ROBOT MANIPULATION PLANNING FOR COMPLEX STRUCTURES

by

ZEBANG ZHANG

A thesis submitted to the University of Birmingham for the degree of
DOCTOR OF PHILOSOPHY

Department of Mechanical Engineering
School of Engineering
College of Engineering and
Physical Sciences
University of Birmingham
March 2022

Abstract

Robots performing pick-and-place tasks need to be given a specific end-effector pose to grasp the object. This pose is usually generated by a grasp planner without considering the future motion of the robot. This is acceptable for manipulating small objects with simple geometry inside a large free space. However, for complex structures being manipulated in a cluttered environment, some grasp poses may lead to redundant robot motion or even failure of transferring the object to the desired goal location. This thesis addresses this limitation by designing methods and algorithms that can solve the integrated problem of grasp selection and motion planning in three different scenarios. In the first scenario, feasible grasps can be sampled directly given the geometry of a complex 3D pipe structure. The robot needs to manipulate the pipe following a given object path while optimising several objectives. In the second scenario, a set of precomputed feasible grasps of an arbitrarily complex object is assumed and only the start and goal pose of the object are given. An optimised path needs to be generated offline and the robot will execute this path repeatedly. Finally, the third scenario assumes the same planning problem as the second scenario but requires the robot to be extremely responsive. The robot needs to generate a feasible motion online as fast as possible to manipulate the object to the desired pose. In these scenarios, a desirable grasp can be chosen, and the subsequent motion of the robot is planned simultaneously by using the proposed methods. The developed methods are generally applicable. However, they will be most useful in cases involving the manipulation of large and complex objects that have many available grasps in a relatively narrow environment. A specific application scenario of the developed algorithms is the Factory-In-A-Box (FIAB) project. The project aims at manufacturing complex pipe structures fully automatically inside a compact container and an industrial robot is employed to transfer pipe structures between different manufacturing processes. Experiments show the advantages of the developed algorithms in terms of both path costs and planning time compared to the traditional manipulation planning method where only a single grasp is generated for motion planning.

Acknowledgements

A doctorate is a long journey. I would not have completed this thesis without the support of my colleagues, friends, and family. It is my great pleasure to take this opportunity to express my gratitude.

First of all, I would like to thank my supervisor, Dr. Mozafar Saadat for giving me the chance to start this journey and always believing in me along the journey. This work would not have been possible without his guidance and encouragement. We have spent a lot of time together discussing ideas and editing papers together. I really thank you for all the efforts you put into me.

I would also like to thank Prof. Duc Pham and Dr. Marco Castellani for sharing their knowledge with me and giving me useful feedback along the journey. A special thank you to Prof. Pham. without his connection with WUT, I may never have had the opportunity to study at the University of Birmingham.

I want to thank my friends and colleagues in the AIM and AUTOREMAN group: Soran, Amir, Ferhat, Marco, Thomas, Arran, Yongjing, Yongquan, Mo Qu, Senjing, Feiying, Kaiwen, Shuihao and everyone else. It is my great pleasure to have you as colleagues. Discussions with you guys have always been fruitful and enjoyable.

I also have the great pleasure to share a flat with some wonderful flatmates: Ruiya, Jiayi, Yibo, Jun Huang, Wupeng and Yue Zang. Thank you all for being part of the great time that we had together and being tolerant to me. Although it's a cheap place, it leaves me with the best memories.

I would like to express my deepest appreciation to my beloved parents. Thank you, mom and dad, I can never make it this far without your mental and financial support. Thank you for always being considerate, reminding me to relax, and encouraging me. The appreciation extends to my grandparents, especially my mother's parents, for bringing me up while my parents were at work. Thank you, grandpa and grandma. I could never ask for better grandparents than you.

This project was funded by Manufacturing Technology Centre. I would like to thank them for their support in case study and allowing me to visit their facilities which inspired a lot of ideas of this thesis.

Finally, this thesis was completed during the pandemic of Covid19. Therefore, I would also like to express my gratitude to whoever devotes their time and efforts to keeping us safe and saving people's lives, especially frontline workers. Thank you all for risking your lives for us.

Contents

Lists of Figures

Lists of Tables

Lists of Notations

General Notation

$\mathbf{A}$:	Upper class characters in bold are matrices.
X:	Upper class characters in italic are spaces or sets, e.g., Robot configuration space X, Graph G, Edge set E,Vertex set V.
$\boldsymbol{x}$:	Lower case characters in this italic and bold are column vectors.
x:	Lower case characters in italic are points in a set or configuration space, e.g., $x \in X_{free}$ means a point x is in the free space, whose specific configuration can be represented by a column vector $\boldsymbol{x}$.
x:	Lower case characters are scalars.

Specific Important Symbols

$\boldsymbol{x}$	The configuration of robot, a column vector.
$\mathbf{A}_{\mathrm{p,i}}$	The pose of a pipe at ith waypoints, a homogeneous transformation matrix with respect to world frame.
$\mathbf{B}_{\mathrm{e}}^{\mathrm{P}}$	The end-effector grasp pose with respect to the pipe local coordinate system.
$\mathbf{T}_{\mathrm{e,i}}$	The end-effector grasp pose with respect to the world coordinate system.
$\mathbf{K}$	A stiffness matrix of the pipe.
$\mathbf{R}$	A rotation matrix.
$\mathbf{I}$	The identity matrix.
$\mathbb{R}$	The set of real numbers.
$\mathbb{R}_{\geq 0}$	The set of nonnegative real numbers.
$\mathbb{R}^{\mathrm{m}\times\mathrm{n}}$	The set of m by n real matrix.
S	The pipe section index.
N	The pipe node index.
ϕ	The grasp angle (see Figure 3.5).
θ	The flip angle (see Figure 3.5).

len	The distance between starting node of a section and the grasp location (see Figure 3.5).
$\mathcal{g}$	A robot grasp $\mathcal{g} = \{\mathrm{S}, \mathrm{len}, \phi, \theta\}$.
$\boldsymbol{w}$	Weights of objectives for optimisation $\boldsymbol{w} = [\mathrm{w}_1\ \mathrm{w}_2\ \mathrm{w}_3]^T$.
σ	The path of the robot, used in sampling-based planners. A path consists of multiple edges each connecting with another.
Σ	A collection of paths σ.
$\mathcal{G}$	A collection of grasps $\mathcal{g}$.
ζ_{d}	Volume of the unit ball in d-dimensional Euclidean space.
d	The dimension of manipulator/planning problem.
n	The number of trajectory waypoints in Chapter 3. The number of sampled vertices in Chapter 4 and Chapter 5.
$\mathrm{r(n)}$	The connection radius for sampling-based planners, which depends on sampled vertices.
$\mathrm{c_i}$	The optimal cost found until ith iteration of a sampling-based algorithm.
$\mathrm{c_{min}}$	The minimum/straight line distance between a pair of start and goal.

List of Abbreviations

FIAB	Factory-In-A-Box
DoF	Degrees of Freedom
BA	Bees Algorithm
GA	Genetic Algorithm
PSO	Particle Swarm Optimisation
AHP	Analytic Hierarchy Process
IK	Inverse Kinematics
PRM	Probabilistic Roadmap
PRM*	Optimal Probabilistic Roadmap
PRM*-MG	Optimal Probabilistic Roadmap for Multiple Grasps
RRT	Rapidly-exploring Radom Trees
RRT*	Optimal Rapidly-exploring Radom Trees
RRT*-MG	Optimal Rapidly-exploring Radom Trees for Multiple Grasps

Chapter 1 Introduction

Robots are widely used in today's factories from automotive to electronics industry. Apart from mass production, deploying a flexible robotic workcell is becoming an increasingly popular choice with the trends of customisation [1], especially for small and mid-size enterprises. A good example of deploying such a flexible robotic workcell is the Factory-In-A-Box (FIAB) project. FIAB is a high-profile project funded by Innovate UK as part of the Thermal Energy Research Accelerator (T-ERA) carried out at the Manufacturing Technology Centre (MTC), Coventry, UK. A simulated FIAB environment is shown in Figure 1.1. FIAB aims at manufacturing cryogenic pipe assemblies inside a compact container with the focus on flexibility (the factory can adapt quickly to manufacture pipes of different structures at different quantities according to customer needs) and mobility (the factory can be rapidly deployed anywhere) [2]. The manufacturing process usually involves cutting, bending, brazing, and pressure testing. An industrial robot is mounted on an overhead rail to grasp and transfer pipes between different stations inside FIAB. During the simulation and the actual operation, the path of the robot is planned fully automatically.

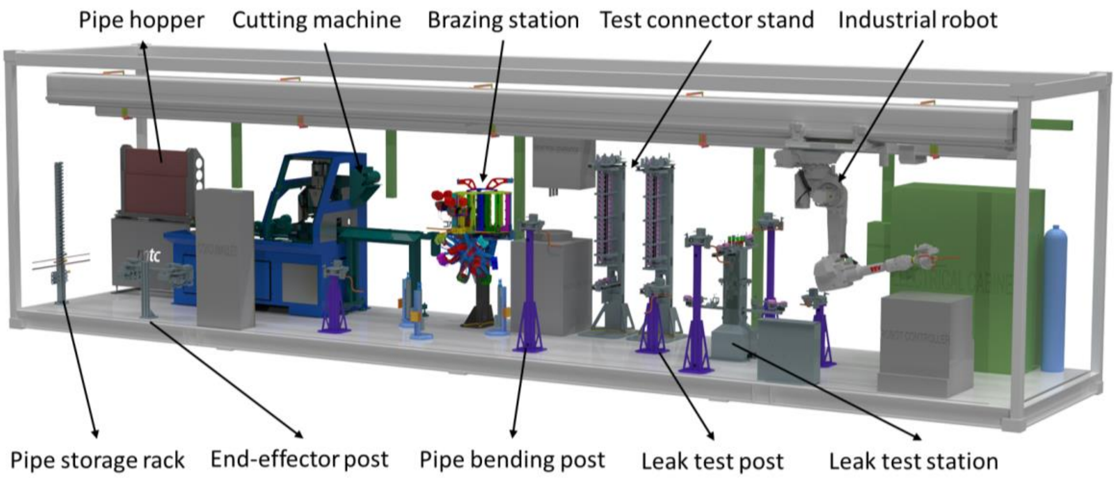


Figure 1.1 A simulated factory-in-a-box layout (Image Courtesy of Manufacturing Technology Centre (MTC), Coventry, UK)

Despite the success of using automated planning in the FIAB project, automated motion planners are far from being perfect. Many examples have shown that in a relatively compact environment and/or if the robot is manipulating a relatively large and complex object, the automated planner is likely to fail when there is an obvious feasible path available for the robot to execute [3–6].

Planning in a compact environment usually induces a common issue in motion planning known as the narrow passage problem [3]. Sampling-based motion planners [7] become inefficient in this case since the probability of generating samples inside the narrow passage is extremely small. However, in the context of manipulation planning, another equally important reason for the failure is because standard motion planning algorithms do not exploit available grasp information sufficiently. Traditional motion planning algorithms always aim at solving a problem given a single start configuration of the robot, but this is not the case for manipulation problems, especially when the objects being manipulated is large and complex. In this context, there are usually many available grasps poses for the robot to choose from. Choosing different grasps leads to different motion planning problems. In each problem, not only the start and goal configuration are different, but also the free configuration space for the robot to move is different because the geometry of the robot-object combination is changed. However, manipulation tasks (e.g., pick-and-place) do not require the robots to be at a specific start or goal configuration, as long as they transfer the object to the desired location and stay away from obstacles. Being able to choose from multiple available grasps brings both opportunities and challenges to the manipulation planning algorithms. In many cases, choosing a good grasp makes the planning problem rather trivial as a straight-line path may be enough to avoid obstacles in the environment. However, an unsatisfactory choice of grasp may result in a very difficult problem where the robot needs to circumvent obstacles carefully, or even a problem that is infeasible for the planner to solve in any reasonable amount of time. Since motion planning algorithms generally build on the assumption that only a single start configuration is available, they are not able to take advantage of multiple grasps to accelerate the search or find a better path. Therefore, currently, the standard planning pipeline for manipulation tasks still decouples the process of grasp planning and the motion planning, which is very inefficient.

1.1 Aims, Objectives and Innovations

This thesis attempts to exploit the opportunity brought by the grasp redundancy (i.e., multiple available grasps) when manipulating complex structures to accelerate the motion planning and find a better path for the robot to execute. The overall aim of this thesis is to develop planners that consider explicitly the impact of grasp choice on the robot motion after grasping.

To achieve this aim, three specific objectives will be completed:

- Develop a grasp pose planning method for robots to manipulate pipe assemblies following a predefined trajectory of the pipe while optimising post-grasp trajectory objectives such as distance of robot joint motion, manipulability, and the deformation of the objects (Chapter 3).
- Given a set of precomputed feasible grasps of an arbitrary object, develop an integrated grasp selection and motion planning algorithm for robots to manipulate objects from start to goal location in the free space (Chapter 4).
- Design a new algorithm for solving the integrated grasp selection and motion planning problem to generate a feasible path as fast as possible (Chapter 5).

Some innovations of the thesis are highlighted as follows:

- In Chapter 3, a novel local search strategy is developed to accelerate the optimisation. Besides, multiple inverse kinematics solutions for a given end-effector pose are considered explicitly to further optimise the objective costs.
- In Chapter 4, a novel algorithm (PRM*-MG) is proposed by converting the single-query manipulation planning problem to a multi-query motion planning problem. Newly designed data structure allows the algorithm can process a large number of

grasps. Batch processing of feasible grasps improves the anytime performance of the algorithm and allows the use of upper bound to prune suboptimal solution.

- In Chapter 5, a novel algorithm (RRT*-MG) is proposed. The proposed algorithm features a new connection criterion that biases the search towards grasps that are more likely to contain the optimal path. In addition, a number of other strategies are proposed to improve the search efficiency by pruning and reusing sampled vertices.

1.2 Layout of this Thesis

The rest of the thesis is structured as follows: Chapter 2 introduces important concepts and reviews existing works related to motion and manipulation planning with a focus on sampling-based methods. Chapter 3 presents a grasp pose optimisation method for manipulating 3D pipe assemblies. The method is specifically developed for manufacturing cryogenic pipe assemblies autonomously in a Factory-In-A-Box scenario, but the same framework can be applied to other types of objects. In Chapter 4, an asymptotically optimal manipulation planning algorithm PRM*-MG is developed to solve the integrated grasp selection and motion planning problem. Chapter 5 presents RRT*-MG, an anytime algorithm that can return the first feasible solution within a very short time and keep improving the solution if more time is available. Finally, Chapter 6 concludes this thesis and provides future research directions.

1.2.1 Logical Sequences of the Chapters

The three main chapters (Chapter 3, Chapter 4, and Chapter 5) of the thesis are organised to consider scenarios from simple to challenging. Chapter 3 considers a very practical and simplified setting. Therefore, the method developed in Chapter 3 can only deal with objects whose grasp poses can be continuously sampled and it requires a reference trajectory as input. In the next Chapter (Chapter 4), the setting is more general and challenging, i.e., the algorithm has no assumption on the object and gripper type, In addition, the reference trajectory is not required. Finally, in Chapter 5, the setting is similar to Chapter 4 except being more time restrictive, i.e., the solution needs to be generated online for the robot to execute as fast as possible.

Chapter 2 Literature Review

The literature review consists of four subsections. Section 2.1 first reviews concepts and definitions in motion planning and then introduces methods developed for the standard motion planning problem, i.e., given a start and a goal (or a goal region), how can a collision-free path be planned. In Section 2.2, methods for manipulation planning especially prehensile manipulation are reviewed. Section 2.2 mainly focus on rigid object, however, many objects in industrial and domestic environment are flexible such as the pipe assemblies manufactured inside FIAB. Therefore, Section 2.3 reviews methods specifically designed for flexible objects like deformable linear objects (DLOs) and compliant sheet metals. Section 2.4 discusses the relationship between this thesis and the reviewed existing studies.

2.1 Motion Planning

Motion planning concerns finding a collision-free path from a start state to a goal state of robot. Before introducing specific class of motion planning algorithms, it is useful to introduce some important concepts and differentiate between several different planning problems.

The configuration (x) of the robot is a set of independent parameters that characterise the position of every point in the robot [8]. The configuration space or C-space (X) of the robot is the space of all possible configurations of a robot. For example:

1. The configuration of a point robot on a 2D plane is defined by its (x, y) coordinates respective to the origin. The configuration space is $\mathbb{R}^2$.

2. The configuration of an autonomous car on a 2D map is given by its 2D position and orientation, i.e., $(\mathrm{x}, \mathrm{y}, \theta)$. The configuration space is the special Euclidean group $SE(2)$.

3. The configuration of a freely movable rigid body in 3D space (e.g., drones) is given by its 3D position $(\mathrm{x}, \mathrm{y}, \mathrm{z})$ and 3D rotation (α, β, γ). The configuration space is the special

Euclidean group $SE(3)$.

4. The configuration of a fixed-base d-axis articulated robot manipulator is given by its joint coordinates $x_1, x_2, \ldots, x_d$. The configuration space is $\mathbb{R}^d$.

Definitions and explanations for some close related problems in motion planning are given below.

- *Path Planning*: Given the environment, path planning aims at finding a collision-free geometric path for the robot to move from its start configuration to a goal configuration.
- *Trajectory Planning*: Given a path of the robot, design a timed trajectory subject to kinematic constraints (usually collision constraints are not considered in the context of trajectory planning).
- *Kinodynamic Planning*: Given the environment, planning a collision-free and dynamically feasible trajectory for the robot to move from its start state to a goal state.
- *Motion Planning*: In many cases, motion planning is equivalent to Path Planning. However, it is also used as an umbrella word for all the planning problems defined above.

In this thesis, only the geometric path of the robot and the object are concerned, leaving the trajectory planning part to the low level controller of robot, which is the usual case for industrial robots. Therefore, this thesis will use "path", "motion", and "trajectory" interchangeably, however, they should all be interpreted as geometric paths unless specifically explained. In addition, "configuration" and "state" will be used interchangeably while they should all be interpreted as robot configurations in $\mathbb{R}^d$, where d is the number of joints or degrees of freedom.

Next, three classes of widely used motion planning methods, namely, sampling-based method, optimisation-based method, and search-based method will be reviewed.

2.1.1 Sampling-based method

Sampling-based motion planning algorithms are perhaps the most widely used algorithms for planning high dimensional robot motion (e.g., manipulators). The basic idea of sampling-based algorithms is to avoid representing the obstacles explicitly, since it is generally difficult in high dimensional configuration space. Rather, they generate random samples in the whole configuration space and use an external collision checker to check whether a specific robot configuration is in collision or not. Probabilistic Roadmap (PRM) [9] and Rapidly exploring Random Trees (RRT) [10] are two most widely used sampling-based algorithms.

PRM is a two phase planner, it first constructs and stores a graph offline by randomly sampling a bunch of configurations and using a simple local planner to connect them (learning phase). Then, during the query phase, the planner connects the start and goal configuration to the offline constructed graph and then uses a graph search algorithm (e.g., Dijkstra's or A*) to search a path from the start to the goal. PRM is probabilistically complete [11] while the simplified version (s-PRM or PRM*) is asymptotically optimal as well [12] with an appropriate connection radius. The aim of PRM is to build a graph that can be used for answering multiple queries. Therefore, the graph needs to cover the entire configuration space. This is efficient for robot that operates in a static environment as the graph is always valid. However, in many cases, the environment may change between different queries, which means the feasibility of the graph also changes. In these cases, variants of PRM such as Lazy PRM [13] or Quasi-random Lazy PRM [14] can be employed. In these variants, the graph is constructed without collision checking, i.e., all edges are considered to be valid. Then, during the query phase, the planner still searches for the shortest path connecting the start and goal. However, this shortest path may contain infeasible edges. Therefore, the planner needs to call a collision checker online to test the feasibility and removes edges in collision. This process (graph search, collision checking, and edge removing) is run iteratively until a feasible path is found.

Incremental tree-based planners like RRT are typically more efficient in the case of single query planning problems. An example of growing RRT in a 2D environment is given in Figure 2.1.

RRT initialises a tree with its root at the start configuration of the robot. In each iteration, RRT samples a random node in the configuration space and queries the nearest neighbour from the current tree. Then, a new node is created. The new node should be closer to the random node than the nearest neighbour. If the new node and the edge connecting the new node to the nearest neighbour is collision free, then the new node and the new edge are added to the tree. During the expansion of the tree, the probability that a node is selected to extend is proportional to the volume of its Voronoi region. In this way, the algorithm biases its search towards the unexplored space. Apart from being efficient in the single query planning problem, RRT can also solve kinodynamic planning problem more easily [15].

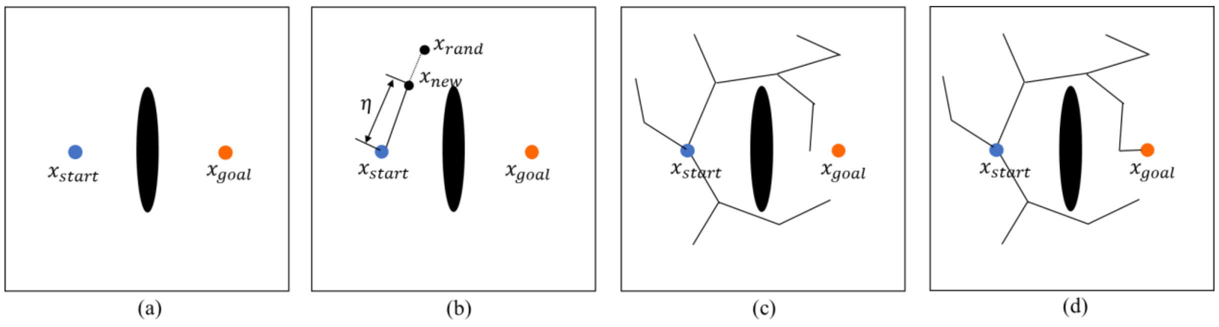


Figure 2.1 An example of a sampling-based planner (RRT) working in a 2D environment. (a) The planning problem. The blue circle is the start configuration while the orange circle is the goal configuration. The black ellipsoid is the obstacle. (b) The planner samples a random configuration x_{rand} in the free configuration space. The tree is extended from the nearest configuration (in this case x_{start}) to x_{rand} with a user defined step size η, creating a new vertex x_{new}. (c) The tree keeps growing, exploring the free configuration space while rejecting connection in collision. (d) A path to goal is found by occasionally sampling the goal configuration directly.

RRT explores the whole planning domain. However, in the case of single query planning problems, only a path between start and goal configuration is of interest. It is possible to avoid exploring the entire planning domain by using a bidirectional variant e.g., RRT-Connect [16]. RRT-Connect grows two trees from the start node and goal node respectively. In each iteration the first tree is grown in the same manner as RRT. If a new node is added successfully to the first tree, the second tree uses the new node as a random sample and attempt to grow towards the new node until a collision is encountered. By the end of each iteration, th two trees are swapped. Therefore, in the next iteration, the new node will be added to the tree that attempts to connect in the current iteration.

heuristically-guided RRT (hRRT [17]) uses heuristic functions to probabilistically reject or accept a uniformly distributed random sample. This strategy guides the search to find low cost solutions. The experiments demonstrate significant improvement in the path quality compared to RRT while very little additional computational costs.

Anytime RRTs [18] is another sampling-based algorithm focuses on improving the path quality found by RRT. The algorithm achieves this behaviour by growing a series of Rapidly-exploring Random Trees and each tree uses information from previous searches to restrict the search domain. In this way, a first solution can be obtained very quickly and by shrinking the search domain, better solution can be found in trees grown in the later iteration.

Although above sampling-based algorithms are fast in generating solutions even in very high-dimensional planning problems, the solution of the generated path is arbitrary. hRRT and Anytime RRTs all try to improve the path quality found by RRT, but no optimality guarantee can be provided by these algorithms. RRT* [12,19] on the other hand, provides asymptotically optimal guarantee for the solution path it finds, i.e., the solution found by RRT* will converge to the optimum with probability 1 if given an infinite number of samples.

Instead of always connecting the new vertex to the nearest vertex in the tree like RRT, RRT* considers a neighbourhood and connects the new vertex to the vertex in this neighbourhood that has the least cost-to-come value to the new vertex. After connecting the new vertex to the tree, RRT* also checks whether the connected new vertex can improve the path to reach other vertices in the neighbourhood. If such an improvement can be made, the parent of the improved vertex will be rewired to the new vertex. This allows RRT* to improve the existing tree with the help of future samples.

However, RRT* is not a particularly fast algorithm, it usually converges very slowly in practice. One of the reasons for the slow convergency is because RRT* attempts to find the optimal path to every point in the planning domain, which is not necessary in the case of single query motion planning since the problem only cares the optimal path to a specific goal configuration.

Therefore, many variants [20–24] and other asymptotically optimal planners [25–27] were proposed to accelerate the convergency of RRT*. Two variants Bi-RRT*[24] and Informed-RRT* [22,23] will be introduced in detail in Chapter 5.

2.1.2 Optimisation-based method

Sampling-based planning algorithms can find feasible or even optimal solutions when solving very high dimensional difficult problems. However, optimisation based method provides distinct features like being more natural in incorporating other types of objectives other than path length and generating smooth path without post processing. However, optimisation-based algorithms generally suffer from local optimality, and if the local optimum is an infeasible solution, optimisation-based method may fail on this problem. Therefore, optimisation-based methods do not have guarantees like probabilistic completeness and asymptotic optimality, and its performance relies heavily on a good initial guess. Otherwise, the method may need to restart multiple times before finding a satisfactory solution path. The behaviour of an optimisation-based planner is shown in Figure 2.2.

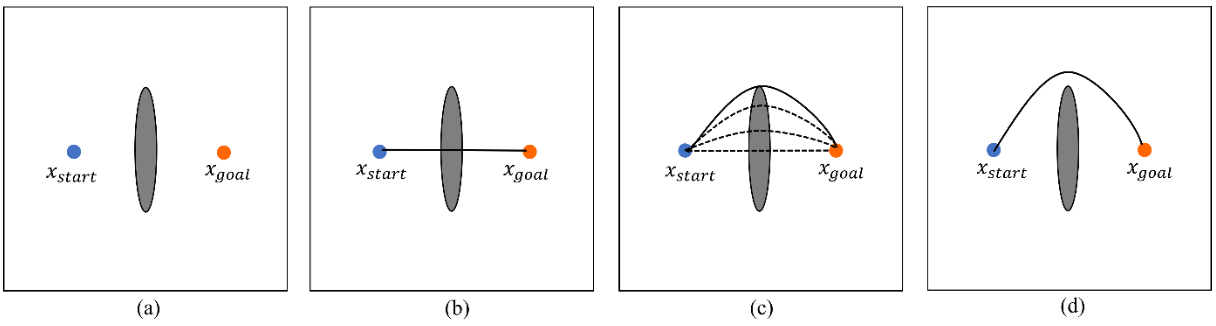


Figure 2.2 An example of an optimisation-based planner working in a 2D environment. (a) The planning problem. The obstacle is shown in grey. The blue circle is the start configuration while the orange circle is the goal configuration. (b) An initial guess to the solution is provided as input. (c) The planner keeps optimising the objective costs (here the objective costs contain both path length and collision cost) by generating a series of trajectories. (d) A solution path is returned. Following this path, the robot travels a short distance as well as stays away from the obstacle.

CHOMP (Covariant Hamiltonian Optimisation for Motion Planning [28,29]) solves the trajectory optimisation planning problem using covariant gradient descent. The method considers two objectives, i.e., the smoothness objective and the obstacle objectives. However, other objectives and constraints can also be dealt with. Smoothness is defined as the squared velocity along the trajectory. The second objective is related to obstacles, which is defined as the integral of the maximum obstacle cost over the body along the trajectory. The obstacle cost is defined by a function over precomputed signed distance field where points in the interior of the obstacles have higher costs, and points away from obstacles have zero costs. The robot manipulator is approximated by a set of spheres in order to perform fast collision queries. CHOMP also uses Hamilton Monte Carlo algorithm to perturb the trajectory out of local minima.

STOMP (Stochastic Trajectory Optimisation for Motion Planning [30]) presents another method for solving motion planning problem using a trajectory optimisation framework. The optimisation objectives and obstacle representations are similar to CHOMP. However, instead of using gradient based techniques for optimisation, STOMP uses a gradient free stochastic optimisation approach. In each step, a bunch of noisy trajectories are created and evaluated. Then the probability of contribution of each trajectory to the update is computed. High quality trajectories will contribute more to the update, while lower quality trajectories contribute less. STOMP has two advantages over CHOMP. The first one is the method does not need the objective being differentiable since it uses derivative free method for optimisation. The second one is due to the stochastic nature of the update rule, it is less prone to local optima.

TrajOpt [31,32] uses sequential convex optimisation to solve the optimisation problem. The outer loop of the optimisation is a penalty loop which drives the constraint costs to zero by increasing the penalty coefficient each iteration if the constraint is not satisfied after the finishing optimisation process of inner loop. TrajOpt also features a different collision formulation. Instead of using signed distance field, it computes the signed distance between two convex objects directly by GJK [33,34] (for minimum distance) and EPA [35] (for penetration depth) directly. In this way, the collision checking is performed without approximation as in CHOMP. TrajOpt also consider continuous time collision checking instead of discrete time collision checking. This allows few waypoints being generated when creating the trajectory which further accelerate the method.

Other optimisation based motion planning approaches includes GPMP (Gaussian Process Motion Planner) and GPMP2 [36–38], ITOMP (Incremental Trajectory Optimization) [39], Functional Gradient Motion Planning [40] and CFS (Convex Feasible Set) [41] etc.

2.1.3 Search-based method

The third class of method for solving motion planning is search-based method. Search-based methods rely on discretisation of the planning domain to individual states (or configurations), and the fact that states adjacent to each other are connected (i.e., can be reached directly from one to another).

Most search-based methods rely on classic graph search algorithms such as Dijkstra's algorithm [42] and A* algorithm [43].

Dijkstra's algorithm computes the shortest distance from the start state to every state in the planning domain. The algorithm constructs a queue of vertices ordered by the cost-to-come value. The cost-to-come values for all vertices are initialised to infinity except the start vertex, which is initialised to zero. In each iteration, the algorithm removes the top of the vertex queue (i.e., the vertex with the least cost-to-come value) and updates the cost-to-come values for its children. In the case of a single goal state, the algorithm can stop as soon as the goal is removed from the queue. Figure 2.3 shows how Dijkstra's algorithm works in a 2D environment.

Dijkstra's is an uninformed algorithm, i.e., it does not use any information about the goal state. A* on the other hand modifies the priority associated with the queue from cost-to-come to the sum of cost-to-come and optimistically estimated cost-to-go. This allows the algorithm to always focus search on the vertices that can potentially provide better solutions. A* works the same as Dijkstra's algorithm except using a different priority. A* is also optimal in the number of vertices expanded before finding the goal. It is proved in [43] that any resolution optimal algorithm will expand at least same number of vertices as A* when using a same heuristic.

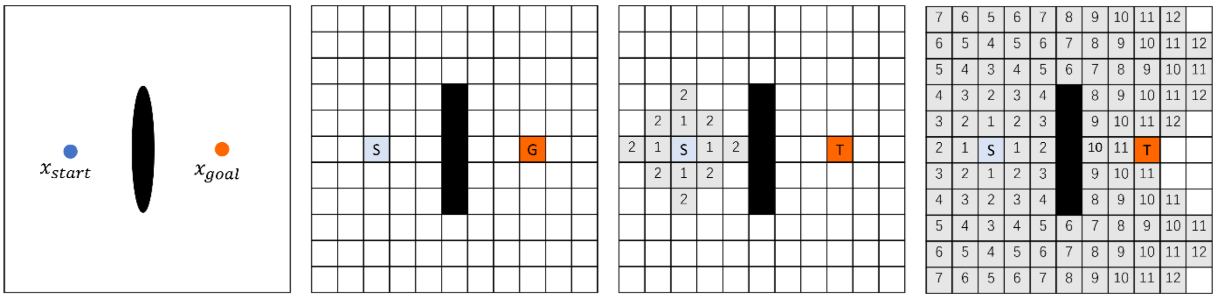


Figure 2.3 An example of a search-based planner (Dijkstra's Algorithm) working in a 2D environment. (a) The original motion planning problem. The blue circle is the start state while the orange circle is the goal state. The black ellipsoid is the original obstacle. (b) The problem is discretised into grid world representation. (c) Dijkstra's expands to neighbouring states uniformly according to cost-to-come. The number on the grid is the distance to start state (assuming no diagonal movement is allowed). (d) The next state to be expanded will be the target. However, the shortest distance to the target can only be determined when it is removed from the queue in the future.

However, A* as a resolution-optimal algorithm finds difficulty in planning for high dimensional configuration space e.g., manipulators. Methods exist to accelerate A* at the cost of sacrificing optimality. For example, weighted A* [44,45] inflates the admissible heuristic by $\epsilon > 1$ which bias the search towards vertices that closer to the goal and expand fewer vertices that A*. However, the solution cost could be ϵ times worse than the optimal cost.

Search-based techniques are also good at replanning. LPA* [46] searches first solution as A* but can keep finding new solutions quickly if the edge costs are changed. D*[47], focussed D* [48] and D* lite [49] also attempt to quickly replan on a changing graph, However, they assume a robot is moving on the graph. Therefore, the start state is changed each time a path needs to be replanned.

For a long time, search-based methods are considered not suitable for manipulation problems due to the high dimensionality of such problems. However, recently, it has been shown that by using carefully designed heuristics and graph representations, the performances of search-based methods are comparable to sampling-based and optimisation-based algorithms [50–55]. Search-based methods do not rely on random samples and can generally provide optimality guarantees for given resolutions, which is arguably more desirable than sampling-based planners. However, designing informative heuristics remains to be a challenge for search-based method in solving high-dimensional problems.

2.1.4 Combined Sampling and Optimisation

Recently, there is a trend in merging methods from different classes of algorithms, especially combining sampling-based algorithm and optimisation based method [56–63]. This is due to the fact that sampling-based methods can generally avoid local optimum while optimisation based methods can locally improve a feasible path very efficiently. Some important methods in this class are briefly introduced below.

A parallel planning method is presented in [56] for cartesian trajectory planning. The method precomputes a roadmap that considers static obstacle. After receiving a planning request, a few paths are computed from the roadmap and then parallelly optimised by a CHOMP like planner considering singularity, static and dynamic obstacles as well as the violation of Cartesian constraints. By using such a parallel scheme, the success rate in solving the motion planning problem is increased while the planning time is reduced.

BiRRTOpt [57] uses bidirectional RRT (RRT-Connect [16]) to search an initial feasible solution and then pass to TrajOpt for optimisation. The combined method is faster than running TrajOpt from scratch. In addition, since the initial solution is provided by a probabilistically complete planner, the combined method is also probabilistically complete.

Regionally Accelerated Batch Informed Trees (RABIT* [58]) uses CHOMP to accelerate the search of BIT*. Instead of optimising the entire trajectory computed by BIT*, this method optimised edges before adding them to the graph if the heuristic cost of the edge is less than its true cost. Optimising individual edges avoids the paths computed by sampling-based planners being too long (i.e., containing too many waypoints), which would be difficult for the optimisation-based planners to process.

Dancing PRM* and Batch Dancing Tree [59,60] are hybrid motion planning algorithms that do not rely on priori information of the environment. Instead, the methods learn spatial collision information during execution of the algorithm and use this information to approximate collision-free configuration space, ultimately guiding trajectories towards these regions. They rely on Lazy PRM* and BIT* for sampling-based planning respectively while the trajectory optimisation is based on CHOMP.

2.2 Manipulation Planning

Robotic manipulation can consist of many different tasks related to objects being moved by robots e.g., pick-and-place, in-hand manipulation, mechanical assembly of parts etc [64]. In this thesis, the focus is on pick-and-place tasks and no regrasping is allowed. Although the setup seems to be simple and restrictive, it is actually the most common manipulation tasks performed by robots in industry. More general setups consider the problem of regrasping and movable obstacles inside environment [65–71]. However, these manipulation strategies are usually avoided in the real production environment.

2.2.1 Grasp Planning

In order to perform pick-and-place manipulation planning, this thesis assumes grasps can be directly generated from simple geometric primitives (e.g., cylinders or a combination of multiple cylinders) (Chapter 3), or a set of feasible grasps is given a grasp planner (Chapter 4 and Chapter 5). Generated grasp candidates should consist of a 6-DoF end-effector pose and an d-dimensional hand/gripper configuration (e.g., a simple parallel two-finger gripper has one DoF). There are two class of grasp planning approaches i.e., analytical approaches and empirical (or data-driven) approaches [72–74]. The analytical method aims at finding a force closure or form closure grasp while optimizing some other objectives [75,76]. Recently, deep learning based data-driven methods like [77–81] have become more popular because they do not require geometry information as a priori information and can detect grasps for unseen objects using vision.

2.2.2 Pre-grasp Motion and Manipulation Planning

The grasp motion alone is only one stage of the entire manipulation process. Before grasping, the robot needs to reach the desired grasp pose. After successfully grasping the object, the robot manipulator is expected to move the object to the goal pose and perform subsequent tasks. Therefore, pre-grasp and post-grasp manipulation are of great importance in planning the entire robot trajectory.

Many integrated approaches have also been proposed to solve the combined problem of grasping and Pre-grasp motion planning. GraspRRT is proposed in [82] to integrate grasp planning and pre-grasp motion planning. To evaluate the quality of reachable grasp poses, grasp wrench space analysis is used. The same problem is studied in [83], where independent contact regions are used to achieve greater robustness in grasping. A hierarchical contact optimisation method is integrated with a sampling-based motion planner (CBiRRT [84]) to simultaneously plan the collision-free arm motion as well as the fingertip grasp in [85]. A heuristic-guided sampling-based search algorithm in proposed in [5] to increase the success ratio for planning pre-grasp motion in dense clutter. The method first planning motion for the end-effector only ignoring the constraint of manipulator and then use the planning results as a heuristic to guide the planning for the manipulator.

In some cases, where a single grasp cannot produce desired motion to satisfy the task requirement, the robot can perform pre-grasp manipulation (e.g., sliding, pushing or rotation [86–89]) to reorient the object to the desired pose. However, this is also generally avoided in production environment.

2.2.3 Grasp selection based on post-grasp motion objectives

The impact of choosing different grasp poses on the post-grasp manipulation process has been analysed in [6,90–92]. In their work, the optimal path of the manipulated object is assumed to be found either by human knowledge or an optimal motion planner. Then a few grasp candidates are generated by a grasp planner. Next, for each grasp pose, an inverse kinematics (IK) solver is used to generate robot motion and an objective function is defined to evaluate the quality of the grasp candidates. In this way, a more efficient grasp pose is found by considering the robot motion after grasping. The objective used in [6] is to maximise its distance from any collisions along the trajectory while [91] considers how the impact force can be minimised by choosing appropriate grasp if a collision does happen. The inertia and dynamics of the grasped objects are considered in [90] when choosing a grasp, which results in optimised robot joint torques for executing a specified object trajectory. A human-in-the-loop scenario is considered in [92], where the human is assisted by an autonomous agent to select grasp that maximise the post-grasp velocity manipulability along a specified trajectory.

2.2.4 Grasp Optimised Motion Planning

Methods presented in this section attempt to solve the integrated grasp and motion planning problem. Unlike methods presented in Section 2.2.2, The entire manipulation process or at least the post-grasp manipulation is considered. Considering post-grasp manipulation brings special challenges to the planning problem since there are more than one potential start configuration (grasp) for robot to choose. Besides, choosing different start configurations (grasps) leads to different collision checking results for the same robot configuration, which means the planning results for one grasp cannot be reused efficiently by another grasp.

Therefore, methods developed for this problem are mainly optimisation-based, attempting to approximate a continuous cost function over the grasp and trajectory space in [93–95]. However, the fundamental drawback of these methods is they are very conservative about the shape of the object being manipulated (mainly cuboid, sphere, cylinder etc.) and also the gripper type (mainly two finger parallel gripper) since the grasp space for complex objects is difficult to parametrise.

An optimal control problem is considered in [93] so that the locally optimal grasp contact position, grasping force and robot arm trajectory can be obtained by solving a single optimisation problem. However, they only consider two dimensional objects. A multi-level optimisation framework is proposed in [94]. The framework can simultaneously plan the grasping location, robot configurations for pick, drop, and handover objects, as well as the collision-free arm motion for dual-arm assembly tasks. However, only cuboid objects are considered with potential extension to other trivial objects like spherical and cylindrical objects. The grasp space is divided into multiple regions in [95] to allow continuous optimisation on each region, however, the method can only be applied to limited type of objects as [93] and [94].

In [96], the grasp optimized motion planning problem is studied. After the grasp planner generates a feasible grasp, an optimisation-based planner is used to find the optimal grasp pose (while maintaining the same contact positions as the initial grasp) as well as generate the post-grasp motion. This method can extend to arbitrary objects, but it needs to solve an optimisation problem for each single grasp generated by the grasp planner, which is not efficient in case of many feasible grasps are available.

2.3 Manipulation Planning for Non Rigid object

The methods reviewed in Section 2.2 are mainly for grasp and motion planning of rigid objects, However, many objects in industrial and domestic environments are deformable. Manipulation of deformable objects has been reviewed in [97–99]. In this section, the manipulation strategies for two classes of deformable objects related to pipe assemblies are reviewed.

2.3.1 Deformable Linear Object

This thesis analyses the manipulation of 3D pipe (or in general, frame) structures. The 3D pipe is bent or assembled by joints from multiple one-dimensional pipes. A 1D pipe can be viewed as a deformable linear object (DLO), i.e., it is much larger along one dimension than the other two. Manipulation planning strategies have been widely researched for DLOs [100–102]. In [100], A two-phase path planner together with a cable pose measurement approach is proposed for cable grasping. The method focuses on the pre-grasp stage and uses force directional manipulability to determine the optimal grasp pose. A method for automatic mating of a wire harness onto a car body by wire tracing operation is proposed in [101]. In [102], A manipulation framework is proposed to shape the cable by environmental contacts. However, common techniques used in the literature for DLO manipulation like structure reshaping and exploiting environmental contacts are not suitable for 3D pipe manipulation due to the complex structure and relatively large strain. Instead, the method presented in Chapter 3 of this thesis aims at maintaining the geometry and reducing the deformation of the pipe during the manipulation process.

2.3.2 Compliant Sheet Metal

Researchers have investigated the problem of minimising deformation when handling compliant sheet metal parts. A trajectory optimisation approach is used in [103] to plan a minimal deformation trajectory while maintaining the same productivity. A response surface model is generated based on finite element analysis (FEA) of the handled part and end-effector, so that deformation can be quickly estimated during optimisation. Other approaches attempt to reduce the deformation of an object during handling by designing a part-specific end-effector layout. A simple methodology is proposed in [104] to determine the number of vacuum cups needed for handling a compliant sheet metal part as well as their locations. The method focuses on placing vacuum cups evenly based on the gravity distribution of the object. Although the deformation and holding force decrease significantly by using the proposed method, the deformation information is not directly used during the optimisation process, which means the positions of the end-effector could be further optimised. A methodology is proposed in [105] to optimise the design of end-effector and robot motion simultaneously in order to achieve less cycle time as well as less deformation for a multi-robot system. Although the deformation information is used directly, it also relies on a precomputed model to estimate the deformation during optimisation as in [103], which is not suitable for manipulating objects with different dimensions and geometries.

2.4 Discussion

The current literature on motion and manipulation planning can generate a feasible solution very fast and find a near-optimal solution given more time. However, only a few works consider the impact of selecting different grasps on the post-grasp robot motion as reviewed in Section 2.2.3 and 2.2.4. They also tend to formulate the problem in a very restrictive way (e.g., limited number of feasible grasps, precomputed deformation models, and/or simple gripper and object models). This thesis attempts to address these issues.

Specifically, Chapter 3 studies a similar problem as reviewed in Section 2.2.3 and 2.3.2. However, unlike works in Section 2.2.3 which assume a small number of grasps are available, this thesis deals with the case that grasps can be continuously sampled. Besides, Chapter 3 also considers the impact of grasp on the grasped object, while works in Section 2.2.3 focus on the impact on robots and humans. Unlike methods reviewed in Section 2.3.2, the end-effector used in Chapter 3 is a general two-finger gripper so that the optimisation is performed at the actual manipulation stage rather than the design stage and no precomputed deformation model is required. Hence, the proposed method is more versatile to different pipe geometries.

Chapter 4 and Chapter 5 study a similar problem to Section 2.2.4. However, the formulation in this thesis is more versatile i.e., no constraint is put on the type of gripper or the object shape. As long as a suitable grasp planner is available, the methods will work. Besides, one of the planners proposed in this thesis can find globally asymptotically optimal paths, while methods in Section 2.2.4 are only locally optimal. However, since methods in Section 2.2.4 are optimisation-based, they usually generate smoother trajectories and can be executed on the robot without further processing.

Methods presented in Section 2.2.2 all focus on pre-grasp manipulation strategies and motion planning. They can be viewed as a complementation to methods developed in this thesis since it is possible to use the methods in Section 2.2.2 to generate multiple reachable grasps and then pass to planners developed in this thesis to select the optimal grasp based on post-grasp motion costs. It should be noted that, although this thesis also studies the integrated grasp and motion planning problem, methods reviewed in Section 2.2.2 cannot be applied to the problem studied in this thesis because of two fundamental differences. The first one is methods in Section 2.2.2 assume the robot starts from a single start configuration and has a goal region to reach. The second difference is methods in Section 2.2.2 do not need to consider the problem of change of geometry when grasping objects with different poses since they focus on the pre-grasp stage. These two differences make the pre-grasp motion planning problem satisfies the assumption of most motion planning algorithm thus modifying existing motion planning algorithms to solve the pre-grasp motion planning problem is more straightforward.

Chapter 3 Multi-objective grasp pose optimisation for robotic 3D pipe assembly manipulation

3.1 Introduction

Currently, the solution for grasping pipe assemblies in FIAB is by designing a part-specific end-effector. While this solution provides a robust grasp and reduces excessive deformation during manipulation, it also increases the weight of the end-effector which makes the process less energy efficient. Besides, as the custom-designed end-effector is larger and more complex in terms of its geometry, some pipes may deem to be infeasible to manufacture due to possible collision with the end-effector. More importantly, the custom-designed gripper is much more expensive than a general gripper.

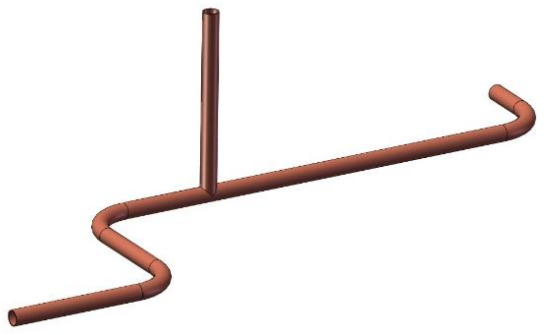


Figure 3.1 A 3D pipe assembly.

Therefore, this chapter proposes a method based on using a general two-finger gripper. As shown in Figure 3.1, the pipe assembly consists of several sections. The simple cylindrical shape of each section makes it possible for the assembly to be grasped by a low-cost parallel gripper robustly with a properly designed attachment (as shown in Figure 3.2). Although the grasping motion alone is easy, other issues may arise when the robot motion after grasping is considered. For example, some grasp poses may lead to collisions between the robot (or grasped object) and the environment. This is very common inside FIAB since the space is relatively compact. Other grasp poses may lead to redundant motion of the robot, proximity to robot singularity and excessive deformation of the pipe assemblies. Therefore, an optimisation process needs to be implemented to find a suitable and possibly optimal grasp pose for the pipe assembly.

In order to address these issues, this chapter considers the problem of grasp pose optimisation for manipulating 3D pipe assemblies during the manufacturing process. The method presented in this chapter is specifically developed for manufacturing cryogenic pipe assemblies autonomously in FIAB. However, it also can be used for robotic manipulation of general frame structures.

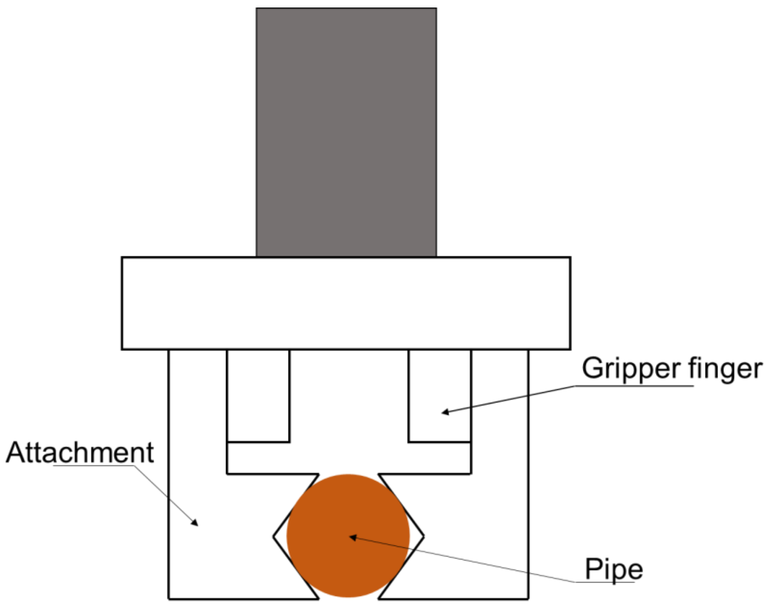


Figure 3.2 The attachment design for securely grasping the pipe.

The problem is formulated as a constrained multi-objective optimisation problem. The optimisation algorithm first searches for solutions that satisfy two constraints: (i) robot workspace reachability (i.e., feasible inverse kinematics solution for a given end-effector pose); and (ii) any possible collision when executing the post-grasp motion, and continuously improves them based on three objectives: (i) minimise robot joint motion for executing a specified pipe trajectory; (ii) minimise the sum of maximum deformation of pipe assembly along the trajectory; and (iii) minimise the sum of robot force manipulability along the trajectory.

The proposed method assumes robot perform repetitive tasks in a static environment. Therefore, once the optimal solution is computed offline, the robot can execute the trajectory repetitively with the optimal grasp to increase the production efficiency. Previous work on the similar problem like [6,90–92] only deals with a limited number of grasp candidates, thus no optimisation algorithm is used. This chapter follows the same assumption in [6] that an optimal object trajectory is known beforehand. However, this chapter considers deformation and other objectives that are not used in previous work and uses Analytical Hierarchy Process (AHP) to determine weights for each objective. A special constraint handling method is also used to decouple the constraint and objective evaluation process, allowing expensive objectives (e.g., deformation) to be evaluated only when constraints are satisfied, thus significantly reducing the computation time. In addition, The algorithm explicitly considers the possibility of multiple inverse kinematics (IK) solutions (for the same end-effector pose) and uses a graph search algorithm (Dijkstra's Algorithm) to find the optimal trajectory among all feasible trajectories, thus further optimising the objective costs. The optimisation problem is solved using the Bees Algorithm with a proposed problem-specific local search strategy. Extensive benchmarks show that the proposed strategy achieves better overall results than the default strategy of the Bees Algorithm and other metaheuristics.

The remainder of this chapter is structured as follows: Section 3.2 formulates the problem, in which optimisation variables, objectives, and constraints are modelled. Section 3.3 describes the details of the objective and constraint evaluation method. Section 3.4 introduces the optimisation algorithm used in this chapter and proposes problem-specific search strategies. Section 3.5 presents the experiments, results, and analyses. Section 3.6 concludes this chapter.

3.2 Problem Formulation

3.2.1 Problem Description

The simulated environment is shown in Figure 3.3. The pipe is placed on a fixture for the robot to grasp. The initial pose of pipe is denoted as $\mathbf{A}_{\mathrm{p},0}$. $\mathbf{A}_{\mathrm{p},0}$ is a homogeneous transformation matrix with respect to world frame, i.e.,

$$\mathbf{A}_{\mathrm{p},0} = \begin{bmatrix} \mathbf{R} & \boldsymbol{p} \\ \mathbf{0} & 1 \end{bmatrix},$$

where $\mathbf{R} \in \mathbb{R}^{3\times3}$, $\boldsymbol{p} \in \mathbb{R}^3$, $\mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = \mathbf{I}$, $\det(\mathbf{R}) = 1$.

After successfully grasping the pipe, the robot will transfer the pipe to a desired pose $\mathbf{A}_{\mathrm{p,n}}$. The trajectory of the pipe is denoted by a series of waypoints: $\mathbf{A}_{\mathrm{p},1}, \mathbf{A}_{\mathrm{p},2}, \dots, \mathbf{A}_{\mathrm{p,n}}$.

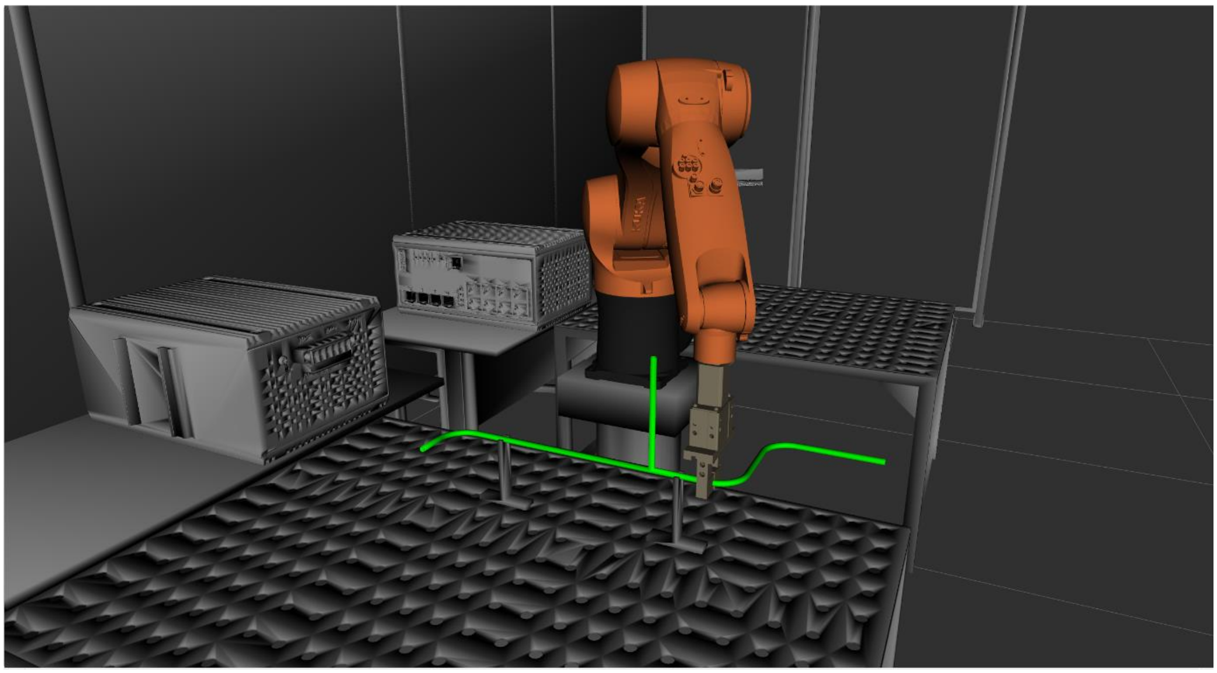


Figure 3.3 A robot picks up a pipe assembly and transfer it to the desired pose

As mentioned above, with a proper design, the end-effector can grasp almost at any position of the pipe. However, not every pose is reachable for the robot to perform grasping, or even if the grasping pose is reachable, the predefined pipe trajectory cannot be followed exactly. In order to achieve fast and efficient production as well as maintain the quality of the pipe, there are several issues that need to be considered when the robot chooses a pose to grasp the pipe:

- the trajectory of the pipe can be followed exactly.
- there is no collision between the robot (with the grasped pipe) and the environment or self-collision.
- minimise the robot joint motion distance.
- minimise the deformation of pipe along the trajectory.
- minimise force manipulability of robot along the trajectory.

The first two issues are modelled as constraints and the latter three are modelled as objectives in Section 3.2.3.

3.2.2 Abstracted Pipe Geometry

The pipe assembly is abstracted to simplify the process of generating grasp poses and computing the deformation after being grasped by the robot. The abstraction of the 3D pipe assembly in Figure 3.1 is shown in Figure 3.4. The assembly consists of several sections $(S_1, S_2, S_3, \ldots)$. Each section is defined by two nodes. Each node stores its own 3D position with respect to its local coordinate system. Each section stores properties like diameter, wall thickness and material density. In this way, theoretically, the pipe can consist of sections with different materials (PVC and copper) and diameters. The geometry of the pipe assembly in Figure 3.4 can be defined by a 6 by 2 matrix $\begin{bmatrix} 1 & 2 & 3 & 4 & 4 & 6 \\ 2 & 3 & 4 & 5 & 6 & 7 \end{bmatrix}^T$, where each entry of the matrix is the index of the node and each row defines a section.

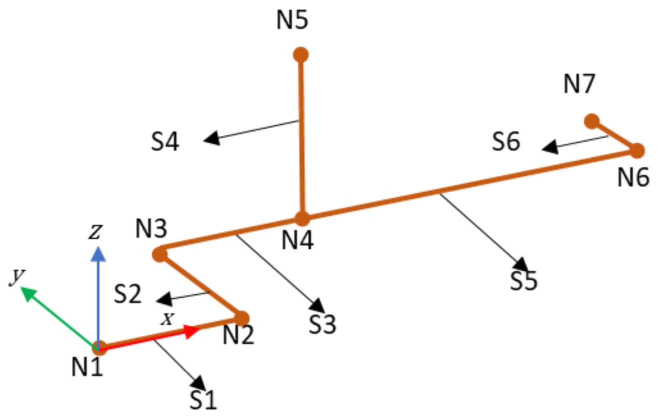


Figure 3.4 An abstracted pipe assembly in its local frame.

3.2.3 Mathematical Modelling

In this section, the optimisation variables, constraints, and objectives of the problem are modelled based on the issues discussed in Section 3.2.1.

3.2.3.1 Optimisation Variables

The problem is aimed at finding the optimal grasp pose. A grasp pose is essentially a rigid body transformation and can be represented by a homogeneous transformation matrix. In our problem, the grasp pose must satisfy certain constraints to enable feasible grasping. Specifically, the position vector must be on the centreline of the pipe and the orientation vector has to be perpendicular to the pipe centreline. Given the geometry of pipe assembly, a grasp pose can be defined with 4 variables as shown in Figure 3.5. The definitions of the variables are given as follows:

- grasp section (e.g., $S1, S2, S3, \ldots$)
- grasp position (len): this is the length between the starting node of a section (e.g., the starting node of $S1$ is $N1$) and the grasp location (where gripper TCP is placed).
- grasp angle (ϕ): this is the angle between the y-axis and the gripper approaching direction (grey arrow).

- flip angle (θ): This angle is around the gripper approaching direction (grey arrow) and can only be either 0 or 180 due to geometric constraint of pipe and the design of parallel gripper.

Given the geometry of pipe P, grasp $\mathcal{g} = \{\mathrm{S}, \mathrm{len}, \phi, \theta\}$ can then be generated. Knowing the grasp parameter, it becomes convenient to calculate the end-effector grasp pose $\mathbf{B}_{\mathrm{e}}^{\mathrm{P}}$ in the matrix form with respect to the pipe local coordinate system.

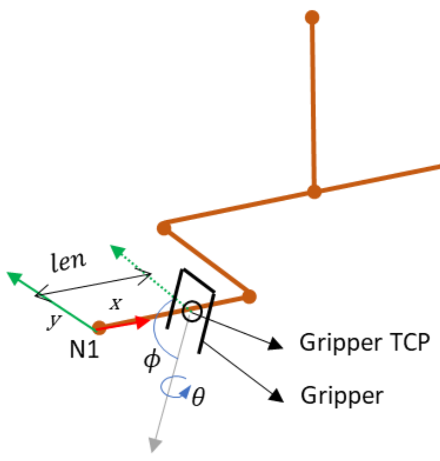


Figure 3.5 Variables that define a grasp.

3.2.3.2 Optimisation Constraints

3.2.3.2.1 Reachability Constraint (C1)

The first constraint that needs to be considered is the reachability of the robot. All the end-effector poses on the trajectory must be inside the workspace of the robot. Given the trajectory of the pipe and the generated local grasp pose $\mathbf{B}_{\mathrm{e}}^{\mathrm{P}}$, the robot end-effector pose in the world frame can be computed as follows:

$$\mathbf{T}_{\mathrm{e,i}} = \mathbf{A}_{\mathrm{p,i}} \mathbf{B}_{\mathrm{e}}^{\mathrm{P}}(\mathcal{g}).$$

Therefore, the robot joint trajectory can be computed by solving the inverse kinematics problem:

$$\boldsymbol{x}_{\mathrm{i}} = \mathrm{IK}(\mathbf{T}_{\mathrm{e,i}}), \qquad (3.1)$$

where $\boldsymbol{x}_i$ is a vector that describes the robot configuration. The dimension of $\boldsymbol{x}_{\mathrm{i}}$ is the degrees of freedom of the robot. If for $\mathrm{i} = 0 \ldots \mathrm{n}$, Eq. (3.1) is solvable and the solution satisfy the robot joint range constraint, then this grasp pose satisfies the reachability constraint.

3.2.3.2.2 Collision Constraint (C2)

The second constraint that needs to be satisfied is that the whole manipulation process must be collision-free. To satisfy this constraint, collision checking is performed for all the robot configurations $(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_\mathrm{n})$ and intermediate states during the manipulation process. When performing collision checking, a safe distance is implemented to ensure that no collision happens in case of any uncertainties e.g., geometric modelling errors, robot motion inaccuracy or part deformations. The collision checking is performed by an open source library FCL [106] which supports both collision detection and distance queries.

3.2.3.3 Optimisation Objectives

3.2.3.3.1 Joint Motion ($\mathbf{O1}$)

Objective $\mathrm{O1}$ is defined as the sum of squared displacements between two consecutive waypoints to encourage minimum robot joint motion. Since the robot configurations required to follow the given pipe trajectory and grasp pose have been calculated already using IK solver in the constraint checking process, the computation of $\mathrm{O1}$ is straightforward:

$$\mathrm{Cost(O1)} = \sum_{\mathrm{i}=1}^{\mathrm{n}} \|\boldsymbol{x}_\mathrm{i} - \boldsymbol{x}_{\mathrm{i}-1}\|_2,$$

where n is the number of waypoints of the trajectory as mentioned above.

3.2.3.3.2 Deformation of the pipe ($\mathbf{O2}$)

This objective intends to minimise the sum of maximum deformation that happens at each waypoint along the trajectory. In this way, the geometric shape of the pipe can be maintained during the manipulation process. The deformation depends on the grasp position with respect to the pipe frame as well as the orientation of the pipe with respect to the world frame. The pipe is modelled as a frame structure that consists of arbitrarily oriented beam members which are connected rigidly. The beam members support bending, shearing as well as axial loads. A custom FEA program is implemented by using the matrix method described in [107]. The key step here is to reconstruct the boundary condition when evaluating different grasp poses. After grasping, the pipe is assumed to be rigidly supported at the grasping location. The initial pipe structure is created before the optimisation process starts as in Section 3.2.2. Once the current grasp pose is determined, a new node is created at the grasp location and the original grasp section is broken into two sections. The matrix used to store the geometry of the pipe assembly is updated accordingly. After creating the new geometry, the stiffness matrix of the pipe can be constructed. The general stiffness matrix for a 1D pipe section in the local frame is given by:

$$\mathbf{K} = \begin{bmatrix}
\frac{EA}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{EA}{L} & 0 & 0 & 0 & 0 & 0 \\
 & \frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} & 0 & -\frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} \\
 & & \frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 & 0 & 0 & -\frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 \\
 & & & \frac{GJ}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{GJ}{L} & 0 & 0 \\
S & & & & \frac{4EI_y}{L} & 0 & 0 & 0 & \frac{6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & 0 \\
 & Y & & & & \frac{4EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{2EI_z}{L} \\
 & & M & & & & \frac{EA}{L} & 0 & 0 & 0 & 0 & 0 \\
 & & & M & & & & \frac{12EI_z}{L^3} & 0 & 0 & 0 & -\frac{6EI_z}{L^2} \\
 & & & & E & & & & \frac{12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & 0 \\
 & & & & & T & & & & \frac{GJ}{L} & 0 & 0 \\
 & & & & & & R & & & & \frac{4EI_y}{L} & 0 \\
 & & & & & & & Y & & & & \frac{4EI_z}{L}
\end{bmatrix}$$

where E and G are the Young's modulus and the shear modulus. I_y and I_z are the second moment of area. J is the torsional constant (same as I_x in the circular case) and L is the length of the pipe section. Since each node has 6 degrees of freedom, assuming the structure has N nodes, the stiffness matrix for the whole structure is a $6N$ by $6N$ matrix. Transform from local to the global coordinate system:

$$\mathbf{K}_{\text{global}} = \mathbf{R}^T\mathbf{K}\mathbf{R},$$

where $\mathbf{R}$ is the rotation matrix which represents how each pipe section is oriented from the global coordinate system. Details about constructing the stiffness matrix for a structure can be found in [107].

The weight of the pipe distributes evenly along its length as shown in Figure 3.6(a). Therefore, the loading conditions is equivalent to having one force and one moment acting on each node of the section as shown in the Figure 3.6(b). Assuming W is the weight per unit length, the equivalent loading can be obtained as follows:

$$\mathrm{F} = -\mathrm{WL}/2$$
$$\mathrm{M1} = \mathrm{WL}^2/12$$
$$\mathrm{M2} = -\mathrm{WL}^2/12$$

The force vector can then be constructed for each node by using the above equations. Once the force vector is obtained, the displacement vector $\boldsymbol{u}$ can be computed by:

$$\boldsymbol{f} = \mathbf{K}\boldsymbol{u}.$$

where both $\boldsymbol{f}$ and $\boldsymbol{u}$ are all column vectors of size $6N$.

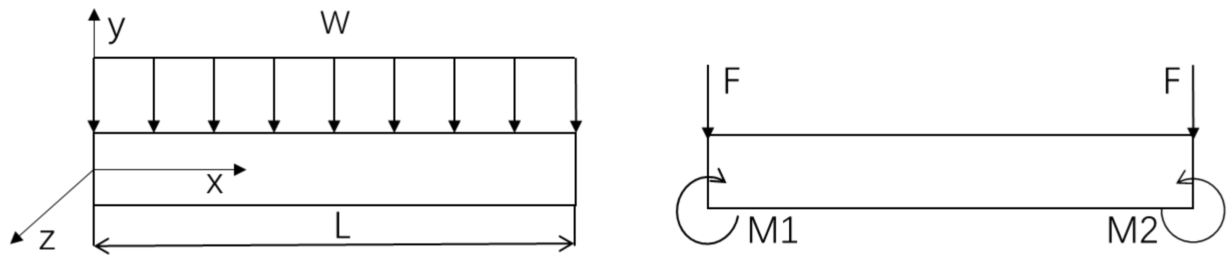


Figure 3.6 Loading condition of a pipe element. (a) Loading condition of pipe element under gravity. (b) Equivalent loading condition.

The maximum deformation $(\mathrm{maxDeform})$ of the pipe at one specific pose can then be obtained by computing the norm of displacement vector $\boldsymbol{u}$ for each node. The $\mathrm{Cost(O2)}$ is defined to be the sum of the maximum deformation at each pose along the trajectory as follows:

$$\mathrm{Cost(O2)} = \sum_{\mathrm{i}=1}^{n} \mathrm{maxDeform_i}$$

Note here the deformation of first trajectory waypoint ($i = 0$) is not computed since the pipe is still placed on a fixture at this pose.

3.2.3.3.3 Force Manipulability (O3)

The third objective is to minimise the force manipulability along the end-effector moving direction. According to the force/velocity duality, minimising the force manipulability is equivalent to maximising its velocity manipulability. Therefore, for a given set of joint velocities, the end-effector can move faster with a large velocity manipulability.[1] In order to compute the manipulability along a specific trajectory, the manipulability ellipsoids are constructed as follows:

$$\boldsymbol{v}^T \left(\boldsymbol{J_f}(\boldsymbol{x}).\boldsymbol{J_f}^T(\boldsymbol{x})\right)^{-1} \boldsymbol{v} = 1 \tag{3.2}$$

$$\boldsymbol{\gamma}^T \left(\boldsymbol{J_f}(\boldsymbol{x}).\boldsymbol{J_f}^T(\boldsymbol{x})\right) \boldsymbol{\gamma} = 1. \tag{3.3}$$

where $\boldsymbol{J_f}(\boldsymbol{x})$ is the Jacobian matrix of joint configuration $\boldsymbol{x}$. $\boldsymbol{v}$ is the velocity vector of the end-effector and $\boldsymbol{\gamma}$ is the force (torque) vector of the end-effector. Eq. (3.2) defines the velocity manipulability ellipsoid and Eq. (3.3) defines the force manipulability ellipsoid. Both ellipsoids are shown in Figure 3.7 for a simple 3-link planar robot manipulator.

[1] Although the intention is to maximise the velocity manipulability, the velocity manipulability is not directly used to avoid the scenario that 2 objectives need to be minimised while the other one needs to be maximised. This scenario often leads to negating the objective to be maximised. However, negative objective costs are not ideal since Dijkstra's Algorithm used later requires edge weights of the graph to be positive. Therefore, the force manipulability is used instead.

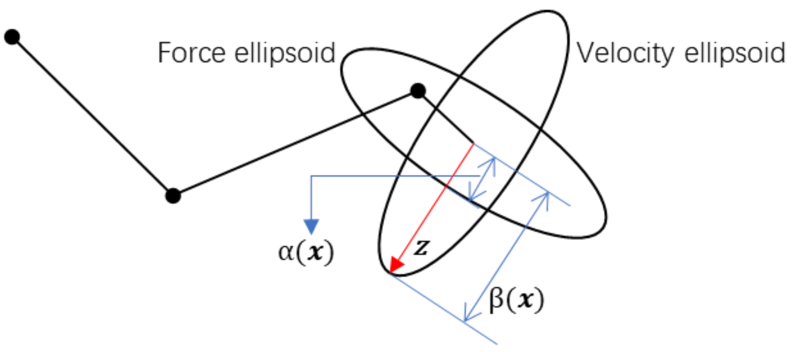


Figure 3.7 Force and velocity ellipsoid for a 3-link planar robot

Given a unit vector $\boldsymbol{z}$ represents the direction of movement of the end-effector, the velocity manipulability ($\beta(\boldsymbol{x})$) and the force manipulability ($\alpha(\boldsymbol{x})$) are defined to be the length of the vector from the centre of the ellipsoid along $\boldsymbol{z}$ to the surface of the respective ellipsoid. $\beta(\boldsymbol{x})$ and $\alpha(\boldsymbol{x})$ can be computed by rearranging Eq. (3.2) and (3.3) as follows:

$$\beta(\boldsymbol{x}) = (\boldsymbol{z}^{\boldsymbol{T}}(\boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x})\boldsymbol{J}_{\boldsymbol{f}}{}^{T}(\boldsymbol{x}))^{-\boldsymbol{1}}\boldsymbol{z})^{-1/2}$$

$$\alpha(\boldsymbol{x}) = (\boldsymbol{z}^{\boldsymbol{T}}\boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x})\boldsymbol{J}_{\boldsymbol{f}}{}^{T}(\boldsymbol{x})\boldsymbol{z})^{-1/2}.$$

As shown in Figure 3.7, a direction with small $\alpha(\boldsymbol{x})$ has a relatively large $\beta(\boldsymbol{x})$ which suggests that the end-effector is relatively easier to move along the given direction $\boldsymbol{z}$. The objective function is defined as follows:

$$\text{Cost}(O3) = \sum_{i=1}^{n} \alpha(\boldsymbol{x}_i).$$

Note here again i starts from 1 rather than 0, since $\boldsymbol{z}_\mathrm{i}$ is defined to be the vector when robot attempts to move from pose $\mathrm{i}-1$ to pose i. In this way, all three objectives are consistent in the sense that they are trajectory based objectives since there has to be at least 2 waypoints in the trajectory.

3.3 Objective and Constraint Evaluation

3.3.1 Constraint Handling Method

The constraints are handled by using the method reported in [108]. The method compares two solutions based on the following 3 criteria:

1. A feasible solution is always better than an infeasible solution.
2. Between two infeasible solutions, the one that violates the constraint less is considered to be better.
3. Between two feasible solutions, the one with a better objective cost is better.

The advantages of this method are twofold. Firstly, it does not require an explicit penalty parameter to handle the constraints. Besides, it allows objectives to be evaluated only when all the constraints are satisfied, which significantly reduces the algorithm running time especially in the case of having an expensive objective function.

The optimisation problem has two constraints. The collision constraint is checked by the FCL library [106]. To check if the reachability constraint is satisfied, a third party IK solver (IKfast [109]) is used. The returned result of the IK solver is binary, either successful or not, which means it is impossible to compare two infeasible grasps which one violates the constraint more. To solve this problem, a combined constraint cost is defined for each waypoint along the trajectory as follows:

$$\mathrm{Cost}(\mathrm{C_i}) = \begin{cases} 0, & \text{if both C1 and C2 are satisfied} \\ 1, & \text{if C1 is satisfied while C2 is not} \\ 2, & \text{if C1 is not satisfied} \end{cases}.$$

And the cost for the whole trajectory is defined as follows:

$$\mathrm{Cost}(\mathrm{C}) = \sum_{\mathrm{i}=0}^{\mathrm{n}} \mathrm{Cost}(\mathrm{C_i}).$$

It should be noted that the cost for the waypoint is set to 2 automatically when $\mathrm{C1}$ is not satisfied. That is because the joint configuration ($\boldsymbol{x}$), which is required for checking $\mathrm{C2}$, can only be obtained when $\mathrm{C1}$ is satisfied.

3.3.2 Weights Selection for Combining Multiple Objectives

A weighted sum approach is used to handle multiple objectives:

$$\mathrm{Cost}(\mathrm{O}) = \mathrm{w_1}\mathrm{Cost}(\mathrm{O1}) + \mathrm{w_2}\mathrm{Cost}(\mathrm{O2}) + \mathrm{w_3}\mathrm{Cost}(\mathrm{O3}).$$

To systematically determine the weights for each objective, an Analytical Hierarchy Process [110] is used. The approach determines the relative importance among objectives by a series of pairwise comparisons. The results of the comparison are used to generate a comparison matrix:

$$\begin{pmatrix} & O1 & O2 & O3 \\ O1 & 1 & 6 & 2 \\ O2 & 1/6 & 1 & 1/4 \\ O3 & 1/2 & 4 & 1 \end{pmatrix}.$$

The entry of the matrix tells the preference level between the 2 objectives. For example, the entry at $(O1,\ O2)$ is 6, which means $O1$ is much preferred to $O2$ (these numbers are subjective since they are selected based on the author's preferences and experiences). Then the consistency of the matrix is verified by computing the consistency ratio (CR) as follows:

$$\begin{cases} \mathrm{CI} = \dfrac{\lambda_{max} - \mathrm{m}}{\mathrm{m} - 1} \\ \mathrm{CR} = \dfrac{\mathrm{CI}}{\mathrm{RI}} \end{cases}$$

where CI is the consistency index of the comparison matrix, RI is the average random index (given in Table 3.1), CR is the random consistence ratio of the comparison matrix, λ_{max} is the maximal eigenvalue, and m is the order of the judgment matrix. If CR is less than 10%, the matrix is considered as having an acceptable consistency. In our case, the CR is 0.79%, which is acceptable.

Table 3.1 Random indices from [110].

m	3	4	5	6	7	8	9	10
RI	0.58	0.9	1.12	1.24	1.32	1.41	1.45	1.49

For a consistency matrix, the weights are computed by normalising each column of the matrix and then calculating the average of each row. The resultant weight vector is $\boldsymbol{w} = [\mathrm{w}_1\ \mathrm{w}_2\ \mathrm{w}_3]^T = [0.56\ 0.12\ 0.32]^T$.

3.3.3 Considering Multiple IK Solutions

Eq. (3.1) assumes only one IK solution is available given an end-effector pose. However, there could be at most 16 different joint configurations for a 6 DoF robot [111]. Analytical IK solvers like IKFast can compute multiple IK solutions efficiently. In this section, the method used to handle multiple IK solutions is introduced. As shown in Figure 3.8, each circle corresponds to a robot configuration (IK solution). All the circles in the same column have the same end-effector pose. A graph can then be constructed. The vertices of the graph are the robot configurations (blue circles in Figure 3.8). The edges are created by connecting each vertex in one column to all the vertices in the next column. The cost of each edge is the weighted sum of joint motion distance (O1) and manipulability (O3). Deformation objective (O2) is not used to compute the cost because the deformation only depends on the orientation of the pipe and the grasp pose. Different arm configurations producing the same end-effector pose will not have an impact on the deformation cost. In this way, evaluating the cost of a single grasp is converted to a graph search problem. The problem is solved by running Dijkstra's algorithm for each start vertex (i.e., vertices in the first column). This process is similar to the Descartes Planner in ROS industrial project [112].

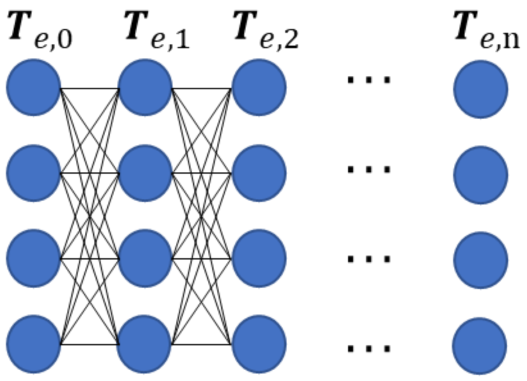


Figure 3.8 The inverse kinematics solution graph for a given object trajectory

3.3.4 Overall Evaluation Process

The complete procedure for evaluating objective and constraint cost is presented in Algorithm 3.1. The function starts by initialising both constraint cost and objective cost to be 0 and then for each waypoint on the trajectory of the pipe, the end-effector pose is computed. Given an end-effector pose, a subfunction MULTI-IK is called to get multiple IK solutions from the IK solver. If the size of $\mathbf{X}_{\mathrm{i}}$ is zero, the end-effector pose is determined to be not reachable (i.e.,

$C1$ is not satisfied), thus the constraint cost is incremented by 2. If there is at least one feasible IK solution, the function will perform collision checking for all IK solutions. If there is no collision-free IK solution in $\mathbf{X}_{\mathrm{i}}$ (i.e., $C2$ is not satisfied), the constraint cost is incremented by 1. After checking whether constraints are satisfied or not, the objective cost will be evaluated for the constraint-free grasp. A weighted graph is created first. Then, for each IK solution ($\mathbf{X}_0$[j]) in $\mathbf{X}_0$, Dijkstra's Algorithm is used to search for the shortest path from $\mathbf{X}_0$[j] to any IK solution in $\mathbf{X}_{\mathrm{n}}$, whose cost is assigned to $current\_cost$. $optimal\_cost$ tracks the cost of the best IK solutions found so far in $\mathbf{X}_0$. Then the cost for the grasp is set to be the sum of the optimal graph cost and the weighted deformation cost.

Algorithm 3.1 GRASP-EVALUATION-MULTI-IK(g)

```
1.  $\mathrm{Cost(C)} = 0, \mathrm{Cost(O)} = 0$
2.  for $\mathrm{i} = 0 : \mathrm{n}$,
3.     $\mathbf{T}_{\mathrm{e,i}} = \mathbf{A}_{\mathrm{p,i}}\mathbf{B}_{\mathrm{e}}^{\mathrm{P}}(g)$
4.     $X_{\mathrm{i}}$ = MULTI-IK($\mathbf{T}_{\mathrm{e,i}}$)  // $\mathbf{X}_{\mathrm{i}}$ is a matrix and each column is a valid IK solution
5.     if $\mathbf{X}_{\mathrm{i}}$.cols() == 0 // IK solver returns 0 solution
6.        $\mathrm{Cost(C)} += 2$
7.     else
8.        for $\mathrm{j} = 0 : \mathbf{X}_{\mathrm{i}}$.cols()
9.           if not COLLISION-FREE($\mathbf{X}_{\mathrm{i}}[j]$)
10.             DELETE $\mathbf{X}_{\mathrm{i}}$[j] from $\mathbf{X}_{\mathrm{i}}$  // delete solution in collision
11.       if $\mathbf{X}_{\mathrm{i}}$.cols() == 0  // no IK solution satisfies collision constraint
12.          $\mathrm{Cost(C)} += 1$
13. if $\mathrm{Cost(C)} == 0$
14.    $optimal\_cost$ = INFINITY
15.    $G$ = CONSTRUCT-GRAPH ($\mathbf{X}_0, \ldots, \mathbf{X}_{\mathrm{n}}$)    // construct graph
16.    for j = 0 : $\mathbf{X}_0$.cols() ,
17.       $current\_cost$ = DIJKSTRA-SEARCH ($G$, j)  // search graph given the start vertex
18.       if $current\_cost < optimal\_cost$
19.          $optimal\_cost = current\_cost$
```

```
20.    $\text{Cost}(O) = optimal\_cost + w_2\text{Cost}(O2)$
21.    $g.cost = \{0, \text{Cost}(O)\}$
22. **else**
23.    $g.cost = \{\text{Cost}(C), 0\}$
```

3.4 Optimisation Algorithm

3.4.1 Original Bees Algorithm

Bees algorithm (BA) [113], a population-based search algorithm which mimics the food foraging behaviour of honey bees, is used to solve the optimisation problem. The algorithm is shown in Algorithm 3.2 and the definitions of hyperparameters are given in Table 3.2

Algorithm 3.2 BA

```
1. Initialise $graspVec$
2. for each grasp $g$ in $graspVec$
3.     GRASP-EVALUATION-MULTI-IK($g$)
4. while stopping condition not true
5.     Locate elite and non-elite best sites
6.     Local search
7.     Site abandonment and neighbourhood shrinking
8.     Global Search
```

Table 3.2 BA parameters and definitions

Parameter	Definition
ns	Number of scout bees
ne	Number of elite sites
nb	Number of best sites
nre	Number of recruited bees for elite sites
nrb	Number of recruited bees for remaining best sites
stlim	Number of no improve iterations before site abandonment (stagnation limit)

The algorithm initialises $graspVec$ by creating a colony of ns scout bees randomly in the search space. Each point in the search space (also known as a site in the Bees Algorithm literature) corresponds to a solution grasp $\mathcal{g}$. After evaluating the cost of each site, all sites visited by scout bees are sorted and the best $\mathrm{nb} < \mathrm{ns}$ sites are selected for local search. Among nb best sites, the scout bees at top ne sites recruit nre bees to perform local search in the neighbourhood. The bees at the remaining $\mathrm{nb} - \mathrm{ne}$ best sites recruit nrb bees to perform local search ($\mathrm{nrb} < \mathrm{nre}$). If the result of local search does not improve and the site is searched again in the next iteration, the neighbourhood size is shrunk. The initial neighbourhood size is defined to be a proportion of the interval where the variable is defined. If the same site is searched for a predefined number of iterations (known as stagnation limit, stlim) without improving, the local minimum is considered to be reached and the scout bee at that site will perform random search again (site abandonment). After finishing local search, the remaining scout bees will be placed randomly in the search space to perform global search. Unlike the standard implementation of BA, the number of global searches in this work is set as $\mathrm{ns} - \mathrm{ne} \times \mathrm{nre} - (\mathrm{nb} - \mathrm{ne}) \times \mathrm{nrb}$ to keep the same number of function evaluations in the initialisation process and later iterations.

3.4.2 Local Search Strategy

The local search strategy of the standard bees algorithm is straightforward. Given a solution site, The neighbourhood of the site is defined as a hyperrectangle. Recruited bees are randomly placed in the hyperrectangle to generate new grasps. The size of the hyperrectangle is shrunk when the same site is searched multiple times without improving. However, the optimisation variables used in this chapter does not ensure newly generated grasps inside hyperrectangular are close to each other in terms of their Cartesian coordinate (as shown in Figure 3.9(a)). Since only variable S and len determine the Cartesian coordinates of a grasp, three different neighbourhood generation strategies for S and len are presented as follows:

- Str1: This strategy performs the default behaviour of the bees algorithm i.e., treating S

and len independently. For example, if the solution site is on S3 and the neighbourhood size for S is 4, the newly generated grasps can be on S1 – S5. If the solution site is close to one end of the section, it is likely that the newly generated grasp position len is out of the range. In this case, the len will be set as the limit (either 0 or the maximum length of the section).

- Str2: This strategy is similar to Str1 except that the neighbourhood size for S is always 0. This ensures the newly generated grasps are always close to the solution site being searched since they are constrained to be on the same section. However, this strategy is very conservative and may lead to early convergence.

- Str3 (proposed): Like Str2, Str3 does not allow the section to be changed in the usual condition. However, if the newly generated grasp position len is out of the range, unlike Str1 and Str2, Str3 will explicitly find if there is any other section connected to the section of the solution site and select randomly from all connected sections to locate the grasp. For example, the feasible range for len on S3 is [0, 150]. If the generated len is -10 on S3, the grasp will be located on S2 with len set as $\text{maxLength}(S2) - 10$. In this way, Str3 ensures the newly generated grasps are close to the solution site and also allow the change of section during local search to avoid early convergence.

The neighbourhoods of Str1, Str2 and Str3 are shown as the blue area in Figure 3.9(a), (b) and (c). The red star is the solution site.

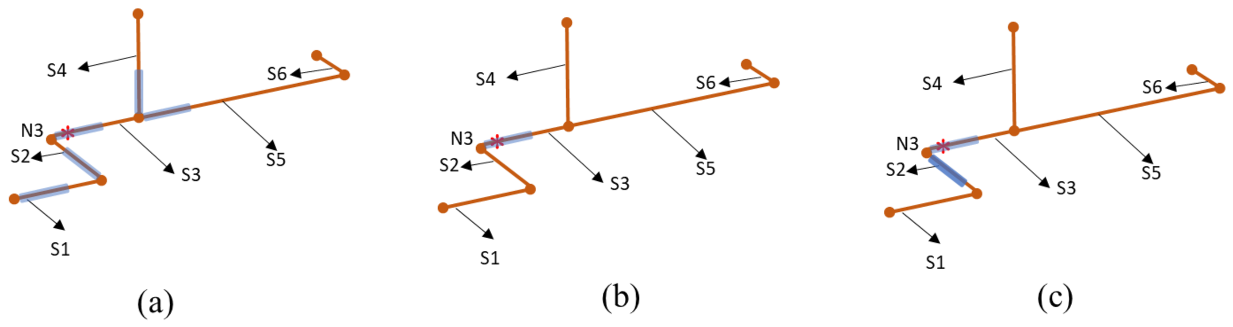


Figure 3.9 Search neighbourhood for Str1 (a), Str2 (b) and Str3 (c) respectively.

3.5 Experiment and Results

3.5.1 Experiment Setup

In the experiment, three different copper pipe assemblies are tested in a simulated environment using the proposed method. The geometries of Pipe1, Pipe2 and Pipe3 are shown in Figure 3.10. The dimension of each pipe section is given in Table 3.3. Other properties of the pipe are listed in Table 3.4. The robot used in the experiment is a standard 6-axis industrial robot with a spherical wrist (KUKA KR6 R900 sixx). The maximum reach of the robot is 900mm and the payload is 6kg. The program is run on a Linux machine with an Intel Core i7-4712MQ CPU @ 2.30 GHz and 8GB RAM. The IK solver and collision checking library are accessed through ROS MoveIt/Descartes API.

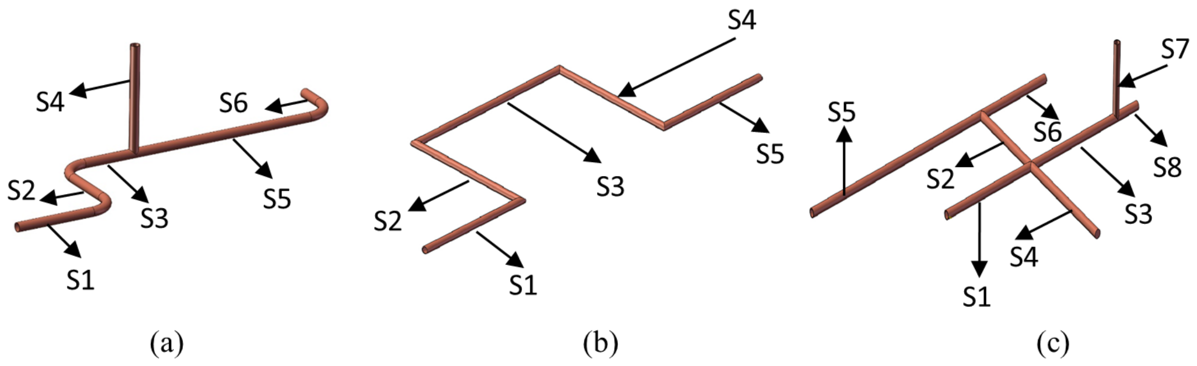


Figure 3.10 The CAD model for Pipe2 (a) and Pipe3 (b).

Table 3.3 Pipe assembly dimensions.

Section (mm)	S1	S2	S3	S4	S5	S6	S7	S8
Pipe1	200	200	150	200	400	100	\	\
Pipe2	200	200	300	200	200	\	\	\
Pipe3	200	150	200	200	400	150	100	50

Table 3.4 Pipe properties.

Outer diameter (mm)	Inner diameter (mm)	Density (kg/m^3)	Young's modulus (GPa)	Shear modulus (GPa)
12	10	8960	110	40

To avoid the potential impact of hyperparameters on the performance of the algorithm, each strategy is tested with 9 different sets of hyperparameters (listed in Table 3.5) for 50 times. In the experiments, three sets of hyperparameters are tested in parallel to accelerate the computation. The percentage 10%, 30% and 45% under the "Random Scout" column indicate the approximate proportion of global search in population. The optimal solution is determined to be found if the objective cost is within $\pm 0.1\%$ range of the known optimal cost. The optimisation process stops if the optimal solution is found, or the maximum iteration is reached. For all 9 sets of hyperparameters, the maximum number of grasp evaluations are the same ($\mathrm{ns} \times \mathrm{iteration} = 1530$).

Table 3.5 Tested 9 sets of hyperparameters for Bees Algorithm.

No.	ne	nre	nb	nrb	Random Scout	ns	Iteration
1	1	12	7	3	4 (10%)	34	45
2	1	15	7	5	6 (10%)	51	30
3	1	20	15	5	12 (10%)	102	15
4	1	12	5	3	10 (30%)	34	45
5	1	15	6	4	16 (30%)	51	30
6	1	20	11	5	32 (30%)	102	15
7	1	10	4	3	15 (45%)	34	45
8	1	12	5	4	23 (45%)	51	30
9	1	15	9	5	47 (45%)	102	15

Genetic Algorithm (GA) [114] and Particle Swarm Optimisation (PSO) [115], two widely used population-based metaheuristics, are also implemented to solve the proposed problem for comparison. To ensure a relatively unbiased comparison, 9 sets of hyperparameters are tested for both GA and PSO and the stopping condition is the same as BA. For GA, 9 sets of hyperparameters are generated from 3 population sizes (34, 51 and 102) and 3 mutation rates (0.2, 0.3 and 0.4). Binary tournament selection is used in the parents selection process of GA. The solutions are real value encoded. Single point crossover is used with crossover rate 1. The mutation operator for the binary variable θ is simply bit-flip. For S, len and ϕ, the mutation operator samples uniformly within a given mutation range. For S, the mutation range is $[\mathrm{S}_1, \mathrm{S}_{\max}]$, where $\mathrm{S}_{\max}$ is the maximum section number of the current pipe. For len and ϕ, if the current solution value is a, the mutation range is $[\mathrm{a} - 100, \mathrm{a} + 100]$, while satisfying the usual lower and upper limits of the variable.

For PSO, 9 sets of hyperparameters are generated from 3 population sizes (34, 51 and 102) and

3 connectivity levels (10%, 50%, and 100%). For example, if the population size is 34, a 10% connectivity level means each particle is connected to 3 closest particles (fractional part is truncated). Inertia weight is set to 0.7 and both acceleration coefficients (c1 and c2) are set to 2. The velocity update is not implemented for the grasp section S and the flip angle θ since they are intrinsically discrete (binary). These two variables are updated to be the same as their personal best, global best or keep their original value with probability 0.3, 0.4 and 0.3, respectively.

An additional experiment is performed to compare the performance of the multi IK evaluation method (Algorithm 3.1) and the single IK counterpart (implemented by removing the graph search step from Algorithm 3.1). Both methods use the same set of hyperparameters (No. 7 in Table 3.5, except the number of iterations). The same 3 pipes in Table 3.3 are tested in this experiment. However, the stopping condition is different since it is difficult to determine the optimal solution for the single IK method. Theoretically, the optimal solution should be the same as the multi IK method. However, in practice, it almost never finds the same optimal solution since 1). the single IK method uses a numerical IK solver which usually only finds the solution closer to the initially provided solution; 2). There are too many possible arm motions for a predefined workspace object trajectory (for example, imagine there are 10 waypoints, and each waypoint has 2-4 possible IK solutions, the probability for the IK solver to generate the exact optimal combination is between $1/2^{10}$ and $1/4^{10}$). Therefore, the stopping condition for both methods is set to be the completion of 30 search iterations (i.e., 1020 grasp evaluations).

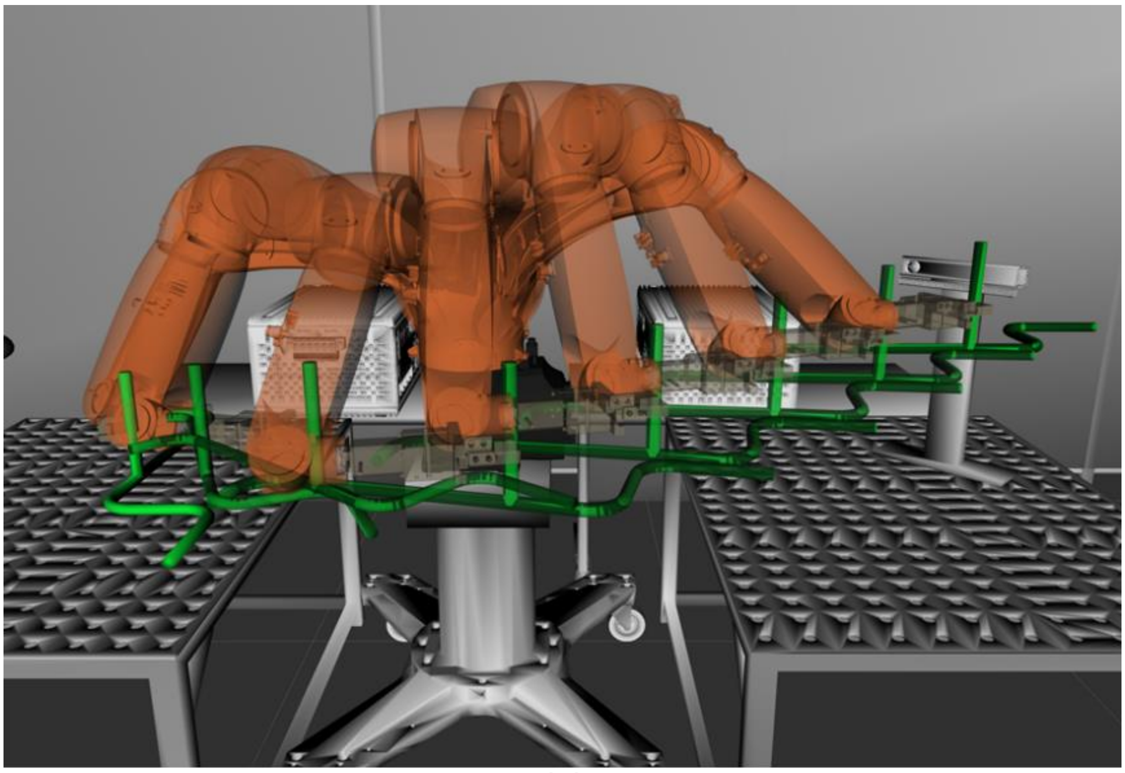
3.5.2 Results and Discussion

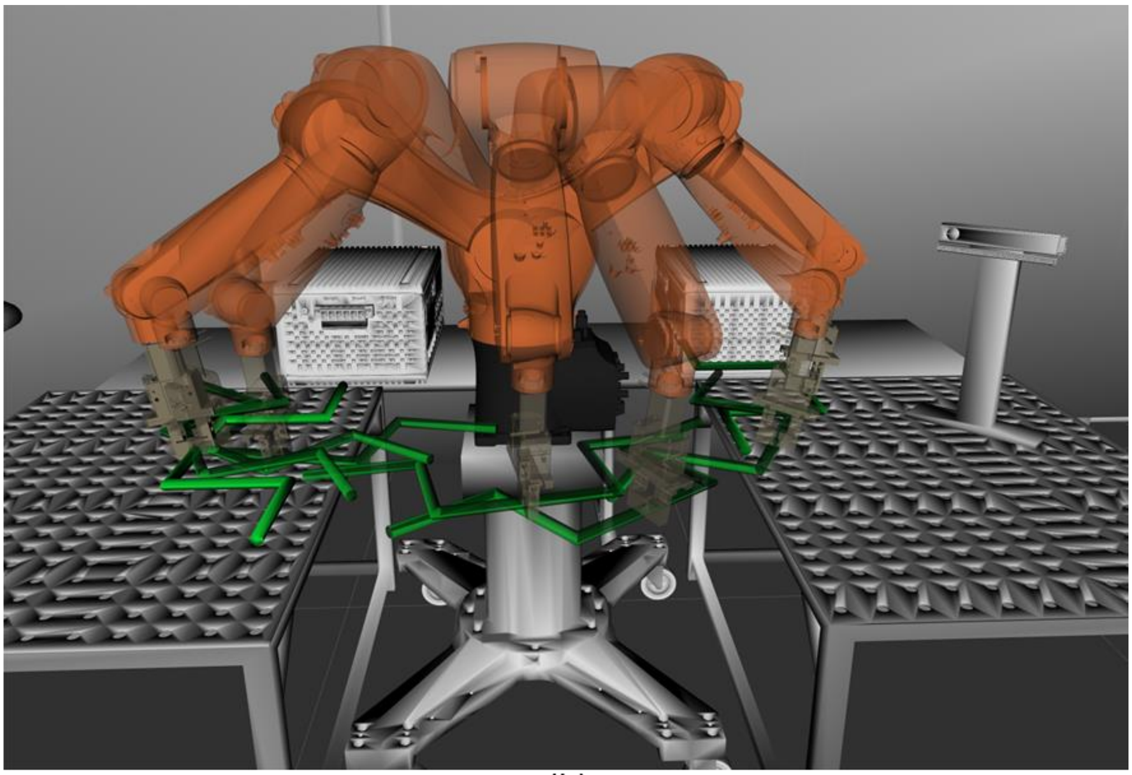
3.5.2.1 Optimal Grasp and Robot Trajectory

The optimal grasp parameters found for each pipe to complete the given pipe trajectory are listed in Table 3.6. The trajectory of the robot for manipulating Pipe1-3 is shown in Figure 3.11. It can be found in Table 3.6 that the flip angle θ can choose either 0 or 180 for all 3 pipes. This is because the gripper that used in this chapter is symmetric and mounted to be aligned with the rotation axis of the last joint of the robot. Therefore, θ can be chosen as either 0 or 180 without affecting the following motion of the robot.

Table 3.6 Optimal solutions found for each pipe

	S	len	ϕ	θ	Objective Cost
Pipe1	S4	79	-155	180/0	4.7638
Pipe2	S3	151	-100	180/0	4.1487
Pipe3	S2	96/97	23	180/0	5.7905


(a)


(b)

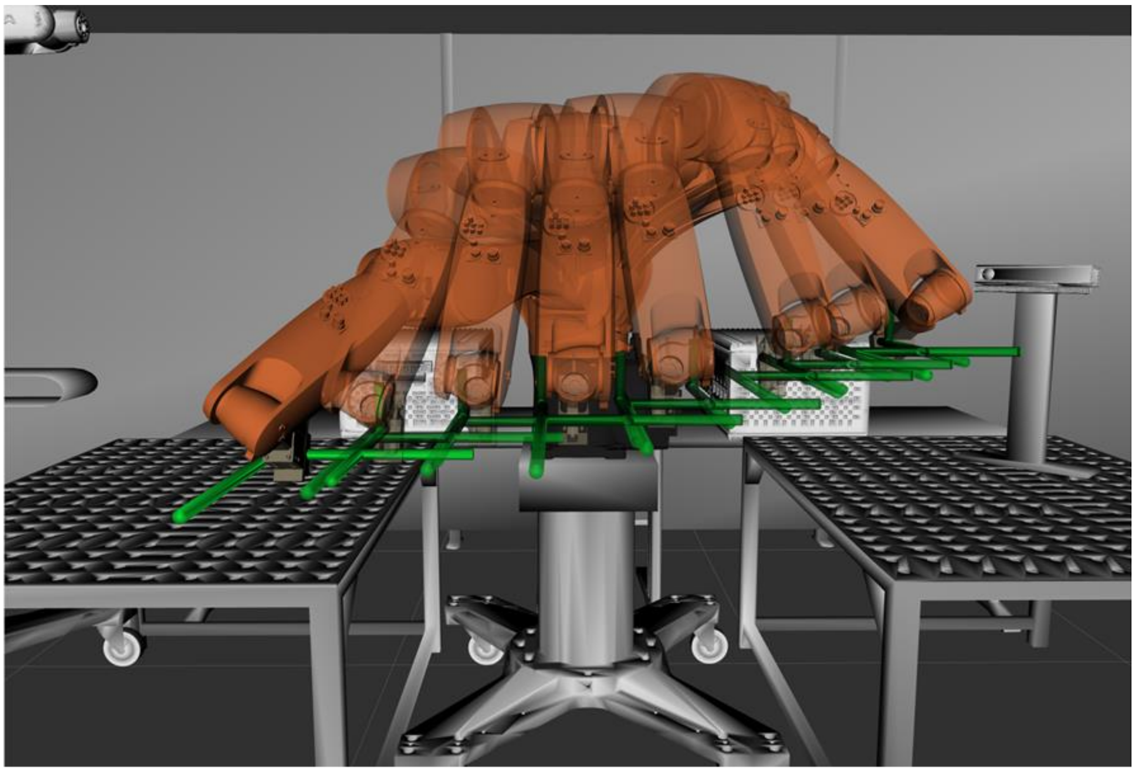

(c)

Figure 3.11 The robot trajectories when manipulating Pipe1 (a), Pipe2 (b) and Pipe3 (c).

3.5.2.2 Comparison among different algorithms

For each algorithm (strategy), the best optimisation results for solving 3 problems are combined and listed in Table 3.7. It is worth noting that the best set of hyperparameters is different for different problems. Therefore, each column in Table 3.7 is not for a single set of hyperparameters but the combination of best results from different sets of hyperparameters on different problems. The results show that by using an appropriate set of hyperparameters, all 3 BA local search strategies can achieve a 100% success rate out of 50 tests in finding the optimal solution. However, the computation speed of each strategy is different. The row named "Grasp Evaluations" lists the number of grasps evaluated for solving 3 problems by each strategy. It is clear that Str2 and Str3 require significantly fewer grasp evaluations than Str1, which suggests that Str2 and Str3 should converge much faster than Str1. However, in terms of the actual running time shown in the next row, Str1 is a lot faster than both Str2 and Str3. The inconsistency is due to the constraint handling method used in this work (see Section 3.3.1). Since Str1 is intrinsically more stochastic and does not focus on a single section during the local search, the search efficiency of Str1 is lower than Str2 and Str3, thus it requires generating a large number of grasps to find the optimal solution. However, the grasp evaluation process is not required to be fully performed if the generated grasp is in constraint. Therefore, although Str1 generates more grasps to be evaluated, most of them can be evaluated within a short time. On the other hand, Str2 and Str3 mainly generate grasps that are close to the feasible grasps, which makes the generated grasps more likely to be feasible and need to be fully evaluated. Since evaluating the deformation objective $O2$ is relatively a time-consuming operation, Str1 has the advantage in terms of time consumption even though it generates more grasps. The results of GA and PSO are listed in the last two columns of Table 3.7. By using the algorithm setup in Section 3.5.1, the GA and PSO generally perform not as good as BA as they cannot achieve a 100% success rate on all three problems and the time required to finish the optimisation is longer.

Table 3.7 Optimisation results of the best performed set of hyperparameters for each strategy

	Str1(BA)	Str2(BA)	Str3(BA)	GA	PSO
Grasp Evaluations	1100	845	832	2172	1205
Time (s)	31.4	40.1	37.5	76.49	49.7
Success Rate	100%	100%	100%	93.3%	98%

The first three columns of Table 3.8 show the average results of 3 strategies over 9 sets of hyperparameters. Still, Str1 evaluates more grasps with less time than both Str1 and Str2. In terms of overall success rate, Str3 achieves the highest success rate. Table 3.9 lists how many times each local search strategy fails to find the optimal solution on the individual problem out of 450 tests (9 configs × 50 tests/config). It can be found that Str1 is likely to fail on Pipe2 and Str2 is likely to fail on Pipe3, while Str3 performs more consistently over different problems. The inconsistency of the algorithm performance is due to the choice of hyperparameters. More analyses regarding the impact of hyperparameters on the performance of the algorithm are presented in the next section. The average results of GA and PSO over all sets of hyperparameters are listed in the last two columns of Table 3.8. It can be found that, by using the algorithm setup in Section 3.5.1, GA and PSO are more sensitive to the selection of hyperparameters than BA as the success rate drops significantly compared to the best results in Table 3.7. The last 2 columns of Table 3.9 also show that the implemented GA and PSO version in this work perform worse than BA on all tested problems except Pipe2, where the result of PSO is comparable to BA.

Based on the above analyses, generally, it is recommended to use Str3 for consistently good performance over different problems. However, if the computation time is extremely critical, Str1 can be considered as well.

Table 3.8 Average optimisation results over 9 sets of hyperparameters for each strategy

	Str1(BA)	Str2(BA)	Str3(BA)	GA	PSO
Grasp Evaluations	1572	1192	1168	2699	1891
Time (s)	41	57.4	51.4	92.4	93.2
Success rate	98.9%	98.5%	99.6%	76.8%	81.3%

Table 3.9 number of times that the algorithm fails to find global optimal

	Str1(BA)	Str2(BA)	Str3(BA)	GA	PSO
Pipe1	0	0	2	46	154
Pipe2	9	2	2	76	7
Pipe3	5	18	1	191	91
Total	14	20	5	313	252

3.5.2.3 The impact of the hyperparameters on the performance of the algorithm

The impact of hyperparameters (i.e., population size and the proportion of global search) on the performance of BA is analysed in this section.

3.5.2.3.1 The size of scout bees

As shown in Table 3.10, by using a large size of population, all 3 strategies take longer to converge in terms of both the number of function evaluations and actual running time. In terms of success rate, Str1 with population size 102 only achieves a 97.7% success rate, which is obviously worse than others. Generally speaking, a large size of population means fewer iterations can be run given a finite number of grasp evaluations, thus the algorithm cannot update current best solutions timely and reallocate computation resources efficiently, which leads to longer running time and less success rate. However, if the strategy lacks stochastics, using a large population size may have some advantages. As in the case of Str2, using 102 achieves a 98.7% success rate, slightly higher than both 34 and 51. This is because a large population size ensures that initially the scout bees can cover the search space as much as possible and avoid early convergence.

Table 3.10 Optimisation results using different population size

	Str1			Str2			Str3		
Population size	34	51	102	34	51	102	34	51	102
Grasp Evaluations	1179	1453	2085	976	1137	1462	975	1090	1441
Time (s)	35.0	39.5	48.4	52.4	57.5	62.3	48.9	49.8	55.8
Success Rate (%)	99.5	99.5	97.7	98.4	98.4	98.7	99.5	99.8	99.5

3.5.2.3.2 The proportion of global search

As shown in Table 3.11, with the increasing of global search proportion (random scout), the numbers of grasp evaluations for Str2 and 3 almost stay at the same level, while Str1 increases steadily. This is because Str1 already has a lot of stochastics, continuing to increase the proportion of global search only makes the algorithm need more evaluations to converge. In terms of actual running time, Str1 is not impacted by the increase of global search proportion, while Str2 and Str3 run faster. This is due to the same reason why Str1 is faster than Str2 and Str3 as explained in Section 3.5.2.2. In terms of success rate, Str2 with 10% global search performs significantly worse than others. This is because Str2 does not allow any change of section during local search and relies heavily on global search to jump out of local minimum. It is also interesting to see that the success rates of all 3 strategies go up when the global search proportion increase from 10% to 30%, and then drops when the proportion keeps increasing. This trend indicates keeping adding more global searches to the algorithm will probably have a negative impact on the success rate.

Table 3.11 Optimisation results using different proportion of global search

	Str1			Str2			Str3		
Random Scout	10%	30%	45%	10%	30%	45%	10%	30%	45%
Grasp Evaluations	1503	1546	1668	1180	1208	1186	1212	1112	1178
Time (s)	41.2	40.3	41.6	67.4	56.8	48.0	61.5	47.4	45.3
Success Rate (%)	98.4	99.3	99.1	96.8	99.6	99.1	99.3	100	99.6

3.5.2.4 Comparison between multi IK and single IK method

The final result presented is the comparison between the multi IK grasp evaluation method and the single IK counterpart. The single IK method has the advantage that it does not need an analytical solver to generate multiple IK solutions, which is preferable if the robot system is not standard and does not have available analytical solutions. Although the details of the single IK grasp evaluation method are not presented, it should be easy to implement by slightly modifying the multi IK method.

As shown in Figure 3.12, for all 3 pipes, the multi IK method achieves significantly lower average objective cost (19.6%, 11.3% and 8.4% respectively). Besides, the standard errors are also smaller, which means the results are more consistent. In terms of the running time, multi IK requires more time to finish for solving Pipe1 and Pipe2 than the single IK. This is predictable as multi IK has an additional graph search process. However, it is interesting to see that single IK requires more time for solving Pipe3 than multi IK. A possible explanation is that the solution single IK converges to may have many feasible solutions in the neighbourhood, which requires a large amount of time to fully evaluate them, while the neighbourhood of the true optimal that multi IK converges to has less feasible solutions, therefore the method skips the objective function evaluation process which results in faster convergence. This result suggests that although multi IK requires an additional graph search process when evaluating a single grasp pose, it is not necessarily slower than the single IK method. Therefore, the multi IK method is preferable whenever a suitable IK solver is available.

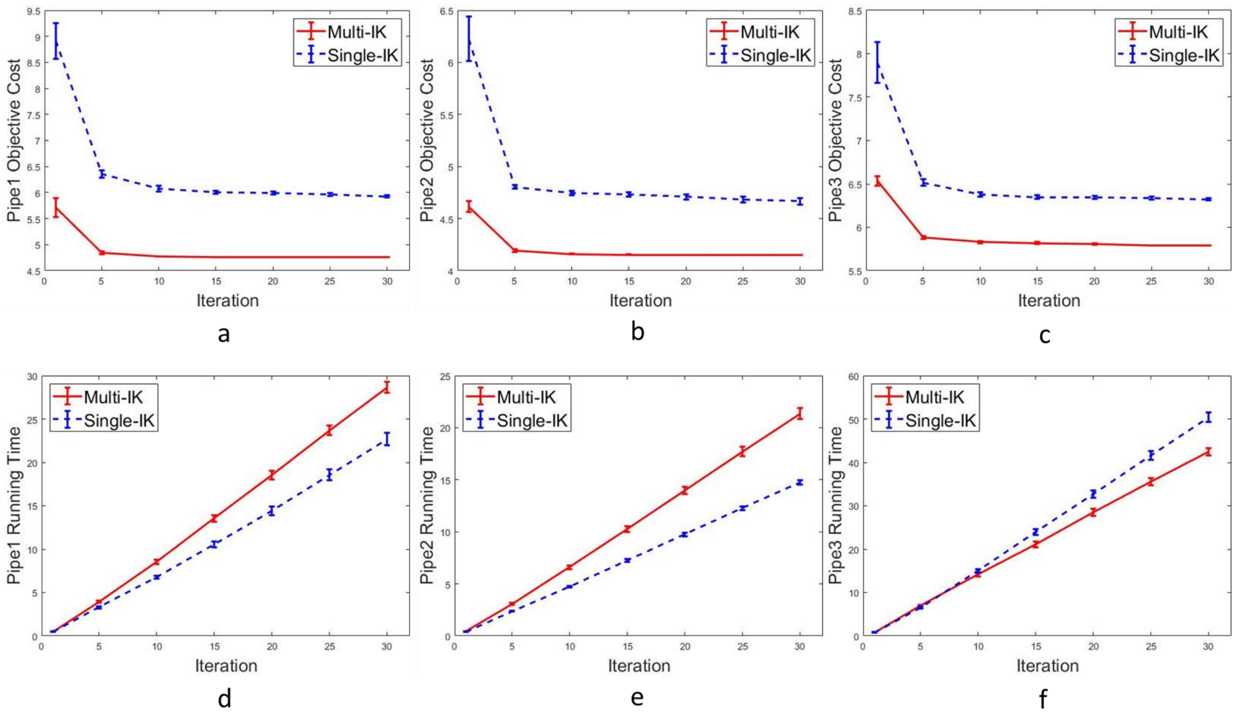


Figure 3.12 Performance comparison between the multi IK and the single IK method. (a) – (c) Obtained objective costs at each iteration for solving Pipe1 – 3 with multi IK and single IK method for 30 iterations. (d) – (e) The time consumption at each iteration for solving Pipe1 – 3 with multi IK and single IK method for 30 iterations

3.6 Conclusion

In this chapter, a methodology is developed to optimise the grasp pose for 3D pipe assembly manipulation. The method can effectively optimise the grasp pose based on three trajectory-based objectives (joint motion, deformation of the object and force manipulability), while satisfying reachability and collision constraint. The grasp evaluation process features a decoupled constraint handling method to reduce grasp evaluation time, an AHP method to select the weights for combining multiple objectives, and a Dijkstra's Algorithm to find optimal trajectory among all possible IK solutions. Bees Algorithms is used to solve the constrained optimisation problem with a proposed problem-specific local search strategy. Extensive benchmarks have been performed to evaluate the performance of 3 local search strategies and 2 other metaheuristics (GA and PSO). It is found that BA with proposed Str3 is less sensitive to the hyperparameters and can achieve consistently good performance on different problems. Besides, the comparison between multi IK and single IK method proves that by considering multiple IK solutions, the objective cost can be improved significantly indeed.

The method is intended for manufacturing pipe assemblies in the Factory-In-A Box (FIAB) scenario to address the limitation of specially designed end-effector, the compact environment of FIAB and the flexibility of pipe structure. However, it can also be generalised to manipulating other compliant objects that have many grasp candidates with alternative deformation estimation methods.

Planning for a given object trajectory is useful in many cases. However, many other manipulation tasks only care about the start and final location of the object regardless of what trajectory it takes to get to the goal. In this case, free space motion is required to be planned. Next chapter will show methods that simultaneously select a grasp from a feasible grasp set as well as plan its post-grasp free space motion.

Chapter 4 Integrated Grasp Selection and Post-grasp Optimal Robot Motion Planning

4.1 Introduction

Manipulation planning problem concerns finding a collision-free path for a robot to transfer an object from the initial pose to the goal pose. To generate a successful grasp for manipulating the object, it is necessary to consider the problem of post-grasp motion planning in the meantime, since a good grasp in terms of force closure and other grasp quality measures may lead to an inefficient or even infeasible post-grasp motion. This problem is especially significant in the case of manipulating relatively large and complex structures. These objects typically have many feasible grasps and choosing different grasps results in very different geometries. One example of this type of object is the 3D pipe assembly shown in Figure 4.1. The green pipe is manipulated by a robot to the desired pose while avoiding obstacles in the environment.

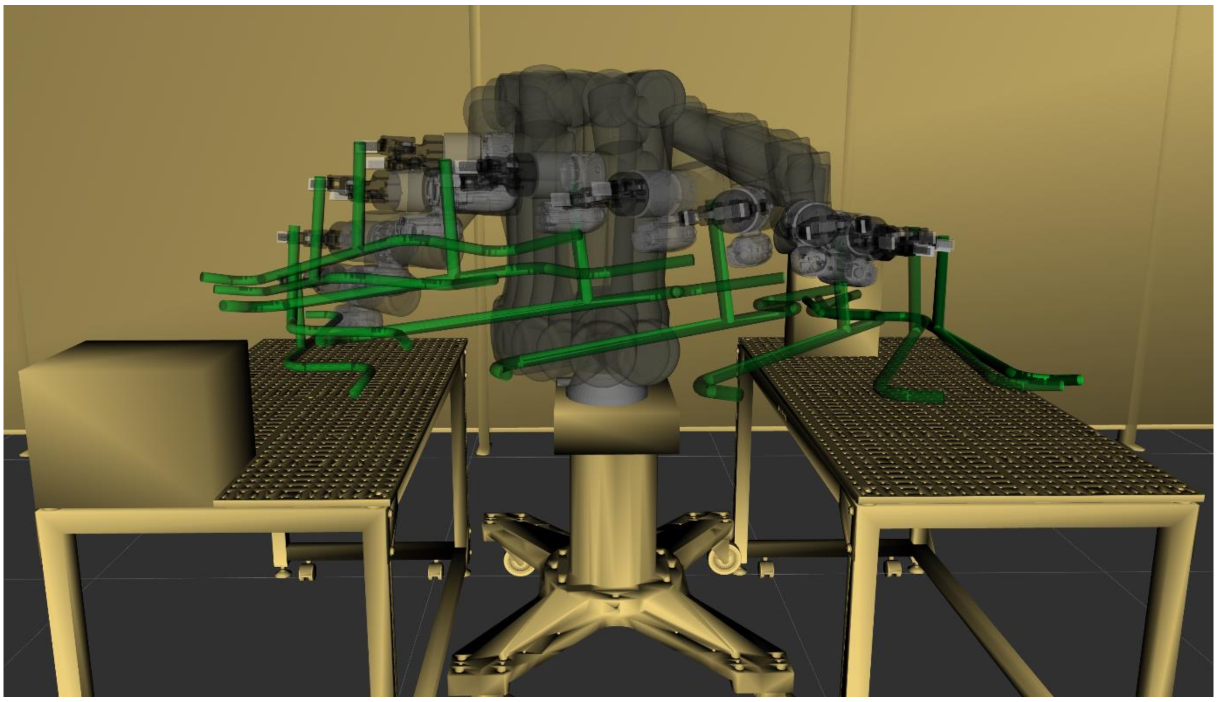


Figure 4.1 A 3D pipe assembly manipulated by an industrial robot

To successfully plan the motion for robot, a grasp planner is assumed to be available to generate a set of feasible grasp candidates for the gripper. Generated grasps should consist of a 6-DoF end-effector pose and a d-dimensional hand/gripper configuration as mentioned in Section 2.2.1. The method presented in this chapter does not limit the choice of grasp planners, however, planners that only produce contact points/regions on the object need additional computation to consider specific hand model to generate a fully feasible grasp.

After generating feasible grasps, manipulation planning methods usually rely on motion planning algorithms to generate continuous motion in the free space. Sampling-based planners (reviewed in Section 2.1.1) like Rapidly-exploring Random Trees (RRT) [10] and Probabilistic RoadMap (PRM) [9] are very effective for solving high dimensional motion planning problems such as planning motion for a robot manipulator. Although they can find feasible path for robot to execute, it has been shown in [12] that the planner will almost surely converge to a suboptimal solution. They also proposed RRT* and PRM* which guarantee finding an asymptotically optimal solution.

This chapter presents PRM*-MG (Optimal PRM for Multiple Grasps). PRM*-MG is based on the PRM algorithm [9] and also incorporates ideas from its optimal and lazy variants (PRM* and Lazy PRM [12,13]). Since the integrated problem involves multiple grasps, it is natural to reformulate the integrated problem to a multi-query motion planning problem and use PRM to solve it. However, to solve the problem correctly and efficiently, two major issues need to be considered. First, the feasibility of the roadmap is different when processing different grasps. Second, processing each grasp sequentially is very slow as the number of grasps grows. To handle the first issue, the proposed algorithm adopts a lazy search strategy, and a data structure is designed to track the feasibility of edges and vertices of the graph (roadmap) for each grasp. The second issue is dealt with by batch processing the grasps. Inside each batch, the collision checking is only performed for the optimal path after the whole batch is searched, which introduces another level of laziness into the algorithm. After searching one batch, if a feasible solution exists, the cost of the solution will be used as an upper bound to prune the roadmap in subsequent searches. PRM*-MG includes two special versions. In the first version, the batch size equals 1, while in the second version, the batch size equals the size of the whole feasible grasp set (in other words, there is only one batch).

The main contribution of this chapter is the development of a manipulation planner that can fast process a large number of grasps to find the one with the optimal post-grasp motion cost. The problem studied in this chapter is similar to [96] but can deal with more general cases. Specifically, in [96], the grasp optimisation is restricted to one degree of freedom, since it needs to maintain the same contacts as the generated grasp, while the method developed in this chapter can deal with a larger set of feasible grasps with different contact points. This chapter also does not restrict the gripper type, while in [96] the method can only apply to suction cups and standard two-finger grippers. However, the method proposed in [96] minimizes the execution time directly and considers the smoothness of trajectory while the proposed method in this work generates an asymptotically optimal path but requires additional post-processing to obtain a smooth trajectory.

The rest of the chapter is organized as follows. The problem is defined formally in Section 4.2. The main algorithm and relative functions are described in Section 4.3. The theoretical properties of the algorithm are analyzed in Section 4.4. The experiments and results are presented in Section 4.5 and Section 4.6 concludes the chapter.

4.2 Problem Formulation

Given an object $\mathcal{O}$, the start and goal pose of $\mathcal{O}$ are denoted as $\mathrm{P}_{OStart} \in SE(3)$ and $\mathrm{P}_{OGoal} \in SE(3)$. Let $\mathcal{G} = \{g_{i=1\ldots m}\}$ be a set of feasible grasps generated by a grasp planner. Let $X \subseteq \mathbb{R}^{\mathrm{d}}$ be the set of all possible robot configurations, $X_{obs} \subset X$ be the set of configurations that are in collision with obstacles, and $X_{free} = X \setminus X_{obs}$ be the set of all collision-free configurations. A feasible path of the robot is defined as $\sigma: [0, 1] \rightarrow X_{free}$ which is a continuous map to a sequence of configurations through collision-free space. Let Σ be the set of all such feasible paths. Given a cost function, $c : (\Sigma, \mathcal{G}) \rightarrow \mathbb{R}_{\geq 0}$, the problem is to find the optimal grasp g^* from $\mathcal{G}$ and the optimal path σ^* from Σ such that $c(\sigma^*, g^*) = min\{c(\sigma, g) \mid \mathrm{FK}(\sigma(0), g) = \mathrm{P}_{OStart}, \mathrm{FK}(\sigma(1), g) = \mathrm{P}_{OGoal}\}$, where FK is the forward kinematics function that maps the given robot configuration and the grasp to the object pose.

4.3 Algorithm

This section starts by giving an overview of the PRM*-MG algorithm in Section 4.3.1. Then, some basic functions and a data structure used in PRM*-MG are introduced in Section 4.3.2. Section 4.3.3 describes the PRM*-MG algorithm in detail. Two special cases of the algorithm are presented in Section 4.3.4. Section 4.3.5 discusses some practical considerations.

4.3.1 Overview of the Algorithm

Given the problem formulated in Section 4.2, a straightforward approach is to run lazy-PRM for each grasp. The lazy strategy delays the collision checking step until a solution is found for a grasp. Therefore, different grasps can use the same graph (with different start and goal states) for initial search and attach the object only when performing collision checking. Since the formulated problem only requires finding the optimal path, it is unnecessary to evaluate the feasibility of the path once the path is known to be non-optimal. Therefore, a simple but powerful improvement can be applied by using the optimal cost from previously searched grasps as an upper bound for pruning during the searching process of the next grasp. This version of the algorithm is referred to as naïve PRM*-MG.

However, naïve PRM*-MG still fully evaluates many grasps before the optimal grasp is found. To further reduce the unnecessary collision checking, it may be helpful to only check collision for the current optimal solution. This idea motivates the development of batch PRM*-MG algorithm, which further delays the collision checking until all grasps in the grasp set are searched once and only performs collision checking for the solution with the optimal cost. This is the key insight of this chapter. By delaying collision checking until all grasps are searched once, the approach essentially applies the lazy strategy at the grasp level rather than robot configuration level as originally introduced in [13].

Since graph searching must be performed on all grasps before generating the first solution, the anytime performance of batch PRM*-MG is not ideal, especially when searching over a large feasible grasp set and/or a large graph. Therefore, it is natural to consider combining the above two ideas to design an algorithm that balances the anytime performance and the total run time. This consideration leads to the standard version of the PRM*-MG. The algorithm first divides the grasp set into several small batches. Then, for each small batch, the batch PRM*-MG is used to search for the optimal solution in that batch. Similar to the naïve version, the optimal cost from previously searched batch is used as an upper bound for pruning in graph search of the next batch. This version of PRM*-MG also includes the naïve and batch versions as special cases.

4.3.2 Function and Data Structure

4.3.2.1 GraphConstruction

Similar to PRM and PRM*, PRM*-MG has a graph construction process and a query process. The graph construction process is shown in Algorithm 4.1. V and E are the vertex set and the edge set of graph G respectively. Function Near queries all vertices within $\mathrm{r(n)}$ distance of vertex v and stores them in U. As presented in the algorithm, the graph will be constructed in the same way as PRM* with the only exception that the no collision checking is performed during the graph construction due to the use of lazy strategy. To guarantee the asymptotic optimality of the algorithm, it is necessary to use a connection radius $r(n) = \gamma_{\mathrm{PRM}}(\log(\mathrm{n})/\mathrm{n})^{1/\mathrm{d}}$, where $\gamma_{\mathrm{PRM}} > \gamma_{\mathrm{PRM}^*} = 2(1+1/\mathrm{d})^{1/\mathrm{d}}(\mu(X_{\mathrm{free}})/\zeta_{\mathrm{d}})^{1/\mathrm{d}}$, n is the number of vertices, d is the dimension of the configuration space, $\mu(\cdot)$ is the Lebesgue measure of a set (i.e., the volume), and ζ_{d} is the volume of the unit ball in d-dimensional Euclidean space [12]. An example of the roadmap constructed by Algorithm 4.1 is shown in Figure 4.2.

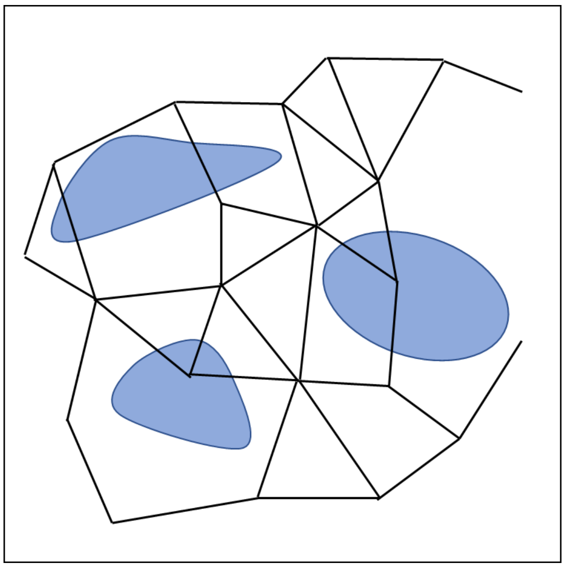


Figure 4.2 An example of roadmap constructed by Algorithm 4.1 in 2D. Obstacles are shown in blue. Because no collision checking is performed, it can contain edges and nodes in collision.

Algorithm 4.1: GraphConstruction()

```
1. $V \leftarrow \{\mathrm{Sample_i}\}_{\mathrm{i=1,\ldots,n}}$; $E \leftarrow \emptyset$
2. **foreach** $v \in V$ do
3.   $U \leftarrow \mathrm{Near}(G = (V,E), v, \mathrm{r(n)}) \setminus \{v\}$;
4.   **foreach** $u \in U$ **do**
5.     $E \leftarrow E \cup \{(v,u),(u,v)\}$
6.   **endfor**
7. **endfor**
8. **return** $G = (V,E)$
```

It is worth noting that the lazy strategy must be used because when the robot grasps the object at different poses, the feasibility of the graph is different. Without lazy evaluation of the graph feasibility, if there are m grasp candidates given, the algorithm ends up performing full collision checking for m graphs, which takes an extremely long time. Even if this process can be performed offline, storing the results would still take a lot of resources. Section 4.3.5 will discuss the necessary steps for generating a graph offline and using it online.

4.3.2.2 PrivateGraph (Data Structure)

PRM*-MG needs to deal with multiple grasps and each grasp corresponds to a graph with different feasibility. Therefore, it seems inevitable to copy the constructed graph multiple times and even store multiple copies of the constructed graph (in the case of batch PRM*-MG). When the graph is large and there are many grasp candidates, this process becomes extremely slow, and the memory size of the computer becomes inadequate. Therefore, instead of copying and storing the whole graph for each individual grasp. A new data structure *privateGraph* (pG) is created to efficiently store the minimum information relevant to a specific grasp and its corresponding graph (e.g., grasp, robot start and goal configurations, edges connected to start and goal vertex, edges in collision, current optimal path, and cost).

By using pG, only one complete graph is generated using Algorithm 4.1 and stays in the memory until the program finishes. Each time before processing (including graph searching and collision checking) the graph for a grasp candidate, the graph is modified based on information stored in pG by calling $\mathrm{ResetStartAndGoalStates()}$, $\mathrm{AddStartAndGoalEdges()}$, and $\mathrm{RemoveEdgesInCollision()}$. After processing the graph, the graph is restored by calling $\mathrm{RemoveStartAndGoalEdges()}$ and $\mathrm{RecoverEdgesInCollision()}$ so that all vertices and edges are kept the same as the originally generated graph. By using pG, the copy operation reduces to adding and removing edges, which can be executed efficiently (depending on specific graph implementation) and the memory problem is relieved as well.

4.3.2.3 ComputeStartAndGoal

This function generates start and goal configurations of the robot by calling third party inverse kinematics (IK) solver for all grasps in the given $\mathrm{pG\_set}$ (set of all *privateGraph*). Depending on the IK solver and the workspace of the robot, the number of valid start and goal configurations may be greater or less than the size of grasp set due to 1). no IK solution for the given grasp pose; 2). multiple IK solutions for the same grasp pose. However, the implementation of this chapter only uses a single IK solution, thus the number of valid grasps is always less than the number of the grasps given. If the IK computation is successful, the function will then search for vertices in G (generated by Algorithm 4.1) within a given ball centred at x_{init} and x_{goal} with radius $\mathrm{r(n)}$. Vertices inside the ball will be recorded by pG. If either IK computation or vertex searching procedure fails for either start or goal vertex. This grasp is determined as infeasible given the current graph and will be removed from pG_set.

4.3.2.4 CollisionFree

This function checks the feasibility for a given path connected between the start and goal configurations. When performing collision checking, the edges in collide are removed from G but tracked in pG for future recovery of the graph. If the collision happens at the vertex and the collision is due to the robot rather than the grasped object, the algorithm simply removes all its connected edges from G without tracking, this leads to an isolated vertex and will never be visited again during graph search. However, isolated vertices are not removed because they do not affect the correctness or performance of the algorithm. Besides, removing a vertex and can be a rather expensive process for some graph implementation.

4.3.2.5 PathConstruction

A path from the start to the goal vertex can be found immediately without searching the graph and the cost of the path can be used as an upper bound for shortest path search. This is achieved by using previous search results. Assume v_0 is the start vertex and $v_{\mathrm{n+1}}$ is the goal vertex. Two grasps are available in the grasp set with start configurations $v_0(1)$ and $v_0(2)$ and goal configurations $v_{\mathrm{n+1}}(1)$ and $v_{\mathrm{n+1}}(2)$. Assume a path from $v_0(1)$ to $v_{\mathrm{n+1}}(1)$ passing v_{i} and v_{j} is found in previous iterations and $v_0(2)$ and $v_{\mathrm{n+1}}(2)$ can also be connected with v_{i} and v_{j}. Since the path from v_{i} to v_{j} is known already, it is easy to construct a path from $v_0(2)$ to $v_{\mathrm{n+1}}(2)$. It is worth noting that v_{i} and v_{j} can be the same vertex and they don't have to be connected with $v_0(1)$ and $v_{\mathrm{n+1}}(1)$ directly.

4.3.2.6 GraphSearch

This function searches for the shortest path from start (s) to goal (t) using Dijkstra's algorithm (Algorithm 4.2). Unlike the original Dijkstra's algorithm, an upper bound is used in the searching process. Before relaxing the neighbour k of a vertex u, if the distance to k is still infinity, the function will compute the sum of distance to k after relaxing and the straight line distance from k to goal vertex t. This is an optimistically estimated cost of a path passing k to the goal vertex. If the sum is greater than the upper bound, it means that vertex k will not be on the optimal path of the problem, therefore the algorithm continues with the next neighbour without relaxing k. The reason that this pruning process only needs to be done when the distance to k is infinity is that, after k is relaxed once, future relaxation will only decrease the distance to k monotonically. Therefore, if the relaxation condition is satisfied (Line 17), the estimated cost will surely be less than the upper bound.

Algorithm 4.2: GraphSearch($G, upper\_bound,$ pG)

```
1.  foreach $v \in V$ do
2.      v.$dist \leftarrow inf$
3.      v.$prev \leftarrow nil$
4.  endfor
5.  s.$dist \leftarrow 0$
6.  $Q \leftarrow V$
7.  while (Q $\neq \emptyset$)
8.      u $\leftarrow$ ExtractMin($Q$)
9.      if u == t
10.         if u.$dist \neq inf$; return TRUE
11.         else if u.$dist == inf$; return FALSE
12.     endif
13.     foreach k $\in$ $neighbour(G,$ u) do
14.         if k.$dist == inf$ and u.$dist + Cost$(u, k) + $||$k, t$||_2 > upper\_bound$
15.                 continue
16.         endif
17.         if u.$dist + Cost$(u, k) $<$ k.$dist$
18.             k.$dist \leftarrow$ u.$dist + Cost$(u, k)
19.             k.$prev \leftarrow$ u
20.         endif
21.     endfor
22. endwhile
```

4.3.3 Detailed Algorithm

The detailed algorithm is given in Algorithm 4.3. The algorithm starts by initialising the current optimal cost to infinity. The pG_set is also initialised given the candidate grasp set. Then a graph is constructed by using Algorithm 4.1. The algorithm then computes the start and goal configurations of the robot for each given grasp and find vertices in the graph that are close to the start and goal configurations of the robot. Next, the pG_set is randomly and equally divided into several small batches and a batch_set is created consisting of all batches.

The algorithm then processes each batch sequentially. For each pG in a batch, the algorithm starts by setting/resetting the start and goal configurations and adding connected edges. An upper bound is obtained by comparing the current optimal cost ($optimal\_cost$) and the cost of newly constructed path (new_path_cost). GraphSearch then computes the optimal path connecting the start and goal configurations and stores the path in pG. pG is then inserted into a min heap Q (ordered by path cost). Since an upper bound is used, only paths with optimal cost less than the upper bound will be found by GraphSearch and inserted into Q. Before processing the next pG, edges connected with start and goal vertices are removed.

After searching all grasps in a batch, if the heap is not empty, the pG with the lowest path cost will be extracted from the heap and checked. Before collision checking, it is important to attach the object to the robot with the correct grasp pose (Line 21) . If the path is collision-free, the loop is terminated, and the optimal cost is reset. Since the upper bound is used for graph searching, the cost of the feasible path is guaranteed to be less than the current optimal cost. If collision checking is failed, the graph is searched again with edges in collision removed, and feasible results will be inserted back to Q (with new cost). Whether collision checking is successful or not, the graph needs to be recovered (Line 25) because edges in collision for one grasp may be feasible for another grasp. The while loop terminates when a collision-free path is found, or the heap is empty. The algorithm then continues to search the next batch until all batches are searched.

It should be noted that on Line 28 the upper bound is set to $optimal\_cost$ rather than $upper\_bound$ as on Line 11. This is because the path construction step on Line 10 may construct a path in collision but its cost (new_path_cost) is less than $optimal\_cost$ (i.e., $upper\_bound$ is equal to new_path_cost in this case). Consider the constructed path happens to be the optimal path found by GraphSearch on Line 11. Since edges in collision are removed during collision checking (Line 22), if $upper\_bound$ is still used in the GraphSearch on Line 28, it is impossible to find a path with a cost less than or equal to the $upper\_bound$, and feasible path with a cost between $upper\_bound$ and $optimal\_cost$ will be ignored.

Algorithm 4.3: PRM*-MG

```
1.  optimal_cost ← ∞ ; pG_set ← $\{pG(g_i)\}_{i = 1......m}$
2.  $G$ ← GraphConstruction()
3.  ComputeStartAndGoal(G, pG_set)
4.  batch_set ← BatchCreation(pG_set)
5.  **foreach** batch ∈ batch_set
6.      **foreach** pG ∈ batch
7.          ResetStartAndGoalConfig()
8.          AddStartAndGoalEdges()
9.          $new\_path\_cost$ ← PathConstruction(pG, $Q$)
10.         $upper\_bound$ ← **min** ($new\_path\_cost$, $optimal\_cost$)
11.         **if** GraphSearch(G, $upper\_bound$, pG)
12.             **insert** ($Q$, pG)
13.         **endif**
14.         RemoveStartAndGoalEdges()
15.     **endfor**
16.     **while** ($Q \neq \emptyset$)
17.         pG ← ExtractMin(Q)
18.         ResetStartAndGoalConfig()
19.         AddStartAndGoalEdges()
20.         RemoveEdgesInCollision()
21.         ReAttachObject()
```

```
22.         if CollisionFree($G$, pG)
23.             $optimal\_cost \leftarrow$ pG.current_optimal
24.             RemoveStartAndGoalEdges()
25.             RecoverEdgesInCollision()
26.             break    //continue with next batch
27.         else
28.             if GraphSearch(G, $optimal\_cost$, pG)
29.                 insert ($Q$, pG)
30.             endif
31.             RemoveStartAndGoalEdges()
32.             RecoverEdgesInCollision()
33.         endif
34.     endwhile
35. endfor
```

4.3.4 Two special cases of PRM*-MG (Naïve and Batch)

The algorithm presented in Algorithm 4.3 is the standard PRM*-MG which includes two special cases i.e., batch PRM*-MG and naïve PRM*-MG. The batch version of the algorithm only creates one single batch. Therefore, the outer loop in Algorithm 4.3 (Line 5-35) only needs to be executed once.

The naïve version of the algorithm treats each grasp (pG) as a single batch. Therefore, maintaining a min heap Q is unnecessary as it has at most one element. A simplified and more efficient implementation of the naïve PRM*-MG is given in Algorithm 4.4.

The naïve PRM*-MG does not offer better performance compared to the other algorithms in most cases. However, it is conceptually important since it can be viewed as a modified Lazy PRM for solving the proposed problem. It also provides a good baseline performance for evaluating the performance of the other two versions. Besides, the use of the upper bound is also proposed when evaluating the performance of the naïve PRM*-MG, which reduces the run time significantly.

Algorithm 4.4: naïve PRM*-MG

```
1.  $optimal\_cost \leftarrow \infty$ ; $\mathrm{pG\_set} \leftarrow \{\mathrm{pG}(g_{\mathrm{i}})\}_{i\ =1......m}$
2.  $G \leftarrow$ GraphConstruction()
3.  ComputeStartAndGoal($G$, pG_set)
4.  **foreach** pG ∈ pG_set
5.      ResetStartAndGoalConfig()
6.      AddStartAndGoalEdges()
7.      **while** GraphSearch(G, $optimal\_cost$ , pG)
8.          ReAttachObject()
9.          **if** CollisionFree($G$, pG. collision_edges, pG. path)
10.             $optimal\_cost \leftarrow$ pG. current_optimal
11.             **break**
12.         **endif**
13.     **endwhile**
14.     RemoveStartAndGoalEdges()
15.     RecoverEdgesInCollision()
16. **endfor**
```

4.3.5 Practical Considerations

4.3.5.1 Offline Graph Construction

The graph construction process can be performed offline. The robot can then answer multiple manipulation queries online using the same graph. When generating a graph offline, it is also useful to perform collision checking to the graph. However, the collision checking is only performed for the robot itself without considering the attached object, i.e., the generated graph can be used to plan robot motion for manipulating any object. Therefore, the graph can still contain vertices and edges in collision due to the attached object, thus collision checking still needs to be performed for the found optimal path during online manipulation query (in contrast to a multi-query motion planning problem, where online query generates feasible path directly). However, all robot configurations in the graph are valid, thus CollisionFree does not need to check the feasibility of the robot.

4.3.5.2 Pre-processing grasps

PRM*-MG assumes a set of feasible grasps is available based on the geometry of an object. As mentioned above, the grasp contains not only a 6-DoF grasp pose but also the configuration of the robot hand. Theoretically, the same grasp pose can have different grasps, especially in the case of dexterous hands, which possibly leads to different solutions. However, in most cases, the workspace of a hand can be enclosed by a bounding box (sphere). Therefore, it is usually redundant to generate different grasps with the same grasp pose as they will have same planning results. Another consideration is that in Algorithm 4.3, all grasps in the given grasp set are pre-processed to find IK results and connected vertices in the graph (Line 3 ComputeStartAndGoal). This process may be expensive depending on the range search algorithm, thus delaying the time to find the first feasible result. To achieve better anytime performance, it is possible to solve IK and find connected vertices when processing each pG in the batch (before Line 7 of Algorithm 4.3, ResetStartAndGoalConfig()).

4.4 Theoretical Properties

Since the proposed algorithm considers all feasible grasp candidates, given a graph (probabilistic roadmap), if there exists a grasp in the given grasp set that can be chosen to manipulate the object optimally from start to goal pose by searching the given roadmap, then this grasp will be found eventually. According to [12], when using a connection radius $r(n)$ with $\gamma_{PRM} > \gamma_{PRM^*}$, the constructed probabilistic roadmap will almost surely contain an optimal path as the number of vertices in the graph goes to infinity. Therefore, it becomes obvious that PRM*-MG inherits the probabilistic completeness and asymptotic optimality from PRM*.

It should be noted that in [12], the assumption is the configuration space is d-dimensional

Euclidean unit hypercube $(0,1)^{\mathrm{d}}$ and the cost metric is Euclidean length. However, the results are still applicable to the targeted system in this thesis i.e., manipulators with rotational degrees of freedom, which is locally Euclidean as discussed in [116,117]. When using a cost function other than Euclidean length, the optimality of the algorithm may not be preserved. It has been mentioned in [118] that the asymptotic optimality still holds in the case of line integral cost function under a slightly modified condition. However, from a practical point of view, it is more convenient to tune the parameter γ_{PRM} directly so that vertices in the graph are connected with a reasonable number of edges.

4.5 Experiment and Results

In this section, the physical experiment and some benchmark results are presented. The objects being manipulated in the experiment are 3D pipe structures (as shown in Figure 4.3). Each pipe is given a set of feasible grasps (the grasp generation process is trivial for cylindrical shaped pipes). The robot needs to manipulate the pipe from the given start pose to the desired goal pose while avoiding obstacles. Figure 4.4 shows a collaborative robot manipulating a 3D pipe assembly in the lab environment, whose motion is generated by the proposed algorithm.

4.5.1 Experiment Setup

Two experiments are performed. The robot used in both experiments is a 6-axis collaborative robot with a non-spherical wrist (Techman TM14-1100). The maximum reach of the robot is 1100mm and the payload is 14kg. The aim of the first experiment is to compare the performance of 3 versions of PRM*-MG when given different numbers of grasps (from 25 to 2000). In this experiment, 8 problems are tested, each with different pipe geometries (as shown in Figure 4.3) and different start and goal poses. For standard PRM*-MG, the batch size is set to be one-fifth of the total grasps. In the second experiment, the behaviour of the standard PRM*-MG using different batch sizes (from 1 to 2000) is studied when the total number of grasps given is 2000. As mentioned before, when the batch size is 1, the standard PRM*-MG reduces to the naïve version. When the batch size is equal to the size of a given grasp set, the algorithm becomes the batch PRM*-MG. Both experiments are run 10 times for each problem. Each time the grasps and graph are regenerated.

In both experiments, the generated graph contains 5000 samples and γ_{PRM} is set to 4. The objective for both experiments is to minimize the path length in the manipulator configuration space. To accelerate the shortest path search (Algorithm 4.2), A* with Euclidean distance heuristic is used instead of Dijkstra's Algorithm. The naïve version can be viewed as a baseline as mentioned in Section 4.3.4. However, since naïve PRM*-MG uses a newly designed data structure and pruning strategy, it is still several magnitudes faster than running existing motion planning algorithms for each available grasp sequentially. Because the graph construction and IK computation process are exactly the same for all three algorithms, they have only been performed once. By using the same graph and grasp set, the optimal grasp and its cost found by all three algorithms should be identical. The time spent on the graph construction and IK computation is not included in the results, since the graph can be constructed offline and IK computation is usually fast when using an analytical solver. Third-party packages are used for computing IK solutions (IKfast) [109], collision checking (FCL) [106] and range search (FLANN) .

To avoid too much time spent on solving a single problem during the benchmark, the time for searching a single grasp (in the case of naïve PRM*-MG) or a single batch (in the case of standard PRM*-MG) is limited to 5 seconds. In the case of batch PRM*-MG, the time limit is set to 15 seconds for executing the loop from Line 16 to Line 34 in Algorithm 4.3. The time limitation leads to an incomplete version of the original algorithm, which means a solution may not be found even there exists one for the current graph and grasp set, or only find a suboptimal solution. However, these situations are not included in the benchmark results since they rarely happen in practice.

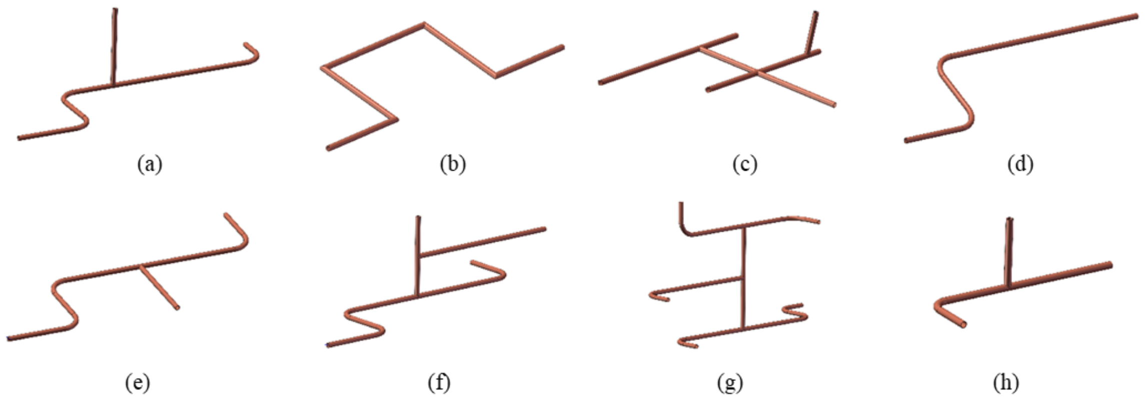


Figure 4.3 Eight different 3D pipe assemblies manipulated by Techman Robot

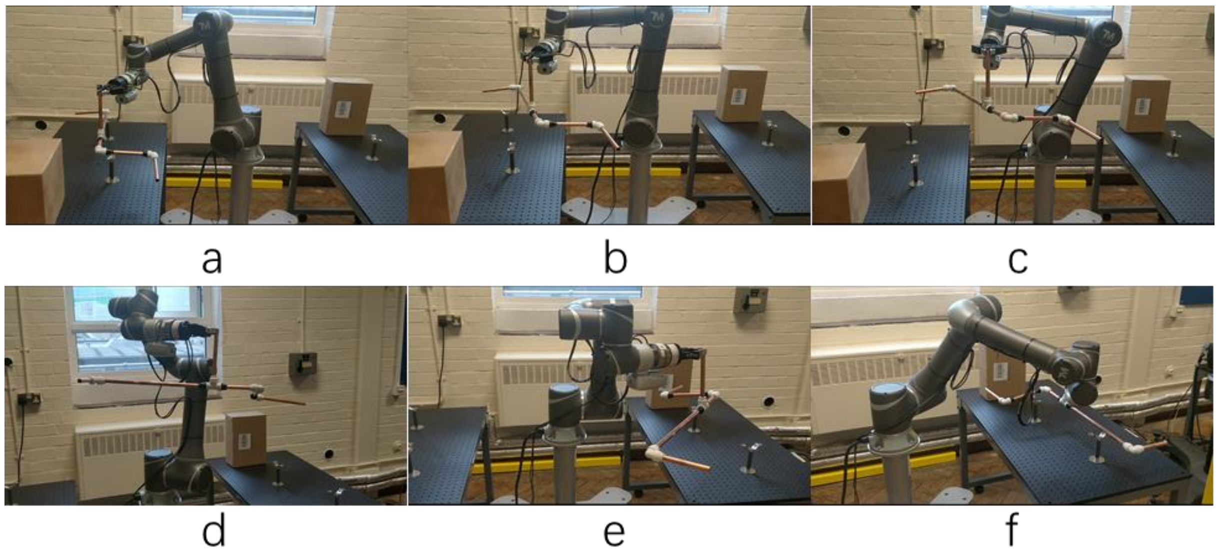


Figure 4.4 A pipe manipulated by Techman Robot from start (a) to goal pose (f)

4.5.2 Results and Discussion

The success rates for solving the same 8 problems given different numbers of grasps are listed Table 4.1. Generally, a larger grasp set will have a higher success rate in solving the given problems. Since all three versions of PRM*-MG use the same graph and grasp set, they will have the same success rate as well. Therefore, in the rest of this section infeasible solutions are not considered when evaluating the performance of the algorithms.

The results of the first experiment are shown in Figure 4.5. Figure 4.5 (a) shows the average time to the initial solution given different numbers of grasps. It can be found that for naïve PRM*-MG, the time required for obtaining the initial solution generally does not increase much as the number of grasps increases. For batch PRM*-MG, since it needs to perform more graph searches to generate the initial solution as the given number of grasps increases, the required time to find the initial solution also grows. In terms of standard PRM*-MG, the time to the first solution first decreases and then increases. This is because the given number of grasps is too small, which leads to an even smaller batch size. Therefore, the algorithm fails to find the feasible solution after evaluating the first few batches but spends unnecessary time on collision checking for infeasible solution paths. Figure 4.5 (b) shows the total computation time. The planning time for all 3 algorithms increases as the number of grasps increases. However, the run time of standard PRM*-MG almost remains the same when the given number of grasps is less than 1000. Batch PRM*-MG also grows very slow when the number of grasps is less than 100 and performs best among the three algorithms. However, beyond 100, since no upper bound is used in the batch version, the planning time grows faster than standard PRM*-MG. When the number of grasps reaches 2000, standard and batch PRM*-MG perform almost the same. It can be predicted that, given a larger grasp set, the performance advantage of batch PRM*-MG when dealing small grasp set will vanish.

Table 4.1 Success rate given different number of grasps

Number of grasps	25	50	100	500	1000	2000
Success rate	0.85	0.9125	0.9625	1	1	1

Figure 4.5 (c) shows the algorithm run time to the optimal solution. The trend is very similar to Figure 4.5 (b). However, the performance gap between batch and standard PRM*-MG vanishes from 500 grasps. When the number of grasps reaches 2000, standard PRM*-MG finds optimal solution faster than the batch version. It should be noted that, for batch PRM*-MG, the time to initial solution, time to optimal solution, and total run time are all the same, since the first solution is always the optimal solution. Figure 4.5 (d) shows the initial path length for batch and standard PRM*-MG and the found optimal path length for all three algorithms. The initial solution found by the naïve PRM*-MG remains at the same level as the number of grasps increases. However, the standard PRM*-MG finds a better initial solution as the number of grasps increases, because it evaluates more grasps before reporting the first result. It can also be found that even with 2000 grasps to evaluate, the standard PRM*-MG can still find an initial solution in around 1 seconds and the cost is very close to the optimal cost.

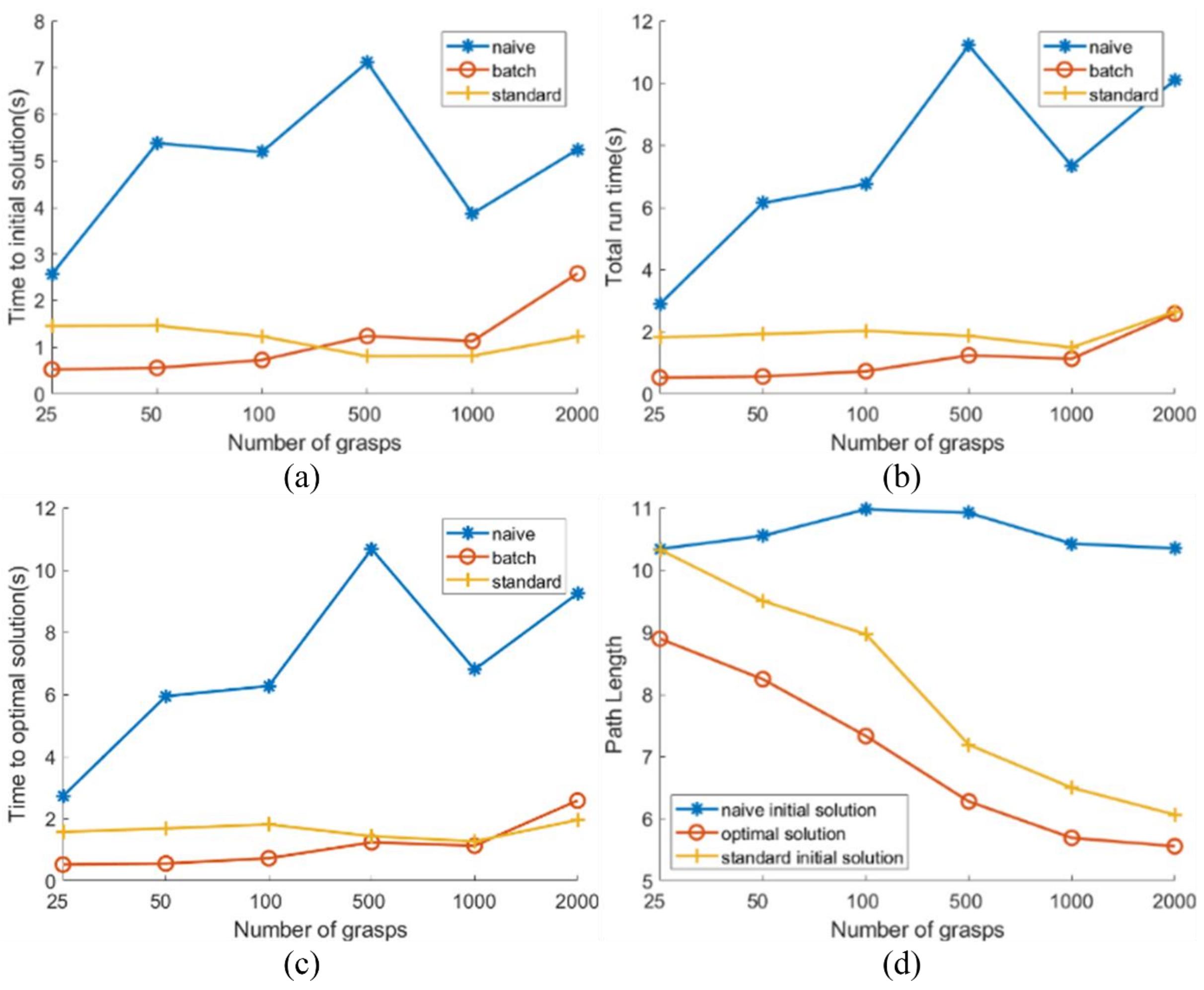


Figure 4.5 Performance comparison between three versions of PRM*-MG. (a) Average time to initial solution. (b) Average total computation time. (c) Average time to optimal solution. (d) Average solution cost.

Table 4.2 shows the performance of each algorithm when solving different problems averaged over different numbers of grasps. The results of the best performed algorithm are in bold. It is obvious that batch PRM*-MG offers the best average performance when the given number of grasps is less than 2000. The advantage is especially significant in the case of some difficult problems like Problem 5-7, where the batch PRM*-MG can be 6-10 times faster than the naïve version.

Based on the presented results, it can be concluded that, when the number of grasps is small (less than 100, which is the usual case if use this algorithm in practice), batch PRM*-MG provides the best performance. When the number of grasps is large, using standard PRM*-MG is preferable. In terms of time to initial solution, although naïve PRM*-MG attempts to return the first feasible solution as soon as possible, the average performance is actually worse due to wasting time on collision checking for infeasible solutions. Also, when the given number of grasps is less than 1000, the planning time does not change significantly as the number of grasps increases for the standard version. For some problems, searching a larger grasp set requires even less time while the solution cost drops significantly, and the success rate increases. This counterintuitive fact suggests that providing a slightly larger grasp set will benefit both the time performance and solution cost.

Table 4.2 Time and cost results for solving 8 different problems with 3 versions of PRM*-MG.

	Time to initial Solution(s)			Total Run Time(s)			Cost of Initial Solution		
Prob. No.	naive	std.	batch	naive	std.	batch	naive	std.	batch
1	1.21	**0.35**	0.63	2.38	0.71	**0.63**	10.39	7.27	**5.81**
2	0.94	**0.68**	0.73	1.95	1.07	**0.73**	10.53	7.79	**6.29**
3	1.24	**0.34**	0.61	2.20	0.71	**0.61**	10.22	7.40	**6.10**
4	1.69	**0.61**	0.82	3.59	1.16	**0.82**	10.99	7.56	**6.25**
5	5.61	2.05	**1.59**	9.65	3.67	**1.59**	10.55	9.04	**7.81**
6	20.25	2.89	**2.29**	26.96	4.99	**2.29**	11.38	9.29	**8.54**
7	5.38	1.29	**1.06**	7.66	1.39	**1.06**	9.81	7.96	**7.76**
8	2.82	**1.12**	1.28	4.82	1.99	**1.28**	10.83	8.40	**7.39**
ave.	4.89	1.16	**1.13**	7.40	1.96	**1.13**	10.59	8.09	**6.99**

Since standard PRM*-MG is a better choice when a large set of feasible grasps is available, a second experiment is performed to study the behaviour of standard PRM*-MG using different batch sizes. Figure 4.6(a) shows that, as the batch size increase, the planning time drops for a period before increasing again. This fact suggests that using a medium batch size (between 100-500) is preferable, which balances the anytime performance as well as the total run time. From Figure 4.6(b), it can also be found that the cost of the initial solution is very close to the optimal cost after evaluating 500 grasps.

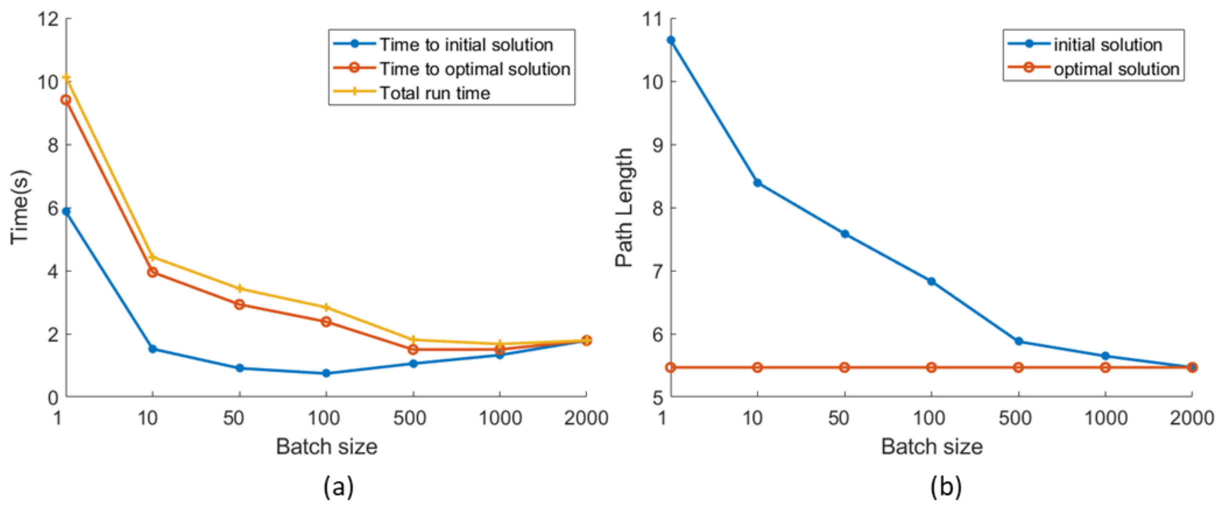


Figure 4.6 Performance comparison of batch PRM*-MG when using different batch sizes. (a) Algorithm run time versus batch size. (b) Solution cost (path length) versus batch size.

4.6 Conclusion

This chapter presents PRM*-MG, an algorithm for integrated grasp selection and post-grasp optimal motion planning. Given a set of feasible grasp candidates for an object as well as the start and goal pose, PRM*-MG can quickly select the optimal grasp from the given grasp set as well generate the corresponding optimal motion to manipulate the object. The algorithm uses a lazy strategy that only checks collision for the current optimal path. The algorithm also divides the whole grasp set into several batches to balance the anytime performance and the total computation time. The optimal cost from the previously searched batches is used as an upper bound to prune the graph in subsequent searches. A data structure that contains minimum information related to a grasp and its corresponding graph is also designed to avoid copying and storing multiple graphs.

Three different versions of the proposed algorithms are compared against each other. All three algorithms can successfully find the optimal solution given the same grasp set and same graph. However, it is found that batch PRM*-MG performs best when the size of the grasp set is small to medium and when the problem is difficult, while standard PRM*-MG performs better in terms of finding the initial solution and when an extremely large grasp set is given. Naïve

PRM*-MG does not perform as well as the other two since it wastes time on checking collision for the suboptimal solution path.

The method presented in this chapter is theoretically sound and can find satisfactory path in the offline scenario. For some online scenario, as long as the time restriction is not too tight, it will still work. However, in order to build a truly responsive robot system, a faster algorithm is required. In the next chapter, it will be shown how existing anytime sampling-based algorithm can be modified to solve the integrated grasp selection and motion planning problem to further accelerate the planning process and find better path.

Chapter 5 An anytime, sampling-based planning algorithm for manipulating objects with multiple available grasps

5.1 Introduction

In Chapter 4, the problem of manipulation planning given multiple available grasps is studied. The algorithm (PRM*-MG) developed in Chapter 4 has nice theoretical properties including asymptotical optimality and probabilistic completeness. However, like PRM, PRM*-MG is not an anytime algorithm i.e., the algorithm stops once the graph search process for all grasps is finished. If the output solution is not desired (or even no feasible solution is found), more vertices have to be added to the graph and the graph has to be searched again from scratch, which is extremely inefficient. Indeed, there are post-processing methods that can quickly improve the solution found by sampling-based motion planning algorithms [119–121]. However, post-processing can only improve the solution locally and it generally requires a feasible path as input.

Therefore, to obtain a satisfactory solution without adding more vertices online, one has to build a large enough graph offline. In this way, the constructed graph will have a high probability of containing a satisfactory solution path. However, a large graph will delay the finding of the first solution, which is not desired in the case of online planning. This is especially more difficult for PRM*-MG than the PRM since it must perform collision checking online due to a change of geometry when grasping the object with a different pose.

Due to the above reasons, although PRM*-MG can be used as an online algorithm, the performance may not be satisfactory. On the other hand, single query planners like RRT and RRT* are good at solving motion planning problem online since they can provide a feasible solution quickly and iteratively improve the solution. Therefore, it would be ideal that a single-query motion planner can be designed to solve the problem. To convert an asymptotically optimal single query planner like RRT* to solve the integrated grasp selection and motion planning problem is not hard. A straightforward idea is to grow a tree for each grasp. After generating a random vertex in each iteration, a random grasp (tree) is chosen to be extended. In this way, the planner will solve the optimal motion planning problem for all grasps and obtain the least cost path and grasp. However, it is obvious that this naïve approach is highly inefficient, especially if the number of grasps is large. Therefore, it is important that the designed algorithm constructs a tree (or multiple trees) biased towards the grasp that is most likely to yield an optimal solution.

In this chapter, RRT*-MG is proposed to solve the grasp optimised motion planning problem. The proposed algorithm achieves the desired behaviour by 1). considering both cost-to-come and cost-to-go value of a vertex when connecting the new vertex to the tree; 2). pruning or reassigning vertices to trees that can be improved; 3) balancing exploitation and exploration. In addition, informed sampling and bidirectional search are also introduced to accelerate the search process. Experiments show that RRT*-MG usually finds a feasible solution faster and yields a better solution when sampling the same number of vertices compared to PRM*MG.

The rest of the chapter is organised as follows: Section 5.2 introduces preliminaries on RRT* and its variants. Section 5.3 presents the developed algorithm in detail. Experiments and benchmark results against PRM*-MG are reported in Section 5.4. Section 5.5 provides concluding remarks for this chapter.

5.2 Preliminaries

The proposed algorithm builds upon existing single query motion planning algorithms like RRT* and its variants. In this section, RRT* is first introduced and then two of its variants (Bi-RRT* and Informed RRT*) are reviewed. To be consistent with previous chapters, an abstract point in the robot configuration space and vertex set is denoted by x while the specific configuration (i.e., the column vector associated with it) is denoted by bold characters e.g., $\boldsymbol{x} \in \mathbb{R}^{\mathrm{d}}$ for matrix computation .

5.2.1 RRT*

The RRT* algorithm is presented in Algorithm 5.1 – Algorithm 5.5. This is a slightly modified version from the original version presented in [12,122]. Some important procedures and subfunctions are introduced below before introducing the full algorithm:

Nearest: This function queries the closet vertex in the tree from the given vertex.

$$\text{Nearest}(G = (V, E), x) := \underset{v \in V}{\text{argmin}} \|\boldsymbol{x} - \boldsymbol{v}\|_2$$

Near: This function queries all the points inside a ball centred at the given vertex (x) within radius r. This is the same procedure used in Algorithm 4.1 in generating the probabilistic roadmap.

$$\text{Near}(\mathrm{G} = (V, E), x) := \{v \in V \mid \|\boldsymbol{x} - \boldsymbol{v}\|_2 < \mathrm{r}\}$$

Steer: Given two points $x, y \in X$, this function returns a point $z \in X$ which is closer to y than x is. For example, on Line 2 of Algorithm 5.2, the function steers $x_{nearest}$ to x_{rand} by generating a new vertex x_{new}. Usually, x_{new} is on the straight line connecting $x_{nearest}$ and x_{rand} with a predefined distance η to $x_{nearest}$. η can be viewed as a step size, which limits the growth of the tree.

$$\text{Steer} := \underset{z \in X}{\text{argmin}} \{\|\boldsymbol{z} - \boldsymbol{y}\|_2 \mid \|\boldsymbol{z} - \boldsymbol{x}\|_2 \leq \eta\}$$

Using a step size instead of connecting $x_{nearest}$ and x_{rand} directly is beneficial since if a step is too long, it is more likely to be in collision. Besides, in some nonholonomic system, it is infeasible to connect two random points in the space directly. Occasionally, Steer will also be used to generate an edge connecting z and x directly (e.g., Line3, Algorithm 5.3).

CollisionFree: This function tests whether the path given is in collision or not. The function returns true if there is no collision (Note this is a simplified version than the one introduced in Chapter 4).

Cost: When this function is applied to a vertex, it will compute the cost of the path from the start vertex to the given vertex along the tree. When this function is called given an edge or a set of edges (i.e., the path) as an input, it will return the cost of the edge (e.g., Euclidean length).

Parent: Given a tree structure and a node, the Parent function queries the parent of the node on the given tree.

ExtendAndRewire (Algorithm 5.2): This is the main part of RRT*. In each iteration, RRT* performs extend and rewire operation after sampling a new vertex. The extend operation grows the current tree towards the newly generated vertex along a minimum cost path, while the rewire operation uses the newly generated vertex to modify the path to existing vertices if a better path is found.

GetSortedLists (Algorithm 5.3): Given a vertex set X_{near}, this function sorts all vertices in the set based on the distance from the start vertex (x_{start}) to x_{new} while passing $x_{near} \in X_{near}$. In this way, the vertex with the shortest distance to x_{new} will be placed on the front of the set, which avoids unnecessary collision checking.

GetBestParent (Algorithm 5.4): This function checks collision for all edges stored in L_{near}. Since L_{near} is sorted, once a collision-free edge is found, the function returns immediately.

RRT* samples a new point in the robot configuration space each iteration and uses it to generate x_{new}. However, it does not simply connect the nearest point $x_{nearest}$ to the new sample x_{new} but considers all points within distance $\mathrm{r(n)}$ from x_{new} where $\mathrm{r(n)} = min\{\gamma_{RRT*}(\log(\mathrm{n})/\mathrm{n})^{1/\mathrm{d}}, \eta\}$. On Line 3 of Algorithm 5.2, the Near function may return an empty set. This is because the connection radius $\mathrm{r(n)}$ may be less than the step size η used in Steer. Therefore, to ensure there is at least one vertex in X_{near}, $x_{nearest}$ is always added. If GetBestParent succeed (Line 6 of Algorithm 5.2), then x_{new} can be connected to the current tree successfully. Therefore, the new vertex and new edge are added to the vertex set and edge set respectively (Line 7 and 8 of Algorithm 5.2). After adding the new vertex, the algorithm examines all vertices in X_{near} again to check whether a path passing x_{new} to x_{near} can yield a better path than the original path to x_{near} (Line 2 of Algorithm 5.5). If there is such a path, the parent of corresponding x_{near} will be set as x_{new} by removing the old edge and adding the new edge (Line 5 and 6 of Algorithm 5.5)

Algorithm 5.1: RRT*

```
1.  $V \leftarrow \{x_{start}\};\ E \leftarrow \emptyset;$
2.  for $i = 1, \dots, n$ do
3.      $x_{rand} \leftarrow \text{SampleFree}(i)$
4.      $\text{ExtendAndRewire}(x_{rand})$
5.  endfor
6.  return $T = (V, E)$
```

Algorithm 5.2: ExtendAndRewire(x_{rand}, E, V)

```
1.  $x_{nearest} \leftarrow \text{Nearest}(V, x_{rand})$
2.  $x_{new} \leftarrow \text{Steer}(x_{nearest}, x_{rand})$
3.  $X_{near} \leftarrow \text{Near}(V, x_{new}) \cup x_{nearest}$
4.  $L_{near} \leftarrow \text{GetSortedLists}(X_{near}, x_{new})$
5.  $x_{parent} \leftarrow \text{GetBestParent}(L_{near})$
6.  if $x_{parent} \neq$ NULL
7.      $V \leftarrow V \cup \{x_{new}\}$
8.      $E \leftarrow E \cup \{(x_{parent}, x_{new})\}$
9.      $E \leftarrow \text{RewireVertices}(x_{new}, L_{near}, E)$
10. endif
11. return $x_{new}$
```

Algorithm 5.3: GetSortedLists(X_{near}, x_{new})

```
1.  $L_{near} \leftarrow \emptyset$
2.  foreach $x_{near} \in X_{near}$ do
3.      $\sigma_{near} \leftarrow \text{Steer}(x_{near}, x_{new})$
4.      $c_{near} \leftarrow \text{Cost}(x_{near}) + \text{Cost}(\sigma_{near})$
5.      $L_{near} \leftarrow L_{near} \cup \{(c_{near}, x_{near}, \sigma_{near})\}$
6.  endfor
7.  $\text{Sort}(L_{near})$
8.  return $L_{near}$
```

Algorithm 5.4: GetBestParent(L_{near})

```
1. foreach ($c_{near}, x_{near}, \sigma_{near}$) $\in L_{near}$ do
2.    if CollisionFree($\sigma_{near}$) then
3.       return $x_{near}$
4.    endif
5. return NULL
```

Algorithm 5.5: RewireVertices(E, L_{near}, x_{new})

```
1. foreach ($c_{near}, x_{near}, \sigma_{near}$) $\in L_{near}$ do
2.    if Cost($x_{new}$) + Cost($\sigma_{near}$) < Cost($x_{near}$) then
3.       if CollisionFree($\sigma_{near}$) then
4.          $x_{oldparent} \leftarrow$ Parent(E, $x_{near}$)
5.          $E \leftarrow E \setminus \{(x_{oldparent}, x_{near})\}$
6.          $E \leftarrow E \cup \{(x_{new}, x_{near})\}$
7.       endif
8.    endif
9. endfor
10. return $E$
```

5.2.2 Bidirectional RRT*

Bi-RRT* is a bidirectional version of RRT* which uses a similar connection strategy from RRT-connect [24]. A slightly simplified version is presented in Algorithm 5.6 and Algorithm 5.7. In Bi-RRT*, Instead of growing a single tree, two trees (T_a and T_b) are grown from the start and goal vertex (x_{init} and x_{goal}) towards each other interchangeably. The algorithm is exactly the same as RRT* for growing the first tree. After the ExtendAndRewire procedure is finished for the first tree. The new vertex is used as a random sample for growing the other tree. Therefore, the nearest vertex to x_{new} in T_b is queried (Line 1 of Algorithm 5.7). However, a direct connection from $x_{new}\ to\ x_{connect}$ may not maintain the optimality of RRT*. Instead, an additional vertex x'_{new} is created by steering from $x_{connect}$ to x_{new}, and all points in T_b within distance r(n) from x'_{new} are considered for potential connection (Line 3-5 of

Algorithm 5.7). If there exists a point in X_{near} that can be connected to x_{new}, then a new edge is added, and the path σ_{sol} is generated by the function GeneratePath. After a successful solution is generated, it is added to the solution set (Line 6-7 of Algorithm 5.6). The iteration ends after two trees are swapped. In this way, the tree extended in this iteration will be used for connection in the next iteration and vice versa.

Algorithm 5.6: Bi-RRT*

```
1.  $V \leftarrow \{x_{satrt}, x_{goal}\}$; $E \leftarrow \emptyset$; $T_a \leftarrow (x_{start}, \mathrm{E})$, $T_b \leftarrow (x_{goal}, E)$; $\Sigma_{sol} \leftarrow \emptyset$
2.  for i = 1,…,n do
3.      $x_{rand} \leftarrow$ SampleFree(i)
4.      $x_{new} \leftarrow$ ExtendAndRewire($x_{rand}$)
5.      $\sigma_{sol} \leftarrow$ ConnectTrees($T_b, x_{new}$)
6.      if $\sigma_{sol} \neq \emptyset$
7.          $\Sigma_{sol} \leftarrow \Sigma_{sol} \cup \sigma_{sol}$
8.      endif
9.      SwapTrees($T_a, T_b$)
10. endfor
11. return $T = (V, E)$
```

Algorithm 5.7: ConnectTrees(T_b, x_{new})

```
1.  $x_{connect} \leftarrow$ Nearest($T_b, x_{new}$)
2.  $x'_{new} \leftarrow$ Steer($x_{connect}, x_{new}$)
3.  $X_{near} \leftarrow$ Near($T_b, x'_{new}$) $\cup\ x_{connect}$
4.  $L_{near} \leftarrow$ GetSortedLists($X_{near}, x_{new}$)
5.  $x_{\min} \leftarrow$ GetBestParent($L_{\mathrm{near}}$)
6.  if $x_{min} \neq$ NULL
7.      $E \leftarrow E \cup \{(x_{\min}, x_{\mathrm{new}})\}$
8.      $\sigma_{sol} \leftarrow$ GeneratePath($x_{min}, x_{new}$)
9.      return $\sigma_{sol}$
10. endif
11. return NULL
```

One may notice that ConnectTrees is actually quite similar to ExtendAndRewire apart from no rewiring is performed. Another difference is that unlike in ExtendAndRewire, since the goal is to generate a connected path, no step size is used to limit the growth of the tree, i.e., the tree attempts to connect to x_{new} directly rather than x'_{new} generated by the steering function. However, these are only implementation choices and can be modified to serve different preferences.

5.2.3 Informed RRT*

Informed RRT* [22] works the same as RRT* before finding the first feasible solution. However, it uses the concept of the informed set to generate new samples in the configuration space once a feasible solution is found. Generating samples inside the informed set is a necessary condition for improving the current solution (see Lemma 12 in [23]). As the cost of the optimal solution improves each iteration, the informed set will also be updated (i.e., the size of informed set will be reduced). In this way, the planning algorithm avoids generating new samples that cannot improve the current solution and results in faster convergence. This section reviews algorithms introduced in [23] with some theoretical details omitted. The definition of informed set will be given in Section 5.2.3.1. In Section 5.2.3.2, a direct sampling method is presented to improve the sample efficiency. Section 5.2.3.3 generalise the direct sampling method to the case of multiple pairs of start and goal state, which is particularly useful for the problem considered in this chapter.

5.2.3.1 L^2 Informed set

Let $\hat{f}(x)$ be a heuristic function that estimates the cost of the solution path from x_{start} to x_{goal} while passing state $x \in X$. An informed set is defined as all points $x \in X$, such that the $\hat{f}(x)$ is less than the optimal solution cost of the current iteration c_i. An informed set is admissible if the heuristic function used is admissible. In other words, it always underestimates the true optimal cost $f(x)^*$ passing x, i.e., $\hat{f}(x) < f(x)^*$. A commonly used, universally admissible $\hat{f}(x)$ is given as follows:

$$\hat{f}(x) = \|\boldsymbol{x} - \boldsymbol{x}_{start}\|_2 + \|\boldsymbol{x} - \boldsymbol{x}_{goal}\|_2$$

where $\|\boldsymbol{x} - \boldsymbol{x}_{start}\|_2$ is the L^2 norm (Euclidean distance) between x and x_{start} while $\|\boldsymbol{x} - \boldsymbol{x}_{goal}\|_2$ is the L^2 norm between x and x_{goal}.

Therefore, the L^2 informed set can be defined as follows:

$$X_{\hat{f}} := \{x \in X_{free} \mid \|\boldsymbol{x} - \boldsymbol{x}_{start}\|_2 + \|\boldsymbol{x} - \boldsymbol{x}_{goal}\|_2 < c_i\}.$$

Obviously, $X_{\hat{f}}$ is an intersection between the space of all collision-free robot configuration (X_{free}) and a d-dimensional hyperellipsoid symmetric about its transverse axis (Figure 5.1). Two focal points of the hyperellipsoid are x_{start} and x_{goal}. The transverse diameter and conjugate diameter are $\mathrm{c_i}$ and $\sqrt{\mathrm{c_i^2} - \mathrm{c_{min}^2}}$ respectively, where $\mathrm{c_{min}} = \|\boldsymbol{x}_{goal} - \boldsymbol{x}_{start}\|_2$ is the Euclidean distance between start and goal.

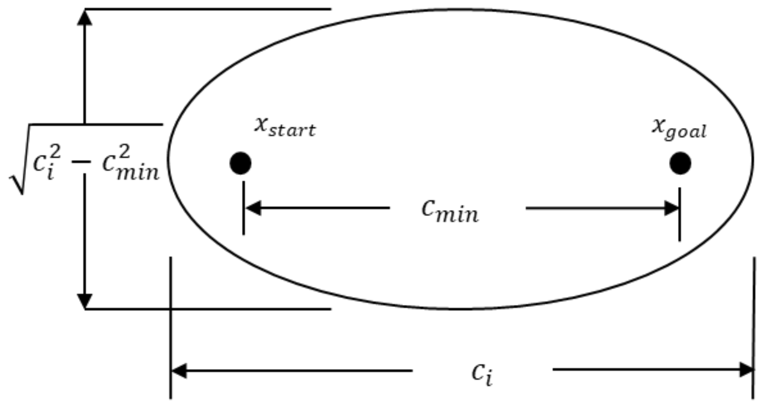


Figure 5.1 The L^2 informed set without obstacles is a hyperellipsoid.

5.2.3.2 Direct Sampling

Direct sampling the informed set is not practical because sampling-based motion planning algorithms always represent the free space implicitly. However, it is possible to sample the hyperellipsoid uniformly. The interior of a hyperellipsoid is defined as

$$X_{ellipsoid} := \{\boldsymbol{x} \in \mathbb{R}^n \mid (\boldsymbol{x} - \boldsymbol{x}_{centre})^T \mathbf{S}^{-1} (\boldsymbol{x} - \boldsymbol{x}_{centre}) < 1\}$$

where $\mathbf{S}$ is a symmetric positive definite matrix that defines the hyperellipsoid and x_{centre} is the centre of the hyperellipsoid [123]. In this chapter, the centre is always the middle point on the straight line connecting x_{start} and x_{goal}, i.e., $\boldsymbol{x}_{centre} := (\boldsymbol{x}_{start} + \boldsymbol{x}_{goal})/2$. To uniformly sample the interior of the hyperellipsoid, first, a new point x_{ball} is sampled from a d-dimensional unit ball. Then x_{ball} is transformed into a point in the hyperellipsoid $x_{ellipsoid}$ by

$$\boldsymbol{x}_{ellipsoid} = \mathbf{L}\boldsymbol{x}_{ball} + \boldsymbol{x}_{centre}$$

where $\mathbf{L} \in \mathbb{R}^{n \times n}$ is a lower triangular matrix and $\mathbf{L}\mathbf{L}^{\mathrm{T}} = \mathbf{S}$. $\mathbf{S}$ can be chosen to be diagonal for hyperellipsoid with orthogonal axes to simplify the computation,

$$\mathbf{S} := \operatorname{diag}(\mathrm{r}_1^2, \mathrm{r}_2^2, \dots, \mathrm{r}_\mathrm{n}^2),$$

where r_j is the radius of the jth axis and $\operatorname{diag}(\cdot)$ represents a diagonal matrix. For the problem considered in this chapter,

$$\mathbf{S} = \operatorname{diag}(\frac{\mathrm{c}_\mathrm{i}^2}{4}, \frac{\mathrm{c}_\mathrm{i}^2 - \mathrm{c}_\mathrm{min}^2}{4}, \dots, \frac{\mathrm{c}_\mathrm{i}^2 - \mathrm{c}_\mathrm{min}^2}{4}).$$

Therefore, $\mathbf{L}$ can be obtained by the following equation:

$$\mathbf{L} = \operatorname{diag}(\frac{\mathrm{c}_\mathrm{i}}{2}, \frac{\sqrt{\mathrm{c}_\mathrm{i} - \mathrm{c}_\mathrm{min}}}{2}, \dots, \frac{\sqrt{\mathrm{c}_\mathrm{i} - \mathrm{c}_\mathrm{min}}}{2}).$$

A rotational transformation $\boldsymbol{C} \in SO(n)$ should be applied to transform the ellipsoid from the frame it is defined to the desired frame (i.e., the frame defined by x_{start} and x_{goal})

$$\boldsymbol{x}_{ellipsoid} = \mathbf{C}\mathbf{L}\boldsymbol{x}_{ball} + \boldsymbol{x}_{centre}.$$

Finding $\boldsymbol{C}$ between two frames is a well-studied problem known as Wahba's problem [124]. The solution can be computed by the following procedures. First a matrix $\mathbf{M}$ is constructed by

$$\mathbf{M} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \dots, \boldsymbol{a}_j][\boldsymbol{b}_1, \boldsymbol{b}_2, \dots, \boldsymbol{b}_j]^{\mathrm{T}}$$

where $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \dots, \boldsymbol{a}_\mathrm{n}\}$ is a set of frame axes and $\{\boldsymbol{b}_1, \boldsymbol{b}_2, \dots, \boldsymbol{b}_\mathrm{n}\}$ is another set of frame axes. $\mathbf{M}$ can still be constructed even when some axes are unknown, which is the case of this chapter:

$$\mathbf{M} = \boldsymbol{a}_1\mathbf{I}_1^{\mathrm{T}}$$

where $\boldsymbol{a}_1$ is the transverse axis after rotational transformation:

$$\boldsymbol{a}_1 = (\boldsymbol{x}_{goal} - \boldsymbol{x}_{start}) / \left\| \boldsymbol{x}_{goal} - \boldsymbol{x}_{start} \right\|_2$$

while $\mathbf{I}_1$ is the first column of the identity matrix $\mathbf{I}$.

Then, a singular value decomposition is performed to get orthogonal matrices $\mathbf{U} \in \mathbb{R}^n$ and $\mathbf{V} \in \mathbb{R}^n$ from $\mathbf{M}$:

$$\mathbf{U\Sigma V^T} = \mathbf{M}.$$

Finally, the rotation matrix is given by

$$\mathbf{C} = \mathbf{U\Lambda V^T}$$

where $\boldsymbol{\Lambda} = \mathrm{diag}\big(1, \dots, 1, \det(\mathbf{U})\det(\mathbf{V})\big) \in \mathbb{R}^{n \times n}$ and $\det(\cdot)$ computes the determinant of the matrix. In this way, the generated samples $x_{ellipsoid}$ are uniformly distributed inside hyperellipsoid.

The full algorithm for informed sampling is given in Algorithm 5.8 and Algorithm 5.9. μ is the volume of a space. In case that the informed set is larger than the planning domain, the algorithm still samples X directly.

Algorithm 5.8: Sample(x_{start}, x_{goal}, c_i)

```
1.  repeat
2.     if $\mu(X_{ellipsoid}) < \mu(X)$ then
3.         $x_{rand} \leftarrow$ SampleEllipsoid($x_{start}, x_{goal}, c_i$)
4.     else
5.         $x_{rand} \leftarrow$ SamplePeoblem(X)
6.     endif
7.  until $x_{rand} \in X_{free} \cap X_{ellipsoid}$
8.  return $x_{rand}$
```

```
Algorithm 5.9: SampleEllipsoid($x_{start}, x_{goal}, c_i$)
1. $c_{min} \leftarrow \|\boldsymbol{x}_{goal} - \boldsymbol{x}_{start}\|_2$
2. $\boldsymbol{x}_{centre} \leftarrow (\boldsymbol{x}_{start} + \boldsymbol{x}_{goal})/2$
3. $\boldsymbol{a}_1 \leftarrow (\boldsymbol{x}_{goal} - \boldsymbol{x}_{start})/c_{min}$
4. $\{\mathbf{U}, \mathbf{V}\} \leftarrow \mathrm{SVD}(\boldsymbol{a}_1 \mathbf{I}_1^{\mathrm{T}})$
5. $\boldsymbol{\Lambda} \leftarrow \mathrm{diag}(1, \dots, \det(\mathbf{U}) \det(\mathbf{V}));$
6. $\mathbf{C} \leftarrow \mathbf{U\Lambda V^T}$
7. $r_1 \leftarrow c_i/2$
8. $\mathbf{L} \leftarrow \mathrm{diag}(c_i/2\,, \sqrt{c_i - c_{min}}/2, \dots, \sqrt{c_i - c_{min}}/2)$
9. $x_{ball} \leftarrow \mathrm{SampleUnitBall(n)}$
10. $\boldsymbol{x}_{rand} \leftarrow \mathbf{CL}\boldsymbol{x}_{ball} + \boldsymbol{x}_{centre}$
11. return $x_{rand}$
```

5.2.3.3 Direct Sampling for Multiple Pairs of Start and Goal

The problem considered in this chapter involves multiple pairs of start and goal which means each pair of start and goal corresponds to an ellipsoid. A method for sampling the informed set for multiple goals is given in [23], which is also applicable for multiple pairs of start and goal. Assuming there are z pairs of start and goal, the generated new samples should uniformly distribute in z ellipsoids. A straightforward approach is to probabilistically select an ellipsoid for a pair of start and goal based on the volume of the ellipsoid. However, since ellipsoids may overlap with each other, the generated sample will be probabilistically rejected based on the number of ellipsoids containing this sample. The algorithm for uniformly sampling multiple informed ellipsoids is given in Algorithm 5.10 - Algorithm 5.12. The algorithm presented uses a loop for clarity, however, when implementing the program, vectorised operations should be used to accelerate the computation, especially when dealing with a large number of start and goal pairs.

```
Algorithm 5.10: Sample($\{x_{start}\}, \{x_{goal}\}, c_i$)
1. $j \leftarrow$ NULL
2. repeat
```

```
3.      if $\frac{1}{z}\sum_{j=1}^{z}\lambda(X_{ellipsoid,j}) < \lambda(X)$ then
4.          $x_{goal}, j \leftarrow$ RandomGoal($\{x_{start}\}, \{x_{goal}\}, c_i$)
5.          $x_{rand} \leftarrow$ SampleEllipsoid($x_{start}, x_{goal}, c_i$)
6.      else
7.          $x_{rand} \leftarrow$ SamplePeoblem(X)
8.      endif
9.  until $x_{rand} \in X_{free} \cap (\bigcup_{j=i}^{z} X_{ellipsoid,j})$ and
    KeepSample($x_{rand}, \{x_{start}\}, \{x_{goal}\}, c_i$)
10. return $x_{rand}, j$
```

```
Algorithm 5.11: RandomGoal($\{x_{start}\}, \{x_{goal}\}, c_i$)
1.  a = $\sum_{j=1}^{z}\lambda(X_{ellipsoid,j})$
2.  $p \leftarrow$ UniformSample(0, 1)
3.  $j \leftarrow 0$
4.  repeat
5.      $j \leftarrow j + 1$
6.      $p \leftarrow p - \lambda(X_{ellipsoid,j})/a$
7.  until $p \leq 0$
8.  return $x_{goal}, j$
```

```
Algorithm 5.12:KeepSample($x_{rand}, \{x_{start}\}, \{x_{goal}\}, c_i$)
1.  $a \leftarrow \mathbf{0}$
2.  for $k = 1, \ldots z$
3.      if $\|\boldsymbol{x}_{rand} - \boldsymbol{x}_{start,k}\|_2 + \|\boldsymbol{x}_{rand} - \boldsymbol{x}_{goal,k}\|_2 < c_i$ then
4.          $a = a + 1$
5.      endif
6.  endfor
7.  $p \leftarrow$ UniformSample(0, 1)
8.  return $p \leq 1/a$
```

5.3 Algorithm

In this section, the proposed algorithm RRT*-MG is described. The proposed algorithm is based on RRT* and incorporates ideas from the variants introduced in Section 5.2. However, a number of key modifications are introduced to bias the search towards grasps that are more likely to generate a good solution. To simplify the presentation of the algorithm, no kinematic redundancy is considered, i.e., each grasp only corresponds to a single pair of start and goal. However, the developed method applies to the case of multiple IK solutions straightforwardly.

Since the problem involves multiple pairs of start and goal configuration, RRT*-MG grows multiple trees instead of a single tree as presented in the original RRT* algorithm. In this way, each tree belongs to a pair of start and goal configuration. Each tree is denoted as T_i and the vertex and edge set of the tree is denoted as V_i and E_i respectively. The graph containing multiple trees is known as a forest $\mathbb{T} := \{T_{i=1,\dots,n}\}$. The vertex set and edge set of the forest are defined as $\mathbb{V} := \{V_{i=1,\dots,n}\}$ and $\mathbb{E} := \{E_{i=1,\dots,n}\}$.

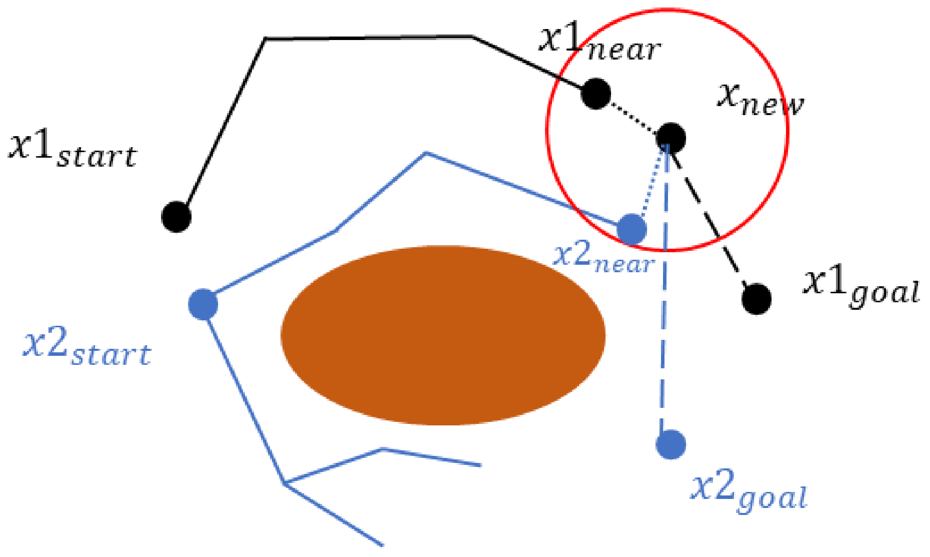
Although multiple trees are initialised, to maintain the efficiency of the algorithm, it would be ideal that only one tree is extended in each iteration. If the extended tree always belongs to the optimal grasp, the algorithm essentially solves a single start single goal planning problem, which can be extremely efficient. However, it is impossible to know which grasp is optimal. Therefore, to avoid wasting time on planning for suboptimal grasps, a critical question is which tree should be extended given a randomly sampled point in each iteration. Section 5.3.1 introduces a simple heuristic for choosing which tree to be extended. Section 5.3.2 discusses the rewiring strategy used in the proposed algorithm and some special cases that need to be careful about. Section 5.3.3 introduces another procedure called pruning and reassigning which discusses how vertices should be managed correctly if they cannot improve the current solution. In Section 5.3.4, a strategy is introduced to balance exploration and exploitation so that search efforts are not spent on a single grasp too much.

5.3.1 Making New Connections

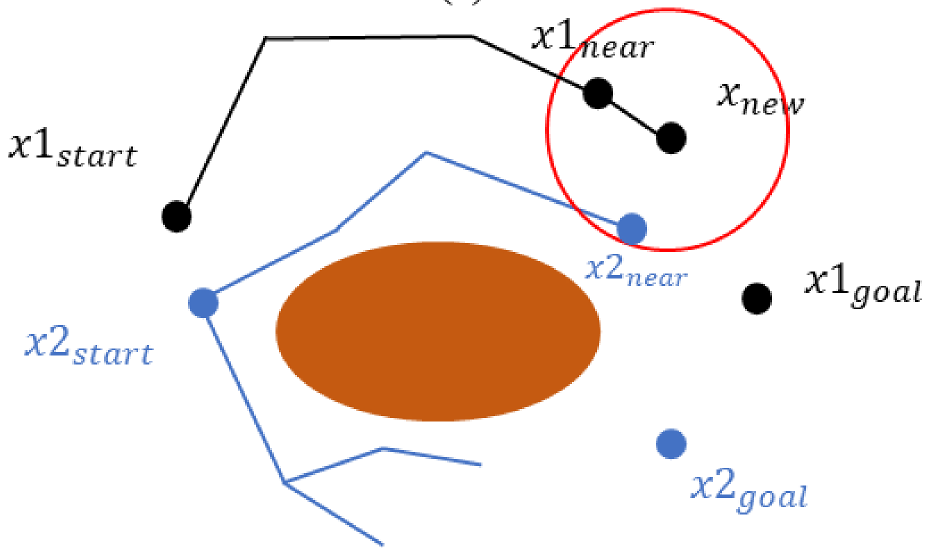
As shown in Algorithm 5.1, the original RRT* algorithm steers the nearest vertex to $x_{r\text{and}}$ to create a new vertex $x_{n\text{ew}}$. Then, all vertices within radius r of x_{new} are queried and the one with the least cost-to-come to $x_{n\text{ew}}$ is connected with x_{new}. In RRT*-MG, a similar procedure can be employed as shown in Figure 5.2. Each iteration, after a random point is generated, the nearest vertex $x_{nearest}$ in the vertex set ($\mathbb{V}$) is queried and steered to a new vertex $x_{n\text{ew}}$. Then all vertices in $\mathbb{V}$ within radius r of $x_{n\text{ew}}$ are queried to construct a set $X_{near} \leftarrow \{x_{near}\}$ and the one with the least cost c_{near} is connected to $x_{n\text{ew}}$, where

$$c_{near} = Cost(x_{near}) + Cost(x_{near}, x_{new}) + \hat{h}(x_{new}). \quad (5.1)$$

Here, considering both cost-to-come along the current tree and optimistically estimated cost-to-go is a critical step to make the proposed algorithm run fast. Since different grasps have different goals and starts, reaching $x_{n\text{ew}}$ from the jth start $x_{start,j}$ with a short distance (cost-to-come) does not mean the grasp will contain an efficient path reaching its corresponding goal vertex $x_{goal,j}$.



(a)



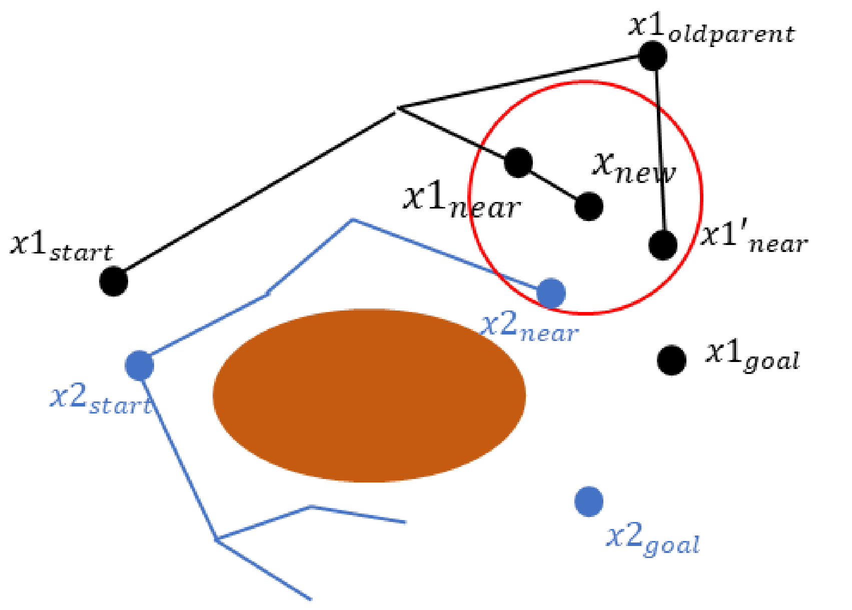
(b)

Figure 5.2 An illustration of how RRT*-MG makes new a connection. (a) Before connecting the new vertex. (b) New connection is made to black tree (dot line becomes solid). The brown ellipsoid represents the obstacle. Two trees are grown from start vertices ($x1_{start}$ and $x2_{start}$) to goal vertices ($x1_{goal}$ and $x2_{goal}$). The red circle is X_{near}. both trees have vertices inside X_{near}. However, the connection to x_{new} is made to the black tree since $Cost(x1_{near})$ (length of solid line) $+Cost(x1_{near}, x_{new})$ (length of dot line) $+ \hat{h}(x1_{new})$ (length of dash line) is smaller. Note this illustration does not represent any real system doing rigid transformation, since $x1_{goal}$ and $x2_{goal}$ should at least maintain the same distance as $x1_{start}$ and $x2_{start}$. If the system only allows translation, the orientation should be maintained as well.

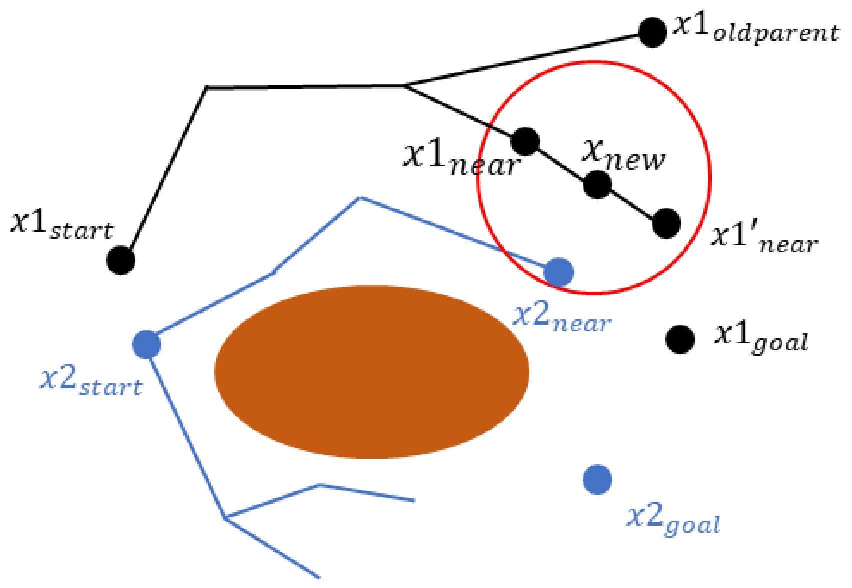
5.3.2 Rewiring and Vertex Duplicating

Another key difference to original RRT* is in the rewiring process. The rewiring process is illustrated in Figure 5.3. Since a grasp has been selected implicitly for extension, the proposed algorithm only rewires vertices that belong to the same grasp rather than all vertices in X_{near}. This is because the two cost-to-come values of a vertex through different paths cannot be compared as in the case of RRT*. A path with a larger cost-to-come value may have a lower cost-to-go value, which results in a better path. Rewiring such a vertex negatively impact the performance of the algorithm. Besides, rewiring a vertex from a different tree also requires checking the collision status of all the descendants, since the change of grasp results in the change of geometry. This also makes the algorithm run slower.

However, it is possible that an x_{near} belonging to another grasp can provide a potentially better path (i.e., belongs to the informed set) for the grasp that x_{new} belongs to. In this case, it is reasonable to add a new edge connecting x_{new} and x_{near}. In practice, to maintain the tree structure, a new vertex should be added with the same robot configuration as x_{near} instead of connecting with the original x_{near} (i.e., vertex duplicating).



(a)



(b)

Figure 5.3 An illustration of how RRT*-MG rewires a vertex. (a) Before rewiring, $x1'_{near}$ is connected to $x1_{oldparent}$. (b) After rewiring, $x1'_{near}$ is connected to x_{new} instead. Assume an additional vertex $x2_{near}$ is inside X_{near}, which belongs to the blue tree. Rewiring will only consider $x1'_{near}$ whether it can be reached from x_{new} with a shorter distance. $x2_{near}$ will not be considered for rewiring, even it may be reached from x_{new} with a shorter distance. However, if $x2_{near}$ is inside the informed set of the black tree, then it can be duplicated and added to the black tree as well.

5.3.3 Pruning and Reassigning

This procedure can be viewed as a more advanced rewiring operation. It has been discussed in Section 5.3.2 that a normal rewiring can only been performed between vertices in the same tree. However, some leaves of the tree (i.e., vertices without descendants) cannot improve the grasp tree it belongs to (i.e., outside of its informed set), which means they should be pruned[2]. This is an admissible pruning method introduced in [23]. However, for the problem studied in this chapter, they may still belong to the informed set of other grasps, then it would be beneficial to reassign them to new grasp tree. The vertices pruned from their current grasp tree but can be reassigned to other trees are denoted as $x_{reassign} \in X_{reassign}$. This behaviour is similar to rewiring, but the criterion is different. Rewiring uses the cost-to-come value while reassigning uses the informed set. Reassigning is actually equivalent to reusing these pruned samples as new samples to extend the tree. Note vertices that satisfy the reassigning condition may already be duplicated. If that is the case, reassigning is not performed (i.e., they will not be added to $X_{reassign}$). Function $\mathrm{PruneAndReassign}$ (Line 9 of Algorithm 5.13) will prune the vertices in the tree and add feasible vertices to $X_{reassign}$. The actual reassign operation is done by $\mathrm{SampleRandom}$ function, in which a new sample will be chosen from $X_{reassign}$ directly (Line 19 -20 of Algorithm 5.13). since a new sample may belong to the informed set of multiple pairs of start and goal, a grasp tree will also be chosen randomly (subject to the informed set constraint). In order to keep vertices in $X_{reassign}$ always belonging to the informed set, the samples in $X_{reassign}$ should be updated (Line 7 of Algorithm 5.13) before sampling from it as the solution cost could be improved after each iteration.

5.3.4 Balancing Exploration and Exploitation

In this section, two strategies (exploration and exploitation) will be presented for choosing

[2] In many cases, the measure of an informed set may even be zero since the cost of a straight line path from the start to the goal of current tree is higher than the optimal cost found already.

which tree to be extended. Exploration means the algorithm extends the tree according to the sampling results. Because each time a new vertex is sampled, a grasp (at the same time, a pair of start and goal state) is selected implicitly. The probability of selecting a grasp is proportional to the measure of the hyperellipsoid defined by the corresponding pair of start and goal, which ensures the tree belongs to the optimal grasp can always be extended with non-zero probability. Therefore, this procedure is useful for avoiding converging to a local minimum (i.e., the found solution path is optimal for one of the grasps, but another grasp may yield a better solution).

On the other hand, the exploitation strategy uses the connection method introduced in Section 5.3.1. It is proposed that the connection is made to vertices in X_{near} with the least cost computed by Eq. (5.1). In this way, trees with more vertices are more likely to be extended again. However, it is also possible that, after a feasible solution is found, the grasp trees of all vertices in X_{near} cannot be improved by connecting to the new vertex x_{new} (i.e., outside of the informed set). Therefore, making new connections with vertices in X_{near} is not necessary. In this case, x_{rand} will be recycled (added to $X_{reassign}$) and start the next iteration immediately without connecting with the other tree (Line 16 - 17 of Algorithm 5.13). ExtendAndRewire-MG is the same as ExtendAndRewire except that it uses Eq. (5.1) to decide which vertex to connect and check whether the connected vertex belongs to the informed set.

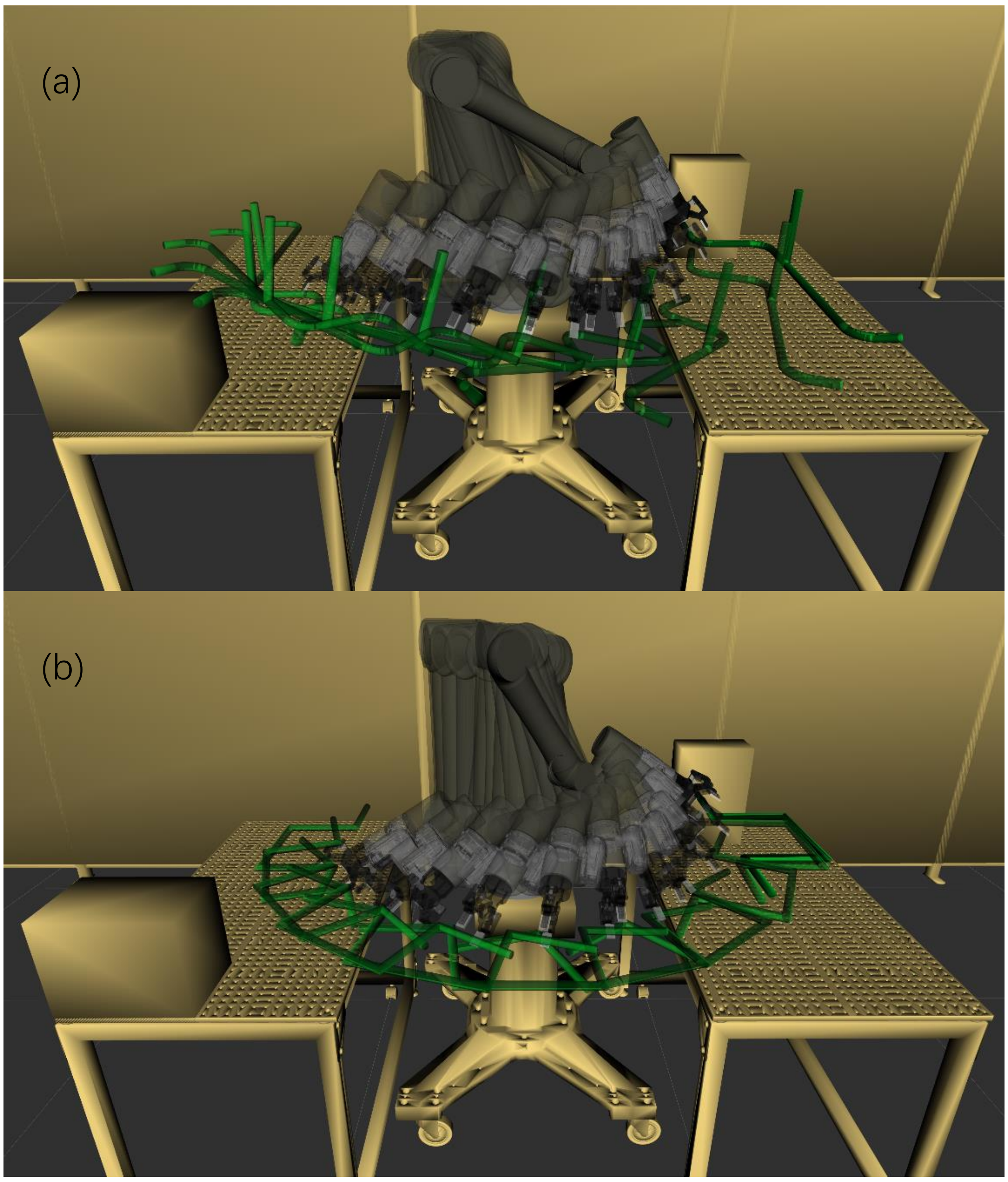
In practice, the implemented software interleaves between querying X_{near} from $\mathbb{T}$ (exploitation) and X_{near} subject to the same grasp constraint (exploration). This implementation can be used to achieve asymptotically optimality once a feasible solution is found. When reassigning vertices in $X_{reassign}$, the algorithm also performs exploration (Line 20 of Algorithm 5.13), since $X_{reassign}$ may contain vertices fail to be connected in the exploitation step of previous iterations. The full algorithm is given in Algorithm 5.13. It is worth noting that collision checking should be performed with objects attached at correct poses.

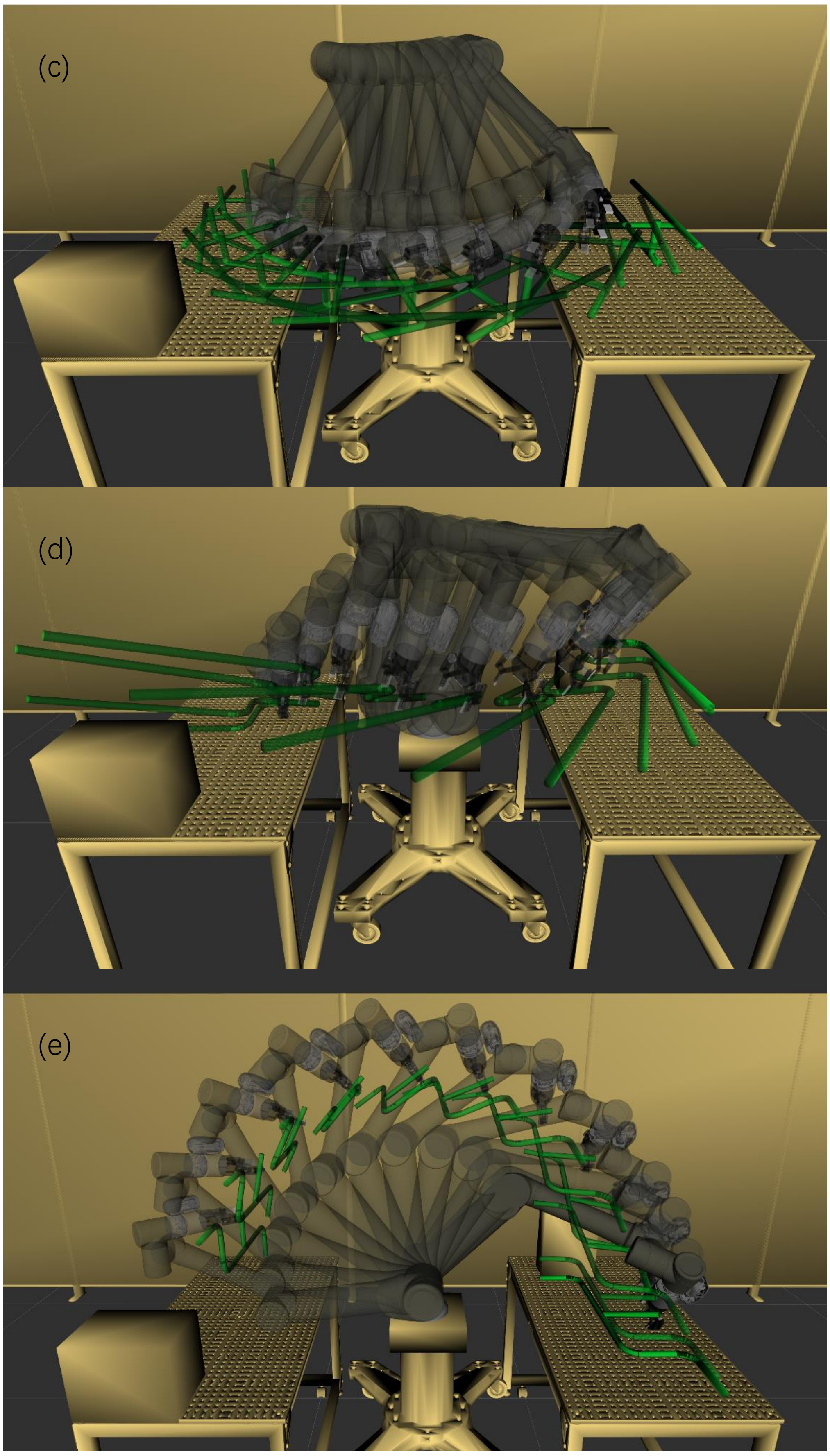
Algorithm 5.13: RRT*-MG

```
1.  for $j = 1, \dots, m$ do
2.      $V_j \leftarrow \{x_{start}, x_{goal}\}$; $E_j \leftarrow \emptyset$ // initialise $m$ trees in the graph for each feasible grasp
3.  endfor
4.  $X_{reassign} \leftarrow \emptyset$; $c_o \leftarrow$ INFINITY
5.  $\mathbb{V} \leftarrow \{V_j\}$; $\mathbb{E} \leftarrow \{E_j\}$; $\mathbb{T} = \{\mathbb{V}, \mathbb{E}\}$
6.  for $i = 1, \dots, n$ do
7.      $c_i \leftarrow$ BestSolutionCosts$(\Sigma_{sol})$; Update$(X_{reassign})$
8.      if $c_i < c_{i-1}$
9.          PruneAndReassign$(\mathbb{T} = \{\mathbb{V}, \mathbb{E}\}, X_{reassign}\ c_i)$
10.     endif
11.     if $X_{reassign} == \emptyset$
12.         $\{x_{rand}, j\} \leftarrow$ Sample$(\{x_{start}\}, \{x_{goal}\}, c_i)$
13.         if $c_i <$ INFINITY and random(0, 1) < $pblt$ // see Section 5.4.3 for performance of
                                                         // using different probabilities here.
14.             ExtendAndRewire$(x_{rand}, V_j, E_j)$ // extend tree with same grasp constraint.
15.         else
16.             $j \leftarrow$ ExtendAndRewire-MG$(x_{rand}, X_{reassign}, \mathbb{V}, \mathbb{E})$ // extend tree close to $x_{rand}$
                          // and with lowest heuristic cost, $j$ is the grasp index $x_{\text{new}}$ inherits.
                          // The $j$ obtained on Line 12 could be updated here.
17.             if $j ==$ NULL, break
18.     else
19.         $\{x_{rand}, j\} \leftarrow$ SampleRandom$(X_{reassign})$ // $j$ is the index of the grasp that $x_{rand}$
                                                         // belongs to
20.         ExtendAndRewire$(x_{rand}, V_j, E_j)$ // extend tree with same grasp
21.     endif
22.     $\sigma_{sol} \leftarrow$ ConnectTrees$(T_{b,j}, x_{new})$
23.     if $\sigma_{sol} \neq \emptyset$
24.         $\Sigma_{sol} \leftarrow \Sigma_{sol} \cup \sigma_{sol}$
25.     endif
26.     SwapTrees$(T_a, T_b)$
27. endfor
28. return $T = (V, E)$
```

5.4 Experiment and Results

Three sets of experiments are performed to study the behaviour of RRT*-MG. All experiments use the same 8 pipes as in Chapter 4. Examples of planned robot motion are shown in Figure 5.4 (a) - (h). The experiments are performed on an Ubuntu 20.04 machine with an Intel Core i7-4712MQ CPU @ 2.30 GHz and 8GB RAM. The program is implemented in C++. The nearest neighbour search is performed by FLANN. The sample unit ball and SVD procedure during informed sampling is done by Boost and Eigen library respectively. The collision checking is performed by FCL accessed through MoveIt API.




(c)
(d)
(e)

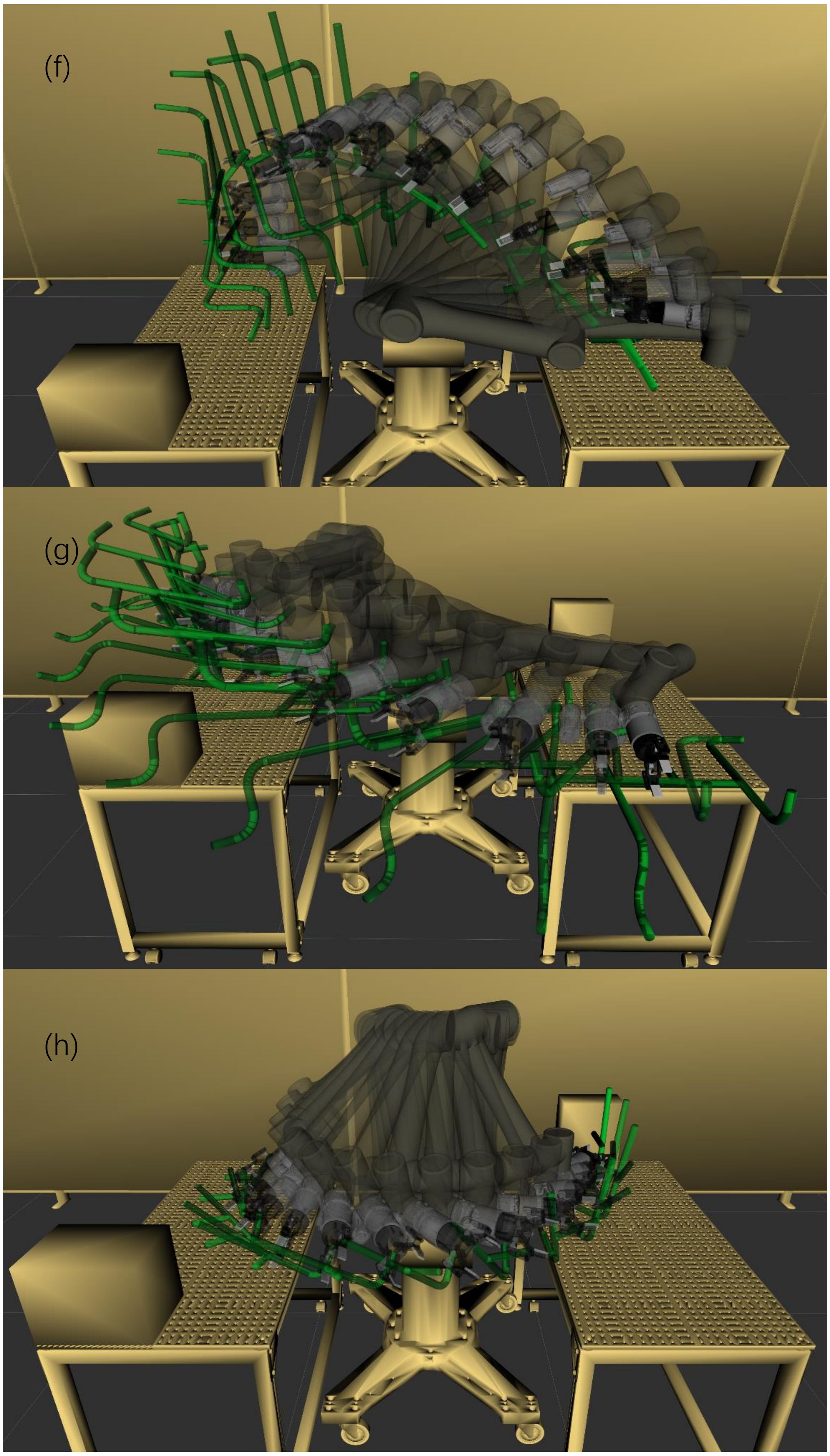
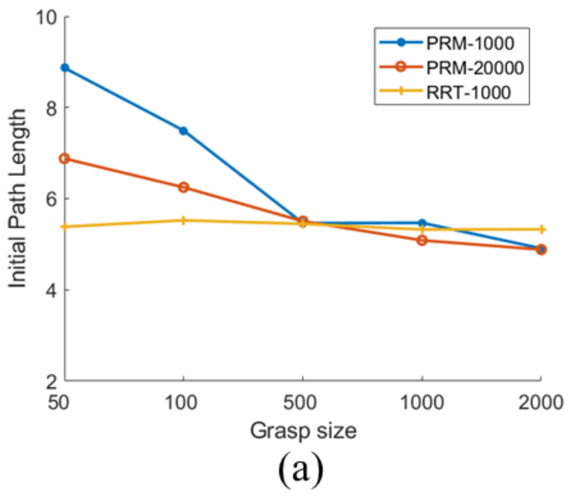


Figure 5.4 Example trajectories of Techman robot manipulating 8 different pipes.

5.4.1 Comparison between PRM*-MG and RRT*-MG

In the first experiment, the performance of PRM*-MG and RRT*-MG is compared. Five different number of grasps are tested from 50 to 2000. The minimum number of grasps is set to 50 as opposed to 25 in the experiments of Chapter 4. This is because a very small sized graph is used for PRM*-MG (1000 vertices) thus using 25 grasps with this small graph results in failure of finding a feasible solution frequently. Since PRM*-MG is not an anytime algorithm, two graphs of different sizes for PRM are tested. The first small graph as mentioned above has only 1000 vertices while the second large graph has 20000 vertices. Using a larger graph size means PRM*-MG is more likely to contain a better solution. This is indeed the case when the number of grasps is small (less than 500). Using a large grasp set makes this difference vanishes as shown in Figure 5.5 (the red and blue line of Figure 5.5 have nearly the same initial and optimal path length when using more than 500 grasps). However, RRT*-MG keeps performing better than PRM*-MG in finding the optimal solution regardless of graph size (RRT*-MG always stops after sampling 1000 vertices since this is enough to show its advantages). In terms of initial solution costs, RRT*-MG is better when only a small number of grasps is given, but slightly worse when there is a large number of grasps available as shown in Figure 5.5 (a). This is because PRM*-MG searches more grasps globally, thus yielding better solutions. However, finding such a solution also takes much more time than RRT*-MG needs (Figure 5.6 (a)). RRT*-MG also maintains the same level of performance irrespective of the number of grasps given. RRT*-MG generally takes 50ms to generate the first solution for tested problems. while PRM*-MG takes 500ms given a pre-constructed graph. RRT*-MG also finds significantly better solution after sampling 1000 vertices (Figure 5.5 (b)) although it may not be a globally asymptotically optimal algorithm as PRM*-MG is. The time taken for algorithms to find the optimal solution and the total run time is shown in Figure 5.6 (b) and (c). RRT*-MG still performs best in almost all tests with the exception of 500 grasps, where the performance gap between RRT*-MG and PRM*-MG with 1000 vertices is small. However, in this case, the solution found by RRT*-MG is much better.



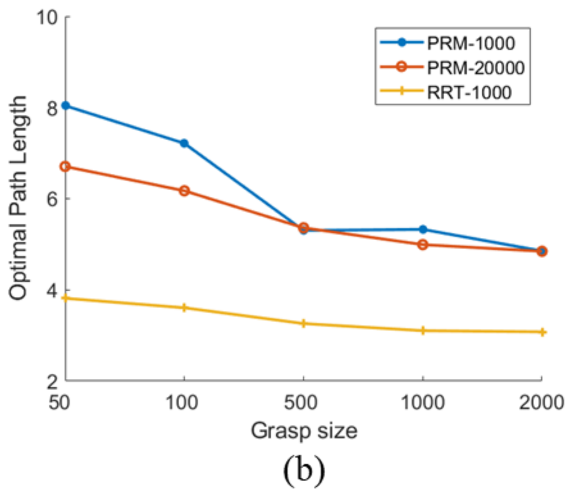
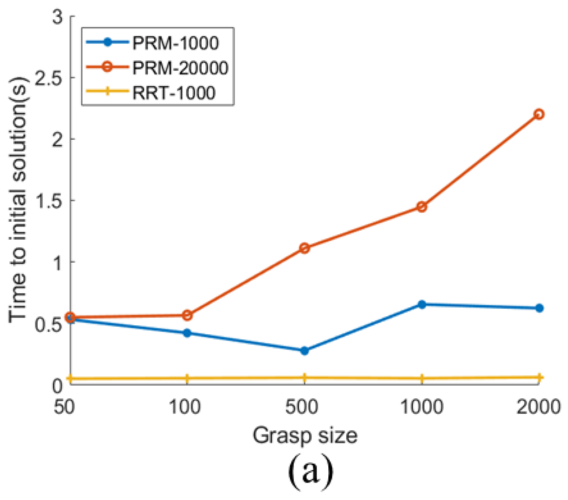
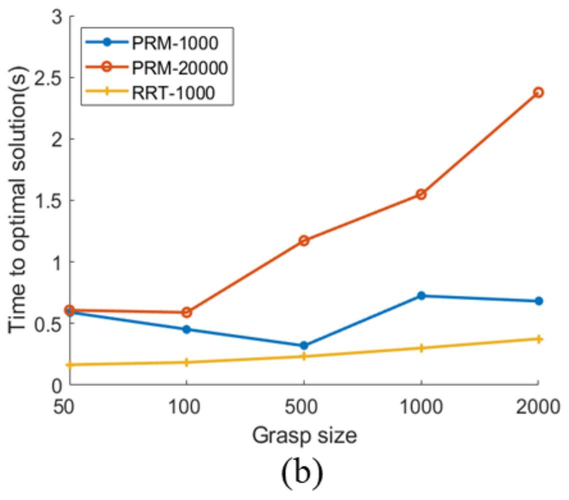


Figure 5.5 Solution cost comparison between PRM*-MG and RRT*-MG. (a) Initial path length versus grasp size. (b) Optimal path length versus grasp size.





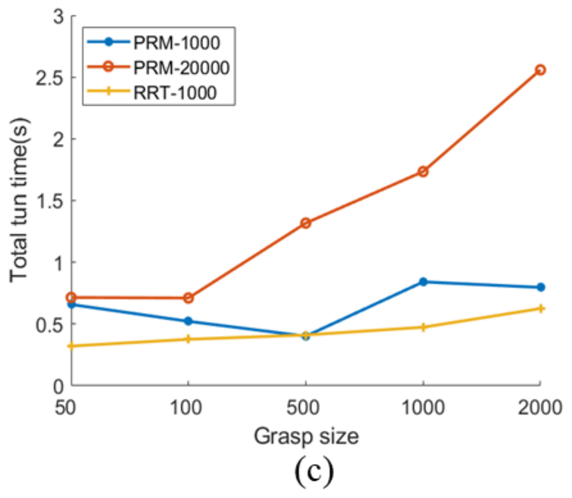


Figure 5.6 Algorithm running time comparison between PRM*-MG and RRT*-MG. (a) Time to initial solution versus grasp size. (b) Time to optimal solution versus grasp size. (c) Total run time versus grasp size.

5.4.2 Effectiveness of Informed Sampling and Bidirectional Search

In the second experiment, the effectiveness of bidirectional search and informed sampling is shown. Both implementations (V1 and V2) of the algorithms are run with 1000 samples. The first row of Table 5.1 (V1) shows of the results of the implementation without bidirectional search and informed sampling while the second row (V2) implements these two strategies. It is obvious that by using bidirectional search and informed sampling, the success rate increases significantly (from less than 70% to nearly 100%) with planning time decreased and solution costs improved.

Table 5.1 Performance comparison between algorithms on whether using bidirectional search and informed sampling.

	Success rate	Initial time (ms)	Optimal time(ms)	Initial cost	Optimal cost
V1	69.75%	2915ms	3301ms	6.29	5.80
V2	99.50%	57ms	252ms	5.40	3.37

5.4.3 Effectiveness of Balancing Exploration and Exploitation

This experiment uses 2000 grasps and 20000 samples to study the convergency property of the algorithm. The value of $pblt$ in the first column means the probability the algorithm chooses to extend X_{near} subject to same grasp constraint (Line 13 of Algorithm 5.13). If $pblt$ is 1, it means the algorithm always extend X_{near} subject to same grasp constraint, which is equivalent to selecting a grasp randomly and extend. This unbiased search is not efficient in most cases. As shown in Table 5.2, when $pblt$ is 1, the algorithm is fast, but does not find good enough solution. While using a $pblt = 0.5$ (i.e., balancing the exploitation and exploration), the algorithm achieves best solution cost with a medium planning time.

Table 5.2 Average results for 8 problems.

$pblt$	Success rate	Optimal time(ms)	Optimal cost
1	100%	2741	3.22
0.5	100%	3146	2.99
0	100%	3859	3.01

However, there is one problem in the test shows that using a large $pblt$ value is useful (Problem 8). In this problem, using $pblt = 1$ finds better solutions than a biased search with $pblt = 0$ (3.49 as opposed to 3.60) as listed in Table 5.3 Results for Problem 8. However, using $pblt = 0.5$ still finds equally good solution but takes more time to generate the results.

Table 5.3 Results for Problem 8.

$pblt$	Success rate	Optimal time(ms)	Optimal cost
1	100%	1779	3.49
0.5	100%	2809	3.49
0	100%	1213	3.60

5.5 Conclusion

In this chapter, a new single-query manipulation planning algorithm RRT*-MG is proposed. The algorithm is based on RRT* but biases the search towards the grasp that is most likely to contain the optimal path. New connection and rewiring strategies are proposed. Additional pruning and reassigning strategies are also introduced to utilise the samples more efficiently. The proposed algorithm can solve manipulation planning problem for multiple grasps in almost the same time as the single grasp manipulation planning problem. In many cases, the algorithm is even faster in generating the first solution because it detects the easiest grasp and focuses on searching it. The final version of the algorithm also incorporates ideas from RRT-connect and informed-RRT*. Experiment results show that informed sampling and bidirectional search improves the search efficiency significantly.

Chapter 6 Conclusion

This thesis investigates the impact of selecting different grasps on the subsequent robot motion and develops methods and algorithms that can select grasps and plan robot motion in an integrated manner. The thesis is motivated by problems encountered when manufacturing pipe assemblies inside a compact factory (Factory-In-A-Box). Three different scenarios are considered, and specific algorithms are developed for each scenario.

In Chapter 2, the existing work on motion and manipulation planning are reviewed. The focus is on sampling-based motion planning algorithms, which are the primary methods used in Chapter 4 and Chapter 5. Manipulation methods that combine grasp and motion planning are also reviewed and the research gaps are identified.

In Chapter 3, a predefined object path is given as input. The robot needs to select an optimal grasp to follow this given trajectory. To solve this problem, an optimisation framework is developed to optimise the grasp pose and select joint configurations (from multiple IK solutions) simultaneously. The proposed method first searches for grasp poses that satisfy workspace reachability and collision-free constraints then continuously optimise solutions based on three objectives (joint motion, deformation of the object and force manipulability). A special constraint handling method is used to decouple the constraint and objective evaluation process, allowing expensive objectives (deformation) to be evaluated only when constraints are satisfied. The method explicitly considers the kinematic redundancy of robot (i.e., multiple IK solutions for a given end-effector pose) and uses a graph search algorithm (Dijkstra's Algorithm) to find the optimal path among all feasible paths. The Bees Algorithm is used to solve the constrained optimisation problem with a proposed problem-specific local search strategy. Extensive benchmarks have been performed to evaluate the performance of 3 local search strategies and 2 other metaheuristics (GA and PSO). It is found that Bees Algorithm with the proposed local search strategy is less sensitive to the hyperparameters and can achieve consistently good performance on different problems.

In Chapter 4, the free space motion planning problem is considered given a set of feasible grasps. The robot needs to select the optimal grasp such that the post-grasp motion cost is minimized. There is no constraint on the path of the object as long as it reaches the final location. A new algorithm (PRM*-MG) based on the probabilistic roadmap (PRM) is proposed to solve the problem efficiently. The new algorithm accelerates the planning process by using: 1) a lazy strategy to avoid collision checking for suboptimal paths; 2) the current optimal cost from previous searches as an upper bound to prune the roadmap in subsequent searches. Given a constructed roadmap, the proposed algorithm can process thousands of grasps within seconds, making it suitable for planning problems without strict time requirements. The algorithm also has desirable theoretical properties such as probabilistic completeness and asymptotically optimality.

In Chapter 5, RRT*-MG is developed to solve the same free space motion planning problem given multiple feasible grasps, but in a more time-limited scenario. The algorithm needs to generate a feasible solution as fast as possible and keep improving it if more time is allowed. RRT*-MG solves the single query manipulation problem directly without reformulating it to a multi-query motion planning problem as opposed to PRM*-MG. The proposed algorithm is based on RRT* and incorporates ideas like informed sampling and bidirectional search from other literature. Besides combining ideas from existing literature, a novel connection criterion is proposed to bias the search towards grasps that are more likely to yield optimal paths. Pruning and reassigning strategies that are specific to the multiple available grasps case are also introduced for reusing the samples more efficiently. The algorithm also explicitly balances exploration and exploitation to avoid converging to local optima.

6.1 Summary of Contributions

Traditional motion planners can produce optimised motion for a robot to move from a single start configuration to a goal configuration. However, by using methods and algorithms developed in this thesis, the motion of the robot can be further optimised at the level of multiple grasps (in other words, multiple start and goal configuration pairs). Apart from proposing several methods and algorithms, perhaps a more important finding of this thesis is that planning motion for a robot to manipulate objects given multiple grasps is not a time-consuming process at all if using a specifically designed algorithm. In many cases, it is even more efficient than planning motion for a single grasp. This finding should encourage both researchers and practitioners to provide more feasible grasps for the robot to choose when planning motion for pick-and-place tasks, especially in the case of manipulating large and complex objects in a narrow environment.

Specifically, this thesis has made the following contributions:

- The developed grasp pose optimisation framework makes it possible for the robot to manipulate 3D pipe assemblies and other complex structures inside a compact factory without specially designed gripper. Experiments demonstrate the advantages of using Bees Algorithm to solve the proposed optimisation problem over other metaheuristics. A novel problem-specific local search strategy is proposed to accelerate the optimisation process (Chapter 3).
- The problem of integrated grasp selection and post-grasp optimal motion planning is formally defined. An asymptotically optimal algorithm (PRM*-MG) is developed to solve the integrated grasp selection and motion planning problem. A novel lazy collision checking strategy at the grasp level and a graph pruning procedure using an upper bound constructed in previous searches are developed. (Chapter 4)
- An anytime algorithm RRT*-MG is developed to solve the integrated grasp selection and motion planning problem under strict time limitations. A new connection strategy is developed to bias the search towards the grasp that is most likely to generate an optimal path. New rewiring, pruning, and reassigning strategies are developed to use the samples more efficiently. (Chapter 5)

6.2 Future work

Multiple potential research directions can be explored based on the work in this thesis. In Chapter 3, the grasp pose is optimised based mainly on objectives that capture the quasi-static performance of robot motion. However, if the robot needs to be operating at high speed, the dynamics of both robot and object (i.e., the deformation of the pipe when being manipulated at high speed) must be considered to enable fast, smooth, and safe motion.

In Chapter 4 and Chapter 5, the cost function used to optimise the robot motion is the path length. Although other cost functions can be considered, however, their performance would be generally worse than optimising path length since few good heuristics are available a priori. One possible solution is by incorporating ideas from advanced sampling-based motion planners like AIT* and EIT* [125], which uses an asymmetric bidirectional search to calculate problem-specific heuristics during the search.

The algorithms developed in Chapter 4 and Chapter 5 are sampling-based. Recently, many integrated motion planners have been proposed by combining sampling-based motion planners and optimisation-based motion planners for standard single start single goal planning problem as discussed in Section 2.1.4. It would be interesting to explore whether this combination can be efficiently applied to the problem studied in this thesis (e.g., integrating GOMP [96] with algorithms presented in Chapter 4 and Chapter 5) to generate smooth and possibly time-optimal trajectories without post-processing.

References

[1] B. Tipary, G. Erdős, Generic development methodology for flexible robotic pick-and-place workcells based on Digital Twin, Robot. Comput. Integr. Manuf. 71 (2021). https://doi.org/10.1016/j.rcim.2021.102140.

[2] M. Jackson, M. Wiktorsson, M. Bellgran, Factory-in-a-box — Demonstrating the next generation manufacturing provider, in: Manuf. Syst. Technol. New Front., 2008. https://doi.org/10.1007/978-1-84800-267-8_70.

[3] D. Hsu, On Finding Narrow Passages with Probabilistic Roadmap Planners, in: Robot. Algorithmic Perspect., A K Peters/CRC Press, 1998: pp. 151–164. https://doi.org/10.1201/9781439863886-18.

[4] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, J. Kuffner, Grasp planning in complex scenes, in: Proc. 2007 7th IEEE-RAS Int. Conf. Humanoid Robot. HUMANOIDS 2007, 2008. https://doi.org/10.1109/ICHR.2007.4813847.

[5] A. Kimmel, R. Shome, Z. Littlefield, K. Bekris, Fast, Anytime Motion Planning for Prehensile Manipulation in Clutter, in: 2018 IEEE-RAS 18th Int. Conf. Humanoid Robot., IEEE, 2018: pp. 1–9. https://doi.org/10.1109/HUMANOIDS.2018.8624939.

[6] T. Pardi, R. Stolkin, A.M.E. Ghalamzan, Choosing Grasps to Enable Collision-Free Post-Grasp Manipulations, in: IEEE-RAS Int. Conf. Humanoid Robot., 2019. https://doi.org/10.1109/HUMANOIDS.2018.8625027.

[7] S.M. LaValle, Planning algorithms, 2006. https://doi.org/10.1017/CBO9780511546877.

[8] T. Lozano-Pérez, Spatial Planning: A Configuration Space Approach, IEEE Trans. Comput. C–32 (1983) 108–120. https://doi.org/10.1109/TC.1983.1676196.

[9] L.E. Kavraki, P. Švestka, J.C. Latombe, M.H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, IEEE Trans. Robot. Autom. (1996). https://doi.org/10.1109/70.508439.

[10] S.M. LaValle, Rapidly-Exploring Random Trees: A New Tool for Path Planning, Comput. Sci. Dep. Iowa State Univ. (1998). https://doi.org/10.1.1.35.1853.

[11] L.E. Kavraki, M.N. Kolountzakis, J.C. Latombe, Analysis of probabilistic roadmaps for path planning, IEEE Trans. Robot. Autom. 14 (1998) 166–171. https://doi.org/10.1109/70.660866.

[12] S. Karaman, E. Frazzoli, Sampling-based algorithms for optimal motion planning, Int. J. Rob. Res. 30 (2011) 846–894. https://doi.org/10.1177/0278364911406761.

[13] R. Bohlin, L.E. Kavraki, Path planning using Lazy PRM, Proceedings-IEEE Int. Conf. Robot. Autom. 1 (2000) 521–528. https://doi.org/10.1109/ROBOT.2000.844107.

[14] M.S. Branicky, S.M. LaValle, K. Olson, L. Yang, Quasi-randomized path planning, Proc. - IEEE Int. Conf. Robot. Autom. (2001). https://doi.org/10.1109/ROBOT.2001.932820.

[15] S.M. LaValle, J.J. Kuffner, Randomized Kinodynamic Planning, Int. J. Rob. Res. 20 (2001) 378–400. https://doi.org/10.1177/02783640122067453.

[16] J.J. Kuffner, S.M. La Valle, RRT-connect: an efficient approach to single-query path

planning, Proc. - IEEE Int. Conf. Robot. Autom. 2 (2000) 995–1001. https://doi.org/10.1109/robot.2000.844730.

[17] C. Urmson, R. Simmons, Approaches for Heuristically Biasing RRT Growth, IEEE Int. Conf. Intell. Robot. Syst. 2 (2003) 1178–1183. https://doi.org/10.1109/iros.2003.1248805.

[18] D. Ferguson, A. Stentz, Anytime RRTs, IEEE Int. Conf. Intell. Robot. Syst. (2006) 5369–5375. https://doi.org/10.1109/IROS.2006.282100.

[19] K. Solovey, L. Janson, E. Schmerling, E. Frazzoli, M. Pavone, Revisiting the Asymptotic Optimality of RRT, Proc. - IEEE Int. Conf. Robot. Autom. (2020) 2189–2195. https://doi.org/10.1109/ICRA40945.2020.9196553.

[20] J. Nasir, F. Islam, U. Malik, Y. Ayaz, O. Hasan, M. Khan, M.S. Muhammad, RRT*-SMART: A rapid convergence implementation of RRT*, Int. J. Adv. Robot. Syst. 10 (2013). https://doi.org/10.5772/56718.

[21] O. Arslan, P. Tsiotras, Use of relaxation methods in sampling-based algorithms for optimal motion planning, Proc. - IEEE Int. Conf. Robot. Autom. (2013) 2421–2428. https://doi.org/10.1109/ICRA.2013.6630906.

[22] J.D. Gammell, S.S. Srinivasa, T.D. Barfoot, Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic, IEEE Int. Conf. Intell. Robot. Syst. (2014) 2997–3004. https://doi.org/10.1109/IROS.2014.6942976.

[23] J.D. Gammell, T.D. Barfoot, S.S. Srinivasa, Informed Sampling for Asymptotically Optimal Path Planning, IEEE Trans. Robot. 34 (2018) 966–984. https://doi.org/10.1109/TRO.2018.2830331.

[24] M. Jordan, A. Perez, Optimal Bidirectional Rapidly-Exploring Random Trees, Comput. Sci. Artif. Intell. Lab. (2013). https://dspace.mit.edu/bitstream/handle/1721.1/79884/MIT-CSAIL-TR-2013-021.pdf.

[25] K. Hauser, Lazy collision checking in asymptotically-optimal motion planning, in: Proc. - IEEE Int. Conf. Robot. Autom., 2015. https://doi.org/10.1109/ICRA.2015.7139603.

[26] L. Janson, E. Schmerling, A. Clark, M. Pavone, Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions, Int. J. Rob. Res. 34 (2015) 883–921. https://doi.org/10.1177/0278364915577958.

[27] J.D. Gammell, T.D. Barfoot, S.S. Srinivasa, Batch Informed Trees (BIT*): Informed asymptotically optimal anytime search, Int. J. Rob. Res. 39 (2020) 543–567. https://doi.org/10.1177/0278364919890396.

[28] N. Ratliff, M. Zucker, J. Andrew Bagnell, S. Srinivasa, CHOMP: Gradient optimization techniques for efficient motion planning, Proc. - IEEE Int. Conf. Robot. Autom. (2009) 489–494. https://doi.org/10.1109/ROBOT.2009.5152817.

[29] M. Zucker, N. Ratliff, A.D. Dragan, M. Pivtoraiko, M. Klingensmith, C.M. Dellin, J.A. Bagnell, S.S. Srinivasa, CHOMP: Covariant Hamiltonian optimization for motion planning, Int. J. Rob. Res. (2013). https://doi.org/10.1177/0278364913488805.

[30] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, S. Schaal, STOMP: Stochastic trajectory optimization for motion planning, in: 2011 IEEE Int. Conf. Robot. Autom., IEEE, 2011: pp. 4569–4574. https://doi.org/10.1109/ICRA.2011.5980280.

[31] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, P. Abbeel, Motion planning with sequential convex optimization and convex collision checking, Int. J. Rob. Res. 33 (2014) 1251–1270. https://doi.org/10.1177/0278364914528132.

[32] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, P. Abbeel, Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization, in: Robot. Sci. Syst. IX, Robotics: Science and Systems Foundation, 2013. https://doi.org/10.15607/RSS.2013.IX.031.

[33] G. Van den Bergen, A Fast and Robust GJK Implementation for Collision Detection of Convex Objects, J. Graph. Tools. 4 (1999) 7–25. https://doi.org/10.1080/10867651.1999.10487502.

[34] E.G. Gilbert, D.W. Johnson, S.S. Keerthi, A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space, IEEE J. Robot. Autom. 4 (1988) 193–203. https://doi.org/10.1109/56.2083.

[35] G. van den Bergen, Proximity queries and penetration depth computation on 3d game objects, Game Dev. Conf. (2001). https://graphics.stanford.edu/courses/cs468-01-fall/Papers/van-den-bergen.pdf.

[36] M. Mukadam, X. Yan, B. Boots, Gaussian Process Motion planning, Proc. - IEEE Int. Conf. Robot. Autom. 2016-June (2016) 9–15. https://doi.org/10.1109/ICRA.2016.7487091.

[37] M. Mukadam, J. Dong, X. Yan, F. Dellaert, B. Boots, Continuous-time Gaussian process motion planning via probabilistic inference, Int. J. Rob. Res. 37 (2018) 1319–1340. https://doi.org/10.1177/0278364918790369.

[38] M. Bhardwaj, B. Boots, M. Mukadam, Differentiable Gaussian Process Motion Planning, Proc. - IEEE Int. Conf. Robot. Autom. (2020) 10598–10604. https://doi.org/10.1109/ICRA40945.2020.9197260.

[39] C. Park, J. Pan, D. Manocha, ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments, ICAPS 2012 - Proc. 22nd Int. Conf. Autom. Plan. Sched. (2012) 207–215.

[40] Z. Marinho, B. Boots, A. Dragan, A. Byravan, G.J. Gordon, S. Srinivasa, Functional gradient motion planning in reproducing kernel hilbert spaces, Robot. Sci. Syst. 12 (2016) 1–17. https://doi.org/10.15607/rss.2016.xii.046.

[41] C. Liu, C.Y. Lin, M. Tomizuka, The convex feasible set algorithm for real time optimization in motion planning, SIAM J. Control Optim. 56 (2018) 2712–2733. https://doi.org/10.1137/16M1091460.

[42] E.W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1959) 269–271. https://doi.org/10.1007/BF01386390.

[43] P. Hart, N. Nilsson, B. Raphael, A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Trans. Syst. Sci. Cybern. 4 (1968) 100–107. https://doi.org/10.1109/TSSC.1968.300136.

[44] I. Pohl, Heuristic search viewed as path finding in a graph, Artif. Intell. 1 (1970) 193–204. https://doi.org/10.1016/0004-3702(70)90007-X.

[45] Ira Sheldon Pohl, The avoidance of (relative) catastrophe, heuristic competence, genuine

dynamic weighting and computational issues in heuristic problem solving, in: 3rd Int. Jt. Conf. Artif. Intell., 1973: pp. 12–17. https://doi.org/10.5555/1624775.1624777.

[46] S. Koenig, M. Likhachev, D. Furcy, Lifelong Planning A*, Artif. Intell. 155 (2004) 93–146. https://doi.org/10.1016/j.artint.2003.12.001.

[47] A. Stentz, Optimal and efficient path planning for partially-known environments, in: Proc. 1994 IEEE Int. Conf. Robot. Autom., IEEE Comput. Soc. Press, 1995: pp. 3310–3317. https://doi.org/10.1109/ROBOT.1994.351061.

[48] A. Stentz, The Focussed D* Algorithm for Real-Time Replanning, IJCAI Int. Jt. Conf. Artif. Intell. 95 (1995) 1652--1659.

[49] S. Koenig, M. Likhachev, D* Lite, in: Eighteenth Natl. Conf. Artif. Intell., ACM Press, New York, New York, USA, 2002: pp. 476–483. https://doi.org/10.5555/777092.777167.

[50] B.J. Cohen, S. Chitta, M. Likhachev, Search-based planning for manipulation with motion primitives, in: 2010 IEEE Int. Conf. Robot. Autom., IEEE, 2010: pp. 2902–2908. https://doi.org/10.1109/ROBOT.2010.5509685.

[51] B.J. Cohen, G. Subramania, S. Chitta, M. Likhachev, Planning for Manipulation with Adaptive Motion Primitives, in: 2011 IEEE Int. Conf. Robot. Autom., IEEE, 2011: pp. 5478–5485. https://doi.org/10.1109/ICRA.2011.5980550.

[52] B. Cohen, S. Chitta, M. Likhachev, Search-based planning for dual-arm manipulation with upright orientation constraints, in: 2012 IEEE Int. Conf. Robot. Autom., IEEE, 2012: pp. 3784–3790. https://doi.org/10.1109/ICRA.2012.6225008.

[53] B. Cohen, S. Chitta, M. Likhachev, Single- and dual-arm motion planning with heuristic search, Int. J. Rob. Res. 33 (2014) 305–320. https://doi.org/10.1177/0278364913507983.

[54] B. Cohen, M. Phillips, M. Likhachev, Planning Single-Arm Manipulations with N-Arm Robots*, Proc. 8th Annu. Symp. Comb. Search, SoCS 2015. 2015-Janua (2015) 226–227. https://doi.org/10.15607/rss.2014.x.033.

[55] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, M. Likhachev, Multi-Heuristic A*, Int. J. Rob. Res. 35 (2016) 224–243. https://doi.org/10.1177/0278364915594029.

[56] C. Park, F. Rabe, S. Sharma, C. Scheurer, U.E. Zimmermann, D. Manocha, Parallel cartesian planning in dynamic environments using constrained trajectory planning, IEEE-RAS Int. Conf. Humanoid Robot. 2015-Decem (2015) 983–990. https://doi.org/10.1109/HUMANOIDS.2015.7363489.

[57] L. Li, X. Long, M.A. Gennert, BiRRTOpt: A combined sampling and optimizing motion planner for humanoid robots, IEEE-RAS Int. Conf. Humanoid Robot. (2016) 469–476. https://doi.org/10.1109/HUMANOIDS.2016.7803317.

[58] S. Choudhury, J.D. Gammell, T.D. Barfoot, S.S.D. Srinivasa, S. Scherer, Regionally accelerated batch informed trees (RABIT*): A framework to integrate local information into optimal path planning, Proc. - IEEE Int. Conf. Robot. Autom. 2016-June (2016) 4207–4214. https://doi.org/10.1109/ICRA.2016.7487615.

[59] D. Kim, Y. Kwon, S.E. Yoon, Dancing PRM*: Simultaneous Planning of Sampling and Optimization with Configuration Free Space Approximation, Proc. - IEEE Int. Conf. Robot. Autom. (2018) 7071–7078. https://doi.org/10.1109/ICRA.2018.8463181.

[60] D. Kim, S.-E. Yoon, Simultaneous planning of sampling and optimization: study on lazy evaluation and configuration free space approximation for optimal motion planning

algorithm, Auton. Robots. 44 (2020) 165–181. https://doi.org/10.1007/s10514-019-09884-x.

[61] A. Kuntz, C. Bowen, R. Alterovitz, Fast Anytime Motion Planning in Point Clouds by Interleaving Sampling and Interior Point Optimization, Springer Proc. Adv. Robot. 10 (2020) 929–945. https://doi.org/10.1007/978-3-030-28619-4_63.

[62] S. Dai, M. Orton, S. Schaffert, A. Hofmann, B. Williams, Improving Trajectory Optimization Using a Roadmap Framework, IEEE Int. Conf. Intell. Robot. Syst. (2018) 8674–8681. https://doi.org/10.1109/IROS.2018.8594274.

[63] J. Leu, G. Zhang, L. Sun, M. Tomizuka, Efficient Robot Motion Planning via Sampling and Optimization, in: ACC, American Automatic Control Council, 2021: pp. 4196–4202. https://doi.org/10.23919/acc50511.2021.9483146.

[64] M.T. Mason, Toward Robotic Manipulation, Annu. Rev. Control. Robot. Auton. Syst. 1 (2018) 1–28. https://doi.org/10.1146/annurev-control-060117-104848.

[65] T. Siméon, J.P. Laumond, J. Cortés, A. Sahbani, Manipulation planning with probabilistic roadmaps, Int. J. Rob. Res. 23 (2004) 729–746. https://doi.org/10.1177/0278364904045471.

[66] R. Alami, T. Simeon, J.-P. Laumond, A Geometrical Approach to Planning Manipulation Tasks. The Case of Discrete Placements and Grasps, Fifth Int. Symp. Robot. Res. (1990) 453–463.

[67] R. Alami, J. Laumond, T. Siméon, R. Alami, J. Laumond, T. Siméon, Two manipulation planning algorithms, (1994).

[68] M. Stilman, J.-U. Schamburek, J. Kuffner, T. Asfour, Manipulation Planning Among Movable Obstacles, in: Proc. 2007 IEEE Int. Conf. Robot. Autom., IEEE, 2007: pp. 3327–3332. https://doi.org/10.1109/ROBOT.2007.363986.

[69] M. Stilman, J. Kuffner, Planning Among Movable Obstacles with Artificial Constraints, Int. J. Rob. Res. 27 (2008) 1295–1307. https://doi.org/10.1177/0278364908098457.

[70] W. Wan, H. Igawa, K. Harada, H. Onda, K. Nagata, N. Yamanobe, A regrasp planning component for object reorientation, Auton. Robots. (2018). https://doi.org/10.1007/s10514-018-9781-y.

[71] P.S. Schmitt, W. Neubauer, W. Feiten, K.M. Wurm, G. V. Wichert, W. Burgard, Optimal, sampling-based manipulation planning, in: Proc. - IEEE Int. Conf. Robot. Autom., 2017. https://doi.org/10.1109/ICRA.2017.7989390.

[72] A. Sahbani, S. El-Khoury, P. Bidaud, An overview of 3D object grasp synthesis algorithms, Rob. Auton. Syst. (2012). https://doi.org/10.1016/j.robot.2011.07.016.

[73] J. Bohg, A. Morales, T. Asfour, D. Kragic, Data-driven grasp synthesis-A survey, IEEE Trans. Robot. (2014). https://doi.org/10.1109/TRO.2013.2289018.

[74] J.P.C. de Souza, L.F. Rocha, P.M. Oliveira, A.P. Moreira, J. Boaventura-Cunha, Robotic grasping: from wrench space heuristics to deep learning policies, Robot. Comput. Integr. Manuf. 71 (2021) 102176. https://doi.org/10.1016/j.rcim.2021.102176.

[75] M.A. Roa, R. Suárez, Grasp quality measures: review and performance, Auton. Robots. 38 (2015) 65–88. https://doi.org/10.1007/s10514-014-9402-3.

[76] G. Liu, J. Xu, X. Wang, Z. Li, On quality functions for grasp synthesis, fixture planning, and coordinated manipulation, IEEE Trans. Autom. Sci. Eng. 1 (2004) 146–162.

https://doi.org/10.1109/TASE.2004.836760.

[77] A. ten Pas, M. Gualtieri, K. Saenko, R. Platt, Grasp Pose Detection in Point Clouds, Int. J. Rob. Res. (2017). https://doi.org/10.1177/0278364917735594.

[78] Q. Yu, W. Shang, Z. Zhao, S. Cong, Z. Li, Robotic Grasping of Unknown Objects Using Novel Multilevel Convolutional Neural Networks: From Parallel Gripper to Dexterous Hand, IEEE Trans. Autom. Sci. Eng. 18 (2021) 1730–1741. https://doi.org/10.1109/TASE.2020.3017022.

[79] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J.A. Ojea, K. Goldberg, Dex-Net 2.0: Deep learning to plan Robust grasps with synthetic point clouds and analytic grasp metrics, in: Robot. Sci. Syst., 2017. https://doi.org/10.15607/rss.2017.xiii.058.

[80] D. Morrison, P. Corke, J. Leitner, Learning robust, real-time, reactive robotic grasping, Int. J. Rob. Res. 39 (2020) 183–201. https://doi.org/10.1177/0278364919859066.

[81] G. Peng, Z. Ren, H. Wang, X. Li, M.O. Khyam, A Self-Supervised Learning-Based 6-DOF Grasp Planning Method for Manipulator, IEEE Trans. Autom. Sci. Eng. PP (2021) 1–10. https://doi.org/10.1109/TASE.2021.3128639.

[82] N. Vahrenkamp, M. Do, T. Asfour, R. Dillmann, Integrated grasp and motion planning, in: Proc. - IEEE Int. Conf. Robot. Autom., 2010. https://doi.org/10.1109/ROBOT.2010.5509377.

[83] J. Fontanals, B.A. Dang-Vu, O. Porges, J. Rosell, M.A. Roa, Integrated grasp and motion planning using independent contact regions, in: IEEE-RAS Int. Conf. Humanoid Robot., 2015. https://doi.org/10.1109/HUMANOIDS.2014.7041469.

[84] D. Berenson, S.S. Srinivasa, D. Ferguson, J.J. Kuffner, Manipulation planning on constraint manifolds, i (2009) 625–632. https://doi.org/10.1109/robot.2009.5152399.

[85] J.A. Haustein, K. Hang, D. Kragic, Integrating motion and hierarchical fingertip grasp planning, Proc. - IEEE Int. Conf. Robot. Autom. (2017) 3439–3446. https://doi.org/10.1109/ICRA.2017.7989392.

[86] M.R. Dogar, S.S. Srinivasa, A framework for push-grasping in clutter, Robot. Sci. Syst. 7 (2012) 65–72. https://doi.org/10.7551/mitpress/9481.003.0014.

[87] D. Kappler, L.Y. Chang, N.S. Pollard, T. Asfour, R. Dillmann, Templates for pre-grasp sliding interactions, Rob. Auton. Syst. (2012). https://doi.org/10.1016/j.robot.2011.07.015.

[88] L.Y. Chang, S.S. Srinivasa, N.S. Pollard, Planning pre-grasp manipulation for transport tasks, in: Proc. - IEEE Int. Conf. Robot. Autom., 2010. https://doi.org/10.1109/ROBOT.2010.5509651.

[89] J. King, M. Klingensmith, C. Dellin, M. Dogar, P. Velagapudi, N. Pollard, S. Srinivasa, Pregrasp Manipulation as Trajectory Optimization, in: Robot. Sci. Syst., 2013. https://doi.org/10.15607/rss.2013.ix.015.

[90] N. Mavrakis, A.M.E. Ghalamzan, R. Stolkin, L. Baronti, M. Kopicki, M. Castellani, Analysis of the inertia and dynamics of grasped objects, for choosing optimal grasps to enable torque-efficient post-grasp manipulations, in: IEEE-RAS Int. Conf. Humanoid Robot., 2016. https://doi.org/10.1109/HUMANOIDS.2016.7803274.

[91] N. Mavrakis, E.A.M. Ghalamzan, R. Stolkin, Safe robotic grasping: Minimum impact-force grasp selection, in: IEEE Int. Conf. Intell. Robot. Syst., 2017.

https://doi.org/10.1109/IROS.2017.8206258.

[92] E.A.M. Ghalamzan, F. Abi-Farraj, P.R. Giordano, R. Stolkin, Human-in-the-loop optimisation: Mixed initiative grasping for optimally facilitating post-grasp manipulative actions, in: 2017 IEEE/RSJ Int. Conf. Intell. Robot. Syst., IEEE, 2017: pp. 3386–3393. https://doi.org/10.1109/IROS.2017.8206178.

[93] M.B. Horowitz, J.W. Burdick, Combined grasp and manipulation planning as a trajectory optimization problem, in: Proc. - IEEE Int. Conf. Robot. Autom., 2012. https://doi.org/10.1109/ICRA.2012.6225104.

[94] S. Zimmermann, G. Hakimifard, M. Zamora, R. Poranne, S. Coros, A Multi-Level Optimization Framework for Simultaneous Grasping and Motion Planning, IEEE Robot. Autom. Lett. 5 (2020) 2966–2972. https://doi.org/10.1109/LRA.2020.2974684.

[95] T. Murooka, A.I. Karoly, F. Von Drigalski, Y. Ijiri, Simultaneous Planning of Grasp and Motion using Sample Regions and Gradient-Based Optimization, IEEE Int. Conf. Autom. Sci. Eng. 2020-Augus (2020) 1102–1109. https://doi.org/10.1109/CASE48305.2020.9217027.

[96] J. Ichnowski, M. Danielczuk, J. Xu, V. Satish, K. Goldberg, GOMP: Grasp-Optimized Motion Planning for Bin Picking, Proc. - IEEE Int. Conf. Robot. Autom. (2020) 5270–5277. https://doi.org/10.1109/ICRA40945.2020.9197548.

[97] M. Saadat, P. Nan, Industrial applications of automatic manipulation of flexible materials, Ind. Rob. (2002). https://doi.org/10.1108/01439910210440255.

[98] J. Sanchez, J.A. Corrales, B.C. Bouzgarrou, Y. Mezouar, Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey, Int. J. Rob. Res. (2018). https://doi.org/10.1177/0278364918779698.

[99] P. Jiménez, Survey on model-based manipulation planning of deformable objects, Robot. Comput. Integr. Manuf. (2012). https://doi.org/10.1016/j.rcim.2011.08.002.

[100] W. Wu, Y. Zhu, X. Zheng, Y. Guo, A novel cable-grasping planner for manipulator based on the operation surface, Robot. Comput. Integr. Manuf. 73 (2022) 102252. https://doi.org/10.1016/j.rcim.2021.102252.

[101] X. Jiang, Y. Nagaoka, K. Ishii, S. Abiko, T. Tsujita, M. Uchiyama, Robotized recognition of a wire harness utilizing tracing operation, Robot. Comput. Integr. Manuf. (2015). https://doi.org/10.1016/j.rcim.2014.12.002.

[102] J. Zhu, B. Navarro, R. Passama, P. Fraisse, A. Crosnier, A. Cherubini, Robotic Manipulation Planning for Shaping Deformable Linear Objects WithEnvironmental Contacts, IEEE Robot. Autom. Lett. (2020). https://doi.org/10.1109/LRA.2019.2944304.

[103] E. Glorieux, P. Franciosa, D. Ceglarek, Quality and productivity driven trajectory optimisation for robotic handling of compliant sheet metal parts in multi-press stamping lines, Robot. Comput. Integr. Manuf. (2019). https://doi.org/10.1016/j.rcim.2018.10.004.

[104] H. Hoffmann, M. Kohnhäuser, Strategies to optimize the part transport in crossbar transfer presses, CIRP Ann. - Manuf. Technol. (2002). https://doi.org/10.1016/S0007-8506(07)61458-9.

[105] E. Glorieux, P. Franciosa, D. Ceglarek, End-effector design optimisation and multi-robot motion planning for handling compliant parts, Struct. Multidiscip. Optim. (2018). https://doi.org/10.1007/s00158-017-1798-x.

[106] J. Pan, S. Chitta, D. Manocha, FCL: A general purpose library for collision and proximity queries, in: 2012 IEEE Int. Conf. Robot. Autom., IEEE, 2012: pp. 3859–3866. https://doi.org/10.1109/ICRA.2012.6225337.

[107] A.J.M. Ferreira, MATLAB codes for finite element analysis: Solids and structures, Solid Mech. Its Appl. (2009).

[108] K. Deb, An efficient constraint handling method for genetic algorithms, Comput. Methods Appl. Mech. Eng. (2000). https://doi.org/10.1016/S0045-7825(99)00389-8.

[109] R. Diankov, Automated Construction of Robotic Manipulation Programs, 2010. https://doi.org/isbn 9781124535470.

[110] T.L. Saaty, A scaling method for priorities in hierarchical structures, J. Math. Psychol. (1977). https://doi.org/10.1016/0022-2496(77)90033-5.

[111] R. Manseur, K.L. Doty, A Robot Manipulator With 16 Real Inverse Kinematic Solution Sets, Int. J. Rob. Res. (1989). https://doi.org/10.1177/027836498900800507.

[112] J. Meyer, descartes_planner - ROS Wiki, (n.d.). http://wiki.ros.org/descartes_planner (accessed January 11, 2022).

[113] D.T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, M. Zaidi, The Bees Algorithm - A Novel Tool for Complex Optimisation Problems, in: Intell. Prod. Mach. Syst. - 2nd I*PROMS Virtual Int. Conf. 3-14 July 2006, 2006. https://doi.org/10.1016/B978-008045157-2/50081-X.

[114] D. Goldberg, Genetic algorithms in optimization, search and machine learning, Addison Wesley. (1988).

[115] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proc. ICNN'95 - Int. Conf. Neural Networks, IEEE, 1995: pp. 1942–1948. https://doi.org/10.1109/ICNN.1995.488968.

[116] R. Shome, K. Solovey, A. Dobson, D. Halperin, K.E. Bekris, dRRT*: Scalable and informed asymptotically-optimal multi-robot motion planning, Auton. Robots. 44 (2020) 443–467. https://doi.org/10.1007/s10514-019-09832-9.

[117] A. Dobson, K.E. Bekris, A study on the finite-time near-optimality properties of sampling-based motion planners, IEEE Int. Conf. Intell. Robot. Syst. (2013) 1236–1241. https://doi.org/10.1109/IROS.2013.6696508.

[118] L. Janson, B. Ichter, M. Pavone, Deterministic sampling-based motion planning: Optimality, complexity, and performance, Int. J. Rob. Res. 37 (2018) 46–61. https://doi.org/10.1177/0278364917714338.

[119] K. Hauser, V. Ng-Thow-Hing, Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts, Proc. - IEEE Int. Conf. Robot. Autom. (2010) 2493–2498. https://doi.org/10.1109/ROBOT.2010.5509683.

[120] R. Geraerts, M.H. Overmars, Creating high-quality paths for motion planning, Int. J. Rob. Res. 26 (2007) 845–863. https://doi.org/10.1177/0278364907079280.

[121] R. Luna, I.A. Sucan, M. Moll, L.E. Kavraki, Anytime solution optimization for sampling-based motion planning, Proc. - IEEE Int. Conf. Robot. Autom. (2013) 5068–5074. https://doi.org/10.1109/ICRA.2013.6631301.

[122] A. Perez, S. Karaman, A. Shkolnik, E. Frazzoli, S. Teller, M.R. Walter, Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms,

IEEE Int. Conf. Intell. Robot. Syst. (2011) 4307–4313. https://doi.org/10.1109/IROS.2011.6048640.
[123] S. Boyd, J. Skaf, N. Moehle, Ellipsoid Method, Notes for EE364b, Stanford University (2018) 1–15.
[124] G. Wahba, A Least Squares Estimate of Satellite Attitude, SIAM Rev. 7 (1965) 409–409. https://doi.org/10.1137/1007077.
[125] M.P. Strub, J.D. Gammell, AIT* and EIT*: Asymmetric bidirectional sampling-based path planning, (2021). http://arxiv.org/abs/2111.01877.

Appendix A. Publications arising from this Thesis

The chapters of this thesis are based on the following articles that are published or under review:

- Z. Zhang and M. Saadat, "Multi-objective grasp pose optimisation for robotic 3D pipe assembly manipulation," *Robot. Comput. Integr. Manuf.*, vol. 76, no. March 2021, p. 102326, 2022.

- Z. Zhang and M. Saadat, "Integrated Grasp Selection and Post-grasp Optimal Robot Motion Planning," submitted to *IEEE Trans. Autom. Sci. Eng. (T-ASE)*, Under review.

- Z. Zhang and M. Saadat, "An anytime, sampling-based planning algorithm for manipulating objects with multiple available grasps," In preparation.