Robot Trajectory Generation and Placement Under Motion Constraints

by

Rishi Malhan

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

August 2022

Acknowledgements

I am grateful to my doctoral advisor Dr. Satyandra K. Gupta for the opportunity to join his research group at the University of Southern California (USC). Dr. Gupta has inspired me to become a competent researcher, have the technical expertise, be an effective communicator, and be a person who brings positive change to society. I am fortunate to have worked at the state-of-the-art Center for Advanced Manufacturing facility at USC. Dr. Ariyan Kabir and Dr. Brual Shah were senior members of the group who played a crucial role in shaping my thought process, making me challenge myself, and providing unconditional friendship. I would like to thank Dr. Steve Nutt for enabling me to learn in the M.C. Gill Composites Center and for his advice on composites research. I am thankful to Dr. Yong Chen, Dr. Assad Oberai, and Dr. Quan Nguyen for their valuable insights, suggestions, and feedback on my work. I am grateful to have worked with late Dr. Timotei Centea. He passed away in March 2019 but his teachings will continue to influence and inspire my life. Next, I am fortunate to have collaborated and learned from my colleagues and friends Shantanu Thakar and Prahar Bhatt. I am thankful to my amazing peers and lab mates Aniruddha Shembekar, Alec Kanyuck, Pradeep Rajendran, Neel Dhanaraj, Omey Manyar, Jason Gregory, Sarah Al-Hussaini, Yeo Jung Yoon, Yash Shahapurkar, and Rex Jomy Joseph. Scott Zanio from Hexagon AB always facilitated state-of-the-art equipment used for my experimental results. The funding for my work was granted by National Science Foundation and USC Viterbi Fellowship support. I am grateful for unconditional support from my parents Kamal Malhan and Anjali Malhan, brother Harsh Malhan, cousins Mohit Malhan and Amit Malhan, sister-in-law Isha Malhan, and rest of my family.

Table of Contents

List Of Tables

List Of Figures

Abstract

Robots such as manipulator arms or mobile manipulators are being widely used for manufacturing, medical, research, logistics, and other fields. Point to point and path constrained are two broad categories of robot motions. Point to point demands a manipulator to move from one joint configuration to another avoiding obstacles along the way. Path constrained motion is more restrictive where robots are expected to perform complex tasks like tracing a tool attached to the end-effector along assigned paths in its workspace. Additional motion constraints like velocity, force, collision avoidance, singularity avoidance, and continuity are also imposed on the tool's motion. Manually programming the robots is a common practice in mass production lines. Even though tedious and involves several trials and errors, it is economically viable for high volumes. The future of robotics caters toward high-mix low volume production as the industry moves towards customization and intelligent systems. Manual programming becomes unrealistic and uneconomical. Automated planning algorithms will therefore be needed to make the robotic deployment process faster and more efficient.

Robot placement is challenging due to a complex interaction of constraints and their variation within its range. Robot motion generation under constraints becomes difficult due to the computational complexity involved in constraint evaluations and the presence of several infeasible solutions acting as local minima during the search for optimal or near-optimal solutions. The robot motion generation and placement problems are also intertwined. Motions can only be optimized if placements are suitable. Finding optimal placements is dependent on the evaluation of the quality of robot motion for a candidate placement. Computational advances need to be

made in both areas to make the deployment of robots for a task more efficient and streamlined. In this dissertation, five fundamental advancements in the fields of graph-based planning, tree-based planning, non-linear derivative-based optimization, and physics-informed motion planning are made to address the two problems.

Conventional path constrained solutions constrain all the degrees of freedom (DOF) of the tool while executing a path-constrained task. Fully constrained motions reduce solution success rate and quality as the robot has to exactly meet the requirements which make it harder to find feasible solutions. Semi-constrained paths relax constraints on the tool without disrupting the task output. The semi-constrained approach is more practical in modern applications. For instance, the robot can freely rotate about the spinning axis of a sanding disk without affecting the process. However, the graph-based planning for optimal solutions is challenging due to the immense search space to be explored and the existence of several low and average quality solutions that increase computation time. This work develops an incremental construction of a reduced graph strategy to solve the problem as opposed to the construction of an immense exhaustive graph. Heuristics that transfer information from Cartesian space to robot Configuration space are developed for efficiently adding nodes in the graph. A biasing scheme has been developed that characterizes the quality of solutions in the search space with initial explorations and exploits high-quality neighborhoods to speed up convergence. The algorithms enhance graph-based planning and make it easier for the user to deploy semi-constrained paths for low-execution time and smoother robot motions.

There are applications where the geometry of the path itself can be relaxed to find solutions or to improve the process. The path and robot motion is simultaneously generated using motion constraints imposed by the physics-informed model of the interacting objects in the workspace. This dissertation provides examples of planning under physics-informed motion constraints using 1) deformable viscoelastic materials and 2) selective sensor workspace. The multi-robot manipulation of deformable materials imposes challenges such as unacceptable defects caused due to

a wrong robot movement imparting high energy to the system. Physics-informed planners are needed that accurately simulate material state while planning and correct the motions online. This work develops a multi-robot manipulation planner that uses ordering heuristics based on physics simulators to quickly explore the search space and find optimal solutions. The paths that robots take are found alongside robot trajectories such that manipulation constraints defined by material physics are satisfied. The selective sensor workspace constraint is based on the planning accountability of using a selected region of a sensor's full operational range to increase data collection accuracy. Using a sensor like an RGB-D camera for constructing a pointcloud of a large part is economical but can lead to poor accuracy if fixed paths are followed. The algorithms presented in this work find path-constrained robot motions under a sophisticated camera performance constraint that allows selective admission of points during construction. The motions are found such that camera properties are accounted for during planning.

Appropriate placements of the robot relative to the paths are necessary for generating feasible robot motions under the constraints. The arm will have a limited reach for a fixed base location. Mobile manipulators have an infinite arm reach in free space, however, the number of base repositioning should be minimized to lower execution time and position uncertainty of the system. The combinations between several candidate placements and evaluating a different robot motion that can be planned for each of them make base placement a very challenging problem. The algorithms presented in this work first encode the robot reachability in a quick-access data structure to generate candidate placements efficiently. Advances in the field of non-linear sequential optimization to evaluate a candidate placement computationally faster. The constraints are formulated as smooth non-linear functions and successively applied to reduce computation time. Advances are made in developing branch guiding and pruning heuristics for depth-first branch and bound search algorithm to then find the optimal sequence from the previously generated set of candidate placements. The anytime method quickly finds the base sequence first and then keeps on minimizing the number of placements and execution cost.

Chapter 1

Introduction

1.1 Motivation

Industrial manipulator arms or mobile manipulators are used in a wide variety of tasks to improve throughput and consistency. Point to point and path constrained are two broad categories of motions executed by the robots (see Figure 1.1). Point-to-point motions demand the manipulator to move from one joint configuration to the other avoiding obstacles and meeting other joint constraints. The robot is relatively free to choose the configurations between the start and end positions. Path constrained motions are more restrictive where the robot must select joint configurations that enable the tool to trace a path in the workspace and also meet other process constraints like velocity, force, continuity, collision, and singularity. This work deals with path-constrained motions. Manually finding robot placements and generating the path-constrained motions involve several trials and errors. The process is cumbersome, time-intensive, and uneconomical when high-mix low volume production systems are deployed. Smart manufacturing cells of the future will cater to more customization and high-mix low volume production. Automated planning of the robots to meet the motion constraints and finding suitable placements is needed.

Path constrained robot trajectories are largely based on the type of motion constraints applied. The manipulator can be carrying a tool on its end-effector that is responsible for following

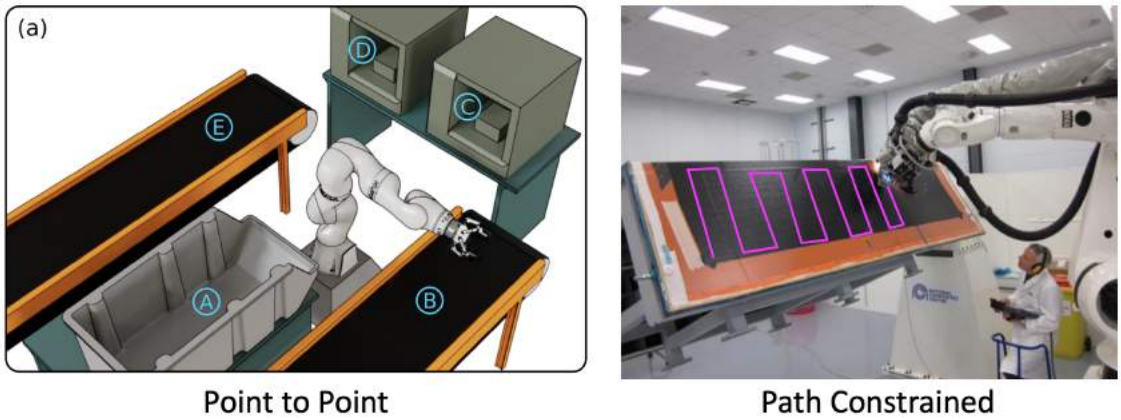

Figure 1.1: (left) Point to point motion demands the robot to move between any pair of labeled locations in space (courtesy: Dr. Pradeep Rajendran). (Right) Path constrained motion demands the robot to trace a tool along a path assigned in the workspace.

a path on a part. The interaction could be contact or non-contact-based. Figure 1.2(Top,Left) shows the constraint. Some motions may not define a fixed path but the robot has to choose a path such that process is feasible or output quality is improved. Consider the example in Figure 1.2(Top,Right) where the robot has to carry a sensor around the part to obtain an augmented pointcloud representation. Fixed paths lead to poor pointcloud accuracy. Selective use of the operational range of the camera leads to a higher pointcloud accuracy, however, robot path and trajectory are needed to be simultaneously generated using the sensor model constraints. Figure 1.2(Bottom,Right) shows a multi-robot system where fixed defined paths and physics-informed constraints that find paths and trajectories simultaneously are required together. The conforming robot is executing defined paths and two manipulating robots are avoiding undesired defects in the process by carrying the deformable material around using its physical characteristics during automated planning. Figure 1.2(Bottom,Left) shows an example where the manipulator is executing trajectories under motion constraints while a mobile base is carrying the manipulator around. Placement constraints enable finding the appropriate sequence of base locations that enable the execution of the trajectories.

Figure 1.2: Examples of different motion constraints. (Top,Left) Paths are defined and constraints are applied between the path and the robot. (Top,Right) Paths are not defined but found simultaneously with robot trajectories by enforcing the physical characteristics of the sensor (source: Hexagon). (Bottom,Right) Fixed paths and physics-informed constraints are all used in a multi-robot cell that is doing composite sheet layup. (Bottom,Left) A sequence of mobile base locations is also required as a placement constraint along with corresponding robot trajectories under motion constraints (source: KUKA).

This work considers the motion of a tool(s) attached to the robot end-effector over a set of discrete waypoints representing a set of toolpaths on the workpiece under motion constraints. Placement of the robot with respect to the workpiece must encapsulate the paths within the manipulator workspace. The motion constraints define the poses that the tool can take along the path, velocity and force to be sustained, collision avoidance, robot singularity, motion continuity, and other parameters. The workspace of a serial manipulator is defined as the set of positions and orientations (poses) that can be reached by the end-effector or a tool attached to the last link in the kinematic chain. A robot's motion or trajectory is a sequence of joint positions and its higher derivatives that allow the robot to perform the desired motion. Automated solutions find the robot trajectories and placements. Algorithmic advances need to be made to make automated planning fast and computationally efficient. For instance, robot placement planners make frequent

calls to trajectory planners to evaluate the quality of a placement. Improvements in the trajectory planner will have a direct impact on the performance of the placement planner. Finding good quality placements will make trajectory generation faster and improve success rate as fewer cycles are wasted in search of good solutions. The intertwined nature of the problems makes them very challenging to solve and interesting to address for realizing smart robotic assistants.

The introduction of motion constraints in trajectory planners decreases the computation efficiency due to the evaluation of constraint violations. An efficient algorithm requires heuristics and techniques to make trajectory generation faster and more pragmatic for offline and online use. The robot or mobile base needs a single or a sequence of placements that will be traversed for maximizing the reachability and performance of the manipulator over the task. The search space is vast with multiple placements needing evaluation before finding high-quality solutions. Multiple manipulators may also collaborate to perform a task that further increase complexity. The user will position the part and robot in the world after a suitable placement is determined. There will be localization inaccuracy due to manufacturing and positioning errors. The robot must then perceive the environment using sensors and automatically determine where the part is truly located to compensate for inaccuracies before execution. Sensors collect a mix of poor and high-quality data within their operational range. High-fidelity localization should characterize the sensor performance and admit high-quality data. Robot motion planning is challenging under such performance constraints since improving the quality of data will lead to slower execution and vice versa. The algorithms will need to account for the trade-off and find plans within minimum computation time.

1.2 Objectives and Scope

Our work addresses the challenges of 1) Path constrained trajectory generation under motion constraints. 2) Robot base placement. 3) Fast and accurate part localization with respect to the robot. We are interested in making fundamental advances in the following five areas.

1. Develop robot trajectory planning algorithms that will use semi-constrained toolpaths where the path is predefined and position is fixed. Semi-constrained toolpaths allow the user more flexibility in choosing the orientations within specific tolerances. Tolerances over tool position and orientation should be easily incorporated to improve the success rate and quality of solutions. Advances in graph-based planning will be made.

2. Develop methods that will use low-cost depth cameras mounted on commonly available industrial robots to automatically create accurate pointclouds for large parts. Fixed paths can reduce output accuracy so the relaxation of paths and freedom to choose a position as well as orientation can improve the process. The algorithm should carefully use regions of the sensor operational range to impose physics-informed constraints and find the camera paths and robot trajectories. An online refinement planner will be needed to fill in zero or low-density point regions on the part. Advances will be made in coverage planning and pointcloud collection method in this work.

3. Develop a multi-robot trajectory planner that will synchronize a team of robots manipulating deformable viscoelastic material while it is simultaneously draped over a mold. The multi-robot system will need predefined semi-constrained paths and determine physics-informed paths and robot trajectories together. Algorithms that will use physics-simulation-based graph-search techniques will be realized. Heuristics will be developed that will simulate the total energy of the sheet for faster convergence of the planner. Advances in the physics-informed motion planning field will be made.

4. Develop computationally efficient robot base placement algorithms that 1) will approximate the robot workspace and configuration space relationship using a capability map, 2) enforce smooth constraint violation functions that capture the motion constraints, and 3) will use a multi-stage optimization approach to use the approximate map for finding promising placements and a non-linear optimization algorithm for generating exact solutions. Advances will also be made in the sequential optimization-based trajectory planning field. The faster planners will improve the success rate and reduce computation time so quick trajectory evaluations can be done by the robot placement planner.

5. Develop algorithms that will find an optimal sequence of base placements for a manipulator mounted on a mobile base assigned a set of tasks in the workspace. The planner will have to deal with immense search space generated due to a combination of base placements and robot reachability solutions. The planner will have to quickly prune the search space where high-cost solutions lie and smartly explore low-cost solutions. The planner will have to be anytime that should generate low-cost initial solutions quickly and keep improving them over time. Advances will be made in tree-search-based planning field particularly depth-first branch and bound techniques.

The dissertation will demonstrate the algorithms for applications like robotic sanding, multi-robot composite sheet layup, robotic additive manufacturing, and robotic inspection.

1.3 Overview

The dissertation discusses five algorithmic advances from chapters 2 through 6. Chapter-2 makes algorithmic advances in the graph-based planning approach to the path-constrained trajectory planning problem. The method takes more general-purpose constraints into account and provides flexibility to the user in terms of giving high-level constraint bounds to the robot. Improved computation performance is obtained by successively constructing a reduced graph instead of

a full-sized graph. Heuristics will be developed in the work leading to faster convergence to near-optimal solutions. Chapter-2 mainly addresses rigid body constraints. Chapter-3 develops foundations for accurate part localization using an RGB-D sensor. Careful usage of sensor operational range leads to high-fidelity construction of a large complex part that can be used for accurate localization of the part with respect to the robot. Chapter-4 generates robot motions for the multi-robot cell where a deformable viscoelastic material is to be simultaneously manipulated and draped over a mold. A graph-based planner that will use drape simulations to characterize the physical behavior of the material and evaluate the energy of the sheet state will be developed. Heuristics will allow faster convergence and lead to high-quality paths. Chapter-5 uses path constrained trajectory generation based on solving a successive optimization problem for a faster computation of robot motions. The fast trajectory generation algorithm is used to find suitable single robot placements for a given task. Once a suitable placement is found, the more sophisticated graph-based planner from chapter-2 can be used to find near-optimal robot motions. The chapter makes algorithmic advances in the robot placement field. Chapter-6 uses a manipulator mounted on a mobile base to find a sequence of base placements for tasks such as inspection of large complex parts. The algorithm makes advances in tree-search planners by using novel branch guiding and branch pruning heuristics for the depth-first branch and bound approach.

Chapter 2

Algorithms for Generation of Configuration Space Trajectories over Semi-Constrained Cartesian Paths for Robotic Manipulators

2.1 Introduction

Industrial manipulators are used in a wide variety of tasks to improve throughput and consistency [181, 179, 4, 95]. The tasks often require interactions between a tool and a workpiece on which the task is executed. Figure 5.1 illustrates four of the several robotic applications: (i) cutting, (ii) deburring, (iii) wire arc additive manufacturing [18], and (iv) composite sheet layup [121, 122, 128]. The tool may or may not be in direct contact with the workpiece, but the robot, tool, and workpiece system form a closed chain kinematic tree. The chain is characterized by motion constraints such as following the desired Cartesian path, maintaining velocity and force limits, and avoiding collisions and singularities. The corresponding planning problem of finding a robot trajectory that satisfies the motion constraints is defined as path-constrained trajectory planning.

Traditionally, manipulators or robots have been programmed manually by a human operator to execute motion commands. Recent advances in motion planning algorithms have furthered robots' capability to generate plans and execute tasks automatically. The tasks demand single

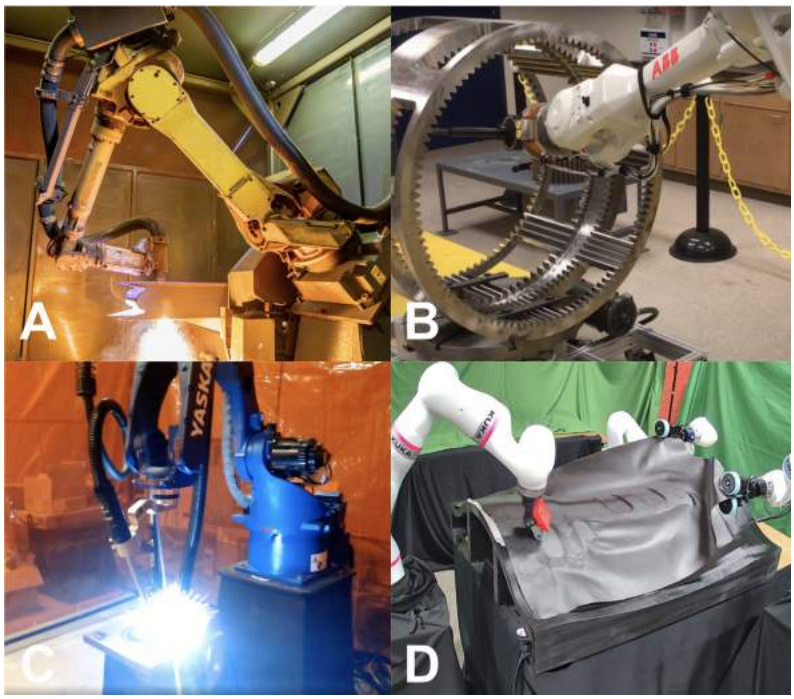

Figure 2.1: Example robotic applications that require a tool to track a Cartesian path. (A) Cutting (source: CBR Laser), (B) Deburring, (C) Wire arc additive manufacturing, and (D) Composite Sheet Layup

or multiple Cartesian paths to be followed by the robot end-effector. The path constrained planning algorithms are used by the robotic cells to generate end-effector constrained trajectories for automated task execution.

The end-effector aligns a tool center point (TCP) on the tool with the waypoints along the path. The TCP is defined by using a coordinate frame (refer Figure 2.2). Using a single TCP may lead to infeasible robot configurations due to collisions or robot exceeding its joint limits. Consider the case illustrated in Figure 2.2. A sanding operation is illustrated over a highlighted curved face of a part. The edge of the sanding disk is tilted by 10° and used to make contact with the face under force to press into the curved face. A single TCP will cause frequent reorientations of the tool while executing the path. A full 360° rotation of the tool will not be possible since wires and vacuum hose is attached to the back of the tool. The use of multiple TCPs is required to execute the motion without changing the orientation but transitioning from one TCP to the other. More

complex applications may require several more TCPs to be defined to make convenient transitions and facilitate robot reachability. Therefore, the use of multiple TCPs with the waypoints becomes necessary for most practical applications [15, 17].

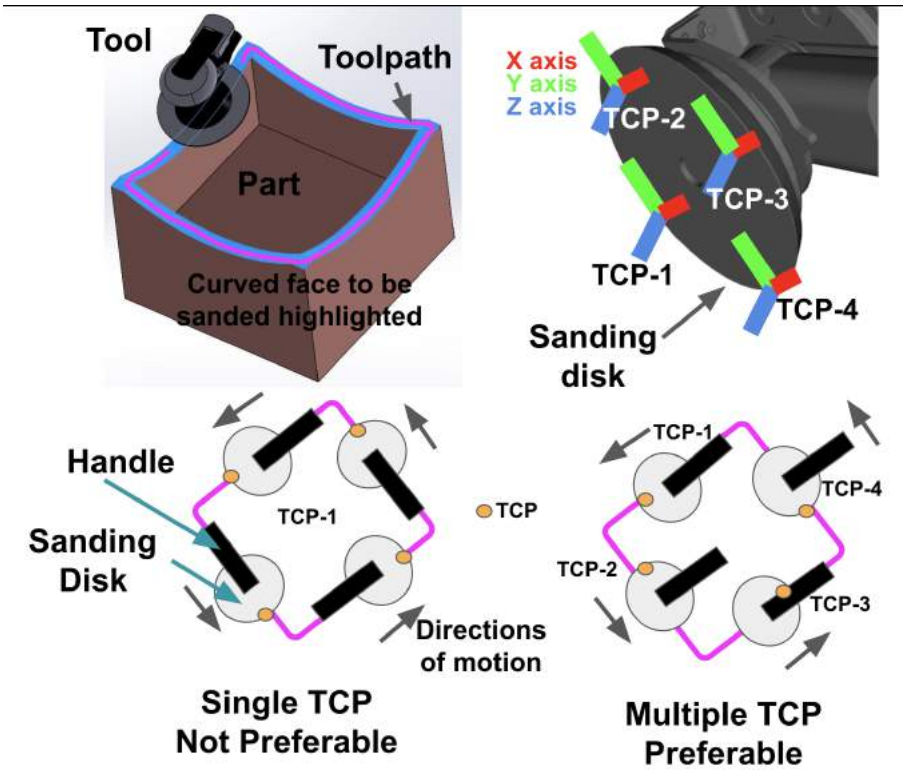

Figure 2.2: A sanding tool is used to sand the curved face top of the part shown. The edge of the sander is used to apply force and make contact with the face. Use of single TCP along the edge is not preferred (bottom left) since frequent reorientations of the tool are needed causing collisions, poor quality paths, or robot joint limits being exceeded. Using four discrete TCPs (bottom right) generates fewer tool reorientations and better paths.

A fully constrained path specifies that the tool must be aligned using all six degrees of freedom (DOF) i.e position and orientation with the waypoints along the path. Consider a long complex path that needs to be traced using a welding torch by the robot shown in Figure 2.3(top). All the waypoints along the path may not have consistent inverse kinematics (IK) solutions that belong to the same family and lead to discontinuities in motion (refer Figure 2.3(center)). Some waypoints may be completely unreachable. Most practical applications may require fewer than six DOF. A semi-constrained Cartesian path comprises semi-constrained waypoints that relax the motion

constraints and allow alignment using fewer than six DOF. For instance, Figure 2.3(bottom) illustrates introduction of tolerances about waypoints. The robot can reorient the tool within the allowable orientation margins. The waypoints may now have consistent IK solutions and previously unreachable waypoints may be reachable. Hence, a semi-constrained toolpath will increase the success of finding feasible and also leads to better quality trajectories. Applications can require the use of both TCPs and tolerances over a single toolpath. The existing methods are limited in incorporating TCPs, tolerances, or both in the planning framework. Section-5.2 describes the capabilities and limitations of the related work in this area in detail. We develop a trajectory planning algorithm for semi-constrained Cartesian paths that exploits the benefits of the graph and sampling-based approaches. We make the following contributions in the paper.

1) We construct a reduced graph method for path or trajectory planning for a robot that is required to execute Cartesian constrained paths on a workpiece. The graph is constructed by selectively adding new nodes and edges. The strategy uses a Cartesian space heuristic that exploits the kinematics of the robot.

2) We develop the Cartesian space heuristic that searches for high-quality solutions by using cues from the Cartesian space. We show that the quality of the robot trajectory depends on the starting robot IK solution. The graph will have multiple starting IKs from which trajectories originate. The trajectory costs differ based on which starting IK is selected.

3) We develop a sampling strategy that balances exploration and exploitation and selects starting IKs from neighborhoods that generate configuration space paths having a lower total length.

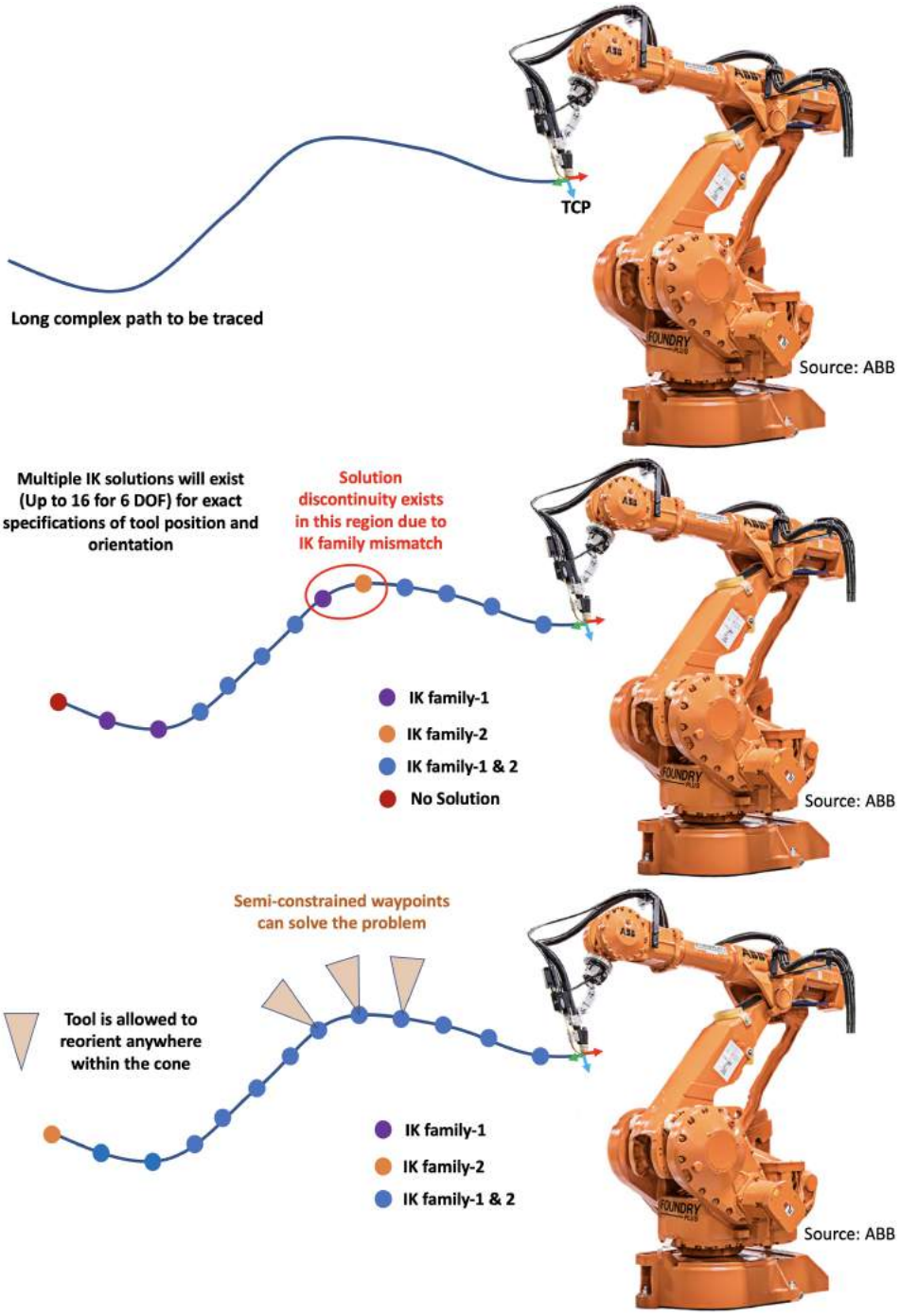

Figure 2.3: (Top) Long complex path to be traced by a welding torch tool center point frame. (Center) Waypoints may have inconsistent IK solutions or be unreachable by the robot. (Bottom) Introduction of tolerances lead to consistent IK solutions and improve reachability.

2.2 Related Work

2.2.1 Trajectory Generation For a Given Cartesian path

Optimization, graph construction and search, and sampling-based planning algorithms are widely used to solve path-constrained trajectory generation problem. This section discusses the state-of-the-art methods in gradient-based optimization, gradient-free optimization, graph construction and search, and sampling-based planners and described how our work is useful in overcoming the existing limitations.

Optimization based algorithms [132, 133] widely use B-splines to parameterize the joint variables and formulated the minimum-effort trajectory generation problem as a sequential quadratic program. Authors' previous work on path-constrained trajectory generation for an ensemble of robots [97, 96, 183] uses a parametric representation of the joint variables to minimize the time. The work done in [81, 80] used various distance metrics to measure the deviation of the desired path from the generated path after path constrained optimization. The authors in [174] show that numerical integration-based velocity planning can generate time-optimal trajectory with lower computational complexity. Jerk optimized trajectories were generated in [40]. These methods require an initial sequence of robot configurations as an initial guess to the optimization. The quality of the solutions and success rate will then depend on this initial guess. The methods are also sensitive to selection of an appropriate parametric representation. Additionally, the ability to incorporate multiple discrete TCPs and tolerances for different waypoints is not considered. It is also challenging to incorporate discrete TCPs during optimization.

The aforementioned approaches are based on gradient based optimization. Another class of methods are based on gradient-free optimization. The methods described in [175, 84] used a B-spline representation of the joints and an objective function incorporating time, smoothness, and energy. Both the approaches use a genetic algorithm to minimize cost functions with joint limits. Tarokh et al. [177] also used a genetic algorithm to solve the problem where the population

diversity and fitness were used to adjust the probabilities of the operators for the next generation. The approach in [137] uses a discretized version of the trajectory and a search is performed at each step for a new position of the end effector to reach the desired position. The genetic algorithm-based approaches can be computationally expensive and may fail to find a feasible solution. Genetic algorithms end up using a massive number of samples for complex problems as they do not use cues from why the solutions are failing in order to guide the generation of the future samples.

Search-based algorithms are more suitable for tackling discrete TCP and tolerance inputs given by the user. The methods find the Inverse Kinematics (IK) for manipulators at the discrete waypoints and TCPs on the given Cartesian path. A graph is constructed, taking all the feasible IK solutions into account for each waypoint. A graph search algorithm is 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 [166] is an open-source library developed by ROS-Industrial Consortium that uses a search-based method to generate paths. The work done in [41] uses a global sequence planning formulation on a directed graph to fabricate 3D models on a robotic printing system equipped with multi-axis motion. Search-based approaches are computationally expensive. Inefficiencies in graph construction are also present due to repeated robot configurations added as nodes from given TCP and waypoint combinations. However, efficient construction of the graph can provide the benefits during planning. We use an approach that is inspired by the graph based methods. Our work enables fast generation of good quality paths in this work using efficient graph construction techniques to solve the trajectory generation problem.

Sampling-based methods are also used for generating trajectories for redundant manipulators from the given Cartesian paths. Cefalo et al. [31] discuss a sampling-based approach that produces cyclic, collision-free paths in configuration space and guarantees continuous satisfaction of the task constraints. The method was later extended to generate closed collision-free paths in configuration

space [155]. The authors in [185] presented techniques to accelerate the sampling in a bi-directional tree based framework for a non-holonomic mobile manipulator. The work done in [14] introduced task space regions framework for encoding tolerances and integrated the framework with sampling-based algorithm. Sampling based methods are computationally expensive. A large number of samples in configuration space of the robot are required to be explored in order to generate paths. TCPs and tolerances at the waypoints increases the size of the searchable configuration space which increases the the exploration time during search. Additionally, it is difficult to include the kinematic and higher order constraints while connecting new nodes in the tree as opposed to graph based methods where timing information can be encoded in the graph edges. However, sampling based algorithms can achieve a faster convergence with a good exploration vs exploitation strategy. We use this benefit of sampling algorithms to design a heuristic that allows faster convergence of our method.

2.2.2 Trajectory Optimization Using Pre-defined Configuration Space Path

The robot trajectories generated from pre-defined configuration space paths using optimization based methods are discussed in this section for completeness of the work in the field. Pre-defined configuration space paths have been used to minimizing time, jerk, and effort under joint position, velocity, and torque constraints by researchers in [23, 55]. Optimal control algorithms [67] and discrete parameter optimization algorithms [37] have been used to find trajectories. The work done in [160] used a dynamic programming approach to minimize time by representing the curve in the robot's Cartesian space using arc-length parameterization. The authors in [161] used cubic splines to represent the motion of each joint and a minimax algorithm and interval analysis to solve the minimum jerk trajectory generation problem. Chettibi et al. [35] used a cubic spline representation and solved the problem using sequential quadratic programming. Gasparetto

et al. [68] used B-splines [43] with an objective function consisting of jerk and time in their objective function. However, these approaches can only generate trajectories from a given initial configuration space path. We need an approach to generate the initial configuration space path itself given a semi-constrained Cartesian toolpath.

2.3 Background and Terminology

The positions of all the joints of a robot are defined by a configuration vector $\vec{q}$. For an N degrees of freedom (DOF) robot, $\vec{q} \in \mathbb{R}^N$. In this work, N=6 as we use a six DOF robot. The space of all configurations $\vec{q}$ is the configuration space ($\mathscr{C}$-space). A continuous trajectory τ is defined as $\tau = \{\vec{q}(u) | \vec{q}(u) \in \mathscr{Q}_f\}, u \in [0, 1]$, where u is a parameter and Q_f is the set of feasible configurations in $\mathscr{C}$. The Cartesian space allows us to define all the positions and orientations of bodies by using frames O. Each frame is defined as a homogeneous transformation matrix. All the frames are represented in a reference coordinate system (robot base frame in our case). For instance, the transformation matrix ${}^bT_f$ gives the robot flange or end-effector frame O_f with respect to the robot base frame O_b. The robot joint velocity ($\dot{\vec{q}}$) and the Cartesian velocity of the flange $\dot{x}$ are related by the Jacobian J as $\dot{x} = J \times \dot{\vec{q}}$, where $J \in \mathbb{R}^{6\times N}, \dot{x} \in \mathbb{R}^6$ and $\dot{\vec{q}} \in \mathbb{R}^N$.

A toolpath is represented by $\mathscr{P}$. We define $\mathscr{P}$ as a sequence of n waypoints $\{p_1, p_2, \ldots, p_n\}$. Each waypoint is a frame defined with respect to O_b. For simplicity, we consider the expressions p_i in place of ${}^bp_i$. The tool traces $\mathscr{P}$ when a TCP selected on the tool's rigid body aligns with the waypoint frames along the path. A fully constrained tool-path requires the TCP to be exactly aligned with the waypoints. A semi-constrained tool-path $\mathscr{W}$ relaxes constraints on the pose. The relaxation is done by providing tolerances. Figure 2.4(left) illustrates the tolerance in position introduced at the nominal waypoint. The TCP is allowed to occupy any point within a sphere centered around the waypoint frame's origin. The orientation constraint is relaxed by allowing the TCP's unit direction vectors (X, Y, and Z axes) to change the angle with unit direction vectors of

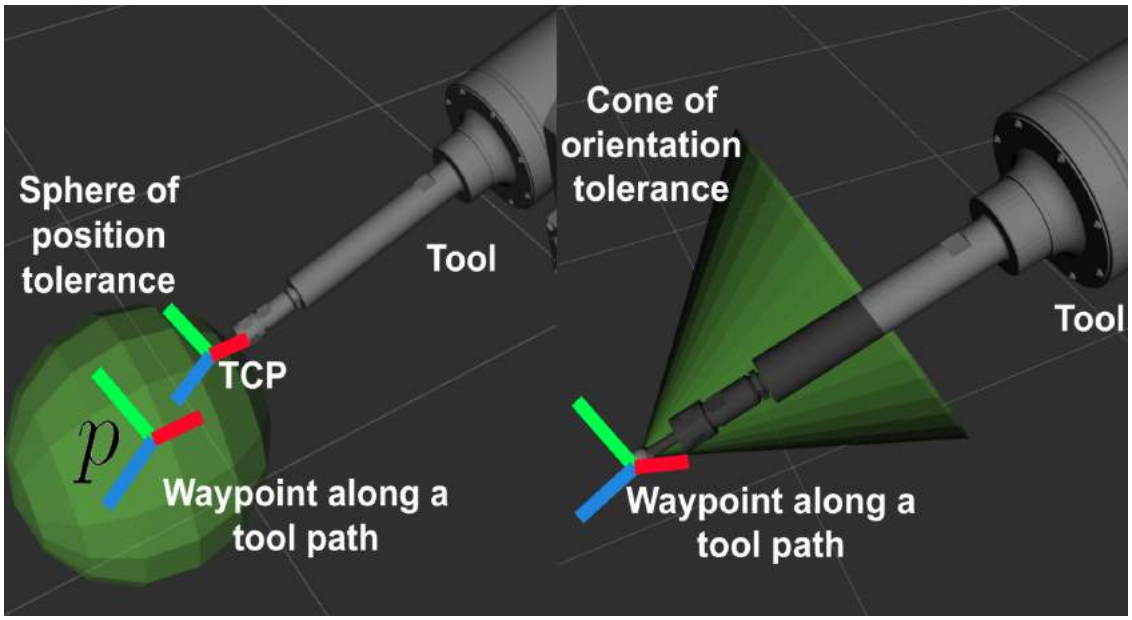

Figure 2.4: (Left) Tolerance in position of the origin of the waypoint frame creates a sphere of radius equal to the magnitude of tolerance distance. (Right) Tolerance about X and Y axes is shown. The tool can orient the Z-axis of the TCP with the Z-axis of the waypoint within a certain angle forming a 3D tolerance cone.

the waypoint frame. Figure 2.4(right) gives a sample illustration of the tolerances about both X and Y axes. The tolerances form a corresponding 3D cone and a circle permitting TCP alignment. We call such waypoints semi-constrained waypoints. A semi-constrained tool-path contains at least one semi-constrained waypoint.

We define a bound matrix B as a 6×2 matrix B to provide tolerances. B comprises of lower and upper bounds on $< x, y, z >$ (position) and $< \alpha, \beta, \gamma >$ (orientation) in Cartesian space. We obtain $\mathscr{B} = \{B_1, B_2, \ldots, B_n\}$ for all p_i from the user specified inputs. A semi-constrained waypoint is thus defined by a nominal frame p which is obtained from the toolpath $\mathscr{P}$ and the corresponding bound matrix B. We represent the semi-constrained toolpath as $\mathscr{W} = \{(p_1, B_1), (p_2, B_2), \ldots, (p_n, B_n)\}$, where (p_i, B_i) is a semi-constrained waypoint.

A transformation T_j can be generated by sampling $< x, y, z, \alpha, \beta, \gamma >$ from the bound matrix. The TCP alignment is equivalent to aligning the TCP with p_i' obtained by applying a transformation T_j to waypoint p_i, hence, $p_i' = p_i * T_j$. The bounds in the set $\mathscr{B}$ are discretized for making computations tractable. A set $\mathscr{T}_i$ for i^{th} semi-constrained waypoint contains all the discretized transformations T_j. Our approach uses uniform sampling for realizing this discretization. We can also incorporate general end-effector constraints on a robot configuration $\vec{q}$ given by $\mathrm{G}(\vec{q})$. For instance, if a tool is required to be within a particular orientation, appropriate B can be defined. If B is a zero matrix at a waypoint, the tool must be aligned with p_i.

Multiple TCPs can be used on a given tool while following $\mathscr{W}$. We define a set of discrete TCPs $\mathscr{E}$ from which a TCP e is sampled to be used over semi-constrained waypoint w. Each TCP e is a frame represented in the robot base frame. A TCP e_j and transformation T_j for the i^{th} waypoint will fully define the closed kinematic chain from the tool TCP to the robot base frame. A sequence of (e_j, T_j) pairs for every semi-constrained waypoint along the toolpath $\mathscr{W}$ will define a fully constrained Cartesian space path $\mathscr{X}$. Several such paths $(\mathscr{X})$ will exist in the Cartesian space passing through different combinations of e and T for every semi-constrained waypoint.

A valid sequence $\mathscr{X}$ is the one for which τ lies in feasible $\mathscr{C}$-space. Feasible $\mathscr{C}$-space consists of robot configurations that meet the following constraints.

1. *Joint Limit Constraint:* The robot configuration should lie between lower and upper limits of the configurations that the motor can provide. The constraint is expressed as $\underline{\vec{q}} \leq \vec{q} \leq \overline{\vec{q}}$, where $\underline{\vec{q}}$ and $\overline{\vec{q}}$ are the lower and upper bounds respectively.

2. *Collision Constraint:* The robot configuration should lie in the collision free space $\mathscr{C}_f$.

2.4 Problem Formulation

We are given the semi-constrained toolpath $\mathscr{W}$, robot model, and a set of TCPs $\mathscr{E}$ to be used. Our goal is to search the feasible $\mathscr{C}$-space for τ that will also minimize the overall trajectory cost.

We define the cost as total $\mathscr{C}$-space path length. Therefore, we formulate the semi-constrained trajectory generation problem for semi-constrained Cartesian path as follows:

$$\text{Given } \mathscr{W}, \mathscr{R}, \mathscr{E}$$

$$\text{Find } \tau^* \leftarrow \arg\min_{\tau}(\text{f}(\tau))$$

$$s.t\ \text{FK}(\tau) = \mathscr{X} \tag{2.1}$$

$$\text{H}(\vec{q})_{lb} \leq \text{H}(\vec{q}) \leq \text{H}(\vec{q})_{ub} \tag{2.2}$$

The objective function f we want to minimize is total length of path τ in $\mathscr{C}$-space. FK$(\cdot)$ is the forward kinematics function which checks if τ follows the path defined by $\mathscr{X}$. The constraint expressed by equation 2.1 requires the tool to trace the Cartesian path $\mathscr{X}$. Equation 2.2 requires the joint configuration to be within prescribed lower (H$(\vec{q})_{lb}$) and upper (H$(\vec{q})_{ub}$) bounds.

2.5 Approach

2.5.1 Overview of Approach

Every frame obtained from discrete transformation $T_j \in \mathscr{T}_i$ can be used to align with a TCP $e_j \in \mathscr{E}_i$ for i^{th} semi-constrained waypoint. We can think of the alignment process as the robot flange positioning itself with a flange frame (FF) denoted by s for a given pair (e, T). A flange frame s can be identical or very close to one another for different combinations of TCP e and transform T. We use an equivalent version of the sets $\mathscr{E}_i$ and $\mathscr{T}_i$ by merging them into one set called the FF set $\mathscr{S}$. The representation allows us to represent tolerances and TCPs efficiently with a minimum number of FF. The number of FF sets will equal the number of waypoints in the semi-constrained toolpath. The sequence of FF sets corresponding to the sequence of semi-constrained waypoints will then be used to build the directed graph $G = (V, E)$ with vertices V

and edges E. Each semi-constrained waypoint will correspond to a level in G. The framework for generating flange frames is presented in Section 2.5.2.

We take a graph-based approach where each FF is mapped to $\mathscr{C}$-space using inverse kinematics (IK). Each IK solution is checked for constraint violations and the valid IKs become a part of a graph. The valid IKs are then connected with the existing configurations of the graph if they satisfy graph edge constraints. A path from root to leaf of the graph will give a feasible τ and τ^* will have the shortest path length. The graph representation is discussed in Section 2.5.3.

The robot trajectory generation algorithm is discussed in Section 2.5.4. The algorithm incrementally builds the graph instead of building it completely before performing a graph search. We develop a Cartesian space heuristic that uses cues from the robot Cartesian space and intelligently selects the FF during incremental graph building. Section 2.5.6 provides details on the Cartesian space heuristic. The strategy allows us to produce low-cost initial paths and eventually near-optimal solutions are generated. We later show in the paper that the trajectory cost also depends on the FF that is selected at the first semi-constrained waypoint or also called the source frame. Selecting the source frame efficiently can save computational resources. We develop a strategy in Section 2.5.7 to bias the source sampling. The path returned by our algorithm is then smoothened. We use a discrete smoother that inserts robot configurations in the vicinity of the current best trajectory τ. The smoother further reduces the path cost and the optimal path thus found by our algorithm is returned.

2.5.2 Representing Flange Frame Sets

We convert all combinations of (e, T) pairs that exist between sets $\mathscr{E}$ and $\mathscr{T}$ for each semi-constrained waypoint into a FF sets ($\mathcal{S}_i$ for the i^{th} waypoint). Our framework to use the FF allows us to facilitate a reduction in the redundancy in Cartesian poses using the following two steps:

1. *Eliminate Redundancy:* Duplicate FF leads to unnecessary computations of robot IK. We remove this redundancy by eliminating duplicate $s \in \mathcal{S}_i, i \in [1, n]$ for all waypoints.

2. *Promote Diversity:* An additional constraint is optionally introduced in the algorithm to save computation costs by considering a FF in $\mathcal{S}$ and removing neighbors that are closer than a threshold distance. The distance metric used is euclidean and FF are represented by a six-dimensional vector comprised of position and quaternion. The process promotes diversity in FF and computation costs are reduced.

The pre-computation of unique and diverse flange frame sets $\mathcal{S}_i$ for every waypoint i is inexpensive. Moreover, the algorithm's spatial and time complexity is not impacted when an excessive number of TCPs and relaxed bounds are given as input since redundancies get filtered out.

2.5.3 Representing Feasible State-Space as Graphs

The vertices V are obtained by mapping each $s \in \mathcal{S}_i$ to the robot configuration space. Figure 2.5 illustrates a sequence of FF sets, each of size three. The figure shows the set of IK solutions for FF s_3 from set $\mathcal{S}_1$ indicated by the dashed magenta box. Each IK solution computed for a $s \in \mathcal{S}_i$ is checked for the joint limit and collision constraints. We use a hierarchical collision checking model where the tool is checked for collisions first. Collision amongst other environment objects is checked if the tool is collision-free. Other constraints can also be included during the configuration validity process. The valid configurations that meet the imposed constraints are then used to create a vertex (node) in G at the i^{th} level corresponding to the set $\mathcal{S}_i$. Invalid configurations are not included in the graph.

Let q_i and q_{i+1} be two successive configurations corresponding to levels i and $i+1$. We compute the edge cost as a second-order norm given by equation 2.3. An infinity norm constraint given by the equation 2.4 is applied to every edge in the graph between the parent and child nodes (configurations). δ is the maximum radian angle change allowed between two configurations. This

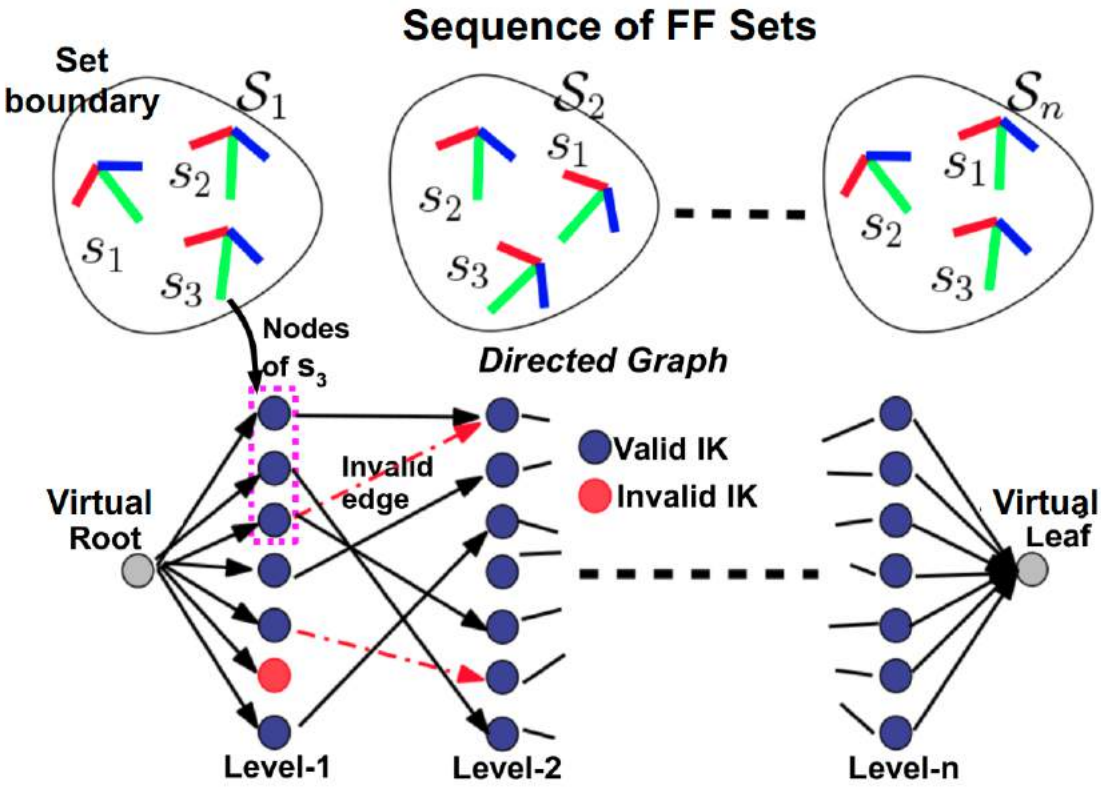

Figure 2.5: A sequence of FF sets with set boundaries is illustrated. The FF are used to solve IK and all the valid solutions are added as nodes to the directed graph G. Invalid nodes are discarded. Dashed magenta box shows the valid configurations for FF s_3.
The edge connections are exhaustively made between the all consecutive node pairs by evaluating the second-order norm. Invalid edges (dashed red lines) that violate kinematic, dynamic, continuity, or other constraints are discarded. A virtual root and leaf node is inserted to avoid multiple Djikstra runs.

helps keep the IK solutions for the parent and the child within the same IK family and the edge is invalid if the constraint is not met. We assume that the robot is collision-free while transitioning from q_i to q_{i+1}. However, the collision avoidance along the edge can be checked using the swept volume of robot links [178]. Optimal path τ^* is the one which minimizes equation 2.5.

$$\text{Edge Cost} = ||q_{i+1} - q_i||_2 \tag{2.3}$$

$$||q_{i+1} - q_i||_\infty \leq \delta \tag{2.4}$$

$$\text{Path Cost} = \sum_{i=1}^{n-1} ||q_{i+1} - q_i||_2 \tag{2.5}$$

The number of nodes at level 0 or the first waypoint can be more than one. To avoid running a search from multiple root nodes, we insert a virtual root node before level 0. The edge cost from the virtual root node to any node at level 0 is zero. Similarly, a virtual leaf node is inserted after the last level n to avoid checking paths to multiple leaf nodes and selecting the lowest cost path. The edge cost from each leaf node to the virtual leaf node is also zero. The virtual nodes inserted at both ends of the graph are shown in Figure 2.5. The number of levels in G is equal to the number of waypoints n. The graph is directed because the configurations can only be connected in a sequence that corresponds to the task's sequence of waypoints. The shortest path from the virtual root node to the virtual leaf node is τ^*.

2.5.4 Generating Semi-Constrained Paths

The trajectory generation for semi-constrained Cartesian paths algorithm is initialized by first computing all FF sets (line 1 of algorithm 1).

We initialize a reduced graph $G_R = (V_R, E_R)$, such that $V_R \subset V$ and $E_R \subset E$ (line 2 of algorithm 1). The selective evaluation of a few FF in each iteration creates a sub-graph g. G_R

Algorithm 1 SemiConstrainedCartesianPlanner($\mathscr{W}, \mathscr{R}, \mathscr{E}$)

1: $FFrameSets \leftarrow$ genFFrames($\mathscr{W}, \mathscr{E}$)
2: $G_R.V = \{\}$ $G_R.E = \{\}$
3: **while** !stoppingcondition AND SRCavailable **do**
4: **for** *progtype* in {"fwd","bck"} **do**
5: $g.V = \Delta V = \{\},\ g.E\ \Delta E = \{\}$
6: **if** srcbias($\cdot$) **then**
7: biasSRC($FFrameSets, SRCmap$)
8: **else**
9: rndSRC($FFrameSets$)
10: **end if**
11: seqProg($FFrameSets, \mathscr{R}, g$,progtype)
12: updateSRCmap($SRCmap$, searchGraph(g))
13: $G \cup g$
14: **end for**
15: $\tau^* \leftarrow$ searchGraph(G)
16: **end while**
17: **while** !stoppingcondition AND FFramesAvailable **do**
18: rndSampling($FFrameSets, \mathscr{R}, G$)
19: **end if**
20: $\tau^* \leftarrow$ searchGraph(G)
21: $\tau^* \leftarrow$ smoothener($\tau^*, \mathscr{R}, FFrameSets$)
22: Return τ^*

grows by merging the sub-graph g to it. As FF keep getting evaluated, $|V_R| \rightarrow |V|$ and $|E_R| \rightarrow |E|$. So the reduced graph G_R will be identical to G after a finite number of iterations.

2.5.4.1 Sequential Progression

The $\mathscr{C}$-space path τ starts from a configuration q_1 corresponding to a FF $s_1 \in \mathcal{S}_1$ and ends at q_n corresponding to a FF $s_n \in \mathcal{S}_n$. The algorithm successively selects FF from s_1 to s_n and constructing a sub-graph g (line 11 of algorithm 1). A depth-first approach is used in our algorithm to generate the sequential progression. Figure 2.6 illustrates the depth-fist sub-graph g_w and the corresponding $\mathscr{C}$-space sub-graph g. The progression starts with pushing s_1 to a stack. The FF s_{21} and s_{22} from the set $\mathcal{S}_2$ are selected by the Cartesian space heuristic such that they are equidistant in position and orientation to s_1 and are pushed to the stack. We will discuss why the Cartesian space heuristic selects equidistant frames in Section 2.5.6. Then, the next FF is then popped out from the stack. The depth-first iterations continue until the stack is empty.

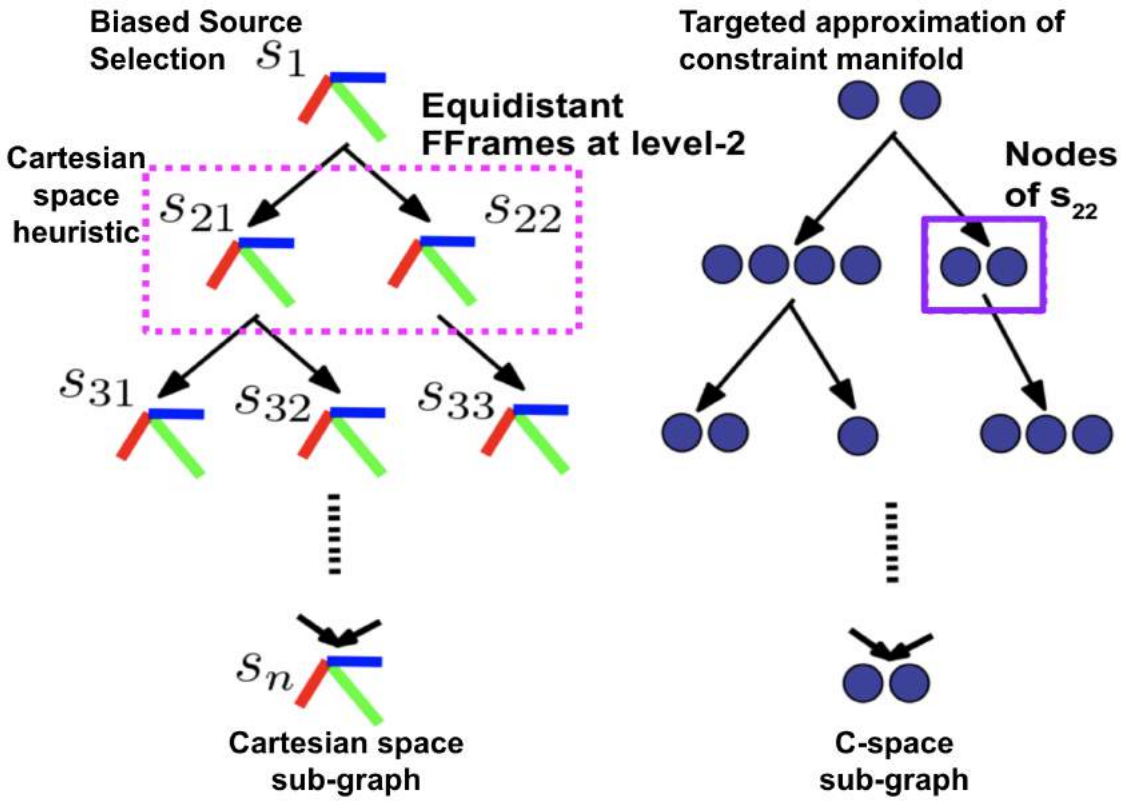

Figure 2.6: (Left) Cartesian space sub-graph g_w generated by FF from a forward sequential progression. (Right) $\mathscr{C}$-space sub-graph g generated by solving IK corresponding to g_w.

2.5.4.2 Bi-directional Sequential Progression

Forward progression picks a source from $\mathcal{S}_1$ and progresses to $\mathcal{S}_n$. To find better paths and for faster convergence, we also perform a backward progression. Backward progression picks a source $s_n \in \mathcal{S}_n$ and progresses towards $\mathcal{S}_1$. The nodes generated from both the progressions generate separate sub-graphs, but both are added to G_R in every iteration. Together the two progressions constitute one iteration of the algorithm. Lines 3-14 of the algorithm 1 show the steps for finding and integrating g into G_R for each progression.

The quality of $\mathscr{C}$-space path is governed by how the source FF is selected. Figure 2.7 illustrates two different paths by selecting different FF within bounds of the semi-constrained waypoint. Reorientation of the tool lead to a frequent repositioning of the robot end-effector causing high path cost in $\mathscr{C}$-space.

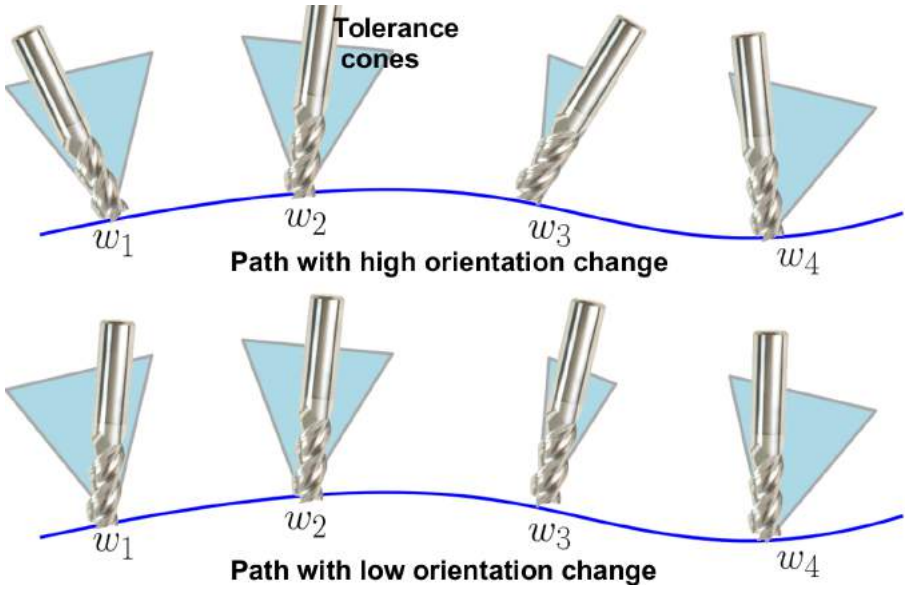

Figure 2.7: The tool tip is required to trace a Cartesian space curve. Semi-constrained waypoints allow reorienting the tool within the tolerance cones. Two paths are shown with different orientation changes are illustrated.

Our algorithm uses a source biasing routine to select the appropriate source from which low-cost paths can be generated during forward and backward progressions. Lines 6-10 of the algorithm 1 return the selected source based on the source biasing routine described in Section 2.5.7.

2.5.5 Generation of Anytime Solution

Running graph search is 10^2 to 10^4 order computationally less expensive as compared to collision checking or evaluation of kinematic or dynamic constraints that make up a major portion of computation costs involved. Due to the efficient nature of the graph search, it is performed after every algorithm iteration to find the current best path. Consequently, our algorithm becomes an anytime algorithm. As the iterations progress and G_R is updated, better paths are obtained. Line-15 of the algorithm 1 corresponds to the graph search step.

The stopping condition is satisfied either in the form of maximum iterations or a designated time limit. We then apply a smoothening routine to further reduce the path cost (line-21 of algorithm 1) and return the best path τ^*. If the stopping condition has not been reached but all the sources in $\mathcal{S}_1$ and $\mathcal{S}_n$ are exhausted, we keep on randomly sampling the FF (lines 17-19 of the algorithm 1) until the stopping condition is reached or all FF are exhausted. This step can ensure the completeness of our algorithm since we exhaust the entire search space asymptotically.

2.5.6 Cartesian space Heuristic

For small values of $\dot{x}$ and $\dot{\vec{q}}$, we can establish the relationship between Cartesian space pose and a robot configuration as $\Delta x = J \times \Delta \vec{q}$. We can visualize the relationship by mapping the magnitudes of the change in robot configuration ($\Delta \vec{q}$) for changes in the robot flange pose (Δx) in the Cartesian space. The graph in Figure 2.8 shows that the relationship is non-linear, and the spread gradually increases as Δx increases showing that the Jacobian approximation starts becoming invalid. We used 30000 random configurations in the robot $\mathscr{C}$-space and added randomly selected Δx from a uniform distribution to determine the corresponding $\Delta \vec{q}$. As the magnitude value of Δx increases

in the Cartesian space, the robot travel in $\mathscr{C}$-space also increases. It is important to note that the relationship will not hold in the case of a redundant robotic system (DOF > 6)

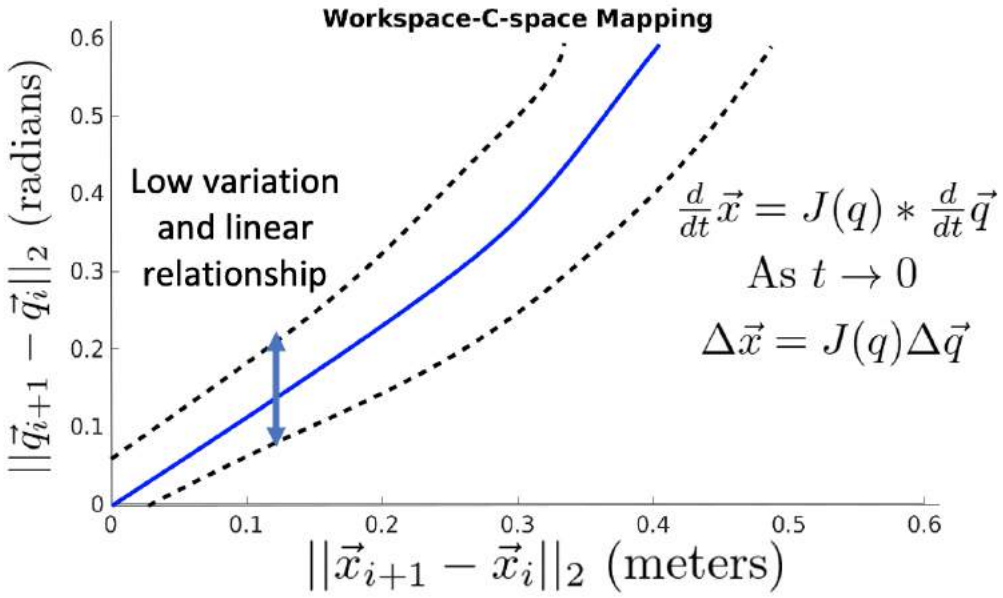

Figure 2.8: The relationship between magnitude of a change in robot configuration $\Delta\vec{q}$ and corresponding magnitude of change in end-effector pose Δx is shown. The blue curve represents the average value and black dotted curves represent the 95 % confidence bounds.

Figure 2.8 also shows that the relationship between Δx and $\Delta\vec{q}$ can be approximated as linear as long as Δx is bounded. The bound on Δx provides how far apart the FF need to be spaced before the linearity is violated. For most practical applications, the waypoints are spaced such that the value of Δx between consecutive waypoints is always less than the upper limit of violation of linearity. Using the linear relationship between Δx and $\Delta\vec{q}$, a minimization of $\sum \Delta x$ or distance traveled by the flange in the Cartesian space will lead to a reduction in path length of τ.

The Cartesian space heuristic selects l number of FF from the set $\mathscr{S}_{i+1} = \{s_{(i+1)1}, s_{(i+1)2}, \ldots, s_{(i+1)m}\}$ during sequential progression from FF s_i. Let d be the shortest distance given by $\min(\text{dist}(s_i, s_{(i+1)j})), j = \{1, 2, \ldots, m\}$. dist$(\cdot)$ computes the euclidean distance between FF represented as a vector of position and quaternion. The set $\mathscr{F}$ containing the l FF to be selected is found such that the distance

between any FF in $\mathscr{F}$ and s_i is equal to d. Alternatively, all the FF selected from the set $\mathcal{S}_{i+1}$ by the heuristic are the shortest euclidean distance from query FF s_i.

All invalid FF at a semi-constrained waypoint lead to discontinuity in the path and a solution does not exist. We detect the discontinuity and account for other edge cases in the following manner.

1. *Evaluated FF:* If a set of FF picked by the heuristic has been evaluated in the past, they are not added to reduced graph G_R to avoid duplication. However, we add the nodes corresponding to the FF by retrieving them from the underlying data structure to sub-graph g since g is used for determining biases in picking sources.

2. *Invalid FF:* The FF picked up by the Cartesian space heuristic are used to generate nodes. The FF for which no IK solution exists is invalid. If a set $\mathscr{F}$ consists of all invalid FF, then we select the FF with the second-best shortest distance and recreate the set $\mathscr{F}$. This allows us to avoid terminating the progression prematurely and increases the likelihood of finding a path. If all the FF are invalid, the algorithm detects a discontinuity (unreachable semi-constrained waypoint) and returns with no solution.

2.5.7 Source Biasing

Sub-graphs g for forward and backward progression are independently generated from their sources. We use the cost of a path found by searching over a sub-graph as a direct measure of the source's quality. We assign a cost to each source and develop a cost map that will help us bias the source selection. We compute path cost for a sub-graph and store it in the underlying data structure with the corresponding source. All such costs associated with already evaluated sources give us distinct cost maps for the corresponding source sets $\mathcal{S}_1$ for forward progression and $\mathcal{S}_n$ for backward progression. Figure 2.9 shows a sample cost map for the costs of paths from sources within a FF set.

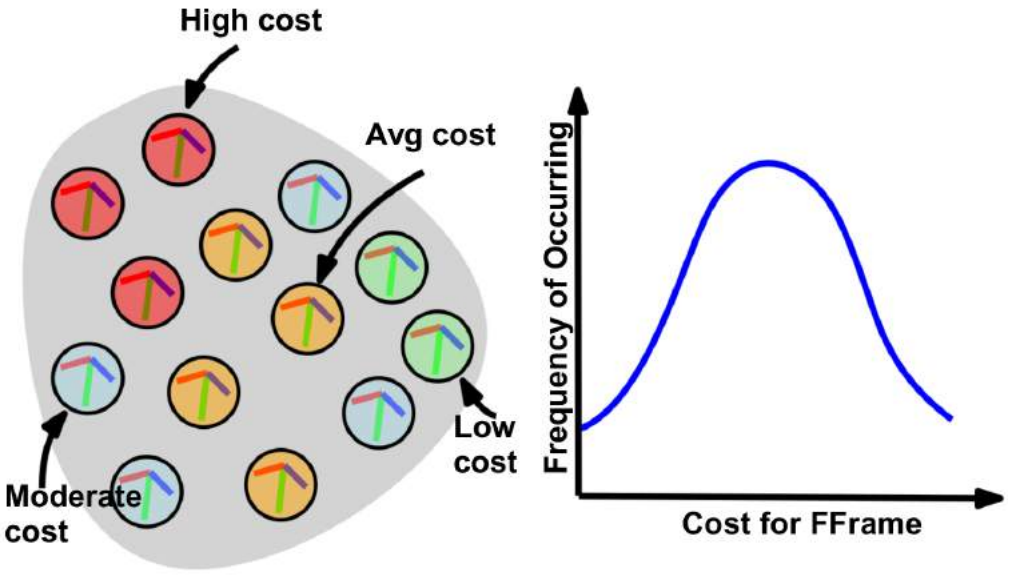

Figure 2.9: (Left) Cost map for a source set showing the variations in trajectory costs. (Right) Distribution of number of sources throughout the cost map. Average cost sources occur more as compared to low and high cost sources.

The sources also follow a cost distribution as illustrated by Figure 2.9 (right). Sources with moderate cost occur more in numbers as compared to high and low-cost sources. The distribution allows us to use a random exploration strategy during the initial iterations of the algorithm to identify the low-cost neighborhoods. The cost of sources smoothly varies within a given neighborhood. Future selection of sources within the low-cost regions will result in a higher probability of finding near-optimal cost sources during the search. The resulting source selection method is a heuristic-based stochastic optimization algorithm similar to simulated annealing. The neighborhood of a source is computed using the K-nearest neighbors search and the distance metric used is Euclidean distance between sources that are represented as vectors of position and orientation.

2.5.7.1 Probability Profile

The number of iterations required for the exploration and exploitation is dictated by a probability profile. Let p be the probability of randomly selecting a source. p will also be dependent on the number of FF remaining in the source set (also denoted as R_{src}). The probability profile is a curve

that describes p vs. R_{src} relationship. Additionally, we define two more parameters r and α. r is used to compute the value $R_{src} = r \times N$ till which we have $p = 1$, where N is the total number of FF in the corresponding source set (e.g. for forward progression, $N = |\mathcal{S}_1|$). α is used to express the rate of decay of the probability after $r * N$. We express the probability profile by the following equation. We have two profiles; one for forward and the other for backward progression.

$$p = \begin{cases} 1, \ R_{src} <= r \times N \\ \exp(-\alpha \times (\dfrac{R_{src} - r \times N}{(1-r) \times N})), \ R_{src} > r \times N \end{cases}$$

2.5.7.2 Source Selection Bias

When the value of a random variable sampled from a uniform distribution between 0 and 1 is less than p, the algorithm randomly selects a source to conduct explorations. A value greater than 1 directs the algorithm to exploit the cost map generated so far to select the next sources. The biasing routine first normalizes the costs in the cost map between 0 and 1. Then an average cost of all the sources in the cost map denoted by C_{avg} is computed. We define parameters k and δ to fix the number of neighbors picked by the k-nearest neighbor search algorithm and a threshold value C_r. We pick all the sources from the cost map having a value greater than $C_{avg} + (1 - C_r) \times (1 - C_{avg})$ and find the k nearest neighbors within a δ Euclidean distance from the picked source. The neighbors, along with the picked sources, are disabled in the source set. The disabled sources are not picked in future iterations until they are enabled again.

If the source set is not empty, we go through all the sources with a value less than $C_r \times C_{avg}$. The selected sources are potentially the ones with a good neighborhood and can contribute to lower-cost paths. We create a set of sources consisting of the picked good sources and their k nearest neighbors. A random sample is then generated from this set and returned by the biasing routine for carrying out the corresponding progression. Therefore, the returned source is selected such that it is biased towards the neighborhood of sources with a cost lower than a threshold

defined by C_r and C_{avg}. The algorithm first visits a good source. Once the good sources are exhausted, the algorithm considers other remaining sources. Eventually, the sources that were disabled in the bad neighborhood are enabled and explored. This ensures completeness.

2.5.7.3 Invalid Sources

The sources which are under collision or the search cannot find a path from the root to leaf in the corresponding sub-graph g due to no connections between any two levels of g are considered invalid. We assign a cost value of 1 for such sources. The biasing scheme will disable all the samples within k neighborhood of such bad sources. Hence, we can avoid potential regions in the source set, which comprises sources under collision or poor paths.

2.6 Results

2.6.1 Test Cases

2.6.1.1 Serial-Link Manipulator

The robot used for testing our planner is a ABB IRB 2600 series which has 6 DOF. We use different sets of industrial tools which attach to the robot end-effector. Figure 2.10 shows the three tools used in our work. The following section describes the characteristics of each tool.

2.6.1.2 Tools

Tool-A is a sanding tool with a pneumatic compliance mechanism attached to the top of the sander. The sanding disc has 15 different TCPs arranged along the sander's perimeter and at the center of the sander. The robot flange attaches along the axis of rotation of the spindle at the top of the tool. Tool-B is a deburring tool used for clearing burrs off the edges of a part. It has a sanding ball tip that comes in contact with the part. The corresponding TCP at the ball tip is illustrated in the figure. The robot flange attaches at a 90 degrees angle with respect

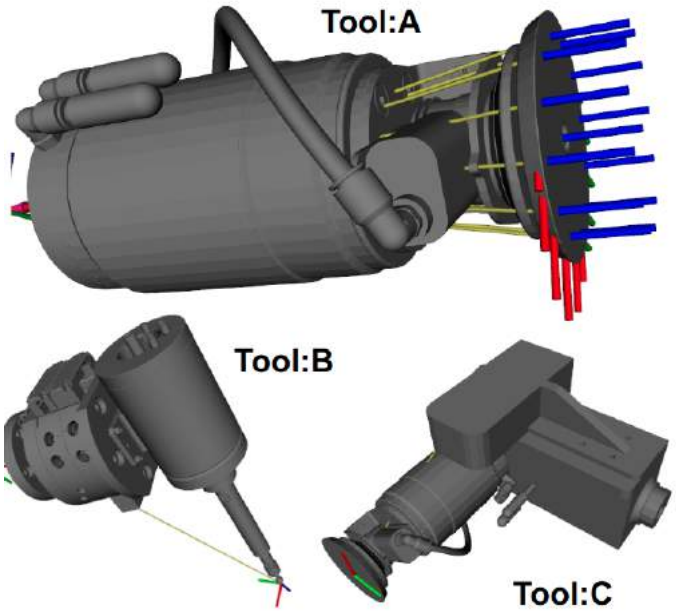

Figure 2.10: (A) Sanding tool with pneumatic compliance head. Multiple TCPs are defined by the user on sanding surface. (B) Deburring tool with a spherical abrasive surface. A single TCP is defined. (C) Sanding tool with a single TCP.

to the rotation of the spindle. Tool-C is another sanding tool. In contrast to tool-A, tool-C requires the robot to hold the tool at an angle of 90 degrees with respect to the sander's axis. The different orientations of the robot flange with respect to the tool TCP produce variations in TCP transformation and help in generating diverse test cases.

2.6.1.3 Parts

We selected a diverse collection of parts with respect to geometry and complexity of toolpaths to evaluate our planner's performance. Figure 2.11 illustrates the five parts and the respective tool paths on them. All tool paths generated follow a systematic method to compute the nominal toolpath $\mathscr{P} = \{p_1, p_2, \ldots, p_n\}$. The Z-axis of p_i goes into the part surface along the normal. The X-axis of p_i is along the direction in which the next waypoint lies. The Y-axis follows the standard Cartesian axes right-hand thumb rule. The bounds B_i defined over the toolpath waypoints are test case-specific, and we will discuss them along with each test case in this section.

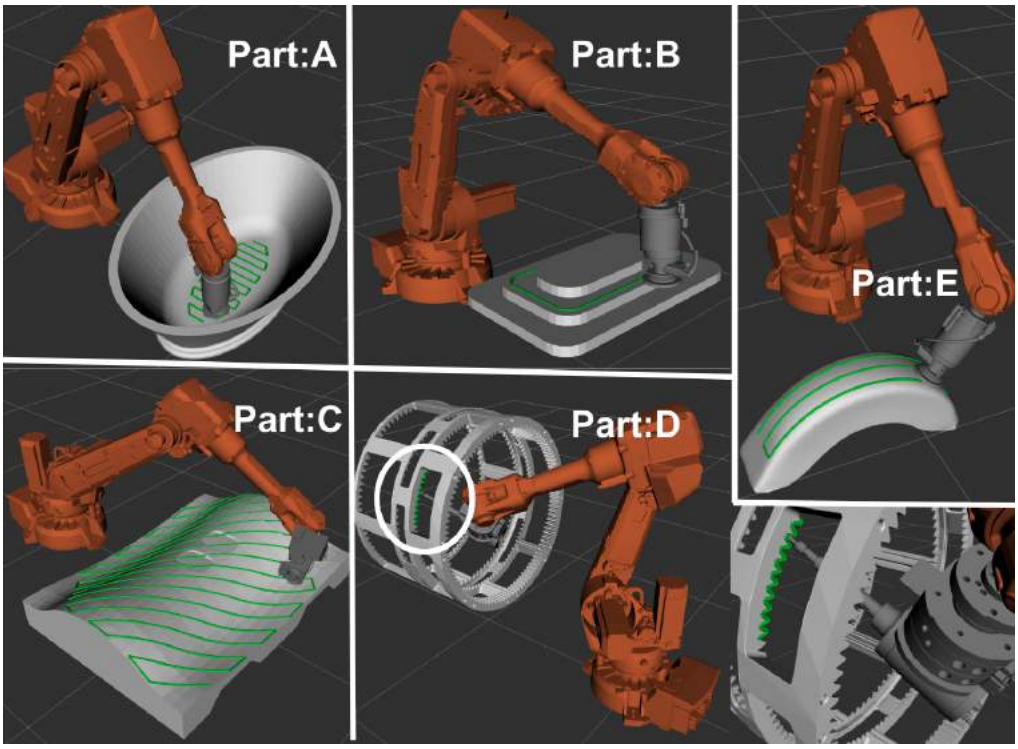

Figure 2.11: Five industry inspired complex parts with the corresponding toolpaths are shown. The parts are placed in the robot Cartesian space such that they are fully reachable by the robot to evaluate algorithm performance.

There are no bounds provided to position for all the test cases. We assume that the parts are not lying out of the robot Cartesian space or are close to the singularity. So the robot is generally reachable in position within a small sphere about the nominal waypoint frame origin. The lower and upper bounds on orientation are denoted by the vector $< \alpha, \beta, \gamma >$ corresponding to rotation about X, Y, and Z axes of the nominal waypoint frame p_i. Table 2.1 shows the orientation bounds and the sampling resolution to discretize them. We also overly constraint the test cases A and B by reducing the bounds at 30 uniformly spaced waypoints. The overly constrained bounds represent general end-effector constraints. The amount of reduction is given by a scalar factor multiplied to the original bounds given in the Table 2.1.

Part	Tool	Multiple TCPs	Orientation Lower Bound (Degrees)	Orientation Upper Bound (Degrees)	Bound reduction factor	Resolution (Degrees)
A	A	Yes	<0, 0, -180>	<0, 0, 180>	0.05	5
B	A	Yes	<0, 0, -180>	<0, 0, 180>	0.2	5
C	C	No	<0, 0, -180>	<0, 0, 180>	NA	5
D	A	Yes	<0, 0, -180>	<0, 0, 180>	NA	5
E	B	No	<-180, -180, -180>	<180, 180, 180>	NA	30

Table 2.1: Bounds in orientation, reduction factor, and sampling resolution is presented for the five test cases.

2.6.1.4 Toolpath Details

The number of waypoints along the tool paths is shown in Table 2.2. The average number of FF created after applying TCPs and bounds to them are also presented. We also present the maximum and minimum number of FF across all the levels. The minimum number of FF is significantly lower than the maximum FF for test cases A and B as they have 30 waypoints where the bounds are reduced. The fewer availability of FF makes the problem more challenging as the number of low-cost paths reduces.

2.6.2 Performance Evaluation

Since it is difficult to incorporate TCPs, tolerances, or both in the existing methods, we will use exhaustive graph construction and random sampling-based techniques to benchmark our algorithm. The exhaustive graph building will provide insights into how redundancy due to tolerances and TCPs causes a reduction in computational efficiency. The random sampling-based technique will provide insights into the importance of having a Cartesian space heuristic and source selection

Part	# Waypoints (Graph levels)	Avg # FFrames per level	Max/Min # FFrames
A	141	756	936 / 65
B	128	771	936 / 208
C	206	72	72 / 72
D	167	746	746 / 746
E	69	936	936 / 936

Table 2.2: The number of waypoints or graph levels and maximum (Max), minimum (Min), and average (Avg) number of FF across all the levels for the five test cases is presented.

routine to selectively evaluate FF instead of sampling. The reduction in path cost with respect to the total number of nodes being evaluated will be used to measure the performance.

2.6.2.1 Exhaustive Graph Construction

We implemented the exhaustive graph construction using C++ and Robot Operating System (ROS) infrastructure. This method is similar to the Descartes planner [166] which is state of the art for the class of problem. We take all the TCPs and tolerances into account while constructing the exhaustive graph. Tables 2.2 and 2.3 provide insights into the immensity of Cartesian waypoints and corresponding nodes and edges in the graph for the test cases used. We can see how a full graph construction becomes infeasible and computationally expensive. The computation time depends on the number of nodes evaluated during the graph search. We benchmark the performance of baseline approaches and our algorithm based on the number of evaluated nodes. Table 2.3 provides the total number of nodes and valid nodes evaluated. The total number of edges and valid edges evaluated are also shown.

Table 2.4 provides more details on the computation time in seconds and the path cost for τ^*. Computation time can be used as a reference to gauge the time complexity of the algorithm and how long it takes to solve commonly found test cases in applications. The table also presents the

Part	# Nodes Evaluated	# Nodes Inserted	# Edges Evaluated	# Edges Inserted
A	274594	175390	222484672	21171855
B	306472	147618	193053856	24579276
C	38914	28488	4028680	458125
D	360812	35635	8831587	510755
E	186816	186818	541363968	41485079

Table 2.3: The number of nodes and edges evaluated, nodes and edges inserted (valid) into the graph G are provided for the five test cases.

time taken to perform a Djikstra search over the fully constructed graph. The graph search time is insignificant as compared to the overall time taken for node evaluations. The search time is lesser for smaller graphs such as reduced graph G_R used in our algorithm.

Execution Time in Seconds and Optimial Path Cost					
Part	Node Generation	Graph Generation	Search Time	Total Time	*C*-space Path Cost
A	165.83	9.06	0.28	175.17	4.16
B	123.92	9.37	0.52	133.81	5.46
C	14.96	0.12	0.01	15.09	46.28
D	168.8	0.193	0.017	169.01	6.09
E	85.49	18.57	0.7	104.76	8.31

Table 2.4: Details on computation time or code execution time in seconds and optimal path costs for exhaustive graph construction is presented.

2.6.2.2 Random Sampling

We use a random sampling method to provide another baseline for benchmarking against our algorithm. The random sampling method goes across all the FF sets and randomly chooses an FF to evaluate and insert into the graph.

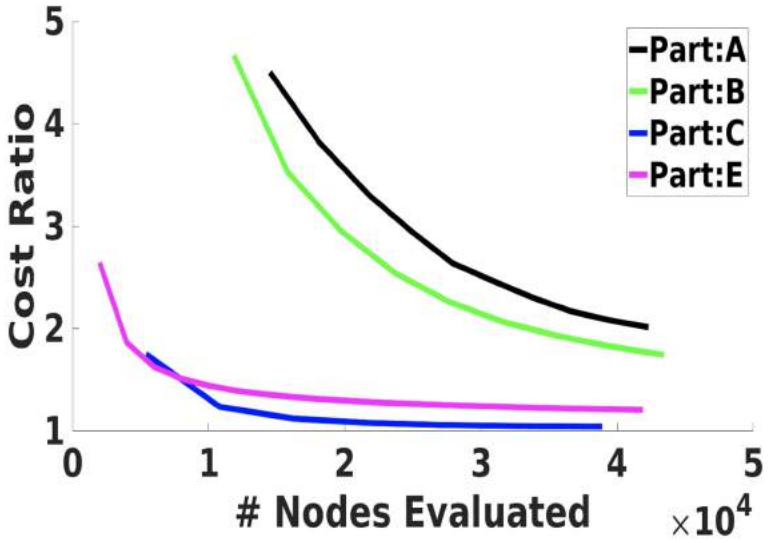

Figure 2.12: Cost ratio (current cost divided by optimal cost) vs. number of nodes evaluated for random sampling method is shown by the curves for four test cases. Random sampling does not find a solution for Part D within 5e4 node evaluations. The cost ratio is averaged over 50 runs of the algorithm.

Figure 2.12 shows a cost ratio or the ratio of path cost to the optimal cost found by exhaustive graph construction vs. the number of nodes evaluated. We terminate the random sampling method between 4×10^4 - 4.5×10^4 node evaluations depending on the test case. The results are generated by averaging the cost over 50 runs of the algorithm. We find that random sampling generates bad quality paths, which is shown by a very high-cost ratio. The method also converges at a much slower rate for parts A, B, and E and does not find a solution at all for part D. For easier test case C, the method converges close to the optimal solution but takes a higher number of node evaluations. However, the random method will also find the optimal solution as it eventually evaluates all the FF within the sets.

2.6.2.3 Our Approach

We evaluate the performance of our approach in two scenarios. We first provide results for running our algorithm without introducing the source bias. The second scenario introduces a

source biasing scheme. Both the scenarios provide insights into how the performance improves when source regions with a lower cost are exploited instead of random sampling.

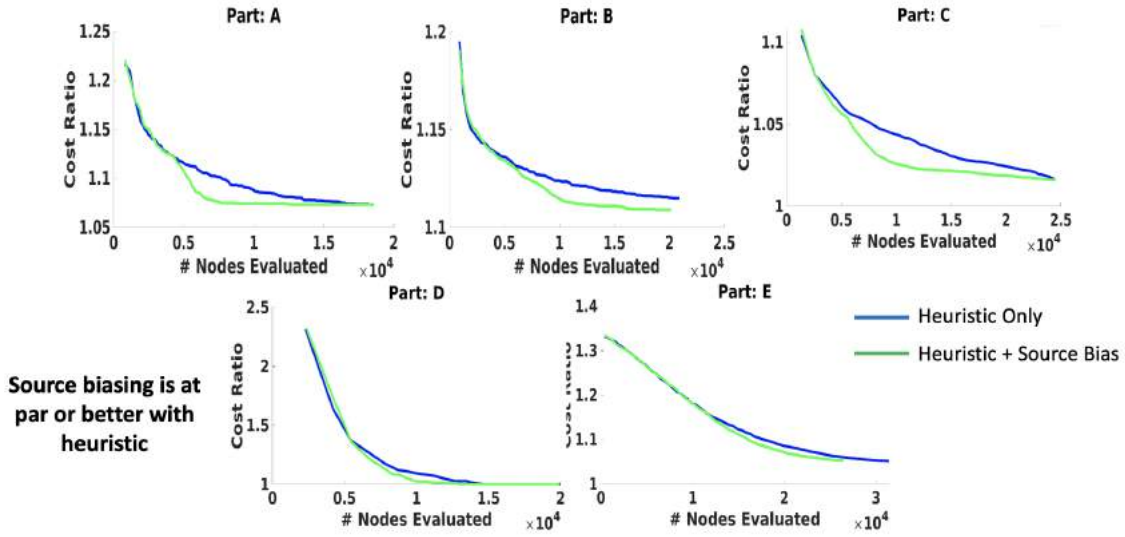

Figure 2.13: Cost ratio (current cost divided by optimal cost) vs. number of nodes evaluated averaged over 50 runs of our algorithm for five test cases. Blue curve represents algorithm runs without source biasing. Green curve represents the scenario where source biasing is introduced.

Figure 2.13 shows the convergence profiles for all the test cases for scenario-1 (without bias) in blue and scenario-2 (with bias) in green. The results are averaged over 50 runs of both scenarios. The algorithm terminates when all sources (forward and backward) are exhausted. It is important to note that there are FF in the sets other than the source sets which remain unexplored and the sampling steps of the algorithm exhaustively explore these sets if the stopping condition is not met. However, for the purpose of evaluating the heuristic and source biasing scheme performance, we consider exhaustion of the source sets as the stopping condition. Our approach finds a near-optimal solution with fewer nodes as compared to the exhaustive graph construction or random sampling scheme. Our approach also generates a near-optimal solution in all the cases as opposed to parts A, B, and D (no solution) random sampling with a cut-off limit of 5e4 node evaluations.

Table 2.5 shows the cost ratios (algorithm output path cost divided by the optimal path cost). We generate the ratio for two scenarios; before and after application of smoothener. The cost

ratio shows the capability of our algorithm to generate near-optimal solutions. The output paths are within 1% more cost of the optimal solution.

Part	Cost ratio before smoothing	Cost ratio after smoothing
A	1.07	1
B	1.11	1.06
C	1.03	1.01
D	1	1
E	1.05	1

Table 2.5: Optimal cost ratios (algorithm output path cost divided by optimal path cost) for paths generated our approach before and after smoothing.

The parameters for determining the probability profile are r and α. r governs how many FF are to be evaluated using random sampling and α is the decay rate of random vs. bias sampling profile. We tuned the parameters for all the test cases. The values of r for parts A, B, C, D, and E are 0.07, 0.04, 0.015, 0.1, and 0.04. The value of $\alpha = 10$ was used for all test cases. The value C_r that determines the normalized source cost threshold to be used for separating good and bad sources was set equal to 0.2. The number of neighbors k that are considered within the neighborhood of a source are set to 10 in our implementation.

2.6.3 Cartesian space Heuristic Performance:

We can observe that the initial cost ratios for all the test cases in Figure 2.13 range from 1.1-2.5. On the other hand, random sampling has an initial cost ratio of 2-5. The gains are obtained by selecting FF in successive levels using our Cartesian space heuristic. The results concur with the kinematic plot discussed in Figure 2.8. The plot showed that the relationship between change in pose of end-effector (Δx) with respect to the change in configuration ($\Delta \vec{q}$) is linear below abound

on δx. Therefore, selecting the nearest FF from successive sets reduces the distance of $\mathscr{C}$-space path.

2.6.4 Benefits of Reduced Graph Construction:

Table 2.6 provides insights into the quality of forward ($\mathcal{S}_1$) and backward ($\mathcal{S}_n$) source sets. The configuration space path cost for the best path in sub-graphs generated from each source is evaluated. The table shows the minimum, maximum, and average cost of paths. The costs are higher compared to the optimal path cost. The reduced graph G_R contains nodes that come from a wide range of FF. Sampling from multiple sources in low-cost regions and using bidirectional progressions enable us to find better paths instead of generating independent paths from a single source.

	Forward Source Set			Backward Source Set			
Part	Avg.	Min	Max	Avg.	Min	Max	Optimal Cost
A	6.65	4.6	8.43	6.62	4.62	8.62	4.16
B	8.49	6.52	11.93	9.15	6.27	12.01	5.46
C	55.92	49.91	63.7	52.63	49.37	61.85	46.28
D	34.45	6.66	157.18	34.51	7.08	127.39	6.09
E	11.73	10.93	24.65	11.69	10.97	24.74	8.31

Table 2.6: Average (avg.), minimum (min), and maximum (max) path costs found by searching through sub-graphs for sources are presented. The sources in forward and backward set that generate a valid trajectory are selected for generating the data. Optimal cost of paths are given for comparison.

2.6.5 Completeness

Our source selection algorithm explores and exploits the sources in $\mathcal{S}_1$ and $\mathcal{S}_n$ for forward and backward progression steps. The algorithm ensures that all the sources (FF) in both sets are

exhaustively selected. The algorithm does not discard any sample. The rejection steps only disable bad regions but later evaluate them as all good and moderate regions are exhausted. Hence a preference is generated, and FF are not discarded.

The FF in intermediary levels may remain unevaluated when all the sources in $\mathcal{S}_1$ and $\mathcal{S}_n$ are exhausted. Therefore, we switch to a random sampling of FF if the termination criteria have not been reached and keep constructing G_R. Random sampling stops as termination criteria are reached, and a solution is returned after smoothening. Therefore, we ensure the resolution completeness of the intermediary waypoints.

In this way, sampling all the FF in source sets and intermediary levels guarantees our algorithm's resolution completeness.

2.7 Summary

A Cartesian planner for using semi-constrained waypoints and multiple TCPs was developed in this work. We developed a method to leverage the benefit of the graph-based planners that explicitly sampled feasible robot configurations. A reduced graph was constructed by mapping the Cartesian space poses that correspond to robot flange locations to the robot configurations. A Cartesian space heuristic was developed that smartly selected the flange frame to find good quality trajectories. A bi-directional progression used the Cartesian space heuristic to build sub-graphs. We showed how the trajectory was dependent on the source flange frame. An approach was developed to compute a cost map from each source's sub-graphs, and source selection was biased to a good neighborhood. A test suite of five industry-inspired complex parts with semi-constrained Cartesian toolpaths was used to evaluate the algorithm performance. We obtained significant gains in computational performance using our approach compared to existing methods.

Chapter 3

Algorithms for Acquiring High Fidelity PointClouds Using a Robot for Accurate and Fast 3D Reconstruction

3.1 Introduction

3D reconstruction of surfaces requires using a sensor to collect data from the part surface. Commonly used sensors have a limited operational range and field of view. This limitation requires us to move the sensor around a complex part to obtain complete information. Robots enhance the sensor capability to collect data. The use of 6 degrees of freedom (DOF) robots enables sensor position and orientations to be selected to avoid occlusions found on complex parts. In most applications, the goal is to perform the 3D reconstruction process with acceptable accuracy in the smallest amount of time.

Many commonly used sensors, such as RGB-D cameras exhibit non-uniform performance in their workspace. A depth camera might provide a large field of view and workspace. However, only a small portion of the workspace provides high accuracy. For instance, Figure 3.1 shows three points observed by the camera at different locations. The location closer to the camera lens has a lower error compared to the point farther away even though they are within the operational range. Many implementations use the entire operational range of the camera to collect the pointclouds to simplify the planning process and reduce the model construction times. Figure 3.2(top) shows a

minimum execution time fixed path generated by using entire camera range. However, the output pointcloud will have a low accuracy in the model. On the other hand, an overly conservative operational range will increase accuracy but result in an increase in the model construction time. Figure 3.2(bottom) shows the longer path that uses a more high fidelity sensor range. We need to balance data collection time and measurement accuracy during the robot motion planning phase.

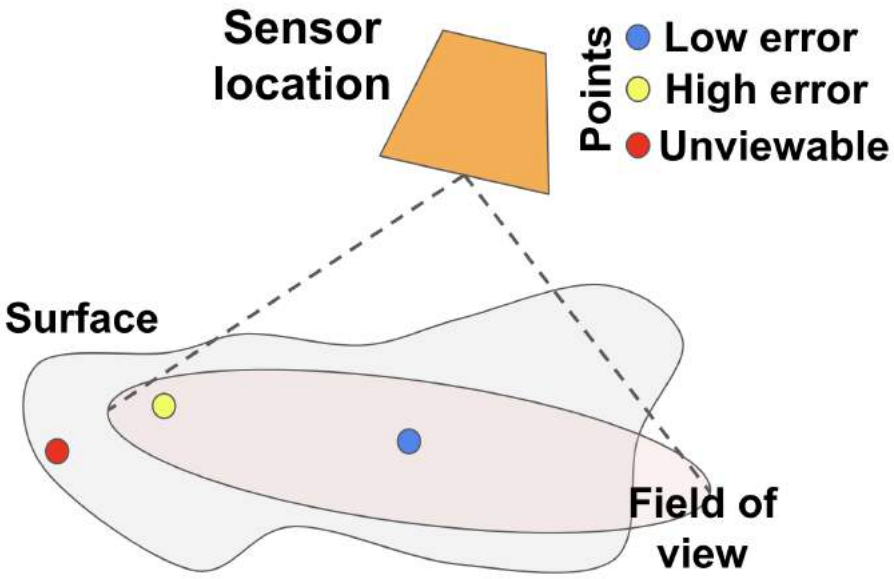

Figure 3.1: Three points on a surface are viewed from different camera locations. The point closer to camera optical lens has a low error and the point farther from the lens has a high error even though both are within the field of view (FOV). Point outside the FOV is unviewable.

We are interested in a camera placement planning approach that enables the construction of the part model with acceptable accuracy in the minimum amount of time. Camera constraints are also required to be enforced during planning. The constraints must meet the following requirements. 1) Generate camera and robot plans that produce high-quality (or low error) pointclouds. 2) The points should be measured from diverse and multiple camera locations. 3) External factors will still create regions on the measured 3D reconstruction to have missing data or low point density. We also need an online plan refinement by using live feedback during data collection to refine the robot motion and guarantee a uniform high density of points across the 3D reconstruction.

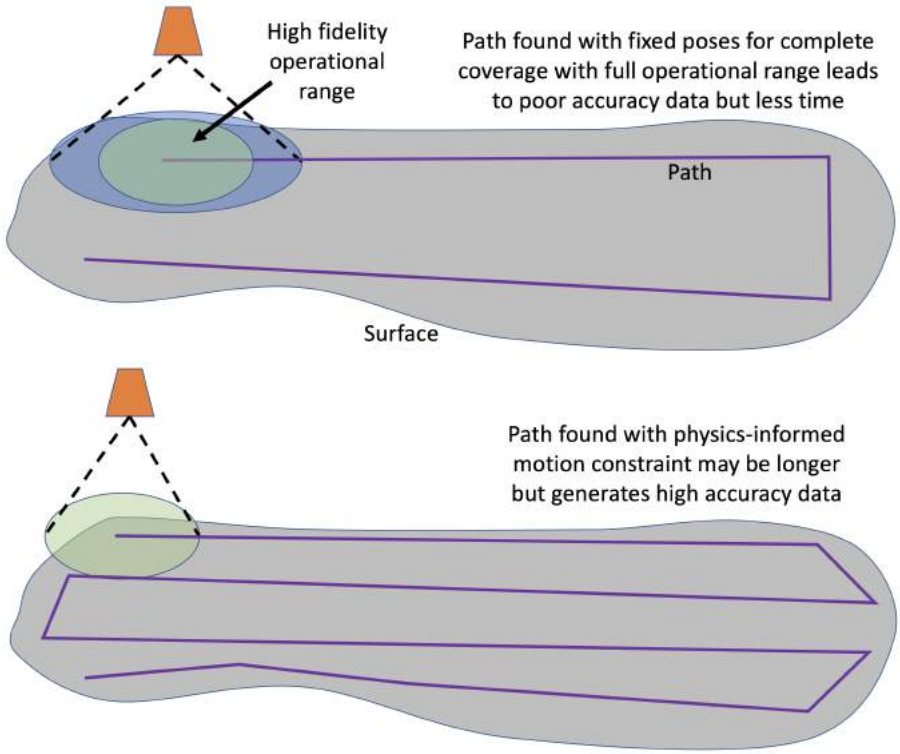

Figure 3.2: (Top) Full operational range of sensor is used to capture an arbitrary surface. Fixed path is generated to minimize execution time, however, output accuracy is reduced. (Bottom) High fidelity operational range of the sensor is used leading to longer paths to cover entire surface but improving accuracy.

Our goal is to realize an automated 3D reconstruction cell equipped with a low-cost RGB-D sensor mounted on an industrial robot to capture a 3D pointcloud of a part that conforms to the part surface with high accuracy and guarantee uniform density. Figure 3.3 shows our automated robotic 3D reconstruction cell. In our previous work [125, 126], we developed a system to compute the robot motions by using the complete operating range of the camera. The system also executed only offline trajectories leading to low or zero density points. In this paper, we extend our previous work by developing the planning constraints using a camera workspace that enables high-quality pointcloud collection. This work also guarantees uniform point density on the part with an online refinement planning module. We make the following contributions in this work: 1) Develop camera performance constraints that will accept high-quality data using the

specific camera operational range. 2) Robot motion planning under the camera performance and robot kinodynamic constraints. 3) An online trajectory refinement method to identify missing low or zero density regions on the part and improve the point density.

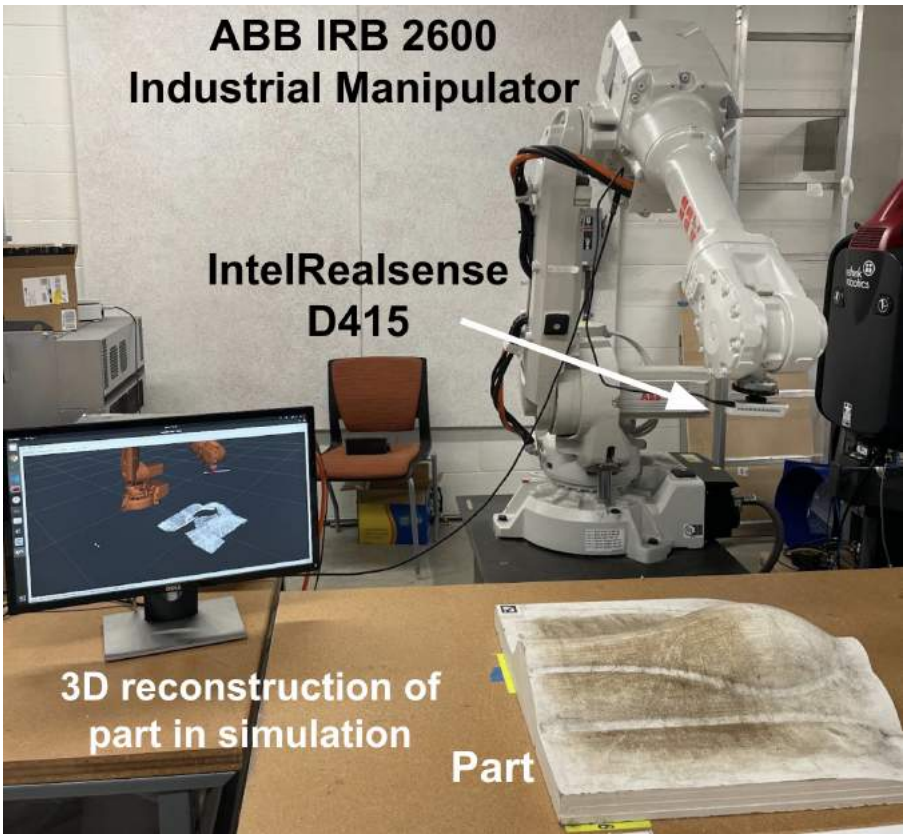

Figure 3.3: Robotic inspectison cell developed in this work. The video for our system is available at: `https://youtu.be/6-d2MBIzApY`

3.2 Related Work

Broadly, the problem of robot motion planning for 3D reconstruction involves three main components: set of camera viewpoint (location) generation, camera path generation, and pointcloud augmentation. We discuss the relevant work done on viewpoint generation in the first paragraph and the work done on camera path generation through the viewpoints in the second paragraph of

this section. The third paragraph focuses on how the pointclouds captured during robot motion execution are augmented to create the output cloud.

Camera viewpoints are used to position the camera around the part. Glorieux et al. used a targeted viewpoint sampling with an optimization strategy to get the viewpoints in [70]. Bircher et al. developed an online method in a receding horizon fashion by sampling possible future viewpoints in a geometric random tree [21]. Vasquez-Gomez et al. developed a next best view algorithm to determine viewpoints for recreating an arbitrary object [195]. Raffaeli et al. developed a clustering-based approach to optimize the number of viewpoint samples [162]. Jing used randomized sampling-based and medial object-based methods to sample the viewpoints for a higher coverage [90]. González-Banos et al. have developed a random-art gallery algorithm to sample the viewpoints [72]. Viewpoint planning was also solved using reinforcement learning in [44, 109]. A MATLAB toolbox-based approach for path planning for non-destructive testing was presented in [142]. Another coverage planning method for eddy current-based non-destructive testing was developed in [154]. Most of these works are focused on generating viewpoints using a complete camera operational range. We develop a method that will enforce camera performance constraints during path generation.

The sensor needs to be moved to the viewpoints by finding a path that minimizes an objective function. Generally, the objective is to reduce execution time. A survey on coverage plan generation is presented in [2]. Siyan-Dong et al. [49] presented a multi-robot collaborative coverage planning based on the traveling salesman problem (TSP). Papadopoulos et al. developed an algorithm called the random inspection tree algorithm. The algorithm generated a coverage path planning concurrently while generating the viewpoints [157]. Janoušek has implemented a method to speed up coverage queries, which is helpful in concurrent path planning and viewpoint generation [87]. The work done in [59] presented a sampling-based subroutine to improve the coverage path by making asymptotically optimal local changes on the feasible path. The authors also presented a hybrid planner to fill in the gaps to get a complete coverage path of complex

3-dimensional structures [60]. The popular approaches for solving coverage path planning are Lin-Kernighan traveling salesman heuristic [76], rapidly exploring random trees [111], probabilistic road map [98] and their modifications. Reinforcement learning has been used for mapping and exploration in coverage planning [8, 32]. Our previous work in [130] uses graph-based planning for robot path generation through viewpoints. Most of the approaches do not incorporate all the robot kinematic and dynamic constraints while solving the problem. In this work, the robot trajectory is generated over the viewpoints under reachability, singularity, continuity, velocity, collision, and camera performance constraints.

The pointclouds captured during robot trajectory execution have to be augmented to produce the output cloud. A widely used approach is to use a grid of cubic volumes of equal size, called voxels to discretize the mapped area. The work done in [86, 147] uses such a representation. An oct-tree-based representation was used in the work done in [82]. Whereas, an R tree was used in [100]. Another solution is to use a hierarchical data structure that was proposed in [82, 100]. Besides the point-based methods, an alternative is to use a fully volumetric data structure to implicitly store samples of a continuous function [79, 39, 198]. In these methods, depth maps are converted into signed distance fields and cumulatively averaged into a regular voxel grid. Newcombe et al. [150] provides a truncated sign distance function implicit surface representation with a GPU accelerated voxel grid pipeline. The work done in [152] provides a hashing scheme to overcome the difficulties posed by hierarchical structures and simple voxel grids. Unlike the conventional point, mesh, and voxel-based 3D reconstruction representations, [139] and [88] employed a 3D geometry-based representation by learning a continuous 3D mapping. Rigid Fusion [199] and Deep Deform [25] creates non-rigid 3d models. We use a voxel-based method to generate the output cloud in this paper. Our method efficiently handles the massive number of pointclouds that continuous camera triggering obtains. Now we will discuss the terminology we use throughout the paper.

3.3 Background and Terminology

A pointcloud is used to represent the 3D geometry of a part. A point in a pointcloud is represented by the coordinate vector $< x, y, z >$ in the Cartesian space. The true part geometries we use in the test cases are represented by reference pointclouds that are captured using high-accuracy metrology instruments. The reference cloud is denoted as $\mathscr{P}$. A 3D reconstruction of the part using our method is represented by a pointcloud denoted as $\widetilde{\mathscr{P}}$.

Pointclouds are used to construct a 3D grid of cubes called as a voxel grid. The data structure makes it computationally efficient to perform operations on the cloud. The cube has an associated side length which is chosen based on the resolution we need. A voxel represents several points together as a center point of the voxel. We denoted such voxelized representations as $\mathscr{V} = \{v_1, v_2, \ldots, v_m\}$, where v_i is a 3D cubic voxel.

The robot, camera, and part used in the setup are rigid objects in the Cartesian space. Locations of these objects are represented by attaching a frame O. The frame O defines the position of the frame origin and orientation of three-unit direction vectors along the X, Y, and Z axes also known as the pose of the frame. Our work uses the frames attached to the camera (O_c), part (O_p), robot end-effector (O_e), and robot base (O_b). O_b is also used as the global reference frame. A homogeneous transformation matrix T establishes the relationship between a pair of frames [176]. For instance, points represented in coordinate system O_c can be transformed to O_b using ${}^bT_c$.

A pose in the Cartesian space with which the camera frame O_c aligns is also called as a camera viewpoint I. A set of viewpoints can be defined around the part to generate a state space. A sequence of viewpoints within the state space defines a camera path $P = \{I_1, I_2, \ldots, I_k\}$.

A serial link manipulator comprises of a sequence of rigid links connected by revolute joints. A configuration vector $\vec{q}$ describes the positions of all the joints. For a K degrees of freedom (DOF)

robot, $\vec{q} \in \mathbb{R}^K$. In this work, we use a 6-DOF ABB IRB 2600 industrial manipulator that makes K=6. All such robot configurations $\vec{q}$ belong to a space called the configuration space ($\mathscr{C}$-space).

We need to represent robot paths needed for moving the camera in terms of robot joints. We will refer to these paths as configuration space paths. A configuration space path τ in $\mathscr{C}$-space is defined as $\tau = \{\vec{q}(u) | \vec{q}(u) \in \mathscr{Q}_f\}, u \in [0, 1]$, where u is a trajectory parameter and Q_f is the set of feasible robot configurations in $\mathscr{C}$-space. The robot joint configuration and pose of the flange are related by forward kinematics or FK. Joint velocity ($\dot{\vec{q}}$) in and the Cartesian velocity of the flange in Cartesian space $\dot{x}$ are related by the robot Jacobian J as $\dot{x} = J \times \dot{\vec{q}}$, where $J \in \mathbb{R}^{6 \times N}, \dot{x} \in \mathbb{R}^6$ and $\dot{\vec{q}} \in \mathbb{R}^K$. The robot description and functionalities are stored under robot model $\mathscr{R}$.

An initial pointcloud estimation $\mathscr{P}_0$ of the part geometry is used for robot motion planning and construction of a voxel grid data structure $\mathscr{V}$. $\mathscr{P}_0$ may contain points with high errors and is only an estimate of the geometry. $\mathscr{P}_0$ is obtained by sampling points on the CAD model (if available) of the part and introducing noise that results from part manufacturing and localization errors. $\mathscr{P}_0$ can also be obtained by taking the robot on a few user-defined viewpoints over the working surface and augmenting all the pointclouds. Section- 6.5.1 describes the generation of $\mathscr{P}_0$.

$\mathscr{P}_0$ is converted to a voxel representation $\mathscr{V}$ and the set of points in a voxel v_i are denoted as $P(v_i)$. The collected points during the reconstruction process are represented as $\mathscr{P}_c$ and mapped to $\mathscr{V}$. The true point will be within a voxel in $\mathscr{V}$ since manufacturing and localization errors are taken into account while constructing $\mathscr{P}_0$. Each surface voxel in $\mathscr{V}$ is used to construct a corresponding $\mathscr{N}$-neighborhood having at least N diverse points. The details on computation of $\mathscr{N}$ are discussed in Section-3.5.6 and binary diversity function diverse(p_i, p_j), $i \neq j$ is defined in Section-3.5.3. The points in the $\mathscr{N}$-neighborhood are then reduced to a single point. Reducing all such $\mathscr{N}$-neighborhoods produces the output pointcloud $\tilde{\mathscr{P}}$.

A point p in $\mathscr{P}_c$ is filtered before it is mapped to $\mathscr{V}$. A point admittance function $\eta(p, \mathscr{M})$ makes a binary decision (True or False) on whether an observed point p should be admitted into

$\mathscr{P}_c$ or rejected thereby controlling the quality of the data. Details on the admittance function and its characterization using a camera performance model ($\mathscr{M}$) are discussed in Section-3.5.3. The Section also introduces the parameters that the function takes. We will now discuss the problem formulation for our work.

3.4 Problem Formulation

The method takes as an input an initial pointcloud $\mathscr{P}_0$, robot model $\mathscr{R}$, the camera performance model $\mathscr{M}$, $\eta(p, \mathscr{M})$, and N. Our goal is to find an output pointcloud $\tilde{\mathscr{P}}$ of the part using all the captured $\mathscr{P}_c$. We formulate the problem in three steps given by equations 3.1, 3.2, and 3.3.

$$\text{Add point } p \text{ to } P(v_i) \text{ if} \tag{3.1}$$

$$(i)\ p \in v_i$$

$$(ii)\ \forall v_i \in \mathscr{V}, \exists \mathscr{N}_i$$

$$(iii)\ \forall i, j\ \text{diverse}(p_i, p_j) = True,\ i \neq j$$

$$(iv)\ \forall p \in \mathscr{N}_i,\ \eta(p, \mathscr{M}) = True$$

$$\text{Terminate point insertion if} \tag{3.2}$$

$$(i)\ \text{cardinality of } \mathscr{N}_i \geq N$$

$$\text{Finally, construct } \tilde{\mathscr{P}} \text{ from } \mathscr{P}_c \text{ by averaging points in } \mathscr{N}_i \tag{3.3}$$

$$\text{where, } P_c = \bigcup_i P(v_i)$$

$\mathscr{R}$ is constructed based on the robot used. $\mathscr{M}$ is decided by the camera used in the system. The selection of N is a user-specific choice. The selection is based on the accuracy required from the final pointcloud. The accuracy is related to N by representing the uncertainty of a point captured by the camera as a Gaussian distribution. σ is the variance for the distribution of error

for a point. The new variance for N points will be given by $\sigma/\sqrt{N}$. Increasing N improves accuracy but will lead to higher execution times with diminishing gain in accuracy. Section-6.6 will further present the influence of N on the accuracy.

3.5 Approach

3.5.1 Overview of Approach

Figure 3.4 gives the overview of our system along with the individual modules and the libraries we have used. The framework is built on top of Robot Operating System (ROS) architecture and uses libraries such as MoveIt, open3d, scipy, Intel RealSense for ROS, and RViz. Input to the system is as follows:

1. A pointcloud estimation $\mathscr{P}_0$.

2. The user estimated transformation ${}^{e}T_{c}$ and ${}^{b}T_{p}$ that will be used for camera localization and placement of voxel grid $\mathscr{V}$ with respect to the robot base.

Our system performs the 3d reconstruction using the following different components.

1. *Camera localization:* The transformation ${}^{e}T_{c}$ can be estimated using the CAD model of the camera mount but is not accurate to produce reliable measurements due to manufacturing and assembly errors. Our system uses a multi-stage optimization strategy to localize the camera with sub-millimeter accuracy and the process is described in the Section-3.5.2. Another variant for an accurate camera localization method can also be found in the work done in [115].

2. *Generation of $\mathscr{P}_0$:* $\mathscr{P}_0$ is generated by sampling points over the surface of the CAD (if available). However, $\mathscr{P}_0$ is not the true representation of the real part due to manufacturing and localization errors. Additional points are therefore introduced in $\mathscr{P}_0$ by sampling the measurement uncertainty for each point to include all possible regions where the true point can exist. For

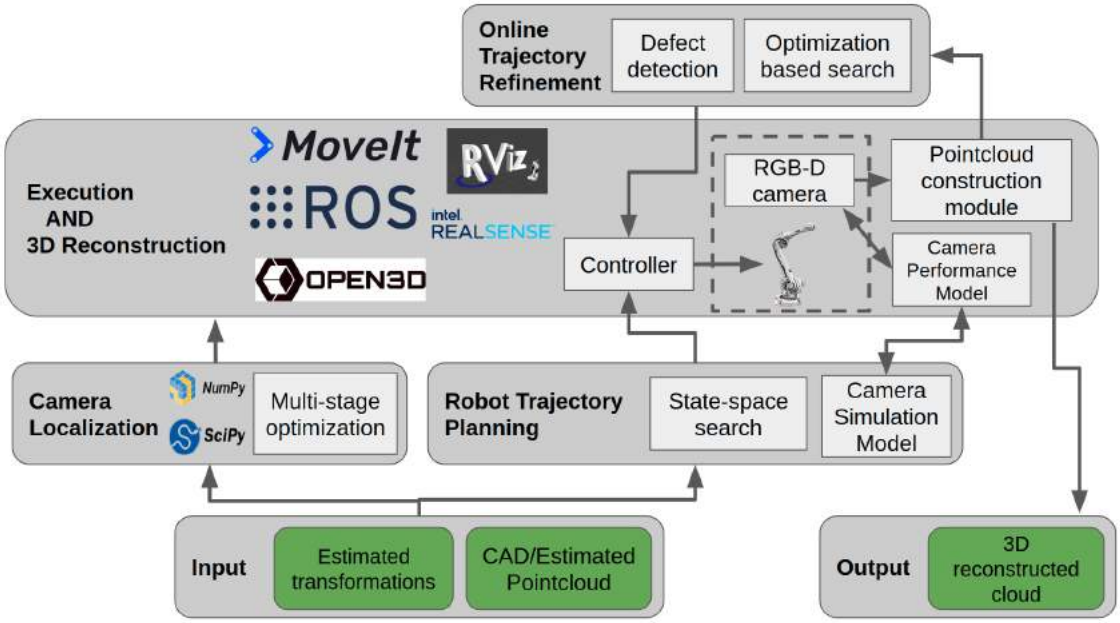

Figure 3.4: Block diagram representing the system architecture used by our method during reconstruction.

instance, in our implementation, we used a maximum position and orientation uncertainty of 40 mm and 30 ° respectively for each point. Each point on the CAD model was then be perturbed with many positions and orientation errors sampled within the limits -40 to 40 mm and -30° to 30° respectively. The resultant $\mathscr{P}_0$ is a cloud of points around the CAD surface that will entail the true point.

When the CAD model is not available, the robot will have to take the depth camera to a few predefined viewpoints over the part and augment all the points that correspond to the part without filtering them using admittance function η.

3. *Camera Path Generation:* Viewpoints are generated around the part using the CAD model. A state-space search is then performed over feasible viewpoints using a simulated camera model under the camera performance model constraints to find P. Section-3.5.4.1 describes this step.

4. *Robot Trajectory Generation:* The robot trajectory is generated in the $\mathscr{C}$-space from the camera path P. Robot trajectory generation algorithm enforces robot-specific constraints like reachability, collision avoidance, path consistency, continuity, and velocity on top of the camera performance constraints to find an executable robot trajectory. Section-3.5.5 describes this step. Pointclouds are captured and augmented using the 3D reconstruction step of our method.

5. *3D Reconstruction:* Pointclouds are continuously captured instead of discretely capturing them at the viewpoints. An efficient method to handle the pointclouds in a parallel thread is described in Section-3.5.6 that merges the pointclouds by accepting points that reduce the overall inaccuracy of the output cloud $\tilde{\mathscr{P}}$. A data structure of voxel grid $\mathscr{V}$ is used to process the incoming pointclouds filtered using the camera performance constraints.

The points in each $\mathscr{N}$ neighborhood of the voxels in $\mathscr{V}$ are then averaged to find the points in $\tilde{\mathscr{P}}$.

6. *Online Trajectory Refinement:* Section-3.5.7 describes this module which continuously examines regions with zero or low point density, also called anomalous regions on the part. The regions do not satisfy pointcloud density requirement. The module generates a refinement plan over such regions to capture more points.

Once the trajectory execution is complete the system returns the output $\tilde{\mathscr{P}}$. We will now discuss the 3D reconstruction algorithm and how the aforementioned components fall in place.

3.5.2 Camera Localization

The transform ${}^bT_c$ is needed to obtain points from the camera frame in the robot base or global frame. We can write the transform ${}^bT_c$ as ${}^bT_e *^e T_c$. ${}^bT_e$ is the robot end-effector transform with respect to the robot base which is determined by the forward kinematics. ${}^eT_c$ is initially estimated by the user based on the CAD model of the mount used to fix the camera to the end-effector. The estimate can also be obtained by measuring the location of the camera frame with a ruler

or caliper. The estimated transformation is inaccurate and is only used by our algorithm as an input to a multi-stage optimization process that finds the improved estimate of this transform, ${}^{e}T_{c}^{*}$. The following steps describe the multi-stage optimization process.

We move the robot to different camera poses around a rectangular glass plate and capture the surface pointclouds. The camera poses follow the acceptable range constraint from the camera performance model and points within the field of view are accepted. Outlier points are removed using depth-based filters and box filters and downsampled using a 5 mm voxel size.

All the captured pointclouds must coincide after transforming to the base frame since they belong to the same planar surface. However, due to the error in estimated ${}^{e}T_{c}$, the pointclouds do not coincide. Figure 3.5 illustrates how the pointclouds may look like in relation to a best-fit plane computed for all the points using the least square fit.

Every point in the clouds deviates from the best-fit plane with a value measured by the shortest distance from the plane. The average of the distances for all the points is taken as an error for the multi-stage optimization process. The algorithm aims at finding an optimal ${}^{e}T_{c}$ that minimizes the average deviation error. We will now discuss the multi-stage optimization algorithm.

1. *Decision Variable:* Initial transformation ${}^{e}T_{c}$ estimated by the user is converted to a 6 dimensional vector $\vec{x} =< x, y, z, r, p, y >$ where $< x, y, z >$ are the positional coordinates of the camera frame O_c origin and $< r, p, y >$ describe the Euler angles for unit X, Y, and Z vectors of O_c. The vector $\vec{x}$ is used as the decision variable.

2. *Stage 1:* A discrete optimization-based method is used as the first stage. The method uniformly samples around the vector $\vec{x}$ in discrete steps. The objective function used is the average over deviation distances of the points in the cloud from the best-fit plane.. The objective function is evaluated at these samples and a fixed number of good samples having lower values of the objective function are selected as initial guesses for the next stage of optimization.

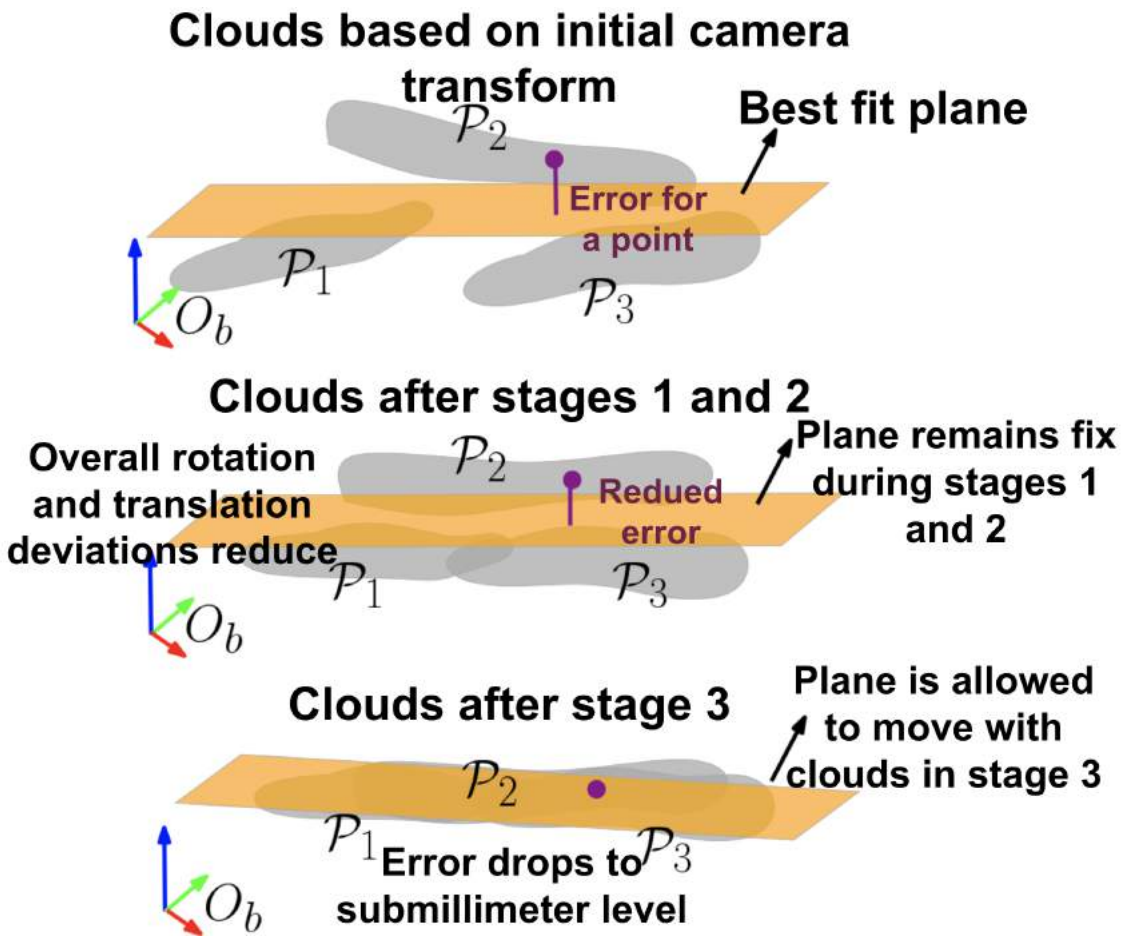

Figure 3.5: $\mathscr{P}_1$, $\mathscr{P}_2$, and $\mathscr{P}_3$ illustrate pointclouds collected by using user provided estimate ${}^bT_c'$. The clouds do not coincide in the beginning (top). Optimization stages 1 and 2 keep the plane fixed and reduce the deviations (center). Optimization stage 3 finds optimal ${}^bT_c^*$ and by allowing the plane to move (bottom).

3. *Stage 2:* The second stage uses a gradient descent algorithm over the selected good initial guesses from stage-1. The objective function is the same as the stage-1. Gradient descent is conducted using every initial guess and the corresponding optimal decision variables are found. The optimal decision variable corresponding to the lowest objective function value is chosen as the initial guess to stage-3 of the algorithm.

Figure 3.5 shows the output from Stages 1 and 2 of the algorithm. The clouds still do not coincide with the plane P as we kept it fixed during the stages. Stage-3 will now allow the plane P to move and the plane parameters will be a part of the optimization decision variable.

4. *Stage 3:* This stage extends the vector $\vec{x}$ by including the plane parameters and finds an optimal value $\vec{x}^*$ that minimizes the objective function. The objective function is same as the stage-1. Figure 3.5 shows output from the stage-3 of optimization. The best-fit plane now closely coincides with the pointclouds.

The output $\vec{x}^*$ is then is converted to homogeneous transform ${}^eT_c^*$. ${}^eT_c^*$ will allow all the pointclouds to be determined with respect to O_b such that the error between the clouds and the true plane is minimum. We use this transformation for getting the pointclouds during the 3D reconstruction process. For simplicity, we will denote ${}^eT_c^*$ as ${}^eT_c$. We will now discuss the 3d reconstruction algorithm that uses the optimal camera transformation and the determined camera performance model to generate robot trajectories and capture the pointclouds.

3.5.3 Camera Performance Model

An RGB-D camera captures an image using a 2D grid of pixels where each pixel will also have an associated depth value (referred to as the Z-depth in this work). The depth camera uses the intrinsic parameters to convert the 2D image into a 3D pointcloud that is represented in camera frame O_c, also the camera depth optical frame for Intel RealSense D415 camera used in our work. Camera intrinsic properties are a geometric transformation incorporating the optical center and focal length. The pointcloud is then converted to a global reference frame O_b using the transformation ${}^bT_c$, also called the camera extrinsic property. The conversions can be referred in Figure 3.6(right).

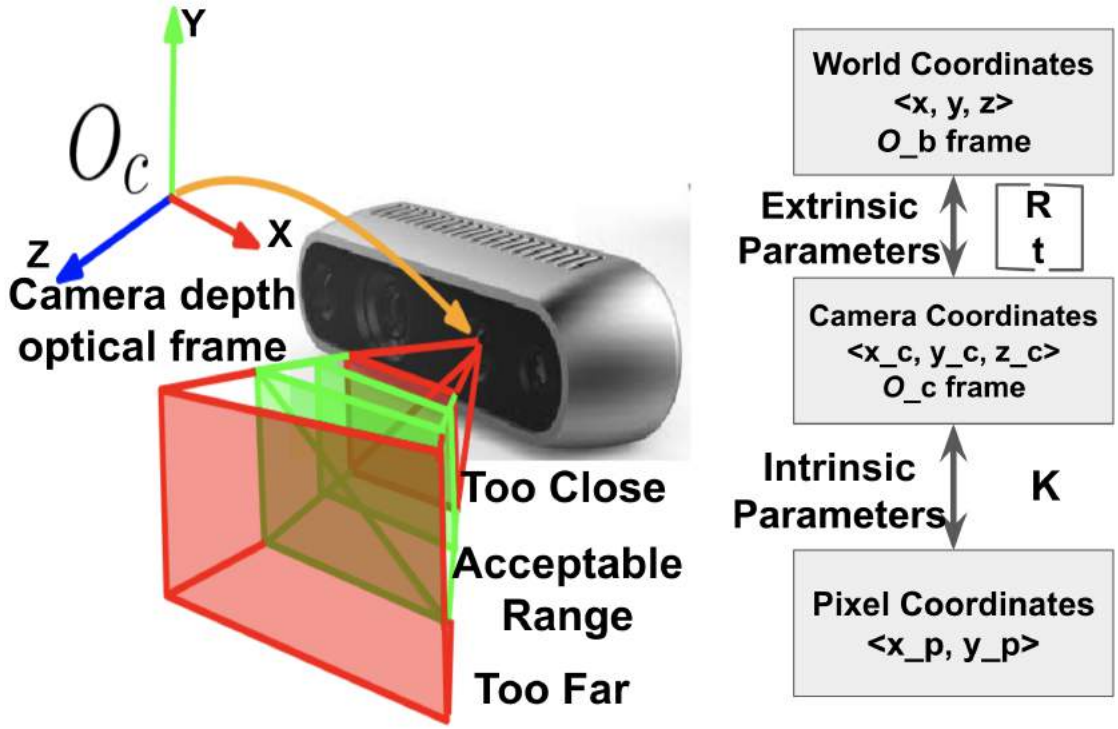

Figure 3.6: (Left) The field of view for the D415 depth camera is illustrated as a frustum. The Center region of the frustum is the acceptable distance of the surface from the camera frame. (Right) A block diagram showing the relationship between the conversion of coordinates between world, camera, and pixel frames. R and t are the rotation and translation elements that make up the homogeneous transform for extrinsic parameters. K is the matrix representing intrinsic parameters.

All the points obtained in the global reference frame do not have uniform accuracy. Larger distances of the points on the surface from O_c lead to high errors. The points are out of focus when they get too close. Refer to Figure 3.6(left) that illustrates the field of view of the camera. There is an acceptable range which we set within which the points can be successfully captured by the camera and the error is low. It is important to keep the observable points within the acceptable range from the camera frame while capturing the pointclouds.

The camera performance model in our method captures three parameters, α, β, and γ to characterize the camera performance. Figure 3.7 illustrates a point p viewed by the camera. α is the Z-depth or normal distance of the point from O_c. β is the radial distance of p from O_c that defines how far out the point is from the camera frame within the field of view. The point p also

has an associated normal vector $\vec{n}$ of the underlying surface. The angle γ is made by this normal and the direction Z-axis of the camera frame O_c.

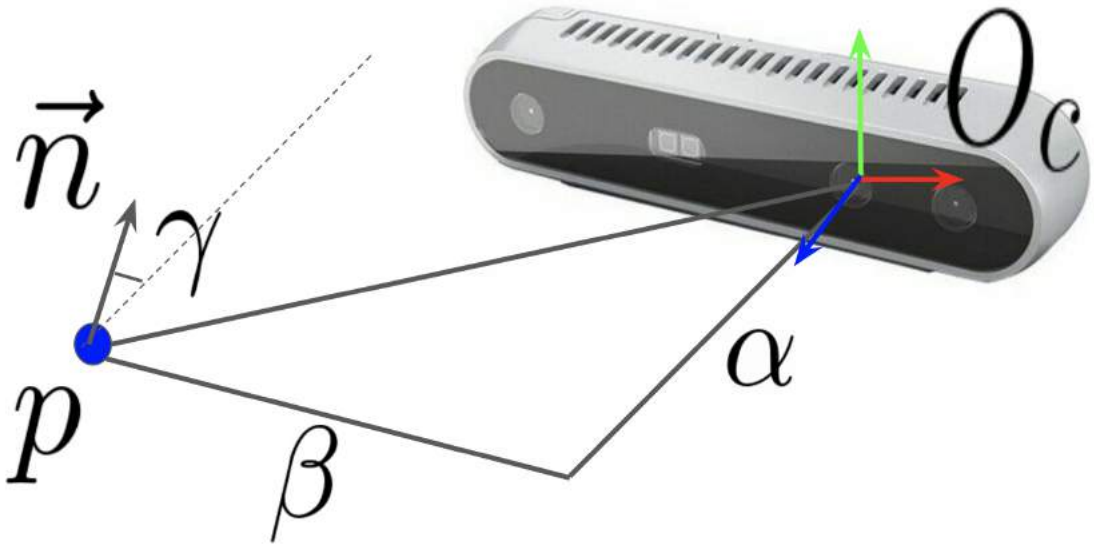

Figure 3.7: The parameters α, β, and γ for an observed point p in the world with respect to the camera frame O_c is illustrated. $\vec{n}$ is the normal vector that corresponds to p.

$\alpha, \beta,$ and γ govern whether the observed point will be labeled as acceptable or unacceptable data. Let the error of the point captured p with respect to the true point p^* be computed as the second-order norm $||p^* - p||_2$. The observed points from the camera have a distribution associated with the error. We represent the variance of the distribution by σ. As the point moves away from or gets too close to the camera frame, the σ increases that are given by the σ vs. α profile. Points are not recorded if the value of α decreases below a threshold since the observable world goes out of view. The σ vs. β profile emphasizes the increase in σ as the point moves radially outward from the camera centerline extended along the depth direction. Finally, the parameter γ captures whether the camera is normal to a point on the surface or it is oriented at a very high angle that increases the σ.

Determining the camera performance model involves finding the admittance function $\eta(\cdot)$. Many non-linear complex relationships can exist for a sensor and a more sophisticated function will

lead to better output. In this work, we empirically determine the lower and upper on acceptable α, β, and γ that make a linear admittance function η. The empirical relationship is found using the following steps.

1. *Planar surface reference:* A high fidelity pointcloud of a plane surface is obtained using the Hexagon Romer arm with an integrated laser scanner used in our work as a reference metrology instrument.

2. *Depth camera scan:* The planar surface is then scanned by the depth camera from various camera viewpoints. Values of α, β, and γ are computed for each observed point p_i on the planar surface.

3. *Error computation:* The error for p_i is computed as deviation from the true point p_i^t using second order norm $||p_i^t - p_i||_2$. Variance of the error is obtained by using error values obtained from multiple measurements taken for the point p_i.

4. *Determining bounds:* The lower and upper limits on α, β, and γ are selected by using user-defined threshold on the computed variance value.

Figure 3.8 shows a 3D plot with values of α, β, and γ plotted on the axes with units meters, meters, and radians respectively. A colormap is overlaid on the cuboid representing the values of a variance σ. σ is computed for all the deviation errors for all the observed points captured in small elements within the plot. We can observe that for a α value of less than 0.25, we do not obtain any valid point since the surface goes out of the camera field of view. Figure 3.8 also shows three other 2D plots where the average error for each element is plotted. For each plot, two of the three parameters are kept constant and the third parameter is allowed to vary. The values of the constant parameters are shown in the legend. We can notice that the aforementioned increasing trend of error with respect to increasing values of the parameters holds for all the cases. We will now define all the constraints that will be applied for camera path planning and robot trajectory planning based on the aforementioned discussion.

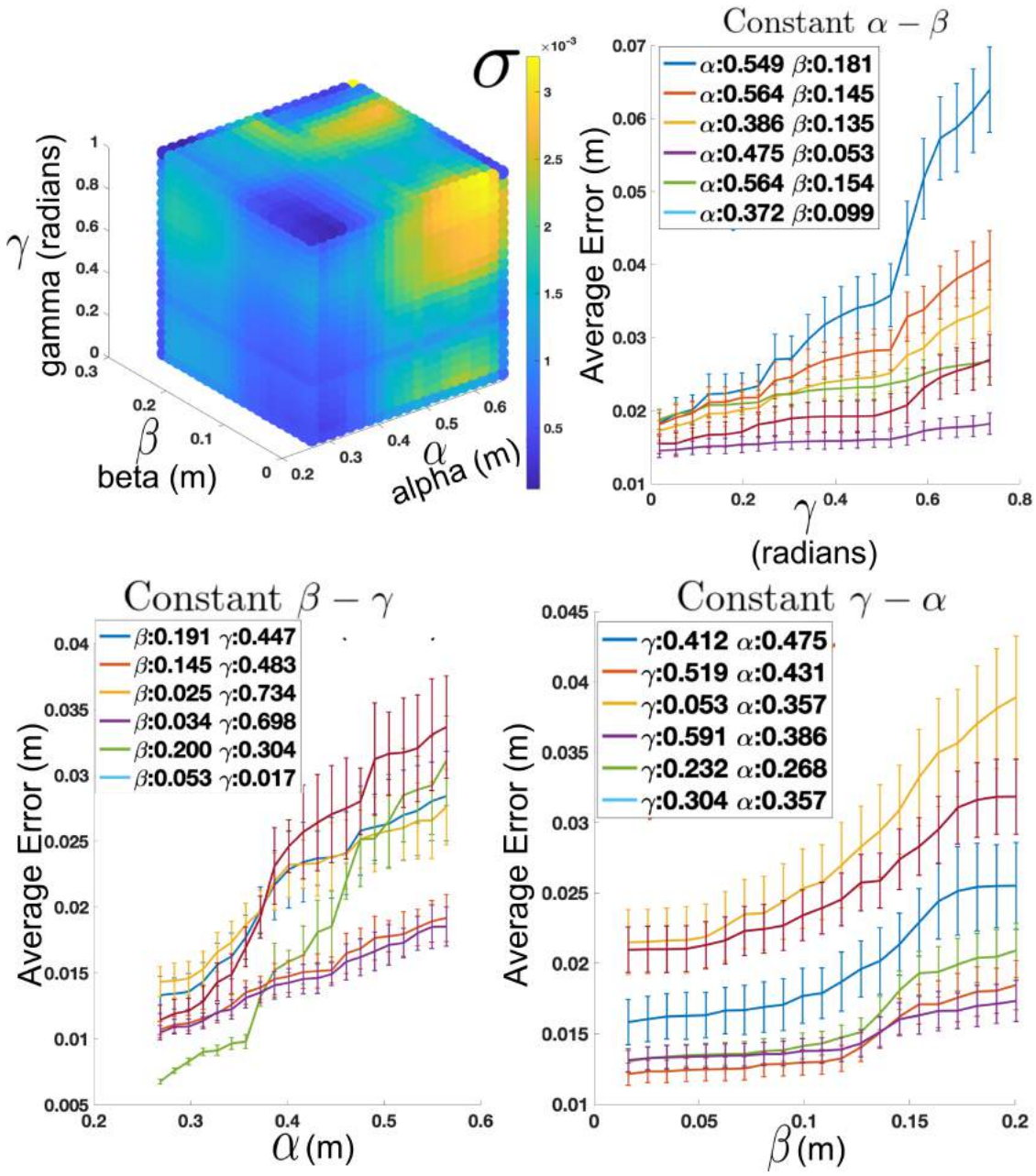

Figure 3.8: (Top, Left) 3D colormap cuboid showing the variance for every small element $\Delta\alpha, \Delta\beta$, and $\Delta\gamma$ within the range of values observed in the dataset produced from the planar surface. (Top, Right) Profile for deviation error with respect to γ for a few sampled constant α and β. (Bottom, Left) Profile for deviation error with respect to α for a few sampled constant β and γ. (Bottom, Right) Profile for deviation error with respect to β for a few sampled constant γ and α.

1. *Acceptable Range Constraint:* The parameters of a camera view point I from the observed point p located in the world need to be constrained with the limits defined over $\alpha, \beta,$ and γ.

$$\alpha_{lb} \leq \alpha \leq \alpha_{ub} \tag{3.4}$$

$$\beta_{lb} \leq \beta \leq \beta_{ub}$$

$$\gamma_{lb} \leq \gamma \leq \gamma_{ub}$$

where lb and ub are the lower and upper bounds on the parameters. We use only the range constraint given by equation-3.4 for planning purposes to make sure that the camera viewpoints are generated within the camera field of view. Other constraints can also be incorporated for a more sophisticated plan.

2. *Performance Constraint:* The lower and upper bounds on the three camera parameters are given by the point admittance function $\eta(p, \mathscr{M})$. $\eta(p, \mathscr{M})$ makes a binary decision of whether to accept or reject the point. The performance constraint is then defined as:

$$\eta(p, \mathscr{M}) = True \tag{3.5}$$

The η function used in our work applies the following thresholds on the three parameters to filter the points from original pointcloud. $\alpha : [0.26, 0.4], \beta : [0, 0.2], \gamma : [0, 1.0]$.

3. *Diversity:* Each point has an associated camera frame from which it was captured. The diversity value is then calculated using equation 3.6. The new point will be rejected if the value is less than a threshold δ.

$$|| < \alpha_i, \beta_i, \gamma_i > - < \alpha, \beta, \gamma > ||_2 \leq \delta, \forall i \in [1, k] \tag{3.6}$$

where, k is the number of points currently present in the $\mathcal{N}$-neighborhood and $< \alpha, \beta, \gamma >$ is the vector for camera parameters of the new incoming point.

3.5.4 3D Reconstruction Algorithm

3.5.4.1 Camera Path Generation

Feasible state space of viewpoints is constructed around the part and then a greedy planning method is used to find the camera path.

Camera path needs to be designed such that the optical frame follows the part surface geometry. The optical frame also needs to be within the acceptable distance from the surface. Hence, we can use the surface voxels in $\mathscr{V}$ and shift them outward by the acceptable distance to construct the state space. The method first downsamples $\mathscr{V}$ from the resolution of 0.5 mm to 40 mm to generate a new voxel representation $\mathscr{V}_c$. The coarser resolution controls the viewpoints in state-space since a large number of viewpoints are unnecessary and increase computation cost. The coarse resolution is empirically selected based on the number of viewpoints required to cover the part surfaces, camera field of view, and part geometries used in our test cases. The Center of each voxel in $\mathscr{V}_c$ represents the position of viewpoints in the state-space ($\mathcal{S}$) and also has a direction normal associated with it.

The viewpoint positions are then shifted outward along their direction normals at the acceptable distance sampled within the bounds α_{lb} and α_{ub}. We choose a range constraint of 35 cm for our state space. The camera needs to view the surface as close to the surface normal for capturing low error points. Orientation of each state is therefore assigned by using the viewpoint normal. Orientation vectors are uniformly sampled within a 3D cone about the normal direction of a voxel and each sample is added to the state space. The 3D cone allows some flexibility about the direction normal so motion plans can be easily generated without over constraining the camera to

align with the normal. Figure 3.9 illustrates the state space that is generated for a part. Line-1 of algorithm-2 specifies the state-space generation step.

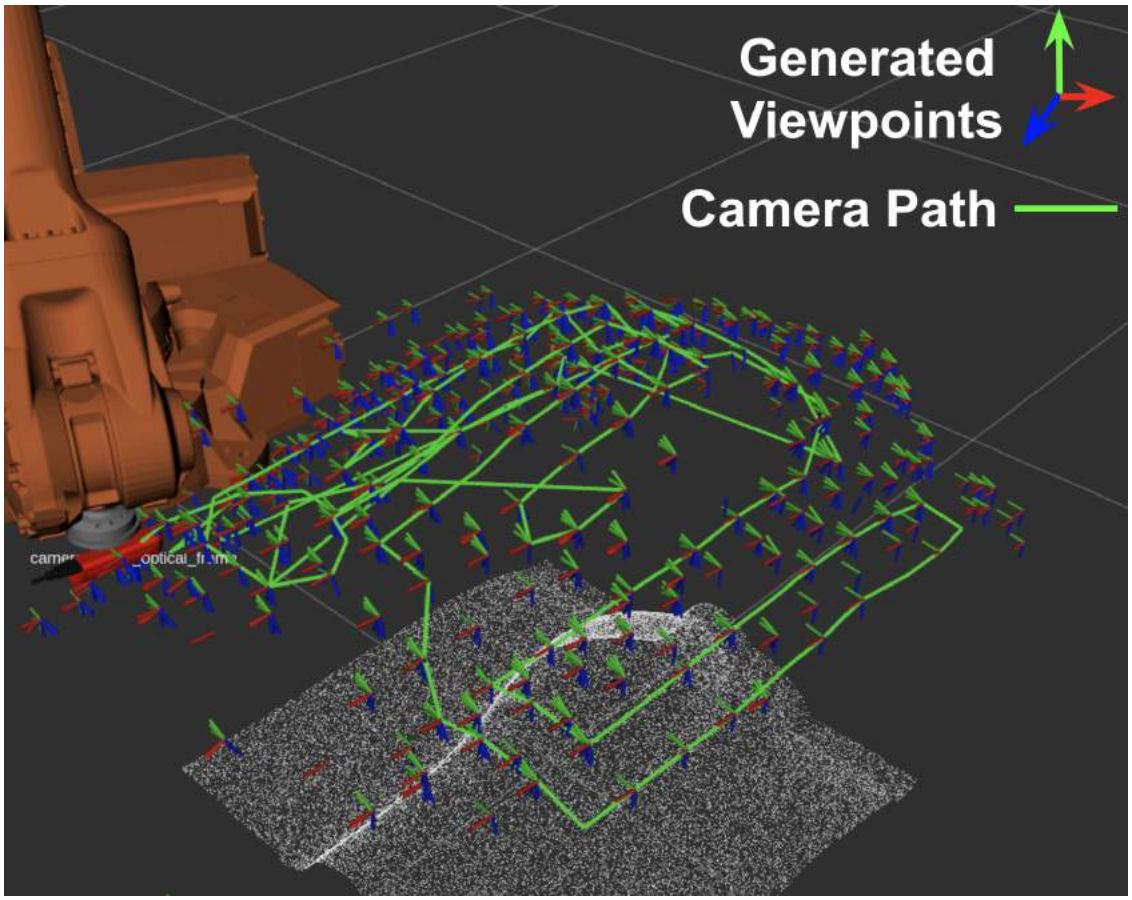

Figure 3.9: The state space of feasible camera viewpoints are shown around a part. A camera path passing through a set of viewpoints is executed by the robot to capture the pointclouds across the surface.

3.5.4.2 Path Generation

A state-space search algorithm then finds the camera path that passes through a minimum set of viewpoints generated in the state space. The algorithm takes additional inputs as the robot model $\mathscr{R}$, a simulated camera model $\mathscr{C}_s$, voxel grid $\mathscr{V}$, and camera performance model $\mathscr{M}$ for path generation. The camera path generation is conducted in the following steps.

Algorithm 2 CameraPathGeneration($\mathscr{R}$, $\mathscr{V}$, $\mathscr{M}$, $\mathscr{C}_s$)

1: $state\_space \leftarrow$ generate_state_space($\mathscr{V}$)
2: $kdtree \leftarrow$ generate_KDTree($state\_space$)
3: $path$.append($kdtree$.query(current_camera_pose($\mathscr{R}$), $K = 1$))
4: **while** !stopping_condition **do**
5: $children \leftarrow kdtree$.query(current_camera_pose($\mathscr{R}$), $K = 20$)
6: $parent \leftarrow path[-1]$ $rewards \leftarrow [\,]$
7: **for** *child* in *children* **do**
8: $rewards$.append($\mathscr{V}$.simulate_update($parent, child, \mathscr{M}, \mathscr{C}_s$))
9: **end for**
10: **if** all($rewards == 0$) **then**
11: expand(K)
12: **if** $K >$ $state\_space$.size() **then**
13: **return** *path* #Incomplete solution
14: **end if**
15: repeat steps 8-10
16: **else**
17: $path$.append($children[rewards$.argmax()])
18: $\mathscr{V}$.update($parent, children[rewards$.argmax()], $\mathscr{M}, \mathscr{C}_s$)
19: **end if**
20: **if** $\mathscr{V}$.coverage_reached() **then**
21: stopping_condition $\leftarrow$ True
22: **end if**
23: **end while**
24: **return** *path*

1. *KDTree Generation:* A KDTree data-structure is initialized (line-2 of algorithm-2) over all the viewpoints in $\mathscr{S}$. The start state of the camera path P is the nearest neighbor to the current camera pose or a user-specified pose. The start state is appended to the path.

2. *Selection of child nodes:* A fixed K number of viewpoints from $\mathscr{S}$ are queried from the KDTree for the last state in P as the parent node. The viewpoints are added to a list of child nodes for the next step. Line-5 of algorithm-2 specifies this step.

3. *Camera segment generation:* A Cartesian segment from parent node to each child node is then generated under the range constraint described by equation-3.4. Position of the camera frame while transitioning from the parent node to a child node along the Cartesian segment is maintained within the acceptable range. All the Cartesian segments formed between the parent

and each child node are then evaluated using a camera simulation.

4. *Voxel data-structure update:* A camera model $\mathscr{C}_s$ is used to simulate how pointclouds will be captured in the real world while moving the camera along each Cartesian segment computed in the previous step. The simulated pointcloud is filtered using the point admittance function η. The pointcloud is then admitted into a simulated copy of the voxelized data structure $\mathscr{V}$.

5. *Reward computation:* A point from the filtered pointcloud is considered to be useful towards the reward computation for a Cartesian segment if it helps increase the number of points in a neighborhood $\mathscr{N}_i$ for $v_i \in \mathscr{V}$ and gets it closer to N. Given the number of new points that are considered useful from the simulated pointcloud and the total number of points in $\mathscr{V}$, the reward is computed using the following equation.

$$r = (\text{new\_points} \ / \ \text{total\_points})/\text{segment\_length} \tag{3.7}$$

The reward is higher for a parent-to-child segment if a large number of high quality and diverse points are added to $\mathscr{V}$ to meet our objective with a shorter travel distance in the Cartesian space.

6. *Selection of highest reward child:* The reward is computed for all segments generated from the parent to each child in the list of child nodes and the child which gives the maximum reward is chosen as the next viewpoint along the path along with the corresponding Cartesian segment. Lines 6-19 of algorithm-2 encompass steps 3-6 of the path generation process.

We will now discuss what the stopping conditions for the iterations of adding child nodes to the path are and when the algorithm returns an incomplete solution.

3.5.4.3 Stopping Conditions

1. *Incomplete solution:* The neighborhood size we use in our approach for selecting child nodes is $K = 20$ which we obtained by manually tuning the method for low computation speed and path lengths. There are situations where no reward can be obtained while transitioning from a parent to possible child nodes (lines 10-15 of algorithm-2). The neighborhood K is then increased and more number of child nodes are evaluated (line-11 of algorithm-2). The neighborhood keeps expanding until the entire state space is exhausted. The algorithm will then return an incomplete solution. Such a situation will arise if state space is inadequately sampled and regions of the part are completely unvisited. We generate a state space that is adequate to find a complete solution.

2. *Complete solution:* A complete coverage is obtained if all the $\mathscr{N}$ neighborhoods for every voxel in $\mathscr{V}$ have contained N valid points. The algorithm then returns the path found so far.

3.5.5 Robot Trajectory Generation

A robot configuration path $\tau = \{q_1, q_2, \ldots, q_m\}$ needs to be found in the configuration space ($\mathscr{C}$) of the robot corresponding to the camera path $P = \{I_1, I_2, \ldots, I_m\}$. The configuration space path is then converted to a trajectory $\tau(t)$, where t is the time. The computation efficiency and success rate of finding a trajectory reduces by enforcing all the constraints at once. Several evaluations of the constraint function are required to compute the gradient and the optimization algorithms are susceptible to local minima. We, therefore, use a hierarchical approach to this trajectory generation problem to improve the computational efficiency of generating a solution. The stages in robot trajectory generation are as follows.

3.5.5.1 Reachability check

Checking the reachability of the robot first before checking for meeting other constraints can save computation costs since constraints will never be met if the robot is unreachable. An inverse kinematics (IK) filter is applied during camera path generation we discussed earlier in Section-3.5.4.1. The robot IK filter provides a computationally fast reachability check before camera motion is simulated from parent to child. Given that the robot configuration at parent pose is q_{i-1}, the validity of configuration for a candidate child pose denoted by w is checked by using the logic in equation-6.3.

$$q_i \text{ is valid if } \underline{q} \leq q_i \leq \overline{q} \tag{3.8}$$

$$q_i \leftarrow \arg\min_{q} PE(q_{i-1}, w, \mathscr{R})$$

$$PE = || < x_p, y_p, z_p > - < x_w, y_w, z_w > ||_2 +$$

$$\sum_{i=\{bx,by,bz\}} (1 - p_i * w_i')$$

$$\text{where } p = FK(q_i)$$

where $FK(\cdot)$ is a forward kinematics function for the robot, $< x_p, y_p, z_p >$ and $< x_w, y_w, z_w >$ are the 3D position only vectors for from poses p and w respectively, and bx, by, bz are the unit X, Y, and Z direction vectors of the frames corresponding to the poses p and w. The pose error PE essentially minimizes the position and orientation difference between the robot's current frame p and the target frame w. The robot is reachable at the target w if a solution is found to the aforementioned minimization problem and is within the lower and upper joint limits $\underline{q}$ and $\overline{q}$ respectively. The camera viewpoint is discarded if the robot is unreachable and the child node is not considered for planning of camera path.

3.5.5.2 Sequential $\mathscr{C}$-space path generation

A sequential IK solver then generates a sequence of robot configurations by using an iterative optimization process given by the equation-3.9. The solver finds robot $\mathscr{C}$-space path τ by introducing the path consistency constraint defined by equation-3.10 and collision constraint given by equation-3.11 to generate a continuous path.

$$\tau = \{q_i | q_i \leftarrow \arg\min_{q} PE(q_{i-1}, I_i, \mathscr{R}), \forall i \in [1, m]\} \tag{3.9}$$

$$||q_i - q_{i-1}||_\infty \leq \Delta\theta \tag{3.10}$$

$$in\_collision(q_i) = false \tag{3.11}$$

The path consistency constraint enforces that the two consecutive robot configurations q_{i-1} and q_i do not have an absolute maximum joint angle change of more than $\Delta\theta$ radians given by the infinity norm. This ensures that the path is executed continuously without dramatic robot joint angle changes. The collision check between all the bodies in the environment is done using built-in collision checking the functionality of MoveIt [36].

3.5.5.3 $\mathscr{C}$-space trajectory generation

$\mathscr{C}$-space path generated in the previous step needs to be time parameterized for the robot controller to execute it. Additional constraints like velocity and singularity are introduced in this step and the sequential problem given by equation-3.9 is resolved to find a trajectory $\tau(t)$.

1. *Velocity:* Velocity constraint for i^{th} configuration q_i along the path is given by the robot Jacobian at previous configuration q_{i-1}. The robot Jacobian defines the relationship between robot joint velocity $\dot{q}$ and the Cartesian velocity of the tool attached to the robot. The tool frame in our case is the camera frame $\dot{I}$ and the velocity is represented as $\dot{I} = J(q) * \dot{q}$. For a small-time

change Δt, the relation is expressed as $\Delta I = J(q) * \Delta q$. The velocity constraint is then given by the following equation.

$$\underline{\dot{q}} \leq \text{inv}(J(q_{i-1})) * (I_i - I_{i-1}) \leq \overline{\dot{q}} \tag{3.12}$$

where $\underline{\dot{q}}$ and $\overline{\dot{q}}$ are the robot lower and upper velocity limits, inv() is a linear algebraic inverse function, and I_i and I_{i-1} are the i^{th} and $(i-1)^{th}$ camera viewpoint in the sequential path P.

2. *Singularity:* Robot singularity constraint is also given by the robot Jacobian for a given configuration q_i as:

$$\det(J(q_i) * J(q_i)^T) \geq \omega \tag{3.13}$$

where det() is the determinant of the queried matrix J^T is the transpose of the Jacobian matrix. ω is a constant. Higher values of ω ensure good robot manipulability at a camera viewpoint which makes it feasible to generate online trajectory refinement plans to improve point density.

3.5.5.4 Point-to-Point trajectory generation

The trajectory found using the sequential solver is used to capture pointclouds. Other trajectories are needed for moving the robot from one point to the other in free space without tracing Cartesian paths. We use a sampling-based planner from MoveIt [36] for such point-to-point planning. More sophisticated approaches have been studied and can be used where the air-time or the time between transitioning from one point to the other is minimized for further improving the overall execution time [197, 164, 163].

3.5.6 Pointcloud Construction

The pointclouds are captured continuously in a parallel thread as the robot executes the trajectory $\tau(t)$. We developed an efficient method to handle the massive number of pointclouds. The

method can deal with a finer resolution to cover more intricate details of the part geometries and also saves a larger number of points in the cloud before generating the output improving our previous approach in [125, 126]. The pointcloud processing method is used both for the simulation environment used by the camera path generation and for the real-time execution. The difference is that the simulation environment used a camera model $\mathscr{C}_s$ whereas the pointclouds in real time are captured using the real RGB-D sensor. We will now discuss the elements of the pointcloud construction method.

3.5.6.1 Pointcloud filtering

A pointcloud captured from the camera is first transformed to a robot base frame using the corresponding ${}^bT_c$. All the outliers are removed and the constraint-based on point admittance function η given by equation-3.5 is applied to eliminate high error points. The resulting pointcloud is then mapped to $\mathscr{V}$.

3.5.6.2 Point mapping

Each point in the incoming cloud is checked if it belongs to $\mathscr{V}$ and the points that do not belong to any voxel are removed. More points are admitted during the scan in the same way. The voxels are processed to the output pointcloud by using an *update rule* in our method.

3.5.6.3 Update rule

The surface voxels in $\mathscr{V}$ are identified using the intersection of voxels with the rays projected from viewpoints in the camera path P. The first voxel to intersect the ray is a candidate surface voxel. If the candidate surface voxel has neighbors greater than a threshold number then it is labeled as a valid surface voxel in $\mathscr{V}$. Invalid candidate voxel is disregarded and the next intersecting voxel is considered for evaluation. The method allows us to disregard but not eliminate the voxels that may be filled out with noisy data and consider the voxels that have a high density of neighbors.

Every surface voxel v_i^s in the original grid $\mathscr{V}$ also has a direction normal associated with it. A cylindrical geometry is generated for each v_i^s with the principal axis (longer dimension) along the surface normal. The center of each cylinder coincides with the center of v_i^s. The principal axis aims at capturing points that would be spread along the normal due to the Z-depth uncertainty of the camera. The radius of the cylinder aims at capturing points spread due to the lateral uncertainty of the camera. We choose a radius of 1.5 mm in our work and a principal axis of 20 mm.

All the points that fall within this cylinder (may belong to other voxels) are considered as the $\mathscr{N}$-neighborhood of voxel v_i^s. These are the points that we use to determine whether N diverse points have been attained for the voxel v_i^s. The $\mathscr{N}$-neighborhood points are then checked for outliers and any point with a deviation of more than $2 * \sigma$ is rejected. The remaining points are averaged and become the new location of voxel v_i^s. The voxel v_i^s is considered as low-density voxel if the points within the $\mathscr{N}$-neighborhood are less than N.

The update-rule is applied to $\mathscr{V}$ after a threshold total number of points within $\mathscr{V}$ is reached. The update rule also helps with saving memory as points are averaged and reduced to a single point. Successive iterations of the update-rule gradually allow the surface voxels to converge to the true surface of the part.

3.5.7 Online Trajectory Refinement

The observations we accept during the scanning process correspond to the conservative bounds expressed by the camera performance constraint. The camera path is generated such that all points captured by the simulated camera are within the bounds and the part is fully covered. However, complete coverage may not be possible in reality due to simplifications in the camera model or external environmental factors. There may exist regions with zero or low point density as the $\mathscr{N}$-neighborhood for surface voxels in $\mathscr{V}$ did not reach the desired minimum number of points N.

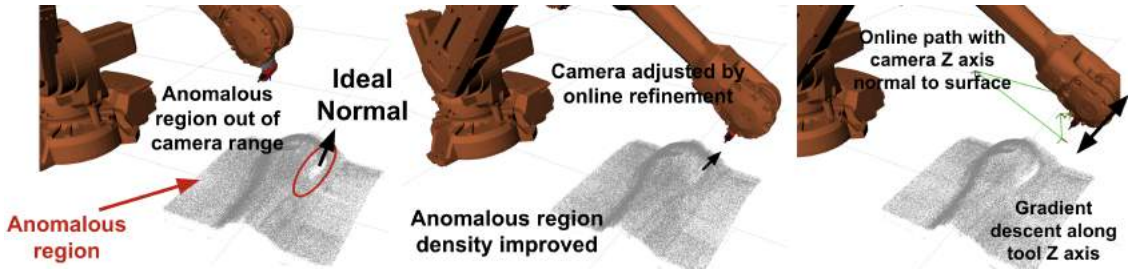

Figure 3.10: The camera was not oriented normal to the ideal surface normal of the double curvature region leading to zero or low point density, also called anomalous regions (left). The point density of anomalous regions can be improved by aligning the camera with the ideal normal (center). An example online camera path generated using the refinement module to increase point density is shown (right).

It is necessary to pay particular attention to such anomalous regions and orient the camera with respect to surface normals of the anomalous region to improve the density of the points. Figure 3.10(left) illustrates an example where pointcloud obtained by the robot during the execution of camera path had anomalous regions with low or zero pointcloud density. The region was situated where the part geometry had double curvature and high error points were being captured by the camera which was rejected by the camera performance constraint.

Figure 3.10(center) shows how aligning the camera with the normal vector of an anomalous region can help to increase the point density since the robot slows down and pays particular attention to such regions. The online trajectory refinement module in our system is responsible for such online corrections in the robot trajectory. We will now discuss the sequence of steps that go into the online trajectory refinement module.

3.5.7.1 Identification of unobserved clusters

The grid $\mathscr{V}$ is used to identify the unobserved regions on the part during the scanning process. We first identify the voxels in $\mathscr{V}$ that have low or zero density of points. A custom clustering algorithm is then used to cluster the regions such that the minimum number of voxels in the cluster is greater than a threshold. This allows us to ignore several smaller regions on the part as

noise since the voxel size used is 0.5 mm and many clusters are identified and may not necessarily be an unobserved region. The unobserved regions are used to identify anomalous regions.

3.5.7.2 Identification of anomalous regions

The simulated camera is used to determine what regions will be covered by the remainder of the camera path that has not been executed. A cluster will be an anomalous region if it does not fall under the simulated camera coverage. The anomalous regions will not be visited by the robot in the offline plan and are holes in the part model. The robot is then commanded to stop the execution of the trajectory and first execute the online trajectory to improve the anomalous region point density before resuming the offline trajectory.

3.5.7.3 Online trajectory generation

The generation of an online trajectory to fill the anomalous regions is done in two steps.

1. *Viewpoint generation:* The anomalous region is discretized into smaller regions using a 3D surface grid. Each element on the grid has a direction normal vector associated with it. Camera frames that are oriented with the normal vector are generated such that they obey the range constraint (equation-3.4) and the robot reachability constraint (equation-6.3). The set of all such camera frames for every grid element across the grid are the viewpoints for the online planner.

2. *Optimization based execution:* Robot is then moved from the location where it was stopped to one of the generated viewpoints. The motion allows the camera to be oriented normal to the small grid element of the anomalous region improving the density of points for that region. A gradient descent-based motion primitive is also used based on a predefined set of actions if the density does not improve. For instance, the camera pose is changed along the Z-depth axis in an increasing direction until new points are obtained. If no information is added, the motion is

reversed to the decreasing direction. Similar motions are executed along the camera tilting axis as well.

The stopping condition used during online execution is based on a time-out. The online execution is terminated if the point density can not be restored in the concerned region.

3.6 Results

3.6.1 Test Cases

Six parts A, B, C, D, E, and F as shown in Figure 6.9 are used as test cases in this paper. The part geometries have single and double curvatures, convex and concave contours, corners and cavities, and different reflectivity properties due to wood, aluminum, and plastic materials used to manufacture them. The variation makes it a challenging case study to benchmark the performance of our method. The length, width, and height of parts A, B, C, D, E, and F are 380 x 280 x 80 mm, 500 x 450 x 200 mm, 300 x 600 x 300 mm, 570 x 830 x 95 mm, 480 x 470 x 480 mm, and 400 x 1150 x 90 mm respectively. The dimensions cover medium-large industrial parts and the robot is reachable over all the parts and can successfully move the camera to cover their surfaces.

3.6.2 Algorithm Performance

3.6.2.1 Reconstruction Performance

Baseline approaches are used to provide insights into the importance of using the four main components described in this paper: range constraint during planning, camera performance model constraint, minimum number of observations (N) and diversity constraint, and online plan refinement methods.

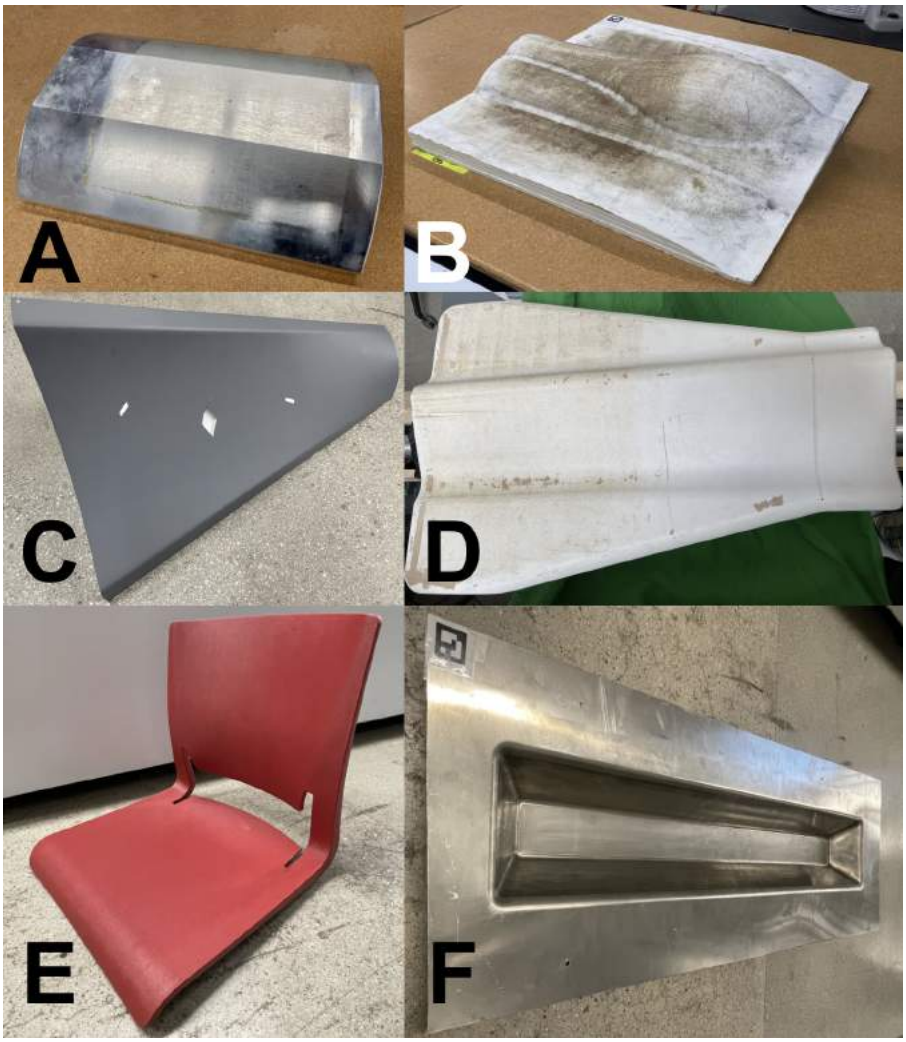

Figure 3.11: Parts A, B, C, D, E, and F that are used in our work. The parts are fabricated with complex geometries and different surface properties to provide insights into the performance of our method.

Baseline-1 does not use any of the aforementioned components and the camera viewpoints are generated around the part without the range constraint. A camera path is then generated using the same planner as we discussed in Section-3.9. The simulated camera model for the planner does not truncate points based on the point admittance function η and all the points are used to fill the $\mathcal{N}$-neighborhood. The method aims at collecting raw camera data during the scan and

then post-processing it to create the output pointcloud. The collected points will be a mix of low and high error points.

The baseline-2 method generates the state space using the range constraint. All the viewpoints are within the Z-depth distance specified by lower and upper limits of α from the part. Camera performance and diversity constraint are not applied and therefore accepting all the camera points without truncating them using η. The method collects raw pointcloud data during the scan, similar to baseline-1. Although, in this case, the camera distance from the surface is maintained within the acceptable range. The collected data therefore should have lower error compared to baseline-1. However, this would still have a mix of high and low error points.

The third set of pointclouds is generated using all the constraints but not using the online refinement to fill the anomalous regions. The approach imposes the range, camera performance, and diversity constraints while generating the camera path and capturing the pointclouds continuously during execution. We label this approach as *offline only*. The output pointcloud will only have low error points since *eta* function eliminates high error points. The overall quality of the data will also be better than baseline-1 and 2 since diverse measurements are taken for each point on the surface. The offline-only approach also provides insights into how zero or low-density regions can leave holes in the output while only using offline planning.

Lastly, the fourth set of pointclouds uses both offline and online refinement. The approach imposes all the constraints and also obtains points for anomalous regions online to guarantee the pointcloud density is consistent throughout the scan. We label this method as *offline + online*.

Figure 3.12 shows a comparison between all the aforementioned methods using a point deviation colormap for all the parts. The output pointcloud $\tilde{\mathscr{P}}$ is overlaid on the reference pointcloud $\mathscr{P}$ using iterative closest point (ICP) algorithm and point to point correspondence is found. Each point in $\tilde{\mathscr{P}}$ is then assigned a color based on the deviation error from the reference cloud. The colormap demonstrates the overall accuracy of the surface scan.

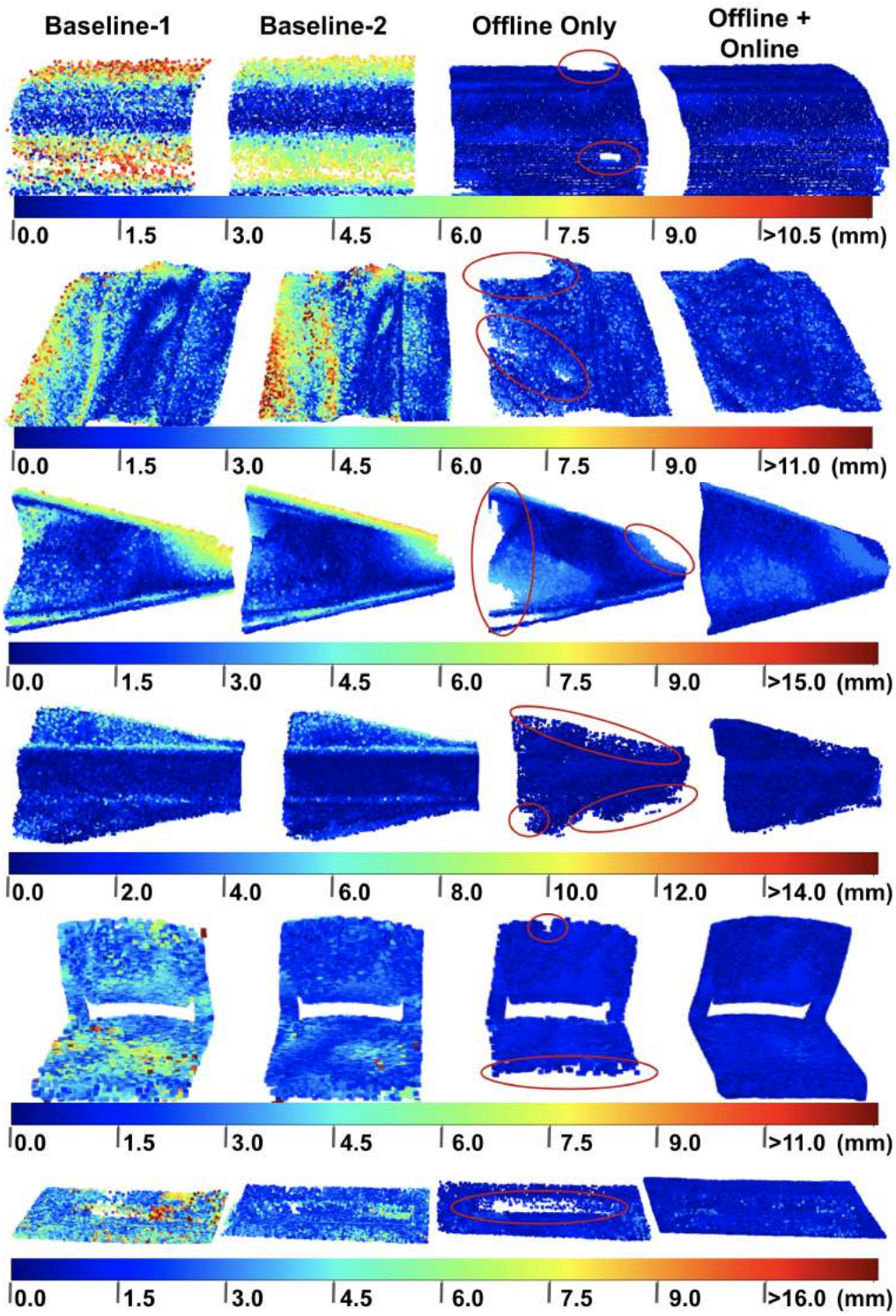

Figure 3.12: Point deviation error plotted as a colormap across the surface of all the test parts used in our work. The deviation errors are computed in mm. Baseline approaches 1 and 2, offline only execution, and offline + online execution approaches are used to present the colormaps. The circled contours on the pointclouds generated using offline only method indicate zero or low density point regions.

We can observe that the errors are higher for baselines 1 and 2 since we do not use range, camera performance, and diversity constraints during planning and execution. A good post-processing technique will not be able to generate high-quality output construction since the raw data collected contains high error points. Baseline-1 and 2 both show the behavior since η function is not used during planning and diversity in data collection is not enforced either. Deviation errors using baseline-2 are lower than deviation errors using baseline-1 since range constraint is enforced in baseline-2 providing slightly better quality data. Significant improvements are further obtained in the offline approach as we enforce all the constraints. The offline-only approach, however, leaves anomalous regions in the pointcloud since anomalous regions are not identified and rescanned online. Offline and online components together are required to generate a complete scan of the surface filling the anomalous regions and also marginally improving accuracy due to the addition of more high-quality points.

We also summarize the values from the point deviation map in the Table 3.1 for all the parts. The average and maximum error across all the points for a part are computed and presented in millimeters. We can observe similar improvements in the error values using our approach (offline + online) as compared to baseline-1 and baseline-2.

High average and maximum errors are generated for parts manufactured using reflective material. Parts A and E are examples of such materials. Errors also increased in the regions having double curvature or cavities. Double curvature offers challenges to the planner to align the camera very close to the surface normals since a good estimation of the surface normals is not available during the scan. The poor surface normal estimation is due to localization and manufacturing errors that create uncertainty in the location of the part. A good quality construction is obtained at single curvature and mildly curved convex regions. We expect the observations to extend to a wide range of similar geometries as our test cases cover commonly used industrial parts.

	Baseline-1		Baseline-2		Our Approach	
Part	Avg	Max	Avg	Max	Avg	Max
A	7.2	13.6	4.8	11.6	1.4	2.3
B	5.2	12.9	3.3	8.6	1.5	3.3
C	6.5	15.6	4.4	12.5	1.9	2.9
D	4.8	12.7	3.8	10.4	1.7	2.8
E	5.6	13.2	4.1	13.6	1.4	3.8
F	6.4	18.3	5.7	12.8	2.1	4.3

Table 3.1: Average and maximum values of point deviation error in millimeters computed using the points across the entire surface for all the test parts used in our work. The baseline methods and our approach (offline + online planning and execution).

3.6.2.2 Planning Performance

An alternate way to guarantee a high density of points across the part is to first scan the part using an offline plan, then identify the anomalous regions, and lastly, generate another offline plan to scan the anomalous region. The process is repeated iteratively until the density is improved across the entire part. The insights into the effectiveness of using an online approach over the iterative offline approach are provided by comparing against the iterative offline + replanning method as a baseline. Three iterations of replanning paths over anomalous regions were sufficient to cover the entire part and produce the same quality of output pointcloud $\mathscr{P}$. Part B was used for obtaining the Table 3.2 that provides the lengths of the paths (used as a surrogate for execution time) for offline + replan offline execution, and the offline + online execution.

We can observe that the path length for offline + replan is higher than offline + online due to the execution of multiple passes required to acquire the desired pointcloud. The path length of offline + online is lower than offline + replan, but higher than offline only. Although, the offline-only approach is not acceptable since regions with zero or low density of points are present.

Part	Offline + Replan	Offline	Offline + Online
A	4.1	2.7	3.1
B	12.2	6.8	8.5
C	9.3	4.7	6.7
D	15.3	10.6	12.1
E	10.2	6.5	7.8
F	17.9	9.5	13.6

Table 3.2: Path lengths in meters for offline+replan approach used for comparing against the offline+online approach are presented. Offline only path length are presented as a reference for obtaining pointclouds without filling the anomalous regions.

The savings in correcting the density online compared to iteratively executing replanned trajectories is higher for parts B and C. Parts B and C were found to have multiple anomalous regions with poor point density. We can expect the behavior whenever multiple anomalous regions are found since the robot has to finish execution and then travel back to all anomalous regions from the home location as compared to executing shorter paths in online execution. Parts A, E, and F required longer trajectories even for the online planner and had one anomalous region which led to similar execution times for replanning and online methods. We can expect fewer improvements in path lengths when fewer anomalous regions are present.

3.6.3 Performance Analysis

3.6.3.1 Selection of point admittance function η

Our *eta* function uses bounds on α, β, and γ as $[0.26, 0.4], [0, 0.2], [0, 1.0]$ respectively. A more sophisticated η function can also be used with our method which will admit points more effectively based on the sensor capabilities. The selection of η is important since it trades off the accuracy of the measurement with respect to the time taken for process execution. A very conservative η function will reject several points and execution times will be very high as compared to the gains

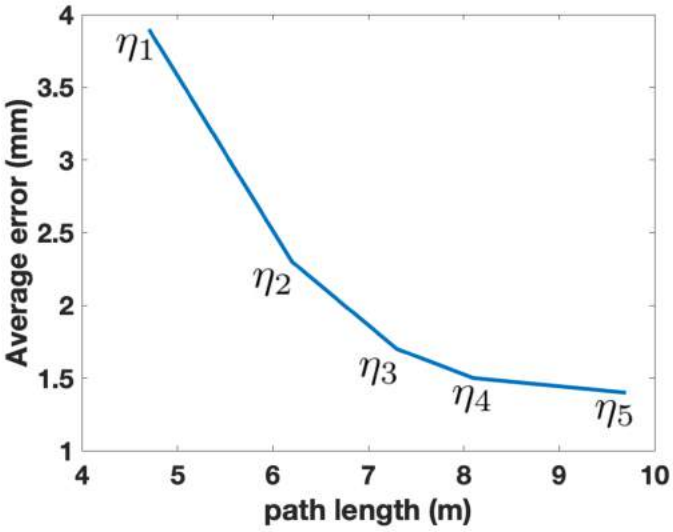

Figure 3.13: The relationship between average error and path length found by using five different bounds in the η function. The functions η_1 to η_5 successively constrain the acceptable range of the α, β, and γ showing an increase in path length. However, the corresponding reduction in average error across the output pointcloud diminishes eventually.

in accuracy and vice versa. We will present an experiment conducted using part B that shows the trade-off.

Figure 3.13 shows the relationship between choice of η function and accuracy of the measurement using deviation error values for part B. Average deviation over all the points of the measured cloud $\tilde{\mathscr{P}}$ and true cloud $\mathscr{P}$ are used along the Y-axis of the plot and camera path length along X-axis for five different cases of η function. The bounds over α, β, and γ for the five η functions η_1, η_2, η_3, η_4, and η_5 are $\alpha_1 : [0.0, 0.9], \beta_1 : [0, 0.8]$, and $\gamma_1 : [0, 3.14]$, $\alpha_2 : [0.1, 0.7], \beta_2 : [0, 0.6]$, and $\gamma_2 : [0, 2.5]$, $\alpha_3 : [0.2, 0.6], \beta_3 : [0, 0.5]$, and $\gamma_3 : [0, 2.0]$, $\alpha_4 : [0.3, 0.5], \beta_4 : [0, 0.4]$, and $\gamma_4 : [0, 1.5]$, $\alpha_5 : [0.3, 0.4], \beta_5 : [0, 0.2]$, and $\gamma_5 : [0, 1.0]$ respectively. We can observe that as the bounds over the parameters for which a point will be accepted get conservative, the average error across the output reduces. However, the reduction in error is diminishing with more conservative η function at the cost of high gains in path length. Therefore, an optimal η function should be selected that provides the desired accuracy with least path length.

Any η function that provides more conservative bounds will admit fewer points in the data. The camera error is high for points that are at the boundary of the operational range and reduces as they get closer to the center of the operational range. The reduction in error is not linear. Reduction in error is higher in the beginning and gradually decays. Hence, making the η function more conservative is unnecessary. We recommend using more sophisticated learning-based models to find the non-linear relationship and provide reasonable bounds on the camera parameters. We also expect similar behaviors when other sensors are used with our approach.

3.6.3.2 Effects of N on performance

We use four different values of N as $N_1 = 20, N_2 = 60, N_3 = 100, N_4 = 225$ that is 0.1, 0.4, 0.7, and 1.5 times the number of points used to generate the results presented earlier for part B. Figure 3.14 shows how the average error across all the points in output cloud $\tilde{\mathscr{P}}$ for part B varies with each of the N selected. The error reduces as the number of points increases in the data. However, a further increase in N past a value will result in increased computational efficiency and higher execution time with little gains in error reduction. The figure also provides insights into issues with the selection of a lower value of N causing insufficient points and loss of accuracy.

The variance of point error reduces by factor of $\sqrt{N}$ according to the relationship $\sigma_{new} = \sigma_{old}/\sqrt{N}$. σ_{new} will decrease faster as N increases at the beginning leading to lower values of average error in the plot. A similar relationship will be obtained irrespective of the part used. However, the reduction in error value saturates with higher values of N as the camera will not produce zero error results and will always have some error value for a point. Larger values of N will simply converge to this small value of error. However, the execution time will significantly increase with larger N so a value of 30-200 points within a $\mathscr{N}$-neighborhood is recommended.

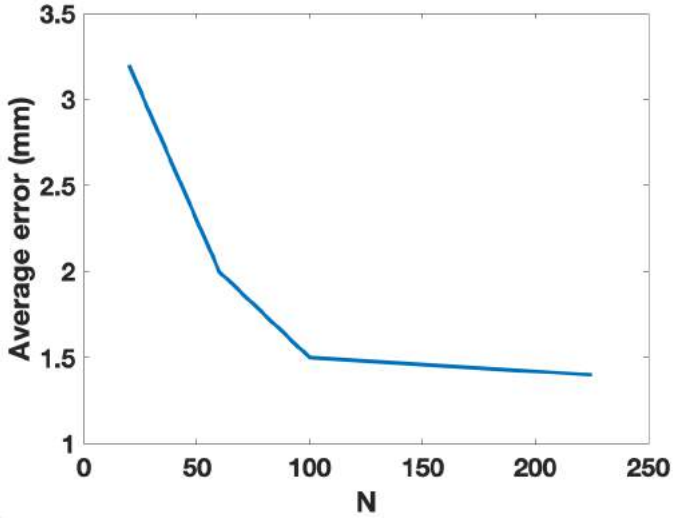

Figure 3.14: Diminishing reduction in average error across the entire output pointcloud for part B is observed with increasing the minimum number of observations N for a $\mathcal{N}$-neighborhood. Additionally, a lower value of N will also lead to a high average error.

3.6.3.3 Consistency in Performance

Part B was also used to perform multiple consecutive 3D reconstructions to evaluate the consistency in the scans. Five different measurements were taken and the average and maximum error in millimeters across all the points are presented in the Table 3.3. Five measurements are also shown next to one another as a point deviation colormap in Figure 3.15. The scale of the error used for the colormap is different from the scale previously used to compare the baseline approaches. Therefore, we notice high error red points on the deviation map. However, the red points are within the maximum error reported for part B.

3.7 Summary

We have demonstrated that by characterizing the depth sensor properties using an admittance function only high-quality points were accepted during planning and execution. Camera performance model constraints were developed to also introduce diversity and a minimum number of

Scan Number	Average Error (mm)	Maximum Error (mm)
1	1.61	3.5
2	1.65	3.5
3	1.48	3.2
4	1.70	3.9
5	1.56	3.4

Table 3.3: Five measurements of average and maximum error across all points for part B are presented.

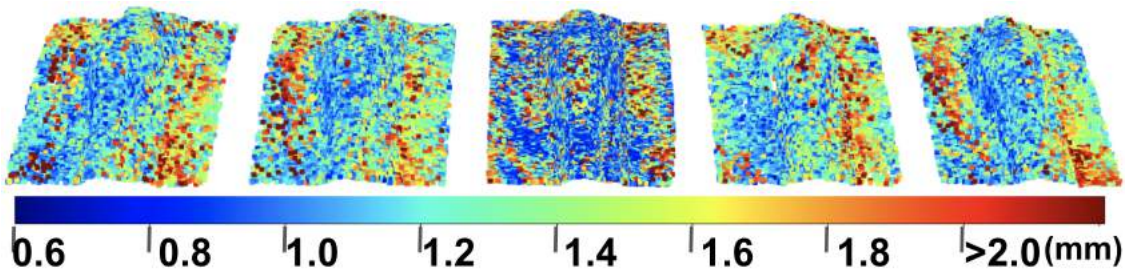

Figure 3.15: Point deviation colormap for five consecutive scans of part B. Left to right correspond to scan numbers 1-5 in
Table 3.3.

admitted points in local neighborhoods to produce a reduced error output. Additional kinodynamic robot constraints were developed for generating robot trajectories that can execute paths to capture the high-quality pointclouds. An online planning algorithm was presented to guarantee a uniform point density throughout the part by covering the anomalous regions. The contributions from this work can also be applied to a wide range of sensors used in the industry.

Chapter 4

Algorithms for Multi-Robot Manipulation of Deformable Viscoelastic Materials

4.1 Introduction

Composites are widely used in the industry to realize lightweight structures with high performance. The global composites industry was valued at $66 billion in 2015 and is expected to grow to $130 billion by 2024. [73]. Carbon fiber is a widely used raw material for the production of composites. For low-volume production and complex parts, a hand layup process is used to laminate plies of carbon fiber prepreg (a resin-infused fiber bed). An expert technician deposits pre-cut plies of prepreg in prescribed orientations on a mold to build up the desired thickness. The layup is then vacuum-bagged and cured at a prescribed temperature and pressure to produce the final component. Figure 4.1 illustrates the layup process on a medium-sized mold. The hand layup process is ergonomically challenging and skill-intensive. Human operators must apply various levels of pressure to the plies. Multiple operators must collaborate to conform larger plies to complex contours. The hand layup process is labor-intensive and can exhibit inconsistency due to variability in human operation.

Current automation solutions in the composites industry are limited primarily to automated tape layup (ATL) and automated fiber placement (AFP). In these processes, prepreg tape of

Figure 4.1: Illustration of hand layup process being done by multiple operators [143].

various widths (comprised of unidirectional fibers) is used, as opposed to plies of fabric prepreg. These solutions are limited primarily to larger and simpler geometries due to the large tape-dispensing end-effectors. We have shown the feasibility of hand layup automation in our prior work in [121, 122] which shows a robotic cell to handle and conform to prepreg plies. Figure 4.2 shows the robotic cell used for automating the process. In an industrial setting, we envision a robotic cell to perform the complex layup process under human supervision.

Multiple articulated robot arms are required to perform the layup operation in the robotic cell. Significant planning challenges arise as a consequence of using robots in the cell. We cannot expect humans to program the robots in the cell. The tool paths for the draping tools and grippers used for the operations of the cell must be automatically planned. Planners that generate the point-to-point motion and path-constrained motions are required. Moreover, manipulating the plies is a complex physics-based motion planning problem. Vision and haptics-based sensors must

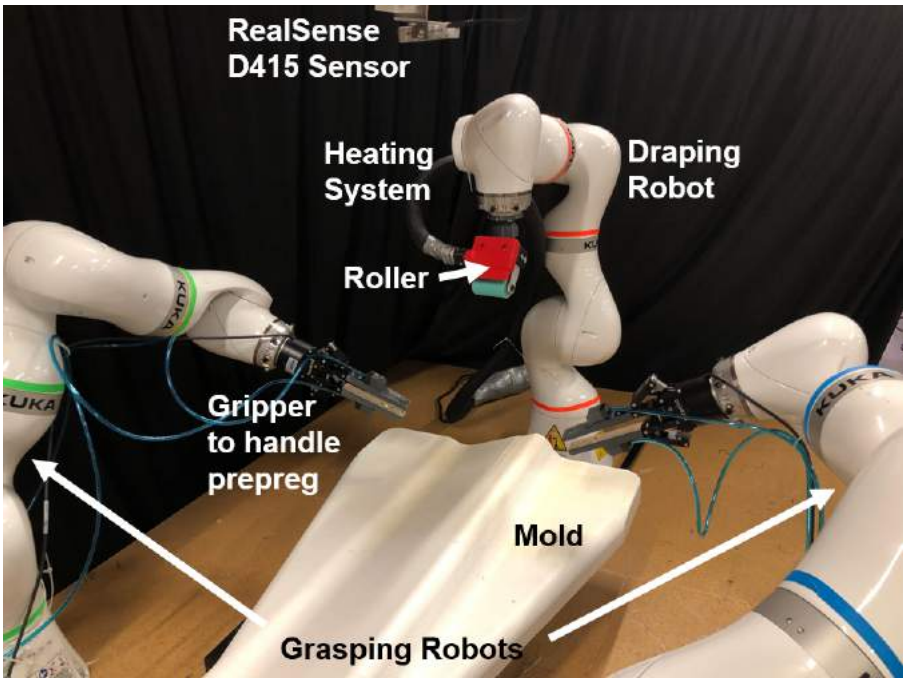

Figure 4.2: Robotic cell comprising of manipulator, specialized grippers, heating system, feedback systems, and conforming tools required to automate the carbon fiber ply layup process.

be integrated into an online plan refinement to avoid defects and handle contingencies. Path-constrained synchronous trajectories are needed, which satisfy process and dynamic constraints. The feasibility of these trajectories is largely dependent on the placement of the robots around the mold. Planners that can automatically determine where to place the robots around the mold are required. The system must address both trivial and challenging planning problems in order to automate the process.

We identified two planning problems which are challenging for the system: (a) finding grasp plans that generate the tool paths for grippers to handle the ply, and (b) finding suitable robot placements. This work addresses the grasp planning task. Grasp plans are the tool paths for the grippers that manipulate the ply during the process. Several grasp plans can exist in the space around the tool. The system must determine a grasp plan which leads to a defect-free layup and minimizes a cost defined by our approach. We test our algorithms on a mold inspired

by an industrial component and also develop interfaces through which humans and robots can collaborate.

4.2 Related Work

4.2.1 Automation in Composite Manufacturing

In this section, we describe the progress towards automating the hand layup process.

Previous work [75] demonstrated how a numerical model could be created to predict the draping of a ply on the mold. This drape can then be used to estimate the sequence of motions that can be performed by a technician or a machine to achieve the desired layup.

A categorization of the layup techniques used by technicians has been reported [56]. These layup techniques can serve as baseline motion primitives to automate the ply draping. In [28], a complete system to handle and lay up prepreg on a mold using an industrial robot was proposed. A concept of cell design was illustrated with four manipulation mechanisms. Molfino et al. proposed a hyperflexible cell design in [144]. Although these works lacked algorithms to automatically generate instructions for the robot, they demonstrated concepts of automation solutions that can be applied. Researchers in [57] used an industrial robot to apply pressure to drape the ply. The robot was manually programmed, and the ply was preformed in a hydraulic press to avoid manipulation and shear. This work furnished insights into what end-effectors can be used with manipulators, and challenges in the layup process. Other works also studied the development of end-effectors to handle composites, which requires the design of specialized grippers for carbon fiber prepreg [173, 62]. Newell et al. presented a numerical model to predict the behavior of prepreg when gripped at certain points [151]. Behavior or woven prepreg plies on a doubly curved mold was investigated using a kinematic mapping in [104]. Automated handling for pick and

place operations of composites during manufacturing was briefly reviewed in [22]. A brief review of grasping technologies and advances in automation of composites is provided in [58].

4.2.2 Multi-arm Manipulation of Deformable Materials

As illustrated in Figure 4.2, the robotic cell requires multi-arm manipulation of a prepreg ply (deformable material). In this section, we will discuss the work related to this topic. Manipulation of flexible materials requires coordinated motion and/or control of multiple different robots [159, 209, 188, 42, 78, 46, 3, 106, 187, 74]. Recent survey papers on the manipulation of deformable objects include [99, 89]. Different applications impose different requirements on the underlying approach for solving the problem. Specialized planning, learning, and control approaches have been developed for 1-D problems (e,g., wires and ropes) [146, 170, 167, 156, 169, 196, 107, 134], cloth folding [13, 116, 50, 118, 194, 103, 140, 141, 9, 172, 135, 136, 101], and ply manipulation [64, 105, 83, 202]. A method to generate trajectories for multi-robot systems to handle sheet metal while minimizing deformations was proposed in [69]. Some approaches, such as rope manipulation and cloth folding, do not impose strict requirements on the final shapes. These approaches seek only the qualitative presence of the desired feature after the operation. Some applications require the final shape to be controlled precisely. In applications such as composite layup, the behavior of the underlying material can be complex. Therefore, we rely on advanced simulations to estimate the state of the flexible material.

Sampling-based motion planning approaches have been developed for manipulating deformable objects in static environments [108, 11, 145]. Planning approaches that consider deformable environments have also been investigated [165, 65, 66, 158]. Learning from demonstration methods has been adopted to learn task-specific actions for deformable parts [112, 113, 85]. Recent work in this area formulates a constrained optimization problem to compute a plan for a team of mobile manipulators to manipulate a highly deformable object [3]. In [48], a team of mobile robots was used to grasp and assemble a part comprised of multiple components. They formulated the

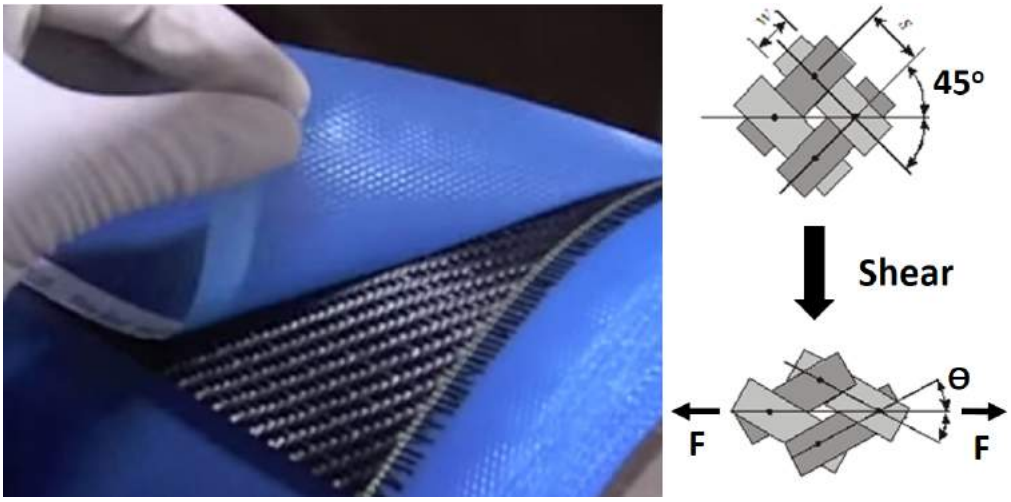

Figure 4.3: Prepreg (carbon fiber infused with epoxy) covered with blue backing paper [5] (left), and an illustrative example of in-plane shearing mechanics (trellis shear) for woven fabric (right).

problem as a constraint satisfaction problem, then divided it into independent smaller problems to achieve faster computation times.

4.3 Background and Terminology

4.3.1 Carbon Fiber Layup

The carbon fiber prepreg ply is a dry fiber bed infused with a viscous thermoset polymer resin (e.g., Figure 4.3). Fibers can be unidirectional (UD) or woven fabrics. Depending upon the type of weave and resin system used, various mechanical properties can be obtained. Multiple plies are stacked on top of the mold at prescribed orientations. The designer can tailor the anisotropic mechanical response as per loading conditions by controlling the orientation of each ply and type of materials being used.

The prepreg structure can undergo in-plane deformation (e.g., trellis shear, as shown in Figure 4.3). In-plane deformation allows conformation of flat prepreg plies onto non-developable and complex surfaces. Technicians apply local pressure to shear the ply and conform it to the mold.

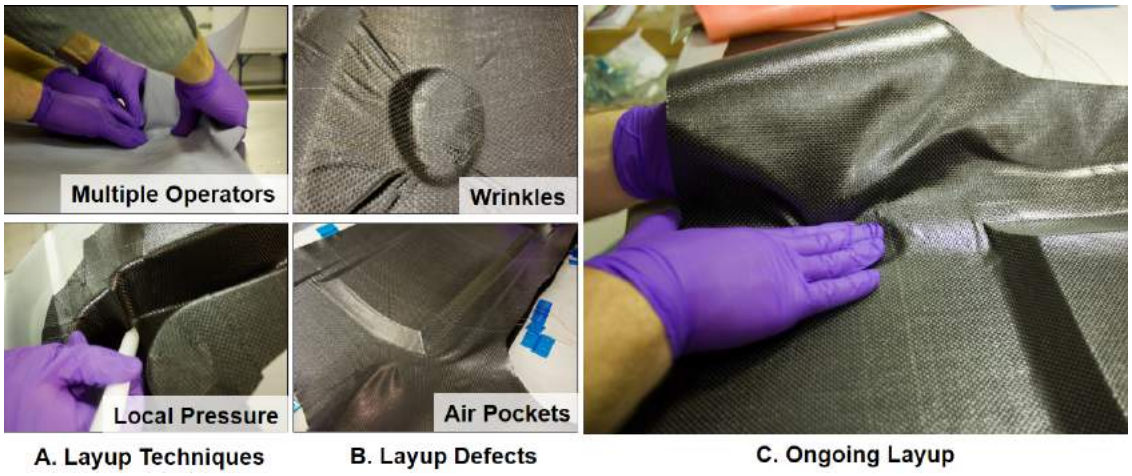

Figure 4.4: (A) Layup techniques involving use of multiple operators and local pressure application [122]. (B) Layup defects like wrinkles and air pockets. [122]. (C) Example of an ongoing layup.

Hand-held tools or hands are used to apply this pressure [92]. Typically, a 1-10 cm^2 area is conformed at a time. Figure 4.4 shows the different techniques used during the process and illustrates defects that can occur. Moreover, material and environment-related factors can also influence the drapeability and quality of the layup. Prepreg tack (adhesion), for instance, depends on ambient temperature, humidity, out-time, and through-thickness compliance (bulk factor) of the prepreg. After a layup, the laminate is enclosed in a vacuum bag and cured using a prescribed pressure and temperature cycle to produce the final component.

Quality of the layup is evaluated on the basis of how well the plies are conformed to the substrate (mold or the underlying prepreg). In-plane and out-of-plane shearing can cause changes in fiber orientations. The fiber orientations should be within specified tolerances. In addition, defects such as air pockets, wrinkles, or foreign object contamination should be absent. Defects can require scrapping the entire laminate or reduced part strength. Wrinkles can cause delamination between layers. Another important factor in determining laminate quality is the resin distribution across the laminae. Resin-starved regions can cause the fibers to fail prematurely, as no matrix is present to distribute. Despite these challenges, hand layup is used for complex molds or production volumes where current automation solutions are technically or economically infeasible.

4.3.2 Draping and Grasping Robots

The cell consists of two types of robots, (a) the draping robot and (b) the grasping robot. The draping robot applies local pressure on the ply to conform it to the surface of the mold. The grasping robots manipulate the ply to facilitate the layup operation. The robots are equipped with custom end-effector tools that are designed as per the mold geometry. An RGBD-camera is used for monitoring the layup process during execution.

The draping robot needs to account for machining errors during the manufacturing of the mold and the registration errors while placing the mold in the work cell. This requires the robot to operate under impedance control. We have selected 7 DOF KUKA iiwa robot for draping that provides impedance controller. The controller allows the robot to comply with the surface irregularities as well as maintain constant force during the draping operation. Same robots are used for grasping as well. For larger and more complex geometries, the grasping robots require force feedback while manipulating the ply. Tension in the ply also must be accurately controlled, and the robots need to comply if the ply is being sheared or stretched too much. We can achieve these objectives under impedance control.

4.3.3 End-effector Design

Draping Tools: A roller is used to replicate the motion of human hands to drape the ply. A cylindrical roller is effective for the conforming ply on single- and double-curvature convex regions of the mold. Figure 4.5(A) shows the fabricated cylindrical roller. The main body is additively manufactured with ABS. The support structures for the roller are aluminum for reducing the dimensions to access compact features within a mold without making any unwanted contact with the ply. For concave regions with small radii, we have designed a conical roller Figure 4.5(B), and a dibber Figure 4.5(E) tool. The conical roller is used for improving the draping speed along the concave regions, but it poorly conforms plies to areas with tight corners. This limitation of

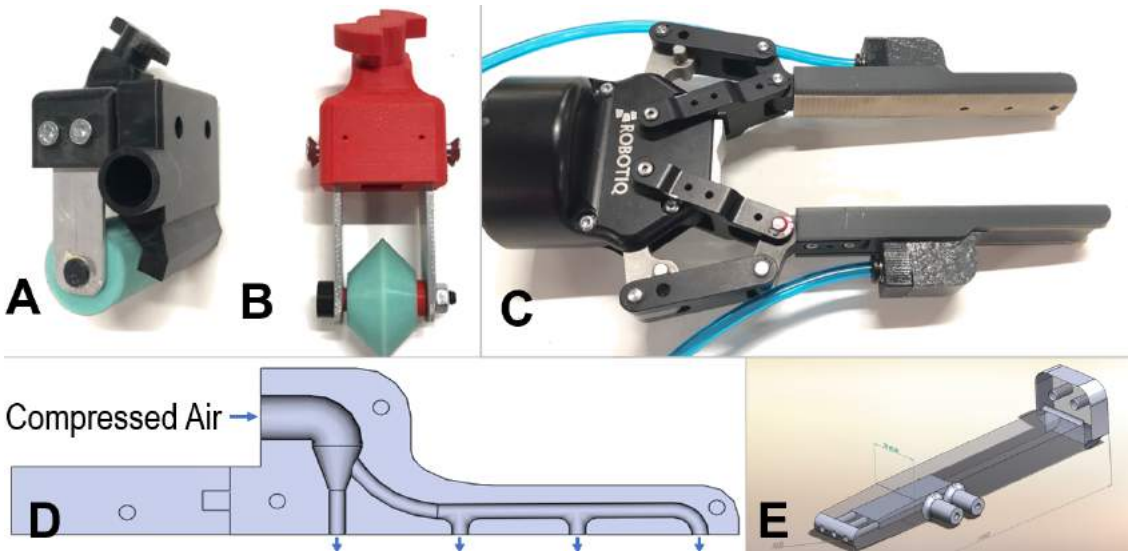

Figure 4.5: (A) Fabricated cylindrical roller. (B) Fabricated conical roller. (C) The gripper used. (D) Section view of the gripper fingers showing the air channels. (E) CAD model of the dibber used in the process [121].

the conical roller is overcome by the dibber tool, although it reduces the conforming speed. The end-effectors used in our work are inspired by the work done in [57].

Better conformity can be obtained by heating the prepreg. Heating improves tack and assists in adhering the ply to the mold. Heating also reduces the viscosity of the infused resin so fibers can deform easily and take shape. We have designed a heating system that applies heat locally to the region that will be conformed as the draping robot moves.

Grasping Tools: Commercial grippers (Robotiq) are used for grasping. Prepreg readily adheres to the rubber fingers of the gripper. Teflon (PTFE) is appropriate for handling prepreg. We have designed custom gripper fingers to grasp and manipulate the ply. Along with it, internal channels are provided to carry compressed air. A high-velocity jet of air detaches the ply from the gripper fingers if it adheres to the gripper. The air jet is deployed to prevent the ply from being dragged with the grippers when they change location. Figure 4.5(C and D) shows the gripper and the CAD model with internal channels used in the setup.

4.3.4 Part Inspection

We assess layup quality using four metrics described as follows:

1. *Conformity:* The degree to which the sheet has conformed to the substrate. A map is generated to compare the point cloud of the layup with respect to a reference (CAD or substrate).

2. *Fiber Alignment:* Measure of the deviation of fibers in the woven fabric with respect to a reference or average alignment in a region.

3. *Resin Distribution:* Measure of distribution of resin (ideally homogeneous) throughout the fabric after the layup process.

4. *Other Defects:* Foreign object deposition (FOD), air pockets, wrinkles, or fiber damage.

The cell is equipped with Apodius Vision System 2D, Hexagon absolute arm with an integrated laser scanner, Intel RealSense D-415 depth camera, and DinoLite digital microscope. Figure 4.6 shows these instruments in images A, B, C, and D respectively. We use the depth camera for monitoring the state of the sheet in real time. The state of the sheet is extracted after occluding the robots and the grippers from the cell. Online modification to grasp plans generated in this paper can be made by using feedback from the depth camera. Defects such as air pockets, and wrinkles can also be detected using the sheet state. Torque sensors in the robots monitor the force being applied at the end-effectors. Necessary compliance and local adjustments are made using the data from these sensors. DinoLite digital microscope is used to study the distribution of the resin in the layup.

4.3.5 System Architecture

Figure 4.7 shows the system architecture of the robotic layup cell. The cell takes an expert input and automatically generates the plans for the robots. We categorize the automated composite

Figure 4.6: Inspection equipment used in the robotic cell. (A) Apodius 2D sensor. (B) Hexagon absolute arm with integrated laser scanner. (C) DinoLite digital microscope. (D) Intel realsense D415 depth camera.

layup process into three phases: (a) planning phase, (b) setup phase, and (c) execution phase. The expert user input is discussed in detail in Section 4.3.6. We will now discuss the three phases of our system.

4.3.5.1 Planning Phase

The planning phase is responsible for taking the expert generated inputs, and converting them into actions that can be executed by the robots for performing layup operation. Expert user input provides the draping tool paths and some additional information that is used to generate the grasp plans and robot placements. The process parameters based on the mold geometry, grasp plans for manipulating the ply, robot placements with respect to the mold, and the draping and grasping robot trajectories are computed in this phase. The grasp plans are visualized through an interface. The user can modify these plans through the interface, if required.

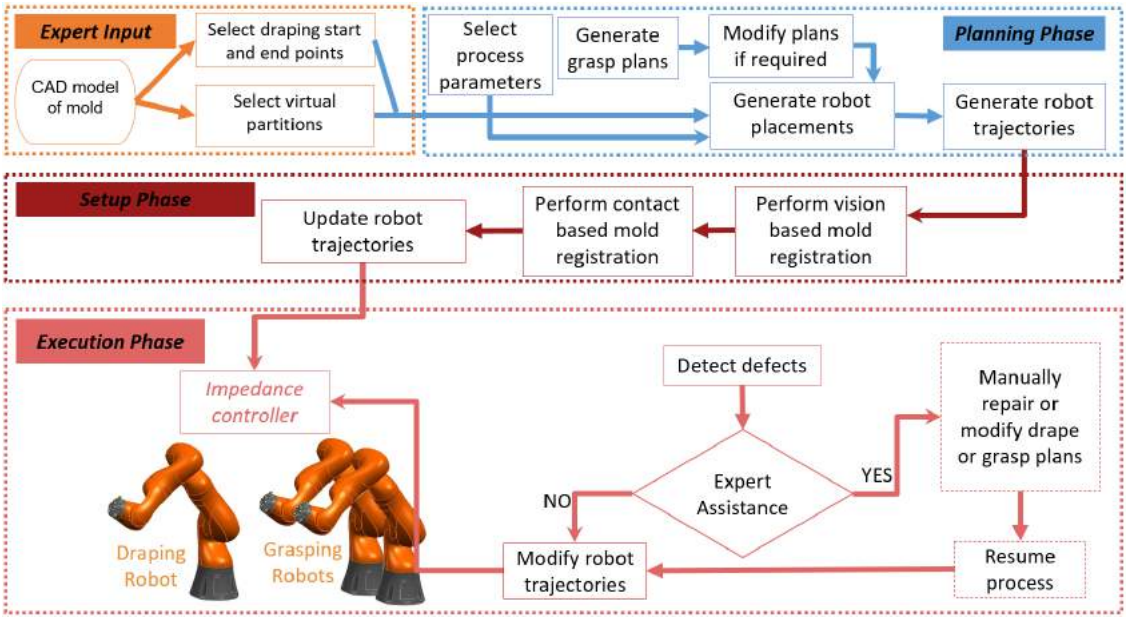

Figure 4.7: System architecture diagram illustrating the three main phases in the robotic cell for composite layup.

The system selects the process parameters required, given the CAD geometry and the type of material. We investigated the bounds on these process parameters by conducting offline experiments on common mold geometries [121]. For this work, the draping force (F_{drape}) and the dibbing force (F_{dibb}) will be bounded within $[15, 30]$ N. Maximum Cartesian velocity of the roller used will be 50 mm-s^{-1} and the temperature of airflow used will be 45° C. The stiffness of the end-effector under impedance control is bound within $[1500, 3000]$ N-m^{-1}.

4.3.5.2 Setup Phase

The setup phase consists of all the components that are required before the physical layup process begins. This phase involves placing the robots at the computed locations with respect to each other and to the mold, performing a vision and contact-based mold registration, and recomputing the robot trajectories.

The robot placement algorithm will give the transformation of the robots with respect to the mold. The cell operator can then place the robots using this transformation. However, in practice,

the exact transformation of the mold is different so the transformation error must be corrected. The CAD model provides a reference point cloud. We then generate a point cloud from the depth camera and compare it with the reference point cloud. An iterative process similar to iterative closest point (ICP) is used to find the transformation of the point cloud with respect to the robots. We then perform contact-based registration to obtain an accurate transformation of the mold. The new transformation is used to recompute the robot trajectories.

4.3.5.3 Execution Phase

The robot trajectories are executed by the impedance controller. A Cartesian impedance controller measures the joint torques and computes the force at the end-effector to comply as per hooke's law. Sensors are used to monitor defects during the process. We use torque measurements and camera feedback for monitoring. The feedback is used to refine the drape and grasp plans to prevent the defects. An expert is called for assistance if the robots cannot repair the defects. The human operator also performs minor adjustments to the grasp plans if required to prevent defects from occurring. An online grasp plan modification strategy has also been presented in our work in [119].

4.3.6 Getting Expert User Input

Expert input is a part of the initial setup process which simplifies the grasp planning problem. The user selects regions on the mold which deconflicts the workspaces of the robots. Grasp points which can be used by the robots are also specified. More details on how expert input is used in planning is described in Section 4.5.3. A drape simulator is used to simulate how a fully draped ply appears on the mold. Figure 4.8 shows the molds we are using to determine the grasp plans and the corresponding drape states simulated. The expert can estimate the drape sequence to be followed to drape the ply from these simulations. We use a virtual fiber placement simulator or VFP provided by the research group at the University of Bristol.

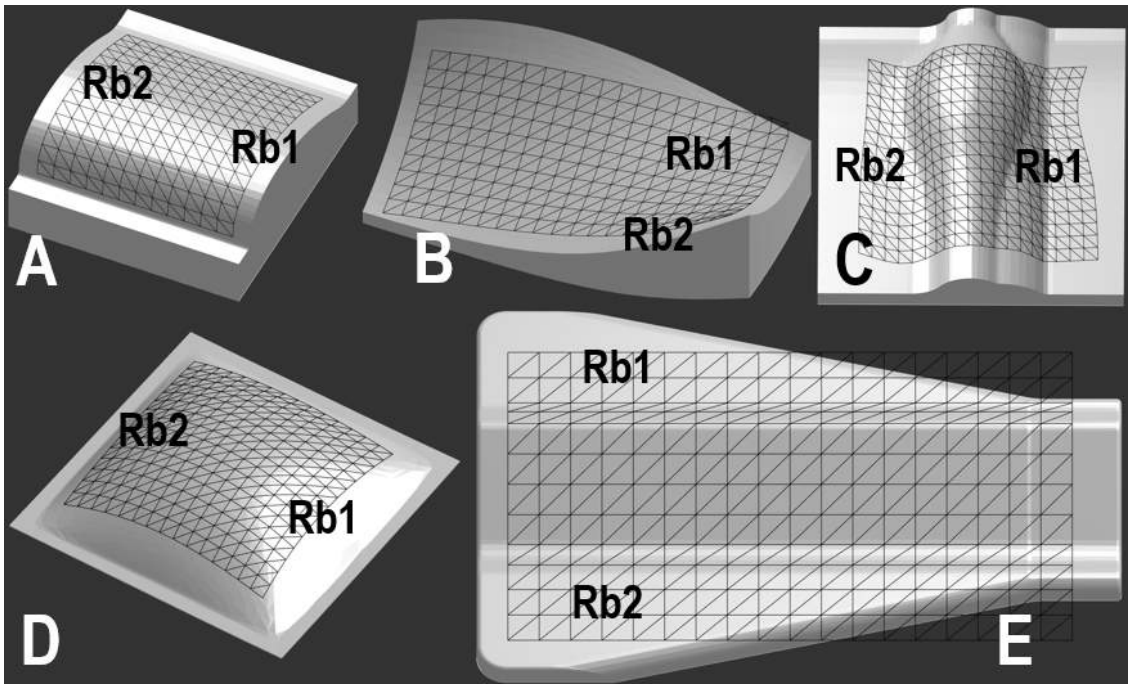

Figure 4.8: The molds used in our work. Fully draped mesh obtained from virtual fiber placement is shown on the molds.

The robots must follow the drape sequence. We have developed an interface - the human operator selects the mesh triangles that belong to each discrete stage. Figure 4.10 shows a few of the discrete stages in the draping sequence for one of the molds used for evaluation. Stage 1 in the figure shows that the operator starts by adhering to the ply in the center portion of the mold, which is also the highest point on the mold. Later, the operator proceeds downwards towards the right in stage 5, and finishes one half of the mold by stage 10. Finally, the left half of the mold requires four more discrete steps to cover the entire mold surface. Figure 4.9 shows the draping regions selected by the technician for all the molds that are used to evaluate the layup operation. Each region in the figure is shown in a distinct color and labeled as RX, where X is the number of the region. For example, Region 1 is denoted by $R1$.

In each region RX of the draping sequence, the robot draping tool must make contact with the ply and apply the necessary force to attach the ply to the mold. In this paper, we use the

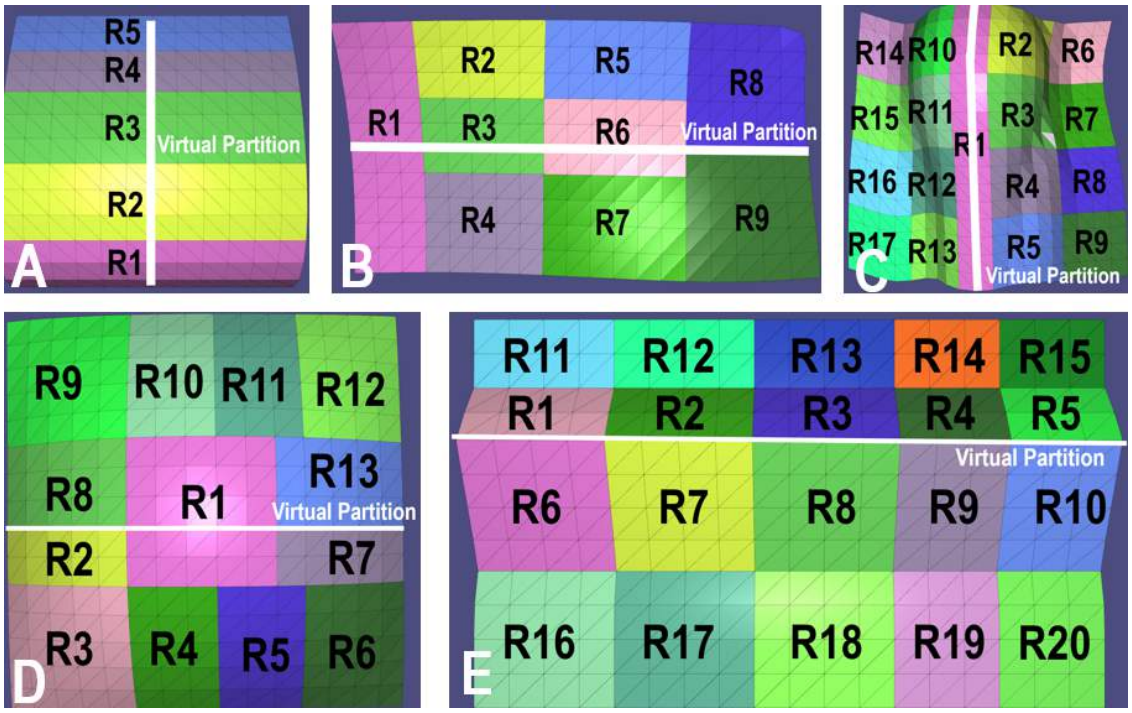

Figure 4.9: The user interface is used to select region (R) in the draping sequence, virtual partitions, and mesh triangles along the ply edges which the robot can use to grasp the ply.

technician's assistance to provide the path that the draping tool should follow while laying up the ply. We have developed an interface (see Figure 4.11) which the technicians can use to input the draping paths. The user can select the start and endpoints of each drape path on the mold surface, and the software generates geodesic curves on the mold that connect these points. These geodesic curves comprise discrete waypoints of the surface of the mold, and we term this collection of discrete waypoints as draping tool paths, which are used to compute draping robot trajectories.

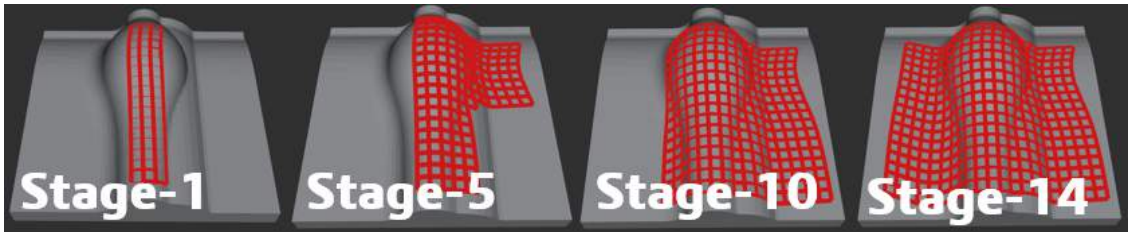

Figure 4.10: Four intermediary steps of the draping sequence are shown. In all the draping sequence is discretized into 14 steps. Step-14 is the fully draped ply. Unconformed ply is not simulated.

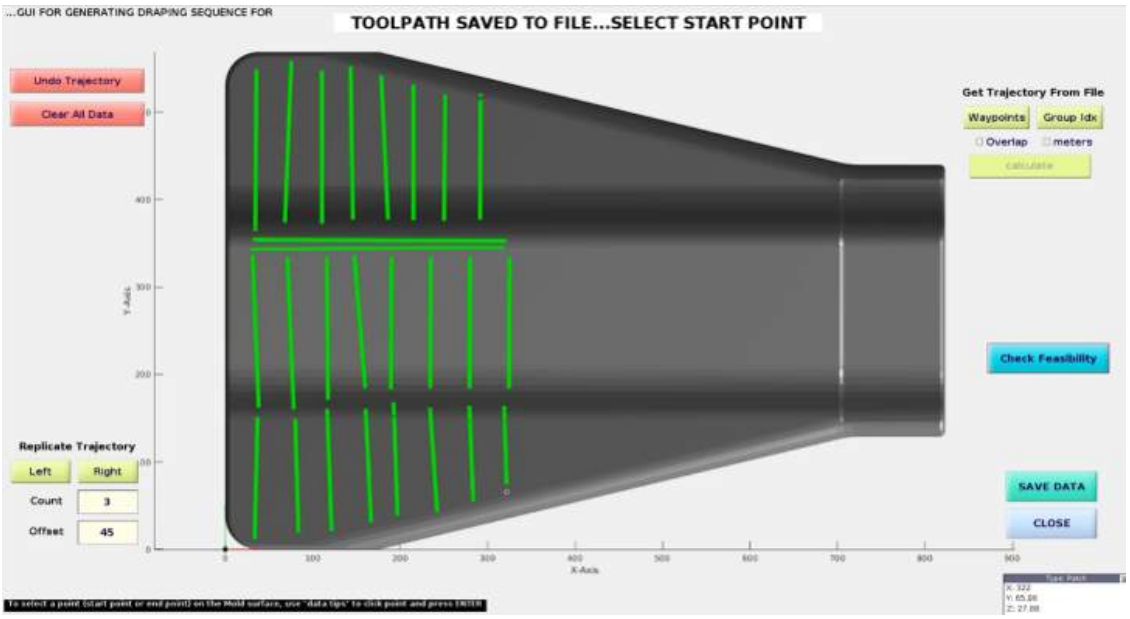

Figure 4.11: The graphical user interface (GUI) used for the selection of draping tool paths is shown. The CAD model and a draped mesh approximating the ply is also illustrated.

During the layup process, the grasping robots are used to manipulate the ply before the draping robot adheres the ply to the mold. Given the size of the ply, the number of robots used for grasping the ply varies. In the current version for generating the grasp plans, we deconflict the regions on the ply that must be handled by each robot using a virtual partition. These virtual partition lines are provided by the technicians for all the mold and shown by the white center line in Figure 4.9. Section 4.5.2 provides more details about the partitioning and its use in the grasp planner.

4.3.7 Grasp Planning

The grasping robots are responsible for manipulating the ply during the draping process. If accidental contact is made between prepreg layers, air entrapment is possible. Ply collision with the mold must be checked in order to prevent contact. Alignment of the ply must be maintain with a specific coordinate axis on the mold. Misalignment of the ply can be caused if the ply

excessively bends in a region, and the draping tool moves over the region. Excess tension or bending in the ply are high deformation states that can cause wrinkles. Such high deformation states can be characterized by an increase in the energy of the ply. Thus, we must generate the plans which keep the energy of the ply low.

The ply is represented by a mesh. Figure 4.12 illustrates a mesh over the mold. We represent a grasp point as the center of a pair of consecutive vertices along the edge of the mesh. The robots can hold the ply from a specific grasp point along the edge of the ply. We can determine the combination of grasp points and their a location in space to obtain a state which is favorable to satisfying some constraints for a given conformed region. Figure 4.12 illustrates different grasp points along the edge of the ply and ply state by holding it at a grasp location from a specific grasp point. In this work, our planner only considers the 3D coordinates along X, Y, and Z axes in Cartesian space for assigning locations.

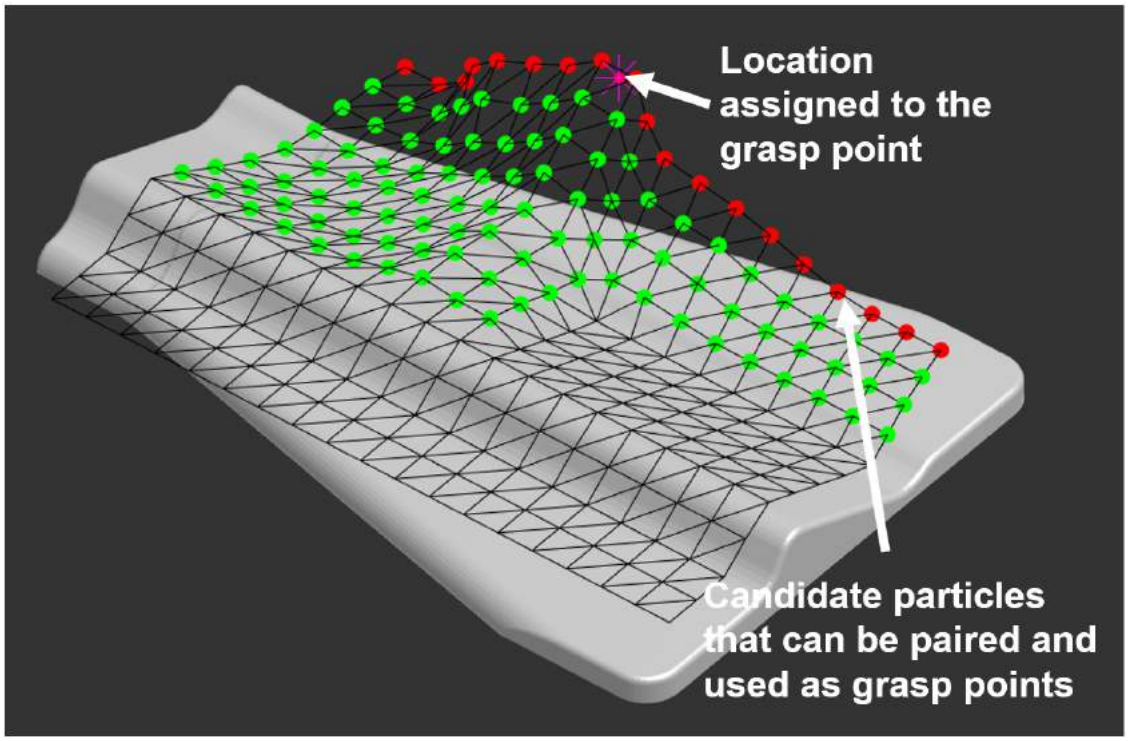

Figure 4.12: Grasp points along the edge of the mesh are illustrated with red markers. The state of the ply simulated while holding it from a grasp point at a location in space is also shown.

In the proposed framework, we have discretized the draping process into a sequence of regions. As stated earlier, this sequence of regions is given by a human expert. Each region with conformed and unconformed ply vertices in it is termed a *stage*. Figure 4.10 shows only the conformed vertices of the mesh for some stages in the discrete sequence on a mold. Unconformed vertices are not shown, as their state is determined by where the ply is being held. In total, there were 14 discrete stages selected in the drape sequence. The figure illustrates four intermediate stages. Stage-14 is the fully draped ply. We can also increase the resolution between stages by discretizing them with smaller areas, which will result in a smoother solution but at the cost of more planning time.

The sequence of grasping points and their locations for every stage in the process defines the *grasp sequence*. If the grasp points are different for consecutive stages, a re-grasp motion is required. If the grasp point remains the same between the stages, but the locations are different, a repositioning motion is required. Figure 4.13 illustrates this concept between two consecutive stages. There are two robots holding the unconformed ply, which is represented in green. One of the robots has to grasp from a different grasp point, so a regrasp motion is required. The other robot had to reposition the ply from the same grasp point. The sequence of grasp points are computed by the grasp planner. Deformation of the sheet during regrasping motion is not accounted for during offline computation. An online refinement strategy is implemented for considering the deformations that happen when the sheet is released [119]. The draping robot waits for regrasping motion(s) (if any) to be completed between stages. However, trajectories are continuously executed to conform to the area that is represented by a stage.

We present a solution framework by posing the grasp sequence generation problem as an optimization problem with constraints. Expert users seem to prefer certain types of sequences. These preferences are validated by conducting experiments. User preferences are modeled as a cost function, developed as a part of our approach. Different components of cost function describe aspects of user preferences. We use a weighted cost to capture different aspects of preferences. A grasp sequence that satisfies all constraints and minimizes the given cost is the solution.

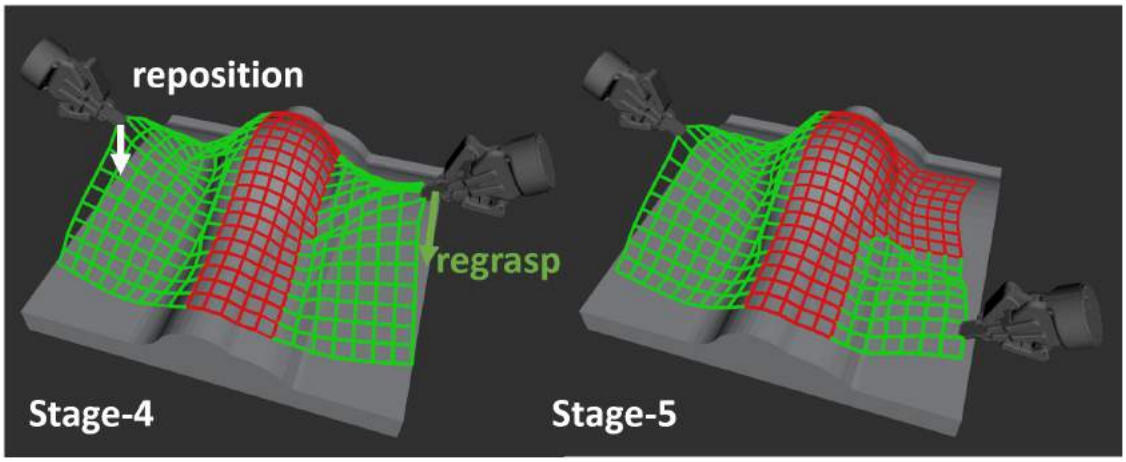

Figure 4.13: Regrasp and repositioning concepts are illustrated for two consecutive stages.

4.4 Problem Formulation

We assume the CAD $\mathscr{M}$ model of the mold is given. $\mathscr{M}_v$ represents the voxelized map of the CAD model. The set of parameters (material properties and scaling constants) used in the drape simulator is represented by $\vec{P}$. Consider a draping sequence $\mathscr{W}_d = \{W_1, W_2, \ldots, W_r\}$, where W_i is a stage and r is the total number of discrete stages in the draping sequence. A set of available grasp points along the edge of the unconformed ply is computed. For every stage, we will have this set given by $G_r = \{g_1, g_2, \ldots, g_k\}$. For an unconformed ply, all the grasp points will be available, in which case k equals the total number of grasp points on the ply. If we assign a location $< x, y, z >$ in the Cartesian space to the grasp point, we will be able to simulate the state of the ply. We denote this *location assignment* as $l(g_k)$ to the k^{th} grasp point. A grasp sequence can be represented as $G_j = \{l(g_{k1}), l(g_{k2}), \ldots, l(g_{kr})\}$ which is essentially a sequence of assignments of location to the corresponding grasp point for every stage $W_i \in \mathscr{W}_d$. The grasp planning problem is formulated as finding a grasp sequence that minimizes a cost. Equation (4.1) defines the grasp planning problem.

$$G_s \leftarrow \arg\min_{G_i} \ \mathscr{C}_t(G_i, \mathscr{W}_d, \mathscr{M}_v, \vec{P}) \tag{4.1}$$

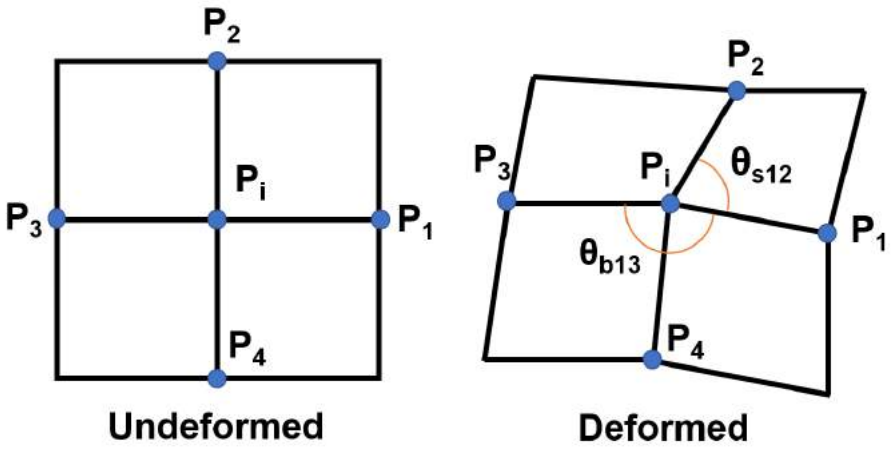

Figure 4.14: Neighborhood of a particle P_i in the mesh. Illustrations of the deformed and undeformed mesh are provided. The position of the particles and shear and bending angles are used to compute the potential energies.

G_s is found by minimizing a cost function that has three main components, as discussed earlier. The next section defines the cost function.

4.5 Approach

4.5.1 Cost Function Components for Guiding Grasp Planning

Energy Function: Breen et al used a physics-based simulation to generate the grasp plan, and our approach is similar to their work [26, 27]. Each particle (vertex) is allowed to translate along the X, Y, and Z axes in 3D Cartesian space. The material properties are encoded as an inter-particle relationship. Figure 4.14 shows a particle P_i surrounded by its manhattan neighbors. Deformed and undeformed configurations of the mesh surrounding the particle are shown. The link between a particle under consideration and a neighbor is represented by a spring. The mass of the ply is divided amongst all the particles. This mass-spring model is widely used for simulations in the computer graphics community. The physical state of the ply represented by n particles in the 3D Cartesian space is defined by the $3n \times 1$ coordinate vector $\vec{X} =< x_1, x_2, x_3, \ldots, x_{3n} >$.

At any state, the total energy of the ply is represented by the following four potential energies: spring, bending, shear, and gravitational. A particle pair is connected by a linear spring in our model, so the spring potential is given by $k * dx^2$, where dx is the change in the distance between the particles. We define the shear and bending energies with respect to the radian angle between the concerned particle, and its neighbor. Figure 4.14 shows the shear angle between particle P_i and its immediate neighbor P_1. Three other shear angles between P_i and the neighbors $P_2, P_3,$ and $, P_4$ can be computed. Bending energy is defined between the pairs P_i, P_1 and P_i, P_3. We conducted two different tests to determine the relationship between shear and bending energies and the corresponding radian angles of deformation. The tests and the relationships obtained are discussed in Section 4.6.2. The gravitational potential is represented by $m * g * h$, where m is the mass of the particle, g is the gravitational acceleration, and h is the change in height (z coordinate of the particle) relative to the mold.

The state of the ply under given boundary conditions is found by minimizing the overall energy of the ply. The boundary conditions in our case include restricting the particles in the draped region and those gripped by the robot from any motion. The rest of the particles are free to move. We are interested in finding the lowest energy state, which will be the solution to the simulation. The unconstrained optimization problem given by the equation (4.2) is solved to find the solution. We use the limited memory Broyden-Fletcher-Goldfarb-Shanno (BFGS) [153, 117] algorithm from the nlopt library [91] to solve the problem. The simulator model is inspired by work done in [131].

$$\vec{X} \leftarrow \arg\min_{\vec{X}} \sum_{i=1}^{n} (U^i_{spring} + U^i_{shear} + U^i_{bending} + U^i_{gravity}) \tag{4.2}$$

Deformation energy cost can also incorporate misalignment constraint for the ply. For instance, if an edge of the ply needs to be aligned with edge of the mold, the cost will be higher if the edge

excessively bends and misaligns. The grasp point and location will be biased towards such regions in order to keep it relatively flat so that alignment is maintained. Alignment costs can also be incorporated as separate inequality constraint over grasp point and location where robot is forced to grasp from predefined points and stay within location tolerances. Such constraints are often defined by the process itself.

Regrasping Cost Function: When the robot needs to *regrasp* the ply, the gripper will open and release the ply, slide along the edge to the next grasp location, and close. The cost function used in our approach minimizes the number of regrasps required for the process. We use the Euclidean norm of the distance between two consecutive grasp locations of the respective grasp points to find the regrasping cost.

Collision Cost: We use the Euclidean distance transform (EDT) of the mold to detect collisions. The position of each particle in the unconformed region of the mesh is used to query the EDT map. The collision has a binary value. If the EDT value is below a certain threshold, the particle in the ply collides and vice versa. The threshold value for declaring if a particle is colliding is the same in case of uniform collision cost. Different threshold values for every particle are used for a non-uniform collision cost. For instance, draped, to-be draped, and free regions on the sheet can be assigned different collision costs for ease of implementing feedback systems in the future.

Weighted Cost: In order to maintain the quality of the layup, we should minimize the area under the absolute energy vs. stages curve. Collision and regrasping costs are then incorporated into the cost function along with the integration of the absolute energy over stages. This cost function $\mathscr{C}_t$ is given by the equation (4.3). C_u, C_c and C_g are the energy cost, collision cost, and regrasp cost respectively. We measure collisions using $\mathscr{M}_v$ and the regrasp cost as the euclidean distance between two expanded node states $l(g_{k1})$ and $l(g_{k2})$ for two consecutive stages.

$$\mathscr{C}_t = w_1 * \int_{W_r}^{W_i} C_U + w_2 * \sum_{W_r}^{W_i} C_c + w_3 * \sum_{W_r}^{W_i} C_g, \tag{4.3}$$

$$C_u = U * ds$$

$$C_c = n_c / n_f$$

$$C_g = \| l(g_{k1}), l(g_{k2}) \|_2$$

where n_c is the number of colliding particles and n_f is the number of free particles for that stage. Also, $w_1 + w_2 + w_3 = 1$. U is the absolute energy of the ply and ds is the normalized value representing a stage. For instance, if the process has r discrete steps, $ds = 1/r$.

4.5.2 Virtual Partitioning of State Space

We can use any number of robots for manipulating the ply. In the current version for generating the grasp plans, we deconflict the regions on the ply that needs to be handled by a single robot using a virtual partition. Figure 4.9 illustrates this concept for our case. The user selects the regions which will be handled by the corresponding robots. For two robots, we represent the two regions separated by a white line. The corresponding robot number is labeled in the figure. Each of the regions is termed as a virtual partition. Usually, in a draping process, the ply is conformed along a line path that runs on the mold. The path inherently deconflicts the workspaces for the molds. In Figure 4.9, for the sequence shown on mold C and E, the first conformed region R1 will split the robot workspaces in two without conflicts. However, for other molds, both the robots will have shared unconformed regions as the free regions for both the robots interact with each other. A robot action in conflicting regions will influence the grasp location of another robot as an unconformed ply state is being shared between them.

Even though the robots will have shared unconformed regions, we solve the grasp planning problem for each robot separately. We find nominal grasp sequences for each robot. We then use

these nominal grasp sequences to account for the conflicting regions. Section 4.5.4 discusses the algorithm in detail. In our case, we use two robots. We solve two independent, grasp planning problems. Figure 4.15 illustrates the use of virtual partitions in our approach. The stages and grasp points available for grasping in the case of mold A for the robot-2 are shown in the figure. While planning for robot-2, all the particles that belong to the virtual partition of the robot-1 remain in their draped position.

4.5.3 State Space Representation

The available grasp points are known beforehand. We can construct the graph $\mathscr{G}$ with the nodes as the available grasp points in every stage. In our approach, selecting a node in the graph is equivalent to selecting a grasp point. A grasp point is selected and then moved around in a discrete Cartesian space. Cost is evaluated at each discrete location. A location that reduces the total cost of the grasp sequence is assigned to the grasp point for that stage. We call this a location assignment search. The location assignment search is discussed in detail later in this section. We evaluate the node cost by using the location assignment search.

We use an approach based on a backward simulation, where we start with a fully draped ply. The root node corresponds to the fully draped ply stage, where no grasp point is available as we progress along with the discrete set of stages; the ply unconforms until it is fully unconformed. This framework can be used to propagate information into later stages. The graph is an acyclic directed graph. Figure 4.15 illustrates the different stages in the process. Unconformed ply particles are highlighted in green. The corresponding graph $\mathscr{G}$ is also shown. As the stages progress, the grasp points that are available to be used are represented by integers along the edge of the ply. G_i essentially is a grasp sequence of nodes, forming a path from the root to the leaf node having the lowest cost. The figure also shows one such path in the graph highlighted by forwarding arrows. As the graph is acyclic and directed, minimum cost grasp points in every stage lead to an optimum grasp sequence.

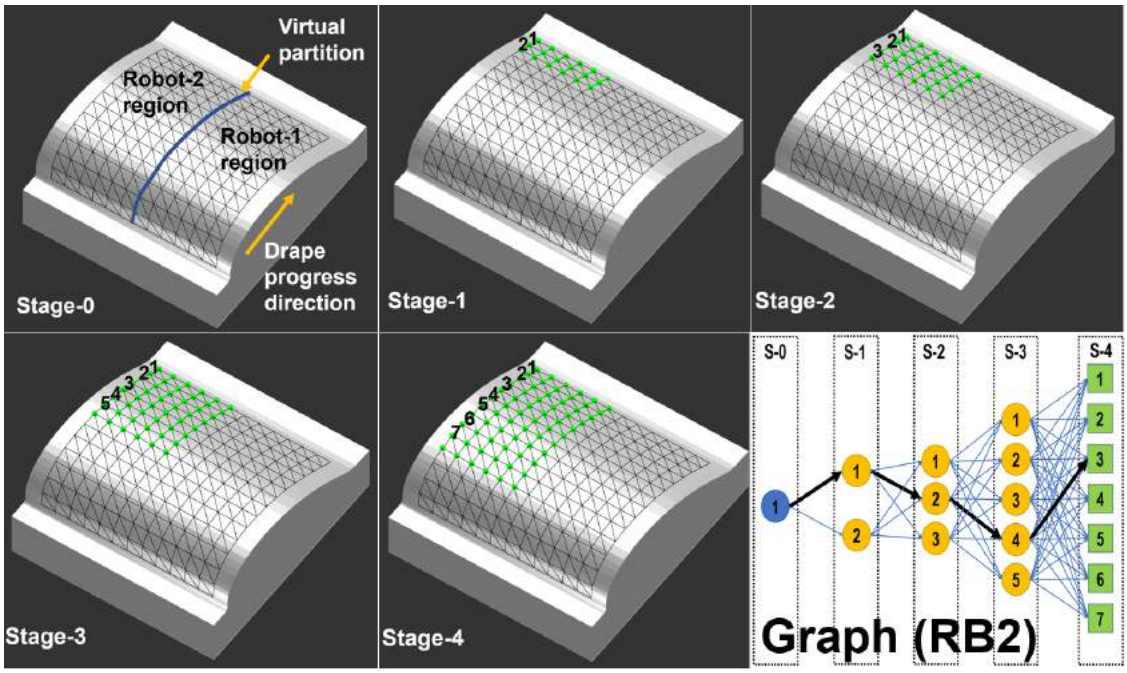

Figure 4.15: Different stages in the backward tracking graph are illustrated. The available grasp points are marked by integers along the edges of the ply. Virtual partition generated by the user deconflicts the regions handled by the robots. Separate graphs are generated for the robots. The graph generated for the robot 2 (RB2) is shown here.

A location assignment search is conducted by varying the height (Z-axis) of the grasp point and moving it forward or backward (Y-axis) relative to the mold. Figure 4.16 illustrates the Z and Y axes of the motion for a grasp point. We discretize the motion along these axes. Motion along the edge of the ply (X-axis) is equivalent to changing the grasp point. The cost varies as a grasp point is moved during the location assignment search. Out of all the different locations that can be assigned to a grasp point, we pick the one with the minimum cost. A state space search is then used to find the optimum solution. In our implementation, Gradient descent algorithm is used to find the minimum cost location. The initial location assignment is given by a heuristic function, which also reduces the computation cost. The heuristic used is described in detail in Section 4.5.4.

The local minimum found by an optimization algorithm, such as gradient descent, is the global minimum due to the behavior of the cost function in this application domain. Figure 4.16

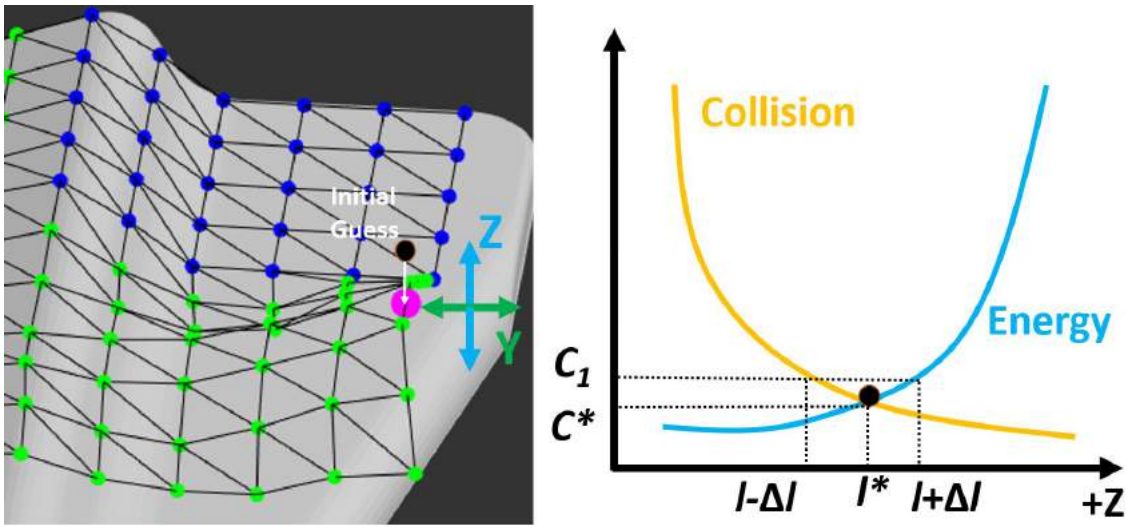

Figure 4.16: (Left) Optimum grasp location (magenta) obtained using an initial guess (black) in the local assignment search. Z and Y axes along which the state space is discretized are shown. (Right) Behavior of collision and energy in the local neighborhood of optimum grasp location. Global minimum (l^*) is found by a gradient based algorithm within the domain. Any step Δl leads to an increase in the cost.

shows the behavior of the energy and collision costs as a function of the Z location of a grasp point. Collision cost decreases as the ply is lifted, but energy cost starts increasing due to high deformation and potential. At one point, a steep increase in energy is obtained with less reduction in a collision. Similar behavior is obtained with these costs when the ply is lowered.

4.5.4 Algorithm for Generating Grasp Plan

Our algorithm is an any-time algorithm. It generates a grasp plan and then updates it further to reduce the cost. The algorithm can be terminated if a time limit is reached, the cost cannot be decreased any further, or all the nodes have been evaluated.

Our goal is to use domain-dependent heuristics to generate a low-cost initial plan and obtain an optimal resolution solution in a computationally tractable time. We propose an ordering heuristic that prioritizes the nodes in every stage. The ordering heuristic first explores the state space to prioritize the nodes (grasp points) which are promising. The planner begins by exploring a

small number of grasp locations for every node. Two grasp locations which are equally spaced between the mold surface and a maximum height above the mold are evaluated. The location with a lower cost is picked and assigned to the corresponding grasp point. All the grasp points in a stage are then ordered based on the respective assigned costs. The procedure is represented as $PrioritizeNodes$ in Algorithm 3. We then use the generated lower cost grasp locations as an initial guess for the gradient descent step used in local assignment search. The heuristic significantly reduces the computational cost. It allows us to generate good initial guess for the gradient descent search during location assignment. A minimum cost solution is then found with fewer iterations as opposed to exploring the entire search space for the optimal solution. Additionally, prioritizing nodes helps us evaluate the grasp points which can have a lower cost in the beginning of the algorithm. As a consequence, our algorithm generates grasp plans with a low initial effort compared to randomly selecting nodes.

The planner goes through all the stages and evaluate the nodes which have the highest priority as determined by the ordering heuristic in the first iteration. The first iteration provides an initial grasp plan having a low cost. In the successive iterations, steps 6 to 16 of the Algorithm 3 are followed. We maintain a list of boolean values (1 or 0) corresponding to each stage. All the members in the boolean list are initialized to one. If a stage showed improvement in the previous iteration (boolean value 1), Another node is evaluated from the top of the prioritized queue. The path is updated ($UpdateOrder$) only if the cost of the new path is less than the previous cost. The newly evaluated node provides information to reprioritize the order of the nodes. We can update this order using the new information at $UpdateOrder$. If the stage does not show any improvement, a node is not evaluated in the next iteration from that stage. The boolean value corresponding to the stage is set in the list to zero. When all the members in the boolean list are zero, and the stopping condition has not been met, we reinitialize all the elements to one (lines 8 - 10 of Algorithm 3). This allows us to bias the search towards evaluating more nodes from the stages that are showing a reduction in cost.

Algorithm 3 GenGraspPlan($\mathscr{W}_d, \mathscr{M}_v, \vec{P}$)

$\mathscr{G} \leftarrow GetGrspPts(\mathscr{W}_d)$
$PriorQ \leftarrow PrioritizeNodes(\mathscr{W}_d, \mathscr{M}_v, \vec{P})$
$path \leftarrow GenInitialPath(\mathscr{G}, PriorQ)$
$iter \leftarrow 0$
$boolList \leftarrow Identity(sizeof(\mathscr{W}))$
while $UntilConvergence$ **do**
  for $s \in [0, \mathscr{W}]$ **do**
    if isZero(binList) **then**
      $binList \leftarrow Identity(sizeof(\mathscr{W}))$
    end if
    if binList(s) **then**
      $path \leftarrow UpdatePath(\mathscr{G}, \mathscr{W}, PriorQ, \vec{P})$
      $PriorQ \leftarrow UpdateOrder(PriorQ, path, \mathscr{G}, \mathscr{W})$
    end if
  end for
end while

This algorithm is used to find solutions for the grasp plans in each of the two-state spaces corresponding to each robot separately in our case. Consider the mold A shown in Figure 4.17 as an illustration. The two independent solutions are found for robot-1 (RB1) and robot-2 (RB2). The first two images show that only the unconformed particles in a robot's virtual partition are considered during evaluation. The unconformed particles in the other robot's partition are considered to be fixed at their draped position during the simulation. The third image in the sequence shows the combined solution, which is generated from the nominal grasp locations. The ply collides with the mold so a state space search is used to reposition the grippers of both the robots in order to minimize the cost. Gradient descent is used to adjust for the grasp locations, which gives us a solution shown in the fourth image.

4.6 Results

4.6.1 Test Cases

We have tested the grasp planner on five different mold geometries. Figure 4.8 shows these molds and the fully draped plies on them. The molds are inspired by industrial use cases. They cover the

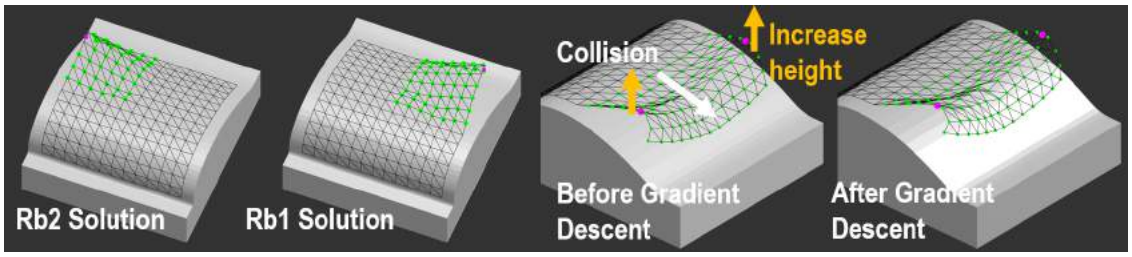

Figure 4.17: (Left to Right) Independent solutions obtained for robot-1 (RB1) and robot-2 (RB2), ply state simulated for combined solution obtained from the nominal grasp locations, solution improved using a state space search using the nominal grasp locations.

	Dimensions (mm)			Link length in undeformed mesh (mm)	Number of stages excluding stage-0
Mold	Length	Width	Height		
A	500	450	200	32	4
B	440	220	140	20	8
C	570	570	110	25	16
D	300	300	56	15	12
E	820	570	75	40	19

Table 4.1: Dimensions, length of each link in undeformed mesh, and number of stages excluding the stage-0 are provided.

basic geometries like single curvature to more complex concave and double curvature geometries. We do not use the entire mold for draping the ply. A portion of the mold around the perimeter is used for vacuum bagging, which is a part of the curing process. These unused regions are visible on the molds. The dimensions, link length, and the number of stages for each mold are provided in Table 4.1.

4.6.2 Drape Simulator

We will first discuss the results from the drape simulator being used for determining the unconformed state of the ply during the process. Shear and bending energy relationships to be used for the simulation are determined. For shear, we have used the bias extension test data, which

generates the force (F) vs. deflection (dS) curves. The shear energy relations is found using the equation $U_{shear} = \int FdS$. Bending energy, on the other hand, is found by using the beam theory. Different samples of prepreg are used to find the 1-dimensional bending under its own weight. The bending energy for a cantilever beam is given by the equation $U_{bending} = FL^3/(6EI)$, where $F, L, E,$ and I are the force applied, lever arm, young's modulus, and the moment of inertia respectively. We determine the bending angle at discrete points along the length of the sample and compute the energy stored for that point. This gives us the bending energy relationship. Exponential function is used to best-fit the data collected from the experiments to estimate these relations. The shear and bending energy relationships with respect to the respective radian angles are given in equation (4.4 and 4.5) respectively.

$$U_{shear} = 9.132 * e^{(-4.485*\theta_{shear})} \tag{4.4}$$

$$U_{bending} = 1.124e10 * e^{(-15.34*\theta_{bending})} + 0.06109 * e^{(-1.1386*\theta_{bending})} \tag{4.5}$$

The spring potential energy is adjusted to simulate the ply as closely as possible for different test cases under various conditions of grasping. Plies used to tune the simulator are 250 × 250 mm and 500 × 250 mm in size. The current version of the simulator is implemented in C++ on a computer with an Intel Xeon 3.50GHz processor and 32GB of RAM. Figure 4.18 shows the computation time for producing a solution state of the ply relative to the number of free particles in the ply. We can observe a linear increment in the computation time for the simulation with respect to the number of free particles. As numerical gradients are used in the current version of the implementation, the simulation times are high. This leads to high planning times. With analytical gradients and an adaptive mesh-based model, we can significantly reduce the simulation time.

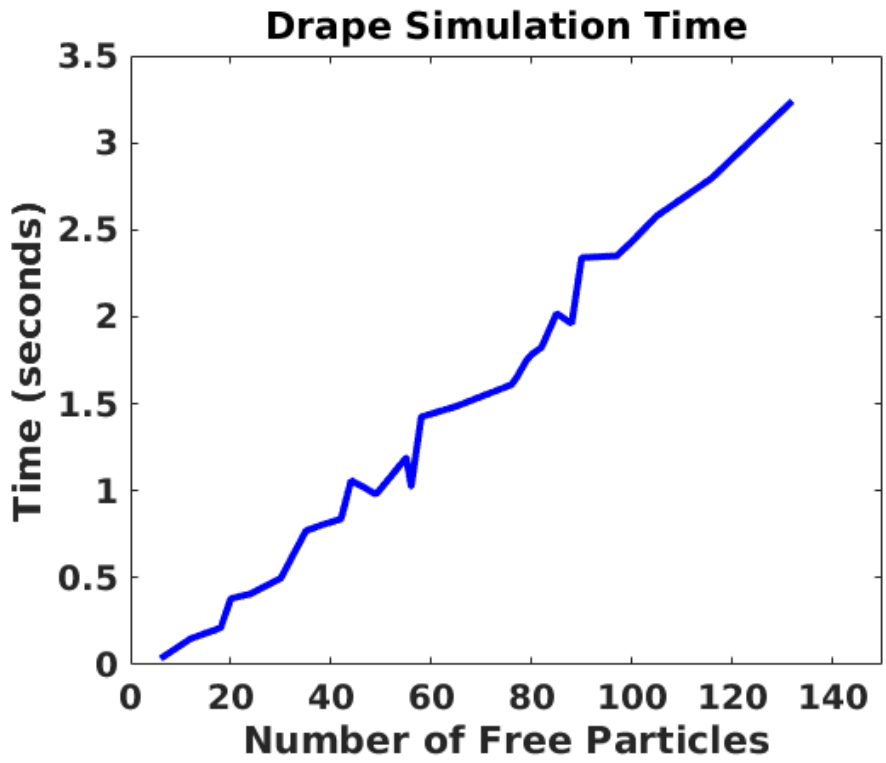

Figure 4.18: The simulation time (seconds) for one simulation run vs the number of free particles in the unconformed region of the ply.

4.6.3 Computational Performance of Grasp Planner

We benchmark the grasp planner with a brute force approach where all the nodes are evaluated to find the optimum solution. The time taken for evaluating each node is recorded by taking the average time for all simulation runs in the local assignment search. We present the number of nodes evaluated and the computation time taken for both the robots. Table 4.2 shows these results. The brute force approach is labeled as the baseline approach we are benchmarking against. The number of nodes reported is the total number of grasp points present in all the stages in the graph. MATLAB was used for the implementation of the grasp planner. We used parallel processing with 4 cores during the local assignment search to explore the discrete state space, which reduced the overall computation time. The location assignment search essentially uses drape simulations.

Computation time (seconds) and Nodes evaluated										
	Robot-1 (RB1)					Robot-2 (RB2)				
	Baseline Approach		Our Approach			Baseline Approach		Our Approach		
Mold	Nodes evaluated	Time taken	Nodes evaluated	Time taken	% Reduction in Time	Nodes evaluated	Time taken	Nodes evaluated	Time taken	% Reduction in Time
A	22.00	43.10	7.00	4.17	90.32	22.00	35.45	5.00	2.60	92.67
B	50.00	164.76	14.00	17.90	89.14	73.00	223.09	47.00	61.67	72.36
C	102.00	742.11	25.00	113.89	84.65	230.00	2041.40	66.00	386.83	81.05
D	94.00	479.35	24.00	52.51	89.05	154.00	762.38	81.00	185.49	75.67
E	300.00	695.21	193.00	211.12	69.63	210.00	785.49	76.00	125.94	83.97

Table 4.2: Comparison of the baseline approach with our approach. Results reported are for robot-1 and robot-2

Comparison of Cost of Initial Grasp Plan								
	Robot - 1				Robot - 2			
Mold	Randomly Generated	Heuristic Based	Optimal Path Cost	Percentage Improvement	Randomly Generated	Heuristic Based	Optimal Path Cost	Percentage Improvement
A	0.36	0.10	0.10	72.10	0.38	0.13	0.13	65.69
B	0.71	0.20	0.19	72.30	1.42	0.87	0.75	38.96
C	0.85	0.64	0.33	23.73	1.60	0.93	0.92	41.45
D	0.67	0.34	0.31	49.12	1.37	0.84	0.62	39.12
E	2.31	1.41	1.29	38.83	1.32	0.91	0.86	31.34

Table 4.3: Comparison of the the cost of initial grasp plan for using ordering heuristic and random generation. The percentage reduction in the initial path cost when using ordering heuristic for both the robots is provided.

We compared the cost of the initial grasp plan generated by using the ordering heuristic with a randomly generated path. Table 4.3 shows the comparison. We can observe that the initial grasp plan costs are lower for our approach as compared to randomly generated paths. The initial grasp plan costs are also closer to the optimum costs, as reported in the table. The heuristic can provide initial plans that are closer to the optimum, which further reduces the computation time to find the optimum solution.

Figure 4.19 shows the solution obtained for mold C. As we mentioned earlier, the stages are numbered in ascending order from the root node. The root node is the fully draped ply in a backward tracking search. We report the solution obtained in simulation for both the robots

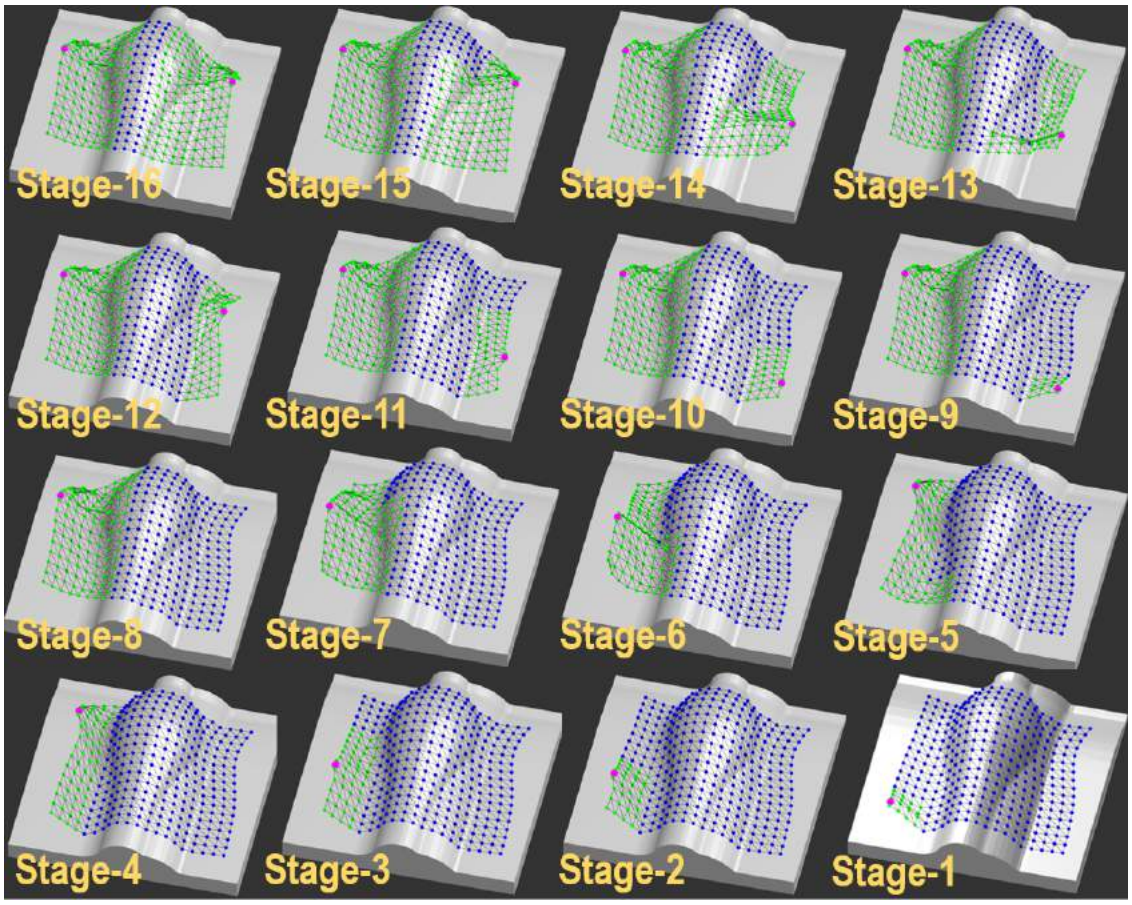

Figure 4.19: Simulation of the solution obtained for mold C. Particles in blue are part of the conformed region. Particles in green are unconformed. Grasp point at the assigned location is highlighted by a magenta marker.

in the order of the drape sequence. Particles in the conformed regions are highlighted in blue, unconformed ones in green, and the grasp point is highlighted by a magenta marker. We can see the regrasps that occur during the process.

4.6.4 Scalability

The molds we have used range from smaller to medium-sized geometries. For instance, mold A has only 4 stages in the graph excluding stage-0 with a total number of 22 nodes. Mold E, on the other hand, has 19 stages and 300 nodes for robot-1. However, the computation time reduces

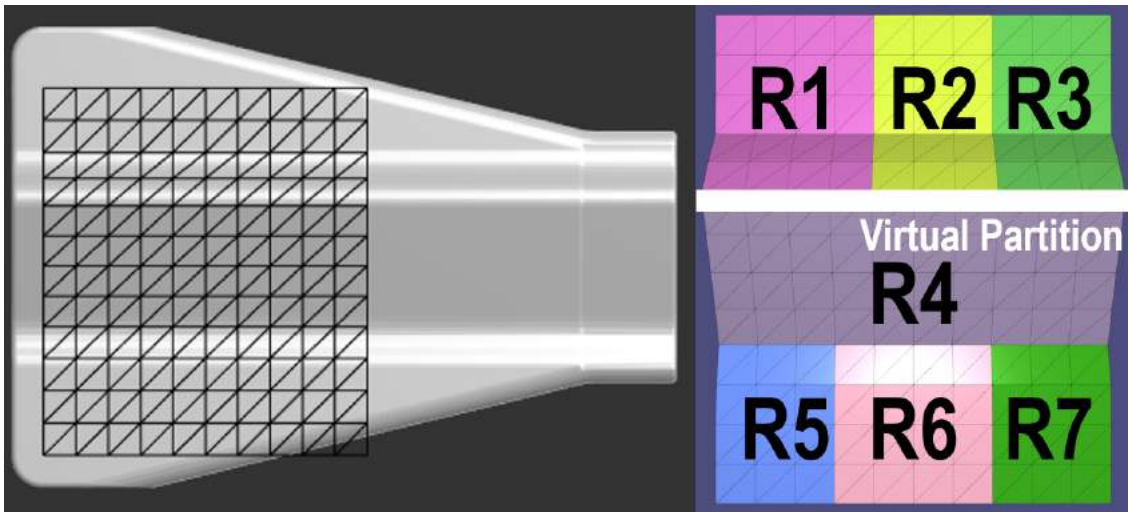

Figure 4.20: Drape simulated over the mold used for physical experiments. Virtual partitions and the sequence of regions for this mold is also illustrated.

as compared to the brute force approach by around the same factor for these molds. This shows that if we increase the size of the molds to larger-scale dimensions of over 2 meters, our approach will still evaluate a fewer number of nodes for generating the plans.

We can also have the same number of grasp points, even with a higher mesh density. This is because the gripper will physically hold the ply using a 10-15 mm edge length. So even if the cell size in the mesh is 5 mm, we can combine more particles instead of two particles as used in our approach to create a grasp point. However, as the number of grasp points increases, the computation time increases.

4.6.5 Physical Experiment

We conducted physical experiments using mold shown in Figure 4.20. The figure also shows the draped region and virtual partition on the mold used in our experiments.

Figure 4.21 shows the different simulation stages of grasp planning. Vertical motion of the mold with respect to the robots is allowed while computing placement, which offers more reachability. Thus, the mold is elevated in the physical setup. We first conveniently place the mold on the

table, and the robot placements are determined. The physical setup is shown in Figure 4.2. We set the temperature of the air to 45° C, force applied to 35 N, end-effector stiffness along the X, Y, and Z axes to $3000, 3000, 2000$ N-m^{-1} respectively, and the velocity of the robot to 15 mm-s^{-1}. Figure 4.21 also shows the stages during the layup process. The grasping robots perform regrasps and reposition the end-effector according to the plans generated by our algorithm. We assume that some form of automation will exist to feed the pre-cut plies to the cell. In our experiments, an operator aligns the sheet with mold edges and feeds the edges of the sheet to the grippers.

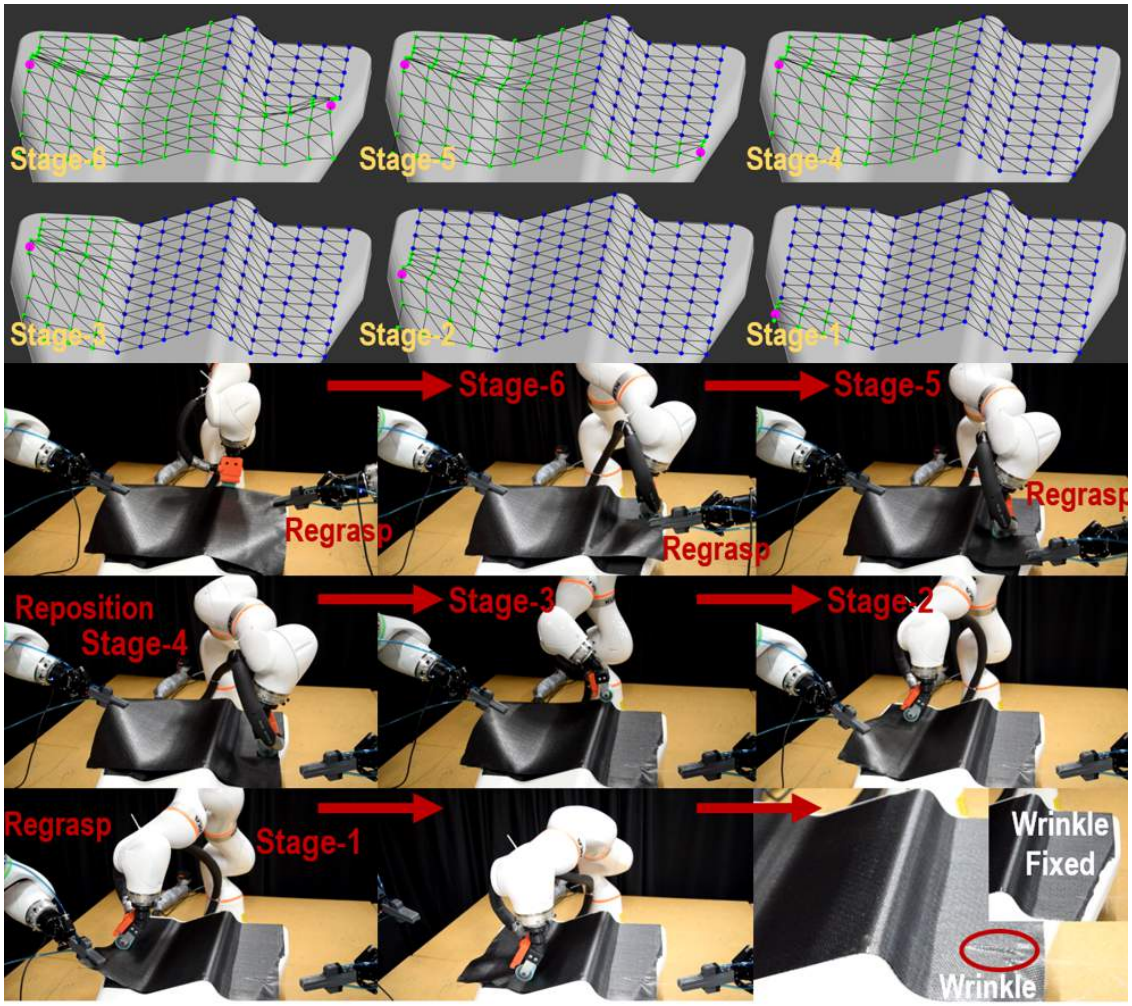

Figure 4.21: Simulated ply states in the different stages of backward tracking search are illustrated. Instances of the layup process, the repositioning and regrasping motions performed by the grasping robots are also shown in the figure. Human interference is required to fix the wrinkle generated.

There is some uncertainty between the simulated and the actual ply state. If the grasping robots do not react to the excess tension in the ply, the layup process may have defects. The grasping robots are therefore set in a compliance mode. By that, we mean that the robots are under impedance control mode, and if tension builds during execution, the robots comply and reposition the end-effector along the direction of the tension. This can prevent defects from forming during execution. Figure 4.22 illustrates the difference between a layup performed under compliance and without compliance. We observe a high-cost ply state when robots do not comply and defects are present in the layup. On the other hand, compliance mode leads to reduced defects.

Even under compliance mode, there are small defects like a wrinkle that appears. The system analyzes this defect and calls for human intervention. The operation is halted and a human repairs the defect after which the operation can be continued. The grasping robots maintain their locations under compliance mode. The operator can adjust these locations if required while intervening. The grasping robots resume from new locations. Figure 4.21 illustrates such a defect that was generated and was repaired by the human operator before proceeding with the next layer. No visual defects were present in the completed layup after human intervention. Completed layup using 15 plies is shown in Figure 4.23. Table 4.4 also lists the time taken by each operation during the process. Some of the operations are to be done only once for a mold. Human intervention time includes the time taken by the operator to place the sheet and average time taken to repair defects throughout the process. We can see that the overall human input required for the process can be reduced by a significant amount for medium scale production. For instance, manufacturing 100 parts will result in a 88% reduction in human input. These results show that the process is feasible to be automated and used for commercial purposes. The laminate was also inspected for defects. Resin distribution studies were done to tune the parameters of the process in our previous work in [121, 122]. The work also demonstrates physical experiments on other smaller molds. In

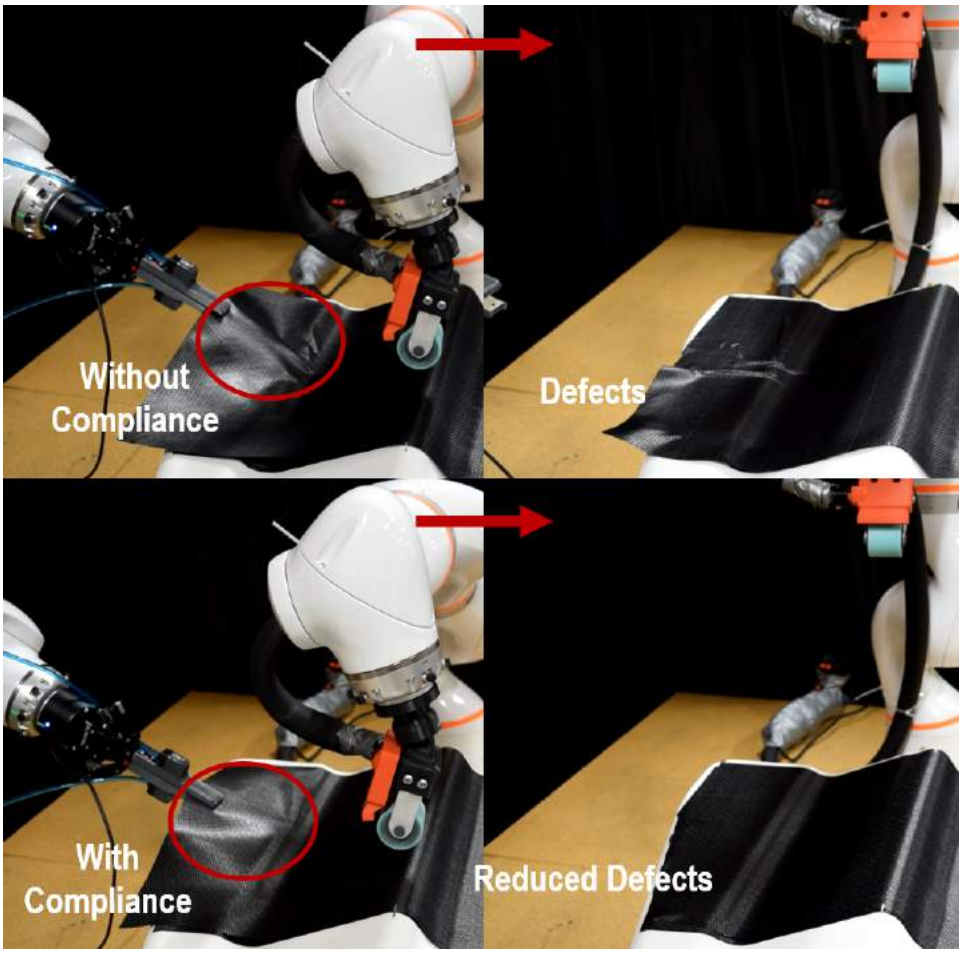

Figure 4.22: Physics based simulation has some uncertainty in predicting the real ply state. A high cost ply state is obtained as a result of absence of grasping robot compliance (feedback control) which leads to defects in the layup. The defects can be prevented by using compliance mode for the grasping robots.

this work, we perform a visual inspection for air pockets and defects and primarily present the conformity and fiber alignment results from the laser scanner and apodius system, respectively.

4.6.6 Conformity Analysis

In order to ensure that the ply is conformed well to the mold and no air pockets are present, we conducted a conformity analysis on the mold. A point cloud is generated using hexagon absolute arm integrated with a laser scanner. This point cloud is compared to the reference point cloud,

	Time (minutes)	Frequency of operation	Total time (minutes)
Expert Input	14	1	14
Planning time	6.6	1	6.6
Trial run	9.4	2	18.8
Grasp plan modification	3	1	3
Layup	8.5	15	127.5
Human Intervention	1.2	15	18
			187.9

Table 4.4: Time taken by different operations in the entire layup process.

Figure 4.23: The completed layup shown on the mold.

which represents the surface of the mold offset with the thickness of the laminate. The deviation is within tolerance. The laminate with 15 sheets was scanned and maximum deviation was found to be 0.6 mm. The tight concave regions are usually difficult to layup as air does not get completely pushed out. The curing process applies vacuum pressure on the lamina, which removes the voids present.

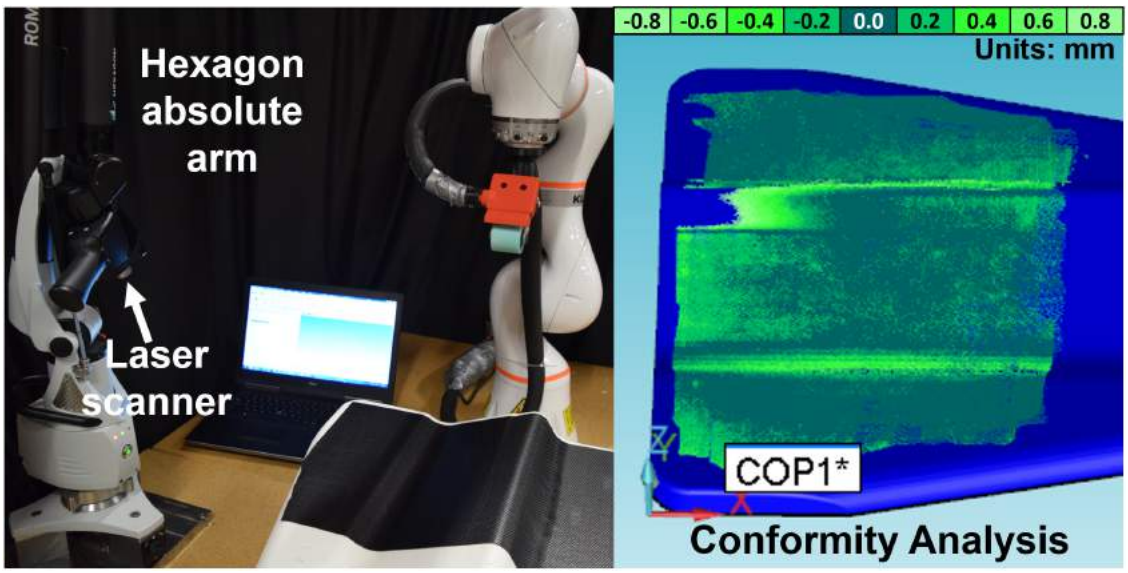

Figure 4.24: Conformity analysis using hexagon absolute arm. The color map showing the deviation of the ply with the mold surface is shown.

4.6.7 Fiber Alignment

We inspect the fiber alignment in the ply using the Apodius 2D sensor. The sensor is attached to the robot flange, and trajectories are generated so that the robot can scan the different regions of interest on the mold. Figure 4.25 shows the draping robot with apodius sensor. The sensor analyzes the image and computes the deviation of fibers. We input a reference orientation to the software, and all the fiber angles are compared against this reference. A reference orientation is specified by the design engineer. In our case, our goal was to orient the fibers parallel to the edges of the mold. The distribution of fibers with respect to this reference is obtained using the software. Respective alignments for different points on the mold are collected and analyzed for defects. The fibers did not deviate at the concave regions of the mold during layup. Fiber angles were found to be within 0.1° angle. The process parameters were appropriate and did not cause any damage to the fibers in the right concave regions.

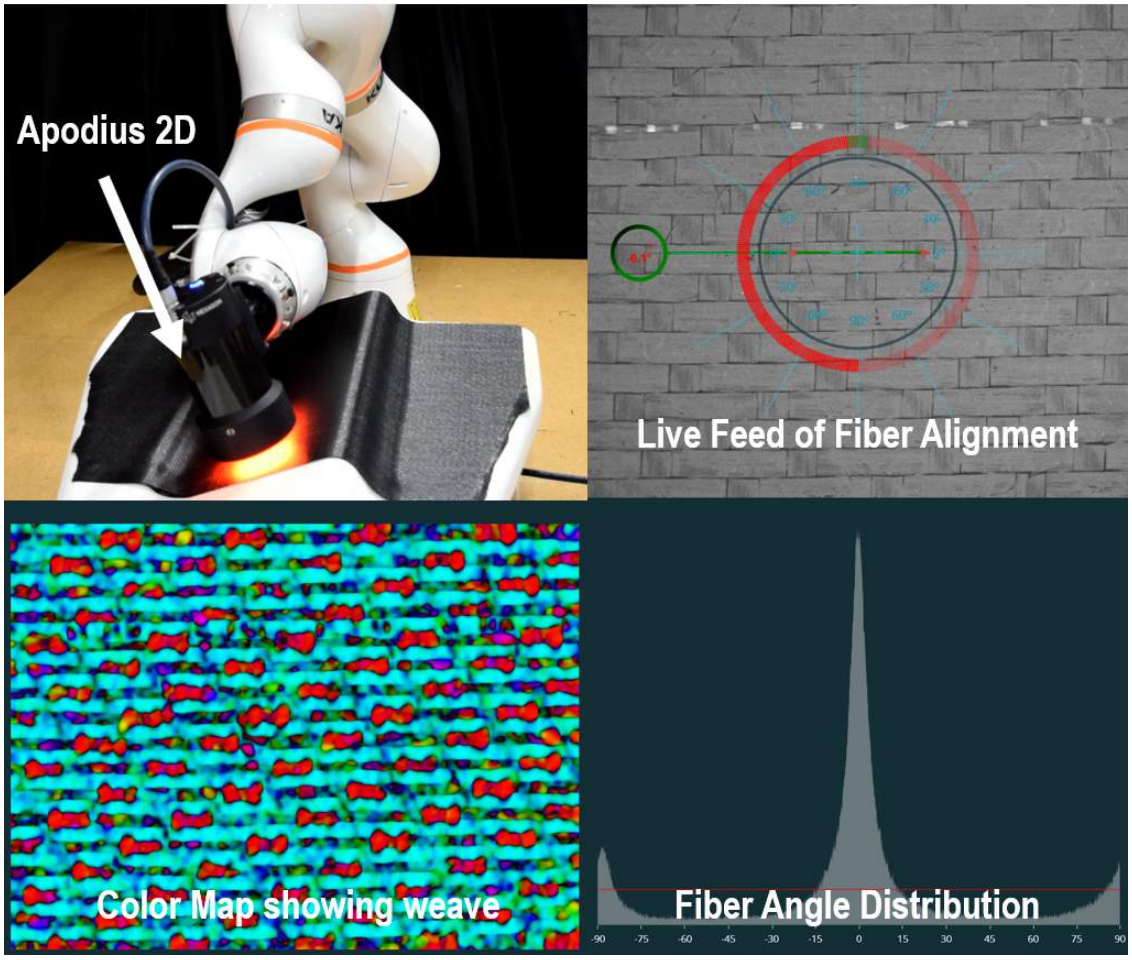

Figure 4.25: Apodius 2D sensor attached to the robot flange. Image captured by the sensor at a region of interest. Image is analyzed for determining fiber angle deviation.

4.6.8 Visual Inspection

The automated layup quality was comparable with hand layup. Air pockets or bridging were not observed.

4.6.9 Discussions

There is uncertainty in the state of the sheet due to environmental factors and material properties. It is challenging to capture these uncertainties in the physics-based simulations. The grasp plans generated needed some modifications as the sheet was coming in contact with the mold.

Translating two grasp points corresponding to stage-2 and stage-1 shown in Figure 4.21 by 20-30 mm solved the problem. Using a high fidelity simulation and machine learning models to tune the grasp planner parameters can lead to improved solutions. Drape simulation parameters like the bending and shear energy constants also cause inaccuracies in the simulation. A more sophisticated method to tune these parameters by performing experiments and using machine learning can produce better grasp plans. End-effector dimensions (e.g. roller width) influence the layup quality and sophisticated research is needed to identify correct dimensions for a given mold geometry. Trials were conducted to find suitable roller dimensions before the layup was perfected. Same grasp plans were used for all the trial runs which shows how using incorrect roller dimensions can introduce defects. Commonly observed defects during the trials were wrinkles, bridging, air pockets, and misalignment. Although, once the grasp plans and parameters are tuned, the process is consistent and several plies can be stacked without observable defects.

4.7 Summary

We addressed the generation of grasp plans for automated composite prepreg layup process. We presented a state-space search assisted by physics-based simulations to automate the grasp planning process. This approach successfully generated grasp plans for parts with varying level of complexity. The use of heuristic we developed in this work enabled us to significantly reduce computation time compared to exhaustive search.

Chapter 5

Algorithms for Identifying Feasible Workpiece Placement with Respect to Redundant Manipulator for Complex Manufacturing Tasks

5.1 Introduction

The workspace of a serial manipulator is defined as the set of positions and orientations that can be reached by the end-effector (or a tool attached to it) of the robot. The workspace for an articulated robot can be quite complex due to the joint limit and self-collision-avoidance constraints. The robot workspace includes singularities, and it is generally preferred for a robot to carry out tasks away from singularities. Different locations in the robot workspace impose different constraints on achievable forces and velocities. Variations in achievable forces and velocities can be quite large. Moreover, there might not exist a continuous path between two points in the robot workspace that are far apart. Redundancy exists when the degrees of freedom (DOF) of the manipulator is higher than that required by the process. Workspace characteristics can be quite complex for redundant manipulators, and unfortunately, there is no simple model to capture these.

Many complex manufacturing tasks such as welding, robotic finishing [93], composite layup [124, 123], painting [34], 3D printing [16], accuracy improvement [20, 19], etc. use manipulators

to follow continuous paths under constraints on the workpiece. Moreover, the end-effector or tool needs to apply certain force on the workpiece. This means that only certain portions of the robot workspace may be suitable for carrying out a task based on the force and velocity requirements. It is important to find the right location and orientation for the workpiece with respect to the robot to meet all the task constraints. This problem is called a workpiece placement problem. Figure 5.1 shows three different potential placement of the workpiece for a composite layup task. Only one of these placements meets task constraints.

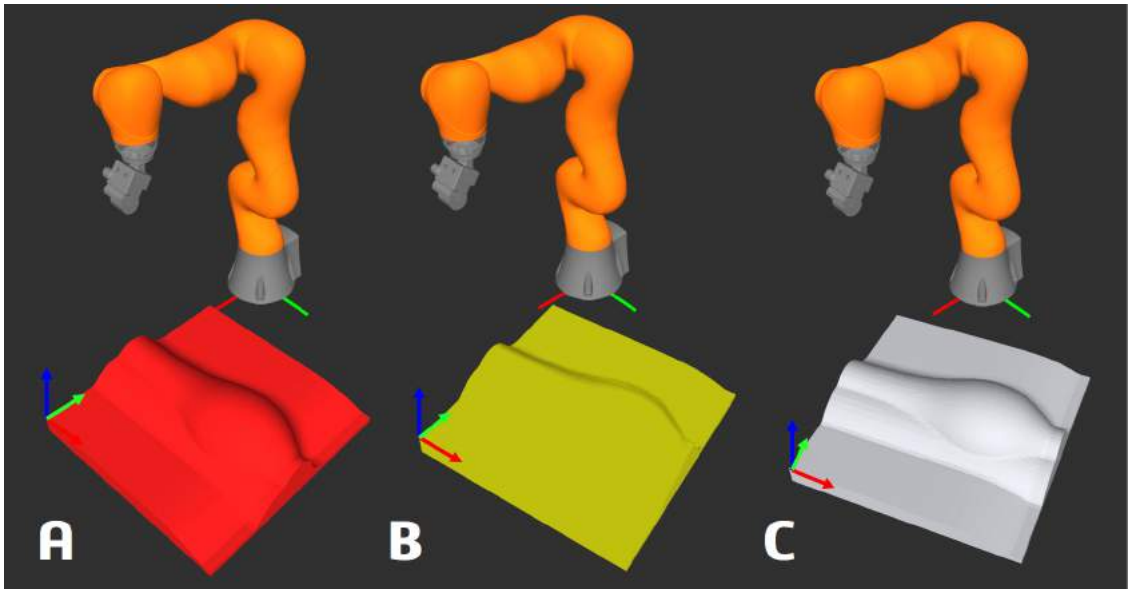

Figure 5.1: (A) Workpiece is reachable in workspace, but velocity, force, and continuity is violated. (B) It is reachable, velocity constraint is met, but force, and continuity are violated. (C) All constraints are satisfied.

Sometimes, workpiece placement problems are solved by doing a conservative approximation of the workspace. This involves including significant factors of safety in approximating robot capabilities. This means that in some cases this approach may fail to find a placement solution for a large part that fits in the robot's workspace and satisfies all the constraints. In this paper, we convert the part placement problem into a non-linear optimization problem that minimizes the constraint violations. This problem is very complex and has many local minima due to

complex interactions among constraints. Many of these local minima are not feasible solutions. This can be overcome by using a large and diverse number of randomly generated initial part poses for the optimizer. This leads to significant computation overhead. We have developed a new approach to solve this problem by successively improving the reachability and applying constraints incrementally. This helps us in navigating the search space more efficiently and reduces the number of initial part poses that are needed to find a feasible placement solution.

5.2 Related Work

The problem of determining the workpiece placement with respect to a manipulator has been extensively studied. Placement of the components for robotic assembly has been addressed in [149]. A gradient of manipulability in Cartesian space was derived. An optimization routine to maximize manipulability index (this index quantifies ease of maneuvering the end-effector), which is obtained from the Jacobian [205, 204, 203, 114] was implemented. In [191], reachability representation was precomputed and the base of the robot was placed for grasping task. Minimization of cycle time as a measure, for a manipulator was used in [186].

The paths executed by a robot on the workpiece can be represented with respect to a single workpiece reference frame. Aspragathos et al. considered a simple path and maximized the mean value of manipulability along this path in [6]. In [63], optimum robot placement with respect to workpiece was found for minimum time coordinated joint motion to improve cycle times. In [7], a detailed overview of the manipulability index, condition number, hybrid manipulator measures, and global integral measures being used in this area was given. A new measure called Manipulator velocity performance (MVR) was proposed which improved the velocity performance of the robot. MVR [52, 148] maximizes end-effector velocity with minimum required joint velocities.

In [189], a path placement for a orthoglide parallel robot was solved, by minimizing the energy consumption, actuator torques, and forces. A similar problem was solved for machining operation

in [29, 30, 54], where deflection of spindle caused due to cutting forces, was minimized. A genetic algorithm was used to provide initial guess to a nonlinear optimization solver. Deflection was obtained from Cartesian stiffness matrix [53] of the robot. As the operation only demanded 5 DOF, and 6 DOF manipulator was used, an approach to solving the placement under redundancy was shown. In a similar work [47], two multi-objective optimization loops were integrated. The first loop aimed towards fixing the redundant angle by merging joint limits, singularities, and collision constraints as a single non-differentiable objective function. A solution was then fed to the second loop which determined the workpiece placement. using a modified particle swarm optimization. Work in [77] gave a brief overview of local vs global techniques widely used in this area. The overall joint motion was minimized by formulating it as an unconstrained problem. Redundancy of the system was simultaneously resolved. Path constraint trajectories were generated for high DOF systems by successively applying constraints in [94]. A successive solution search strategy was also adopted in [183, 179].

An explicit representation of a redundant robot workspace was captured in the form of a capability map in [207, 206]. In [208], this capability map was used to position the mobile base of the robot, such that it could execute 3D trajectories. Another representation proposed in [192], captured the manipulability distribution of a robot in a quality index. This was done to favor more manipulable grasp locations in the workspace for grasping tasks.

Measures such as manipulability, velocity performance, overall cycle time, joint torques, or overall joint motion over some constraints are usually optimized. As stated earlier, approximating the constraints relaxes the problem but does not guarantee that they are met in an actual case. This limits the optimum use of the robot. Redundancy in the task was solved for simpler cases where rotation about one tool axis was permitted. In actual processes, rotations about, up to 2 axes of the tool can be permitted within tolerances. Formulating these constraints as continuous functions is needed. Moreover, redundancy needs to be fully defined and accounted in finding a solution.

5.3 Background and Terminology

We are interested in placing a workpiece with respect to the robot's base in its workspace. The robot (redundant manipulator) has a tool attached to its end-effector and needs to execute trajectories such that the tooltip travels through a set of paths ($\mathcal{S}$) defined on the workpiece. Each path $P_k \in \mathcal{S}$ ($k \in [1, 2, \ldots, m]$) is a set of waypoints. Each waypoint $p_i^k \in P_k$ ($i \in [1, 2, \ldots, n^k]$), is represented by its position and orientation. We denote ${}^r\vec{x}_{wp}$ as the pose of a frame attached to the workpiece with respect to the robot frame. A pose in the robot's workspace is represented by its position (Cartesian coordinates, x, y, z) and orientation (quaternion q_0, q_1, q_2, q_3). Finding the workpiece placement is equivalent to finding the workpiece pose in the robot frame.

We denote the orientation of a frame by X, Y, Z unit vectors and use color code red, green, and blue respectively to represent them. We consider Z axis of any p_i to be normal to the workpiece's surface, and X, Y axes are assigned appropriately depending on the task. For some tasks, they can be arbitrarily assigned, whereas some tasks specify their direction. We consider a frame attached to the end-effector or flange of the manipulator, represented with respect to robot base, by a homogeneous transformation matrix ${}^RT_E$ [38]. A tool center point (TCP) frame is attached to the tool-tip which defines the transformation of TCP with respect to end-effector ${}^ET_T$. We consider each of the waypoints on the path to be reached by the TCP satisfying the constraints. We consider the description of a robot is constituting its kinematics and dynamics model. Configuration space of the redundant manipulator is its joint-space. We denote the joint vector of variables as $\Theta = \theta_1, \theta_2, \ldots, \theta_n$ for a n-DOF manipulator. For KUKA iiwa 7 manipulator used in our work, n = 7. A manufacturing process specifies the minimum process velocity and force to be exerted during contact of the TCP with the workpiece at a given waypoint. These process requirements are set in the form of constraint violation functions (see Section-5.5.2).

5.4 Problem Formulation

Given a CAD model of the workpiece, robot, tool paths $\mathcal{S}$, and the process requirements (velocity and force constraints) as inputs to the system, our goal is to find the pose of the workpiece with respect to the robot base, ${}^{r}\vec{x}_{wp}$, such that the process constraints are satisfied. Thus, we formulate the workpiece placement problem as a constraint satisfaction problem:

$$find\ {}^{r}\vec{x}_{wp},$$

$$s.t.\ lb_i \leq\ g_i(\Theta) \leq ub_i \tag{5.1}$$

$$h_i(\Theta) \leq 0 \tag{5.2}$$

$$\forall p_i^k \in P_k, \forall P_k \in \mathcal{S}$$

Constraints in equations (5.1) & (5.2) are defined in Section-5.5.2. Here lb_i and ub_i are lower and upper bounds for the g_i constraint in consideration.

5.5 Approach

If ${}^{r}\vec{x}_{wp}$ is a solution, a feasible joint configuration can be obtained for each waypoint on the workpiece. In other words, an inverse kinematics solution exists. If we generate a random pose as an initial guess and directly enforce all constraints, the probability of finding a solution significantly decreases. It might violate the workspace of the robot and computation cycles will be spent to improve reachability. Computational overheads are large and in some cases leads to intractable time. Moreover, any optimization method is prone to local minima existing in the workspace. In this paper, we have introduced a successive solution search method by incrementally applying constraints to find the pose of the workpiece with respect to the robot. A constraint satisfaction problem is converted into an optimization problem by defining the constraint violation functions. We can quickly explore the workspace first to find the poses which are at least within position

and orientation reachability of the robot. Inverse kinematics is not feasible for these explorations as it is computationally expensive. Therefore, we precompute a capability map for our redundant robot.

5.5.1 Generation of Capability Map

5.5.1.1 Capability Map Construction

A capability map is a precomputed discretized representation of the workspace that encodes the position and orientation reachability of the robot. However, computing a high fidelity capability map for redundant manipulators by solving IK for each discrete position and orientation in the workspace is computationally intractable. We have developed a sampling-based approach to compute the capability map $\mathscr{C}$ that is computationally efficient.

1. *Sampling in Robot Joint Configuration Space:* We randomly sample a joint configuration Θ and compute the pose of end-effector with respect to the robot base using forward kinematics. The position of end-effector is mapped to nearest voxel center and marked reachable in the underlying data structure. The orientation of end-effector is stored as three unit vectors each along X, Y, Z axes of the end-effector for that particular joint configuration Θ. We also store the manipulability index.

2. *Sampling in Robot Workspace:* In order for us to ensure uniformity throughout the capability map, we evaluate the number of samples in a neighborhood ($\mathscr{N}$) of each voxel. The density of sparsely sampled voxels is improved by adding samples by explicitly solving IK.

3. *Sampling Near Workspace Boundaries:* We detect and label voxels that are on the boundary after initial sampling. We check neighbors of the current boundary voxel using inverse kinematics to confirm if it actually on the boundary or not. If needed, we expand the boundary by adding new voxels.

4. *Constructing Orientation Cones:* The orientation in each voxel is represented by three unit vectors each along X, Y, Z axis. All the orientation vectors belonging to the voxel along each axis can be represented by a principal axis and angle. For this, we use a hierarchical clustering algorithm to compute the minimum number of cluster required to encapsulate all the vectors. Each cluster is represented by a centroid of these vectors as principal axis and an angle which defines the bounding cone [10].

5. *Filtering Singularities:* We can explicitly filter voxels violating the singularity constraint using the mean value of manipulability index we stored for each voxel. Figure 5.2 shows the manipulability index and capability map from a top view for a KUKA iiwa 7 robot workspace. Workspace is sliced by a plane parallel to XY plane. Regions in green are the usable workspace of the robot which avoids singular regions shown in blue. Regions shown in red are completely out of reach.

The capability map is computed once for a robot and used for all placements associated with that robot. We can adapt the capability map with any tool which is attached to the robot end-effector or flange.

5.5.1.2 Adaptation of Capability Map

We need to account for redundancy or tolerances for applications where DOF required by the process is less than DOF of the manipulator. We introduce the concept of 1 axis and 2 axis tolerances. Consider 1 axis illustration in Figure 5.3. Y-axis of TCP and waypoint need to be perfectly aligned, but due to cylindrical geometry of roller, there is a tolerance permitted in rotating the TCP about Y. This generates a 2D cone about Z of TCP. 2 axis illustration, on the other hand, allows tolerance in rotation about X and Y both of the TCP, which generates a 3D cone about Z of TCP.

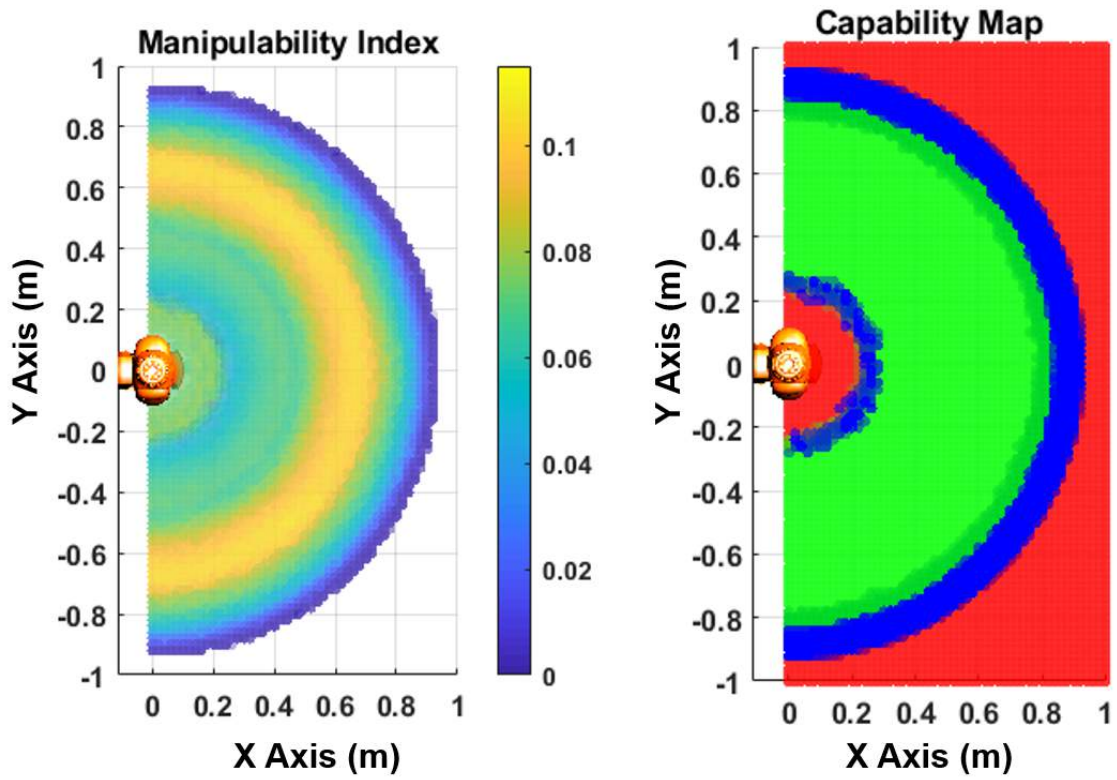

Figure 5.2: Filtering workspace singularities (shown in blue) using capability map. Feasible region (green) is used to find the pose for workpiece.

Capability map was constructed for the end-effector. We now discuss the way to use this map when any tool is attached to end-effector. If no tolerances are permitted, the TCP needs to be perfectly aligned with a waypoint on the workpiece. In this case, a rigid body transformation ${}^{E}T_{T}$ is applied to the waypoint and position and orientation of end-effector can be obtained. The corresponding voxel can be queried if the orientation is enclosed in any orientation cone present in that voxel. In order to account for tolerance, ${}^{E}T_{T}$ is used to discretely apply rotations about the axis of TCP which permits tolerance. For each discrete rotation, end-effector can be mapped to a voxel. A set of voxels and corresponding end-effector orientations are thus obtained. Hence, all permissible orientations of TCP can be mapped to a set of voxels and corresponding orientations. Figure 5.3 illustrates the voxels traced at end-effector due to 1 axis and 2 axis tolerances. Hence, the map is generic for any tool given the rigid body transform. In our approach, we use only Z

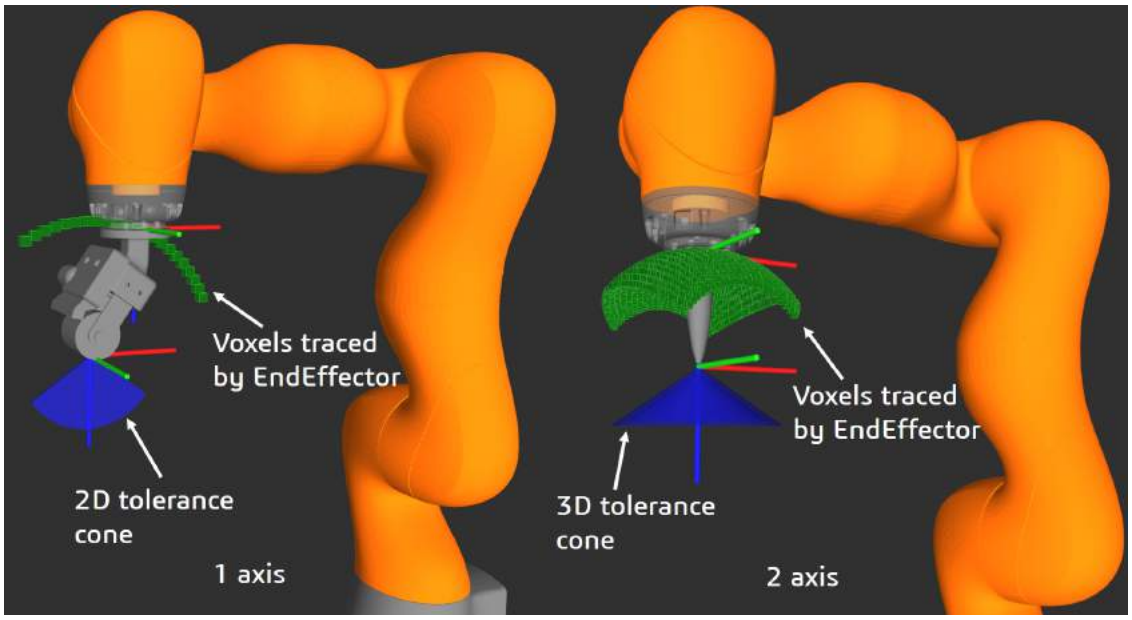

Figure 5.3: 1 axis (left) and 2 axis(right) tolerances are shown. Applying rotation to TCP to account for tolerances, causes the end-effector to trace set of voxels. These voxels are shown in green at the end-effector.

orientation of the end-effector for computing orientation cones. This is based on the assumption that, as most modern robots are equipped with a last joint which can almost rotate 360°, we can focus on orienting the Z axis of TCP and X, Y axes can be oriented with the rotation of the last joint.

5.5.2 Constraint Violation Functions

We define all the constraints as continuous violation functions. These functions are developed for any non-linear optimization algorithm to compute well-behaved gradient and hessian. Θ is the decision variable over which these violations are enforced. If we can find a solution Θ for each waypoint on the path for a given ${}^{r}\vec{x}_{wp}$ then it is a valid solution pose. Inverse kinematics is solved under constraints to find these joint configurations (Θ) for each waypoint. We describe the constraints below.

1. *Boundary Constraints:* The physical capability of the robot enforces constraints on joint position ($\Theta_{lb} \leq \Theta \leq \Theta_{ub}$), joint velocity ($\dot{\Theta}_{lb} \leq \dot{\Theta} \leq \dot{\Theta}_{ub}$), and torque limits ($\tau_{lb} \leq \tau \leq \tau_{ub}$)

2. *Workspace Constraints:* Some process requirements restrict the pose of the workpiece. Let, the feasible set for pose of the workpiece be $\mathscr{X}$. Therefore the workspace constraint requirement is ${}^{r}\vec{x}_{wp} \in \mathscr{X}$.

3. *Instantaneous and Transition Velocity Constraints:* At any given configuration (Θ), we need to make sure that the velocity limit of the joint variables are sufficient to execute the tool path at the minimum velocity required by the process at every waypoint. The instantaneous velocity constraint is given in equation (5.3), where $\mathscr{J}$ [38] is the geometric jacobian and $\vec{v} =< \dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z >$ is the velocity of the TCP. However, we need to ensure that the robot can maintain the joint velocity requirement while moving the TCP through the waypoints. Let us assume that the TCP takes t_i time to transition from p'_{i-1} to p'_i as the robot moves from Θ_{i-1} to Θ_i in the joint space. Therefore, to maintain a feasible velocity transition, we need to enforce the transition constraint defined in equation (5.4) while solving for Θ_i.

$$\mathscr{J}^{-1}\vec{v}_i - \dot{\Theta}_i \leq 0 \tag{5.3}$$

$$\dot{\Theta}_{lb} * t_i + \Theta_{i-1} \leq \Theta_i \leq \dot{\Theta}_{ub} * t_i + \Theta_{i-1} \tag{5.4}$$

4. *End-Effector Force:* Different processes may require the TCP to apply force on the workpiece. We need to ensure that force requirement is permissible by the joint torque limits. For most manufacturing applications, a constant velocity is used during the process and inertial effects can be assumed to be negligible. By ignoring the inertial effects, we can use static force equilibrium to estimate the joint torque. Under this assumption, we can define

the torque constraint as equation (5.5) using the principle of virtual work [38], where $\vec{f}$ is the force, moment vector.

$$\tau_{lb} \leq \mathscr{J}^T * \vec{f} \leq \tau_{ub} \tag{5.5}$$

5. *Continuity:* We need to ensure that the configuration space trajectories are continuous during the process. Infinitely many inverse kinematic solutions exist for redundant manipulators. For any given path, configurations Θ_{i-1} and Θ_i need to be close to each other to ensure continuity. For continuous configurations, Jacobian matrices are similar to each other. We express this constraint in equation (5.6).

$$-corr(\mathscr{J}_{i-1}, \mathscr{J}_i) + 0.9 \leq 0 \tag{5.6}$$

   We denote the correlation coefficient [33] between two matrices with corr(). For large differences in Θ_{i-1} and Θ_i, even when they lead to identical forward kinematics solutions, the Jacobian matrices have low correlation value and are dissimilar.

6. *Collision:* We represent the rigid bodies as a collection of spheres and evaluate collision score by computing the penetration depth of sphere pairs [171]. We incorporate collision as an equality constraint.

7. *Singularity:* Manipulability index σ [205, 204, 114] at any Θ_i, can be computed as follows:

$$\sigma = \sqrt[2]{det(\mathscr{J} * \mathscr{J}^T)} \tag{5.7}$$

det is the determinant of a matrix. $\sigma = 0$ indicates singularity. Manipulability improves as σ approaches higher number. We conservatively define the constraint as given by equation (5.8) to stay away from singularity.

$$-\sigma + 0.05 \leq 0 \tag{5.8}$$

5.5.3 Solution Strategy

The constraints we are enforcing include position, orientation, singularity, velocity, force, and continuity. As stated, in our approach we successively find a solution pose by incrementally applying constraints. So let us consider four levels of reachability constraints.

- *Reachability Level-0 (RL-0):* All waypoints for a given workpiece pose satisfy position and singularity constraints using the capability map.
- *Reachability Level-1 (RL-1):* All waypoints for a given workpiece pose satisfy position, singularity, and orientation constraints using inverse kinematics.
- *Reachability Level-2 (RL-2):* Valid solution to equation (5.9) which solves for exact inverse kinematics should exist for all waypoints, for a given pose. Constraints for this equation are singularity and orientation.
- *Reachability Level-3 (RL-3):* All waypoints for a given workpiece pose should now satisfy velocity, force, and continuity constraints in addition to meeting RL-2. Pose that meets RL-3 is a solution pose for the workpiece.

5.5.3.1 Generating Poses that Meet RL-1

A few random poses can be generated using the capability map. RL-0 is automatically met if all waypoints for a given pose of workpiece belong to the feasible region defined by the capability

map. We can evaluate if waypoints for this pose meet RL-1 by querying $\mathscr{C}$. This query will check if the orientation vector belongs to any cone in the corresponding voxel for every waypoint. If no violations occur, we can store that pose as it meets RL-0 and RL-1. If waypoints for a random pose violate RL-1, an optimization or search based routine can be used to explore the workspace and alter the random pose where RL-1 is met. In our implementation, a Best-First Search (BFS) algorithm [110, 168] is used for this purpose. In BFS we compute a search tree having the random pose violating RL-1 as a root node in the tree. The children nodes in the tree are computed by altering one of the values in the random pose by a delta increment. For example, the x coordinate of the random pose is increased and/or decreased by δx, this constitutes towards translating the part in along X-axis. Similarly, the tree is populated with children nodes that alter the parent node by delta increments. The next best node expanded is determined in a best-first manner that minimizes the number of waypoints on the part that do not meet RL-1. Finally, the search encounters a node in the tree that satisfies all the constraints of RL-1.

5.5.3.2 Generating Poses that Meet RL-2

Capability Map is a discrete representation of robot workspace. Thus, it only provides an estimate of meeting these constraints. As a high fidelity capability map cannot be constructed in practice, we solve an exact IK for all waypoints at each pose that meets RL-1. This makes sure if the waypoints truly meet position, orientation, and singularity constraint. Inverse kinematics is solved using the equation (5.9) with position, orientation, and singularity constraints enforced. $\mathscr{PE}$ computes pose error of a single waypoint with the TCP. Let us consider, $\vec{e}$ as the 6×1 error vector comprising of position and orientation errors. Then the pose error is computes as $0.5*(\vec{e}*W*\vec{e}^T)$. W represents the weight matrix. In our case, the position and orientation components of $\vec{e}$ are given weights of 1 and 0.25 respectively. $\vec{e}$ is computed taking into account tolerance information as angles between unit vectors along X, Y, Z of the TCP. Let us consider $\vec{x}_{init}$ to be the initial

pose guess generated by this approach. We will now discuss the approach to generate a solution that meets RL-3 by using this $\vec{x}_{init}$.

$$\Theta_i \leftarrow \arg\min_{\Theta}(\ \mathscr{PC}(\ \Theta, \Theta_{guess}, p_i^k, {}^r\vec{x}_{wp}, robot \ ) \ ) \tag{5.9}$$

$$s.t. \ lb_i \leq \ g_i(\Theta) \leq ub_i$$

$$h_i(\Theta) \leq 0$$

$$\forall p_i^k \in P_k, P_k \in \mathcal{S}$$

5.5.3.3 Generating Poses that Meets RL-3

This is the last step in our approach of successively searching for the solution. As stated earlier, a solution pose is valid if a joint configuration Θ can be found for each waypoint on the workpiece which meets RL-3. Θ is found by solving an inverse kinematics problem given by equation (5.9) with all the constraints enforced. Θ_{guess} can be taken as a joint configuration for a previous waypoint on the path. This ensures that the initial guess is close enough to the solution if it exists. Position coordinates are in meters. Orientation angles are in radians.

So far, we just considered one waypoint. We can compute violation cost ($\mathscr{VC}$) over all the waypoints, and then find a solution pose which meets RL-3 by solving the following non-linear optimization problem:

$${}^r\vec{x}_{wp} \leftarrow \arg\min_{x}(\ \mathscr{VC}(\ \vec{x}, \mathcal{S}, \vec{x}_{init}, robot \ ) \ ) \tag{5.10}$$

$\mathscr{VC}$ is computed by aggregating the pose errors for all waypoints, where no solution existed for Θ when equation (5.9) was solved. A good initial pose is provided to a non-linear optimization method solving equation (5.10). The initial pose was randomly generated using the capability map and improved to meet RL-0, RL-1, and RL-2. Providing a good initial pose can significantly

reduce the computation time and increase the success rate of finding solutions. We use an interior-point algorithm to solve the non-linear equations (5.10 & 5.9) [200, 12, 71, 210]. We now discuss the performance of our approach.

5.6 Results

5.6.1 Test Workpieces

We use five workpieces of different geometry and size for illustrating our approach. Figure 5.4 shows the CAD models of these workpieces with paths to be executed. (A) represents a mockup of GE90 engine turbine fan mold, (B) and (C) molds for 3D printing body armor and a car hood, (D) is a hemisphere with a spiral path to be executed, and (E) is a replica of a mold used for composite sheet layup by Ascent Aerospace. A,B,C,D, and E have dimensions 300x170x93 mm, 218x155x135 mm, 300x356x80 mm, 190x190x60 mm, and 570x570x160 mm respectively. A, B, C, and D consist of a single path with 163, 55, 109, and 47 points. Finally, workpiece E has 50 paths for composite layup throughout the surface, and 441 points in total.

5.6.2 Evaluation of the Approach

We benchmark our approach compared to a baseline approach that generates a set of random poses and directly uses them with the non-linear optimization problem in equation (5.10). In contrast, our approach successively searches for a solution by incrementally applying the constraint violation functions. Table 5.1 gives the computation time taken by the three steps during this search which find poses to meet RL-0 & RL-1, RL-2, and RL-3 respectively. We give a comparison between these two approaches, on the basis of success rates based on 50 random initial poses and average time taken for each run or instance. Table 5.2 gives this comparison. When the success rate is low, a high number of random initial poses are needed to find a feasible solution.

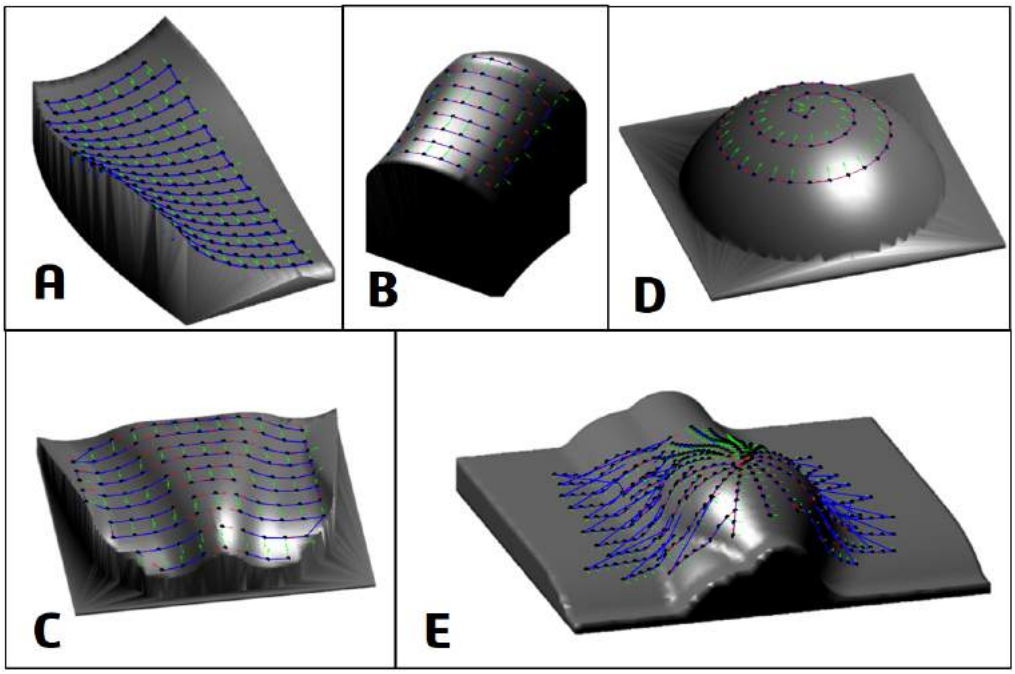

Figure 5.4: A,B,C,D, and E workpieces along with the paths.

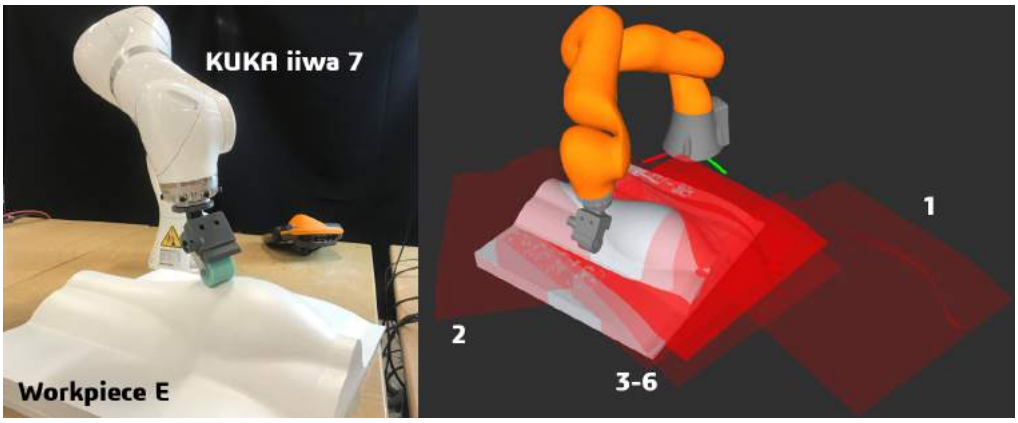

Figure 5.5: (Left) Physical setup used. (Right) Workpiece pose labeled 1 meets RL-0. 2 meets RL-1 and RL-2. 3-6 are iterations conducted by non-linear optimization. Workpiece colored in grey meets RL-3. A shorter version of the video is available at: `https://www.youtube.com/watch?v=aKGL47-xb6s`

We can also estimate the total number of initial random poses that would be necessary to achieve a 99% probability of success of finding a solution. This estimate can be used to compare the computation time. Let N be the number of initial random poses required. N is found by

Table 5.1: Average time in seconds and standard deviation per run for different steps in successive approach.

	RL-0 & RL-1		RL-2		RL-3	
Workpiece	Avg time (sec)	Std Dev	Avg time (sec)	Std Dev	Avg time (sec)	Std Dev
A	3.3	2.4	4.42	0.62	13.98	1.98
B	2.22	3	1.6	0.45	12	0.7
C	14.2	5.38	4	1.002	187.98	18
D	1.43	1.002	2	0.44	330	73
E	146	55	12.3	1.782	1185	425

Table 5.2: Success rate and computation time for both approaches

	Baseline Approach		Our Approach	
Workpiece	Success rate %	Avg Time (min) per instance	Success rate %	Avg Time (min) per instance
A	42	13.58	88	0.36
B	36	8.95	84	0.26
C	12	18	25	3.43
D	24	9.8	36	5.56
E	0	NA	15	22

Table 5.3: Estimated number of initial poses required for 99% probability of success & estimated computation time

	Baseline Approach		Our Approach		
Workpiece	Estimated # of initial poses	Total time (min)	Estimated # of initial poses	Total time (min)	% Time Reduction
A	9	122	3	0.96	99
B	11	98.45	3	0.7	99
C	36	648	16	28	96
D	16	156	10	36	77
E	inf	NA	28	163	NA

solving $1-p_f^N = 0.99$. p_f represents the probability of failure of finding a solution. We estimate p_f based on the samples which do not lead to a solution. For the baseline approach, the corresponding

total time can be estimated by multiplying N with average values of computation time. For our approach, we compute the average time taken by each successive step RL-0 & RL-1, RL-2, and RL-3. The failure probability at each step is considered for estimating p_f. These results are shown in Table 5.3. Computations were done using MATLAB on a computer with an Intel Xeon 3.50GHz processor and 32GB of RAM. An efficient implementation can further reduce the computation time. We also studied the increase in computation time taken by our approach for two cases. Increasing the density of points along the same path, and scaling the part by a factor of 1.1,1.2, and 1.3. This was done for workpiece C. Linear increment in computation time was observed for both cases.

5.6.3 Physical experiments

Workpieces E and D were used for physical experiments. E is used in automated composite layup where the robot uses a roller tool. Robot traces a spiral path on workpiece D using a probing tool. Velocity and force requirements for workpiece E were set to 125 mm-s^{-1} and 50 N and for D were set to 200 mm-s^{-1} and 20 N. In both cases, the robot was able to execute the paths meeting the constraints. Figure 5.5 shows the physical setup with workpiece E. It also shows the iterations incurred in our approach.

5.7 Summary

We have formulated a workpiece placement problem for challenging manufacturing tasks as a non-linear optimization problem over constraint violation functions. We have demonstrated that this problem can be solved by using a successive solution search-based approach by incrementally applying constraints. This enables incremental refinement of solutions and reduces conflicts among competing constraints. Simulations and physical experiments have been used to validate the approach.

Chapter 6

Algorithms for Finding Optimal Sequence of Mobile Manipulator Placements For Automated Coverage Planning Of Large Complex Parts

6.1 Introduction

A mobile manipulator is a combination of a mobile base and a manipulator that enhances the capabilities of manipulation by providing additional degrees of freedom for locomotion. The overall workspace of the mobile manipulator is larger than conventional fixed base manipulators. Mobile manipulators are widely used in a variety of applications like spraying painting [45], disinfection [182], machining [61], pick and place [180, 184] etc. Another important suitable application is 3D reconstruction or part inspection. The mobile manipulator is used for positioning a sensor around a large complex part and obtaining information about the part (see Figure 6.1). The use of 6DOF manipulator enables sensor positions and orientations to be selected to avoid collisions around the part and mobile base allows an extension in the dimensions of the part being scanned. The mobile manipulator is a good choice for using a RGB-D camera as a sensor for scanning the part for 3D reconstruction or inspection and obtain the entire pointcloud of the part.

Figure 6.1: A KUKA mobile manipulator operating on a large part. (source: KUKA)

The mobile base moves to a location near the part and the manipulator uses a set of locations, also called viewpoints, to to align the camera and capture pointclouds for a region of the large part. The location of the mobile base is fixed until the manipulator obtains the necessary data. The base is then relocated to capture a different region on the part. Figure 6.2 illustrates the mobile manipulator at two different base locations and the corresponding captured regions on the part. The optimal mobile base sequence planning aims at finding a sequence of mobile base locations that are required to cover the entire part with the minimum number of repositioning and execution time.

An increase in number of repositioning in the sequence increases the overall execution time. The base has to accelerate and decelerate before relocating which is a time consuming process. There is an associated uncertainty in the position of the mobile base each time the base is moved. Uncertainty needs to be compensated by online localization methods which also increase the execution time. A incorrectly selected base sequence will require frequent base movements by

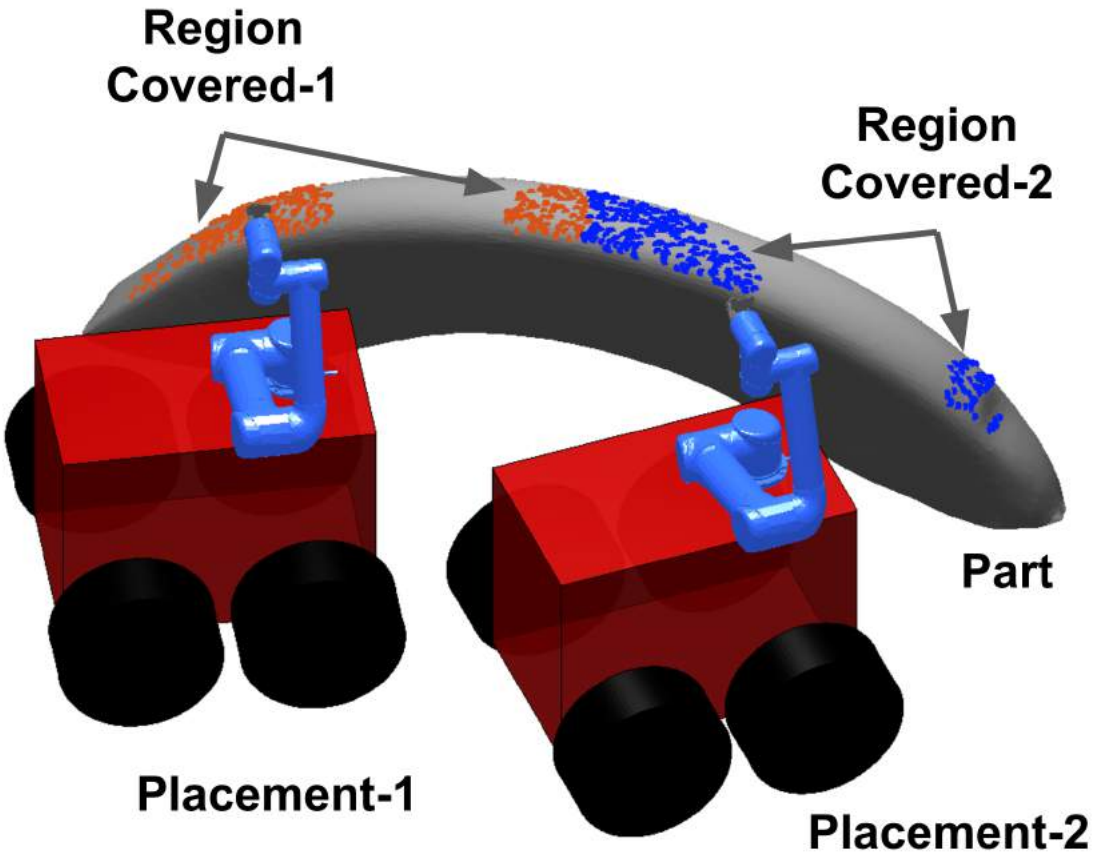

Figure 6.2: Mobile manipulator is shown at two different placements 1 and 2 near a large part. The area covered corresponding to the two locations is highlighted as patches on the part.

covering regions inefficiently when larger area can be covered with fewer motions. Therefore, it is important to prune out unnecessary base repositioning and find an optimal sequence.

The base sequence planning problem is challenging due to a very large number of feasible base locations that need to be evaluated around the part. The viewpoints also need to be selected for a given base location for the manipulator. The larger number of combinations between a base location and viewpoints that can be selected makes searching for a solution in the feasible space as a computationally hard set cover problem. Additional robot kinematic and dynamic constraints also need to be incorporated. The constraints need expensive inverse kinematic computations that add to the overall search complexity.

In this work, we develop a tree search based method for finding the sequence of mobile base locations. We make the following contributions: 1) A sampling strategy to sample the initial set of mobile base locations around the part using a fast query capability map that avoids expensive inverse kinematic calculations, 2) A branch guiding heuristic that allows us to choose the next base location which maximizes the rate gain of information, and 3) A branch pruning heuristic that allows us to remove the branches which will lead to sub-optimal solution in order to increase efficiency, and 4) A diversity based sampling method to evaluate base locations from a different branch in the tree and encourage explorations during the search. Combined, the techniques provide a computationally efficient way to find optimal solutions to the base sequence problem. Our work is applicable to a variety of tasks where optimal base sequencing for mobile manipulator is needed such as spray painting, sanding, pick and place, etc.

6.2 Related Work

Part coverage using depth camera is widely used for inspection and 3D reconstruction. The methods generally sample viewpoints around parts using different strategies and then plan a path through the viewpoints. The problems are either solved individually or together. Glorieux et al. used a targeted viewpoint sampling with an optimization strategy to get the viewpoints in [70] and reduce the inspection process time. Bircher et al. presented an online method in a receding horizon fashion. The method used sampling possible future viewpoints in a geometric random tree [21]. The work done in [2] presents a survey on coverage planning algorithms. Siyan-Dong et al. [49] developed a multi-robot collaborative coverage planning method based on travelling salesman problem (TSP). View point planning was also solved using reinforcement learning in [44, 109]. The other widely used approaches for solving coverage path planning are Lin-Kernighan traveling salesman heuristic [76], rapidly exploring random trees [111], probabilistic road map [98] and their modifications. The work done in [157] generated a coverage path planning concurrently

while generating the viewpoints. These methods are very efficient when using a single manipulator and a part. However, we are using a mobile manipulator to cover large parts which makes the problem challenging due to the additional degrees of freedom provided and increase in size of the search space.

There have been researches done on the base placement of a mobile manipulator as well. Du et al. [51] used the manipulability index as an objective to determine a suitable placement. The work done in [24] formulated the robot motion planning and base placement problem together as a quadratic program. The method found robot trajectory for the mobile manipulator using inverse kinematics branching. The work done in [201] studied the optimal base sequencing problem for handling multiple tasks given to the mobile base. They also took base uncertainty into account in their approach. Other groups have also made advances in the task seqeuncing area in [1, 138]. Researchers in [61] took an optimization based approach to position the base for a single placement during milling operation. They optimized for mean stiffness performance index.

Researches have also made advances in encoding the robot reachability in data structures to make the placement process more efficient. Reachability analysis was used to estimate the inverse kinematics of a humanoid robot and robot placements were determined in the work [190]. Vahrenkamp et. al. also conducted a series of work [193, 191] on reachability analysis and its application. The base positions with high probability of reaching a target pose can be efficiently found from the inverse reachability distribution. Our previous works in [127, 120, 129] also used capability maps to accelerate the base placement computations. We used a hierarchical approach to provide high quality initial guess to an optimization solver for determining feasible placements. The work done in [207, 206] used a capability map to encode the reachability of the robot in a discrete representation. In [208], they extended their work and used the capability map as an approximation to the robot inverse kinematics (IK) to find suitable placements for a mobile base. The base placement methods aim at finding a single placement for a given task. The methods

deal with very complex constraints for the task. We are interested in finding a sequence of base placements for the part coverage in this paper instead of a single placement.

6.3 Background and Terminology

The task for the mobile manipulator is defined as a scanning coverage of an arbitrary surface $\Gamma \in \mathbb{R}^3$. Let $l \in \mathbb{R}^6 = \{x, y, z, \alpha, \beta, \gamma\}$ represent the general pose in the Cartesian space where (x, y, z) are the position coordinates and (α, β, γ) are the orientation coordinates. Each viewpoint around the part is represented by a pose $p \in \mathbb{R}^6$. The locations of the mobile base are represented by a restricted pose vector $s = \{x, y, \alpha\}$ as other coordinates of the pose are either constant or zero. The restricted pose allows the mobile manipulator to move longitudinally and laterally on a flat floor and yaw about the vertical axis. Figure 6.3 illustrates the mobile manipulator in front of a part with the permitted degrees of freedom.

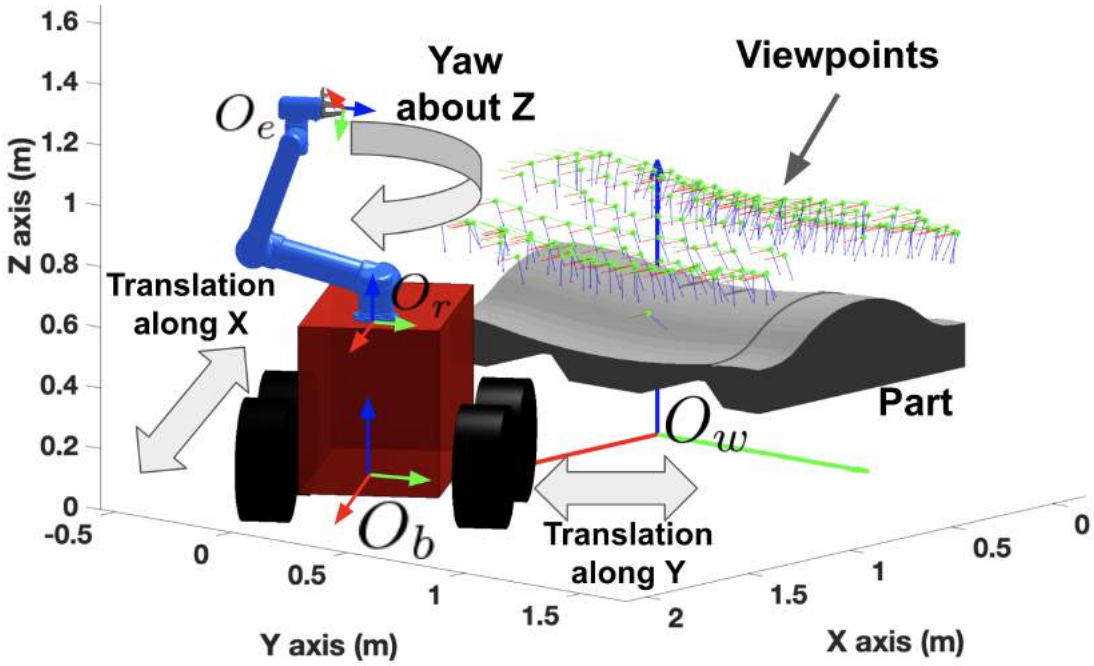

Figure 6.3: The degrees of freedom of the mobile base are shown. Few of the frames are also illustrated with the convention X (red), Y (green), and Z (blue) unit direction vectors. An example large part with viewpoint frames sampled over it is also illustrated.

The mobile robot, camera, and part used are either stand alone or a sequence of rigid bodies in the Cartesian space. Each rigid body is localized by position and orientation in the Cartesian space using frames. A frame O has a position associated with its origin and an orientation for the unit direction vectors X, Y, and Z. Figure 6.3 illustrates the frames representing the world reference O_w, mobile base O_b, robot base O_r, and the robot end-effector O_e out of many used in the representation. We use the convention X (red), Y(green), and Z (blue) colors for the frame axes in this paper. A homogeneous transformation matrix T will represent the relationship between a pair of frames [176]. For instance, the position and orientation of robot end-effector with respect to the world frame can be given by the transform ${}^{w}T_{e}$. A pose is essentially visualized by using frames in the space.

The set of viewpoints around the part is represented as $P = \{p_1, p_2, \ldots, p_m\}$. Figure 6.3 shows a sampled viewpoint frames over a part in the scene. The set of mobile base locations as $S = \{s_1, s_2, \ldots, s_n\}$. A mobile base location s_i will correspond to a set of viewpoints $P_i = \{p_1, p_2, \ldots, p_k\}$ that the manipulator will be able to reach. The set of viewpoints P_i will cover an area set a_i over the part which we represent as the set of points seen by the camera from the pointcloud $\mathscr{P}$ of the part. The number of points are then given by cardinality of the set a_i as $|a_i|$. A camera simulation model is used by our method to estimate the coverage a_i. The simulation model is discussed in detail in Section-6.5.2. Hence each base location s_i can be mapped to the area captured on the part a_i using the viewpoints.

The serial link manipulator used on the mobile base is connected by a series of revolute joints. A configuration vector $\vec{q}$ describes the joint angles for each joint. For a K degree of freedom system, the vector $\vec{q} \in \mathbb{R}^K$. We use a 6DOF Universal Robot UR5 arm for the manipulator and an inspector bot for the mobile base. The set of all configurations of the manipulator is the configuration space $\mathbb{Q}$. Feasible configurations are present in the set $\mathbb{Q}_f$. In this work, the feasible configurations are collision free, non-singular, and reachable at viewpoints.

6.4 Problem Formulation

We can generate a number of ordered sequences by permuting $s_i \in S$. Let $X^j = \{s_1^j, s_2^j, \ldots, s_m^j\}$ represent a sequence, where cardinality of the sets meet the requirement $|X| \leq |S|$. Let T_s^j be the total scanning time taken for the robot to capture pointclouds at all the different placements in the sequence. The time T_s^j is found by a path planner that plans the robot trajectory between all the viewpoints. The total execution time for relocating between the placements for the mobile base is written as T_e^j. Therefore, the total time taken for the scanning process for a given sequence X_j is $T_s^j + T_e^j$. Lastly, the number of placements in the sequence is given by cardinality of the set as $|X^j|$.

The problem is formally stated as follows: Given a CAD model of the part to be scanned, find an optimal sequence X^*, such that the cost described as $T_s^* + T_e^* + |X^*|$ corresponding to the sequence X^* is the minimum compared to all other feasible sequences.

6.5 Approach

6.5.1 Overview of Approach

Our approach uses the following steps to find the optimal base sequence X^* for the mobile manipulator.

1. CAD model of the part is used to generate a pointcloud $\mathscr{P}$ for the part.
2. Our method then uses a uniform sampling scheme over $\mathscr{P}$ to generate the set P such that each viewpoint is at operational distance of the camera and facing normal to the surface. A camera simulation model (Section 6.5.2) is used to obtain pointcloud of the part from the given viewpoint and coverage is determined.
3. The boundary B for feasible mobile base locations around the part is identified and represented as vertices of a polygon.

4. A set S of mobile base locations within the identified boundary B is created by using random sampling with diversity. Diversity is defined in Section 6.5.3.2. Each placement s_i is feasible and added to the set S if the placement does not result in collisions and the manipulator is reachable on at least one viewpoint without violating any constraint.

5. Each placement $s_i \in S$ is used as a node in a tree search algorithm (Section 6.5.4). The goal is to start from the mobile base current location as a root node and then branch out into children nodes in a depth first branch and bound [102] manner while expanding the tree. The expansion continues until a complete part cover is found. The path is a sequence X. The last node or the leaf node becomes the terminal node and the search backtracks to explore other branches in search of a better solution.

6. A greedy search is first performed in the aforementioned depth first manner over the set S using the branch guiding heuristic we develop in this work. The branch guiding heuristic is described in Section 6.5.4.1. The greedy search produces an initial solution which gives us the lower bound number of placements, also the cardinality of set $|X|$, and the overall execution time.

7. A depth first branch and bound search is then performed after the greedy search using the branch pruning and guiding heuristics with the lower bounds found in the previous step imposed and updated during the search. Branch pruning heuristic is discussed in detail in Section 6.5.4.2.

8. A sampling based technique is also used to deviate from existing branch in the tree and jump to a diverse and new branch in order to improve the diversity of search. The sampling technique is discussed in Section 6.5.4.3.

9. A timeout is provided as a stopping condition to the search and the best solution found for far is returned as the output of the method.

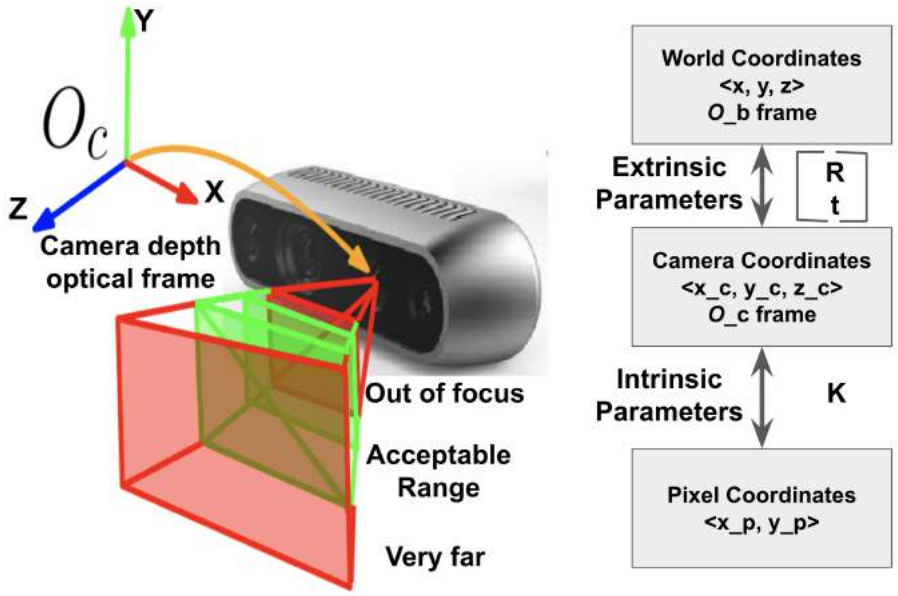

Figure 6.4: (Left) Intel RealSense D415 depth camera is shown with the camera optical frame O_c. The depth map is a square pixel grid and the frustrum shows the grid extended along the Z direction of O_c. Acceptable range is the center region of the frustrum. (Right) A block diagram showing the relationship between conversion of coordinates between world, camera, and pixel frames.

6.5.2 Camera Simulation Model

The RGB-D camera uses a grid of pixels to capture an image. The resolution of the grid specifies the number of pixels that will be used to represent the image. A higher number of pixels captures more details and vice versa. The RGB channel of the camera will produce the pixel matrices for values of red, green, and blue color composition of the image. Depth data is stored as the distance of a point on the part with respect to the camera internal optical frame O_c as shown in Figure 6.4 for each pixel, also called the depth map.

A simulated model of the camera projects rays from the optical frame to the external environment and finds the surface points in the environment with which the way intersects. The points are then used to create the depth map. We do not use any color information since we are dealing with only pointclouds. Intrinsic properties of the camera are then used to produce a pointcloud from the depth image with respect to the camera frame O_c. In this work, we use the properties of

an Intel RealSense D415 camera. Camera intrinsic properties is represented as a geometric transformation incorporating the optical center and focal length. The pointcloud is then converted to world frame O_w using the transformation of the camera frame with respect to the world frame ${}^{w}T_{c}$, also called the camera extrinsic property. The conversions can also be referred in Figure 6.4 as a block diagram.

The accuracy of the points captured varies within the camera workspace. Larger distance of the camera frame O_c from the part surface will lead to a coarse pixel density on the depth image and thus points are lost or accuracy reduces. The points are completely lost if the surface is very close and gets out of focus. Refer to Figure 6.4 that illustrates the field of view of the camera. There is an acceptable range which is used during the scanning process to keep the camera at a certain distance from the part surface. We will use this distance as a constraint while generating the viewpoints around the part. Let r be the normal distance of an observable point from the camera frame O_c. The acceptable range constraint is then given as follows.

$$r_{lb} \leq r \leq r_{ub} \tag{6.1}$$

6.5.3 Candidate Placement Generation

Our method converts the input CAD model into a pointcloud $\mathscr{P}$ representation of the part. The CAD model is uniformly sampled over the surface at a prescribed density to generate $\mathscr{P}$. $\mathscr{P}$ is then used as input to generate the set of viewpoints P. We will first discuss the generation of P in detail.

6.5.3.1 Generation of Viewpoints Set P

The pointcloud $\mathscr{P}$ is downsampled by using a lower resolution. In our work, we maintain a euclidean distance of 0.1 meters between a pair of neighboring points. The distance of 0.1 meters

was found enough to have a subset of viewpoints that would cover the entire part. Each point in the new low density cloud $\mathscr{P}_c$ will be used to create a viewpoint.

The normal information for every surface point in $\mathscr{P}_c$ is used to shift the surface points outward from the surface along the normal by a distance r. The value of r is chosen such that it obeys the range constraint given by the equation-6.1. In this work, we choose a value of 35 cm for the Intel RealSense D415 camera. The position of each shifted point is used as position of the viewpoints in the set P. Since each viewpoint is a frame, we also need three unit direction vectors. Unit Z vector for each viewpoint is chosen as the direction normal of the corresponding surface point found earlier. The unit X vector is chosen parallel to the world frame and unit Y vector is found using right hand rule. Each viewpoint is thus fully defined with a position and orientation in the Cartesian space.

The set of feasible mobile base placements (S) needs to be generated next in order to search for a sequence of placements.

6.5.3.2 Generation of Set S

A sampling based approach is used to generate the initial set of candidate placements for the mobile base. The sampling requires a well defined boundary B in the three dimensional configuration space for the mobile base. In this work, we constraint the configuration vector for the mobile base using lower and upper limits, $< x_{lb}, y_{lb}, \alpha_{lb} >$ and $< x_{ub}, y_{ub}, \alpha_{ub} >$ respectively. The bounds are manually provided by the user. The sampling procedure is then given by the following steps.

1. A sample placement $s_i =< x_i, y_i, \alpha_i >$ is randomly drawn within the boundary B.

2. For the base placement s_i, the set of viewpoints that the manipulator can reach are found. Determining reachability of the manipulator requires solving a computationally expensive optimization based inverse kinematics (IK) for every single viewpoint in the set P, for the given s_i. The optimization based IK is described in detail later in this Section-6.5.3.3. To

avoid iterating through all the viewpoints, we develop a capability map (Section-6.5.3.4) that allows us to iterate through only a small portion of all the viewpoints in P that will actually be reachable by the manipulator.

3. The placement s_i is invalid if no viewpoint is found to be reachable by the IK solver.

4. If s_i is valid and also satisfies all the kinematic constraints imposed by the IK solver, it is appended to the set S along with the set of points covered a_i, which is determined by the camera simulation model.

5. All the previous steps are then again repeated for a new random placement s_{i+1} in the next iteration. However, the new placement sample s_{i+1} is appended if it satisfies the diversity constraint on top of IK constraint. The diversity constraint enforces any two pair of setups s_k and s_l within the set S to have less than a threshold overlap of coverage between them. Formally, the following difference between the sets gives the diversity constraint.

$$a_k - a_l \geq \delta$$

where δ is the number of points that are required to be added by the newly sampled placement.

Focused sampling: The diversity rule sometimes leaves small regions uncovered since any samples which add less than δ points will be rejected. The rejected samples are stored in the memory and reevaluated if the algorithm does not converge within a maximum number of iterations. The small uncovered regions are covered by incorporating some of the rejected samples and stopping condition is then checked.

Stopping condition: The sampling terminates if the total number of distinct points covered by all the sampled placements equals the set of points in the pointcloud $\mathscr{P}$.

6.5.3.3 Optimization based Inverse Kinematics (IK)

The appropriate manipulator base frame O_r is first located with respect to the world frame by transforming the mobile base frame O_b to a placement s_i. The viewpoint p_j is already expressed in the world frame. Given the robot model for the manipulator $\mathscr{R}$, target viewpoint p_j, and the current manipulator configuration q_0, the optimization based IK is found by solving the following equation.

$$q^* \leftarrow \arg\min_q PE(q_0, p_j, \mathscr{R}) \tag{6.2}$$

$$PE = || < x_p, y_p, z_p > - < x_w, y_w, z_w > ||_2 +$$

$$\sum_{v=\{bx,by,bz\}} (1 - p_v * w'_v)$$

$$\text{where } w = FK(q)$$

where $FK(\cdot)$ is a forward kinematics function for the manipulator which finds the camera frame for the query configuration q. bx, by, bz are the unit X, Y, and Z direction vectors of the frames corresponding to the poses p and w. The pose error PE minimizes the position and orientation difference between the camera's frame w corresponding to q and the target frame p.

The validity of the found configuration q^* for the manipulator is determined by checking for three constraints. 1) The solution q^* should be within the lower and upper joint limits $\underline{q}$ and $\overline{q}$ respectively for the manipulator. 2) q^* should not result is collisions of the rigid bodies. 3) The configuration q^* should be non-singular. Following equations describe these constraints.

$$
\begin{aligned}
&q \text{ is valid if } \underline{q} \leq q^* \leq \overline{q} \\
&\text{InCollision}(q^*) == false \\
&\sqrt{\det(J(q^*) * J(q^*)^T)} > \eta
\end{aligned}
\tag{6.3}
$$

where $J(q)$ is the manipulator Jacobian that relates the Cartesian velocity of the end-effector frame with respect to the joint velocities. $J(q)^T$ is the transpose of the matrix. The determinant $\det(\cdot)$ evaluates the manipulability for a given configuration q. η is a constant.

Every valid reachable viewpoints will correspond to the set of reachable viewpoints for the placement s_i. The camera frame O_c is then aligned with every reachable viewpoint and simulated pointclouds are captured. The overall augmented pointcloud gives us the total covered area a_i for the placement s_i. The area is stored with the placement and viewpoints are now not needed for future computations.

6.5.3.4 Capability Map Construction

The IK solving and validation of a viewpoint is an expensive process. Generation of set S will take unreasonably long time if we iterate through all the viewpoints in the set P for every sampled placement s_i. A capability map helps us make the process more efficient by encoding manipulator reachability information in a data structure and quickly pruning viewpoints that may be completely unreachable. We will now discuss the construction of the map and how it is used in this work.

1. *Voxelization:* The manipulator workspace is first discretized into 3D cubes called voxels. We use a conservative estimate of the position reachability of the manipulator and add all the possible voxels with respect to the base frame O_r.

2. *Sampling of Configurations:* A randomly sampled manipulator configuration will lead to position and orientation of the end-effector in Cartesian space. The position of the origin of the end-effector frame is then mapped to the closest voxel in the workspace. Figure 6.5 shows an illustration of the mapping. All such configurations sampled will map to voxels in the voxelized workspace of the manipulator. A large number of configuration space sampling will lead to a defined boundary in positions reachable by the manipulator. All the reachable positions are basically the voxels where at least one sampled configuration has landed the manipulator end-effector.

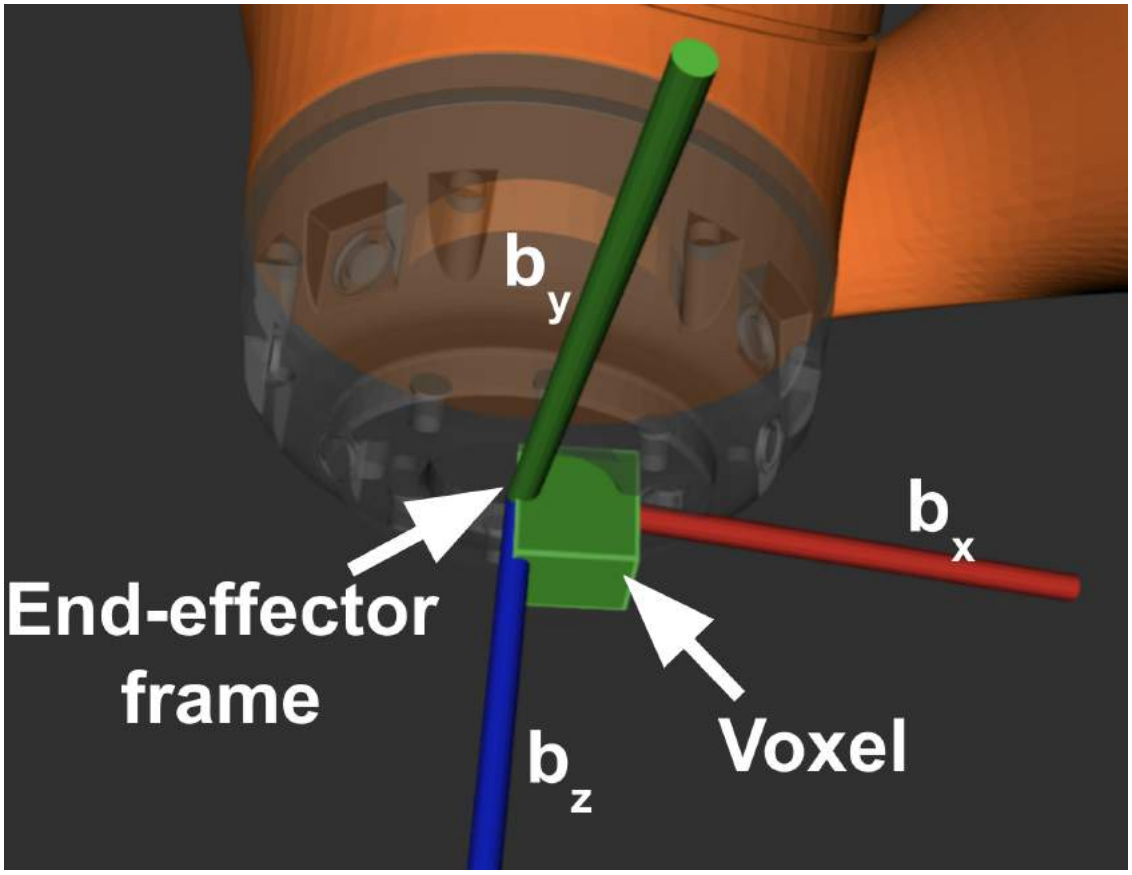

Figure 6.5: End-effector frame for the manipulator for a sampled configuration is shown. bx, by, bz are the unit X, Y, and Z direction vectors of the frame. The closest voxel to the end-effector position is also shown.

3. *Representing Orientations:* The end-effector frame also has unit X, Y, and Z direction vectors associated with a sampled configuration. These vectors are mapped to a data structure within the voxel that holds the orientations. Figure 6.6 illustrates the some example orientations originating from the center of the voxel. The orientations are shown in 2D for simplicity. We represent all the orientation vectors for unit Z direction vector by a principal axis. Two angles β_1 and β_2 are then used to enclose the sampled orientations within lower and upper bounds as shown in the figure. The angles are in relation with the principal axis and form a cone in 3D. Unit vectors X and Y for the end-effector are not stored since manipulators have almost 360° full rotation about the Z axis. Hence reachability in Z will increases the chances of end-effector reachable in X and Y axes as well.

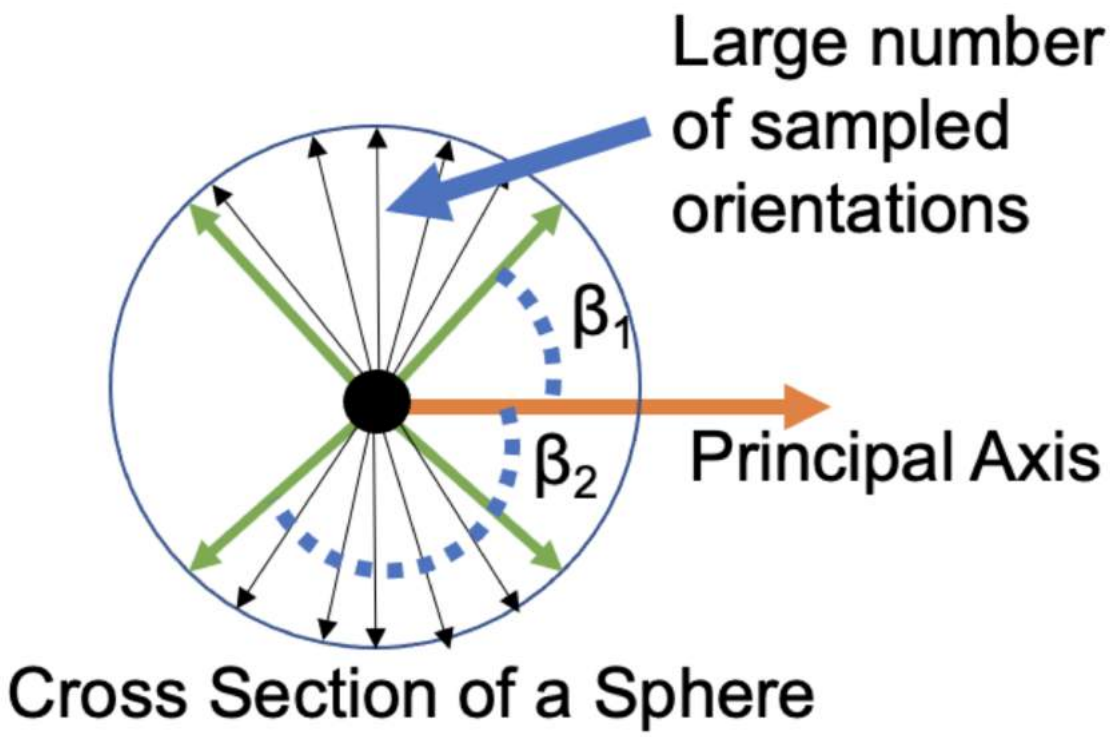

Figure 6.6: A 2D representation of the unit Z orientations originating from center of the voxel is shown. All the orientations are represented by a principal axis and two angles relative to the principal axis that provide a lower and upper bound.

4. *Sampling at Boundaries:* Due to non-linear mapping from configuration space to the workspace of the manipulator, many voxels at the boundaries would be reachable but a sample would not have landed there. We use inverse kinematics for such voxels to make sure that they are truly not reachable. If the boundary voxels are reachable using IK, we heavily sample within that voxel to create the orientation data structure.

For a sampled placement s_i, the capability map is transformed based on the manipulator base transform ${}^{w}T_{r}$. The position of all viewpoints in the set P are then transformed to find corresponding end-effector poses. The end-effector poses are then mapped to corresponding voxels in the capability map. The viewpoint positions that are outside the reachable voxels are not considered for IK validity step for placement s_i. Z axis of each reachable viewpoint is checked for reachability in the orientation data structure stored in the corresponding voxel. The viewpoints that are reachable by the capability map in position and orientation are sent for IK and constraint validity checks before admitting them in the set S. The capability map is an overestimate of the reachability and hence truly reachable viewpoints are not eliminated during the process. Querying the capability map is $1e3$ to $1e5$ order faster than solving IK and constraint equations. Filtering the viewpoints beforehand and only solving the IK and constraint equations for potentially reachable viewpoints makes the computation process much faster. We will now discuss how we find the optimal sequence X^* using our search method.

6.5.4 Branch and Bound Search

The generated placements in set S have the mobile base position and the set of points reachable on the part given by area set a. The information is be used to find the optimal sequence X^*. We use a depth-first branch and bound (DFBnB) approach to permute through the placements and find X^*.

The current mobile base location or home location is used as a root node or s_0 as shown in Figure 6.7. All the placements in the set S are candidate children nodes from the root node. We make a small change in terminology for better expression and the nodes are denoted as s_i^{jl}, where j is the sequence number corresponding to sequence X^j, l is the depth in the tree, and i is the index for the placement. The figure shows all n placements from the set S in the depth-1 of the tree.

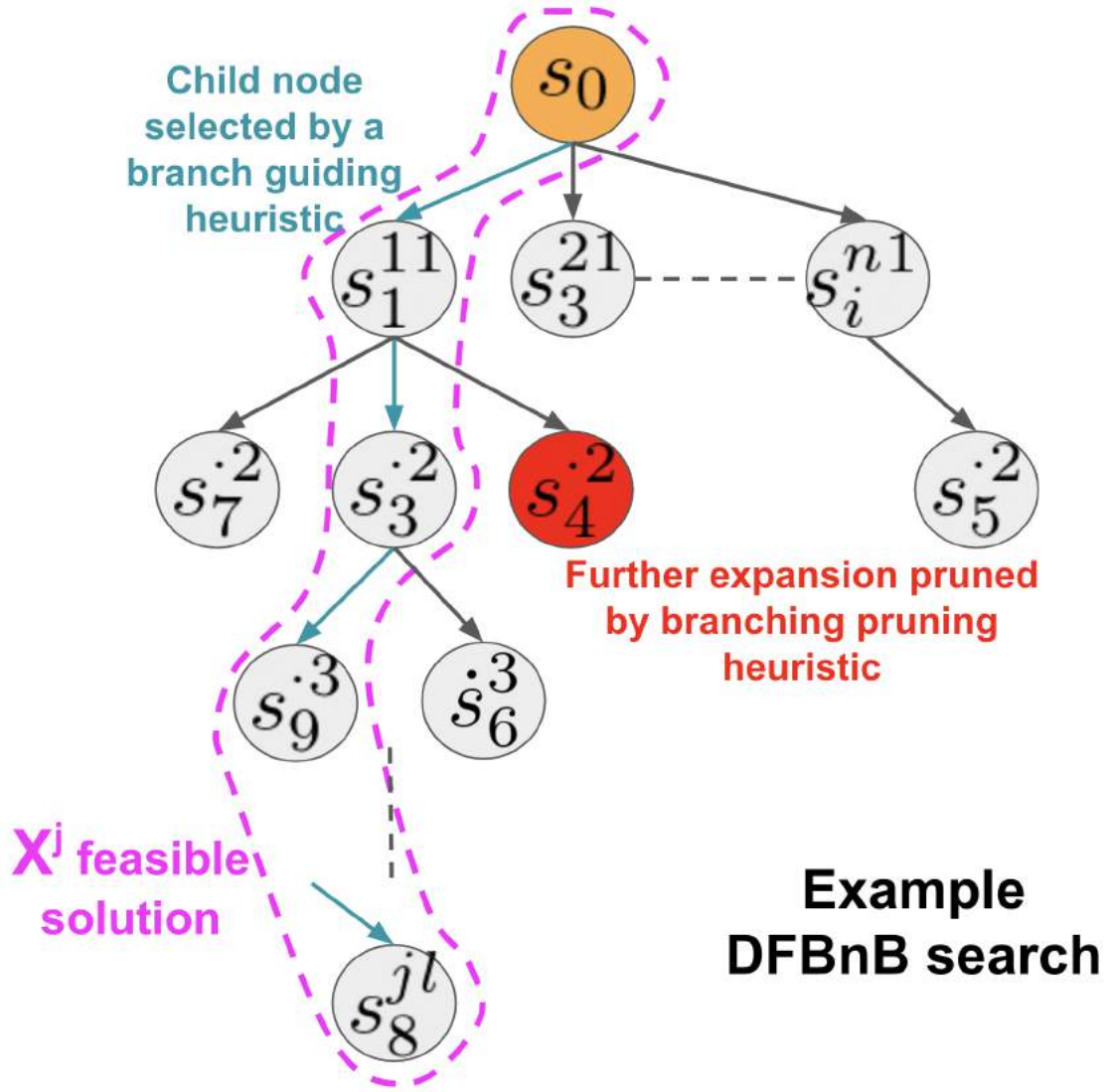

Figure 6.7: An example of the Depth-first branch and bound search algorithm used in our work.

A branch guiding heuristic which we will discuss later in this section will then evaluate all the n child nodes and order them. The first node in the order will be picked as shown in Figure 6.7 for expansion. The node s_1^{11} is picked in the illustration shown. It is important to note that the node which is a sibling can also be a part of the branch for another node. The algorithm is simply permuting $s \in S$ to find a solution. Figure 6.7 shows the node s_3 occurring as both $s_3^{\cdot 2}$ and s_3^{21}. Here, the dot in the symbol $s_i^{\cdot l}$ is used as a compact notation and solution number is excluded. The expansion from s_1^{11} is then done in a similar manner where branch guiding heuristic picks the next node. The expansion manner is also called depth first since we are progressing along the depth of the tree first.

The total part coverage for the sequence X^j is computed for up to k^{th} depth and for node s_i^{jk} using the equation-6.4.

$$A_i^{jk} = (\bigcup_{l=1}^{k} a_i^{jk} - \bigcap_{l=1}^{k} a_i^{jk}) \tag{6.4}$$

Once $|A_i^{jk}| = |\mathscr{P}|$, the node is considered as leaf node and the sequence X^j is returned as the first solution. Since the first solution sequence is found by using the branch guiding heuristic only, the first iteration of the algorithm is a greedy set cover.

As a leaf node is hit, the branch terminates after returning the solution and next node from the queue is picked. The depth-first manner results in the sibling of the leaf node being picked next. Depth-first search will waste enormous resources in exploring lower depths first before coming first level and branching to a different child. Branch pruning heuristic helps in reducing these explorations as further expansion of a node is terminated if a solution with lower cost than the previous solution can not be found. We will discuss the branch pruning heuristic in detail later in this Section. Figure 6.7 also illustrates the termination of a branch using branch pruning.

6.5.4.1 Branch Guiding Heuristic

Branch guiding is an important heuristic to direct the algorithm to find low execution time and low number of setups solution very quickly. The massive number of nodes to be explored due to a high branching factor will make the search intractable and return poor quality solutions without proper guiding. We use the idea that nodes that provide a higher gain in information with less travel time should be preferred. A branch guiding heuristic is then developed based on the rate of gain idea. The total area covered for a X^j sequence until depth k for a node s_i^{jk} is given by equation-6.4 as A_i^{jk}. The area covered by adding the child node $s_h^{j(k+1)}$ is then given by the equation-6.5. Let the path length executed by the mobile base while relocating from s_i^{jk} to $s_h^{j(k+1)}$ be given by the second order norm $||s_h^{j(k+1)} - s_i^{jk}||_2$. The branch guiding heuristic will pick the child node such that the rate gain given by equation-6.6 is maximized.

$$A_i^{j(k+1)} = A_i^{jk} \bigcup a_i^{j(k+1)} - A_i^{jk} \bigcap a_i^{j(k+1)} \tag{6.5}$$

$$A_i^{j(k+1)} / ||s_h^{j(k+1)} - s_i^{jk}||_2 \tag{6.6}$$

We use the path length as the surrogate for time taken for repositioning of the mobile base. The time for executing the manipulator trajectories can also be added to the cost function but we do not consider it during our computations since repositioning cost is much higher.

The branch guiding heuristic is essentially giving more preference to nodes that provide the maximum addition of area seen on the part with the given amount of travel time, also called the rate gain. A very low addition of part cover with a large travel time will naturally lead to higher execution times since the mobile base will be moving unnecessarily (refer Figure 6.8).

The branch guiding heuristic also orders the children nodes in decreasing order of the rate gain. The heuristic then disables a number of nodes given by the factor $f \in [0, 1]$ of the total number of nodes in that depth. These f portion of nodes having low rate gains are only enabled

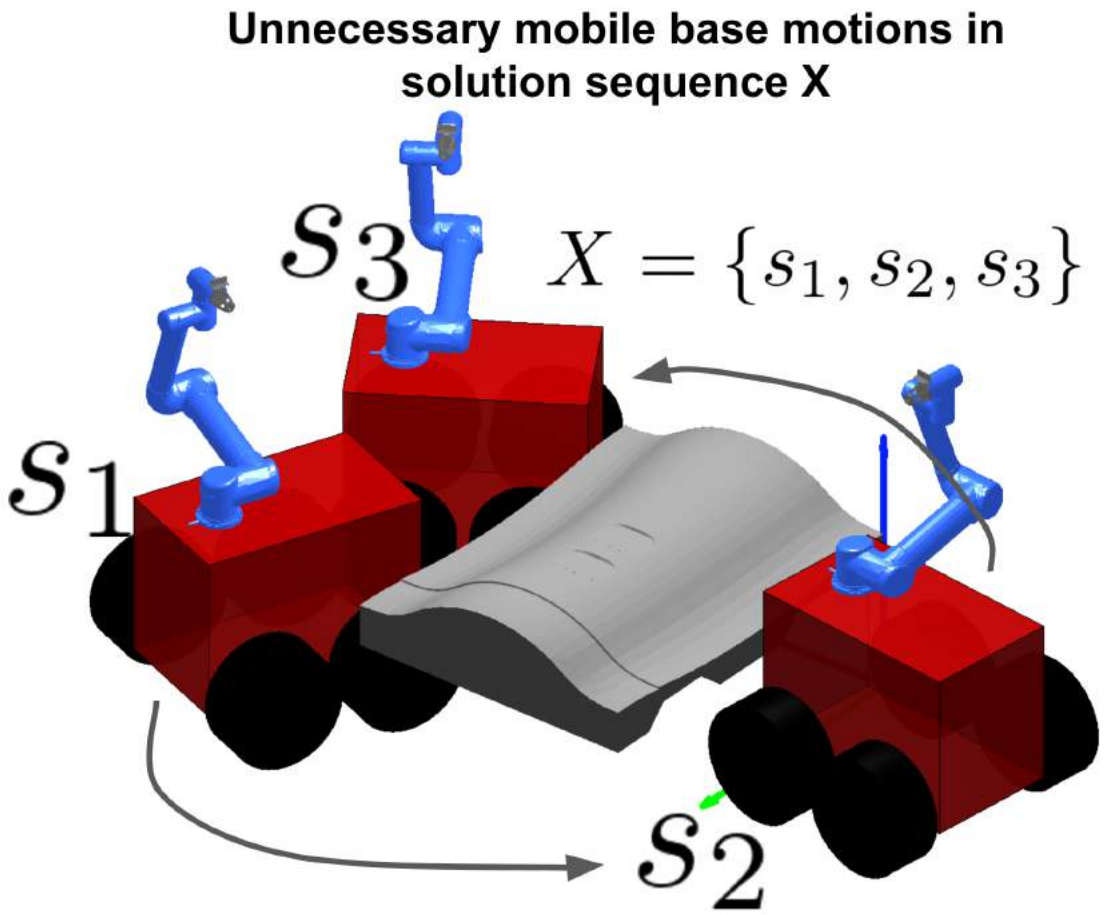

Figure 6.8: A solution X that leads to two base repositioning instead of one repositioning given by the sequence $\{s_1, s_3, s_2\}$

when all the good higher rate gain nodes have been expanded from the entire tree. The disabling is required since overall size of the tree is so large that the algorithm will be stuck in few branches at lower depths for a long time and not explore other diverse branches in the tree.

6.5.4.2 Branch Pruning Heuristic

The branch pruning heuristic is based on the best solution found so far. Consider the current best solution as X_c. The total path length of the mobile base repositioning (L^c_{reloc}) for this solution

is given by the equation-6.7. For simplicity we have omitted the subscript for placement index number s_i from the expression.

$$L^c_{reloc} = \sum_{l=2}^{k} (||s^{ck} - s^{c(k-1)}||_2) \tag{6.7}$$

$$L^j_{reloc} = \sum_{l=2}^{k} (||s^{jk} - s^{j(k-1)}||_2)$$

$$L^j_{reloc} + \min_{u=1}^{n}(||s^u_c - s^{jk}_i||_2) \geq L^c_{reloc} \tag{6.8}$$

$$|X^j| > |X^c| \tag{6.9}$$

For a given node s^{jk}_i in the branch j at depth k, let the set of children nodes be $S_c = \{s^1_c, s^2_c, \ldots, s^n_c\}$. The node s^{jk}_i will be considered as a terminal node and pruned from further expansion if either of the equations-6.8 and 6.9 is satisfied, in other words, an OR operator is used for the condition. The equation calculates the minimum of path lengths between the parent being evaluated in the j^{th} sequence and each child node in the next depth in the tree. If this minimum path length added to the repositioning path length L^j_{reloc} of the j^{th} sequence is greater than the minimum repositioning path length found so far L^c_{reloc}, the parent node is terminated and the branch is pruned. The branch is also pruned if the number of placements in the branch are greater than the number of placements in the current best solution.

6.5.4.3 Random Exploration

We also introduce random explorations during the search process. A probability p is associated with these explorations. If a sampled uniform value between 0 and 1 is lower than p, a random node from any other part of the tree is considered for expansion instead of using the next node in the queue maintained by our algorithm. The explorations allow for new branches to be discovered instead of getting locally stuck in the existing branch. We run the random sampling routine before a new node is picked from the queue and expanded in every iteration.

6.6 Results

6.6.1 Test Cases

We conducted experiments in simulations using an inspector bot mobile base equipped with a Universal Robots UR5 robot. UR5 is a 6DOF collaborative robot arm. The mobile base is a 3DOF platform with x, y translation motions along the X and Y axis and α rotation motion about the vertical or Z axis. Overall dimensions of the mobile base used are 800 X 610 X 400 mm. We represent the mobile base as a cuboid with four wheels. We chose four different industry inspired, large, and complex part geometries labeled A, B, C, and D to take the depth camera and scan in order to construct the pointclouds. Figure 6.9 shows the parts used in our work.

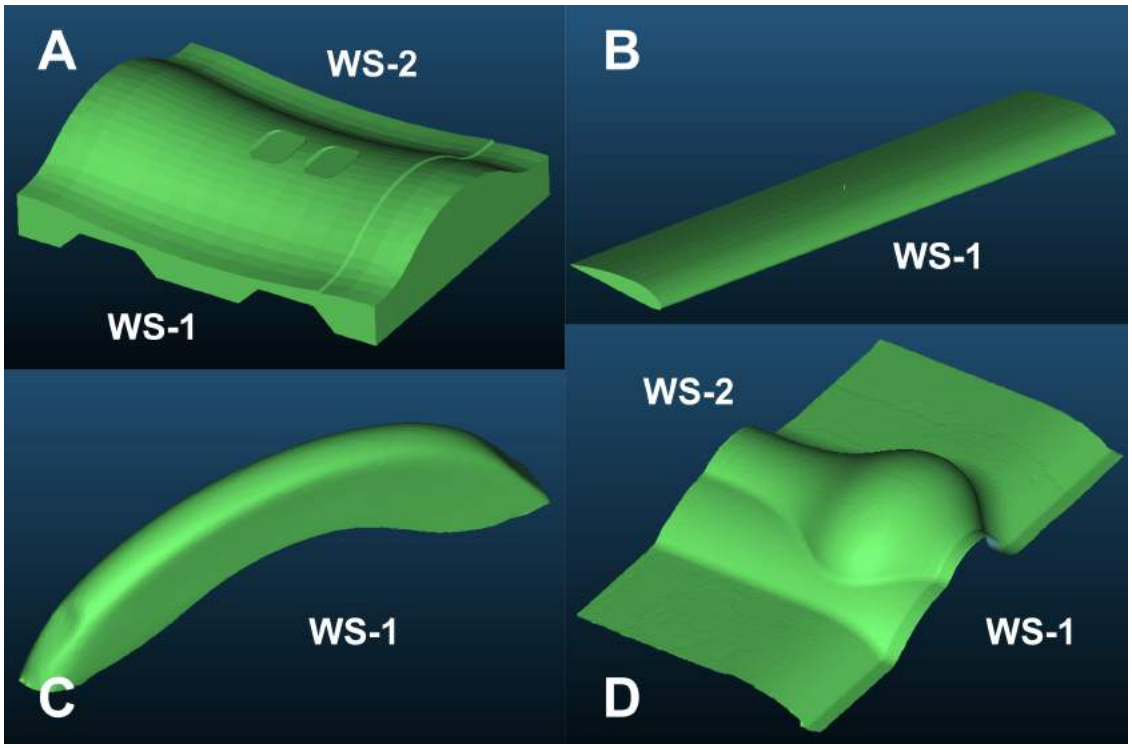

Figure 6.9: CAD models of the industry inspired test cases used in this work to evaluate the performance of our optimal base sequencing algorithm. Part A is similar to an aerospace mold used to manufacture composite parts. Part B is an airfoil. Part C is a scaled version of a motor bike fender. Part D is another aerospace composite layup mold. The mobile base workspace (labeled WS in the figure) shows the sides of the parts where mobile base is allowed to operate for complete coverage.

The dimensions of each part A, B, C, and D are given in the Table 6.1. Each dimension is in millimeters and describes the longest dimension (length), breadth, and height for the parts. The number of viewpoints that are sampled over the parts for aligning the camera frame O_c is also provided. Lastly, the number of sampled mobile base placements in the set S for each part is provided.

Part	Dimensions	Number of viewpoints	Number of sampled placements
A	1000 X 1300 X 300	241	92
B	2250 X 400 X 100	108	46
C	2420 X 200 X 600	95	125
D	1750 X 850 X 450	275	160

Table 6.1: Dimensions, number of viewpoints sampled over the part surface, and number of mobile placements sampled around the part are presented for each test case used in this work.

The viewpoints are sampled over the surface of the parts which needs to be scanned. The parts are very large for the mobile base to cover using placements from one side of the part. Therefore, mobile base workspace on the floor is generated on two sides of parts A and D. The sides where the workspace is present is labeled as WS in the Figure 6.9.

6.6.2 Algorithm Performance

We benchmark the performance of the DFBnB algorithm by comparing it to two different baseline methods. A naive way to approach the base sequencing would be to randomly permute $s \in S$ and find out sequences with complete coverage. We call the random sampling strategy as baseline-1. Baseline-2 approach is what is commonly used as greedy set cover. The greedy search picks

placements that provide the maximum rate gain described by equation-6.6 and progresses in depth-first manner. As complete cover is reached, the solution is outputted and algorithm terminates. We compare these two baselines with our approach based on the number of placements $|X^*|$ in the optimal base sequencing solution X^*, the overall path length L^*_{reloc} of the mobile base, and the number of nodes evaluated in the tree. Table 6.2 presents the comparisons. All the cases were limited to a timeout with maximum 100,000 nodes to be evaluated. The best solution returned within that timeout is presented in the table.

	Random Sampling Strategy			Greedy Search		Our Approach		
Part	Number of placements	Path length (meters)	Nodes evaluated	Number of placements	Path length (meters)	Number of placements	Path length (meters)	Nodes evaluated
A	15	22.18	32767	11	9.31	8	6.11	8477
B	10	8.55	30443	10	14.07	6	3.66	3162
C	18	20.11	38545	12	10.31	9	4.28	15742
D	11	16.11	25825	9	8.09	6	5.57	8803

Table 6.2: The table presents comparisons between baseline-1 (random strategy), baseline-2 (greedy search), and our approach with respect to the number of placements, path lengths, and number of nodes evaluated.

We can observe in Table 6.2 that random sampling finds solutions with higher number of placements and significantly high path lengths as compared to greedy or our approach. The greedy method does better since the branch guiding heuristic is used. However, a greedy solution may not be the optimal solution as we can see by comparison with our approach where relatively fewer number of placements and path lengths are observed in the solution. Our approach will evaluate more nodes to produce further lower cost solutions. The node evaluation still remains within practical time limits of computation costs under 0-120 seconds. Our approach performs better due to the combined effect of branch guiding and pruning heuristics and random sampling strategy to explore more branches of the tree. Random exploration probability p is kept at 0.2 for all the cases. The results produced for random strategy and our approach are average values of the solutions found over 50 runs as random component is involved. The number of placements is converted into an integer using ceiling function and presented in the Table 6.2.

6.6.3 Branch Pruning vs Guiding Heuristic

The insights into usefulness of the branch pruning and guiding heuristic and a comparison between them is presented in Table 6.3. The random node selection technique for exploring other branches of the tree is not enabled to generate these results. Disabling random sampling helps isolate the behavior of the deterministic search algorithm. The table has columns which indicate if the branch guiding and pruning heuristic is turned ON or OFF during the search. We can notice that for all the test cases when the branch guiding heuristic is turned OFF, the number of nodes evaluated is much higher as children nodes are randomly selected instead of selecting them based on rate gain. The path lengths show a similar trend. The number of placements and path lengths significantly reduce when branch guiding is turned back ON and pruning is turned OFF. Branch guiding and pruning both together are turned ON in the third scenario where very good quality solutions are obtained with lower placements and path lengths with fewer node evaluations. Lastly, the results presented in Table 6.2 of our approach are better than both pruning and guiding being turned ON due to the random node selection strategy used.

6.6.4 Effectiveness of Capability Map

The capability map helps us reduce the number of expensive optimization based IK calculations while generating the set S. Table 6.4 presents the IK evaluations for both scenarios; with and without capability map use. The percentage improvement observed is significantly high since much of the viewpoints are discarded by the map as mobile manipulator workspace is very small and limited as compared to part sizes.

6.6.5 Placements Generated

The optimal base sequencing solutions found by using our approach are plotted in 3D in front of the respective parts for all the test cases in Figure 6.10. The viewpoints are also shown over the

Part	Branch Pruning	Branch Guiding	Number of setups	Path length	Nodes evaluated
A	On	Off	17	27.03	379671
	Off	On	11	8.31	2738
	On	On	10	7.81	1041
B	On	Off	11	10.21	343411
	Off	On	9	8.81	2203
	On	On	8	5.2	254
C	On	Off	19	34.73	626922
	Off	On	12	9.31	2838
	On	On	10	8.86	838
D	On	Off	14	20.11	576342
	Off	On	8	7.09	1865
	On	On	7	6.71	897

Table 6.3: Three scenarios where branch guiding and pruning are turned ON and OFF during the search without random selection technique are evaluated. A timeout with maximum number of allowed node evaluations as 1,000,000 is kept and the best solution found is returned. The number of placements and path lengths are used as comparison metric.

part surface. Parts A and D have two workspace locations WS-1 and 2 where placements were allowed to be generated. The manipulator workspace was very restricted due to high demand in manipulability and make the test cases more challenging. Therefore, we see more number of placements in the solutions than we intuitively consider exists.

Part	Number of IK Evaluations		Percentage Improvement
	Without Capability Map	With Capability Map	
A	36971	4920	86.69%
B	12054	2341	80.58%
C	11875	2110	82.23%
D	52360	4718	90.99%

Table 6.4: The table shows percentage reduction in the number of optimization based IK evaluations by using the capability map to prune the viewpoints that are unreachable by the manipulator for a mobile base location.

6.7 Summary

We presented an optimal mobile base placement sequencing algorithm to carry a depth camera around a large complex part for inspection. The approach is also applicable to tasks like 3D reconstruction, painting, polishing, or sanding which require optimal base sequencing of the mobile manipulator. We developed heuristics that made the search for an optimal sequence in the large search space efficient and real time. A capability map was then developed to also construct the search space with improvements in the number of inverse kinematics solved. We then conducted simulation experiments on four challenging test cases inspired by industry parts to evaluate the performance of our algorithm. Our method performed better than presented baselines and demonstrated fast, real time performance with very few node evaluations required to generate low path length and number of placement solutions.

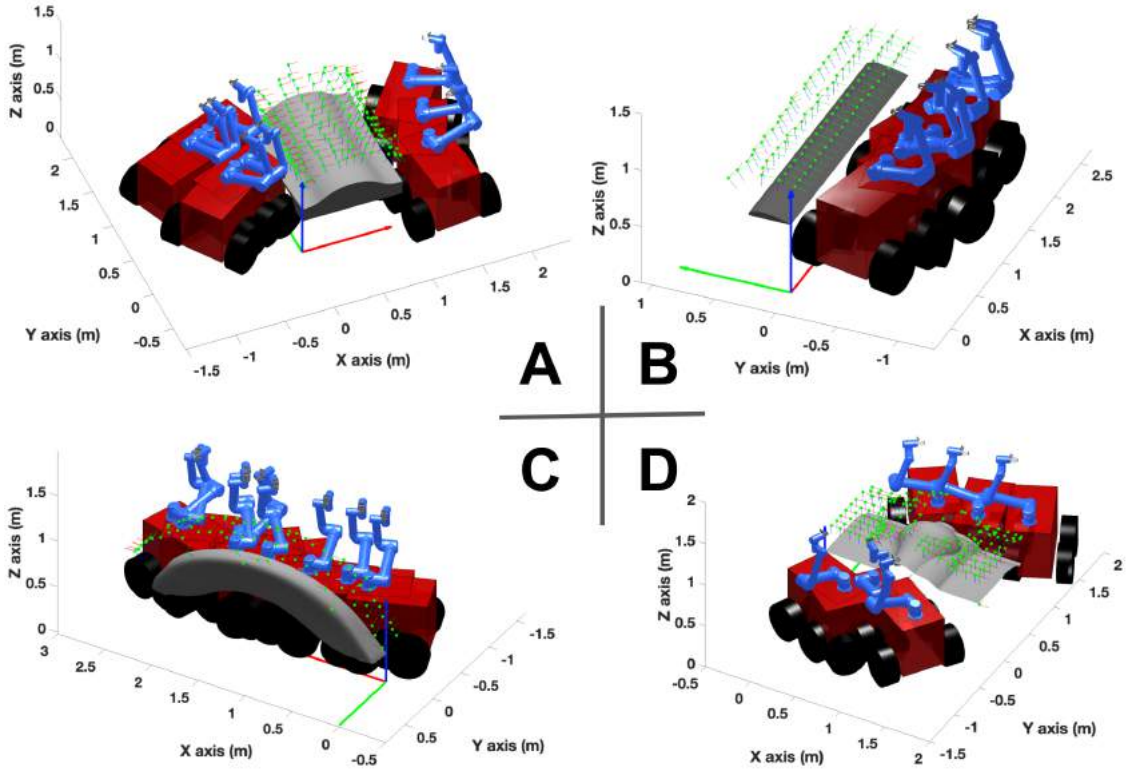

Figure 6.10: The optimal base sequence solutions are plotted in 3D for each test case. The example viewpoints are also shown as frames over the parts.

Chapter 7

Conclusions

7.1 Intellectual Contributions

The dissertation makes the following five fundamental contributions.

1. A robot trajectory generation algorithm for semi-Constrained Cartesian paths was developed in Chapter-2. This work made fundamental advances in the field of graph-based path planning by developing an iterative reduced graph strategy to improve computational performance. The workspace heuristic and source biasing strategy lead to paths within 1-6% of the optimal solution by using an average of 12 times fewer node evaluations over five generic test cases. The overall size of the graph also reduces as a consequence of improving graph-search performance. A user can now provide high-level constraints with increased flexibility over Cartesian paths and find near-optimal solutions with an improved success rate.

2. Chapter-3 made advances in the field of 3D reconstruction algorithms by developing coverage planning that selectively uses the RGB-D camera operational range to admit only high fidelity pointclouds. The planner optimized the accuracy of the output pointcloud and the robot execution time. An online refinement algorithm was developed that guaranteed a high density of points across the entire part. The contributions will be useful in manufacturing

inspection, medical imaging, part localization, and similar fields. Users can accurately localize the part within two millimeters and also limit the maximum error on the part.

3. The third contribution in this work is the development of algorithms for multi-robot manipulation of deformable viscoelastic materials. A composite sheet layup application is used as an example where manipulation and draping happen simultaneously. The behavior of the deformable material is characterized by a cost function that delays the injection of manipulation energy into the system. A manipulation planner was developed which made advances in graph-based planning by incorporating simulations to evaluate the cost of states and heuristics to guide the search by prioritizing low-cost nodes. The planner can be used by users to automatically generate manipulation plans for multi-robot setup. Our heuristics reduced computation time by 80-90% and found paths with 30-70% lower initial cost over five generic test cases.

4. The fourth contribution made advances in the field of optimization-based robot placement planning. This work developed a capability map that characterized the relationship between robot configuration space and the Cartesian space. The map captured robot reachability with a 3% overestimation and improved inverse kinematics computation time by an order of 2-3 in magnitude. The map was used by the state-space search algorithm to generate high-quality initial placements that were fed to an optimization algorithm that used a successive application of constraints strategy to improve computational performance by over 95% and increase success rate by 15-88% over five generic test cases. A user can efficiently deploy a robotic system by using the automated robot placement algorithms and find trajectories by using planners described in this work.

5. The dissertation made advances in tree-search algorithms as the fifth contribution to path planning by developing branch guiding and branch pruning heuristics for depth-first branch and bound algorithm for finding the optimal sequence of mobile base placements for a given

set of tasks in the workspace. The heuristics reduced node evaluations by over 95% compared to sampling strategy and path costs by 30-70% over four generic test cases. The number of placements was also minimized. The mobile robots used can automatically perform a set of tasks with a minimum number of placements and execution time using our algorithms.

7.2 Anticipated Benefits

The fundamental advances made in this dissertation will be used in any robotic application where path-constrained trajectories, robot placement, and accurate localization using low-cost sensors are needed. The work has made contributions at the intersection of artificial intelligence and robotics. Smart manufacturing robotic cells of the future that use technologies from the two intersecting fields will have intelligent robots that will be capable of programming themselves efficiently. Users can deploy robots faster since cell designs will be automatically computed using placement algorithms and robot motions will be generated by path constrained algorithms. This dissertation particularly expands the capabilities of the robots by utilizing the complete robot workspace. The focus was mainly on large complex paths imposing multiple motion constraints.

7.3 Summary

The previous section went over major contributions from this dissertation. This section is focused on lessons learned from the work and how they can be used by readers towards deploying a robotic system effectively.

The work first explores a discrete graph-based search for generating robot trajectories for semi-constrained toolpaths. The flexibility provided by defining tolerances and multiple TCPs increases the state space size. Trajectories are found in the robot configuration space. Environment obstacles and the task is defined in the workspace. It is important to guide the search in configuration space using cues from the workspace. Failure to convey the information will lead

to unnecessary explorations and an increase in search time. The workspace heuristic uses the Jacobian to encode the information. Another important aspect is to learn which regions are providing good quality solutions. The problems where search space has a structure where solutions have a reasonably continuous variation in the local neighborhood can be solved efficiently using optimization strategies. Instead of deterministically exploring the search space, a bias should be created towards low-cost neighborhoods.

The dissertation then developed algorithms for robot trajectory generation when camera performance constraints were involved. The constraint did not describe a fixed path for interaction between the tool (camera) and the part but was found using the current robot configuration and part geometry. Planning under performance constraints required exploring a vast search space to determine which camera poses lead to high fidelity pointcloud collection and complete coverage. Online planners executed defect detection during the robot motion and zero or low-density points were filled. The idea of using only a small region of sensor operational range to select high fidelity pointclouds by compromising slightly on execution time leads to overall high output accuracy. Moreover, the defects created due to external environmental factors and restricted sensor operational range were fixed using online monitoring.

The multi-robot deformable sheet manipulation shows the importance of encoding physics during the search. Physics-based simulations extend to a wide variety of materials by changing the simulation parameters. Offline plans are close to the desired physical plans and online refinement corrects errors that can lead to defects. The work also presents how a system can be studied empirically and behavior can be encoded as a cost function to compute plans. The sheet manipulation depended on the total energy of the sheet which was encoded as a cost function for the planner. The heuristic that was based on a gradient descent scheme ordered the states that needed to be prioritized and quickly found paths with a lower cost. The overall computation time was also reduced using the heuristic.

Robot placement algorithms for a fixed manipulator base and a mobile manipulator revealed the importance of encoding the robot reachability in a pre-computed map. The capability map is a quick access position and orientation look-up table that was constructed within 3% overestimation. Inverse kinematics can be evaluated using the map to select good placements with 1e2-1e3 order faster as compared to expensive optimization calls. Computation performance is further improved when the motion constraints are successively applied instead of enforcing them all at once. This can be seen with improvements in the inverse kinematics-based optimization method and hierarchical robot base placement method. Combined, the methods give rise to algorithms that can fast and effectively explore the large robot workspace for placements and find solutions that meet all the constraints. Mobile base placement on the other hand showed how efficient branch guiding and branch pruning heuristics developed based on the rate of information gain from one state to the other led to a fewer number of placements in the sequence and lower execution times. The capability map also helped speed up the evaluation of each state by selectively solving for inverse kinematics for tasks.

7.4 Future Works

The presented work can be extended in the following directions.

1. *Redundant Manipulators:* The algorithms presented with an exception of sequential optimization-based motion planners were aimed at using 6 DOF industrial manipulator arms. Advanced techniques will be needed when redundant robot systems are used. Infinitely many solutions exist for such manipulators and the search space quickly explodes. Efficient maneuvering through the space will require newer concepts that the algorithms can use.

2. *Machine Learning:* The parameters used by different planners in this work were manually tuned for different test cases. The algorithms will benefit when learning-based methods will be applied to find generic relationships across a wide range of geometries. Machine

learning-based methods will also be useful to replace the look-up table like capability map. Sophisticated deep learning models can be used to characterize the robot workspace and configuration space relationship more accurately.

3. *Mobile Manipulator 3D Reconstruction:* A system that uses a mobile manipulator with an RGB-D sensor and advanced localization for the mobile base can be used for the 3D reconstruction of large parts. With theoretically infinite workspace, the mobile base can travel around the part and build a model for localization or inspection. Uncertainty of the base location can quickly propagate into the output pointcloud. Algorithms will be needed that merge pointclouds using accurate localization of the mobile base. Planners that will find a synchronous mobile base and manipulator trajectory for inspection will be needed.

4. *Using Previous Experience:* Another future direction will be to use the experience of previously generated robot motions or placements. Learning-based models can be used to store the experiences and use them for speeding up planning in similar instances encountered in the future. For instance, a large portion of geometries used in a manufacturing facility may be similar. The feasible robot configurations from previous optimal solutions can be linked to the geometrical features and similar configurations can be prioritized in future searches. Robot placements that were of higher quality can be encoded in the feature space and can be used for good initial guesses for future searches if the environments are similar.

References

[1] Nicholas Adrian and Quang-Cuong Pham. Mobotsp: Solving the task sequencing problem for mobile manipulators. *arXiv preprint arXiv:2011.04461*, 2020.

[2] Randa Almadhoun, Tarek Taha, Lakmal Seneviratne, Jorge Dias, and Guowei Cai. A survey on inspecting structures using robotic systems. *International Journal of Advanced Robotic Systems*, 13(6):1729881416663664, 2016.

[3] Javier Alonso-Mora, Ross Knepper, Roland Siegwart, and Daniela Rus. Local motion planning for collaborative multi-robot manipulation of deformable objects. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5495–5502, 2015.

[4] Vivek Annem, Pradeep Rajendran, Shantanu Thakar, and Satyandra K Gupta. Towards remote teleoperation of a semi-autonomous mobile manipulator system in machine tending tasks. In *ASME 2019 14th International Manufacturing Science and Engineering Conference*. American Society of Mechanical Engineers Digital Collection, 2019.

[5] apcm. Prepreg sheet. `https://www.prepregs.com/da4518/`, 2019. [Online; accessed August-2019].

[6] N. A Aspragathos. Optimal location of path following tasks in the workspace of a manipulator using genetic algorithms. In *Recent Advances in Robot Kinematics*, pages 179–188. Springer, 1996.

[7] N. A. Aspragathos and S. Foussias. Optimal location of a robot path when considering velocity performance. *Robotica*, 20(2):139–147, 2002.

[8] Shi Bai, Fanfei Chen, and Brendan Englot. Toward autonomous mapping and exploration for mobile robots through deep supervised learning. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2379–2384. IEEE, 2017.

[9] Yunfei Bai, Wenhao Yu, and C Karen Liu. Dexterous manipulation of cloth. *Computer Graphics Forum*, 35(2):523–532, 2016.

[10] G. Barequet and G. Elber. Optimal bounding cones of vectors in three dimensions. *Information Processing Letters*, 93(2):83–89, 2005.

[11] O. Burchan Bayazit, Jyh-Ming Lien, and N. M. Amato. Probabilistic roadmap motion planning for deformable objects. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 2126–2133 vol.2, 2002.

[12] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear programming: theory and algorithms*. John Wiley & Sons, 2013.

[13] D. Berenson. Manipulation of deformable objects without modeling and simulating deformation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4525–4532, Nov 2013.

[14] Dmitry Berenson, Siddhartha Srinivasa, and James Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *The International Journal of Robotics Research*, 30(12):1435–1460, 2011.

[15] Prahar M Bhatt, Cheng Gong, Ariyan M Kabir, Rishi K Malhan, Brual C Shah, and Satyandra K Gupta. Incorporating tool contact considerations in tool-path planning for robotic operations. In *International Manufacturing Science and Engineering Conference*, volume 84256, page V001T05A018. American Society of Mechanical Engineers, 2020.

[16] Prahar M. Bhatt, Ariyan M. Kabir, Rishi K. Malhan, Brual C. Shah, Aniruddha V. Shembekar, Yeo J. Yoon, and Satyandra K. Gupta. A robotic cell for multi-resolution additive manufacturing. In *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019.

[17] Prahar M Bhatt, Ariyan M Kabir, Rishi K Malhan, Aniruddha V Shembekar, Brual C Shah, and Satyandra K Gupta. Concurrent design of tool-paths and impedance controllers for performing area coverage operations in manufacturing applications under uncertainty. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 1151–1156. IEEE, 2019.

[18] Prahar M Bhatt, Ashish Kulkarni, Alec Kanyuck, Rishi K Malhan, Luis S Santos, Shantanu Thakar, Hugh A Bruck, and Satyandra K Gupta. Automated process planning for conformal wire arc additive manufacturing. *The International Journal of Advanced Manufacturing Technology*, pages 1–26, 2022.

[19] Prahar M Bhatt, Rishi K Malhan, Pradeep Rajendran, Aniruddha V Shembekar, and Satyandra K Gupta. Trajectory-dependent compensation scheme to reduce manipulator execution errors for manufacturing applications. In *International Manufacturing Science and Engineering Conference*, volume 85062, page V001T05A009. American Society of Mechanical Engineers, 2021.

[20] Prahar M. Bhatt, Pradeep Rajendran, Keith McKay, and Satyandra 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.

[21] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart. Receding horizon path planning for 3d exploration and surface inspection. *Autonomous Robots*, 42(2):291–306, 2018.

[22] Andreas Björnsson, Marie Jonsson, and Kerstin Johansen. Automated material handling in composite manufacturing using pick-and-place systems–a review. *Robotics and Computer-Integrated Manufacturing*, 51:222–229, 2018.

[23] J.E. Bobrow, S. Dubowsky, and J.S. Gibson. Time-optimal control of robotic manipulators along specified paths. *The International Journal of Robotics Research*, 4(3):3–17, 1985.

[24] Daniel M Bodily, Thomas F Allen, and Marc D Killpack. Motion planning for mobile robots using inverse kinematics branching. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5043–5050. IEEE, 2017.

[25] Aljaž Božič, Michael Zollhöfer, Christian Theobalt, and Matthias Nießner. Deepdeform: Learning non-rigid rgb-d reconstruction with semi-supervised data. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, pages 7002–7012, 2020.

[26] D. E. Breen, D. H. House, and P. H. Getto. A physically-based particle model of woven cloth. *The Visual Computer*, 8(5-6):264–277, 1992.

[27] D. E. Breen, D. H. House, and M. J. Wozny. A particle-based model for simulating the draping behavior of woven cloth. *Textile Research Journal*, 64(11):663–685, 1994.

[28] R. O. Buckingham and G. C. Newell. Automating the manufacture of composite broadgoods. *Composites Part A: Applied Science and Manufacturing*, 27(3):191 – 200, 1996.

[29] S. Caro, C. Dumas, S. Garnier, and B. Furet. Workpiece placement optimization for machining operations with a kuka kr270-2 robot. In *IEEE International Conference on Robotics and Automation*, pages 2921–2926, Karlsruhe, Germany, 2013.

[30] S. Caro, S. Garnier, B. Furet, A. Klimchik, and A. Pashkevich. Workpiece placement optimization for machining operations with industrial robots. In *IEEE International Conference on Advanced Intelligent Mechatronics*, pages 1716–1721, Besançon, France, 2014.

[31] M. Cefalo, G. Oriolo, and M. Vendittelli. Planning safe cyclic motions under repetitive task constraints. In *IEEE International Conference on Robotics and Automation*, pages 3807–3812, May 2013.

[32] Fanfei Chen, Shi Bai, Tixiao Shan, and Brendan Englot. Self-learning exploration and mapping for mobile robots via deep reinforcement learning. In *Aiaa scitech forum*, page 0396, 2019.

[33] P. Y. Chen and P. M. Popovich. *Correlation: Parametric and nonparametric measures*. Number 137-139. Sage, 2002.

[34] Wei Chen, Yang Tang, and Qiang Zhao. A novel trajectory planning scheme for spray painting robot with bézier curves. In *IEEE Chinese Control and Decision Conference (CCDC)*, pages 6746–6750, 2016.

[35] T Chettibi, HE Lehtihet, M Haddad, and S Hanchi. Minimum cost trajectory planning for industrial robots. *European Journal of Mechanics-A/Solids*, 23(4):703–715, 2004.

[36] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785*, 2014.

[37] Daniela Constantinescu and Elizabeth A Croft. Smooth and time-optimal trajectory planning for industrial manipulators along specified paths. *Journal of robotic systems*, 17(5):233–249, 2000.

[38] J. J. Craig. *Introduction to robotics: mechanics and control*, volume 3. Pearson/Prentice Hall Upper Saddle River, NJ, USA, 2005.

[39] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images,. In *in Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. ACM,*, pages 303–312, 1996.

[40] Chengkai Dai, Sylvain Lefebvre, Kai-Ming Yu, Jo MP Geraedts, and Charlie CL Wang. Planning jerk-optimized trajectory with discrete time constraints for redundant robots. *IEEE Transactions on Automation Science and Engineering*, 17(4):1711–1724, 2020.

[41] Chengkai Dai, Charlie CL Wang, Chenming Wu, Sylvain Lefebvre, Guoxin Fang, and Yong-Jin Liu. Support-free volume printing by multi-axis motion. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[42] Jadav Das and Nilanjan Sarkar. Autonomous shape control of a deformable object by multiple manipulators. *Journal of Intelligent & Robotic Systems*, 62(1):3–27, Apr 2011.

[43] Carl De Boor. *A practical guide to splines*, volume 27. Springer-Verlag New York, 1978.

[44] Mustafa Devrim Kaba, Mustafa Gokhan Uzunbas, and Ser Nam Lim. A reinforcement learning approach to the view planning problem. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6933–6941, 2017.

[45] Neel Dhanaraj, Yeo Jung Yoon, Rishi K Malhan, Prahar M Bhatt, Shantanu Thakar, and Satyandra K Gupta. A mobile manipulator system for accurate and efficient spraying on large surfaces. In *International Conference on Industry*, volume 4, 2021.

[46] Feng Ding, Jian Huang, Yongji Wang, Takayuki Matsuno, and Toshio Fukuda. Vibration damping in manipulation of deformable linear objects using sliding mode control. *Advanced Robotics*, 28(3):157–172, 2014.

[47] N. C. N. Doan and W. Lin. Optimal robot placement with consideration of redundancy problem for wrist-partitioned 6r articulated robots. *Robotics and Computer-Integrated Manufacturing*, 48:233–242, 2017.

[48] Mehmet Dogar, Andrew Spielberg, Stuart Baker, and Daniela Rus. Multi-robot grasp planning for sequential assembly operations. *Autonomous Robots*, 43(3):649–664, 2019.

[49] Siyan Dong, Kai Xu, Qiang Zhou, Andrea Tagliasacchi, Shiqing Xin, Matthias Nießner, and Baoquan Chen. Multi-robot collaborative dense scene reconstruction. *ACM Transactions on Graphics*, 38(4), 2019.

[50] A. Doumanoglou, J. Stria, G. Peleka, I. Mariolis, V. Petrík, A. Kargakos, L. Wagner, V. Hlaváč, T. K. Kim, and S. Malassiotis. Folding clothes autonomously: A complete pipeline. *IEEE Transactions on Robotics*, 32(6):1461–1478, Dec 2016.

[51] Bin Du, Jing Zhao, and Chunyu Song. Optimal base placement and motion planning for mobile manipulators. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 45035, pages 1227–1234. American Society of Mechanical Engineers, 2012.

[52] Rajiv. Dubey and John. Luh. Redundant robot control using task based performance measures. *Journal of robotic systems*, 5(5):409–432, 1988.

[53] C. Dumas, S. Caro, M. Cherif, S. Garnier, and B. Furet. Joint stiffness identification of industrial serial robots. *Robotica*, 30(4):649–659, 2012.

[54] C. Dumas, S. Caro, S. Garnier, and B. Furet. Workpiece placement optimization of six-revolute industrial serial robots for machining operations. In *ASME Biennial Conference on Engineering Systems Design and Analysis*, pages 419–428, Nantes, France, 2012.

[55] Olav Egeland. Task-space tracking with redundant manipulators. *IEEE Journal on Robotics and Automation*, 3(5):471–475, 1987.

[56] M. Elkington, D. Bloom, C. Ward, A. Chatzimichali, and K. Potter. Hand layup: understanding the manual process. *Advanced Manufacturing: Polymer & Composites Science*, 1(3):138–151, 2015.

[57] M. Elkington, C. Ward, and K. D. Potter. Automated layup of sheet prepregs on complex moulds. In *SAMPE Long Beach Conference*, 2016.

[58] M. Elkington., C. Ward, and A. Sarkytbayev. Automated composite draping: a review. In *SAMPE*. SAMPE North America, May 2017.

[59] Brendan Englot and Franz Hover. Sampling-based coverage path planning for inspection of complex structures. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 22, page 392, 2012.

[60] Brendan Englot and Franz S Hover. Sampling-based sweep planning to exploit local planarity in the inspection of complex 3d structures. In *International Conference on Intelligent Robots and Systems*, pages 4456–4463. IEEE/RSJ, 2012.

[61] Qi Fan, Zeyu Gong, Bo Tao, Yi Gao, Zhouping Yin, and Han Ding. Base position optimization of mobile manipulators for machining large complex components. *Robotics and Computer-Integrated Manufacturing*, 70:102138, 2021.

[62] Gualtiero Fantoni, Marco Santochi, Gino Dini, Kirsten Tracht, Bernd Scholz-Reiter, Juergen Fleischer, Terje Kristoffer Lien, Guenther Seliger, Gunther Reinhart, Joerg Franke, et al. Grasping devices and methods in automated production processes. *CIRP Annals*, 63(2):679–701, 2014.

[63] John. T. Feddema. Kinematically optimal robot placement for minimum time coordinated motion. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3395–3400, Minnesota, USA, 1996.

[64] Stefan Flixeder, Tobias Glück, and Andreas Kugi. Modeling and force control for the collaborative manipulation of deformable strip-like materials. *IFAC-PapersOnLine*, 49(21):95 – 102, 2016. 7th IFAC Symposium on Mechatronic Systems MECHATRONICS 2016.

[65] B. Frank, R. Schmedding, C. Stachniss, M. Teschner, and W. Burgard. Learning the elasticity parameters of deformable objects with a manipulation robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1877–1883, Oct 2010.

[66] B. Frank, C. Stachniss, N. Abdo, and W. Burgard. Efficient motion planning for manipulation robots in environments with deformable objects. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2180–2185, Sept 2011.

[67] M. Galicki. Time-optimal controls of kinematically redundant manipulators with geometric constraints. *IEEE Transactions on Robotics and Automation*, 16(1):89–93, Feb 2000.

[68] A Gasparetto and V Zanotto. A new method for smooth trajectory planning of robot manipulators. *Mechanism and machine theory*, 42(4):455–471, 2007.

[69] Emile Glorieux, Pasquale Franciosa, and Dariusz Ceglarek. Quality and productivity driven trajectory optimisation for robotic handling of compliant sheet metal parts in multi-press stamping lines. *Robotics and Computer-Integrated Manufacturing*, 56:264–275, 2019.

[70] Emile Glorieux, Pasquale Franciosa, and Dariusz Ceglarek. Coverage path planning with targetted viewpoint sampling for robotic free-form surface inspection. *Robotics and Computer-Integrated Manufacturing*, 61:101843, 2020.

[71] D. Goldfarb. Extension of davidon's variable metric method to maximization under linear inequality and equality constraints. *Journal on Applied Mathematics*, 17(4):739–764, 1969.

[72] Héctor González-Banos. A randomized art-gallery algorithm for sensor placement. In *Proceedings of the seventeenth annual symposium on Computational geometry*, pages 232–240, 2001.

[73] Grandviewresearch. Composites market article. `https://www.grandviewresearch.com/press-release/global-composites-market`, 2018.

[74] Gudmundur G Gunnarsson, Ole W Nielsen, Christian Schlette, and Henrik G Petersen. Fast and simple interacting models of drape tool and ply material for handling free hanging, pre-impregnated carbon fibre material. In *International Conference on Informatics in Control, Automation and Robotics*, pages 1–25. Springer, 2018.

[75] S. G. Hancock and K. D. Potter. The use of kinematic drape modelling to inform the hand lay-up of complex composite components using woven reinforcements. *Composites Part A: Applied Science and Manufacturing*, 37(3):413 – 422, 2006.

[76] Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[77] James. S. Hemmerle and Fritz. B. Prinz. Optimal path placement for kinematically redundant manipulators. In *IEEE International Conference on Robotics and Automation*, pages 1234–1244, Sacramento, CA, USA, 1991.

[78] Dominik Henrich and Heinz Wörn. *Robot Manipulation of Deformable Objects*. Springer Science & Business Media, 2012.

[79] Adrian Hilton, Andrew J Stoddart, John Illingworth, and Terry Windeatt. Reliable surface reconstruction from multiple range images. In *European conference on computer vision*, pages 117–126. Springer, 1996.

[80] Rachel Holladay, Oren Salzman, and Siddhartha Srinivasa. Minimizing task-space frechet error via efficient incremental graph search. *IEEE Robotics and Automation Letters*, 4(2):1999–2006, 2019.

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

[82] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. 34:189–206, 2012.

[83] Zhe Hu, Peigen Sun, and Jia Pan. Three-dimensional deformable object manipulation using fast online gaussian process regression. *Robotics and Automation Letters*, 3(2):979–986, 2018.

[84] Junsen Huang, Pengfei Hu, Kaiyuan Wu, and Min Zeng. Optimal time-jerk trajectory planning for industrial robots. *Mechanism and Machine Theory*, 121:530–544, 2018.

[85] S. H. Huang, J. Pan, G. Mulcaire, and P. Abbeel. Leveraging appearance priors in non-rigid registration, with application to manipulation of deformable objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 878–885, Sept 2015.

[86] R. Jain. Building an environment model using depth information. *Computer*, 22(06):85–88, jun 1996.

[87] Petr Janoušek and Jan Faigl. Speeding up coverage queries in 3d multi-goal path planning. In *International Conference on Robotics and Automation*, pages 5082–5087. IEEE, 2013.

[88] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, Thomas Funkhouser, et al. Local implicit grid representations for 3d scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6001–6010, 2020.

[89] P Jiménez. Survey on model-based manipulation planning of deformable objects. *Robotics and computer-integrated manufacturing*, 28(2):154–163, 2012.

[90] Wei Jing. *Coverage planning for robotic vision applications in complex 3d environment.* PhD thesis, Carnegie Mellon University, 2017.

[91] Steven G. Johnson. The nlopt nonlinear-optimization package. 2019.

[92] H.V. Jones, A.P. Chatzimichali, R. Middleton, K.D. Potter, and C. Ward. Exploring the discrete tools used by laminators in composites manufacturing: application of novel concept. *Advanced Manufacturing: Polymer & Composites Science*, 1(4):185–198, oct 2015.

[93] 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.

[94] Ariyan Kabir, Alec Kanyuck, Rishi K. Malhan, Aniruddha V. Shembekar, Shantanu Thakar, Brual C. Shah, and Satyandra 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.

[95] Ariyan M Kabir, Krishnanand N Kaipa, Jeremy Marvel, and Satyandra K Gupta. Automated planning for robotic cleaning using multiple setups and oscillatory tool motions. *IEEE Transactions on Automation Science and Engineering*, 14(3):1364–1377, 2017.

[96] Ariyan M Kabir, Alec Kanyuck, Rishi K Malhan, Aniruddha V Shembekar, Shantanu Thakar, Brual C Shah, and Satyandra K Gupta. Generation of synchronized configuration space trajectories of multi-robot systems. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8683–8690. IEEE, 2019.

[97] Ariyan M Kabir, Shantanu Thakar, Rishi K Malhan, Aniruddha V Shembekar, Brual C Shah, and Satyandra K Gupta. Generation of synchronized configuration space trajectories with workspace path constraints for an ensemble of robots. *The International Journal of Robotics Research*, page 0278364920988087, 2021.

[98] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[99] Fouad F Khalil and Pierre Payeur. Dexterous robotic manipulation of deformable objects with multi-sensory feedback-a review. In *Robot Manipulators Trends and Development.* InTech, 2010.

[100] Sheraz Khan, Athanasios Dometios, Chris Verginis, Costas Tzafestas, Dirk Wollherr, and Martin Buss. RMAP: a rectangular cuboid approximation framework for 3d environment mapping. *Autonomous Robots*, 37(3):261–277, feb 2014.

[101] N. Koganti, T. Tamei, K. Ikeda, and T. Shibata. Bayesian nonparametric learning of cloth models for real-time state estimation. *IEEE Transactions on Robotics*, 33(4):916–931, Aug 2017.

[102] Peter J Kolesar. A branch and bound algorithm for the knapsack problem. *Management science*, 13(9):723–735, 1967.

[103] Panagiotis N. Koustoumpardis, Konstantinos I. Chatzilygeroudis, Aris I. Synodinos, and Nikos A. Aspragathos. Human robot collaboration for folding fabrics based on force/rgb-d feedback. In *Advances in Robot Design and Intelligent Control*, pages 235–243. Springer International Publishing, 2016.

[104] Christian Krogh, Jens A Glud, and Johnny Jakobsen. Modeling the robotic manipulation of woven carbon fiber prepreg plies onto double curved molds: A path-dependent problem. *Journal of Composite Materials*, 53(15):2149–2164, 2019.

[105] D. Kruse, R. J. Radke, and J. T. Wen. Collaborative human-robot manipulation of highly deformable materials. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3782–3787, May 2015.

[106] D. Kruse, R. J. Radke, and J. T. Wen. Human-robot collaborative handling of highly deformable materials. In *American Control Conference (ACC)*, pages 1511–1516, May 2017.

[107] Andrew M. Ladd and Lydia E. Kavraki. Using motion planning for knot untangling. *The International Journal of Robotics Research*, 23(7-8):797–808, 2004.

[108] Florent Lamiraux and Lydia E. Kavraki. Planning paths for elastic objects under manipulation constraints. *The International Journal of Robotics Research*, 20(3):188–208, 2001.

[109] Christian Landgraf, Bernd Meese, Michael Pabst, Georg Martius, and Marco F Huber. A reinforcement learning approach to view planning for automated inspection tasks. *Sensors*, 21(6):2030, 2021.

[110] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[111] Steven M LaValle, James J Kuffner, BR Donald, et al. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and computational robotics: new directions*, 5:293–308, 2001.

[112] A. X. Lee, S. H. Huang, D. Hadfield-Menell, E. Tzeng, and P. Abbeel. Unifying scene registration and trajectory optimization for learning from demonstrations with application to manipulation of deformable objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4402–4407, Sept 2014.

[113] A. X. Lee, H. Lu, A. Gupta, S. Levine, and P. Abbeel. Learning force-based manipulation of deformable objects from multiple demonstrations. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 177–184, May 2015.

[114] Jihong. Lee. A study on the manipulability measures for robot manipulators. In *IEEE International Conference on Intelligent Robots and Systems*, volume 3, pages 1458–1465, Grenoble, France, 1997.

[115] Mingyang Li, Zhijiang Du, Xiaoxing Ma, Wei Dong, and Yongzhuo Gao. A robot hand-eye calibration method of line laser sensor based on 3d reconstruction. *Robotics and Computer-Integrated Manufacturing*, 71:102136, 2021.

[116] Y. Li, Y. Yue, D. Xu, E. Grinspun, and P. K. Allen. Folding deformable objects using predictive simulation and trajectory optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6000–6006, Sept 2015.

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

[118] Jeremy Maitin-Shepard, Marco Cusumano-Towner, Jinna Lei, and Pieter Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2308–2315, 2010.

[119] R. K. Malhan, R. Jomy Joseph, A. V. Shembekar, A. M. Kabir, P. M. Bhatt, and S. K. Gupta. Online grasp plan refinement for reducing defects during robotic layup of composite prepreg sheets. In *IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France, May 2020.

[120] R. K. Malhan, A. M. Kabir, B. Shah, T. Centea, and S. K. Gupta. 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.

[121] 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.

[122] R. K. Malhan, Ariyan. M. Kabir, Brual. Shah, Timotei. Centea, and Satyandra. 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, 2018.

[123] R. K. Malhan, Ariyan. M. Kabir, Brual. Shah, Timotei. Centea, and Satyandra. K. Gupta. Automated prepreg sheet placement using collaborative robotics. In *North America Society for the Advancement of Material and Process Engineering (SAMPE)*, CA, USA, 2018.

[124] R. K. Malhan, Ariyan. M. Kabir, Aniruddha. V. Shembekar, Brual. Shah, Timotei. Centea, and Satyandra. K. Gupta. Hybrid cells for multi-layer prepreg composite sheet layup. In *IEEE Conference on Automation Science and Engineering*, Munich, Germany, 2018.

[125] Rishi Malhan, Rex Jomy Joseph, Prahar Bhatt, Brual Shah, and Satyandra K Gupta. Fast, accurate, and automated 3d reconstruction using a depth camera mounted on an industrial robot. In *ASME International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, IDETC/CIE 2021*, 2021.

[126] Rishi Malhan, Rex Jomy Joseph, Prahar Bhatt, Brual Shah, and Satyandra K Gupta. Algorithms for improving speed and accuracy of automated 3d reconstruction with a depth camera mounted on an industrial robot. *Journal of Computing and Information Science in Engineering. Accepted for publication.*, 2022.

[127] Rishi K Malhan, Ariyan M Kabir, Brual Shah, and Satyandra K Gupta. Identifying feasible workpiece placement with respect to redundant manipulator for complex manufacturing tasks. In *2019 international conference on robotics and automation (ICRA)*, pages 5585–5591. IEEE, 2019.

[128] Rishi K Malhan, Aniruddha V Shembekar, Ariyan M Kabir, Prahar M Bhatt, Brual Shah, Scott Zanio, Steven Nutt, and Satyandra K Gupta. Automated planning for robotic layup of composite prepreg. *Robotics and Computer-Integrated Manufacturing*, 67:102020.

[129] Rishi K Malhan, Aniruddha V Shembekar, Ariyan M Kabir, Prahar M Bhatt, Brual Shah, Scott Zanio, Steven Nutt, and Satyandra K Gupta. Automated planning for robotic layup of composite prepreg. *Robotics and Computer-Integrated Manufacturing*, 67:102020, 2021.

[130] Rishi K Malhan, Shantanu Thakar, Ariyan M Kabir, Pradeep Rajendran, Prahar M Bhatt, and Satyandra K Gupta. Generation of configuration space trajectories over semi-constrained cartesian paths for robotic manipulators. *IEEE Transactions on Automation Science and Engineering*, 2022.

[131] Omey M Manyar, Jaineel Desai, Nimish Deogaonkar, Rex Jomy Joesph, Rishi Malhan, Zachary McNulty, Bohan Wang, Jernej Barbič, and Satyandra K Gupta. A simulation-based grasp planner for enabling robotic grasping during composite sheet layup. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 930–937. IEEE, 2021.

[132] 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*, volume 3, pages 2044–2049, Apr 1997.

[133] Bryan J. Martin and James E. Bobrow. Minimum-effort motions for open-chain manipulators with task-dependent end-effector constraints. *The International Journal of Robotics Research*, 18(2):213–224, 1999.

[134] T. Matsuno, T. Fukuda, and F. Arai. Flexible rope manipulation by dual manipulator system using vision sensor. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics. Proceedings (Cat. No.01TH8556)*, volume 2, pages 677–682 vol.2, 2001.

[135] Dale McConachie and Dmitry Berenson. Bandit-based model selection for deformable object manipulation. *arXiv preprint arXiv:1703.10254*, 2017.

[136] Dale McConachie, Mengyao Ruan, and Dmitry Berenson. Interleaving planning and control for deformable object manipulation. In *International Symposium on Robotics Research (ISRR)*, 2017.

[137] Riad Menasri, Amir Nakib, Boubaker Daachi, Hamouche Oulhadj, and Patrick Siarry. A trajectory planning of redundant manipulators based on bilevel optimization. *Applied Mathematics and Computation*, 250:934–947, 2015.

[138] Yuhao Meng, Yujing Chen, and Yunjiang Lou. Uncertainty aware mobile manipulator platform pose planning based on capability map. In *2021 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 123–128. IEEE, 2021.

[139] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019.

[140] Stephen Miller, Mario Fritz, Trevor Darrell, and Pieter Abbeel. Parametrized shape models for clothing. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4861–4868, 2011.

[141] Stephen Miller, Jur van den Berg, Mario Fritz, Trevor Darrell, Ken Goldberg, and Pieter Abbeel. A geometric approach to robotic laundry folding. *The International Journal of Robotics Research*, 31(2):249–267, 2012.

[142] Carmelo Mineo, Stephen Gareth Pierce, Pascual Ian Nicholson, and Ian Cooper. Robotic path planning for non-destructive testing–a custom matlab toolbox approach. *Robotics and Computer-Integrated Manufacturing*, 37:1–12, 2016.

[143] Modernmachineshop. Managing risk is this shop's bold strategy. `https://www.mmsonline.com/articles/managing-risk-is-this-shops-bold-strategy`, 2018.

[144] R. Molfino, M. Zoppi, F. Cepolina, J. Yousef, and E. E. Cepolina. Design of a hyper-flexible cell for handling 3d carbon fiber fabric. *Recent advances in mechanical engineering and mechanics*, 165, 2014.

[145] M. Moll and L. E. Kavraki. Path planning for deformable linear objects. *IEEE Transactions on Robotics*, 22(4):625–636, Aug 2006.

[146] Mark Moll and Lydia E Kavraki. Path planning for deformable linear objects. *IEEE Transactions on Robotics*, 22(4):625–636, 2006.

[147] H Moravec. Robot spatial perceptionby stereoscopic vision and 3d evidence grids. *Perception*, 1996.

[148] A. Nektarios and N. A. Aspragathos. Optimal location of a general position and orientation end-effector's path relative to manipulator's base, considering velocity performance. *Robotics and Computer-Integrated Manufacturing*, 26(2):162–173, 2010.

[149] B. Nelson and M. Donath. Optimizing the location of assembly tasks in a manipulator's workspace. *Journal of Robotic Systems*, 7(6):791–811, 1990.

[150] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew W. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *10th International Symposium on Mixed and Augmented Reality*, pages 127–136. IEEE, 2011.

[151] G.C. Newell and K. Khodabandehloo. Modelling flexible sheets for automatic handling and lay-up of composite components. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 209(6):423–432, 1995.

[152] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (ToG)*, 32(6):1–11, 2013.

[153] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

[154] P Olivieri, L Birglen, X Maldague, and I Mantegh. Coverage path planning for eddy current inspection on complex aeronautical parts. *Robotics and Computer-Integrated Manufacturing*, 30(3):305–314, 2014.

[155] G. Oriolo, M. Cefalo, and M. Vendittelli. Repeatable motion planning for redundant robots over cyclic tasks. *IEEE Transactions on Robotics*, 33(5):1170–1183, Oct 2017.

[156] A Papacharalampopoulos, S Makris, A Bitzios, and G Chryssolouris. Prediction of cabling shape during robotic manipulation. *The International Journal of Advanced Manufacturing Technology*, 82(1-4):123–132, 2016.

[157] Georgios Papadopoulos, Hanna Kurniawati, and Nicholas M Patrikalakis. Asymptotically optimal inspection planning using systems with differential constraints. In *International Conference on Robotics and Automation*, pages 4126–4133. IEEE, 2013.

[158] Sachin Patil et al. Motion planning under uncertainty in highly deformable environments. *Robotics science and systems: online proceedings*, 2011.

[159] Zhang Peng and Li Yuanchun. Position/force control of two manipulators handling a flexible payload based on finite-element model. In *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2178–2182, Dec 2007.

[160] Friedrich Pfeiffer and Rainer Johanni. A concept for manipulator trajectory planning. *IEEE Journal on Robotics and Automation*, 3(2):115–123, 1987.

[161] A. Piazzi and A. Visioli. Global minimum-jerk trajectory planning of robot manipulators. *IEEE Transactions on Industrial Electronics*, 47(1):140–149, Feb 2000.

[162] Roberto Raffaeli, Maura Mengoni, Michele Germani, and Ferruccio Mandorli. Off-line view planning for the inspection of mechanical parts. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 7(1):1–12, 2013.

[163] Pradeep Rajendran, Shantanu Thakar, and Satyandra K Gupta. User-guided path planning for redundant manipulators in highly constrained work environments. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 1212–1217. IEEE, 2019.

[164] Pradeep Rajendran, Shantanu Thakar, Ariyan M Kabir, Brual C Shah, and Satyandra K Gupta. Context-dependent search for generating paths for redundant manipulators in cluttered environments. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5573–5579. IEEE, 2019.

[165] S. Rodriguez, Jyh-Ming Lien, and N. M. Amato. Planning motion in completely deformable environments. In *IEEE International Conference on Robotics and Automation*, pages 2466–2471, May 2006.

[166] ROS-I. Descartes- a ros-industrial project for performing path-planning on under-defined cartesian trajectories. `http://wiki.ros.org/descartes`, 2015. Accessed: January, 2019.

[167] Olivier Roussel, Andy Borum, Michel Taix, and Timothy Bretl. Manipulation planning with contacts for an extensible elastic rod by sampling on the submanifold of static equilibrium configurations. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3116–3121, 2015.

[168] Stuart. J. Russell and Peter. Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[169] M. Saha and P. Isto. Motion planning for robotic manipulation of deformable linear objects. In *IEEE International Conference on Robotics and Automation*, pages 2478–2484, May 2006.

[170] M. Saha and P. Isto. Manipulation planning for deformable linear objects. *IEEE Transactions on Robotics*, 23(6):1141–1150, Dec 2007.

[171] D. W. Sandberg and R. B. Wodtli. Collision detection using sphere approximations. In *Robotics and Factories of the Future*, pages 456–460. Springer, 1988.

[172] John Schulman, Jonathan Ho, Cameron Lee, and Pieter Abbeel. *Learning from Demonstrations Through the Use of Non-rigid Registration*, pages 339–354. Springer International Publishing, Cham, 2016.

[173] G. Seliger, F. Szimmat, J. Niemeier, and J. Stephan. Automated handling of non-rigid parts. *CIRP Annals*, 52(1):21–24, 2003.

[174] Peiyao Shen, Xuebo Zhang, Yongchun Fang, and Mingxing Yuan. Real-time acceleration-continuous path-constrained trajectory planning with built-in tradeoff between cruise and time-optimal motions. *IEEE Transactions on Automation Science and Engineering*, 17(4):1911–1924, 2020.

[175] Xiangling Shi, Honggen Fang, and Lijie Guo. Multi-objective optimal trajectory planning of manipulators based on quintic nurbs. In *Mechatronics and Automation (ICMA), 2016 IEEE International Conference on*, pages 759–765. IEEE, 2016.

[176] Mark W Spong, Seth Hutchinson, Mathukumalli Vidyasagar, et al. *Robot modeling and control*. 2006.

[177] Mahmoud Tarokh and Xiaomang Zhang. Real-time motion tracking of robot manipulators using adaptive genetic algorithms. *Journal of Intelligent & Robotic Systems*, 74(3-4):697–708, 2014.

[178] Holger Täubig, Berthold Bäuml, and Udo Frese. Real-time swept volume and distance computation for self collision detection. In *International Conference on Intelligent Robots and Systems*, pages 1585–1592. IEEE, 2011.

[179] S. Thakar, L. Fang, B. C. Shah, and S. K. Gupta. Towards time-optimal trajectory planning for pick-and-transport operation with a mobile manipulator. In *IEEE International Conference on Automation Science and Engineering (CASE)*, Munich, Germany, Aug 2018.

[180] Shantanu Thakar, Liwei Fang, Brual Shah, and Satyandra 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)*, pages 981–987. IEEE, 2018.

[181] Shantanu Thakar, Ariyan Kabir, Prahar M Bhatt, Rishi K Malhan, Pradeep Rajendran, Brual C Shah, and Satyandra K Gupta. Task assignment and motion planning for bi-manual mobile manipulation. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 910–915. IEEE, 2019.

[182] Shantanu Thakar, Rishi K Malhan, Prahar M Bhatt, and Satyandra K Gupta. Area-coverage planning for spray-based surface disinfection with a mobile manipulator. *Robotics and Autonomous Systems*, 147:103920, 2022.

[183] Shantanu Thakar, Pradeep Rajendran, Vivek Annem, Ariyan Kabir, and Satyandra 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.

[184] Shantanu Thakar, Pradeep Rajendran, Ariyan M Kabir, and Satyandra K Gupta. Manipulator motion planning for part pickup and transport operations from a moving base. *IEEE Transactions on Automation Science and Engineering*, 2020.

[185] Shantanu Thakar, Pradeep Rajendran, Hyojeong Kim, Ariyan M Kabir, and Satyandra K Gupta. Accelerating bi-directional sampling-based search for motion planning of non-holonomic mobile manipulators. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6711–6717. IEEE, 2020.

[186] Mohamed. B. Trabia and Murali. Kathari. Placement of a manipulator for minimum cycle time. *Journal of Robotic Systems*, 16(8):419–431, 1999.

[187] Panagiota Tsarouchi, Jason Spiliotopoulos, George Michalos, Spyros Koukas, Athanasios Athanasatos, Sotiris Makris, and George Chryssolouris. A decision making framework for human robot collaborative workplace generation. *Procedia CIRP*, 44:228–232, 2016.

[188] Dimitrios Tzeranis, Yoshiyuki Ishijima, and Steven Dubowsky. Manipulation of large flexible structural modules by space robots mounted on flexible structures. *Proc. Int. Sym. on Artificial Intelligence, Robotics and Automation in Space*, 2005.

[189] R. Ur-Rehman, S. Caro, D. Chablat, and P. Wenger. Multi-objective path placement optimization of parallel kinematics machines based on energy consumption, shaking forces and maximum actuator torques: Application to the orthoglide. *Mechanism and Machine Theory*, 45(8):1125–1141, 2010.

[190] Nikolaus Vahrenkamp, Tamim Asfour, and Rüdiger Dillmann. Efficient inverse kinematics computation based on reachability analysis. *International Journal of Humanoid Robotics*, 9(04):1250035, 2012.

[191] Nikolaus Vahrenkamp, Tamim Asfour, and Rüdiger Dillmann. Robot placement based on reachability inversion. In *IEEE International Conference on Robotics and Automation*, pages 1970–1975, Karlsruhe, Germany, 2013.

[192] Nikolaus. Vahrenkamp, Tamim. Asfour, Giorgio. Metta, Giulio. Sandini, and Rüdiger. Dillmann. Manipulability analysis. In *International Conference on Humanoid Robots*, pages 568–573, Osaka, Japan, 2012.

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

[194] Jur Van Den Berg, Stephen Miller, Ken Goldberg, and Pieter Abbeel. Gravity-based robotic cloth folding. In *Algorithmic Foundations of Robotics IX*, pages 409–424. Springer, 2010.

[195] J Irving Vasquez-Gomez, L Enrique Sucar, Rafael Murrieta-Cid, and Efrain Lopez-Damian. Volumetric next-best-view planning for 3d object reconstruction with positioning error. *International Journal of Advanced Robotic Systems*, 11(10):159, 2014.

[196] Hidefumi Wakamatsu, Eiji Arai, and Shinichi Hirai. Knotting/unknotting manipulation of deformable linear objects. *The International Journal of Robotics Research*, 25(4):371–395, 2006.

[197] Tiago Rodrigues Weller, Daniel Rodrigues Weller, Luiz Carlos de Abreu Rodrigues, and Neri Volpato. A framework for tool-path airtime optimization in material extrusion additive manufacturing. *Robotics and Computer-Integrated Manufacturing*, 67:101999, 2021.

[198] Mark D Wheeler, Yoichi Sato, and Katsushi Ikeuchi. Consensus surfaces for modeling 3d objects from multiple range images. In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pages 917–924. IEEE, 1998.

[199] Yu-Shiang Wong, Changjian Li, Matthias Nie{ssner, and Niloy J. Mitra. Rigidfusion: Rgb-d scene reconstruction with rigidly-moving objects. *Computer Graphics Forum*, 40(2), 2021.

[200] S. Wright and J. Nocedal. Numerical optimization. *Springer Science*, 35(67-68):7, 1999.

[201] Jingren Xu, Kensuke Harada, Weiwei Wan, Toshio Ueshiba, and Yukiyasu Domae. Planning an efficient and robust base sequence for a mobile manipulator performing multiple pick-and-place tasks. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11018–11024. IEEE, 2020.

[202] Pin-Chu Yang, Kazuma Sasaki, Kanata Suzuki, Kei Kase, Shigeki Sugano, and Tetsuya Ogata. Repeatable folding task by humanoid robot worker using deep learning. *Robotics and Automation Letters*, 2(2):397–403, 2017.

[203] T. Yoshikawa. Analysis and control of robot manipulators with redundancy. In *Robotics research: the first international symposium*, pages 735–747. MIT press Cambridge, MA, 1984.

[204] T. Yoshikawa. Manipulability of robotic mechanisms. *International Journal of Robotics Research*, 4(2):3–9, 1985.

[205] T. Yoshikawa. *Foundations of robotics: analysis and control*. Mit Press, 1990.

[206] F. Zacharias. *Knowledge representations for planning manipulation tasks*, volume 16. Springer Science & Business Media, 2012.

[207] F. Zacharias, C. Borst, and G. Hirzinger. Capturing robot workspace structure: representing robot capabilities. In *IEEE International Conference on Intelligent Robots and Systems*, pages 3229–3236, San Diego, CA, USA, 2007.

[208] F. Zacharias, W. Sepp, C. Borst, and G. Hirzinger. Using a model of the reachable workspace to position mobile manipulators for 3-d trajectories. In *IEEE International Conference on Humanoid Robots*, pages 55–61, Paris, France, 2009.

[209] P. Zhang and Y. c. Li. Simulations and trajectory tracking of two manipulators manipulating a flexible payload. In *IEEE Conference on Robotics, Automation and Mechatronics*, pages 72–77, Sept 2008.

[210] J. Zhao and N. I. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *Transactions on Graphics*, 13(4):313–336, 1994.