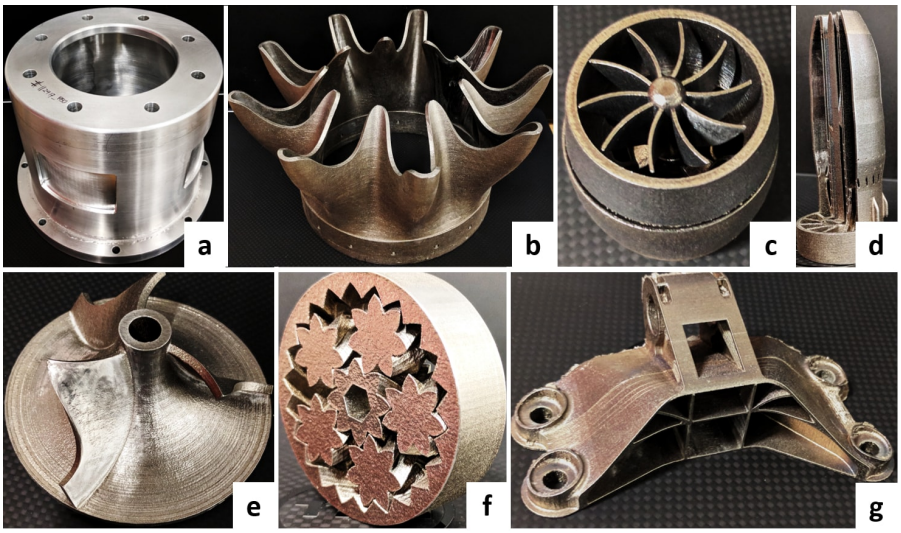


Figure 8.10: Some of the parts with complex geometry we used for robotic finishing. (a) Shaft Housing, (b) Lobbed Flow Mixer, (c) Miniature Turbine with Epicyclic Gear Train, (d) Fuel Injection Nozzle, (e) Pump Impeller, (f) Gear Bearing, (g) Jet Engine Bracket

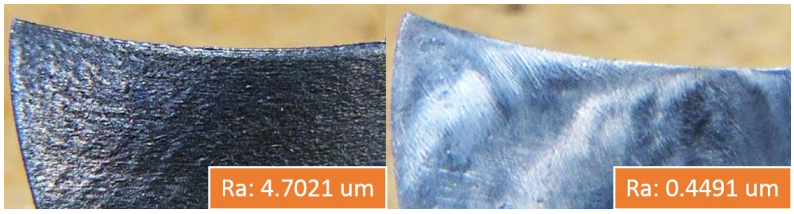


Figure 8.11: Surface roughness of a pump impeller blade before and after finishing.

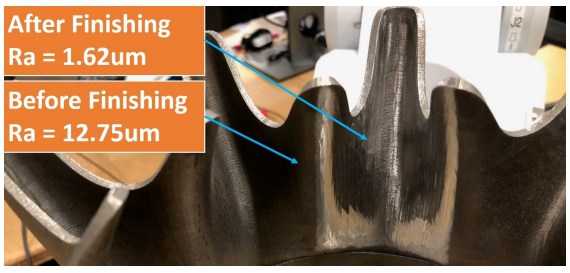


Figure 8.12: Surface roughness of finished and unfinished segments of the lobbed flow mixer.

Chapter 9

Conclusions

9.1 Intellectual Contributions

This dissertation develops computational foundations for trajectory planning for manipulators performing complex tasks. This dissertation presents a unified framework that integrates three traditionally diverse technologies- discrete state-space search, non-linear parametric optimization, and self-directed learning technologies to solve the problem of trajectory planning for manipulators performing complex tasks. The work presents advancements in each of these three technologies. Moreover, the work presents novel representations to be used with the unified framework for improving computational efficiency.

The work presents a context-dependent search strategy switching algorithm to enable automated heuristic selection for discrete state-space search. In a discrete state-space search, the computation time increases with an increase in the branching factor and the depth of the search tree. The developed algorithm reduces branching factor by adaptively selecting different heuristic functions for different regions in the state-space and by pruning nodes in the bottleneck regions. It reduces the depth of the search tree by adaptively scaling the action primitives in different representations of the state space (e.g., workspace and configuration space). The algorithm also backtracks in the search tree and repairs trajectories by connecting a node directly to precursor

nodes from different depths. Finally, the algorithm guides the search towards promising directions by analyzing the cost and diversity of the frontier node.

The work introduces a successive refinement strategy to identify feasible solutions using non-linear optimization in high dimensional parameter space with conflicting constraints. Single-stage optimization often fails to find a feasible solution in high-dimensional parameter space when there are conflicting constraints. It often gets trapped in infeasible local minima. The developed solution strategy solves a relaxed version of the constrained non-linear optimization problem at the beginning by activating a smaller number of constraints. It keeps on adding more constraints at every stage to funnel the initial seed to the final feasible solution. The constraints with larger attractor basins are applied in the earlier stage, and the constraints with smaller basin size are applied in the later stages to reduce the computation time for identifying a feasible solution.

The work presents a self-directed learning approach to identify optimal parameters by focusing on promising regions in the parameter-space without learning the complete model. The trajectory parameters of a robot influence the outcome of a physical process. The physics-based models are often not available for these processes. Conducting a large number of experiments to learn the complete model is often not feasible due to limitations in available resources. Moreover, learning the complete model may not be useful for non-repetitive tasks. The presented method utilizes probabilistic decision making to identify promising regions in the parameter space and estimate the optimal set of parameters to conduct a physical experiment from the candidate sets. The iterative approach enables the algorithm to identify the optimal parameters while minimizing the utilization of resources.

The key contributions from the chapters of this dissertation are summarized below.

- *Point-to-Point Trajectory Planning for Manipulators Operating in Confined Workspaces:* The work has developed algorithmic foundations to generate a point-to-point trajectory

for manipulators working in a confined workspace. The key contribution is the context-dependent search strategy switching (CODES3) algorithm. Manipulators have high dimensional state space. Therefore trajectory planning for manipulators is a p-space hard problem. The CODES3 algorithm can efficiently reduce the search space by pruning ineffective branches in the search tree and by reducing the depth of the search tree. The algorithm understands the context by analyzing neighborhood in the search space and history of search progress. It then determines the right heuristic to use from a set of available heuristics to come out of the bottleneck region and rapidly move towards the goal state. The algorithm adaptively scales and selects motion primitives. It improves the smoothness and execution time of the trajectory by backtracking through the search nodes. The results show significant improvement in computation time and trajectory execution time compared to established methods. This is expected to enable automated trajectory planning for manipulators in robotic finishing, assembly, maintenance, and service applications. Moreover, the CODES3 algorithm can be adapted for unmanned vehicles. This will enable unmanned vehicles to navigate through complex and unstructured environments efficiently.

- *Generation of Synchronized Configuration Space Trajectories with Workspace Path Constraints for Multi-Robot Systems:* The work has developed algorithmic foundations for generating constrained trajectories for synchronous motion among multiple robots. The method poses the problem of constrained trajectory generation for the synchronous motion of multi-robot systems as a non-linear optimization problem. The method determines appropriate parametric representation for the configuration variables, generates an approximate solution as a starting point for the optimization method, and uses successive refinement techniques to solve the problem in a computationally efficient manner. The work has demonstrated the effectiveness of our approach by solving a wide variety of complex synchronous motion generation problems with high-DOF robotic systems. We have demonstrated the effectiveness

of generating parametric trajectories with multiple spline segments over sequential inverse kinematics based approach. We have also demonstrated the superior performance of the successive refinement strategy over single-stage optimization on the simulation and physical test cases involving complex tasks performed by an ensemble of manipulators. Moreover, the developed algorithmic foundations for synchronous trajectory generation can be adapted for a team of unmanned vehicles for executing complex missions.

- *Bringing Setup Planning Considerations in Trajectory Planning:* The work has developed algorithmic foundations to generate a sequence of setups (relative poses between workpiece and manipulator) for completing the task in minimum time. A minimum number of setup may not guarantee minimum task-completion time. Therefore time taken to execute planned manipulator trajectories associated with the tasks under each setup is taken into account to evaluate the quality of a setup. A method combining sampling and optimization has been presented to generate candidate setups. A depth-first branch and bound search algorithm is presented to generate the optimal sequence of setups from the candidates. The results show that the selection of the right optimization subspace can improve the reachability score for candidate setups. This enabled identifying the optimal sequence of the poses of a workpiece such that the manipulator can reduce the time span of a given task. Moreover, the developed algorithmic foundations can be adapted for humanoid robots to enable their use in complex operations for space or disaster rescue applications.

- *Bringing Task-Agent Assignment Consideration in Trajectory Planning:* The work has developed algorithmic foundations to bring task-agent assignment consideration in motion planning. This aims to reduce the computation time for generating robot trajectories in the configuration space for the given tasks described through complex task networks. Validating arbitrary task-agent assignments by invoking motion planning routines is not computationally efficient. A novel architecture has been presented to reduce the number of calls made

to the motion planning routines by utilizing spatial constraint checking at key-frames of the given tasks. Moreover, the method uses a multi-tree representation to enable efficient caching and avoid repeated computations. The results show that the developed method is effective for operations that require complex interaction among agents and motion generation for high-DOF robotic systems, such as bi-manual mobile manipulators. Moreover, the developed algorithmic foundations can be adapted for a heterogeneous team of unmanned vehicles for challenging disaster rescue applications.

- *Identification of Trajectory and Process Parameters using Physical Experiments:* The work has developed algorithmic foundations for identifying trajectory and process parameters while minimizing process completion time. The method builds surrogate models for black-box task constraints based on exploratory experiments. These models are further improved during the refining experiments. The key contribution is the method for selecting parameters from the parameter space for refining experiments based on the estimated probability of success of optimizing task objective and satisfying task constraints. Proof has been introduced in this work to show that the method will converge faster to solution in the expected sense compared to any other method. The simulation experiments show the improved performance of the method compared to existing approaches. The physical experiments show the effectiveness of the method in process applications such as robotic finishing. Moreover, the developed algorithmic foundation can be adapted for unmanned vehicles to generate and execute complex trajectories in tight space.

9.2 Anticipated Benefits

This dissertation presents methods for trajectory planning to execute complex tasks using manipulators. It is anticipated that the research should be applicable to a wide variety of manufacturing

and service applications. It will enable robots to program themselves based on high-level instructions from the human operator. Having the robots generate motions for new tasks automatically without manual programming will have significant cost savings. Once the planning and learning framework is established, the robot can gain knowledge of a new task and execute it much faster than it would take an expert to program manually or through demonstrations by a human. Having the ability to program own trajectories and adjusting process parameters on the fly for complex tasks will allow the robot to do much more than is currently possible where such tasks are often impossible to manually program and infeasible to compute with a traditional planner.

It is anticipated that the contributions of this dissertation will allow manipulators to be used for high-mix, low volume production manufacturing and service applications. Potential applications include composite sheet layup, robotic finishing, assembly, bin-picking and kitting, additive manufacturing, and machine tending. Moreover, the planning and learning technologies developed in this dissertation can be adapted for unmanned vehicles and humanoid robots. This will enable the use of these robotic platforms in complex missions or tasks.

9.3 Future Directions

The presented work can be extended in the following directions.

- In the future, the point-to-point trajectory planning can be studied for manipulators with more than 7 degrees of freedom. Moreover, the search strategies can be learned and improved based on the analysis of human input. The current work assumes that there is no uncertainty associated with the obstacles. The method can be extended to handle cases where there is uncertainty in the obstacle estimations.

- The presented work on path-constrained trajectory generation adaptively selects parametric representation for the trajectories. However, it selects the same parametric representation for all the configuration space variables. The number of parameters required to sufficiently

represents each configuration variable may not be the same. In the future, the method can be extended to adaptively adjust the number of parameters required for each configuration variable independently to reduce the dimension of the parameter space. This will reduce the computation time for the optimization routines. Moreover, one can explore the possibility of improving the computation speed using analytical gradients and parallel computing.

The test cases presented in this work do not test the method for path-constrained trajectory generation in highly cluttered environments. The presented method may require significant computation time to generate trajectories in a cluttered environment. In the future, the method can be extended to be computationally efficient in a cluttered workspace. Moreover, the presented method can be extended to solve for robot-placement, grasp location selection, and trajectory generation problems simultaneously for complex tasks.

- The presented work on setup planning considers kinematic reachability of the manipulator while scoring the quality of the setups. In the future, one can introduce manipulability factors and dynamic capabilities in the scoring function to make it more robust for a wide range of practical applications. Moreover, the method can be extended for multiple manipulators and part placement problem for using as a work cell designing algorithm.

- In the future, the proposed three-layer architecture for bringing task-agent assignment considerations in trajectory planning can be integrated with a task planning framework to generate the task network from a set of high-level instructions. Moreover, asynchronous trajectory generation methods can be incorporated in the set of available motion planning technologies to broaden the scope of the work.

- In this work, the trajectory and process parameter identification method has been verified on problems with two to ten-dimensional parameter spaces and with one to eleven constraints. The benchmarking problems had single known objective functions and multiple black-box

constraint functions. The method can be extended to be used for the higher-dimensional problems with multiple black-box objective functions.

Reference List

[1] S. Le Digabel, "Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm," *ACM Transactions on Mathematical Software*, vol. 37, no. 4, pp. 1–15, 2011.

[2] R. G. Regis, "Stochastic radial basis function algorithms for large-scale optimization involving expensive black-box objective and constraint functions," *Computers & Operations Research*, vol. 38, no. 5, pp. 837 – 853, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S030505481000208X

[3] A. M. Kabir, A. V. Shembekar, R. K. Malhan, R. S. Aggarwal, B. C. Langsfeld, J. D.and Shah, and S. K. Gupta, "Robotic finishing of interior regions of geometrically complex parts," in *ASMEs 13th Manufacturing Science and Engineering Conference*, College Station, Texas, USA, June 2018.

[4] P. M. Bhatt, A. M. Kabir, R. K. Malhan, A. V. Shembekar, B. C. Shah, and S. K. Gupta, "Concurrent design of tool-paths and impedance controllers for performing area coverage operations in manufacturing applications under uncertainty," in *IEEE Conference on Automation Science and Engineering*. IEEE, 2019.

[5] W. Chen, Y. Tang, and Q. Zhao, "A novel trajectory planning scheme for spray painting robot with bézier curves," in *Control and Decision Conference (CCDC), 2016 Chinese*. IEEE, 2016, pp. 6746–6750.

[6] H.-T. Yau and C.-H. Menq, "Automated cmm path planning for dimensional inspection of dies and molds having complex surfaces," *International Journal of Machine Tools and Manufacture*, vol. 35, no. 6, pp. 861–876, 1995.

[7] B. Shirinzadeh, G. Cassidy, D. Oetomo, G. Alici, and M. H. Ang Jr, "Trajectory generation for open-contoured structures in robotic fibre placement," *Robotics and Computer-Integrated Manufacturing*, vol. 23, no. 4, pp. 380–394, 2007.

[8] R. K. Malhan, A. M. Kabir, A. V. Shembekar, B. Shah, T. Centea, and S. K. Gupta, "Hybrid cells for multi-layer prepreg composite sheet layup," in *IEEE International Conference on Automation Science and Engineering (CASE)*, Munich, Germany, Aug 2018.

[9] A. V. Shembekar, Y. J. Yoon, A. Kanyuck, and S. K. Gupta, "Trajectory planning for conformal 3d printing using non-planar layers," in *ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 1A: 38th Computers and Information in Engineering Conference. American Society of Mechanical Engineers, 2018, p. V01AT02A026.

[10] P. M. Bhatt, M. Peralta, H. A. Bruck, and S. K. Gupta, "Robot assisted additive manufacturing of thin multifunctional structures," in *International Manufacturing Science and Engineering Conference*. ASME, 2018.

[11] P. M. Bhatt, A. M. Kabir, R. K. Malhan, B. C. Shah, A. V. Shembekar, Y. J. Yoon, and S. K. Gupta, "A robotic cell for multi-resolution additive manufacturing," in *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019.

[12] P. M. Bhatt, R. K. Malhan, and S. K. Gupta, "Computational foundations for using three degrees of freedom build platforms to enable supportless extrusion-based additive manufacturing," in *ASMEs 14th Manufacturing Science and Engineering Conference*, Erie, PA, USA, June 2019.

[13] Y. J. Yoon, M. Yon, S. E. Jung, and S. K. Gupta, "Development of Three-Nozzle Extrusion System for Conformal Multi-Resolution 3D Printing with a Robotic Manipulator," in *ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. Anaheim, California: American Society of Mechanical Engineers, August 2019.

[14] A. Shembekar, Y. J. Yoon, A. Kanyuck, and S. K. Gupta, "Generating robot trajectories for conformal 3d printing using non-planar layers," *Journal of Computing and Information Science in Engineering*, vol. 19, pp. 031 011–031 011–13, 2019.

[15] P. M. Bhatt, P. Rajendran, K. McKay, and S. K. Gupta, "Context-dependent compensation scheme to reduce trajectory execution errors for industrial manipulators," in *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019.

[16] P. M. Bhatt, A. M. Kabir, M. Peralta, H. A. Bruck, and S. K. Gupta, "A robotic cell for performing sheet lamination-based additive manufacturing," *Additive Manufacturing*, 2019.

[17] H. Li and D. Ceglarek, "Optimal trajectory planning for material handling of compliant sheet metal parts," *Journal of Mechanical Design*, vol. 124, no. 2, pp. 213–222, 2002.

[18] R. K. Malhan, A. M. Kabir, B. Shah, T. Centea, and S. K. Gupta, "Automated prepreg sheet placement using collaborative robotics," in *North America Society for the Advancement of Material and Process Engineering (SAMPE) Long beach conference*, CA, USA, May 2018.

[19] ——, "Determining feasible robot placements in robotic cells for composite prepreg sheet layup," in *ASMEs 14th Manufacturing Science and Engineering Conference*, Erie, PA, USA, June 2019.

[20] S. Thakar, L. Fang, B. Shah, and S. K. Gupta, "Towards time-optimal trajectory planning for pick-and-transport operation with a mobile manipulator," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2018, pp. 981–987.

[21] T. Hermansson, R. Bohlin, J. S. Carlson, and R. Sderberg, "Automatic assembly path planning for wiring harness installations," *Journal of Manufacturing Systems*, vol. 32, no. 3, pp. 417 – 422, 2013, assembly Technologies and Systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0278612513000393

[22] R. K. Malhan, A. M. Kabir, Y. Shahapurkar, B. Shah, and S. K. Gupta, "Integrating impedance control and learning based search scheme for robotic assemblies under uncertainty," in *ASMEs 13th Manufacturing Science and Engineering Conference*, Texas, USA, June 2018.

[23] S. Shriyam and S. K. Gupta, "Modeling and analysis of subsystem interactions in robotic assembly," in *Proceedings of ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2019.

[24] P. Rajendran, S. Thakar, and S. Gupta, "User-guided path planning for redundant manipulators in highly constrained work environments," in *IEEE International Conference on Automation Science and Engineering (CASE)*, Vancouver, Canada, August 2019.

[25] P. Rajendran, S. Thakar, A. Kabir, B. Shah, and S. K. Gupta, "Context-dependent search for generating paths for redundant manipulators in cluttered environments," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, Macau, China, November 2019.

[26] V. Annem, P. Rajendran, S. Thakar, and S. K. Gupta, "Towards remote teleoperation of a semi-autonomous mobile manipulator system in machine tending tasks," in *ASME Manufacturing Science and Engineering Conference (MSEC)*, Erie, USA, June 2019.

[27] S. Thakar, A. Kabir, P. Bhatt, R. Malhan, P. Rajendran, B. Shah, and S. K. Gupta, "Task assignment and motion planning for bi-manual mobile manipulation," in *IEEE International Conference on Automation Science and Engineering (CASE)*, Vancouver, Canada, August 2019.

[28] N. B. Kumbla, S. Thakar, K. N. Kaipa, J. Marvel, and S. K. Gupta, "Simulation based on-line evaluation of singulation plans to handle perception uncertainty in robotic bin picking," in *ASME 2017 12th International Manufacturing Science and Engineering Conference collocated with the JSME/ASME 2017 6th International Conference on Materials and Processing*, 2017, pp. V003T04A002–V003T04A002.

[29] ——, "Handling perception uncertainty in simulation-based singulation planning for robotic bin picking," *Journal of computing and information science in engineering*, vol. 18, no. 2, p. 021004, 2018.

[30] K. N. Kaipa, S. S. Thevendria-Karthic, S. Shriyam, A. M. Kabir, J. D. Langsfeld, and S. K. Gupta, "Resolving automated perception system failures in bin-picking tasks using assistance from remote human operators," in *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, Aug 2015, pp. 1453–1458.

[31] C. W. Warren, J. C. Danos, and B. W. Mooring, "An Approach To Manipulator Path Planning," *Van Nostrand Reinhold. Schutz, B. Geometrical Methods of Mathematical Physics*, 1981.

[32] L. E. Kavraki, P. Svec, J.-P. Laumond, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration space," *IEEE Transactions on Robotics*, 1996.

[33] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation*, vol. 2, 2000, pp. 995–1001.

[34] S. M. Lavalle and J. J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, 2001.

[35] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1478–1483.

[36] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International journal of robotics research*, vol. 34, no. 7, pp. 883–921, 2015.

[37] 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 *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 3067–3074.

[38] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic Trajectory Optimization for Motion Planning," in *International Conference on Robotics and Automation*, 2011.

[39] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, S. S. Srinivasa, P. Pa, and P. Pa, "CHOMP: Covariant Hamiltonian Optimization for Motion Planning," *The International Journal of Robotics Research*, 2012.

[40] Z. Yao and K. Gupta, "Path planning with general end-effector constraints: Using task space to guide configuration space search," *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pp. 2211–2216, 2005.

[41] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Discrete Search Leading Continuous Exploration for Kinodynamic Motion Planning," in *Robotics: Science and Systems*, 2007.

[42] B. J. Cohen, S. Chitta, and M. Likhachev, "Search-based planning for manipulation with motion primitives," in *International Conference on Robotics and Automation*, 2010.

[43] K. Gochev, B. Cohen, J. Butzke, A. Safonova, and M. Likhachev, "Path Planning with Adaptive Dimensionality," in *The Symposium on Combinatorial Search (SoCS)*, 2011.

[44] B. J. Cohen, G. Subramanian, S. Chitta, and M. Likhachev, "Planning for Manipulation with Adaptive Motion Primitives," in *International Conference on Robotics and Automation*, 2011.

[45] K. Gochev, V. Narayanan, B. Cohen, A. Safonova, and M. Likhachev, "Motion Planning for Robotic Manipulators with Independent Wrist Joints," in *International Conference on Robotics and Automation*, 2014.

[46] F. Islam, V. Narayanan, and M. Likhachev, "Dynamic Multi-Heuristic A *," in *International Conference on Robotics and Automation*, 2015, pp. 2376–2382.

[47] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Backward-forward search for manipulation planning," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2015-Decem, pp. 6366–6373, 2015.

[48] F. Islam, V. Narayanan, and M. Likhachev, "A*-connect: Bounded suboptimal bidirectional heuristic search," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 2752–2758.

[49] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev, "Multi-heuristic a," *The International Journal of Robotics Research*, vol. 35, no. 1-3, pp. 224–243, 2016.

[50] 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," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 2997–3004.

[51] V. Narayanan, S. Aine, and M. Likhachev, "Improved multi-heuristic a* for searching with uncalibrated heuristics," in *Eighth Annual Symposium on Combinatorial Search*, 2015.

[52] J. Bobrow, S. Dubowsky, and J. Gibson, "Time-optimal control of robotic manipulators along specified paths," *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 3–17, 1985.

[53] O. Egeland, "Task-space tracking with redundant manipulators," *IEEE Journal on Robotics and Automation*, vol. 3, no. 5, pp. 471–475, 1987.

[54] M. Galicki, "Time-optimal controls of kinematically redundant manipulators with geometric constraints," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 1, pp. 89–93, Feb 2000.

[55] F. Pfeiffer and R. Johanni, "A concept for manipulator trajectory planning," *IEEE Journal on Robotics and Automation*, vol. 3, no. 2, pp. 115–123, 1987.

[56] D. Constantinescu and E. A. Croft, "Smooth and time-optimal trajectory planning for industrial manipulators along specified paths," *Journal of robotic systems*, vol. 17, no. 5, pp. 233–249, 2000.

[57] A. Piazzi and A. Visioli, "Global minimum-jerk trajectory planning of robot manipulators," *IEEE Transactions on Industrial Electronics*, vol. 47, no. 1, pp. 140–149, Feb 2000.

[58] T. Chettibi, H. Lehtihet, M. Haddad, and S. Hanchi, "Minimum cost trajectory planning for industrial robots," *European Journal of Mechanics-A/Solids*, vol. 23, no. 4, pp. 703–715, 2004.

[59] A. Gasparetto and V. Zanotto, "A new method for smooth trajectory planning of robot manipulators," *Mechanism and machine theory*, vol. 42, no. 4, pp. 455–471, 2007.

[60] C. De Boor, *A practical guide to splines*. Springer-Verlag New York, 1978, vol. 27.

[61] J. Kieffer, "Manipulator inverse kinematics for untimed end-effector trajectories with ordinary singularities," *The International journal of robotics research*, vol. 11, no. 3, pp. 225–237, 1992.

[62] B. J. Martin and J. E. Bobrow, "Minimum effort motions for open chain manipulators with task-dependent end-effector constraints," in *Proceedings of International Conference on Robotics and Automation*, vol. 3, Apr 1997, pp. 2044–2049 vol.3.

[63] ——, "Minimum-effort motions for open-chain manipulators with task-dependent end-effector constraints," *The International Journal of Robotics Research*, vol. 18, no. 2, pp. 213–224, 1999.

[64] E. S. Conkur, "Path following algorithm for highly redundant manipulators," *Robotics and Autonomous Systems*, vol. 45, no. 1, pp. 1 – 22, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889003000836

[65] "Descartes- a ros-industrial project for performing path-planning on under-defined cartesian trajectories," http://wiki.ros.org/descartes, accessed: 2018-05-21.

[66] M. Cefalo, G. Oriolo, and M. Vendittelli, "Planning safe cyclic motions under repetitive task constraints," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 3807–3812.

[67] M. Tarokh and X. Zhang, "Real-time motion tracking of robot manipulators using adaptive genetic algorithms," *Journal of Intelligent & Robotic Systems*, vol. 74, no. 3-4, pp. 697–708, 2014.

[68] X. Shi, H. Fang, and L. Guo, "Multi-objective optimal trajectory planning of manipulators based on quintic nurbs," in *Mechatronics and Automation (ICMA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 759–765.

[69] L. Piegl and W. Tiller, *The NURBS book*. Springer Science & Business Media, 2012.

[70] J. Huang, P. Hu, K. Wu, and M. Zeng, "Optimal time-jerk trajectory planning for industrial robots," *Mechanism and Machine Theory*, vol. 121, pp. 530–544, 2018.

[71] R. Menasri, A. Nakib, B. Daachi, H. Oulhadj, and P. Siarry, "A trajectory planning of redundant manipulators based on bilevel optimization," *Applied Mathematics and Computation*, vol. 250, pp. 934–947, 2015.

[72] S. S. M. Salehian, N. Figueroa, and A. Billard, "A unified framework for coordinated multi-arm motion planning," *The International Journal of Robotics Research*, vol. 0, no. 0, p. 0278364918765952, 0.

[73] M. Stilman, "Task constrained motion planning in robot joint space," *IEEE International Conference on Intelligent Robots and Systems*, pp. 3074–3081, 2007.

[74] ——, "Global manipulation planning in robot joint space with task constraints," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 576–584, June 2010.

[75] R. Colombo, F. Gennari, V. Annem, P. Rajendran, S. Thakar, L. Bascetta, and S. K. Gupta, "Parameterized model predictive control of a nonholonomic mobile manipulator: A terminal constraint-free approach," in *IEEE International Conference on Automation Science and Engineering (CASE)*, Vancouver, Canada, August 2019.

[76] S. Thakar, P. Rajendran, V. Annem, A. Kabir, and S. K. Gupta, "Accounting for part pose estimation uncertainties during trajectory generation for part pick-up using mobile manipulators," in *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019.

[77] S. Dalibard, A. Nakhaei, F. Lamiraux, and J. P. Laumond, "Whole-body task planning for a humanoid robot: a way to integrate collision avoidance," in *2009 9th IEEE-RAS International Conference on Humanoid Robots*, Dec 2009, pp. 355–360.

[78] A. Dietrich, T. Wimbck, and A. Albu-Schffer, "Dynamic whole-body mobile manipulation with a torque controlled humanoid robot via impedance control laws," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2011, pp. 3199–3206.

[79] D. Leidner, A. Dietrich, F. Schmidt, C. Borst, and A. Albu-Schffer, "Object-centered hybrid reasoning for whole-body mobile manipulation," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 1828–1835.

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

[81] J. Alonso-Mora, R. Knepper, R. Siegwart, and D. Rus, "Local motion planning for collaborative multi-robot manipulation of deformable objects," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 5495–5502.

[82] B. Buml, F. Schmidt, T. Wimbck, O. Birbach, A. Dietrich, M. Fuchs, W. Friedl, U. Frese, C. Borst, M. Grebenstein, O. Eiberger, and G. Hirzinger, "Catching flying balls and preparing coffee: Humanoid rollin'justin performs dynamic and sensitive tasks," in *IEEE International Conference on Robotics and Automation*, May 2011, pp. 3443–3444.

[83] M. Giftthaler, F. Farshidian, T. Sandy, L. Stadelmann, and J. Buchli, "Efficient kinematic planning for mobile manipulators with non-holonomic constraints using optimal control," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 3411–3417.

[84] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *The International Journal of Robotics Research*, vol. 33, no. 7, pp. 1006–1028, 2014.

[85] A. Reiter, A. Müller, and H. Gattringer, "On higher-order inverse kinematics methods in time-optimal trajectory planning for kinematically redundant manipulators," *IEEE Transactions on Industrial Informatics*, 2018.

[86] D. M. Bodily, T. F. Allen, and M. D. Killpack, "Motion planning for mobile robots using inverse kinematics branching," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5043–5050.

[87] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," *IEEE Journal of Robotics and Automation*, vol. 17, no. 1-19, p. 16, 2004.

[88] A. Reiter, A. Mller, and H. Gattringer, "Inverse kinematics in minimum-time trajectory planning for kinematically redundant manipulators," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Oct 2016, pp. 6873–6878.

[89] V. Falkenhahn, F. A. Bender, A. Hildebrandt, R. Neumann, and O. Sawodny, "Online tcp trajectory planning for redundant continuum manipulators using quadratic programming," in *Advanced Intelligent Mechatronics (AIM), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1163–1168.

[90] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *2009 IEEE International Conference on Robotics and Automation*, May 2009, pp. 625–632.

[91] D. Berenson, J. Chestnutt, S. S. Srinivasa, J. J. Kuffner, and S. Kagami, "Pose-constrained whole-body planning using task space region chains," in *9th IEEE-RAS International Conference on Humanoid Robots*, Dec 2009, pp. 181–187.

[92] K. Shankar, J. W. Burdick, and N. H. Hudson, *A Quadratic Programming Approach toQuasi-Static Whole-Body Manipulation*. Cham: Springer International Publishing, 2015, pp. 553–570. [Online]. Available: https://doi.org/10.1007/978-3-319-16595-0_32

[93] I. A. Vasilyev and A. M. Lyashin, "Analytical solution to inverse kinematic problem for 6-dof robot-manipulator," *Automation and Remote Control*, vol. 71, no. 10, pp. 2195–2199, Oct 2010. [Online]. Available: https://doi.org/10.1134/S0005117910100218

[94] J. D. Sun, G. Z. Cao, W. B. Li, Y. X. Liang, and S. D. Huang, "Analytical inverse kinematic solution using the d-h method for a 6-dof robot," in *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, June 2017, pp. 714–716.

[95] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," *IEEE Journal of Robotics and Automation*, vol. 17, no. 1-19, p. 16, 2004.

[96] R. Smits, "KDL: Kinematics and Dynamics Library," http://www.orocos.org/kdl.

[97] P. Beeson and B. Ames, "Trac-ik: An open-source library for improved solving of generic inverse kinematics," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, Nov 2015, pp. 928–935.

[98] F. Zacharias, C. Borst, and G. Hirzinger, "Capturing robot workspace structure: representing robot capabilities," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2007, pp. 3229–3236.

[99] P. Anderson-Sprecher and R. Simmons, "Voxel-based motion bounding and workspace estimation for robotic manipulators," in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 2141–2146.

[100] H. Tubig, B. Buml, and U. Frese, "Real-time swept volume and distance computation for self collision detection," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2011, pp. 1585–1592.

[101] H. Sakurai, "Automatic setup planning and fixture design for machining," *Journal of Manufacturing Systems*, vol. 11, no. 1, pp. 30 – 37, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0278612592900158

[102] S. H. Huang, "Automated setup planning for lathe machining," *Journal of Manufacturing Systems*, vol. 17, no. 3, pp. 196 – 208, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0278612598800617

[103] X. G. Ming and K. L. Mak, "Intelligent setup planning in manufacturing by neural networks based approach," *Journal of Intelligent Manufacturing*, vol. 11, no. 3, pp. 311–333, Jun 2000.

[104] S. K. Ong, J. Ding, and A. Y. C. Nee, "Hybrid ga and sa dynamic set-up planning optimization," *International Journal of Production Research*, vol. 40, no. 18, pp. 4697–4719, 2002.

[105] P. Gaoliang, L. Wenjian, and Z. Yuru, "Intelligent setup planning in manufacturing by fuzzy set theory based approach," in *IEEE International Conference on Automation Science and Engineering, 2005.*, Aug 2005, pp. 130–135.

[106] S. Kafashi, M. Shakeri, and V. Abedini, "Automated setup planning in capp: a modified particle swarm optimisation-based approach," *International Journal of Production Research*, vol. 50, no. 15, pp. 4127–4140, 2012.

[107] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *IEEE international conference on robotics and automation (ICRA)*, 2014, pp. 639–646.

[108] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical planning in the now," in *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[109] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[110] C. Reed Garrett, T. Lozano-Pérez, and L. Pack Kaelbling, "FFRob: Leveraging symbolic planning for efficient task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 104–136.

[111] L. De Silva, A. K. Pandey, M. Gharbi, and R. Alami, "Towards combining htn planning and geometric task planning," *arXiv preprint arXiv:1307.1482*, 2013.

[112] J. Bidot, L. Karlsson, F. Lagriffoul, and A. Saffiotti, "Geometric backtracking for combined task and motion planning in robotic systems," *Artificial Intelligence*, vol. 247, pp. 229–265, jun 2017.

[113] F. Lagriffoul and B. Andres, "Combining task and motion planning: A culprit detection problem," *The International Journal of Robotics Research*, vol. 35, no. 8, pp. 890–927, 2015.

[114] F. Lagriffoul, N. Dantam, C. Garrett, A. Akbari, S. Srivastava, and L. Kavraki, "Platform-independent benchmarks for task and motion planning," *IEEE Robotics and Automation Letters*, vol. 3, pp. 3765–3772, Oct. 2018.

[115] R. Chitnis, D. Hadfield-Menell, A. Gupta, S. Srivastava, E. Groshev, C. Lin, and P. Abbeel, "Guided search for task and motion plans using learned heuristics," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, 2016, pp. 447–454.

[116] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Ffrob: An efficient heuristic for task and motion planning," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 179–195.

[117] A. Wells, N. Dantam, A. Shrivastava, and L. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *IEEE Robotics and Automation Letters*, 2019.

[118] S. Alatartsev, S. Stellmacher, F. Ortmeier, S. Alatartsev, S. Stellmacher, and ·. F. Ortmeier, "Robotic Task Sequencing Problem: A Survey," *Journal of Intelligent Robotic Systems*, vol. 80, pp. 279–298, 2015.

[119] S. Alatartsev, V. Mersheeva, M. Augustine, and F. Ortmeier, "On optimizing a sequence of robotic tasks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013, pp. 217–223.

[120] C. Zhang and J. A. Shah, "Co-optimizating multi-agent placement with task assignment and scheduling." in *IJCAI*, 2016, pp. 3308–3314.

[121] N. Michael, M. M. Zavlanos, V. Kumar, and G. J. Pappas, "Distributed multi-robot task assignment and formation control," *IEEE International Conference on Robotics and Automation*, vol. 1, pp. 128–133, 2008.

[122] J. McLurkin and D. Yamins, "Dynamic task assignment in robot swarms." in *Robotics: Science and Systems*, vol. 8, no. 2005. Citeseer, 2005.

[123] L. Luo, N. Chakraborty, and K. Sycara, "Distributed algorithm design for multi-robot task assignment with deadlines for tasks," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 3007–3013.

[124] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *The International Journal of Robotics Research*, vol. 28, no. 1, pp. 104–126, 2009.

[125] L. Karlsson, J. Bidot, F. Lagriffoul, A. Saffiotti, U. Hillenbrand, and F. Schmidt, "Combining task and path planning for a humanoid two-arm robotic system," in *Proceedings of tampra: Combining task and motion planning for real-world applications (icaps workshop)*. Citeseer, 2012, pp. 13–20.

[126] F. Basile, F. Caccavale, P. Chiacchio, J. Coppola, and C. Curatella, "Task-oriented motion planning for multi-arm robotic systems," *Robotics and Computer Integrated Manufacturing*, vol. 28, pp. 569–582, 2012.

[127] J. Wolfe, B. Marthi, and S. Russell, "Combined task and motion planning for mobile manipulation," in *Twentieth International Conference on Automated Planning and Scheduling*, 2010.

[128] K. Harada, T. Tsuji, K. Kikuchi, K. Nagata, H. Onda, and Y. Kawai, "Base position planning for dual-arm mobile manipulators performing a sequence of pick-and-place tasks," in *IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids) 2015*. IEEE, pp. 194–201.

[129] E. Ukar, A. Lamikiz, L. N. L. Lacalle, F. Liebana, and J. M. Etayo, "Laser polishing parameter optimization for die and moulds surface finishing," in *ASME International Manufacturing Science and Engineering Conference*, October 2008.

[130] R. Pavel, X. Wang, and A. K. Srivastava, "Multi-constraint optimization for grinding nickel-based alloys," in *ASME International Manufacturing Science and Engineering Conference*, June 2013.

[131] R. V. Rao and P. J. Pawar, "Parameter optimization of a multi-pass milling process using non-traditional optimization algorithms," *Applied Soft Computing*, vol. 10, no. 2, pp. 445 – 456, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S156849460900132X

[132] I. Mukherjee and P. K. Ray, "A review of optimization techniques in metal cutting processes," *Computers and Industrial Engineering*, vol. 50, no. 1-2, pp. 15 – 34, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360835205001403

[133] Y. Liansheng, Z. Yi, L. Xuemei, L. Aiping, and Z. Ji, "The study on the method of film thickness control by coating spray robot for ocean engineering," in *ASME International Conference on Ocean, Offshore and Arctic Engineering*, June 2015.

[134] W. R. Longhurst, A. M. Strauss, and G. E. Cook, "The identification of the key enablers for force control of robotic friction stir welding," in *ASME Journal of Manufacturing Science and Engineering*, vol. 133, no. 3, 2011, p. 031008.

[135] M. Asadi, J. Goldak, and A. Weck, "Welding distortion can be mitigated if welding current and traveling speed vary optimized along a weld path," in *ASME Pressure Vessels and Piping Conference*, July 2014.

[136] J. D. Langsfeld, A. M. Kabir, K. N. Kaipa, and S. K. Gupta, "Integration of planning and deformation model estimation for robotic cleaning of elastically deformable objects," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 352–359, Jan 2018.

[137] ——, "Robotic bimanual cleaning of deformable objects with online learning of part and tool models," in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, Fort Worth, Texas, USA, Aug 2016, pp. 626–632.

[138] ——, "Online learning of part deformation models in robotic cleaning of compliant objects," in *ASMEs 11th Manufacturing Science and Engineering Conference*, no. 49903, Blacksburg, Virginia, USA, June 2016, p. V002T04A003. [Online]. Available: http://dx.doi.org/10.1115/MSEC2016-8663

[139] N. Yamanobe, H. Fujii, Y. Maeda, T. Arai, A. Watanabe, T. Kato, T. Sato, and K. Hatanaka, "Optimization of damping control parameters for cycle time reduction in clutch assembly," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Aug 2005, pp. 3251–3256.

[140] B. Zhang, D. Gravel, G. Zhang, J. Wang, and A. Bell, "Robotic force control assembly parameter optimization for adaptive production," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2011, pp. 464–469.

[141] H. Cheng and H. Chen, "Online parameter optimization in robotic force controlled assembly processes," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 3465–3470.

[142] L. Ilzarbe, M. J. Alvarez, E. Viles, and M. Tanco, "Practical applications of design of experiments in the field of engineering: a bibliographical review," *Quality and Reliability Engineering International*, vol. 24, no. 4, pp. 417–428, 2008. [Online]. Available: http://dx.doi.org/10.1002/qre.909

[143] T. Zhao, Y. Shi, X. Lin, J. Duan, P. Sun, and J. Zhang, "Surface roughness prediction and parameters optimization in grinding and polishing process for ibr of aero-engine," *The International Journal of Advanced Manufacturing Technology*, vol. 74, no. 5, pp. 653–663, 2014. [Online]. Available: http://dx.doi.org/10.1007/s00170-014-6020-3

[144] X. Ren, M. Cabaravdic, X. Zhang, and B. Kuhlenktter, "A local process model for simulation of robotic belt grinding," *International Journal of Machine Tools and Manufacture*, vol. 47, no. 6, pp. 962 – 970, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0890695506001866

[145] N. Alagumurthi, K. Palaniradja, and V. Soundararajan, "Optimization of grinding process through design of experiment (doe) - a comparative study," *Materials and Manufacturing Processes*, vol. 21, no. 1, pp. 19–21, 2006. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/AMP-200060605

[146] K. V. B. S. K. Kumar and S. K. Choudhury, "Investigation of tool wear and cutting force in cryogenic machining using design of experiments," *Journal of Materials Processing Technology*, vol. 203, no. 1-3, pp. 95 – 101, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0924013607009302

[147] P. Lahaye, C. Chomont, P. Dumont, J. Duchesne, and G. Chabassier, "Using a design of experiment method to improve kdp crystal machining process," pp. 814–820, 1999. [Online]. Available: http://dx.doi.org/10.1117/12.354197

[148] T. Wang, X. Lu, D. Zhao, Y. He, and J. Luo, "Optimization of design of experiment for chemical mechanical polishing of a 12-inch wafer," *Microelectronic Engineering*, vol. 112, pp. 5 – 9, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016793171300498X

[149] H. T. Liao, J. R. Shie, and Y. K. Yang, "Applications of taguchi and design of experiments methods in optimization of chemical mechanical polishing process parameters," *The International Journal of Advanced Manufacturing Technology*, vol. 38, no. 7, pp. 674–682, 2008. [Online]. Available: http://dx.doi.org/10.1007/s00170-007-1124-7

[150] S. C. Tam, N. L. Loh, C. P. A. Mah, and N. H. Loh, "Electrochemical polishing of biomedical titanium orifice rings," *Journal of Materials Processing Technology*, vol. 35, no. 1, pp. 83 – 91, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/092401369290303A

[151] D. Gravel, G. Zhang, A. Bell, and B. Zhang, "Objective metric study for doe-based parameter optimization in robotic torque converter assembly," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2009, pp. 3832–3837.

[152] G. Zhang, A. Bell, H. Zhang, J. He, J. Wang, and C. Martinez, "On-pendant robotic assembly parameter optimization," in *2008 7th World Congress on Intelligent Control and Automation*, June 2008, pp. 547–552.

[153] S. Elangovan, K. Prakasan, and V. Jaiganesh, "Optimization of ultrasonic welding parameters for copper to copper joints using design of experiments," *The International Journal of Advanced Manufacturing Technology*, vol. 51, no. 1, pp. 163–171, 2010. [Online]. Available: http://dx.doi.org/10.1007/s00170-010-2627-1

[154] T. R. Bement, "Taguchi techniques for quality engineering," *Technometrics*, vol. 31, no. 2, pp. 253–255, 1989. [Online]. Available: http://dx.doi.org/10.1080/00401706.1989.10488519

[155] H. Rowlands and J. Antony, "Application of design of experiments to a spot welding process," *Assembly Automation*, vol. 23, no. 3, pp. 273–279, 2003. [Online]. Available: http://dx.doi.org/10.1108/01445150310486549

[156] H. Liao and J. Shie, "Optimization on selective laser sintering of metallic powder via design of experiments method," *Rapid Prototyping Journal*, vol. 13, no. 3, pp. 156–162, 2007. [Online]. Available: http://dx.doi.org/10.1108/13552540710750906

[157] C. E. Rasmussen, "Gaussian processes for machine learning," 2006.

[158] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "Boa: The bayesian optimization algorithm," in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1*, ser. GECCO'99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 525–532. [Online]. Available: http://dl.acm.org/citation.cfm?id=2933923.2933973

[159] B. Wu, D. Qu, and F. Xu, "Improving efficiency with orthogonal exploration for online robotic assembly parameter optimization," in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Dec 2015, pp. 958–963.

[160] R. R. Barton and M. Meckesheimer, "Chapter 18 metamodel-based simulation optimization," in *Simulation*, ser. Handbooks in Operations Research and Management Science, S. G. Henderson and B. L. Nelson, Eds. Elsevier, 2006, vol. 13, pp. 535 – 574. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0927050706130182

[161] J. A. Marvel, W. S. Newman, D. P. Gravel, G. Zhang, J. Wang, and T. Fuhlbrigge, "Automated learning for parameter optimization of robotic assembly tasks utilizing genetic algorithms," in *2008 IEEE International Conference on Robotics and Biomimetics*, Feb 2009, pp. 179–184.

[162] P. J. Pawar and R. V. Rao, "Parameter optimization of machining processes using teaching–learning-based optimization algorithm," *The International Journal of Advanced Manufacturing Technology*, vol. 67, no. 5, pp. 995–1006, 2013. [Online]. Available: http://dx.doi.org/10.1007/s00170-012-4524-2

[163] R. V. Rao and V. D. Kalyankar, "Parameter optimization of machining processes using a new optimization algorithm," *Materials and Manufacturing Processes*, vol. 27, no. 9, pp. 978–985, 2012. [Online]. Available: http://dx.doi.org/10.1080/10426914.2011.602792

[164] C. Audet and J. E. Dennis, Jr., "Mesh adaptive direct search algorithms for constrained optimization," *SIAM Journal on Optimization*, vol. 17, no. 1, pp. 188–217, 2006. [Online]. Available: http://dx.doi.org/doi:10.1137/040603371

[165] M. A. Abramson and C. Audet, "Convergence of mesh adaptive direct search to second-order stationary points," *SIAM Journal on Optimization*, vol. 17.2, no. 2, pp. 606–619, 2006.

[166] A. M. Kabir, B. C. Shah, and S. K. Gupta, "Trajectory planning for manipulators operating in confined workspaces," in *2018 IEEE International Conference on Automation Science and Engineering (CASE)*, Munich, Germany, Aug 2018.

[167] J. H. Reif, "Complexity of the mover's problem and generalizations," in *Foundations of Computer Science, 1979., 20th Annual Symposium on.* IEEE, 1979, pp. 421–427.

[168] J. A. Sethian, "Fast marching methods," *SIAM review*, vol. 41, no. 2, pp. 199–235, 1999.

[169] A. Kabir, B. Shah, and S. K. Gupta, "Simulation and physical experiments using "codes3" algorithm," March 2018. [Online]. Available: https://youtu.be/38746EJLqOI

[170] P. Beeson and B. Ames, "TRAC-IK: An open-source library for improved solving of generic inverse kinematics," in *Proceedings of the IEEE RAS Humanoids Conference*, Seoul, Korea, November 2015.

[171] G. Elber and E. Cohen, "Toolpath generation for freeform surface models," *Computer-Aided Design*, vol. 26, no. 6, pp. 490–496, 1994.

[172] D. M. Bodily, T. F. Allen, and M. D. Killpack, "Motion planning for mobile robots using inverse kinematics branching," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 5043–5050.

[173] A. Kabir, A. Kanyuck, R. K. Malhan, A. V. Shembekar, S. Thakar, B. C. Shah, and S. K. Gupta, "Generation of synchronized configuration space trajectories of multi-robot systems," in *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019.

[174] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010.

[175] J. Pan, S. Chitta, and D. Manocha, "FCL: A General Purpose Library for Collision and Proximity Queries," in *International Conference on Robotics and Automation*, 2012.

[176] D. W. Sandberg and R. B. Wodtli, "Collision detection using sphere approximations," in *Robotics and Factories of the Future87*. Springer, 1988, pp. 456–460.

[177] J. Nocedal and S. J. Wright, "Numerical optimization 2nd edition," 2006.

[178] M. H. Stone, "The generalized weierstrass approximation theorem," *Mathematics Magazine*, vol. 21, no. 5, pp. 237–254, 1948.

[179] A. M. Kabir, J. D. Langsfeld, K. N. Kaipa, and S. K. Gupta, "Identifying optimal trajectory parameters in robotic finishing operations using minimum number of physical experiments," *Integrated Computer-Aided Engineering, special issue on "Enabling Robot Autonomy"*, vol. 25, no. 2, pp. 111–135, March 2018.

[180] A. M. Kabir, J. D. Langsfeld, C. Zhuang, K. N. Kaipa, and S. K. Gupta, "A systematic approach for minimizing physical experiments to identify optimal trajectory parameters for robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, May 2017, pp. 351–357.

[181] A. M. Kabir, K. N. Kaipa, J. Marvel, and S. K. Gupta, "Automated planning for robotic cleaning using multiple setups and oscillatory tool motions," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 3, pp. 1364–1377, July 2017.

[182] A. M. Kabir, J. D. Langsfeld, S. Shriyam, V. S. Rachakonda, C. Zhuang, K. N. Kaipa, J. Marvel, and S. K. Gupta, "Planning algorithms for multi-setup multi-pass robotic cleaning with oscillatory moving tools," in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, Fort Worth, Texas, USA, Aug 2016, pp. 751–757.

[183] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[184] G. Hermann, "Algorithms for real-time tool path generation," *Geometric Modeling for CAD Applications*, vol. PP, pp. 295–305, May 1988.

[185] Y. D. Chen, J. Ni, and S. M. Wu, "Real-time cnc tool path generation for machining iges surfaces," *Journal of Manufacturing Science and Engineering*, vol. 115, no. 4, pp. 480–486, 1993.

[186] R. K. Malhan, A. M. Kabir, B. C. Shah, and S. K. Gupta, "Identifying feasible workpiece placement with respect to redundant manipulator for complex manufacturing tasks," in *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019.

[187] M. Rickert, A. Sieverling, and O. Brock, "Balancing exploration and exploitation in sampling-based motion planning," *IEEE Transactions on Robotics*, vol. 30, no. 6, pp. 1305–1317, Dec 2014.

[188] A. M. Kabir, J. D. Langsfeld, C. Zhuang, K. N. Kaipa, and S. K. Gupta, "Automated learning of operation parameters for robotic cleaning by mechanical scrubbing," in *ASMEs 11th Manufacturing Science and Engineering Conference*, no. 49903, Blacksburg, Virginia, USA, June 2016, p. V002T04A001. [Online]. Available: http://dx.doi.org/10.1115/MSEC2016-8660

[189] D. C. Montgomery, *Design and Analysis of Experiments*. John Wiley & Sons, Inc., 2013.

[190] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems," *Evolutionary computation*, vol. 4.1, 1996.

[191] D. Karaboga and B. Akay, "A modified artificial bee colony (abc) algorithm for constrained optimization problems," *Applied soft computing*, vol. 11, no. 3, pp. 3021–3031, 2011.

[192] C. A. Coello and E. M. Montes, "Constraint-handling in genetic algorithms through the use of dominance-based tournament selection," *Advanced Engineering Informatics*, vol. 16, no. 3, pp. 193–203, 2002.

[193] E. Dvorkin, M. Goldschmit, and M. Storti, "Constrained optimization problems in mechanical engineering design using a real-coded steady-state genetic algorithm," *Mecnica Computacional*, vol. XXIX, pp. 9287–9303, 2010.

[194] C. A. Floudas and P. M. Pardalos, *A collection of test problems for constrained global optimization algorithms*. Springer Science & Business Media, 1990, vol. 455.

[195] S. N. Kramer, "An augmented lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design," *Journal of mechanical design*, vol. 116, p. 405, 1994.

[196] K. Deb and R. Datta, "A fast and accurate solution of constrained optimization problems using a hybrid bi-objective and penalty function approach," in *IEEE Congress on Evolutionary Computation*, July 2010, pp. 1–8.

[197] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[198] J. A. Hartigan, "Clustering algorithms," 1975.

[199] "Robotic finishing of interior regions of geometrically complex parts (video)," *https://www.youtube.com/watch?v=HJGdn1Wrg3s*.

Synthetic test problems used in chapter 7:
The description of the synthetic test problems used in chapter 5 are given in this section. In our experiments we considered $f(x)$ to be known and $g_i(x)$ to be black-box.

TP1

$$min\ f(x) = (x_1 + x_2)/500.0$$
$$s.t.\ x \in \mathbb{R}^2,\ 0.0 \leq x_i \leq 500.0, i = 1, 2$$
$$0.70 \leq g(x) \leq 1.0$$
$$g(x) = (s(x) + x_1 + 2x_2 + 500.0)/5000.0$$
$$s(x) = 418.9829 \times 5 - \sum_{i=1}^{2} x_i \times \sin(\sqrt{|x_i|})$$

TP2

$$min\ f(x) = -\frac{\sin^3(2\pi x_1) sin(2\pi x_2)}{x_1^3(x_1+x_2)}$$
$$s.t.\ x \in \mathbb{R}^2,\ 0.0 \leq x_i \leq 10.0, i = 1, 2$$
$$g_i(x) \leq 0.0$$
$$g_1(x) = x_1^2 - x_2 + 1.0$$
$$g_2(x) = 1.0 - x_1 + (x_2 - 4)^2$$

TP3

$$min\ f(x) = x_1^2 + (x_2 - 1)^2$$
$$s.t.\ x \in \mathbb{R}^2,\ -1.0 \leq x_i \leq 1.0, i = 1, 2$$
$$g_1(x) \leq 0.0$$
$$g_1(x) = (x_2 - x_1^2)$$

TP4

$$min\ f(x) = (100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2)/100.0$$
$$s.t.\ x \in \mathbb{R}^3,\ 0.0 \leq x_i \leq 10.0, i = 1, 2, 3$$
$$g_1(x) \leq 0.0625$$
$$g_1(x) = (x_1 - 2)^2 + (x_2 - 2)^2 + (x_3 - 2)^2$$

TP5

$$min\ f(x) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3$$
$$s.t.\ x \in \mathbb{R}^4,\ 0.0 \leq x_1, x_2 \leq 1.0$$
$$0.0 \leq x_3 \leq 50.0, 0.0 \leq x_4 \leq 240.0$$
$$g_i(x) \leq 0, i = 1, 2, 3$$
$$g_1(x) = -x_1 + 0.0193x_3$$
$$g_2(x) = -x_2 + 0.00954x_3$$
$$g_3(x) = -3.1416x_3^2x_4 - \tfrac{4\pi}{3}x_3^3 + 1296000$$

TP6

$$min\ f(x) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0 + x_2)$$
$$s.t.\ x \in \mathbb{R}^4,\ 0.125 \leq x_1 \leq 10.0, 0.1 \leq x_2, x_3, x_4 \leq 10.0$$
$$g_i(x) \leq 0, i = 1, 2, \ldots, 7$$
$$g_1(x) = \tau(x) - \tau_{max}$$
$$g_2(x) = \sigma(x) - \sigma_{max}$$
$$g_3(x) = x_1 - x_4$$
$$g_4(x) = 0.10471x_1^2 + 0.04811x_3x_4(14.0 + x_2) - 5.0$$
$$g_5(x) = 0.125 - x_1$$
$$g_6(x) = \delta(x) - \delta_{max}$$
$$g_7(x) = P - P_c(x)$$
$$\delta_{max} = 0.25, \sigma_{max} = 30000, \tau_{max} = 13600$$
$$G = 12 \times 10^6, E = 30 \times 10^6, L = 14, P = 6000$$
$$P_c(x) = \frac{4.013E\sqrt{x_3^2x_4^6 36}}{L^2}(1.0 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}})$$
$$\delta(x) = \frac{4PL^3}{Ex_3^3x_4}$$
$$\sigma(x) = \frac{6PL}{x_4x_3^2}$$
$$J(x) = 2(\sqrt{2}x_1x_2(\frac{x_2^2}{12.0} + (\frac{x_1+x_3}{2})^2))$$
$$R(x) = \sqrt{\frac{x_2^2}{4} + (\frac{x_1+x_3}{2})^2}$$
$$\tau'(x) = \frac{P}{\sqrt{2}x_1x_2}$$
$$\tau''(x) = \frac{M(x)R(x)}{J(x)}$$
$$M(x) = P(L + \frac{x_2}{2})$$
$$\tau(x) = \sqrt{\tau'^2(x) + 2\tau'(x)\tau''(x)\frac{x_2}{2R} + \tau''^2(x)}$$

TP7

$$min\ f(x) = 3x_1 + 0.000001x_1^3 + 2x_2 + \frac{0.000002}{3.0(x_2+0.0000001)^3}$$
$$s.t.\ x \in \mathbb{R}^4,\ 0 \leq x_1, x_2 \leq 1200$$
$$-55 \leq x_3, x_4 \leq 55$$
$$g_i(x) \leq 0, i = 1, 2, 3, 4, 5$$
$$g_1(x) = -(x_4 - x_3 + 0.55)$$
$$g_2(x) = -(x_3 - x_4 + 0.55)$$
$$g_3(x) = 1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25)$$
$$+894.8 - x_1$$
$$g_4(x) = 1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25)$$
$$+894.8 - x_2$$
$$g_5(x) = 1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25)$$
$$+1294.8$$

TP8

$$min\ f(x) = e^{x_1x_2x_3*x_4x_5}$$
$$s.t.\ x \in \mathbb{R}^5,\ -2.3 \leq x_1, x_2 \leq 2.3$$
$$-3.2 \leq x_3, x_4, x_5 \leq 3.2$$
$$g_i(x) \leq 0, i = 1, 2, 3$$
$$g_1(x) = -(x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2)$$
$$g_2(x) = -(x_2x_3 - 5x_4x_5)$$
$$g_3(x) = -(x_1^3 + x_2^3 + 1)$$

TP9

$$min\ f(x) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$$
$$s.t.\ x \in \mathbb{R}^5,\ 78 \leq x_1 \leq 102, 33 \leq x_2 \leq 45$$
$$27 \leq x_3, x_4, x_5 \leq 45$$
$$g_i(x) \leq 0, i = 1, 2, \ldots, 6$$
$$g_1(x) = -(85.334407 + 0.0056858x_2x_5 + 0.00026x_1x_4 - 0.0022053x_3x_5)$$
$$g_2(x) = 85.334407 + 0.0056858x_2x_5 + 0.00026x_1x_4 - 0.0022053x_3x_5$$
$$g_3(x) = -(80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2)$$
$$g_4(x) = 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3x_3$$
$$g_5(x) = -(9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4)$$
$$g_6(x) = -(9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4)$$

TP10

$$min\ f(x) = -(25(x_1 - 2)^2 + (x_2 - 2)^2 + (x_3 - 1)^2 + (x_4 - 4)^2 + (x_5 - 1)^2 + (x_6 - 4)^2)$$
$$s.t.\ x \in \mathbb{R}^6,\ 0 \leq x_1 \leq 5$$
$$0 \leq x_2 \leq 4, 1 \leq x_3 \leq 5$$
$$0 \leq x_4 \leq 6, 1 \leq x_5 \leq 5, 0 \leq x_6 \leq 10$$
$$g_i(x) \leq 0, i = 1, 2, \ldots, 6$$
$$g_1(x) = -(x_1 + x_2 - 2)$$
$$g_2(x) = -(6 - x_1 - x_2)$$
$$g_3(x) = -(2 - x_1 + 3x_2)$$
$$g_4(x) = -((x_3 - 3)^2 + x_4 - 4)$$
$$g_5(x) = -(2 + x_1 - x_2)$$
$$g_6(x) = -((x_5 - 3)^2 + x_6 - 4)$$

TP11

$$min\ f(x) = -((x_1 - 10.0)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7)$$
$$s.t.\ x \in \mathbb{R}^7,\ -10 \leq x_i \leq 10, i = 1, 2, \ldots, 7$$
$$g_i(x) \leq 0, i = 1, 2, 3, 4$$
$$g_1(x) = -(127.0 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5)$$
$$g_2(x) = -(282.0 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5)$$
$$g_3(x) = -(196.0 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7)$$
$$g_4(x) = -(-4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7)$$

TP12

$$min\ f(x) = 0.7854x_1x_2^2(3.3333x_3^2 + 14.9334x_3 - 43.0934)$$
$$-1.508x_1(x_6^2 + x_7^2) + 7.477(x - 6^3 + x_7^3) + 0.7854(x_4x_6^2 + x_5x_7^2) + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7)$$
$$s.t.\ x \in \mathbb{R}^7,\ 2.6 \leq x_1 \leq 3.6, 0.7 \leq x_2 \leq 0.8$$
$$17 \leq x_3 \leq 28, 7.3 \leq x_4 \leq 8.3, 7.3 \leq x_5 \leq 8.3$$
$$2.9 \leq x_6 \leq 3.9, 5.0 \leq x_7 \leq 5.5$$
$$g_i(x) \leq 0, i = 1, 2, \ldots, 11$$
$$g_1(x) = -(x_1x_2^2x_3 - 27.0)$$
$$g_2(x) = -(x_1x_2^2x_3^2 - 397.5)$$
$$g_3(x) = -(x_2x_6^4x_3x_4^{-3} - 1.93)$$
$$g_4(x) = -(x_2x_7^4x_3x_5^{-3} - 1.93)$$
$$g_5(x) = \sqrt{(745x_4x_2^{-1}x_3^{-1})^2 + 16.9 \times 10^6} \times$$
$$(0.1x_6^3)^{-1} - 1100$$
$$g_6(x) = ((745x_5x_2^{-1}x_3^{-1})^2 + 157.510^6)^0.5(0.1x_7^3)^{-1} - 850$$
$$g_7(x) = x_2x_3 - 40.0$$
$$g_8(x) = -(x_1x_2^{-1} - 5)$$
$$g_9(x) = x_1x_2^{-1} - 12$$
$$g_10(x) = 1.56x_6 - x_4 + 1.9$$
$$g_11(x) = 1.1x_7 - x_5 + 1.9$$

TP13

$$min\ f(x) = x_1 + x_2 + x_3$$
$$s.t.\ x \in \mathbb{R}^8,\ 100 \leq x_1 \leq 10000$$
$$1000 \leq x_2, x_3 \leq 10000, 10 \leq x_4, x_5, x_6, x_7, x_8 \leq 1000$$
$$g_i(x) \leq 0, i = 1, 2, \ldots, 6$$
$$g_1(x) = -(1.0 - 0.0025(x_4 + x_6))$$
$$g_2(x) = -(1.0 - 0.0025(x_5 + x_7 - x_4))$$
$$g_3(x) = -(1.0 - 0.01(x_8 - x_5))$$
$$g_4(x) = -(x_1x_6 - 833.33252x_4 - 100x_1 + 83333.333)$$
$$g_5(x) = -(x_2x_7 - 1250x_5 - x_2x_4 + 1250x_4)$$
$$g_6(x) = -(x_3x_8 - 1250000 - x_3x_5 + 2500x_5)$$

TP14

$$min\ f(x) = x_1 + x_2 + x_3$$
$$s.t.\ x \in \mathbb{R}^8,\ 100 \leq x_1 \leq 10000$$
$$1000 \leq x_2, x_3 \leq 10000, 10 \leq x_4, x_5, x_6, x_7, x_8 \leq 1000$$
$$g_i(x) \leq 0, i = 1, 2, \ldots, 6$$
$$g_1(x) = 1.0 - 0.0025(x_4 + x_6)$$
$$g_2(x) = 1.0 - 0.0025(x_5 + x_7 - x_4)$$
$$g_3(x) = 1.0 - 0.01(x_8 - x_5)$$
$$g_4(x) = x_1 x_6 - 833.33252 x_4 - 100 x_1 + 83333.333$$
$$g_5(x) = x_2 x_7 - 1250 x_5 - x_2 x_4 + 1250 x_4$$
$$g_6(x) = x_3 x_8 - 1250000 - x_3 x_5 + 2500 x_5$$

TP15

$$min\ f(x) = \left| \frac{\sum_{i=1}^{10} \cos^4 x_i - 2\prod_{i=1}^{10} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{10} i x_i^2}} \right|$$
$$s.t.\ x \in \mathbb{R}^1 0,\ 0 \leq x_i \leq 10, i = 1, 2, \ldots, 10$$
$$g_i(x) \leq 0, i = 1, 2$$
$$g_1(x) = 0.75 - \prod_{i=1}^{10} x_i$$
$$g_2(x) = \sum_{i=1}^{10} x_i - 75$$

TP16

$$min\ f(x) = x_1^2 + x_2^2 + x_1 x_2 - 14 x_1 - 16 x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2$$
$$+5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45$$
$$s.t.\ x \in \mathbb{R}^1 0,\ -10 \leq x_i \leq 10, i = 1, 2, \ldots, 10$$
$$g_i(x) \leq 0, i = 1, 2, \ldots, 8$$
$$g_1(x) = -(105 - 4x_1 - 5x_2 + 3x_7 - 9x_8)$$
$$g_2(x) = -(-10x_1 + 8x_2 + 17x_7 - 2x_8)$$
$$g_3(x) = -(8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12)$$
$$g_4(x) = -(3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10})$$
$$g_5(x) = -(-3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120)$$
$$g_6(x) = -(-x_1^2 - 2(x_2 - 2)^2 + 2x_1 x_2 - 14x_5 + 6x_6)$$
$$g_7(x) = -(-5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40)$$
$$g_8(x) = -(-0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30)$$