The Pennsylvania State University

The Graduate School

A HYBRID STOCHASTIC MOTION PLANNING ALGORITHM FOR SAFE AND EFFICIENT, CLOSE PROXIMITY, AUTONOMOUS SPACECRAFT MISSIONS

A Thesis in

Aerospace Engineering

by

Lawrence Joseph DiGirolamo

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2014

The thesis of Lawrence Joseph DiGirolamo was reviewed and approved* by the following:

David B. Spencer
Professor of Aerospace Engineering
Thesis Advisor

Robert G. Melton
Professor of Aerospace Engineering

George A. Lesieutre
Professor of Aerospace Engineering
Head of the Department of Aerospace Engineering

*Signatures are on file in the Graduate School

ABSTRACT

As the International Space Station and Earth orbiting satellites age well past their originally planned operational lifespan, improved monitoring of these spacecraft's integrity will be critical to the safety of any crew on board and continued functionality. A small autonomous free flying spacecraft could have the ability to monitor for structural instabilities without the need for astronaut intervention.

This thesis develops a hybrid offline motion planner that determines a fuel efficient trajectory between user specified waypoints for an inspection spacecraft, while avoiding thruster impingement with the target spacecraft. The planner requires the inspection vehicle's dynamics model, a thruster model and the obstacle field within which inspection vehicle operates. The algorithm is shown to find trajectories superior to its predecessor.

The algorithm is a merging of the Optimal Rapidly Exploring Random Tree algorithm, the Covariance Matrix Adaptation Evolutionary Strategy, and the Hill-Clohessy-Wiltshire equations. The Optimal Rapidly Exploring Random Tree algorithm is a directed tree search algorithm with a theoretical basis in random graph theory that acts as the main search framework. The Covariance Matrix Adaptation Evolutionary algorithm is used as a local optimizer which directs search to low cost solutions. Finally, the Hill-Clohessy-Wiltshire equations are used to find dynamically feasible trajectories through the search space.

Simulations are performed for flights around a simple model resembling the International Space Station, with several start and goal points, and several combinations of algorithm parameters. Results obtained from the stand-alone Optimal Rapidly Exploring Random Tree algorithm are compared to results obtained from the hybrid algorithm developed in this thesis.

The hybrid algorithm shows improved performance over the stand-alone Optimal Rapidly Exploring Random Tree algorithm. The hybrid algorithm on average is able to find trajectories

which require less propellant and less flight time, however the hybrid algorithm tends to require more computation time. In addition, planned trajectories that do not require impingement prevention require less propellant and shorter flight times. As such, cold gas thrusters may prove to be more appropriate for close proximity spacecraft missions, despite their lower efficiency as compared to their counterparts which utilize more volatile propellants. Finally, the propellant use and time of flight of a particular trajectory can be effectively tuned by the user through algorithm parameter modification.

TABLE OF CONTENTS

List of Figures vii

List of Tables x

Acknowledgements xii

Chapter 1 Introduction 1

1.1 Motivation 2
1.2 Related Work Review 4
1.3 Outline 6

Chapter 2 Algorithm Introduction and Analysis 7

2.1 Problem Definition 7
2.2 RRT* 8
2.2.1 Notation 8
2.2.2 Theory 9
2.2.3 Algorithm 16
2.3 CMA-ES 20
2.3.1 Theory 22
2.3.2 Algorithm 24
2.4 The Hill-Clohessy-Wiltshire Equations 26
2.4.1 Nonlinear Equations 26
2.4.2 Linearization 28
2.4.3 HCW Equations 28

Chapter 3 The Hill-Clohessy-Wiltshire RRT* Evolutionary Strategy Algorithm 31

3.1 RRT* Integration 31
3.2 CMA-ES Integration 34
3.2.1 CMA-ES Parameters 35
3.2.2 CMA-ES Population Requirements 36
3.3 HCW Steering Function 37
3.3.1 HCWSteer Function 38
3.3.2 Adherence to Optimality Conditions 42
3.3.3 Impulsive Maneuver Accuracy 44
3.4 Impingement Prevention 45

Chapter 4 Simulations 46

4.1 International Space Station Model 46
4.2 Free-Flying Vehicle Model 47
4.3 Experimental Setup 48

Chapter 5 Results and Discussion 55

5.1 Graphical Results 55
5.2 Data Tables 59
5.3 Cost Comparison 61
5.4 Convergence Index 68
5.5 Total ΔV 70
5.6 Time of Flight 74
5.7 Computational Run Time 76

Chapter 6 Conclusions and Future Work 80

6.1 Conclusions 80
6.2 Future Work 82

Appendix A CMA-ES Strategy Parameters 85

Appendix B RRT* 86

Appendix C RRT*-ES 88

Appendix D Data Figures 91

Bibliography 117

LIST OF FIGURES

Figure 1-1 AERCam Sprint in operation (courtesy NASA) 3

Figure 2-1 A directed graph, *G* 9

Figure 2-2 The $\boldsymbol{\delta}$-interior of $\boldsymbol{Xfree}$ [17] 10

Figure 2-3 Path $\boldsymbol{\delta}$-clearance [17] 12

Figure 2-4 A path which does not exhibit weak $\boldsymbol{\delta}$-clearance [17] 12

Figure 2-5 The $\boldsymbol{\epsilon}$-ball centered at $\boldsymbol{z}$ 14

Figure 2-6 Weakened local controllability at $\boldsymbol{z}$ 15

Figure 2-7 A $\boldsymbol{\epsilon}$-collision-free approximate trajectory [16] 16

Figure 2-8 RRT* procedure 19

Figure 2-9 CMA-ES evolution over six generations (Wikimedia Commons) 21

Figure 2-10 HCW reference frame [6] 27

Figure 3-1 GoalConnect and GoalRewire procedure 34

Figure 3-2 Control input bound considerations 39

Figure 3-3 HCWSteerRewire control input bound considerations 42

Figure 4-1 ISS MATLAB model 47

Figure 4-2 Vehicle thruster configuration [2] 48

Figure 4-3 Goal and start state combination A 50

Figure 4-4 Goal and start state combination B 51

Figure 4-5 Goal and start state combination C 51

Figure 5-1 Search characteristics of HCW RRT* and HCW CMA-ES RRT* 57

Figure 5-2 Optimal trajectory with impingement prevention 58

Figure 5-3 Optimal trajectory without impingement prevention 59

Figure 5-4 Run sets with Impingement Prevention off vs. Impingement Prevention on 64

Figure 5-5 Cost vs. number of iterations, impingement prevention on 67

Figure 5-6 Cost vs. number of iterations, impingement prevention off 67

Figure 5-7 Simulation B with Impingement Prevention isometric and side view 72

Figure 5-8 Simulation B run comparison based on scaling factor 73

Figure B-1 RRT* pseudocode 87

Figure C-1 RRT*-ES pseudocode 89

Figure C-2 GoallConnect pseudocode 90

Figure C-3 GoalRewire pseudocode 90

Figure D-1 Run set 1 93

Figure D-2 Run set 1 94

Figure D-3 Run set 3 95

Figure D-4 Run set 4 96

Figure D-5 Run set 5 97

Figure D-6 Run set 6 98

Figure D-7 Run set 7 99

Figure D-8 Run set 8 100

Figure D-9 Run set 9 101

Figure D-10 Run set 10 102

Figure D-11 Run set 11 103

Figure D-12 Run set 12 104

Figure D-13 Run set 13 105

Figure D-14 Run set 14 106

Figure D-15 Run set 15 107

Figure D-16 Run set 16 108

Figure D-17 Run set 17 109

Figure D-18 Run set 18 110

Figure D-19 Run set 19 111

Figure D-20 Run set 20 112

Figure D-21 Run set 21 113

Figure D-22 Run set 22 114

Figure D-23 Run set 23 115

Figure D-24 Run set 24 116

LIST OF TABLES

Table 4-1. Simulation start and goal states 50

Table 4-2. Simulation bounds 52

Table 4-3. NearVertices scale factors 53

Table 5-1. Example average table 60

Table 5-2. Example ANOVA table 61

Table 5-3. Average cost and 95% confidence interval (A) 62

Table 5-4. Average cost and 95% confidence interval (B) 62

Table 5-5. Average cost and 95% confidence interval (C) 62

Table 5-6. ANOVA test on cost and 95% confidence interval 62

Table 5-7. Average cost coefficient of variation (A) 65

Table 5-8. Average cost coefficient of variation (B) 65

Table 5-9. Average cost coefficient of variation (C) 66

Table 5-10. Average convergence index and 95% confidence interval (A) 69

Table 5-11. Average convergence index and 95% confidence interval (B) 69

Table 5-12. Average convergence index and 95% confidence interval (C) 69

Table 5-13. ANOVA test on convergence index and 95% confidence interval 69

Table 5-14. ANOVA test on ΔV and 95% confidence interval 70

Table 5-15. Average ΔV and 95% confidence interval (A) 71

Table 5-16. Average ΔV and 95% confidence interval (B) 71

Table 5-17. Average ΔV and 95% confidence interval (C) 71

Table 5-18. Average TOF(s) and 95% confidence interval (A) 75

Table 5-19. Average TOF(s) and 95% confidence interval (B) 75

Table 5-20. Average TOF(s) and 95% confidence interval (C) 76

Table 5-21. ANOVA test on TOF(s) and 95% confidence interval 76

Table 5-22. Average run time (s) and 95% confidence interval (A) 78

Table 5-23. Average run time (s) and 95% confidence interval (B) 78

Table 5-24. Average run time (s) and 95% confidence interval (C) 78

Table 5-25. ANOVA test on run time (s) and 95% confidence interval 79

Table A-1 CMA-ES strategy parameters 85

Table D-1 Run set number and associated parameter settings 92

ACKNOWLEDGEMENTS

This thesis would not have been possible without the help and support of many individuals. It has been quite the exploratory learning experience, which without proper guidance at crucial times could have slipped into the realm of trivial meandering.

My parents have always been crucial to my education. My family's constant encouragement and enthusiasm about schooling has instilled in me a passion for learning. I would like to thank my mom in particular for removing my undergraduate financial burden, allowing me to pursue a graduate degree with initially uncertain funding. Without this help, my prospects at this time would be considerably different.

My advisor Dr. David Spencer has been integral to many of my achievements over the last couple of years. From helping me achieve an early acceptance to Penn State's graduate program which directly lead to my assistantship at ARL, to pushing me to present my work at the AIAA Space Conference; his help and guidance has been much appreciated. Critically, he encouraged me to take a quite daunting Evolutionary Computation class my first semester of grad school. Although I may have questioned the idea at the time, the knowledge I gained in that class became a foundation for this work.

I would like to thank Dr. Andrew Hoskins and Dr. Kurt Hacker for taking the chance on hiring a fairly inexperienced Aerospace Engineering student, with a focus on space applications, as a research assistant for undersea applications. Although I may have made a different decision in their position, I am happy that it worked out so well. My growth as an engineer over the last few years can largely be attributed to their guidance and confidence in my abilities. They struck a great balance between redirecting me when necessary, while allowing me do my own stumbling through the material that makes up this thesis. This lead to an exceptional learning experience. I

would like to also extend this thank you to everyone I have had the pleasure of working with at ARL. It has been a terrific opportunity.

I would like to thank Dr. Robert Melton for reviewing this thesis, and for reinforcing my excitement for Aerospace engineering throughout undergraduate and graduate courses. He was also great company on the conference trip this summer, however next time we are in San Diego we will have to search out better fish tacos.

Last but certainly not least, I would like to thank my friends, and particularly Giulia for always being there to give me a much needed break from engineering mode. Thank you for your patience and excitement in listening to my rambling explanations of my research, and for your constant support. Although I have often found myself working late into the night, you were always up later in your studio or apartment, and happy to have a visitor. For that I am grateful.

Chapter 1

Introduction

Autonomous systems have the potential to provide currently unrealizable capabilities while decreasing cost and human risk. This is particularly true for inspection operations of spacecraft. This thesis presents work on an offline motion planning framework for International Space Station and satellite monitoring missions in Circular Low Earth Orbit. The framework uses a variation of the Optimal Rapidly Exploring Random Tree algorithm (RRT*)[17] and the Covariance Matrix Adaptation Evolutionary Strategy algorithm (CMA-ES)[12] as its basis. The goal for the missions is to provide sensor coverage of user designated areas of a spacecraft with no need for human-in-the-loop vehicle control.

At the core of this work is an algorithm that plans a trajectory between user specified six degree of freedom waypoints for the purpose of inspection given the autonomous vehicle's dynamics model, a thruster model, and an obstacle field where operations are taking place. In the context of the current effort, obstacles consist of either the International Space Station or some other satellite in a near-circular orbit about Earth, and the inspection vehicle's trajectory adheres to the Hill-Clohessy-Wiltshire (HCW) equations of motion during planning. In addition, the planner takes into consideration the effect of rocket plumes, trajectory time of flight, as well as propellant use.

A hybrid algorithm has been developed for this purpose by combining the RRT* search algorithm with a constrained input, fixed final state steering function, and the CMA-ES algorithm. RRT* is an incremental tree search algorithm that expands stochastically from an initial point and guarantees asymptotic optimality for any system with controllable linear dynamics. It is also able to maintain a computational complexity within a constant factor of its

non-asymptotically optimal predecessor RRT[17, 19]. The steering function used solves the control problem according to the Hill-Clohessy-Wiltshire equations assuming the spacecraft can perform an impulsive change of velocity. The steering function is used to connect nodes within the RRT* search tree. CMA-ES is a stochastic method for real-parameter optimization of non-linear, non-convex functions[14], which is used as a local optimizer in this work.

The Hill-Clohessy-Wiltshire equations for relative motion of two objects in a circular orbit are used within the steering function to plan dynamically feasible optimal trajectories. The use of these equations increases search complexity, but also helps to ensure that the trajectory found by the offline planner is indeed feasible for the spacecraft to execute.

Given that the free flying inspection craft will likely need to use chemical rockets to maneuver in close proximity to delicate satellite and International Space Station equipment, the effect of the inspection craft thruster plume must be taken into consideration. To avoid damage to target craft equipment, a thruster model which designates the plume length of a firing thruster as a keep out zone is used. This length is then fed into the planner's collision detection algorithm and a trajectory that results in plume impingement is deemed infeasible.

Finally, with all constraints on the inspection vehicle satisfied, the planner minimizes two trajectory components. First, propellant use on the trajectory is minimized so as to extend its mission duration, and second, the time of flight is minimized so as to maintain realistic mission scenarios. The user may specify the desired relationship between these two objectives.

1.1 Motivation

Unlike previous large-scale engineering endeavors whose operational lives exceeded their original engineered life, spacecraft lack the benefit of thorough hands-on inspection.

Spacewalks are both costly and dangerous, robotic arms have limited range and maneuverability, and complex structural engineering calculations cannot always predict the true state of a system. As the capabilities of autonomous systems continue to advance, these systems can fill the health monitoring and repair role which is currently unmet for most space assets.

NASA's operational AERCam Sprint (Figure 1-1), along with its proposed successor, Mini AERCam, strongly motivated the work in this thesis. Both vehicles were designed to be free flying spacecraft for remote inspection of manned spacecraft. Although AERCam sprint was successfully flown in 1997, it required remote operation by an astronaut for the entire duration of its flight[2].

Figure 1-1 AERCam Sprint in operation (courtesy NASA)

This work helps to increase the capabilities of free flyer missions by removing the astronaut from the mission requirements. Autonomous motion planning not only frees up astronauts to perform other tasks, but also enhances the planned trajectories by minimizing time of flight and fuel use while maximizing safety through obstacle avoidance and impingement

prevention. Given the complex and non-intuitive dynamic environment of relative motion in a circular low Earth orbit, a human pilot is not as capable of optimizing these flight parameters.

1.2 Related Work Review

To date, research into the area of close proximity spacecraft operations has produced some impressive results. Richards et. al.[22] developed an optimal trajectory planner for close proximity spacecraft operations using mixed integer linear program. Their formulation adheres to the Hill-Clohessy-Wiltshire equations and finds a minimum fuel path between two poses which avoids obstacles and satisfies rocket plume constraints using bang-off-bang control. Although this method is promising and appears to be computationally tractable, it requires an a priori knowledge of the time required to complete the path. This inherently limits the optimality of the given path, and if the time is set incorrectly, could result in a dynamically infeasible path.

McInnes[20] developed a planner which uses ellipse of safety transfers and potential field method to plan a safe obstacle free path for a spacecraft performing close proximity maneuvers with the ISS. The ellipse of safety transfers developed are a novel way to limit the risk to the ISS in the case of thruster failure, but are only useful in getting the spacecraft close to the desired observation point. From there a potential field method is used to plan the rest of the trajectory. The method makes no mention of path optimality however, and potential field methods are often plagued with local minima and oscillatory problems especially in the presence of complex obstacle fields[18].

Breger and How[3] developed an optimal planner that guarantees safe trajectories in the presence of a guidance and control failure using mixed integer linear programming. Although the safety guarantee provided is very important to close proximity missions, the formulation once

again require an a priori flight time. This leads to limited optimality or the need to run an already computationally intensive algorithm many times to find the optimal flight time.

Outside of the algorithms mentioned above there are several grid search based methods that show promise for path planning and obstacle avoidance. They are members of the A*[21], and D*[24] family of algorithms. Both algorithms perform grid searches and rely on a cost function and a heuristic for estimating the cost to arrive to the goal from their current state. A* assumes a prior knowledge of the environment, while D* is designed to be an efficient replanner when the environment changes. The problems that arise with these algorithms are a function of the heuristic required and their grid search basis. Although in low dimension grid search makes these algorithms fast and efficient planners, when dimensions start to lose a lot of their computational efficiency. In addition, it is difficult to capture vehicle dynamics with a grid search. As such, these planners excel when the obstacle field is relatively sparse and the distances traveled are relatively large. When distances are small, it is difficult to ensure that the path being planned is actually achievable by the vehicle. Finally, the effectiveness of the planner relies heavily on the heuristic used to estimate the cost from the current state to the goal state. In the case of an orbital mission in the presence of large obstacles as presented in this thesis, it becomes difficult to develop an accurate heuristic function.

The algorithm formulation posed in this work attempts to address some of the drawbacks of the aforementioned algorithms. The algorithm finds an optimized obstacle free trajectory between a given start and end vehicle state with no requirements for an a priori flight time. The optimized trajectory found avoids plume impingement on the target vehicle and adheres to the dynamics of the orbital system. In addition, a sub-optimal obstacle free trajectory is first found, and then improved as computation time allows, making this an anytime algorithm.

1.3 Outline

The contributions made in this thesis are presented over several chapters. Chapter 2 introduces the algorithmic components that make up the motion planner, along with the theoretical support for their functionality. These components include the RRT* algorithm, the CMA-ES algorithm, and the Hill-Clohessy-Wiltshire equations. Chapter 3 details the integration of the component algorithms into the hybrid algorithm, along with modifications made to the component algorithms to address the orbital relative motion trajectory planning problem. Chapter 3 also discusses the adherence of the new hybrid algorithm to RRT* asymptotic optimality requirements, and vehicle dynamic constraints. Chapter 4 introduces the simulation environment developed to test the hybrid algorithm. Chapter 5 presents the simulation results, and analysis. Finally, Chapter 6 makes conclusions based on these results, and provides suggestions for future work.

Chapter 2

Algorithm Introduction and Analysis

The motion planning algorithm presented in this thesis derives its capabilities from three component algorithms: RRT*, CMA-ES, and a Hill-Clohessy-Wiltshire steering function. The following sections describe the motion planning problem in more rigorous mathematical terms, followed by the theory and functionality of the hybrid algorithm components.

2.1 Problem Definition

Prior to introducing the algorithms utilized to solve the motion planning problem, a more formal definition of the motion planning problem is required. Let $X \subseteq \mathbb{R}^n$define the state space and $U \subseteq \mathbb{R}^m$ the control input space of the system. There is a given obstacle region $X_{obs} \subset X$, start state $\vec{x}_{start}$ and goal state $\vec{x}_{goal}$, and system dynamics

$$\dot{\vec{x}} = f(\vec{x}(t), \vec{u}(t), t) \tag{1}$$

where , $\vec{x}(t) \in X$ is the state of the vehicle at time t, and $\vec{u}(t) \in U$ is the required input at time t.

The goal of the optimal motion planning problem is to find the trajectory $\pi = [\vec{x}(\quad), \vec{u}(\quad), T]$, which connects the start state and goal state while remaining in the obstacle free region $X_{free} = (X \backslash X_{obs})$, adhering to the system dynamics, and optimizing the cost function:

$$c(\pi) = \int_0^T \left(1 + \vec{R}\|\vec{u}(t)\|\right) dt \tag{2}$$

where T is the trajectory duration, and $\vec{R} \in \mathbb{R}^m$ is positive-definite, constant, and given, and weights the cost of the control inputs relative to each other and to the duration of the trajectory. The cost function penalizes for both the amount of control input used as well as the time required to complete the trajectory.

2.2 RRT*

The RRT* algorithm was first proposed by Karaman and Frazzoli[17] as a provably asymptotically optimal, probabilistically complete, and computationally efficient motion planning algorithm. It is a tree search algorithm that lends itself to motion planning problems with differential constraints[17]. Extensions for differential constraints were added to RRT* in [8], [16], and [25]. These extensions build on Karaman and Frazzoli's original work, providing conditions for asymptotic optimality and probabilistic completeness for kinodynamic systems in [17] and [11], and extending RRT* to include a fixed-final-state-free-final-time optimal controller in [25]. In this section the theory behind the asymptotic optimality and probabilistic completeness of the RRT* algorithm is presented, along with the algorithm itself.

2.2.1 Notation

The notation used here in relation to RRT* is analogous to the notation used in the original RRT* paper [17]. A directed graph $G = (V, E)$ on X consists of a set of vertices V which is a finite subset of X, and a set of edges E which is a subset of VxV. A directed path, σ_n, on the directed graph G is a sequence of vertices $(z_1, z_2, \dots, z_n)$ such that $x_{i+1} = (z_i, z_{i+1}) \in E$ for all $1 \leq i \leq n-1$ and Σ is the set of all paths through the space. An example directed graph can be seen in Figure 2-1. RRT* is a directed tree, which is a type of directed graph where every vertex has one unique incoming parent vertex, except for the start vertex which has no parent. A vertex in RRT* is equivalent to a state of the system and is also called a node. For clarity, in this work a vertex will refer to a state that has already been added to the search tree, and a node will refer to a state that is a candidate to be added to the search tree. An edge in RRT* is a path or trajectory that connects two vertices.

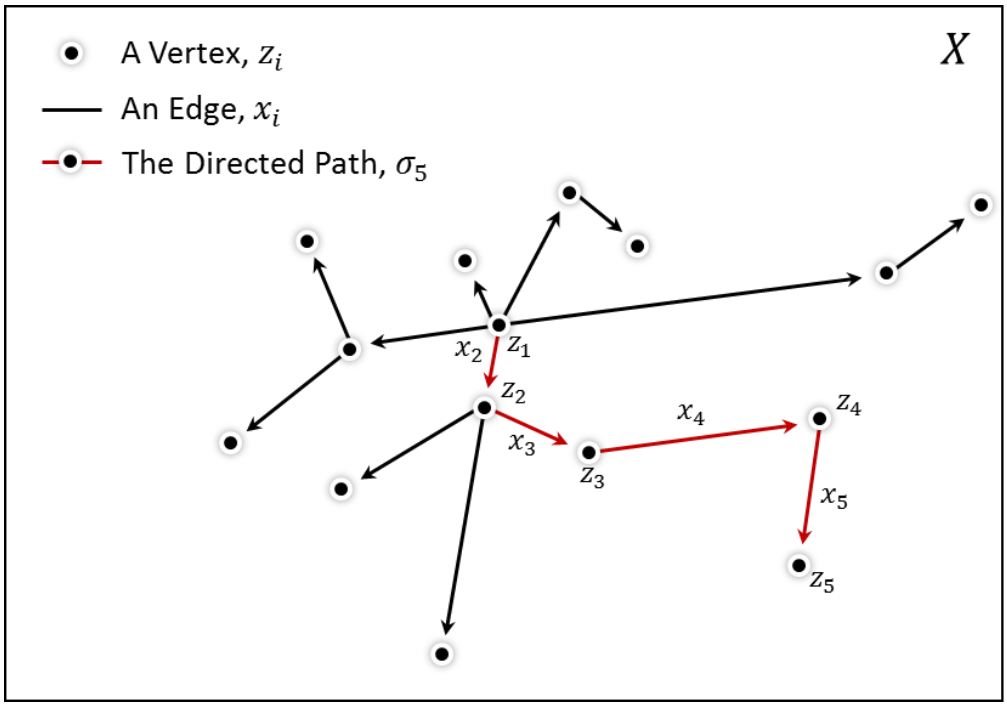


Figure 2-1 A directed graph, *G*

2.2.2 Theory

There are two main characteristics of RRT* and its kinodynamic variants that will be presented in this section. RRT* has been shown to be both probabilistically complete and asymptotically optimal[17]. For a probabilistically complete algorithm, the probability of finding a solution to the problem if one exists goes to one as the number of iterations of the algorithm go to infinity. Asymptotic optimality refers to the ability of the cost of the trajectory returned to almost surely converge to the optimum as iterations go to infinity[17].

2.2.2.1 Probabilistic Completeness

Before a formal definition of probabilistic completeness can be presented, the definition of robust feasibility must be understood. The following definitions are taken from Karman and Fazzoli's work in reference [17].

For a state $z \in X_{free}$ and some real valued $\delta > 0$, the δ-interior of X_{free} is the set of all states that are at least a distance δ away from any point in the obstacle set. In Figure 2-2, z_1 is in the δ-interior of X_{free}, and z_2 is not in the δ-interior. A collision-free path has strong δ-clearance if the entirety of the path lies within the δ-interior of X_{free} (Figure 2-3). Finally, a path planning problem $(X_{free}, z_{start}, z_{goal})$ is robustly feasible if there exists a solution path with strong δ-clearance.

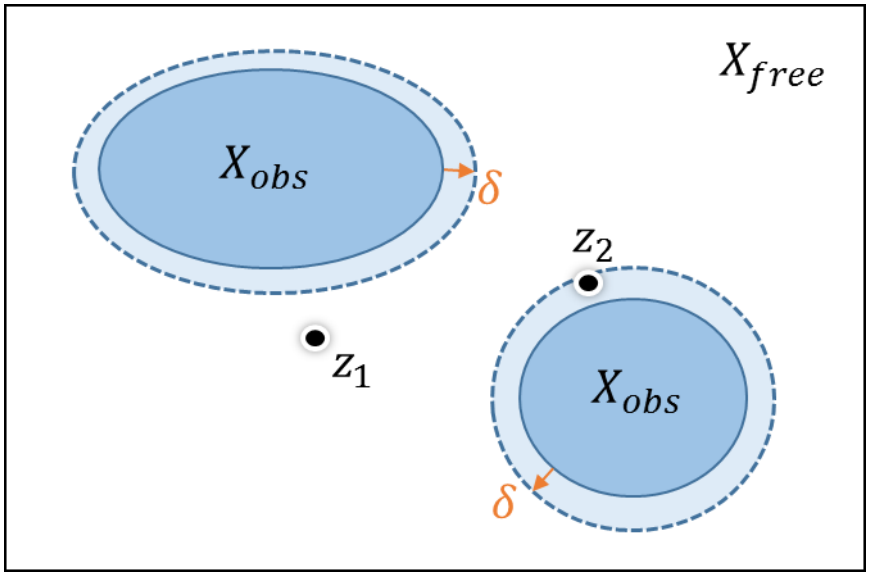


Figure 2-2 The δ-interior of X_{free} [17]

A formal definition of probabilistic completeness as presented by Karaman and Frazzoli[17] proceeds as follows:

> *An algorithm ALG is probabilistically complete, if, for any robustly feasible path planning problem* $(X_{free}, z_{start}, z_{goal})$,

$$\liminf_{n\to\infty} \mathbb{P}\left(\left\{\exists z \in V_n^{ALG} = z_{goal}\ s.t.\ z_{start}\ is\ connected\ to\ z_{goal} in\ G_n^{ALG}\right\}\right) = 1 \quad (3)$$

That is, the probability that a vertex in the algorithm's search tree is the goal vertex, and the search graph connects the start vertex and the goal vertex goes to one as the number of vertices in the graph goes to infinity. However, this limit is equal to zero if the problem is not robustly feasible for any sampling based algorithm, including probabilistically complete ones[17].

The proof of the probabilistic completeness of the RRT algorithm was developed by Lavalle and Kuffner in [19]. The conditions are extended to RRT* by Karman and Frazzoli in [17], where:

> *For any robustly feasible path planning problem* $(X_{free}, z_{start}, z_{goal})$*, there exist constants* $a > 0$ *and* $n_0 \in \mathbb{N}$*, both dependent only on* X_{free} *and* z_{goal} *such that*
>
> $$\mathbb{P}\left(\left\{\exists z \in V_n^{RRT*} = z_{goal}\right\}\right) > 1 - e^{-an},\ \forall n > n_0 \quad (4)$$

That is, the probability that a vertex in the RRT* search graph is the goal vertex approaches one as the number of vertices in the search graph goes to infinity.

2.2.2.2 Asymptotic Optimality

Again, prior to providing a formal definition of asymptotic optimality and sufficient conditions to guarantee asymptotic optimality of RRT*, the definition of a robustly optimal solution must be understood. The following definitions are taken from Karman and Fazzoli's work in [17].

Let $\sigma_1, \sigma_2 \in \Sigma_{free}$ be two collision-free paths with the same end points. A path σ_1 is considered to be homotopic to σ_2 if it can be continuously transformed to σ_2 through X_{free}. In Figure 2-3, path σ_1 is homotopic to σ_2. A collision free path σ is said to have weak δ-clearance if it is homotopic to a path σ' which exhibits strong δ-clearance, and there exists $\delta_\alpha > 0$ such that all paths along the transform from σ to σ' exhibit strong δ_α-clearance. In Figure 2-3, σ_2 exhibits

strong δ-clearance, while σ_1 exhibits weak δ-clearance. A path that does not exhibit weak δ-clearance can be seen in Figure 2-4.

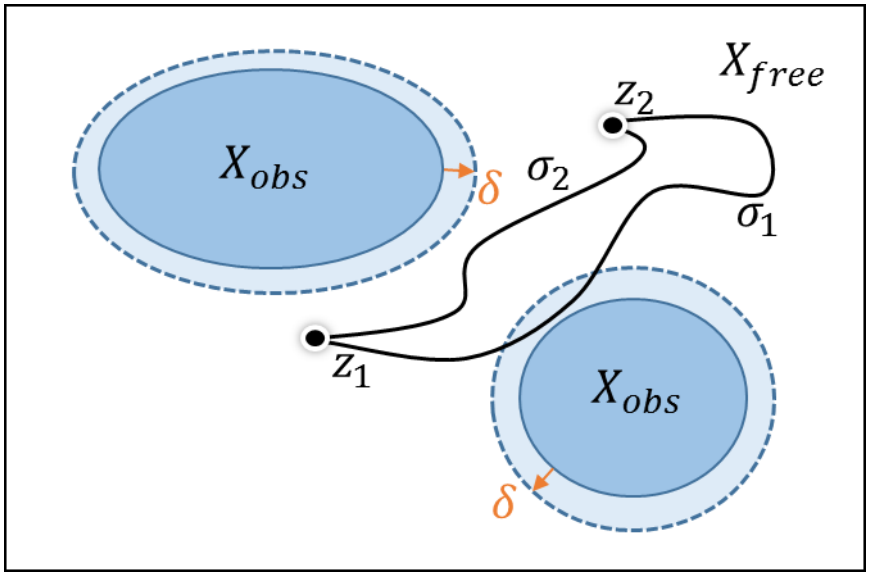


Figure 2-3 Path δ-clearance [17]

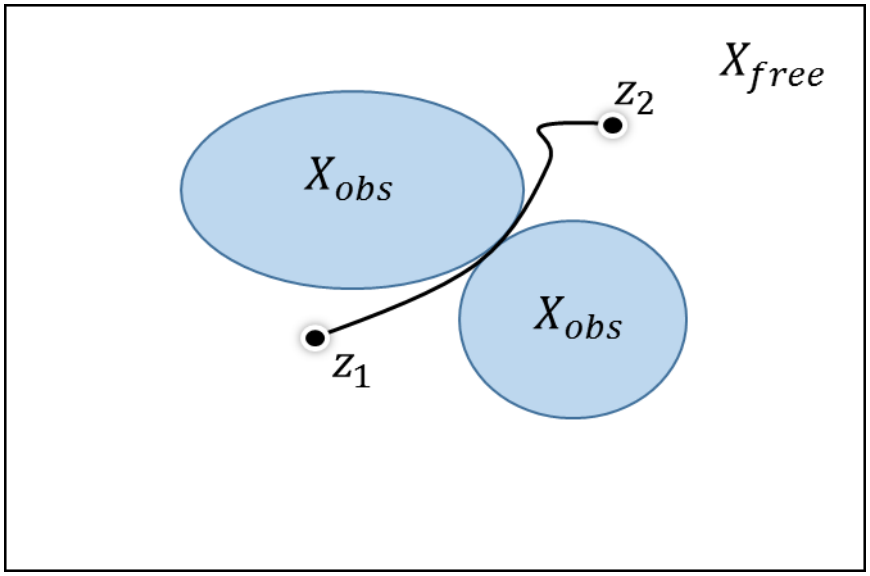


Figure 2-4 A path which does not exhibit weak δ-clearance [17]

Now given that the set of all paths with bounded length is a normal space, which allows us to take the limit of a sequence of a path[17], the definition of a robustly optimal solution is as

follows. A minimum cost feasible path $\sigma^* \in X_{free}$is a robustly optimal solution if it is has weak δ-clearance and, for any sequence of collision-free paths $\{\sigma_n\}_{n\in\mathbb{N}}$, $\sigma_n \in X_{free}$, $\forall n \in \mathbb{N}$, such that $\lim_{n\to\infty} \sigma_n = \sigma^*$, and $\lim_{n\to\infty} c(\sigma_n) = c(\sigma^*)$.[17] This means that the path is optimal if the sequence of collision free paths converges to it and that this optimal path is surrounded by enough collision free space, so that convergence may occur. The definition of asymptotic optimality is then:

> *An algorithm ALG is asymptotically optimal, if, for any robustly feasible path planning problem* $(X_{free}, x_{start}, x_{goal})$ *and cost function* $c: \Sigma \to \mathbb{R}_{\geq 0}$ *that admit a robustly optimal solution with finite cost c*,*
>
> $$\mathbb{P}\left(\left\{\lim_{n\to\infty} sup\, Y_n^{ALG} = c^*\right\}\right) = 0 \quad (5)$$

where Y_n^{ALG} is the variable that stores the lowest cost solution found by ALG after *n* iterations. It should be noted that if an algorithm exhibits asymptotic optimality, it is required to also exhibit probabilistic completeness. For this reason, necessary conditions for only asymptotic optimality are presented in the following paragraphs for kinodynamic systems.

Karaman and Frazzoli extend their RRT* algorithm to kinodynamic systems and give conditions for asymptotic optimality in reference [16]. There are three conditions presented in this work. First, the `Steer` procedure must connect nodes in a locally optimal manner. In addition, the cost procedure to determine the near and nearest nodes must correspond to the optimal cost to reach those nodes.

The second condition is Weakened Local Controllability. Prior to defining Weakened Local Controllability, there is some additional notation which must be explained. Let $\beta_\epsilon(z) = \{z' \in X | \|z' - z\| \leq \epsilon\}$ be the closed ϵ-ball centered at z. Now, given a state $z \in X$ and a constant $\epsilon \in \mathbb{R}_{>0}$, let $\mathcal{R}_\epsilon(z)$ be the set of all states in *X* that are reachable from z with a trajectory *x* that does not leave the ϵ-ball centered at z. $\mathcal{R}_\epsilon(z)$ is referred to as the ϵ–reachable set of a state z, and any state within $\mathcal{R}_\epsilon(z)$ is referred to as ϵ–reachable. Figure 2-5 depicts the ϵ-ball centered at z, as

well as a state z' which is a member of $\mathcal{R}_\epsilon(z)$. Weakened Local Controllability can now be defined as follows (Figure 2-6):

> *There exist constants $\alpha, \bar{\epsilon} \in \mathbb{R}_{>0}, p \in \mathbb{N}$, such that for any $\epsilon \in (0, \bar{\epsilon})$, and any state $z \in X$, the set $\mathcal{R}_\epsilon(z)$ of all states that can be reached from z with a path that lies entirely inside the ϵ-ball centered at z, contains a ball of radius $\alpha\epsilon^p$.*[16]

Weakened local controllability holds true for locally controllable systems, which includes controllable linear systems.

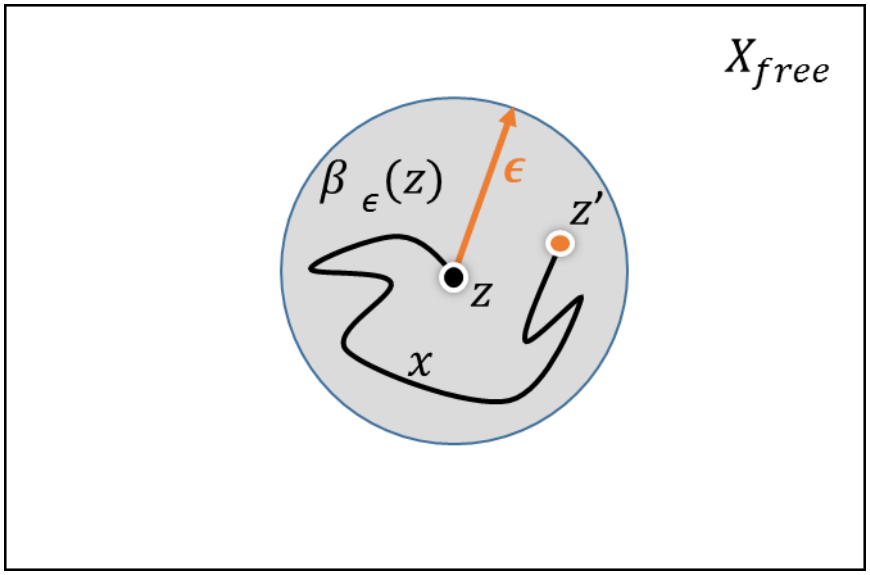


Figure 2-5 The ϵ-ball centered at z

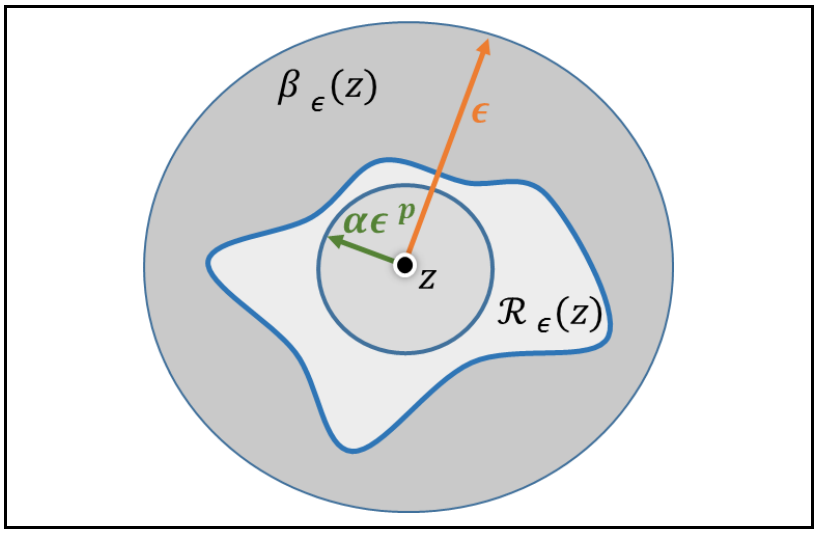


Figure 2-6 Weakened local controllability at *z*

The third condition for asymptotic optimality is that there exists an optimal trajectory with enough space around it to allow almost-sure convergence[16]. This is referred to as a ϵ-collision-free approximate trajectory. A trajectory is considered to be ϵ-collision-free, if the ϵ-ball around every state along the trajectory is within X_{free}. This is analogous to a path that exhibits strong δ-clearance where the necessary separation distance is centered at each state along the trajectory, rather than the obstacle. The definition of a ϵ-collision-free approximate trajectory is as follows (Figure 2-7):

> *There exists an optimal feasible trajectory* $x^*: [0, T^*] \to X_{free}$, *constants* $\alpha, \bar{\epsilon} \in \mathbb{R}_{>0}, p \in \mathbb{N}$, *and a continuous function* $q: \mathbb{R}_{>0} \to X$ *with* $\lim_{\epsilon \downarrow 0} q(\epsilon) = x^*$ *such that for all* $\epsilon \in (0, \bar{\epsilon})$ *the following hold for the path* $x_\epsilon = q(\epsilon): [0, T_\epsilon] \to X_{free}$:
>
> - x_ϵ *is an* ϵ*-collision-free path that starts from* z_{init} *and reaches the goal,*
> - *for any* $t_1 < t_2$, *let* $z_1 = x_\epsilon(t_1)$ *and* $z_2 = x_\epsilon(t_2)$, *then the ball of radius* $\alpha\|z_1 - z_2\|^p$ *centered at* z_2 *is* ϵ*-reachable from* z_1.

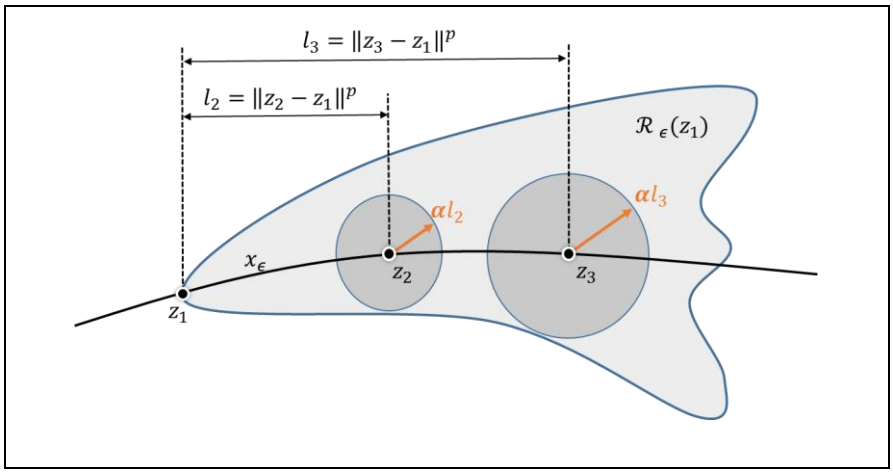


Figure 2-7 A ϵ-collision-free approximate trajectory [16]

Goretkin et. al. [11] extend these conditions to include time into the state of the system. The conditions require that the tree will be able to make local connections with trajectories that remain in an obstacle-free neighborhood, and that the optimal trajectory through the obstacle field is continuously surrounded by a neighborhood of approximately optimal solutions which are collision free. These conditions are satisfied for a system that cannot go backward in time [11]. The systems within the simulations performed for this thesis, do, in fact, adhere to RRT*'s sufficient conditions, as will be presented in Chapter 3.

2.2.3 Algorithm

The variation of RRT* created in this thesis is a combination of several variations of RRT* developed since its inception. For clarity, this section will present the original RRT* algorithm in detail, and then discuss the modifications made for kinodynamic systems. The complete algorithm used in this thesis is presented in Chapter 3.

2.2.3.1 Primitives

The basic primitive procedures, functions that in conjunction form the behavior of the RRT* algorithm, are as follows[17]:

- `SampleFree`: The `SampleFree` procedure returns uniformly distributed independent sample nodes from the obstacle-free space X_{free}.
- `NearestVertex`: The `NearestVertex` procedure returns the nearest vertex *V* in the tree *G* to a given state *z* based on an application specific cost function
- `NearVertices`: The `NearVertices` procedures returns the vertices *V* in the tree *G* that are within a ball of size $r = \gamma \left(\frac{\log(n)}{n}\right)^d$ from a given state *z*. Here *d* is the dimension of the problem, *n* is the size of *V* at the current iteration and γ is an appropriate scaling constant.
- `Steer`: Given two points *x* and *y* the `Steer` procedure returns a point $z \in X$ such that *z* is closer to *y* than *x* based on an application specific cost function.
- `CollisionFree`: The `CollisionFree` procedure returns true if the given edge *x* connecting two states is entirely collision free, and false otherwise.

*2.2.3.2 RRT**

One full iteration of the RRT* algorithm after several iterations have occurred is outlined in visual form in Figure 2-8. Path directions are not drawn in Figure 2-8 because they can be easily deduced given that RRT* is a directed tree. The algorithm begins with the initialization of the search graph *G* through the addition of vertex $V = \{z_{start}\}$ and an empty set of edges, $E = \emptyset$. An iteration of RRT* begins with `SampleFree`, where a random node in free space is returned

(image 1). RRT* then proceeds to extend from the nearest vertex in the search tree toward z_{rand} using `Steer` (image 2). If the path is collision free, the point in the space that `Steer` reaches is dubbed z_{new} and added to the search tree with parent $z_{nearest}$ (image 3). Next the near vertices to z_{new} are found using `NearVertices` (image 4). A connection is attempted from each near vertex in the tree to z_{new}, and the connection that results in the lowest cumulative cost to arrive at z_{new} is kept (image 5 and 6). Finally a connection is attempted from z_{new} to each z_{near} and z_{new} is made the parent to each z_{near} for which the cumulative cost of the path through z_{new} is lower than the cost of the current path to z_{near} (image 7 and 8). This ends one iteration of RRT*. Iterations of the algorithm repeat as computation time allows.

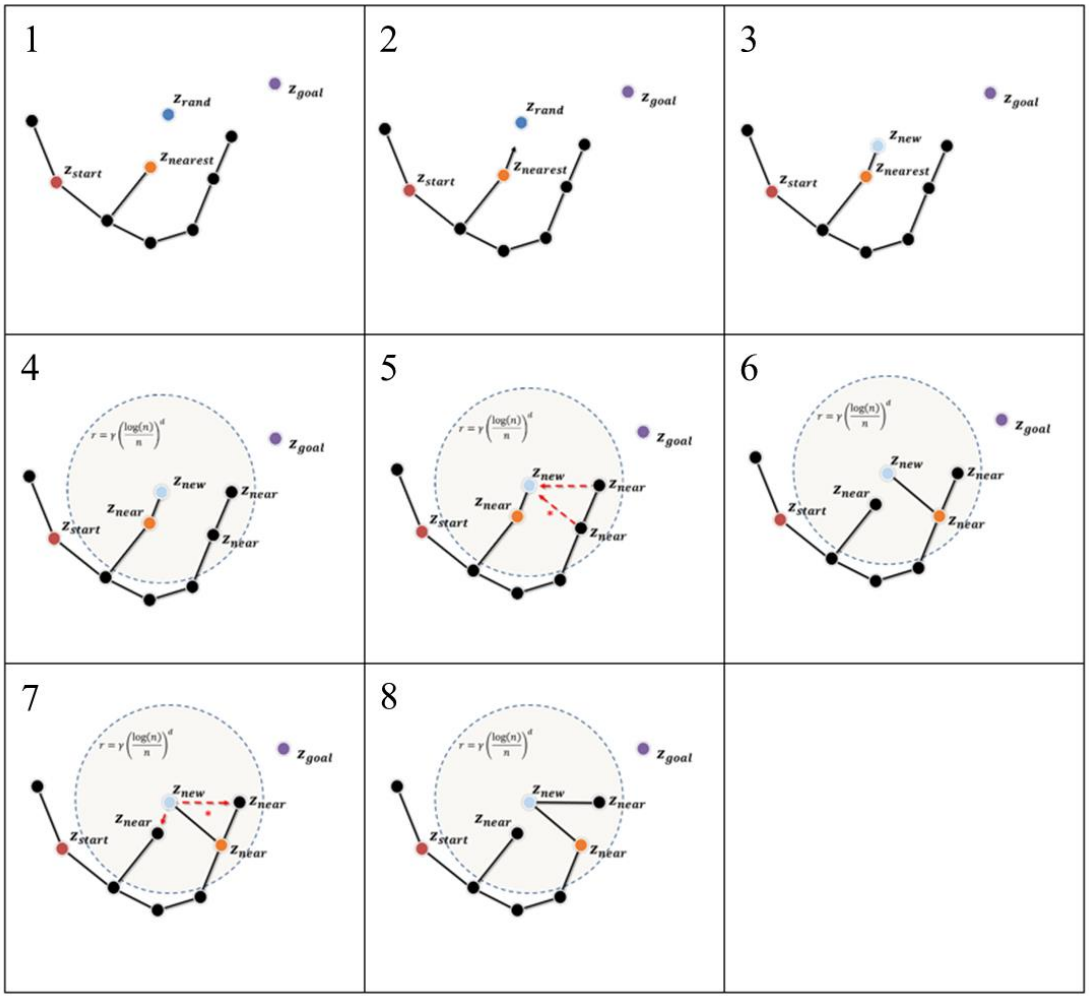


Figure 2-8 RRT* procedure

2.2.3.3 Extensions

Karaman and Frazzoli were the first to extend RRT* to handle differential constraints[16]. In their work they provide a set of sufficient conditions to guarantee asymptotic optimality of RRT* for systems with differential constraints. Webb and Berg extend RRT* by using a fixed-final-state-free-final-time controller that exactly and optimally connects any pair of states for systems with linear differential constraints[25]. This Kinodynamic RRT* algorithm differs from the original RRT* presented above primarily in its steering function. While RRT* only extends in the

direction of the randomly sampled state z_{rand}, Kinodynamic RRT* steers exactly to z_{rand} using an optimal controller. In addition, Kinodynamic RRT* attempts a connection from z_{new} to z_{goal} at every iteration if a collision free connection has already been made to z_{new}. Finally, Goretkin et. al. [11] modified RRT* to use LQR control to connect vertices in the search tree, and adds time to the search space in which the tree grows. This means that connections can only be made when z_{i+1} is located at a time greater than z_i, but also improves near vertex calculations. Additionally these authors provided conditions for optimality when state-time is added to the search space.

2.3 CMA-ES

In the world of optimization, Evolutionary Algorithms (EAs) excel where exhaustive and deterministic search are infeasible, and naïve random search takes too long[1]. An evolutionary algorithm begins with an initial random population, which is also called the first generation. Each generation, the fitness or cost of each member of the generation is evaluated and sorted. The algorithm then searches the decision space in the next generation probabilistically based on the performance of members of the previous generation, mimicking biological evolution, as seen in Figure 2-9. This procedure leads to an algorithm that can perform a global search on a wide variety of objective functions.

Figure 2-9 will be referred to throughout this section as a good visual representation of how the CMA-ES equations adapt search over several generations. In the figure, the bright white circle is the global optimum and the concentric circles represent lines of constant cost. The black dots represent candidate solutions, and the orange circle represents the search distribution. In generation one, search begins in an infeasible region of the space, however there are some lower cost outliers which direct search in generation two. Through the generations, the covariance and

the mean of the search distribution update to increase the likelihood of previous low cost solutions, until the distribution converges onto the global optimum.

CMA-ES is a stochastic method for real-parameter optimization of non-linear, non-convex functions[14]. It is an adaptive evolutionary strategy that chooses, according to a multivariate normal distribution, new search points.

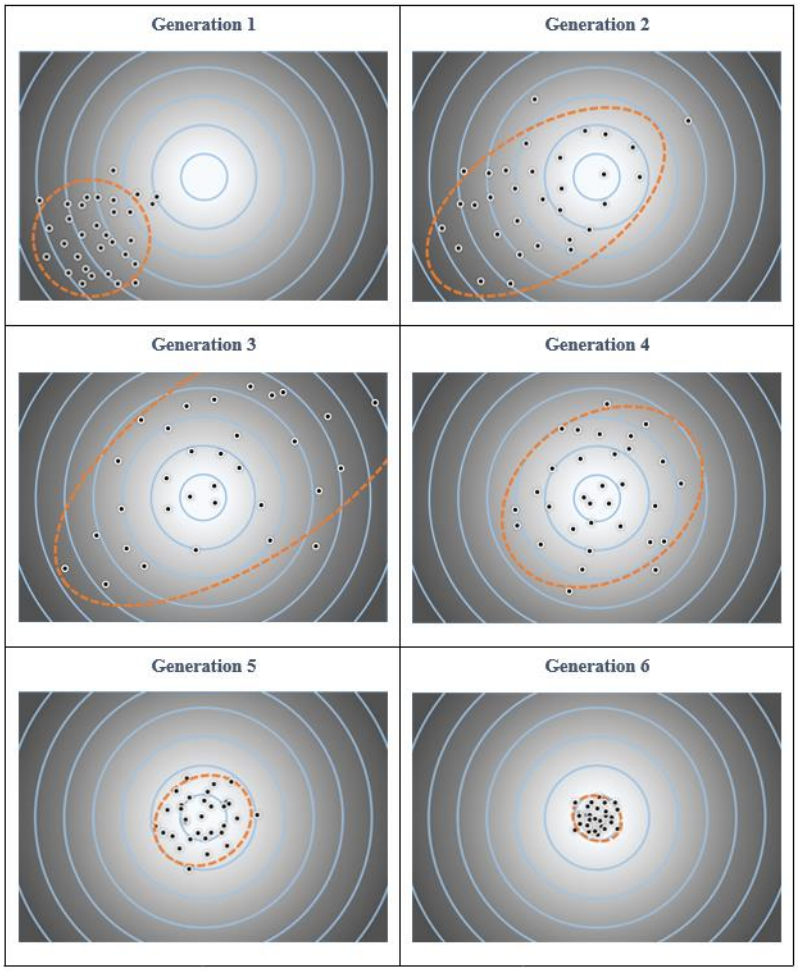


Figure 2-9 CMA-ES evolution over six generations (Wikimedia Commons)

2.3.1 Theory

There are three main components of CMA-ES that govern search. They are the search mean, covariance, and step size. Each are adapted according to the equations presented in Section 2.3.2. The following sections will discuss in more detail the significance of these parameters, however a full treatment can be found in [12], [13], and [14].

2.3.1.1 Mean Update

The first component of candidate solution selection in Eq. 8 is the mean of the search distribution. The mean update can be seen in Figure 2-9 by looking at the change in the location of the center of the orange search ellipse. The adaptation of the mean of the search takes the characteristics of the most traditional evolutionary strategy. Within the mean adaptation, Eq. 9, selection and recombination occurs. Selection is where the algorithm chooses the individuals with the best fitness from each generation to become the parents of the next generation. Recombination is where the algorithm uses the parent solutions to create candidate solutions for the next generation. CMA-ES uses non-elitist selection and truncation selection. Non-elitist selection means the next generation does not explicitly contain members from the previous generation. Truncation selection means that only a percentage of the highest fitness solutions from a generation is used to determine the search criteria for the next generation. In Eq. 9, selection occurs by taking $\mu < \lambda$ solutions for recombination, and recombination occurs as a weighted sum of the μ individuals, where λ is the number of candidate solutions from the previous generation[13].

2.3.1.2 Covariance Update

The second component of the candidate solution selection in Eq. 8 is the product of the step size, or standard deviation, and a multivariate normal distribution with zero mean and covariance matrix *C*. This covariance matrix determines the search distribution, and the desire is to adapt the covariance matrix so search is directed along the contour lines of the objective function to be minimized[13]. The covariance update can be seen in Figure 2-9 by looking at the change in shape of the orange search ellipse. There are two main components to the covariance matrix update in Eq. 15, the rank-μ update (Eq. 6) and the rank-one update (Eq. 7).

$$\boldsymbol{C}^{(g+1)} = \left(1 - c_\mu\right)\boldsymbol{C}^{(g)} + c_\mu \sum_{i=1}^{\mu} w_i y_{i:\lambda}^{(g+1)} \left(y_{i:\lambda}^{(g+1)}\right)^T \tag{6}$$

$$\boldsymbol{C}^{(g+1)} = (1 - c_1)\boldsymbol{C}^{(g)} + c_1 \vec{p}_c^{(g+1)} \vec{p}_c^{(g+1)^T} \tag{7}$$

The rank-μ update uses the information within the population of the current generation to estimate the distribution of successful steps, or steps in which lower cost solutions are found[13]. The information from the current generation is balanced with information from the previous generations with $c_\mu \leq 1$. The rank-μ update is most effective with large population sizes because there are enough candidate solutions in each generation for a good approximation of the distribution of the generation's successful steps.

The rank-one update utilizes the evolution path to estimate the distribution of successful steps[13]. The evolution path is the sequence of steps the evolutionary strategy takes over some number of generations. Essentially, rank-one update uses cumulative information over a number of generations to direct search. The evolution path at generation *g* is described by Eq. 14. This evolution path is able to capture and exploit the correlations between consecutive steps, improving search capabilities when the population size is small and cannot give a good approximation of successful steps in a single generation.

2.3.1.3 Step Size Update

The step size or standard deviation, σ, is the final component of the selection in Eq. 8. It directs how quickly the search distribution can travel to promising areas of the search space. The step size update can be seen Figure 2-9 by looking at the distance between search means from one generation to the next. Although the adaptation of the covariance matrix will inherently change the step size of the search, this change occurs too slowly for good step length adaptation[13]. Step size, like the rank-one covariance matrix, is updated based on the evolution path. If the evolution path is larger than expected, that means the individual search steps are proceeding in the same direction and the step size should be increased to move to the promising search area quicker. If the evolution path is shorter than expected, this means that single steps are cancelling each other out and step-size should be shortened. If the evolution path is the expected length, this means the single steps are uncorrelated, which is the desired result[13].

The problem is that the evolution path p_c's length depends on its direction, so instead CMA-ES utilizes the conjugate of the evolution path, Eq. 12, who's length is independent of its length[13]. The length of p_σ is compared to its expected length, $E\|\mathcal{N}(0, \boldsymbol{I})\|$ [13], and the step-size procedure is followed in Eq. 13.

2.3.2 Algorithm

CMA-ES searches the space by choosing new search points according to a multivariate normal distribution, shown in Eq. 8.

$$\vec{x}_k^{(g+1)} \sim \vec{m}^{(g)} + \sigma^{(g)} \mathcal{N}\left(\vec{0}, \boldsymbol{C}^{(g)}\right) \quad \text{for } k = 1, \dots, \lambda \tag{8}$$

where $\mathcal{N}\left(\mathbf{0}, \boldsymbol{C}^{(g)}\right)$ is a multivariate normal distribution with zero mean and covariance matrix $\boldsymbol{C}^{(g)}$, λ is the child population size, or the search population. The mean vector $\vec{m}^{(g)} \in \mathbb{R}^n$ is the

mean value of the search distribution at generation g, where n is the number of decision variables. The step size $\sigma^{(g)} \epsilon \mathbb{R}_+$ is the overall standard deviation at generation g. Finally $\boldsymbol{C}^{(g)} \epsilon \mathbb{R}^{n \times n}$ is the covariance matrix which determines the shape of the distribution ellipsoid[14].

After a new population is created, the mean vector is updated according to Eq. 9 and Eq. 10. This new mean is a weighted average of μ selected points. $\mu \leq \lambda$ is the parent population size. The parent population consists of the top μ fitness evaluated members of the child population. μ_{eff} is referred to as the variance effective selection mass, and is used frequently in the following equations.

$$\vec{m}^{(g+1)} = \sum_{i+1}^{\mu} w_i \vec{x}_{i:\lambda}^{(g+1)} \tag{9}$$

$$\sum_{i=1}^{\mu} w_i = 1, \qquad w_1 \geq w_2 \geq \ldots \geq w_\mu > 0 \tag{10}$$

$$\mu_{eff} = \left(\sum_{i=1}^{\mu} {w_i}^2\right)^{-1} \tag{11}$$

Here, $w_{i=1\ldots\mu}$ is a positive real weight coefficient for recombination. $\vec{x}_{i:\lambda}^{(g+1)}$ is the *i*-th best ranked individual from the parent population. The step size is updated dynamically according to Eq. 12 and Eq. 13.

$$\vec{p}_\sigma^{(g+1)} = (1 - c_\sigma)\vec{p}_\sigma^{(g)} + \sqrt{c_\sigma(2 - c_\sigma)\mu_{eff}} \left(\boldsymbol{C}^{(g)-\frac{1}{2}}\right) \left(\frac{\vec{m}^{(g+1)} - \vec{m}^{(g)}}{\sigma^{(g)}}\right) \tag{12}$$

$$\sigma^{(g+1)} = \sigma^{(g)} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\left\|\vec{P}_\sigma^{(g+1)}\right\|}{E\|\mathcal{N}(0,\boldsymbol{I})\|}\right) - 1\right) \tag{13}$$

The Covariance Matrix is updated dynamically according to Eq. 14 and Eq. 15.

$$\vec{p}_c^{(g+1)} = (1 - c_c)\vec{p}_c^{(g)} + \sqrt{c_c(2 - c_c)\mu_{eff}} \left(\frac{\vec{m}^{(g+1)} - \vec{m}^{(g)}}{\sigma^{(g)}}\right) \tag{14}$$

$$\boldsymbol{C}^{(g+1)} = \left(1 - c_1 - c_\mu\right)\boldsymbol{C}^{(g)} + c_1 \vec{p}_c^{(g+1)} \vec{p}_c^{(g+1)^T} + c_\mu \sum_{i=1}^{\mu} w_i y_{i:\lambda}^{(g+1)} \left(y_{i:\lambda}^{(g+1)}\right)^T \tag{15}$$

$$y_{i:\lambda}^{(g+1)} = \frac{\left(\vec{x}_{i:\lambda}^{(g+1)} - \vec{m}^{(g)}\right)}{\sigma^{(g)}} \tag{16}$$

There are additional default strategy parameters that govern the above adaptations. They can be found in Appendix A.

2.4 The Hill-Clohessy-Wiltshire Equations

The planner developed in this thesis is proposed for missions where a chase vehicle is navigating in close proximity to a target vehicle in a circular orbit. The Hill-Clohessy-Wiltshire (HCW) Equations govern the dynamics of this system[5]. These equations are a linearized version of the non-linear equations governing relative motion in orbit and have a closed form solution. Thus the HCW equations lend themselves well to implementation within RRT*'s steering function. The development of these equations will follow the methodology put forward by Curtis in [6].

2.4.1 Nonlinear Equations

Prior to developing the equations of motion for this system, a frame of reference must be developed. With the origin at the target spacecraft, $\hat{\imath}$ in the radial direction, $\hat{\jmath}$ in the in-track direction of the orbit, and $\hat{k}$ normal to the orbital plane, this frame is seen in Figure 2-10. Here v_t is the target vehicle and frame origin, v_c is the chase vehicle, $\vec{r}_t$ is the orbital position of the target vehicle, $\vec{r}_c$ is the orbital position of the chase vehicle and $\delta\vec{r}$ is the position of the chase vehicle relative to the target vehicle.

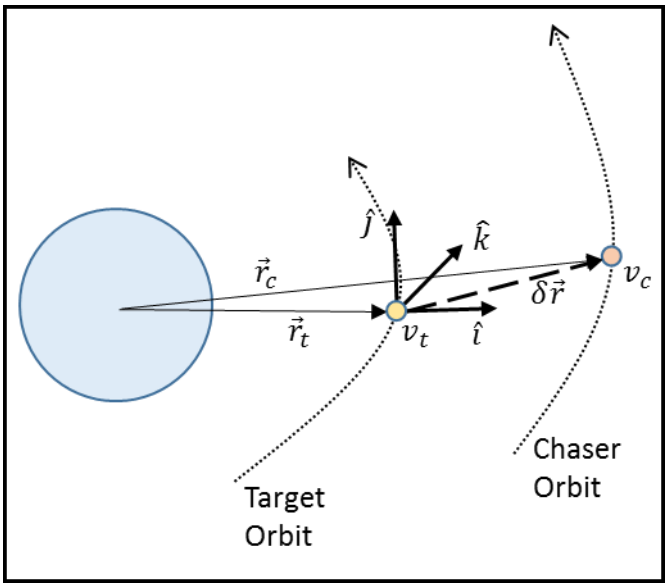


Figure 2-10 HCW reference frame [6]

Given the above notation, it is easy to see

$$\vec{r}_c = \vec{r}_t + \delta\vec{r} \tag{17}$$

where

$$\delta\vec{r} = \delta x\hat{\imath} + \delta y\hat{\jmath} + \delta z\hat{k} \tag{18}$$

For the systems addressed in this thesis and in the HCW Equations, in Eq. 17 the magnitude of $\delta\vec{r}$ is much smaller than $\vec{r}_t$, or

$$\frac{\delta r}{r_t} \ll 1 \tag{19}$$

where $\delta r = \|\delta\vec{r}\|$, and $r_t = \|\vec{r}_t\|$. Treating the system of the Earth and a satellite as a two-body system, we have the equations of motion of an Earth orbiting satellite in the inertial geocentric equatorial frame given by

$$\ddot{\vec{r}} = -\mu\frac{\vec{r}}{r^3} \tag{20}$$

where $r = \|\vec{r}\|$. To obtain the nonlinear equations of motion of v_c relative to v_t, Equation 17 is substituted into Eq. 20, to obtain Eq. 21, where $r_c = \|\vec{r}_c\| = \|\vec{r}_t + \delta\vec{r}\|$.

$$\delta\ddot{\vec{r}} = -\ddot{\vec{r}}_t - \mu\frac{\vec{r}_t+\delta\vec{r}}{r_c^3} \tag{21}$$

2.4.2 Linearization

Linearization of Eq. 21 occurs through use of Eq. 19. The goal is to remove the nonlinear term r_c^{-3}. This is done in [6] by utilizing Eq. 19 and the fact that any term with $\frac{\delta r}{r_t}$ raised to a power greater than one goes to zero. The linearized equations of motion governing a chaser with respect to a target are then given in Eq. 22.

$$\delta\ddot{\vec{r}} = -\frac{\mu}{r_t^3}\left[\delta\vec{r} - \frac{3}{r_t^2}(\vec{r}_t\cdot\delta\vec{r})\vec{r}_t\right] \tag{22}$$

The linearized equations of motions in the frame presented in Figure 2-10 are as follows

$$\delta\ddot{x} - \left(\frac{2\mu}{r_t^3}+\frac{h^2}{r_t^4}\right)\delta x + \frac{2\left(\dot{\vec{r}}_t\cdot\vec{r}_t\right)h}{r_t^4}\delta y - 2\frac{h}{r_t^2}\delta\dot{y} = 0 \tag{23}$$

$$\delta\ddot{y} + \left(\frac{\mu}{r_t^3}-\frac{h^2}{r_t^4}\right)\delta y - \frac{2\left(\dot{\vec{r}}_t\cdot\vec{r}_t\right)h}{r_t^4}\delta x + 2\frac{h}{r_t^2}\delta\dot{x} = 0 \tag{24}$$

$$\delta\ddot{z} + \frac{\mu}{R^3}\delta z = 0 \tag{25}$$

where $h = \left\|\vec{r_t}\times\dot{\vec{r}_t}\right\|$. A complete derivation of these equations can be found in [6].

2.4.3 HCW Equations

Hill and Clohessy took Eqs. 23-25 and applied them to a system with a target vehicle in a circular orbit. In this case, $\dot{\vec{r}}_t\cdot\vec{r}_t = 0$, $h = \sqrt{\mu r_t}$, and the mean motion of the orbit, $n = \sqrt{\frac{\mu}{r_t^3}}$.

Making these substitutions results in the Hill-Clohessy-Wiltshire Equations,

$$\delta\ddot{x} - 3n^2\delta x - 2n\delta\dot{y} = 0 \tag{26}$$

$$\delta\ddot{y} + 2n\delta\dot{x} = 0 \tag{27}$$

$$\delta\ddot{z} + n^2\delta z = 0 \tag{28}$$

The HCW equations exhibit a closed form solution which is fully developed in [6] by solving the above differential equations with initial conditions

$$\begin{gathered}\delta x(0) = \delta x_0, \delta y(0) = \delta y_0, \delta z(0) = \delta z_0 \\ \delta \dot{x}(0) = \delta \dot{x}_0, \delta \dot{y}(0) = \delta \dot{y}_0, \delta \dot{z}(0) = \delta \dot{z}_0\end{gathered} \tag{29}$$

The solution to the HCW equation is as follows, letting $\delta \dot{x} = \delta u, \delta \dot{y} = \delta v,$ and $\delta \dot{z} = \delta w$, and transitioning to matrix form where

$$\{\delta \vec{r}\} = \begin{Bmatrix} \delta x \\ \delta y \\ \delta z \end{Bmatrix}, \ \{\delta \vec{v}\} = \begin{Bmatrix} \delta u \\ \delta v \\ \delta w \end{Bmatrix} \tag{30}$$

$$\{\delta \vec{r}(t)\} = [\boldsymbol{\Phi}_{rr}(t)]\{\delta \vec{r}_0\} + [\boldsymbol{\Phi}_{rv}(t)]\{\delta \vec{v}_0\} \tag{31}$$

$$\{\delta \vec{v}(t)\} = [\boldsymbol{\Phi}_{vr}(t)]\{\delta \vec{r}_0\} + [\boldsymbol{\Phi}_{vv}(t)]\{\delta \vec{v}_0\} \tag{32}$$

The matrices that appear in the compact form of the HCW equations are as follows

$$\boldsymbol{\Phi}_{rr}(t) = \begin{bmatrix} 4 - 3\cos nt & 0 & 0 \\ 6(\sin nt - nt) & 1 & 0 \\ 0 & 0 & \cos nt \end{bmatrix} \tag{33}$$

$$\boldsymbol{\Phi}_{rv}(t) = \begin{bmatrix} \frac{1}{n}\sin nt & \frac{2}{n}(1 - \cos nt) & 0 \\ \frac{2}{n}(\cos nt - 1) & \frac{1}{n}(4\sin nt - 3nt) & 0 \\ 0 & 0 & \frac{1}{n}\sin nt \end{bmatrix} \tag{34}$$

$$\boldsymbol{\Phi}_{vr}(t) = \begin{bmatrix} 3n\sin nt & 0 & 0 \\ 6n(\cos nt - 1) & 0 & 0 \\ 0 & 0 & -n\sin nt \end{bmatrix} \tag{35}$$

$$\boldsymbol{\Phi}_{vv}(t) = \begin{bmatrix} \cos nt & 2\sin nt & 0 \\ -2\sin nt & 4\cos nt - 3 & 0 \\ 0 & 0 & \cos nt \end{bmatrix} \tag{36}$$

Given a travel time, *t*, an initial chase vehicle state, $\delta \vec{x}_0$, and final vehicle chase vehicle state, $\delta \vec{x}_f = \delta \vec{x}(t)$, Eqs. 31 and 32 will be used to determine the changes in velocity required to travel exactly between $\delta \vec{x}_0$ and $\delta \vec{x}_f$ in time *t*.

The components of the hybrid algorithm developed in this thesis have now been established. The next chapter will explain how the components are combined to solve the motion planning problem outlined in Section 2.1 in a novel and effective way.

Chapter 3

The Hill-Clohessy-Wiltshire RRT* Evolutionary Strategy Algorithm

The main contribution of this thesis is the combination of the algorithms described in Chapter 2 and their application to close proximity spacecraft motion planning. The algorithm is called HCW RRT*-ES and pseudocode for the algorithm is provided in Appendix C. At a high level the algorithm functions with RRT* as the main trajectory planner. Within RRT*, CMA-ES evaluates vertices previously explored by RRT* and directs search in promising directions in the space. The HCW equations are used within the steering function to determine the control input required to connect vertices in RRT* in a dynamically feasible way. For the remainder of this work, the hybrid HCW RRT*-ES algorithm will be referred to as RRT*-ES, while the HCW RRT* algorithm which does not use CMA-ES will simply be referred to as RRT*.

To initialize the algorithm the user inputs the desired start and goal state of the vehicle, the bounds on the states, and the bounds on the control input, along with a vehicle model, thruster model, and obstacle model.

3.1 RRT* Integration

The RRT* portion of the hybrid algorithm acts as the main search framework. It investigates candidate trajectories and attempts to find the lowest cost trajectory from the start to the goal. The version of RRT* used in this work takes elements from [11] and [25], along with additions to specifically address the spacecraft relative motion problem.

The algorithm plans in seven degrees of freedom, adding time to the state space as in [11]. The chase vehicle state takes the form

$$\vec{x} = [\delta x \quad \delta y \quad \delta z \quad \delta u \quad \delta v \quad \delta w \quad t]^T \tag{37}$$

The addition of time to the state space make the collision free cost calculation between states computationally efficient and accurate, which is a necessary condition for asymptotic optimality of the RRT* algorithm. A version of the algorithm without time in the state space was presented by the author in [8].

The chase vehicle is assumed to have thrusters pointing in the positive and negative $x, y,$ and z directions of the vehicle body frame. The control input takes the form of the impulsive change in velocity, or $\Delta\vec{V}$ required to connect two states in the time difference between the two states, according to the HCW equations. A control input takes the form

$$\vec{u} = \Delta\vec{V} = [\Delta V_{\delta x} \quad \Delta V_{\delta y} \quad \Delta V_{\delta z}]^T \tag{38}$$

Currently the body frame of the inspection vehicle is assumed to be irrotational with respect to the HCW frame. This simplifies computation while still allowing for the planning of dynamically feasible paths. Theoretically, the $\Delta\vec{V}'s$ given by this algorithm could be mapped into any thruster configuration that exhibits full controllability in the δx, δy, and δz directions.

Since states can be connected exactly using impulsive maneuvers based on the solution to the HCW equations, whenever a state is added to the search tree a connection is attempted between the new state and the goal state as in [25]. Figure 3-1 outlines the process for attempting to connect to the goal state as implemented in HCW RRT*-ES in visual form. Figure 3-1 begins in image 1 with an already developed tree, where a connection has been made from a tree vertex to the new node. A connection is attempted between the new node, now dubbed z_m and the goal state in Figure 3-1, image 2. If this is the lowest cost connection to the goal, the new state is made the parent to the goal state. This is accomplished using the `GoalConnect` function.

If the connection between the new state and the goal state is successful, but not the lowest cost connection to the goal, the connection and state are nonetheless stored in a goal parent structure (Figure 3-1, image 3). Whenever a rewire occurs, this goal structure is revisited. A

rewire has occurred Figure 3-1, image 4. Note that image 4 and 5 occur at some iteration after the iteration when where image 1, 2, and 3 occur. The goal structure revisit occurs within the `GoalRewire` function, and is necessary because a rewire could lower the cost of a previous found trajectory from start to goal. If a member of the goal parent structure is in fact a member of the lowest cost trajectory to the goal vertex after a rewire, it becomes the unique parent to the goal vertex (Figure 3-1, image 5).

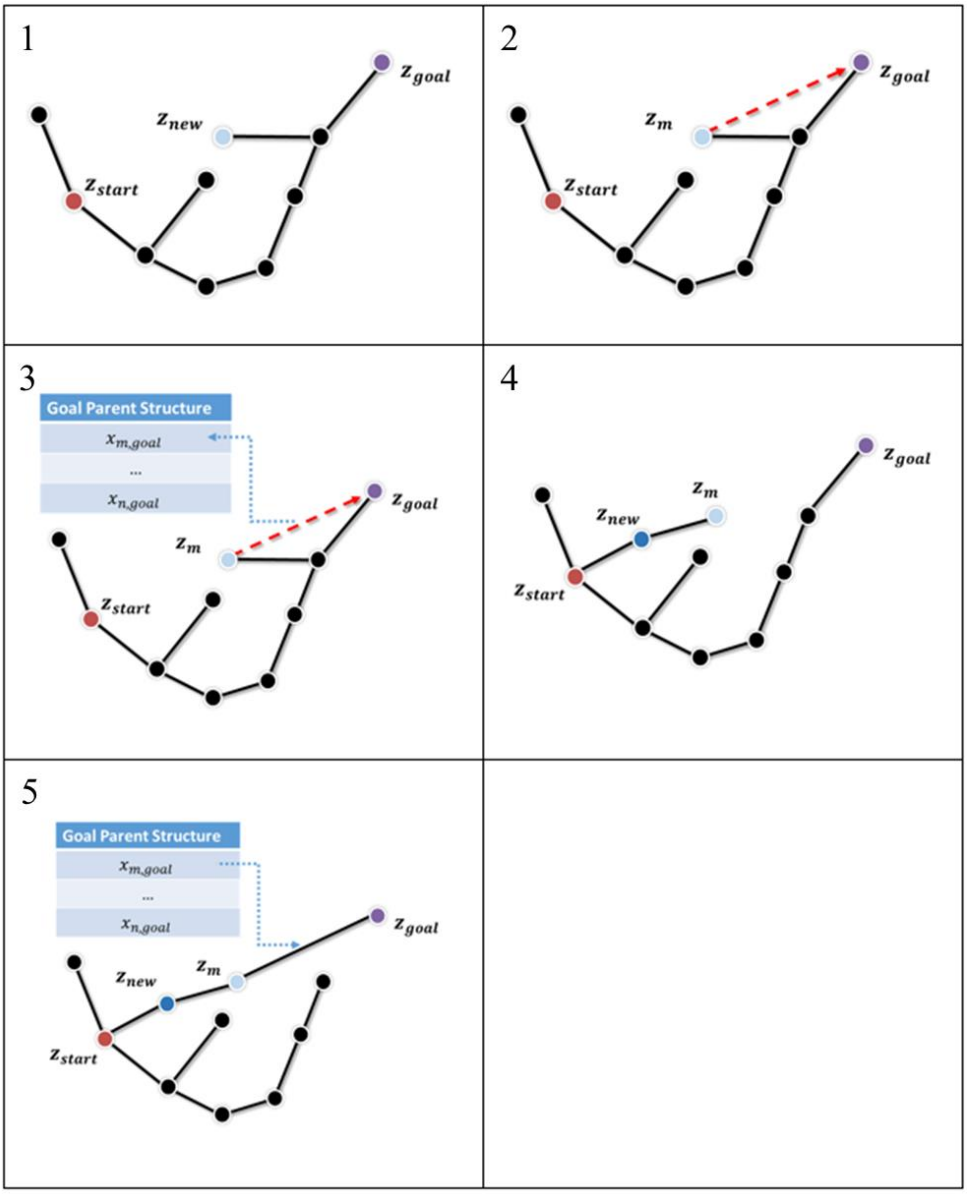


Figure 3-1 `GoalConnect` **and** `GoalRewire` **procedure**

3.2 CMA-ES Integration

One of the main contributions of this work is the integration of the CMA-ES algorithm into RRT* to improve search characteristics. CMA-ES is implemented to act as a local optimizer, converging on low cost areas of the search tree to find the lowest cost node in a region. CMA-ES

based sampling replaces the random sampling of the standard RRT*, and adaptation of CMA-ES parameters occurs based on a separate adaptation procedure within the algorithm.

3.2.1 CMA-ES Parameters

The goal of the CMA-ES integration with RRT* is to exploit areas where RRT* finds low cost nodes. To achieve this goal, CMA-ES must quickly explore promising areas while allowing the hybrid algorithm to maintain strong search characteristics. Three parameters to the CMA-ES algorithm are set to encourage this behavior. They are the child population size, the initial step size, and the parent population size.

To achieve local optimization, a small adaptation population is utilized. The use of a small child population size leads to fast convergence, and encourages convergence on local optima.[14] This is the desired result in this work, but must be accounted for because an optimal trajectory may require multiple vertices. If convergence is not accounted for, the hybrid algorithm would only identify the trajectory containing the vertex that it first converges upon. To overcome pre-convergence and maintain search, when convergence occurs the CMA-ES parameters are reset to their original values. Convergence is detected by the step size, or standard deviation value, as well as the change in the search mean.

The initial step size is set to be half the size of the search space, with all states scaled appropriately. This is a fairly large value for the initial step size, but it allows for a thorough search of the space each time CMA-ES is reset. Step size then adapts to optimize search, and eventually shrinks as CMA-ES converges. Once it reaches a small enough size, the step size is reset to its original value. If the step size becomes too small, and there has been no significant change in mean since the last adaptation, then all CMA-ES parameters reset to their original values.

The parent population is set to be only the top fitness individual of the previous generation. This is known as a (1,λ)-ES, where 1 is the parent population size, and λ is the child population size. It results in fast search that again tends toward pre-convergence, a trait that is actually desired in this work. Recall also that CMA-ES uses information from previous generations when performing covariance and step size updates, so evolution is not happening in a bubble each generation. This means a small parent population size can still result in a good search of the space.

3.2.2 CMA-ES Population Requirements

To perform the local optimization task that CMA-ES is enlisted for in this work, it requires a population of evaluated solutions. As the standard RRT* search occurs, the nodes that have been searched and their associated cost are added to the CMA-ES data structure, or population, under certain conditions. Note that the CMA-ES population is a separate data structure from the RRT* search tree, and is used only to evolve the selection criteria for candidate nodes. The criteria for a node to be added to the CMA-ES population are looser than the criteria for a node to be added to the RRT* search tree because even nodes that are infeasible under certain conditions still contain useful adaptation information.

The CMA-ES population consists of the seven degree of freedom node states and an associated cost which determines the probability a future node will be chosen from a similar region in the space. A node is added to the population if it satisfies the following two conditions:

1. The node is reachable with a feasible amount of thrust from a current vertex in the search tree.
2. The goal is reachable with a feasible amount of thrust from the node.

If a node can satisfy both 1 and 2, and both of these connections are collision free, the node is added to the population with a cost equivalent to the cumulative cost to arrive at the node, plus the cost to reach the goal

$$c_{CMAES}(z) = Cost(z_{parent}) + c\left(x_{z_{parent},z}\right) + c(x_{z,z_{goal}}) \tag{39}$$

where z_{parent} is the parent vertex to z in the search tree, $x_{z_{parent},z}$ is the trajectory connecting z_{parent} to z, and $x_{z,z_{goal}}$ is the trajectory connecting z, to z_{goal}. If a collision occurs during the execution of 1 or 2, the cost of the node in the population, calculated in Eq. 39, is doubled. If a node cannot satisfy both 1 and 2, it is not added to the adaptation population.

3.3 HCW Steering Function

Within the RRT* algorithm, a steering function is required to determine the optimal connection trajectory between vertices. Due to the nature of the controller utilized in this effort, HCW RRT*-ES implements three variations of the basic steering function. `HCWSteer` is for standard connections, `HCWSteerGoal` is for connections to the goal, and `HCWSteerRewire` is for rewiring the tree's connections. The steering functions in this effort utilize the impulsive ΔV solution to the HCW equations. This method is computationally efficient for both trajectory computation and impingement prevention calculations. This method also adheres to the conditions for asymptotic optimality of RRT*.

In this section the `HCWSteer` function will be developed first, as it serves as a basis for the other two steering functions. This development is followed by the development of `HCWSteerGoal` and `HCWSteerRewire`. Then the impulsive ΔV is validated as accurate, and finally the steering functions will be shown to adhere to the requirements for asymptotic optimality.

3.3.1 HCWSteer Function

The `HCWSteer` functions implement the closed form solution of the HCW equations presented in Chapter 2. When `HCWSteer` is called inside of RRT* it takes in an initial vertex $\vec{z}_i$, and a final vertex $\vec{z}_f$ to be connected by a dynamically feasible trajectory. It then computes the impulsive $\Delta\vec{V}$ maneuvers to move the vehicle exactly from the initial vertex to the final vertex. Recall that each vertex consists of the position, velocity, and a time when that vertex is visited. Thus, `HCWSteer` takes in:

$$\vec{z}_i = [\delta x_i, \delta y_i, \delta z_i, \delta u_i, \delta v_i, \delta w_i, t_i]^T \tag{40}$$

$$\vec{z}_f = \left[\delta x_f, \delta y_f, \delta z_f, \delta u_f, \delta v_f, \delta w_f, t_f\right]^T \tag{41}$$

Let $\vec{r}_i$ and $\vec{v}_i^-$ be the position and velocity components at $\vec{z}_i$ respectively, and $\vec{r}_f$ and $\vec{v}_f^+$ be the position and velocity components at $\vec{z}_f$ respectively. Two impulsive $\Delta\vec{V}$ maneuvers are required to transfer the vehicle from $\vec{z}_i$ to $\vec{z}_f$ in the time between the two states, $\Delta t = t_f - t_i$. To travel from $\vec{r}_i$ to $\vec{r}_f$ in time Δt, the required velocity at $\vec{r}_i$ is given by

$$\{\vec{v}_i^+\} = [\boldsymbol{\Phi}_{rv}(\Delta t)]^{-1}\left(\{\vec{r}_f\} - [\boldsymbol{\Phi}_{rr}(\Delta t)]\{\vec{r}_i\}\right) \tag{42}$$

which is a rearrangement of Eq. 31. The first $\Delta\vec{V}$ which must be performed at $\vec{r}_i$ is then given by

$$\Delta\vec{V}_i = \vec{v}_i^+ - \vec{v}_i^- \tag{43}$$

The resultant velocity at $\vec{r}_f$ is given by

$$\left\{\vec{v}_f^-\right\} = [\boldsymbol{\Phi}_{vr}(\Delta t)]\{\vec{r}_i\} + [\boldsymbol{\Phi}_{vv}(\Delta t)]\{\vec{v}_i^+\} \tag{44}$$

which is Eq. 32, using the syntax and variables developed in the current section. The second $\Delta\vec{V}$ which must be performed at $\vec{r}_f$ is then given by

$$\Delta\vec{V}_i = \vec{v}_f^+ - \vec{v}_f^- \tag{45}$$

Now the required control inputs are known given the two vertices which must be connected within RRT*, however there are some constraints on these control inputs.

The first constraint is trivial, however must be addressed. Given that every vertex has time as part of its state, a vertex can only connect to a state which has a time that is greater than its own time.

The other constraints are control input constraints on the vehicle. The thrusters of the chase spacecraft will only be capable of providing a finite amount of thrust, and therefore the amount of $\Delta\vec{V}$ which can be achieved at a given vertex is bound. The bound check equation is given for a connection between vertices $\vec{z}_i$ and $\vec{z}_f$ as

$$\Delta\vec{V}_{min} \leq \left\{\Delta\vec{V}_i^- + \Delta\vec{V}_i \wedge \Delta\vec{V}_f\right\} \leq \Delta\vec{V}_{max} \tag{46}$$

where $\Delta\vec{V}_i$ and $\vec{V}_f$ are as above, and $\Delta\vec{V}_i^-$ is the $\Delta\vec{V}_f$ required to connect from $\vec{z}_i$'s parent, to $\vec{z}_i$. The impulsive maneuvers considered in the bound check are illustrated in Figure 3-2. $\Delta\vec{V}_i^-$ must be considered, because both $\Delta\vec{V}_i^-$ and $\Delta\vec{V}_i$ occur at $\vec{z}_i$ at time t_i. To maintain the accuracy of the impulsive approximation and the bounds, they must be considered as one impulsive maneuver.

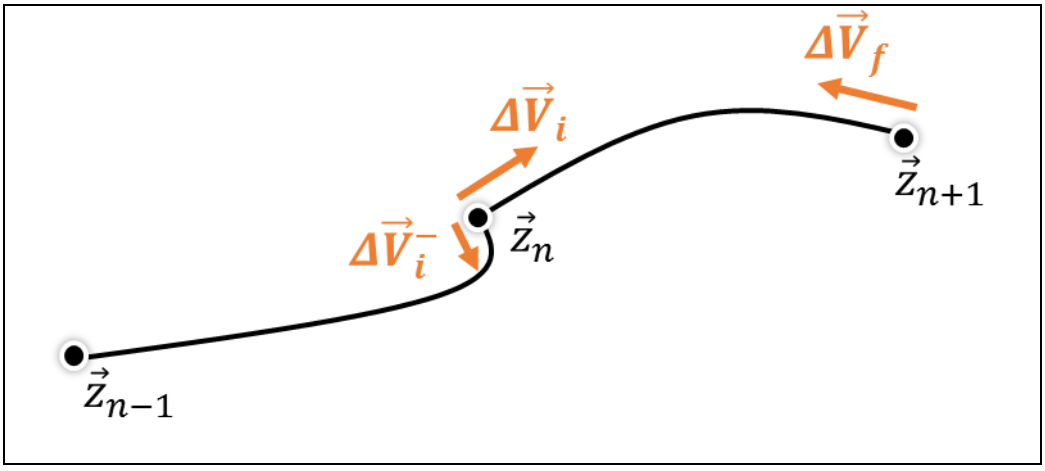


Figure 3-2 Control input bound considerations

If the above constraints hold true, then the steering function assigns a cost to the successful connection. The cost is given by

$$cost = \Delta t + R\left|\Delta\vec{V}_i^{-} + \Delta\vec{V}_i\right| \tag{47}$$

where R is a user defined scaling variable to balance the cost associated with time of flight versus propellant use. It should be noted that $\Delta\vec{V}_f$ is not included in the cost function because it does not represent the entire maneuver occurring at $\vec{z}_f$, following the reasoning given for the Bound Check Equation (Eq. 46) .

The `HCWSteer` function outputs the cost, time, and $\Delta V's$ required to travel between two vertices. If there is not a feasible connection between these two vertices, then the function returns infinity for the cost.

3.3.1.1 HCWSteerGoal Function

First, let $\vec{z}_{goal}$ be the goal node, and $\vec{z}_i$ be the vertex in the search tree for which a connection to $\vec{z}_{goal}$ is attempted. Since one of the parameters that the algorithm is attempting to minimize is time of flight, the time to reach the goal node, t_{goal}, is unknown a priori. This means the `HCWSteerGoal` function must also solve for the time of flight between $\vec{z}_i$ and $\vec{z}_{goal}$ to minimize the cost function.

The minimization occurs by iterating through the time of flight between $\vec{z}_i$ and $\vec{z}_{goal}$. At each time of flight, the cost of the trajectory is calculated, and the lowest cost is tracked. The `HCWSteerGoal` function exits under two conditions:

$$cost^* < \Delta t \tag{48}$$

OR

$$t_{goal} > t_{max} \tag{49}$$

$$t_{goal} = t_i + \Delta t \tag{50}$$

In Eq. 48, $cost^*$ is the lowest cost that has been found at the current iteration. This becomes an exit condition because by Eq. 47, $cost > \Delta t$ for all Δt. Termination of the algorithm at $cost^* < \Delta t$ ensures the global minimum cost and arrival time are found. The second condition in Eq. 49 ensures that the time of flight from start to goal does not exceed the maximum time of flight set by the user. The remainder of `HCWSteerGoal` behaves the same as `HCWSteer`.

3.3.1.2 HCWSteerRewire Function

The third steering function is required for when rewires are attempted. A rewire is when a connection is attempted between two vertices which are already in the search tree (image 7 and 8 of Figure 2-8). Although each vertex can only have a single parent, each vertex can be the parent to multiple child vertices. The consequence of this is that when a rewire occurs, $\vec{z}_f$ could be the parent to one or more vertices, so the control input constraint check must be augmented for a bound check of $\Delta\vec{V}_f$ in addition to $\Delta\vec{V}_i$. The augmented control input constraint check becomes

$$\Delta\vec{V}_{min} \leq \left\{\Delta\vec{V}_i^- + \Delta\vec{V}_i \wedge \Delta\vec{V}_f + \Delta\vec{V}_{f,n}^+\right\} \leq \Delta\vec{V}_{max}, for\ n = 1{:}\, num\_children(\vec{z}_f) \qquad (51)$$

where $\Delta\vec{V}_{f,n}^+$ is the $\Delta\vec{V}_i$ required to traverse from $\vec{z}_f$ to its n^{th} child vertex. Figure 3-3 illustrates a situation where this extra bound check is required. A rewire is being attempted from $\vec{z}_i$ to $\vec{z}_f$, however $\vec{z}_f$ has two child vertices which have established $\Delta V's$ that must be considered in the bound check.

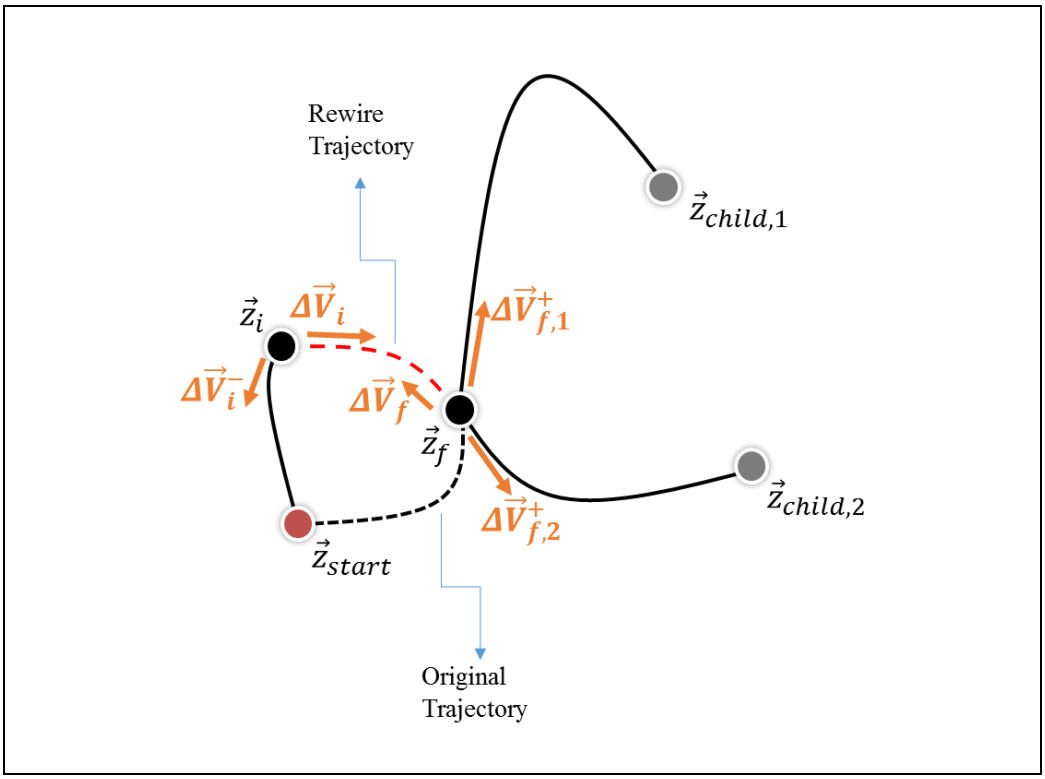


Figure 3-3 `HCWSteerRewire` control input bound considerations

The remainder of the `HCWSteerRewire` function remains the same as in `HCWSteer`, including the cost function. Although the information exists after a rewire to more fully develop the cost of a trajectory between two vertices, to maintain consistency among all vertices in the search tree Eq. 47 is still used to assess the cost of a trajectory resulting from a rewire.

3.3.2 Adherence to Optimality Conditions

In [16], four conditions are presented for asymptotic optimality of the RRT* algorithm. The first condition is that the `Steer` function connects vertices in a locally optimal manner, and that the procedure to determine the near and nearest tree vertices to a new node, must compute the distance in a way that reflects the actual cost to traverse between the vertices and the node. These two conditions are satisfied based on the HCW-based impulsive control developed for this work.

For `HCWSteer` and `HCWSteerRewire`, the control allows for one solution, which is then inherently optimal. For `HCWSteerGoal`, the optimal trajectory is found using iterations over the time of flight to minimize the cost function, as was presented in a previous section. In addition, the procedure to determine the near and nearest tree vertices to a new node uses the `HCWSteer` function for its distance computation. It then naturally follows that these distances reflect the actual cost to travel from vertex to node.

The third requirement is that the system fulfills the Weakened Local Controllability condition, and the fourth requirement is that the system fulfills the ϵ-Collision-Free Approximate Trajectories condition. These requirement are fulfilled for any controllable system whose state is not augmented by time.[16] A vehicle whose dynamics are governed by the HCW equations is in fact controllable. The HCW equations in state space form are given by:

$$A=\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{3\mu}{R^3} & 0 & 0 & 0 & 2\sqrt{\frac{\mu}{R^3}} & 0 \\ 0 & 0 & 0 & -2\sqrt{\frac{\mu}{R^3}} & 0 & 0 \\ 0 & 0 & \frac{-\mu}{R^3} & 0 & 0 & 0 \end{bmatrix} \tag{52}$$

$$B=\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 \\ 0 & \frac{1}{m} & 0 \\ 0 & 0 & \frac{1}{m} \end{bmatrix} \tag{53}$$

$$\begin{bmatrix} \delta\dot{x} \\ \delta\dot{y} \\ \delta\dot{z} \\ \delta\ddot{x} \\ \delta\ddot{y} \\ \delta\ddot{z} \end{bmatrix} = A\begin{bmatrix} \delta x \\ \delta y \\ \delta z \\ \delta\dot{x} \\ \delta\dot{y} \\ \delta\dot{z} \end{bmatrix} + B\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} \tag{54}$$

with controllability matrix given by:

$$C = [B \quad AB \quad A^2B \quad A^3B \quad A^4B \quad A^5B] \tag{55}$$

The system is controllable if the matrix C has full row rank. For the case of the HCW equations, C has a rank of 6, so the HCW equations are indeed controllable. It then follows that the control scheme developed in this work, which uses the solution to the HCW equations also results in a controllable system.

However, the fact that the state of the vehicle in this work is augmented by time must be considered. Goretkin et. al. extend the above conditions to a system where the state of the vehicle is augmented by time in [11]. Therefore, RRT* asymptotic optimality holds for the dynamic system in this work.

3.3.3 Impulsive Maneuver Accuracy

The final portion of the steering functions to be addressed is the accuracy of the impulsive maneuver approximation. The impulsive approximation is often used for planning rendezvous maneuvers, but often these maneuvers occur at greater vehicle separation distances than in the current work. Since separation distance is fairly small in this work, inaccuracies in the impulsive maneuver approximation could directly correlate to collisions when the chase vehicle is attempting to execute its planned trajectory.

The upper bound of the difference between the velocity change for an impulsive maneuver, and the velocity change brought on by a thrust over some finite time is given in [23] as

$$\Delta V_f - \Delta V_I \leq \frac{1}{24}(\omega \Delta t)^2 \Delta V_I \tag{56}$$

where ΔV_f is the velocity change that occurs for a thruster fired over a finite time Δt, ω is the Schuler frequency $\left(\sqrt{\mu/r^3}\right)$, and ΔV_I is the impulsive velocity change approximation. For a given Δt, the maximum finite ΔV is given by

$$\Delta V_{max} = \left(\frac{Thrust_{max}}{m}\right)\Delta t \tag{57}$$

where m is the mass of the spacecraft. In Eq. 56, ΔV_{max} is substituted for ΔV_I on the right side of the equation to give the error bound.

3.4 Impingement Prevention

Impingement prevention is a key component to the motion planning algorithm developed in this thesis. For a trajectory to be free of rocket plume impingement:

$$L_i = \{\vec{x}(t) + s(\vec{x}(t) + P_i u_i(t) \mid s \in [0,1])\} \in X_{free} \tag{58}$$

where L_i is the line segment extending from the position of the vehicle at the time of an impulsive maneuver, to a point a distance away from the position of vehicle proportional to the control input in the direction of the control input, and P_i scales the thrust plume length according to the thruster's characteristics. As an example, if the vehicle is at point (1,1,1) and a ΔV is applied so as to increase the vehicle's velocity in the positive x direction, L would be the line segment from (1,1,1) to (1+PΔV,1,1) where 1+PΔV will be less than 1. If the user desires to have thruster impingement taken into consideration, the test for impingement is called whenever a collision check is called. If an impingement occurs, the maneuver is considered to be infeasible, just as if a collision occurs.

Chapter 4 presents simulation which are used to analyze the effectiveness of the algorithm developed in this work. The new RRT*-ES algorithm is compared with RRT*, and algorithmic capabilities are assessed.

Chapter 4

Simulations

Trial runs with various algorithm parameter settings were performed to validate the algorithm's functionality. The simulations were all performed in MATLAB on Penn State's LionX computing clusters. These clusters consist of Intel Xeon X5675 Six-Core 3.06 GHz processers. The MATLAB code structure, collision detection algorithm, and plotting functions in this work are based on the RRT MATLAB code in [10].

4.1 International Space Station Model

The hybrid algorithm developed in Chapter 3 is applicable to a free-flying spacecraft navigating with respect to a reference object or spacecraft in a circular orbit about a central body. For the purpose of experimentation, the International Space Station was used as the target spacecraft. The ISS not only offers a challenging obstacle environment for the motion planner, but ISS operations would also benefit from the capabilities of an autonomous inspection vehicle.

For experimentation, a full-scale simplified model of the ISS was created in MATLAB. The model is composed of rectangular planes which represent most of the Station's modules and features, as shown in Figure 4-1. Although details are missing, this model provides a structure complicated enough for rigorous algorithm testing. In addition the ISS's natural frequency of 0.0011 rad/s is used in the equations of motion.

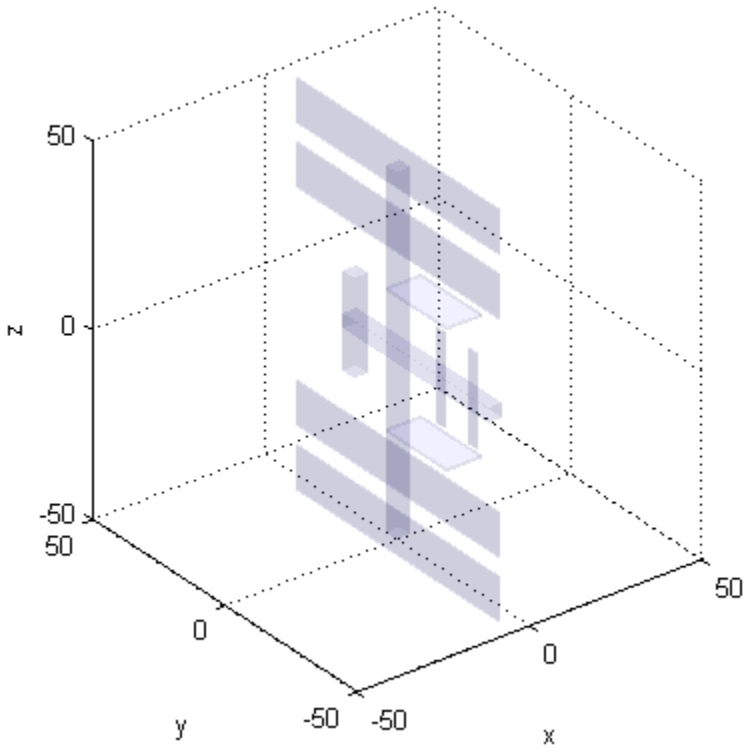


Figure 4-1 ISS MATLAB model

4.2 Free-Flying Vehicle Model

The free-flying vehicle model implemented in these experiments takes its physical parameters from NASA's mini AERCam[9] and SPHERES[4] projects. The thruster configuration is displayed in Figure 4-2. This configuration gives two thrusters pointing in the positive and negative x, y, and z directions respectively. Given this thruster configuration, the vehicle will be capable of achieving the trajectories planned by the motion planning algorithm developed in Chapter 3. Each thruster is modeled as having a maximum thrust of 0.18 N, giving a maximum thrust of 0.36 N in each direction. Finally, the vehicle is modeled as a point mass of 4.5 kg. The point mass model is appropriate for motion planning because obstacles can simply be expanded so as to account for the size of the free-flying vehicle.

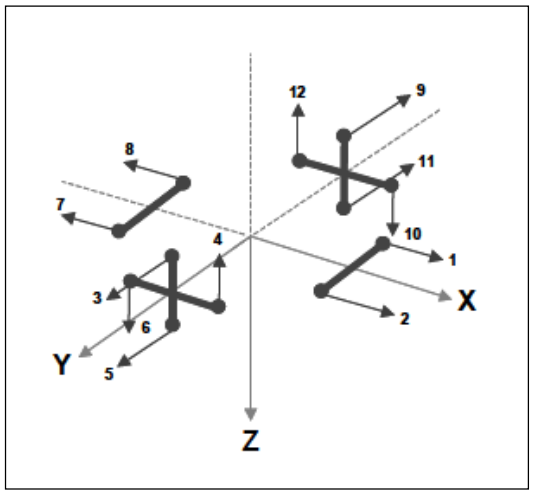


Figure 4-2 Vehicle thruster configuration [2]

Now with the vehicle and thruster model established, the error introduced into the motion planning algorithm by the impulsive maneuver approximation can be analyzed. Recalling Eq. 56 and Eq. 57, the vehicle mass, and the vehicle's maximum thrust, a maximum finite thruster firing time of 10 seconds per maneuver is allowed in these experiments. The maximum ΔV with these parameters as given by Eq. 57 is 0.8 m/s. The maximum impulsive ΔV error is then computed to be 4.0333×10^{-4} m/s, which is 0.05% of the maximum ΔV and therefore deemed an appropriate approximation. However, the maximum thruster fire time was chosen somewhat arbitrarily in this work and further analysis should be performed to find the acceptable error region. If a higher error is acceptable, it would allow for a higher maximum ΔV and more maneuver intensive trajectories.

4.3 Experimental Setup

The experiments were run on the LionX Linux computing clusters at Penn State. Due to the stochastic nature of the algorithm, multiple runs had to be performed for each parameter setting to confirm algorithm characteristics. For each parameter setting, the algorithm was run 11

times for a set number of iterations. Each group of 11 runs with the same parameter settings will be referred to as a set of runs.

Three algorithm characteristics were experimented with during these runs. They were impingement prevention, the propellant vs. time scale factor, and the use of RRT* or RRT*-ES. When impingement prevention is turned off, any thruster impingement is ignored. This setting would be acceptable for systems with cold-gas thrusters, or other thrusters which do not use a volatile propellant. The propellant vs. time scale factor determines how much importance the planner puts on propellant use and time of flight in the optimization procedure. This parameter was set to either heavily weight propellant use, or heavily weight time of flight. If the mission being planned is time critical, the user may heavily weight time of flight, however if the mission requires the inspection of many areas of the target spacecraft, the user may heavily weight propellant use. Finally, either the RRT* or RRT*-ES algorithm was used for planning. This was done so as to enable an analysis of the effectiveness of the CMA-ES local optimizer in the hybrid algorithm.

All parameter combinations were used to plan a mission in proximity to the International Space Station. The parameter combinations were used for three different start and goal combinations. The start and goal state combinations can be found in Table 4-1, from here referred to as simulations *A*, *B*, and *C*. Simulation *A* was run for 5,000 iterations, while simulation *B* and *C* were run for 10,000 iterations. Each combination is shown in Figure 4-3, Figure 4-4, and Figure 4-5 respectively, where the start state is depicted as a red circle and the goal state is depicted as a blue circle. Time is not included in Table 4-1, but the first state always starts at a time of 0, and the goal state time is a variable as discussed in Chapter 3. The start and goal states were chosen to mimic realistic planning situations, as well as to challenge the planner, particularly with impingement prevention.

Table 4-1. Simulation start and goal states

Simulation		x (m)	y (m)	z (m)	u (m/s)	v (m/s)	w (m/s)
A	**Start**	0	14.85	13.7	0	0	0
	Goal	1.8	-1.8	49	0	0	0
B	**Start**	0	14.85	13.7	0	0	0
	Goal	0	-14.8	19.3	0	0	0
C	**Start**	1.8	-1.8	49	0	0	0
	Goal	-1.8	-1.8	-49	0	0	0

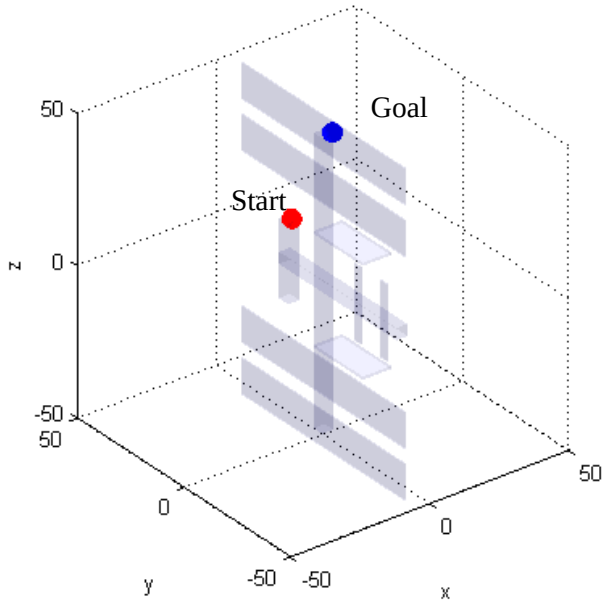


Figure 4-3 Goal and start state combination A

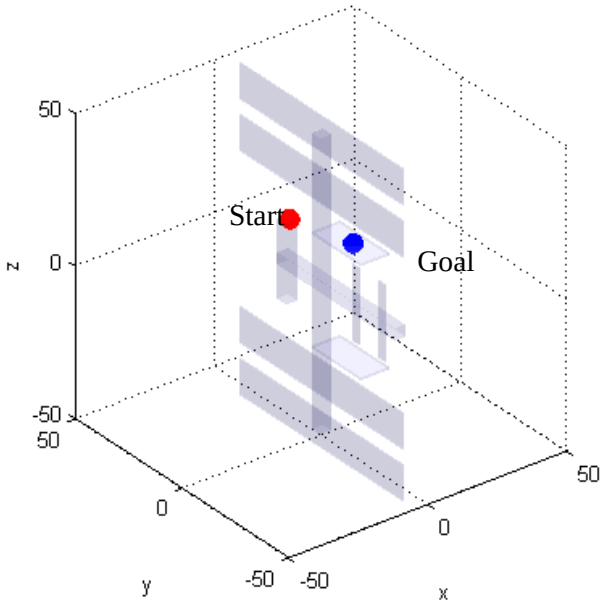


Figure 4-4 Goal and start state combination B

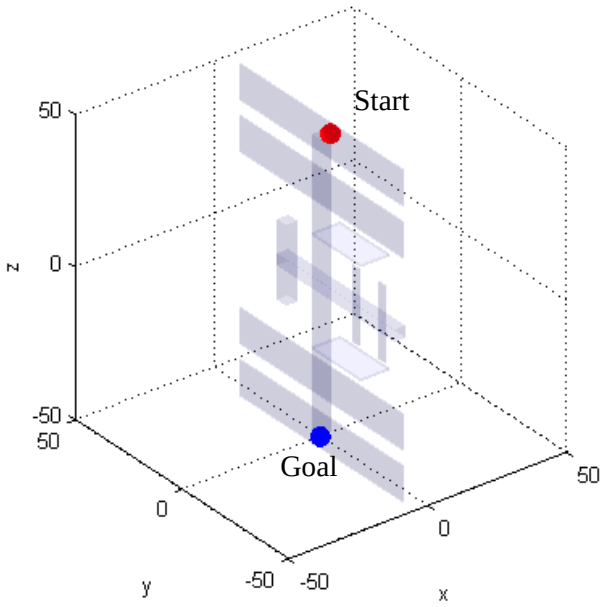


Figure 4-5 Goal and start state combination C

The bounds used for each start and goal combination can be found in Table 4-2. These bounds were developed with several considerations in mind. First, given the non-intuitive nature of the dynamics of the system considered in this work, the size and direction of the optimal trajectory is difficult to predict. However, bounds must exist for efficient search to occur. For these reason, the position bounds were made to be fairly large, and roughly centered at the midway point of the line between the start and goal points. The velocity bounds were set so as to reduce the risk of high speed impact with the target spacecraft if any of the inspection vehicle's systems fail. Finally, a maximum time of flight between the start and the goal state is set based on user preferences for the flight characteristics. It allows the user to weight the cost function so as to minimize propellant use, but do so with some time limit for the flight. In this work it was set to 1000 seconds after experimentation showed the trajectories found by the planner had times of flight less than 1000 seconds.

Table 4-2. Simulation bounds

Simulation		x (m)	y (m)	z (m)	u (m/s)	v (m/s)	w (m/s)	t (s)
A	**Minimum**	-40	-40	-10	-1	-1	-1	0
	Maximum	40	50	70	1	1	1	1000
B	**Minimum**	-40	-40	-50	-1	-1	-1	0
	Maximum	40	50	70	1	1	1	1000
C	**Minimum**	-50	-50	-70	-1	-1	-1	0
	Maximum	50	50	70	1	1	1	1000

As presented in Section 2.2.3, a scaling factor γ is required within RRT*'s `NearVertices` function. It is required to scale the radius of the "ball" for which connections are attempted between a given state and tree vertices. The γ's for simulations in this work were determined experimentally. Short simulations were performed where the average connection cost between vertices was determined for a particular Propellant vs. Time Scale Factor. The `NearVertices` scale factor was then chosen in an attempt to achieve 15 to 20 connection attempts per iteration.

The `NearVertices` scale factors are presented in Table 4-3. The γ's were required to be different for each Propellant vs. Time scale factor because the Propellant vs. Time scale factor greatly influences the cost to travel between nodes. Recalling that the proximity of nodes is determined by the cost to travel between them, it follows then that the `NearVertices` scale factor must change when the Propellant vs. Time scale factor changes.

Table 4-3. `NearVertices` **scale factors**

Simulation	Prop. vs. Time Scale Factor 1,000	Prop vs. Time Scale Factor 10,000
A	3,000	30,000
B	3,000	30,000
C	4,000	40,000

The final set of parameters are for the CMA-ES adaptation. The general requirements for these parameters were specified in Section 3.2.1. The parameters for the simulations performed in this work were chosen after brief experimentation, and with an understanding of the underlying functionality of CMA-ES. The child, or adaptation, population was set to 10 to encourage quick

search. The parent population was set to 1 to encourage quick search and convergence. As stated in Section 3.2.1, the initial step size, or standard deviation, was set to be half the size of the search space to help encourage thorough search. The initial mean was set to be the mean of the search space bounds for each simulation. The initial covariance was set to be a diagonal matrix of ones, with the covariance for the velocities scaled appropriately. Finally step size reset was set to occur at a step size less than 5, and complete reset was set to occur at a step size less than 5 and difference in population mean less than 10.

This experimentation does not provide for an exhaustive parameter analysis, but does allow for some conclusions on how the parameters impact the results. Chapter 5 presents the experimental results and discusses their implications toward autonomous flight in the vicinity of complex space structures.

Chapter 5
Results and Discussion

The experimentation performed for this thesis demonstrates the trade-offs between the RRT* algorithm and the RRT*-ES algorithm, as well as the effect of the parameters outlined in Chapter 4. The results are presented as averages over each set of runs, along with confidence intervals. The confidence intervals presented are the 95% confidence interval over a normal distribution. These confidence intervals are required for proper analysis due to the stochastic nature of the algorithms. In addition Multifactor Analysis of Variance and Tukey's procedure were applied to the run data to obtain insight into the effect of a single parameter on algorithmic properties. These statistical methods were taken from [7], and implemented via the MATLAB functions "anovan," and "multcompare."

In the following sections, graphical results are presented first to help demonstrate the general behavior of the algorithm when operating with different parameter combinations. Then the trajectory cost, convergence index, ΔV requirements, time of flight, and computational run time of the algorithms are compared. Comparisons are made between all parameter combinations, and conclusions on the effectiveness of the algorithms are drawn.

5.1 Graphical Results

To facilitate the analysis performed while comparing trajectory properties, it is important to have an understanding of the types of trajectories that are planned, and how search proceeds with different parameter combinations. For these purposes, a selection of runs will be presented in this section which best demonstrate algorithmic properties. The remainder of figures for the runs are located in Appendix D.

Figure 5-1 is presented to demonstrate the difference in search characteristics between RRT* and RRT*-ES. In image 1 and 2 of the figure, the lowest cost trajectories for a set of runs holding all parameters equal except the algorithm type are depicted for simulation *C*. In image 3 and 4 of the figure, the location of every vertex in the search trees which found the lowest cost trajectories of image 1 and 2 are depicted. The parameter settings for these trajectories are Impingement Prevention on, and a Propellant vs. Time Scale Factor of 10,000.

Some of the characteristics of the search performed to find the trajectories in image 1 and 2 can be established from image 3 and 4. In image 4, when CMA-ES is not used, vertices are dispersed through the search space evenly. However in image 3, when CMA-ES is used, there is a concentration of tree vertices around the vertices of the optimal trajectory. This demonstrates CMA-ES's local optimization capabilities. The CMA-ES algorithm embedded within RRT*-ES has located an area of the search space that contains low cost solutions, and therefore RRT*-ES has thoroughly searched that space. The result is that the trajectory in image 1 is in fact a lower cost trajectory than that of image 2.

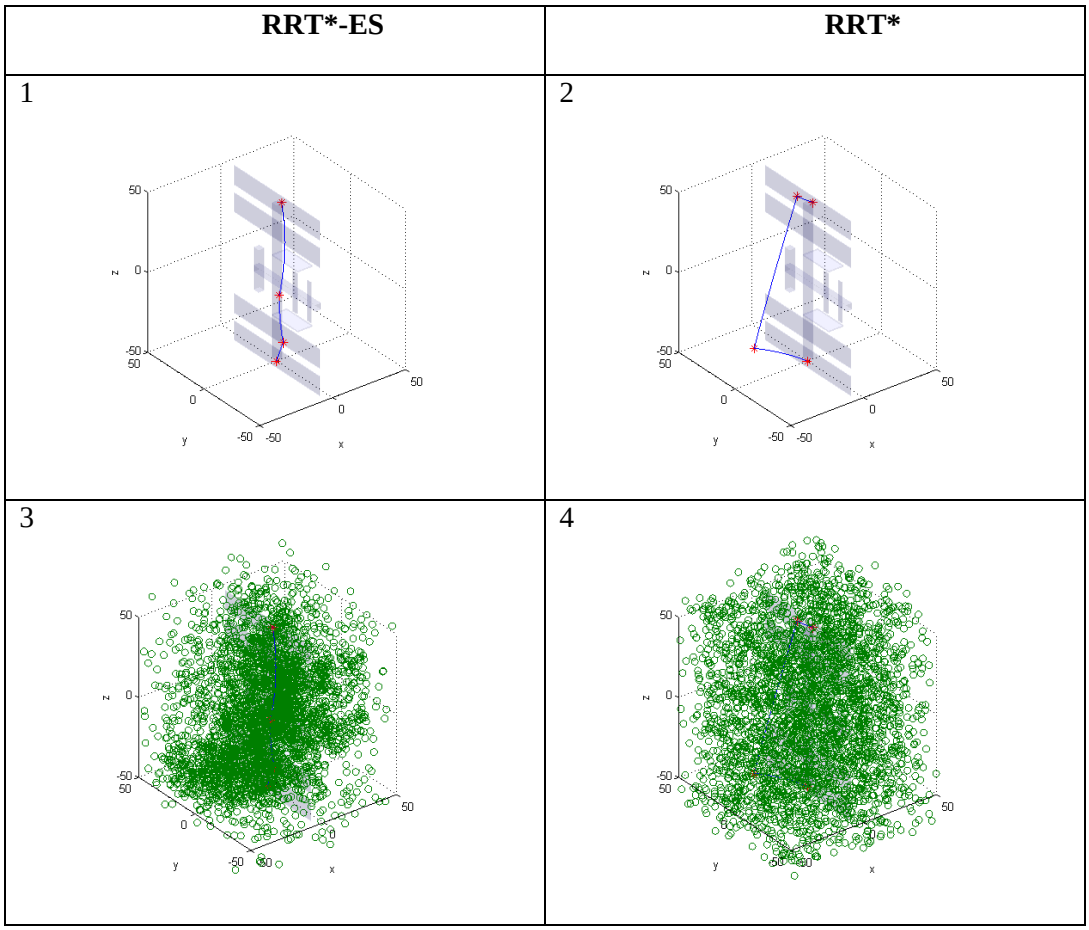


Figure 5-1 Search characteristics of HCW RRT* and HCW CMA-ES RRT*

Although RRT*-ES finds lower cost trajectories than RRT*, it is inferior to RRT* in other ways. Referring to Figure 5-1 again, remember RRT*-ES focuses search on areas of the search space where low cost nodes have previously been found. As a result of this characteristic, at every iteration there will be many tree vertices in proximity to the newly selected search node. Therefore more connections are attempted within the algorithm, and more computation time is required.

Figure 5-2 and Figure 5-3 depict optimal trajectories for simulation *A* keeping all parameters constant except the use of impingement prevention. The algorithm used was RRT*-ES, and the Propellant vs. Time Scale Factor was 10,000. It is immediately clear that an

impingement free trajectory requires a less direct path from start to goal. Notice in Figure 5-2 the vehicle maneuvers in such a way as to leave its start position above a module without thrusting into the module, and approaches the goal position next to the solar panel in such a way as to not thrust into the solar panel. This difference is also reflected in the required propellant use and time of flight of the trajectories. When impingement prevention is required, a large subset of feasible trajectories from start to goal are no longer feasible, requiring more complex and costly trajectories.

Now that a basic behavioral understanding of the algorithm has been developed, the following sections will analyze performance based on numerical values.

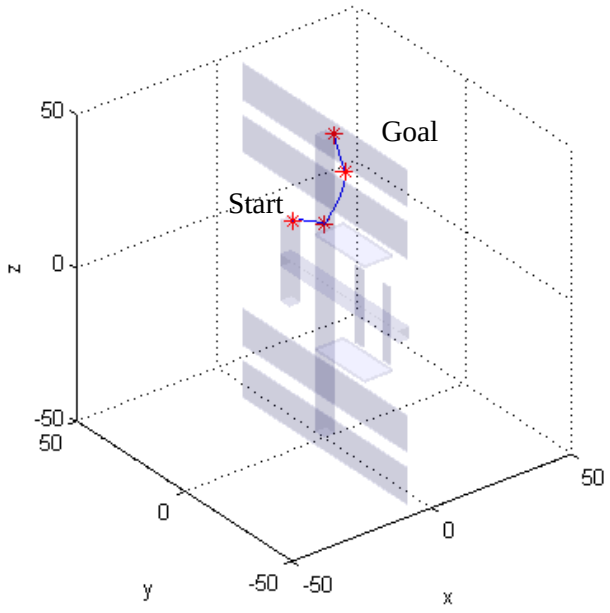


Figure 5-2 Optimal trajectory with impingement prevention

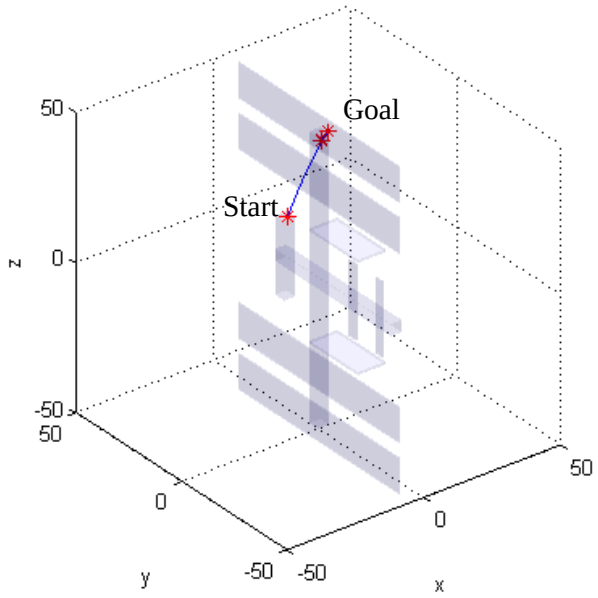


Figure 5-3 Optimal trajectory without impingement prevention

5.2 Data Tables

Data is presented in two types of tables. Table 5-1 is an example of the first type of data table. In this type of table, the entries are the average values over 11 runs with the parameter settings as presented in Table 5-1. In this case, “SF” is the Propellant vs. Time Scale Factor, “IP” is the Impingement Prevention Setting, and “Alg” is the algorithm used to solve the problem. In addition the 95% confidence interval of the average is provided in these tables.

Table 5-1. Example average table

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	SF = 1,000; IP = On; Alg = RRT*-ES	SF = 1,000; IP = On; Alg = RRT*
	Off	SF = 1,000; IP = Off; Alg = RRT*-ES	SF = 1,000; IP = Off; Alg = RRT*
10000	On	SF = 10,000; IP = On; Alg = RRT*-ES	SF = 10,000; IP = On; Alg = RRT*
	Off	SF = 10,000; IP = Off; Alg = RRT*-ES	SF = 10,000; IP = Off; Alg = RRT*

Table 5-2 is an example of the second type of data table. This type of table was assembled by applying a Multifactor Analysis of Variance (ANOVA) and Tukey's procedure on the run data. These tests were performed for a 95% confidence interval. Each row in Table 5-2 contains data comparisons when holding all parameter constant except for the parameter presented in the associated row of the first column. The second column contains the difference in the mean value of the two populations from the first column when going from the first setting to the second setting. For example in the first row, the second column entry is

$$Mean_{\ RRT^*ES} - Mean_{RRT^*} \tag{59}$$

The fourth and fifth columns contain the 95% confidence interval of the difference in means, and the sixth column contains the percent difference between the means. Again, as an example the percent difference of means in row one is calculated as

$$\frac{(Mean_{\ RRT^*ES} - Mean_{RRT^*})}{Mean_{RRT^*}} \times 100 \tag{60}$$

If the confidence interval does not change sign in a given row, the difference between means is statistically significant at a 95% confidence interval. If the difference in means does change sign in a given row, the difference is not statistically significant.

Table 5-2. Example ANOVA table

"Old Value" to "New Value"	Difference in Mean	Confidence Interval		% Difference in Mean
RRT* to RRT*-ES	$Mean_{\ RRT^*ES} - Mean_{RRT^*}$	Confidence interval lower bound	Confidence interval upper bound	$\frac{(Mean_{\ RRT^*ES} - Mean_{RRT^*})}{Mean_{RRT^*}}$
Scale Factor of 10,000 to Scale Factor of 1,000	$Mean_{\ 1{,}000} - Mean_{10{,}000}$	Confidence interval lower bound	Confidence interval upper bound	$\frac{(Mean_{\ 1{,}000} - Mean_{10{,}000})}{Mean_{10{,}000}}$
Impingement Prevention Off to Impingement Prevention On	$Mean_{\ IP\ On} - Mean_{IP\ Off}$	Confidence interval lower bound	Confidence interval upper bound	$\frac{(Mean_{\ IP\ On} - Mean_{IP\ Off})}{Mean_{IP\ Off}}$

5.3 Cost Comparison

The minimum cost found by an optimization algorithm is one of the best ways to evaluate its algorithmic success and capabilities. Regardless of the accuracy of the impulsive approximation and the HCW equations, or the appropriateness of the scaling factor, the cost achieved is still a good metric to compare the capabilities of RRT* and RRT*-ES.

The average cost for all parameter combinations of simulations *A*, *B*, and *C* are presented in Table 5-3, Table 5-4, and Table 5-5 respectively. Each entry in these tables represents the average cost over 11 runs. Table 5-6 presents the Analysis of Variance and Tukey's procedure results for the trajectory costs.

In analyzing Table 5-6, it can be seen that a change in each parameter results in a statistically significant change in the cost of the algorithm. The first comparison that will be made is between RRT*-ES and RRT*. RRT*-ES exhibits a 23% mean cost decrease as compared to RRT* across parameter combinations. This demonstrates the improved optimization capabilities of RRT*-ES. As shown in section 5.1, the CMA-ES addition to RRT* focuses search on promising areas of the space, leading to local optimization behaviors and lower cost solutions.

Table 5-3. Average cost and 95% confidence interval (A)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	1,297.6 ± 57.6	1,476.6 ± 41.7
	Off	960.3 ± 49.2	1,064.2 ± 90.6
10000	On	4,693.3 ± 301.2	6,148.2 ± 355.5
	Off	2,358.9 ± 64.7	2,535.1 ± 64.5

Table 5-4. Average cost and 95% confidence interval (B)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	1,301.3 ± 67.8	1,683.4 ± 143.8
	Off	924.2 ± 71.8	1,363.5 ± 94.5
10000	On	8,265.1 ± 2,327.7	10,482.4 ± 2,362.1
	Off	2,383.7 ± 76.4	2,795.8 ± 216.7

Table 5-5. Average cost and 95% confidence interval (C)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	1,575.8 ± 67.0	2,087.5 ± 96.5
	Off	1,422.0 ± 28.4	1,549.8 ± 37.5
10000	On	8,088.3 ± 609.8	13,081.5 ± 1,173.3
	Off	5,503.0 ± 155.3	5,819.2 ± 167.0

Table 5-6. ANOVA test on cost and 95% confidence interval

Parameter	Difference in Mean	Confidence Interval		% Difference in Mean
RRT* to RRT*-ES	-942.8	-1,242.48	-643.1	-23%
Scale Factor of 10,000 to Scale Factor of 1,000	-4,620.7	-4,920.35	-4,321.0	-77%
Impingement Prevention Off to Impingement Prevention On	2,625.1	2,325.43	2,924.8	110%

The use of impingement prevention within the algorithm also has a significant effect on the cost of solutions. When impingement prevention is used, there is a 110% increase in cost compared to when impingement prevention is not used. As stated previously, impingement prevention imparts significant constraints on the planning algorithm, thus requiring higher cost trajectories. This being said, the algorithm was indeed able to find trajectories avoid thruster impingement on the target spacecraft in all cases.

Note that the effects of impingement prevention on the cost of a trajectory are also dependent on the start and goal states for the inspection vehicle. In simulation *A*, and simulation *B*, with a Propellant vs. Time Scale Factor of 10,000, impingement prevention more than doubles the average trajectory cost in all cases. However for simulation *C*, the effect of impingement prevention is less significant. Figure 5-4 displays the set of 11 runs for all simulations with RRT*-ES, and a scale factor of 10,000. It compares the runs with Impingement Prevention off versus Impingement Prevention On. What can be seen is that for simulations *A* and *B*, the planned trajectories are required to change significantly when Impingement Prevention is on, however this is not as true in simulation *C*. The required change is reflected in the trajectory costs.

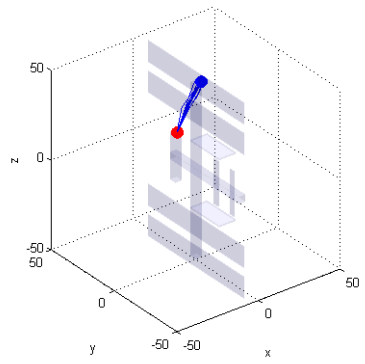
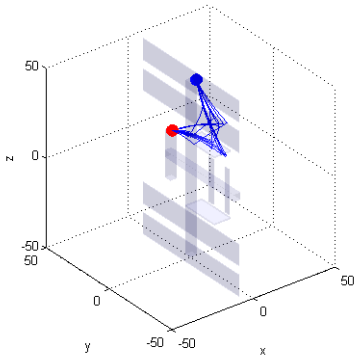
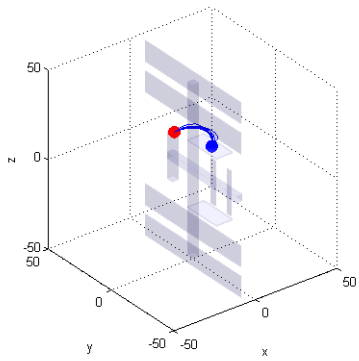
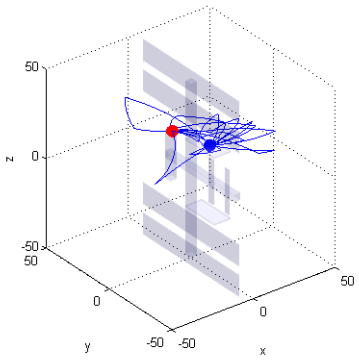
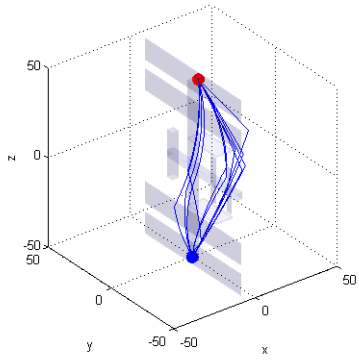
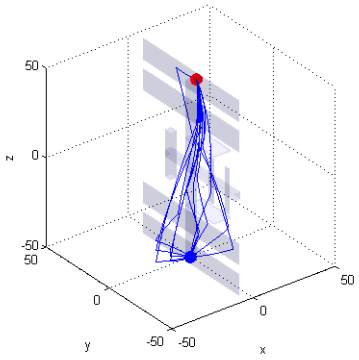
Impingement Prevention Off	Impingement Prevention On
A 	A 
B 	B 
C 	C 

Figure 5-4 Run sets with Impingement Prevention off vs. Impingement Prevention on

It is also worthwhile to analyze the cost coefficient of variation for each set of runs. The coefficient of variation gives a normalized value which can be used to compare convergence of sets of runs. A lower coefficient of variation represents a more consistent algorithm. Table 5-7, Table 5-8, and Table 5-9 display the coefficients of variation for simulations *A*, *B*, and *C* respectively.

When Impingement Prevention is off, and the Propellant vs. Time Scale Factor is 10,000, the coefficient of variation is consistently lower than when Impingement Prevention is on, demonstrating better convergence. However when the Propellant vs. Time Scale Factor is 1,000, the difference in Coefficient of Variation is more simulation dependent. This suggests that when time of flight is more heavily weighted for a given trajectory, impingement prevention may not be as significant of a factor in terms of convergence across runs, as compared to when ΔV is more heavily weighted. In fact, in some cases it would seem that in decreasing the feasible areas of the search space, impingement prevention encourages the algorithm to consistently converge to similar cost trajectories.

Table 5-7. Average cost coefficient of variation (A)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	0.0661	0.0420
	Off	0.0763	0.1267
10000	On	0.0955	0.0861
	Off	0.0408	0.0379

Table 5-8. Average cost coefficient of variation (B)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	0.0776	0.1272
	Off	0.1157	0.1031
10000	On	0.4192	0.3354
	Off	0.0477	0.1154

Table 5-9. Average cost coefficient of variation (C)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	0.0633	0.0688
	Off	0.0298	0.0360
10000	On	0.1122	0.1335
	Off	0.0420	0.0427

The convergence result for a Propellant vs. Time Scale Factor of 10,000 is also reflected in Figure 5-5 and Figure 5-6. These plots display the minimum cost history at each iteration for the set of runs of RRT*-ES on simulation *B*, with a scale factor of 10,000. In Figure 5-5 Impingement Prevention was on, in Figure 5-6 Impingement Prevention was off. Again, the cause for this behavior is the difficulty of the problem when impingement prevention is turned on and ΔV is to be minimized.

Across runs there is little difference in Cost Coefficient of Variation between RRT* and RRT*-ES. This is a reasonable result since both algorithms rely on the core RRT* procedure. As such, both should exhibit similar convergence behaviors. The difference is that RRT*-ES converges to a lower cost solution due to its local optimization abilities.

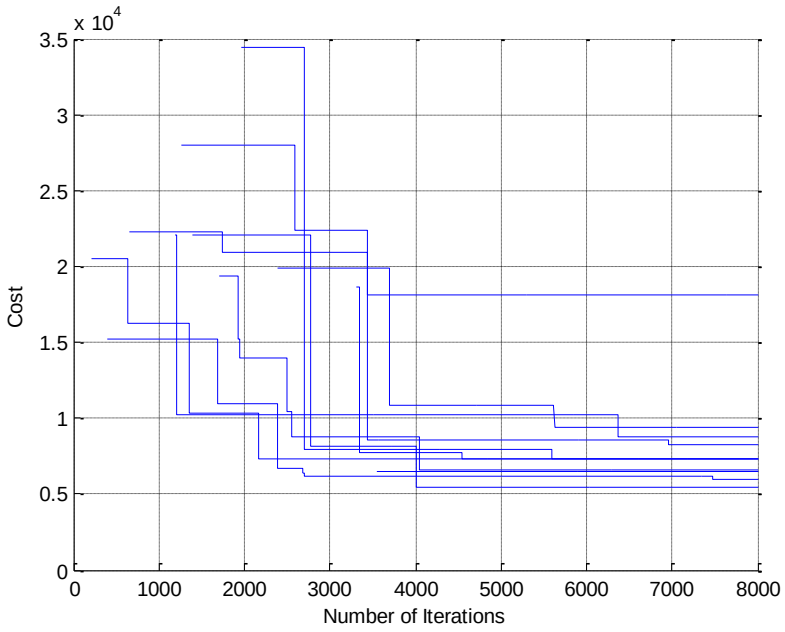


Figure 5-5 Cost vs. number of iterations, impingement prevention on

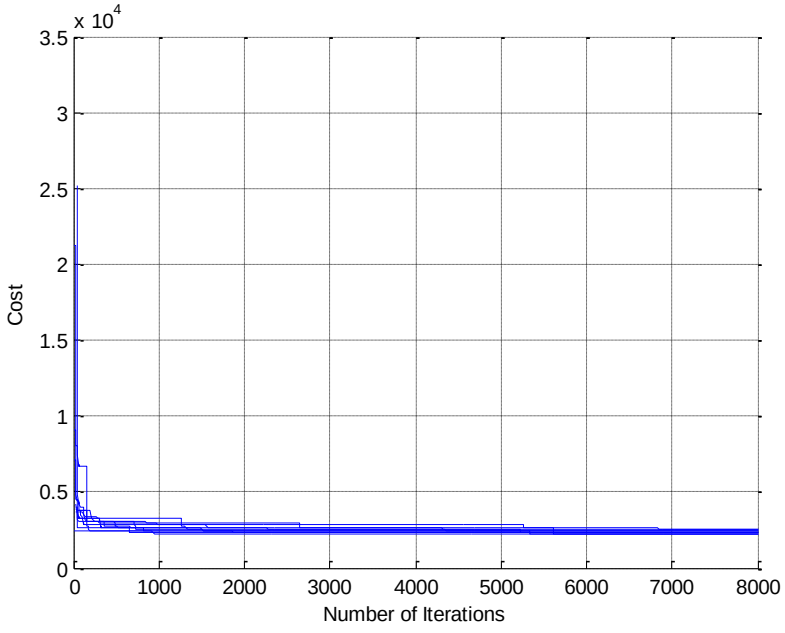


Figure 5-6 Cost vs. number of iterations, impingement prevention off

5.4 Convergence Index

In this work, the convergence index is defined as the iteration at which the lowest cost solution for a run was found. The optimal convergence index is hard to determine. Although a lower convergence index may suggest a faster running algorithm, it could also signal pre-convergence. This is when an optimization algorithm gets stuck in a local minimum which hinders its ability to continue search for the global optimum. The stochastic nature of the algorithms used in this experimentation should help to prevent search from getting stuck in a local minimum. It should also be noted that a better metric of how "fast" the algorithm converges on its best solution would be the number of calls to the steering functions. Unfortunately, this was not tracked during simulations.

The average convergence index for simulations *A*, *B*, and *C* can be found in Table 5-10, Table 5-11, and Table 5-12 respectively. The Analysis of Variance and Tukey's procedure results can be found in Table 5-13.

From Table 5-13, it can be seen that across all runs the only parameter setting that results in a statistically significant difference in Convergence Index is the use of Impingement Prevention. With Impingement Prevention on, there is a 39% increase in the Convergence Index of the algorithms. Again, the added constraints impingement prevention imposes on the motion planning problem result in a much more difficult problem for the algorithms to solve. As a result, it takes more search iterations to find the optimal trajectories. Figure 5-5 and Figure 5-6 give a strong visual representation of this behavior.

Table 5-10. Average convergence index and 95% confidence interval (A)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	3,735.8 ± 1,925.0	2,820.2 ± 1,297.9
	Off	3,244.2 ± 1,969.9	3,102.6 ± 1,481.2
10000	On	4,010.9 ± 1,454.6	4,564.1 ± 1,562.9
	Off	4,252.2 ± 1,458.7	2,552.4 ± 1,566.7

Table 5-11. Average convergence index and 95% confidence interval (B)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	4,746.2 ± 1,473.5	5,315.4 ± 1,658.3
	Off	2,617.0 ± 1,686.3	1,980.9 ± 1,238.3
10000	On	4,888.9 ± 1,105.0	4,330.8 ± 1,706.9
	Off	3,497.9 ± 1,432.9	4,322.4 ± 1,597.7

Table 5-12. Average convergence index and 95% confidence interval (C)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	4,979.9 ± 1,684.6	4,798.9 ± 1,549.2
	Off	3,618.6 ± 1,432.6	4,034.4 ± 1,954.2
10000	On	5,595.1 ± 1,551.8	5,644.9 ± 1,163.1
	Off	3,414.5 ± 1,987.1	3,293.9 ± 1,614.4

Table 5-13. ANOVA test on convergence index and 95% confidence interval

Parameter	Difference in Mean	Confidence Interval		% Difference in Mean
RRT* to RRT*-ES	153.4	-416.81	723.5	4%
Scale Factor of 10,000 to Scale Factor of 1,000	-447.8	-1,017.99	122.4	-11%
Impingement Prevention Off to Impingement Prevention On	1,291.7	721.51	1,861.9	39%

5.5 Total ΔV

In this work the total impulsive ΔV required for a particular trajectory is considered analogous to the amount of propellant that is required for a particular trajectory. As such, it is one of the two trajectory parameters minimized within the cost function. It is also important to recall that the Impingement vs. Time Scale Factor is applied directly to ΔV in the cost function, Eq. 47. The predicted behavior was that when the Impingement vs. Time Scale Factor increased, holding all other parameters constant, the trajectory ΔV would decrease and the time of flight would increase. On average across parameter combinations this result held true for trajectory ΔV. Analysis of Variance and Tukey's procedure results are displayed in Table 5-14. When the Propellant vs. Time Scale Factor is reduced from 10,000 to 1,000, on average across runs the trajectory ΔV increases by 51%.

Table 5-14. ANOVA test on ΔV and 95% confidence interval

Parameter	Difference in Mean	Confidence Interval		% Difference in Mean
RRT* to RRT*-ES	-0.17	-0.21	-0.13	-23%
Scale Factor of 10,000 to Scale Factor of 1,000	0.27	0.23	0.31	51%
Impingement Prevention Off to Impingement Prevention On	0.40	0.36	0.44	89%

However, under certain conditions, the scaling factor did not alter the trajectory parameters as expected. The average ΔVs for each parameter combination are presented in Table 5-15, Table 5-16, and Table 5-17. In some cases when Impingement Prevention was turned on, the increased Propellant vs. Time Scale Factor would result in a slight increase in the ΔV required. The anomalies occurred in simulation *B*, as seen in Table 5-16. A thorough analysis of this behavior is relegated to future work, however there is some evidence that the challenging location of the start and goal point in simulation *B* for impingement prevention may have

hindered the capability of the algorithm to fully optimize trajectories in 8,000 iterations, with the given search bounds.

Table 5-15. Average ΔV and 95% confidence interval (A)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	0.7390 ± 0.0545	0.9053 ± 0.0850
	Off	0.4890 ± 0.0603	0.5754 ± 0.1144
10000	On	0.3835 ± 0.0383	0.5229 ± 0.0442
	Off	0.1404 ± 0.0071	0.1568 ± 0.0053

Table 5-16. Average ΔV and 95% confidence interval (B)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	0.7064 ± 0.1065	0.9196 ± 0.1616
	Off	0.4740 ± 0.0746	0.7241 ± 0.1402
10000	On	0.7808 ± 0.2693	0.9526 ± 0.2333
	Off	0.1388 ± 0.0076	0.1799 ± 0.0217

Table 5-17. Average ΔV and 95% confidence interval (C)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	0.9367 ± 0.1268	1.3641 ± 0.1152
	Off	0.7560 ± 0.0366	0.8476 ± 0.0702
10000	On	0.7942 ± 0.0820	1.2387 ± 0.1164
	Off	0.4648 ± 0.0365	0.4835 ± 0.0158

The challenge arises because in simulation *B*, the goal state is on top of one of the Space Stations solar panels, and the inspection vehicle is approaching from below. This situation essentially requires the inspection vehicle to approach the goal state by traveling on the x-y plane with no z velocity. The behavior can be seen in Figure 5-7 for a group of 11 runs, where the red circle is the start state and the blue circle is the goal state. The approach requirements for impingement prevention for this simulation causes extreme restrictions within the algorithm.

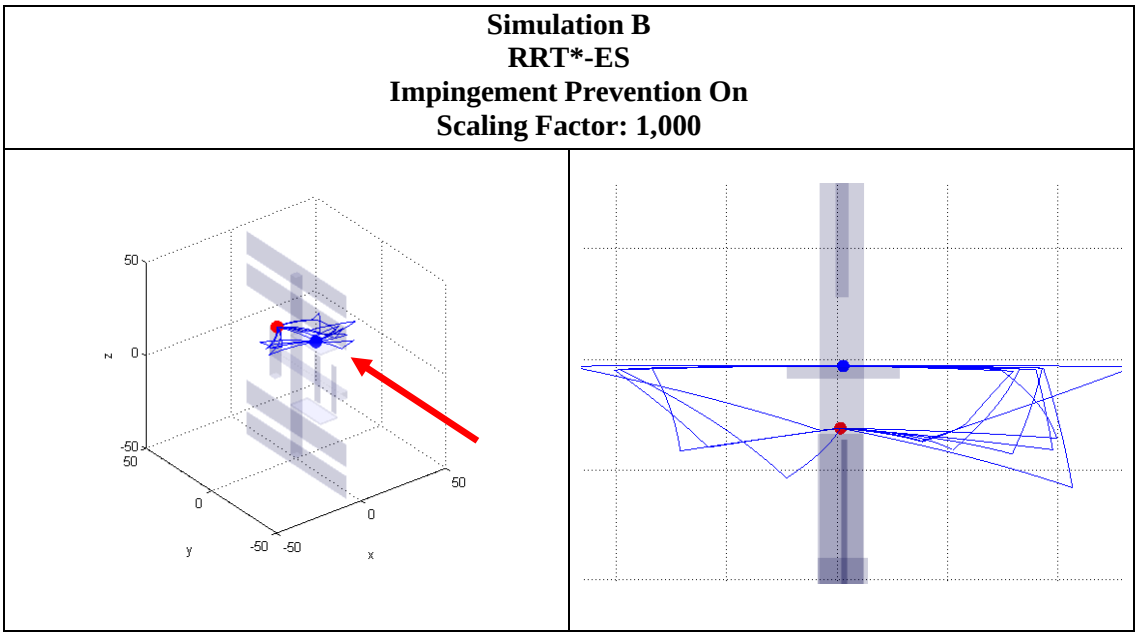


Figure 5-7 Simulation B with Impingement Prevention isometric and side view

In Figure 5-8, the optimal trajectories for the RRT*-ES runs of simulation *B* for all parameter combinations are presented. From these trajectories, some more insight can be gained for the reasons behind the ineffectiveness of the Propellant vs. Time Scale factor when Impingement Prevention is on. When the Propellant vs. Time Scale Factor is 10,000, there is good algorithmic convergence when Impingement Prevention is off, so it is a logical conclusion that these trajectories are near optimal in that image. However, these trajectories also result in impingement. In order to avoid impingement, when the scaling factor is 10,000, the trajectories are limited to trajectories similar to those with a scaling factor of 1,000. Thus, at both scaling factors the trajectories require a similar amount of ΔV. This deficiency may be relieved by opening the bounds of the search space, but further testing is required. Some further analysis of these ΔV behaviors and how they are coupled with Time of Flight behaviors is performed in Section 5.6.

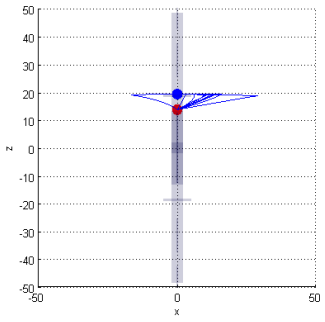
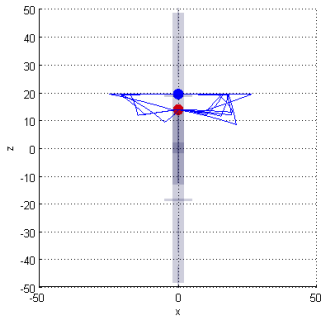
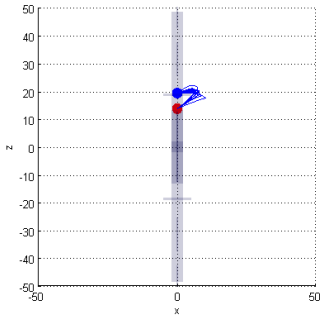
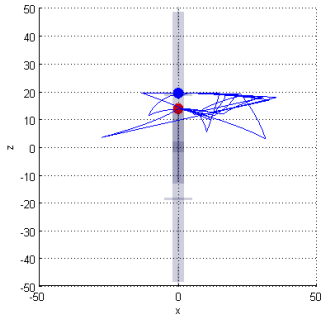
Scaling Factor	Impingement Prevention Off	Impingement Prevention On
1,000		
10,000		

Figure 5-8 Simulation B run comparison based on scaling factor

Not only does the use of Impingement Prevention influence the effectiveness of the Propellant vs. Time Scale Factor in some cases, but it also leads to a significant increase in trajectory ΔV on average. When Impingement Prevention is turned on, there is an 89% increase in ΔV across parameter combinations. Again, this is a result of the constraints put on the trajectories due to Impingement Prevention. This result gives credence to further investigation on propellant type for close proximity space missions. Although caustic propellants like hydrazine have a higher ISP than do cold gas type propellants, the required ΔV to execute safe trajectories for caustic propellants is also significantly higher. A trade must be performed when planning close

proximity missions to analyze if the added efficiency of caustic propellants outweighs the extra maneuvering they would require.

Another result from ΔV comparisons is that RRT*-ES results in a 23% decrease in ΔV. The explanation of this result follows that of the overall cost difference between RRT* and RRT*-ES from Section 5.3.

5.6 Time of Flight

The time of flight of a trajectory is the second component to the cost function which is to be minimized by the motion planning algorithm. This component to the cost function is required for multiple reasons. Even if time of flight is not a critical mission parameter, in most cases some human supervision over the mission will be required, which calls for a limited time of flight. In other cases, the time that the inspection craft can spend completing its mission may be a critical mission parameter, and thus the user is able to minimize time of flight with the correct Propellant vs. Time of Flight Scaling Factor.

Table 5-18, Table 5-19, and Table 5-20 display the average time of flight and 95% confidence interval for every parameter combination and each simulation. Table 5-21 displays the Analysis of Variance and Tukey's procedure results for time of flight across parameters. As discussed in the previous section, the main parameter that should influence the trajectory Time of Flight is the Propellant vs. Time Scale Factor. A high scale factor should result in longer flight times than a lower scale factor. As can be seen in Table 5-21, this held true. A scale factor of 1,000 resulted in a 36% lower time of flight than a scale factor of 10,000 across all other parameter combinations.

Interestingly, the anomalies that were seen in trajectory ΔV were not seen in trajectory time of flight. The response to Propellant vs Time Scale Factor was always as expected. This

result coupled with the ΔV anomalies again require further detailed investigation and experimentation which was not performed in this thesis.

From the experimentation that has been performed, it can be seen that the Impingement Prevention setting and the algorithm type had much less of an effect on the trajectory time of flight than they did on the trajectory ΔV. In the case of impingement prevention this makes sense because impingement prevention is a constraint placed directly on when a ΔV can be applied to the inspection vehicle. In the case of the algorithm used, this result suggests the cost improvement between RRT* and RRT*-ES is being driven by the trajectory ΔV. This could be a result of CMA-ES parameterization, but further experimentation is required.

An important result is that at a Propellant vs. Time Scale Factor of 10,000, the average trajectory time of flight approached the bound in all cases. It may be that for the algorithms to optimize ΔV in simulation *B* at a Propellant vs. Time Scale Factor of 10,000, the time of flight must be greater than 1,000 seconds. The convergence on seemingly non-optimal ΔVs in some of these cases may be a direct result of this bound.

Table 5-18. Average TOF(s) and 95% confidence interval (A)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	634.20 ± 59.15	591.07 ± 72.01
	Off	471.31 ± 71.76	488.85 ± 63.01
10000	On	974.42 ± 29.10	975.59 ± 49.87
	Off	955.18 ± 19.67	967.35 ± 25.08

Table 5-19. Average TOF(s) and 95% confidence interval (B)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	594.97 ± 61.39	732.28 ± 136.07
	Off	450.23 ± 46.40	639.34 ± 162.19
10000	On	942.91 ± 81.80	955.89 ± 56.60
	Off	995.60 ± 3.69	996.29 ± 2.35

Table 5-20. Average TOF(s) and 95% confidence interval (C)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	661.93 ± 104.81	748.80 ± 101.52
	Off	694.50 ± 35.14	712.81 ± 37.77
10000	On	914.97 ± 73.36	986.14 ± 17.83
	Off	995.99 ± 1.97	984.17 ± 20.01

Table 5-21. ANOVA test on TOF(s) and 95% confidence interval

Parameter	Difference in Mean	Confidence Interval		% Difference in Mean
RRT* to RRT*-ES	-41.03	-66.09	-15.97	-5%
Scale Factor of 10,000 to Scale Factor of 1,000	-352.02	-377.07	-326.96	-36%
Impingement Prevention Off to Impingement Prevention On	30.13	5.07	55.19	4%

5.7 Computational Run Time

This section presents the run time for the algorithms under all parameter combinations for the completion of 8,000 iterations. This data gives some more insight into the computational demands that the different algorithms and settings put on a system. However, it should be noted that a stronger result would be the run time until convergence, but this data was unfortunately not collected in this work.

Table 5-22**,** Table 5-23**,** and Table 5-24 display the average run time in seconds along with the 95% confidence interval across all parameter settings and simulations. Table 5-25 displays the results of the Analysis of Variance and Tukey's procedure on the data. It is

immediately clear that the choice of parameter settings has a strong effect on the computation run time of the algorithms.

RRT*-ES requires 77% more computation time than RRT*. This results directly from the way that the algorithms search the space. Remember RRT*-ES will converge search on an area of low cost solutions. The consequence of this convergence is that when a new node is sampled from the space, it is likely that there will be many tree vertices close by. As such, the `NearVertices` function presented in Section 2.2.3.1 is likely to return a larger number of tree vertices, which means there will be more calls to the steering functions and collision detection. These are both among the more computationally costly functions in the algorithm. For RRT*, sampling occurs more evenly through the search space, and thus there tend to be less near vertices than for RRT*-ES. This means less calls to the steering functions and collision detection, and less computation time.

Decreasing the Propellant vs. Time Scale factor resulted in a 60% decrease in average run time. The likely cause of this decrease is parameterization within `NearVertices`. The `NearVertices` function requires a scaling factor γ which determines how large the "ball" around a node is for determining which vertices are considered to be near enough for connection attempts. A large γ will attempt many connections around each node, theoretically resulting in lower cost paths to a particular node, but also larger run times. A small γ will not attempt as many connections around each node, theoretically resulting in higher cost paths to a particular node, but also smaller run times. It is hard to say with certainty without further testing whether γ was appropriately sized for either Propellant vs. Time Scale Factor. However it seems clear from the data that the γ for a Propellant vs. Time Scale Factor of 10,000 is larger in comparison to average connection costs for that setting, than the γ for a Propellant vs. Time Scale Factor of 1,000.

Finally, the use of Impingement Prevention resulted in a 29% decrease in computation time as compared with when Impingement Prevention was off. This is an interesting result because all of the data that has been collected points to the use of Impingement Prevention creating a more difficult problem to solve. However a longer run time with Impingement Prevention off and a less difficult problem are not mutually exclusive. When Impingement Prevention is off, there is more search space which is considered feasible because maneuverability is less constrained. This means more vertices will be added to the search tree, more attempted connections, and a longer run time

Table 5-22. Average run time (s) and 95% confidence interval (A)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	7,398.7 ± 323.6	3,163.9 ± 127.0
	Off	7,712.4 ± 231.1	3,659.1 ± 107.5
10000	On	16,426.8 ± 1,270.9	8,722.6 ± 521.2
	Off	18,095.1 ± 673.8	11,550.2 ± 482.6

Table 5-23. Average run time (s) and 95% confidence interval (B)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	7,056.3 ± 240.2	3,064.7 ± 207.3
	Off	7,529.2 ± 265.9	3,466.5 ± 57.3
10000	On	16,790.0 ± 1,026.2	8,243.1 ± 268.4
	Off	17,817.0 ± 1,094.2	10,897.3 ± 501.5

Table 5-24. Average run time (s) and 95% confidence interval (C)

Propellant vs. Time Scale Factor	Impingement Prevention	RRT*-ES	RRT*
1000	On	18,555.8 ± 1,650.3	9,504.8 ± 1,497.8
	Off	26,244.9 ± 1,313.7	13,438.2 ± 334.0
10000	On	40,978.7 ± 4,411.6	21,942.3 ± 3,332.8
	Off	65,065.1 ± 2,705.4	43,703.2 ± 1,127.9

Table 5-25. ANOVA test on run time (s) and 95% confidence interval

Paramater	Difference in Mean	Confidence Interval		% Difference in Mean
RRT* to RRT*-ES	9,026.2	8,307.27	9,745.1	77%
Scale Factor of 10,000 to Scale Factor of 1,000	-14,119.7	-14,838.64	-13,400.8	-60%
Impingement Prevention Off to Impingement Prevention On	-5,610.9	-6,329.77	-4,892.0	-29%

Chapter 6
Conclusions and Future Work

The results presented and analyzed in Chapter 5 give a good basis for conclusions to be drawn about the capabilities of the RRT*-ES algorithm developed in this thesis, as well as the effect of various user input parameters on the algorithm. With this being said, there are also several areas where further analysis could result in a more in depth understanding of the effectiveness of the algorithm.

6.1 Conclusions

In this thesis the RRT*-ES algorithm was developed as an offline algorithm which would exhibit better performance than the traditional RRT* algorithm for close proximity spacecraft motion planning. Toward this end, the RRT* algorithm was hybridized with the Covariance Matrix Adaptation Evolutionary Strategy to improve local optimization behaviors. In addition, the planned trajectories were governed by the Hill-Clohessy-Wiltshire Equations for relative motion of a chase spacecraft in relation to a target spacecraft in a circular orbit. An impingement prevention capability was also added to the motion planner for use when thruster plume impingement could result in damage to the target spacecraft.

Through experimentation, the effects of the algorithm used to perform the motion planning task (RRT* or RRT*-ES), the Propellant vs. Time Scale Factor, and Impingement Prevention were analyzed. For this analysis, trajectory cost, algorithmic convergence index, required trajectory ΔV, required trajectory time of flight, and computational run time were compared.

It was shown that RRT*-ES was consistently able to find lower cost trajectories than RRT*, due to RRT*-ES's embedded Evolutionary Strategy. RRT*-ES's superior optimization

capabilities were also reflected in trajectory ΔV and time of flight. However, there was more improvement in trajectory ΔV, suggesting RRT*-ES's ability to optimize trajectory ΔV is the source of RRT*-ES's cost reduction. The downside of the RRT*-ES algorithm is that it requires significantly more computation time to complete the same amount of iterations as RRT*. The cause of this behavior was isolated to RRT*-ES's convergence behavior which resulted in more attempted vertex connections.

When averaged across all runs, the Propellant vs. Time Scale Factor behaved as expected. When propellant use was more heavily weighted, ΔV use went down and time of flight went up. When time of flight was more heavily weighted, ΔV use went up and time of flight went down. However, there were some parameter settings that did not follow this behavior, possibly due to boundary constraints placed on the planner. Run time was also significantly affected by the Propellant vs. Time Scale Factor. However, it is believed this behavior stems from the scaling factor within the `NearVertices` function, rather than the Propellant vs. Time Scale Factor.

Finally Impingement Prevention was found to be a powerful capability which also had a significant effect on the cost of planned trajectories. In all cases when impingement prevention was desired, the algorithms were both capable of finding and optimizing to some degree an impingement free trajectory. That being said, the required ΔV for these impingement free trajectories increased. This result could help motivate the designers of future close proximity missions to consider propellants which do not require impingement prevention measures.

6.2 Future Work

Although the experimentation performed in this thesis has provided a good basis for algorithmic comparisons, there are several areas where these experiments could have been improved, and where the algorithm itself could be improved.

One of the most pressing and straightforward future efforts is a better analysis of the computation required until the RRT* and RRT*-ES algorithms converge on their final solution. Analysis was performed on the iteration at which the algorithms converge, and on the run time of the algorithms, but each experiment was lacking in key areas. The iteration of convergence, referred to as the convergence index in this work, overlooks the fact that the amount of computation within each iteration differs significantly between algorithms, and between each iteration. The run time analysis gives some estimate of the computation costs of the algorithm, and inherently takes into consideration the amount of computation within each iteration, but data was only collected for the run time after 8,000 iterations.

In the future the best way to compare the computational requirements of the different algorithms would be to track the number of times the collision detection function was called, and how many calls to this function were made before algorithmic convergence. Since the collision detection function is one of the most computationally demanding functions within the algorithms, it could give a better estimate of how much "computation" is required before convergence occurs. This information could then also be used to better understand if some of RRT*-ES's superiority is a simple function of the number of connections attempted, rather than the intelligent location of these attempted connections.

Another area of future work is a better analysis of how the Propellant vs. Time Scaling Factor and user defined search bounds interact with each other. The data collected in this work suggests that if search bounds are not chosen correctly, the Propellant vs. Time Scaling Factor

will not be effective in creating the balance between propellant use and time of flight that the user desires. A methodology to relate these two user inputs could be useful.

Along these same lines, the scaling factor within the `NearVertices` function requires further analysis. An inappropriate scaling factor results in either computationally inefficient search, or ineffective search where few connections and rewires are attempted. The methodology in this work to identify the scaling factor was to run some short experiments from which an average trajectory cost for a particular simulation could be calculated. From there, the `NearVertices` scaling factor was sized to achieve an average of 15 to 20 attempted connections every iteration. However, experimental determination of this scaling factor was not thorough, and data points to much more variety in the number of connections attempted each iteration. A methodology to relate the `NearVertices` scaling factor, γ, to the cost function and given environment could also be useful.

Along with further analysis, some general algorithmic improvements could be made to the planner. First of which could be the implementation of a more computationally efficient collision detection algorithm. The collision detection algorithm used in this work models the target spacecraft as a collection of planar obstacles, and trajectories are split up into short line segments where each line segment in a trajectory is checked for collision with each planar obstacle. This results in hundreds of millions of collision checks when running the algorithm for 8,000 iterations. As such, even small improvements in the computational complexity of the collision detection algorithm could result in significant computational savings.

Another improvement would be to switch the steering function from an impulsive maneuver steering function, to a continuous thrust steering function. In fact, an early version of the RRT*-ES algorithm utilized a modified LQR steering function based on the work of Webb and Berg[25]. The steering function and algorithm planned continuous thrust trajectories effectively,

however the continuous thrust trajectories planned did not lend themselves to efficient impingement prevention calculations. That being said, if the collision detection algorithm is improved, or impingement prevention is not a concern, continuous thrust trajectories could be useful for certain types of inspection vehicles, such as electric propulsion vehicles.

A final improvement would be a more complete thruster plume model. For this work, the thruster plume was simply modeled as a line segment originating at the vehicle and extending in the direction of thrust. The size of the line was also chosen somewhat arbitrarily. This was done to demonstrate that the algorithms could in fact handle this extra constraint. In the future it may be useful to model the rocket plume more accurately in a cone shape which is proportional in size to the amount of thrust being provided, and based on propulsion system data.

Appendix A
CMA-ES Strategy Parameters

Appendix A contains the formula for various internal strategy parameters of the Covariance Matrix Adaptation Evolutionary Strategy. These internal parameters are functions of user input parameters, as discussed in Section 2.3.

Table A-1 CMA-ES strategy parameters

Step Size Control	$c_\sigma = \frac{\mu_{eff} + 2}{n + \mu_{eff} + 5}$ $d_\sigma = 1 + 2\max\left(0, \sqrt{\frac{\mu_{eff} - 1}{n + 1}}\right) + c_\sigma$
Covariance Matrix Adaptation	$c_c = \frac{4 + {\mu_{eff}}/{n}}{n + 4 + 2\,{\mu_{eff}}/{n}}$ $c_1 = \frac{2}{(n + 1.3)^2 + \mu_{eff}}$ $c_\mu = \min(1 - c_1)$ $\alpha_\mu \frac{{\mu_{eff} - 2 + 1}/{\mu_{eff}}}{(n+2)^2 + \alpha_\mu \,{\mu_{eff}}/{2}}$ with $\alpha_\mu = 2$

Appendix B
RRT*

The RRT* algorithm is outlined in pseudocode form in Figure B-1. The algorithm begins with the initialization of the search graph *G* through the addition of vertex $V = \{z_{start}\}$ and an empty set of edges, $E = \emptyset$ (line 1). The algorithm then samples a random node z_{rand} in X_{free}, and the tree is extended from the nearest vertex, $z_{nearest}$, in tree *G* toward z_{new} (Figure B-1, lines 3 to 5). This trajectory that extends from *G* is denoted $x_{nearest,new}$, and the node that terminates this extension is denoted z_{new}. If $x_{nearest,new}$ is collision free, z_{new} is added to the set of tree vertices, V, and the cost equal to the cumulative cost to arrive at z_{new} through $z_{nearest}$ is stored, along with $z_{nearest}$ as the minimum cost parent to z_{new} found thus far (Figure B-1, line 6 to 8). Next `NearVertices` determines the set of vertices nearby z_{new}, called z_{near} (Figure B-1, line 9). Connection are attempted from all z_{near} to z_{new}. If a connection from a z_{near} to z_{new} is collision free and the cumulative cost of the connection is the lowest found thus far, the z_{near} and its cost are stored (Figure B-1, line 12 and 13). The connection that reaches z_{new} with minimum cost is stored in *E* (Figure B-1, line 14). This makes the vertex in the tree which results in the lowest cost connection to z_{new} the unique parent of z_{new}. Next a rewire attempts connections from z_{new} to all X_{near}. For all of the collision free connections that result in a lower cost member of X_{near}, z_{new} is made the parent, and the edge connecting z_{new} to z_{near} replaces the current edge connecting from the tree to z_{near} (Figure B-1, line 17 to 19). This sequence repeats until the maximum number of iterations has bene reached.

RRT*

$V \leftarrow \{z_{start}\}; E \leftarrow \{\ \};$

for $i = 1, \ldots, n$ **do**

  $z_{rand} \leftarrow \mathrm{SampleFree}_i;$

  $z_{nearest} \leftarrow \mathrm{NearestVertex}(G = (V, E), z_{rand});$

  $x_{nearest,new} \leftarrow \mathrm{Steer}(z_{nearest}, z_{rand});$

  if $\mathrm{CollisionFree}(x_{nearest,new})$ **then**

    $V \leftarrow V \cup \{z_{new}\};$

    $z_{min} \leftarrow z_{nearest};\ c_{min} \leftarrow \mathrm{Cost}(z_{nearest}) + c(x_{nearest,new});$

    $X_{near} \leftarrow \mathrm{NearVertices}\left(G = (V, E), z_{new}, \min\left\{\gamma_{RRT*}\left(\frac{\log(\mathrm{card}(V))}{\mathrm{card}(V)}\right)^{1/d}, \eta\right\}\right);$

    foreach $z_{near} \in X_{near}$ **do**

      $x_{near,new} \leftarrow \mathrm{Steer}(z_{near}, z_{new});$

      if $\mathrm{CollisionFree}(x_{near,new}) \wedge \mathrm{Cost}(z_{near}) + c(\mathrm{Line}(x_{near,new})) < c_{min}$ **then**

        $z_{min} \leftarrow z_{near};\ c_{min} \leftarrow \mathrm{Cost}(z_{nearest}) + c(x_{near,new});$

    $E \leftarrow E \cup \{(x_{near,new})\};$

    foreach $z_{near} \in X_{near}$ **do**

      $x_{new,near} \leftarrow \mathrm{Steer}(z_{new}, z_{near});$

      if $\mathrm{CollisionFree}(x_{new,near}) \wedge \mathrm{Cost}(z_{new}) + c(x_{new,near}) < \mathrm{cost}(z_{near})$ **then**

        $z_{new} \leftarrow \mathrm{Parent}(z_{near});$

        $E \leftarrow (E \backslash \{(x_{parent,near})\}) \cup \{(x_{new,near})\};$

return $G = (V, E);$

Figure B-1 RRT* pseudocode

Appendix C
RRT*-ES

The RRT*-ES algorithm is outlined in pseudocode form in Figure C-1, Figure C-2, and Figure C-3. RRT*-ES has a structure very similar to RRT*. Only the differences between RRT* and RRT*-ES will be addressed here. The first difference arise in line 5 of Figure C-1. Here the CMA-ES parameters are updated according to the equations in Section 2.3.2, if the adaptation population has reached a sufficient size.

The next change is the steering function implemented. On lines 8, 14, 22, and 26, the steering functions developed in Section 3.3.

Another change occurs on lines 9, 15, and 23. Here the impingement detection function `ImpingementFree` is called, in addition to the collision detection function `CollisionFree`.

The final change occurs with the addition of goal state connection functions, as discussed in Section 3.1. On lines 26 and 27, if a connection has been made from the current search tree to the new node, a connection is attempted with the goal via the `HCWSteerGoal` and `GoalConnect` functions. The `GoalConnect` function's pseudocode can be found in Figure C-2. This function makes the new node the parent to the goal node if $x_{new,goal}$ is feasible, and $x_{new,goal}$ is a member of the lowest cost trajectory from start to goal. If $x_{new,goal}$ is feasible, but not a member of the lowest cost trajectory from start to goal, $x_{new,goal}$ and z_{new} are stored in a goal parent structure, where z_{new} is stored in V_{goal} and $x_{new,goal}$ is stored in E_{goal}. This is done for reasons outlined in Section 3.1.

If in a future iteration a rewire occurs, the goal parent structure is revisited on line 29 of Figure C-1 by the function `GoalRewire`. The `GoalRewire` function pseudocode can be found in Figure C-3. `GoalRewire` functions by checking all members of the goal parent structure, and

making a member of the structure the unique parent to the goal node if the member of the structure is now also a member of the lowest cost trajectory from start to goal due to a tree rewire.

RRT*-ES

$V \leftarrow \{z_{start}\}; E \leftarrow \{\}; V_{goal} \leftarrow \{\}; E_{goal} \leftarrow \{\};$

$CMA \leftarrow \{\lambda, \sigma_0, recombination\ weights\}; Pop \leftarrow \{\};$

for $i = 1, \ldots, n$ **do**

  if $\mathrm{Size}(Pop) \geq \lambda$

    $CMA \leftarrow \mathrm{UpdateCMA\text{-}ES}(Pop)$

  $z_{new} \leftarrow \mathrm{SampleCMA\text{-}ES}_i;$

  $z_{nearest} \leftarrow \mathrm{Nearest}(G = (V, E), z_{rand});$

  $x_{nearest,new} \leftarrow \mathrm{HCWSteer}(z_{nearest}, z_{new});$

  if $\mathrm{CollisionFree}(x_{nearest,new}) \wedge \mathrm{ImpingementFree}(x_{nearest,new})$ **then**

    $V \leftarrow V \cup \{z_{new}\};$

    $z_{min} \leftarrow z_{nearest};\ c_{min} \leftarrow \mathrm{Cost}(z_{nearest}) + c(x_{nearest,new});$

    $X_{near} \leftarrow \mathrm{Near}\left(G = (V, E), z_{new}, \min\left\{\gamma_{RRT*}\left(\frac{\log(\mathrm{card}(V))}{\mathrm{card}(V)}\right)^{1/d}, \eta\right\}\right);$

  foreach $z_{near} \in X_{near}$ **do**

    $x_{near,new} \leftarrow \mathrm{HCWSteer}(z_{nearest}, z_{new});$

    if $\mathrm{CollisionFree}(x_{near,new}) \wedge \mathrm{ImpingementFree}(x_{near,new}) \wedge \mathrm{Cost}(z_{near}) + c(x_{near,new}) < c_{min}$ **then**

      $z_{min} \leftarrow z_{near};\ c_{min} \leftarrow \mathrm{Cost}(z_{nearest}) + c(x_{near,new});$

      if $z_{new} \notin V$ **then**

        $V \leftarrow V \cup \{z_{new}\};$

        $E \leftarrow E \cup \{(z_{min}, z_{new})\};$

  if $z_{new} \in V$

    foreach $z_{near} \in X_{near}$ **do**

      $x_{new,near} \leftarrow \mathrm{HCWSteerRewire}(z_{new}, z_{nearest});$

      if $\mathrm{CollisionFree}(x_{new,near}) \wedge \mathrm{ImpingementFree}(x_{new,near}) \wedge \mathrm{Cost}(z_{new}) + c(x_{new,near}) < \mathrm{Cost}(z_{near})$ **then**

        $z_{new} \leftarrow \mathrm{Parent}(z_{near});$

      $E \leftarrow \left(E \backslash \{(x_{parent,near})\}\right) \cup \{(x_{new,near})\};$ // Rewire

    $x_{new,goal} \leftarrow \mathrm{HCWSteerGoal}(z_{new}, z_{goal});$

    (V,E, V_{goal}, E_{goal}) $\leftarrow \mathrm{GoalConnect}(x_{new,goal}, z_{new}, z_{goal})$

    if Rewire Occurred **then**

      (V,E) $\leftarrow \mathrm{GoalRewire}(V_{goal}, E_{goal})$

return $G = (V, E);$

Figure C-1 RRT*-ES pseudocode

GoalConnect

1 **if** CollisionFree$(x_{new,goal})$ ∧ImpingementFree$(x_{new,goal})$ ∧Cost$(z_{new}) + c(x_{new,goal})$ <Cost(z_{goal}) **then**
2 z_{new} ←Parent(z_{goal});
3 $E \leftarrow (E \backslash \{(x_{parent,goal})\}) \cup \{(x_{new,goal})\}$;
4 **elseif** CollisionFree$(x_{new,goal})$ ∧ ImpingementFree$(x_{new,goal})$
5 $V_{goal} \leftarrow V_{goal} \cup \{z_{new}\}$;
6 $E_{goal} \leftarrow E_{goal} \cup \{(x_{new,goal})\}$

Figure C-2 GoallConnect pseudocode

GoalRewire

1 **foreach** $x_{pg} \in E_{goal}$
2 **if** Cost$(z_{pg}) + c(x_{pg})$ < Cost(z_{goal}) **then**
3 z_{new} ←Parent(z_{pg});
4 $E \leftarrow (E \backslash \{(x_{parent,goal})\}) \cup \{(x_{pg,goal})\}$;

Figure C-3 GoalRewire pseudocode

Appendix D
Data Figures

Appendix D contains the remaining figures produced from experiments performed in this thesis. There were 24 sets of 11 runs performed, and for each set of runs four plots were created. The four types of plots are costs plots, trajectories plots, lowest cost trajectory plots, and lowest cost trajectory plots with vertices.

For each set of runs, these four plots are put into one figure, as in Figure D-1. Each costs plot shows the cost versus the algorithm iteration for all of the 11 runs with the same parameter setting. Each trajectories plot shows all of the trajectories found by the 11 runs with the same parameter setting. Each minimum cost trajectory plot shows the minimum cost trajectory found out of the 11 runs with the same parameter setting. And finally, each minimum cost trajectory plot shows the minimum cost trajectory found out of the 11 runs with the same parameter setting, along with the vertices of the search tree that found that trajectory.

For ease of understanding and labelling of the following figures in Appendix D, each set of 11 runs which had the same parameter setting has been given a number. Table D-1 lists these set numbers, along with their associated parameter settings. The following figures are labelled so as to be associated with Table D-1.

Table D-1 Run set number and associated parameter settings

Set Number	Simulation	Impingement Prevention	Propellant vs. Time Scale Factor	Algorithm
1	A	YES	1000	RRT*-ES
2	A	YES	1000	RRT*
3	A	YES	10000	RRT*-ES
4	A	YES	10000	RRT*
5	A	NO	1000	RRT*-ES
6	A	NO	1000	RRT*
7	A	NO	10000	RRT*-ES
8	A	NO	10000	RRT*
9	B	YES	1000	RRT*-ES
10	B	YES	1000	RRT*
11	B	YES	10000	RRT*-ES
12	B	YES	10000	RRT*
13	B	NO	1000	RRT*-ES
14	B	NO	1000	RRT*
15	B	NO	10000	RRT*-ES
16	B	NO	10000	RRT*
17	C	YES	1000	RRT*-ES
18	C	YES	1000	RRT*
19	C	YES	10000	RRT*-ES
20	C	YES	10000	RRT*
21	C	NO	1000	RRT*-ES
22	C	NO	1000	RRT*
23	C	NO	10000	RRT*-ES
24	C	NO	10000	RRT*

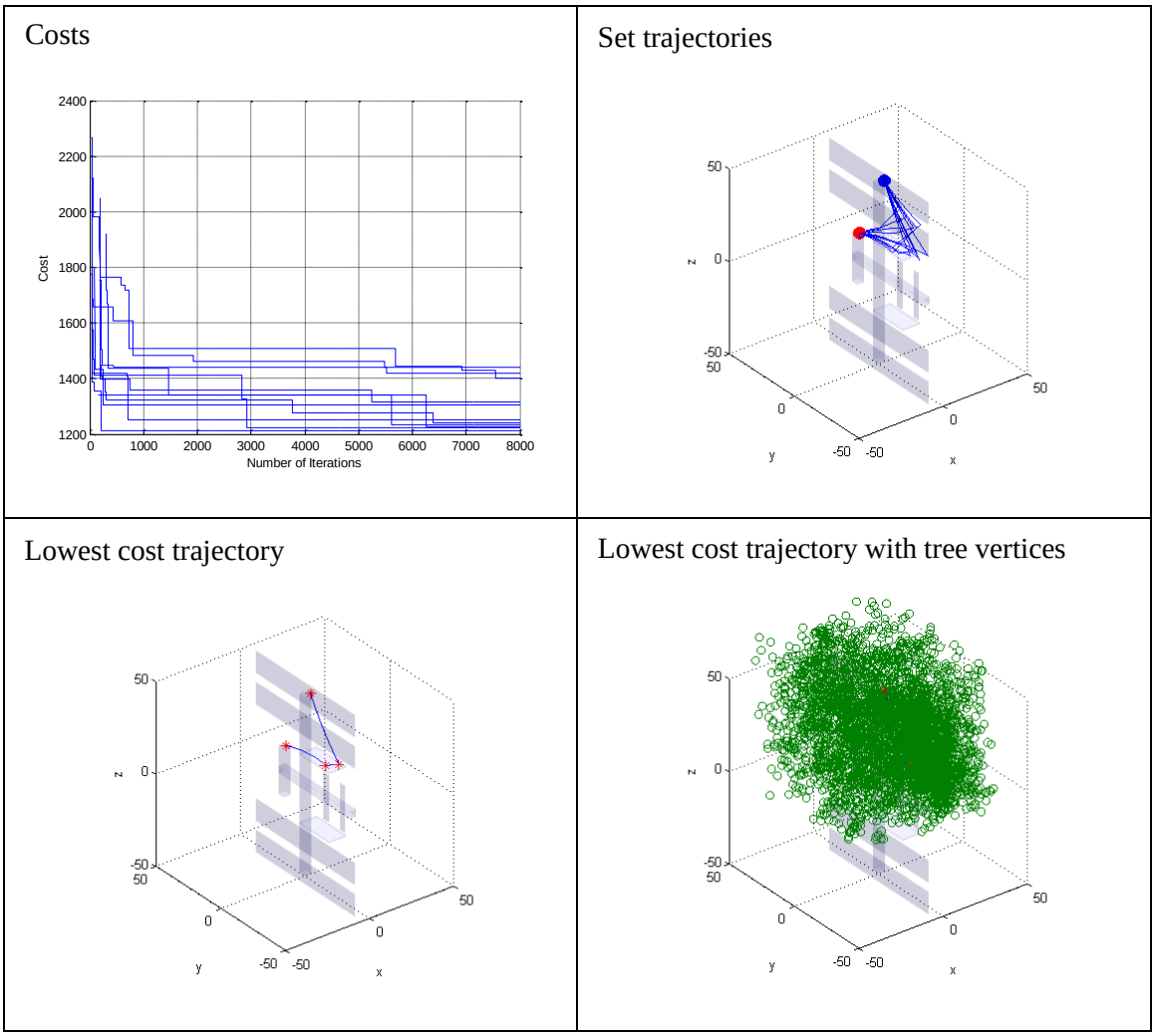


Figure D-1 Run set 1

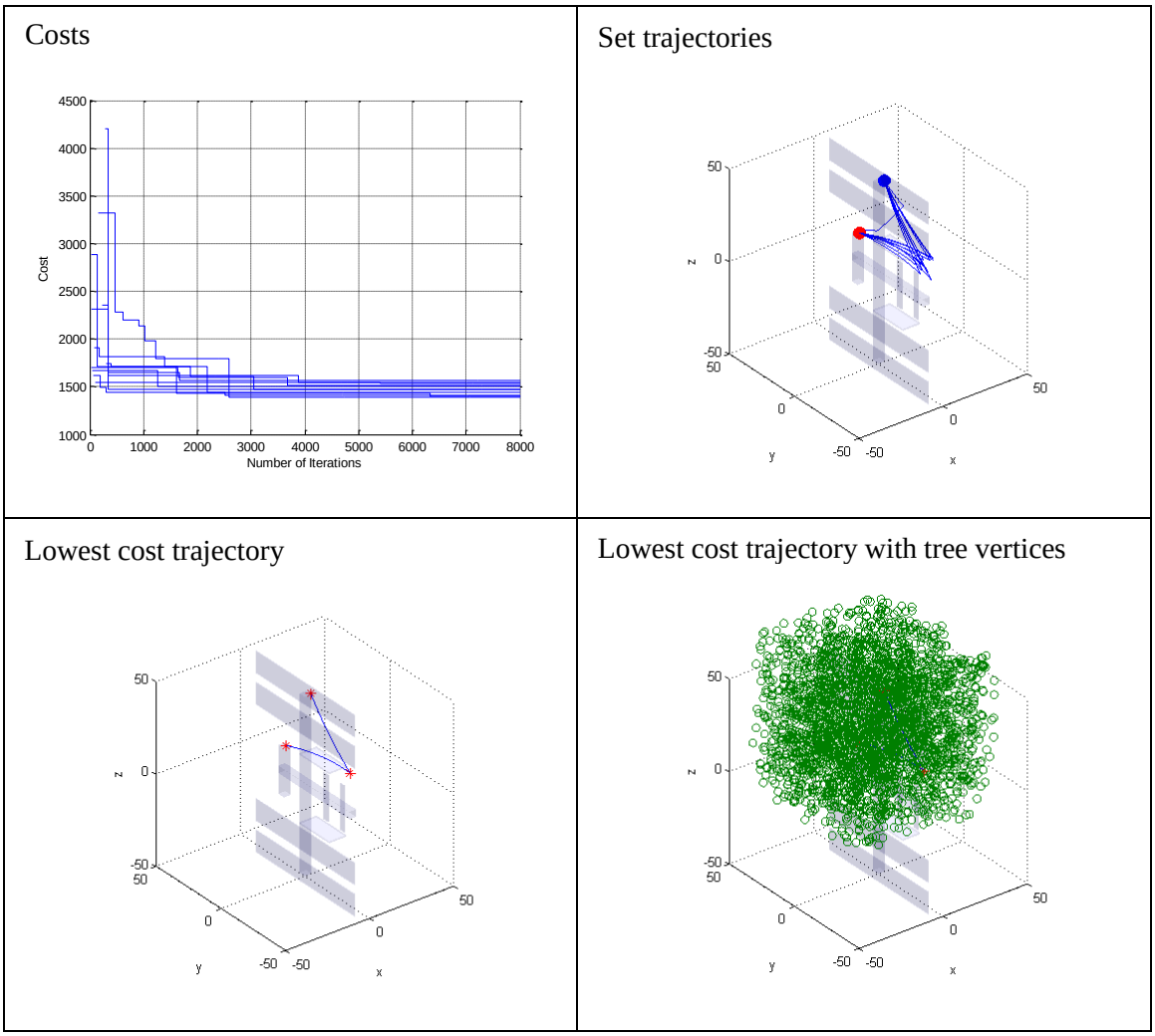
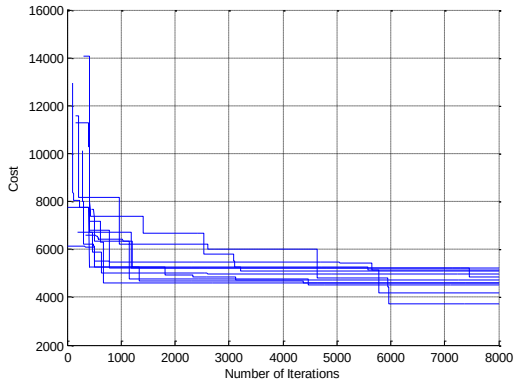
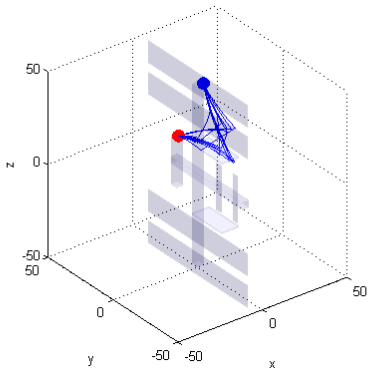


Figure D-2 Run set 1

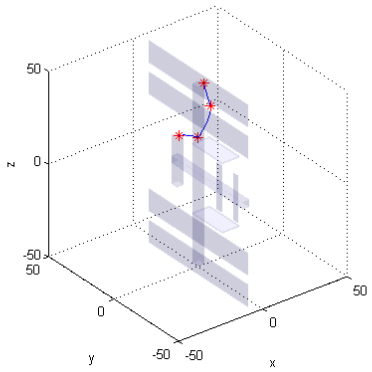
Costs



Set trajectories



Lowest cost trajectory



Lowest cost trajectory with tree vertices

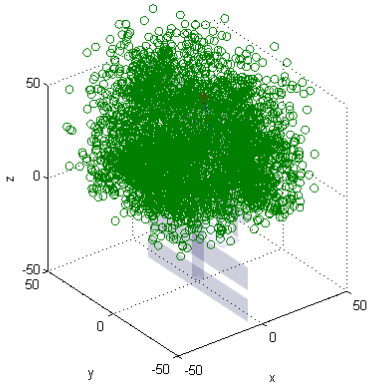


Figure D-3 Run set 3

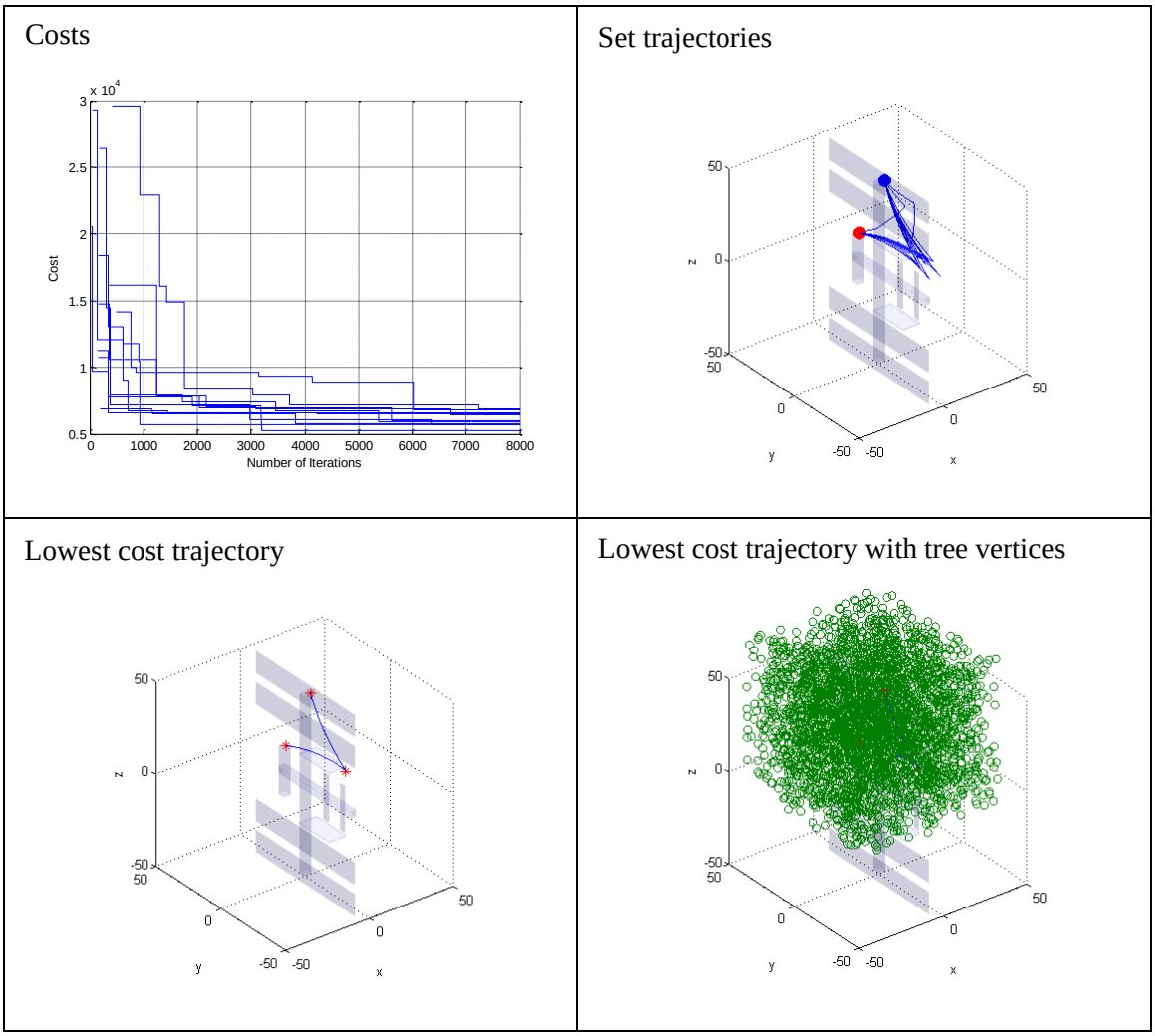


Figure D-4 Run set 4

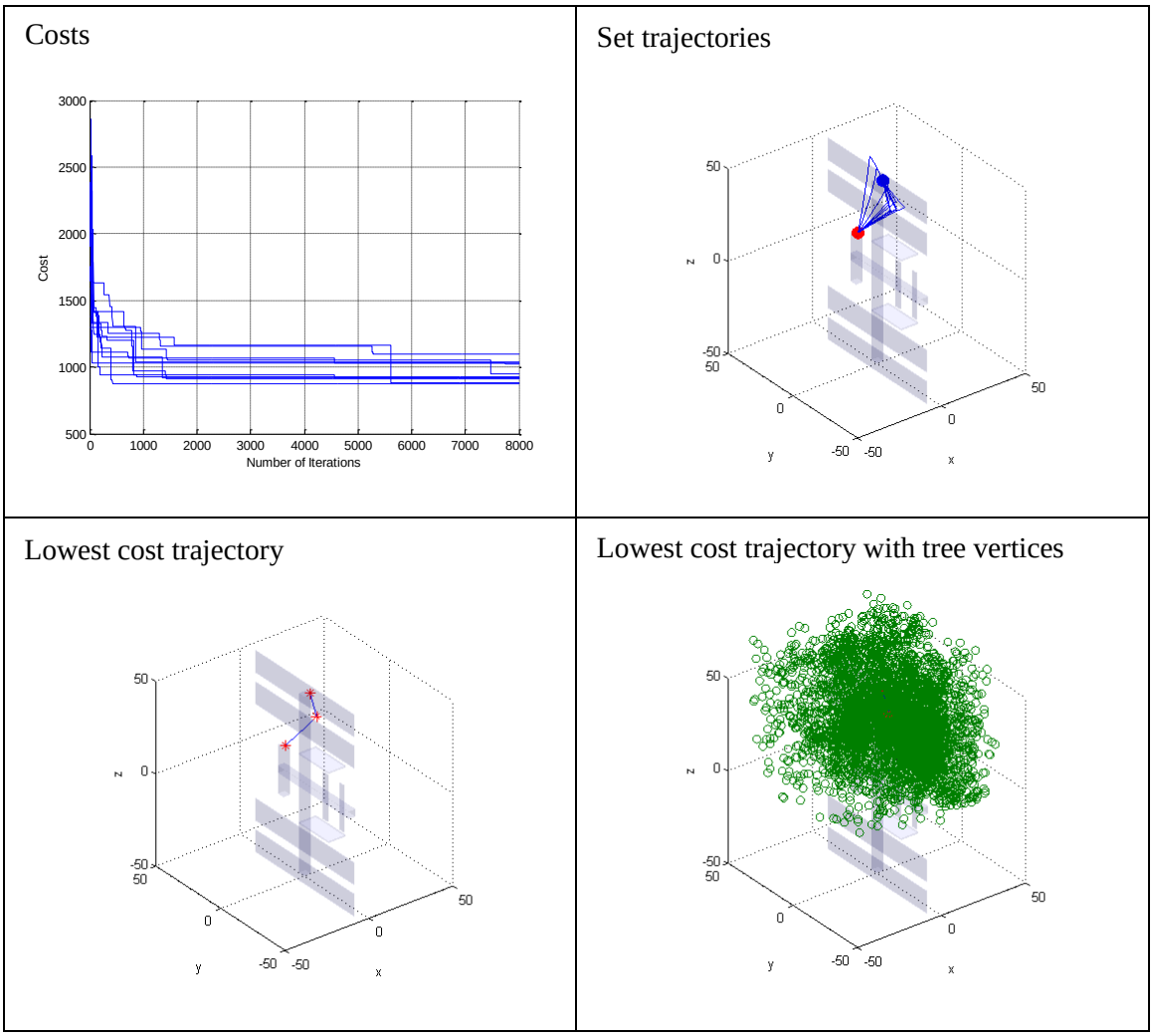


Figure D-5 Run set 5

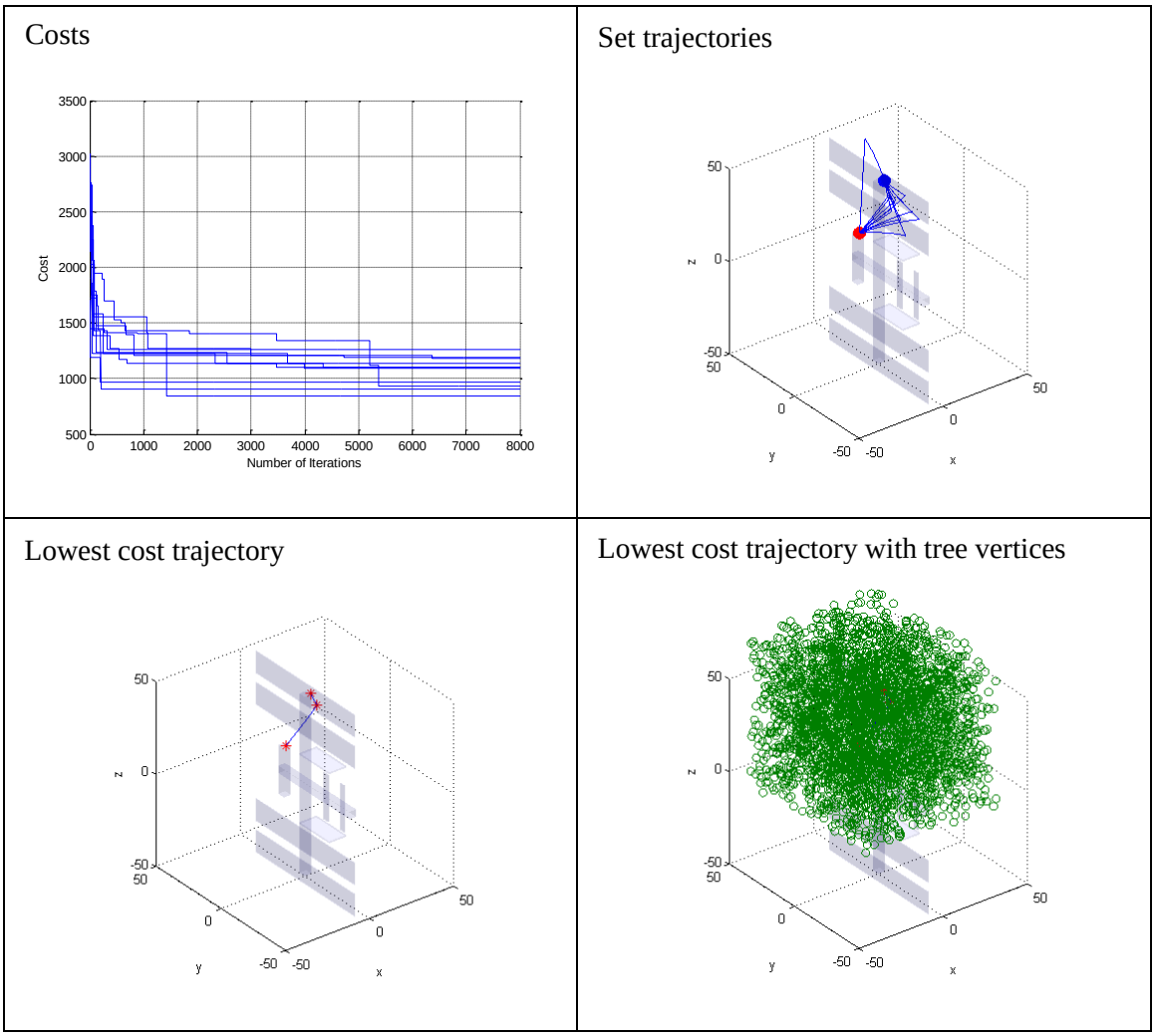
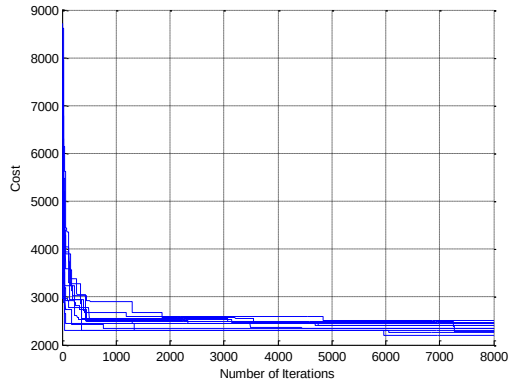
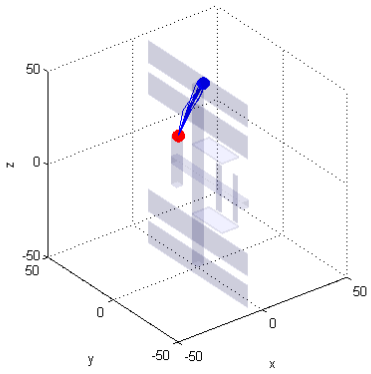


Figure D-6 Run set 6

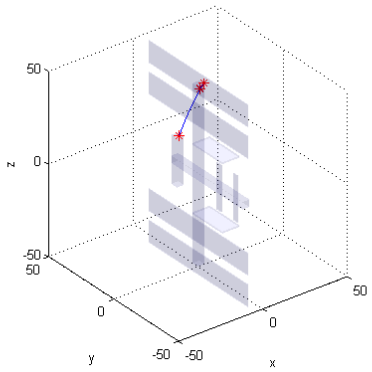
Costs



Set trajectories



Lowest cost trajectory



Lowest cost trajectory with tree vertices

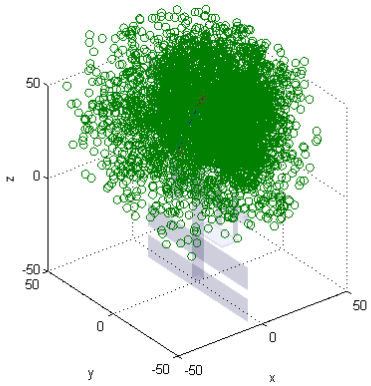


Figure D-7 Run set 7

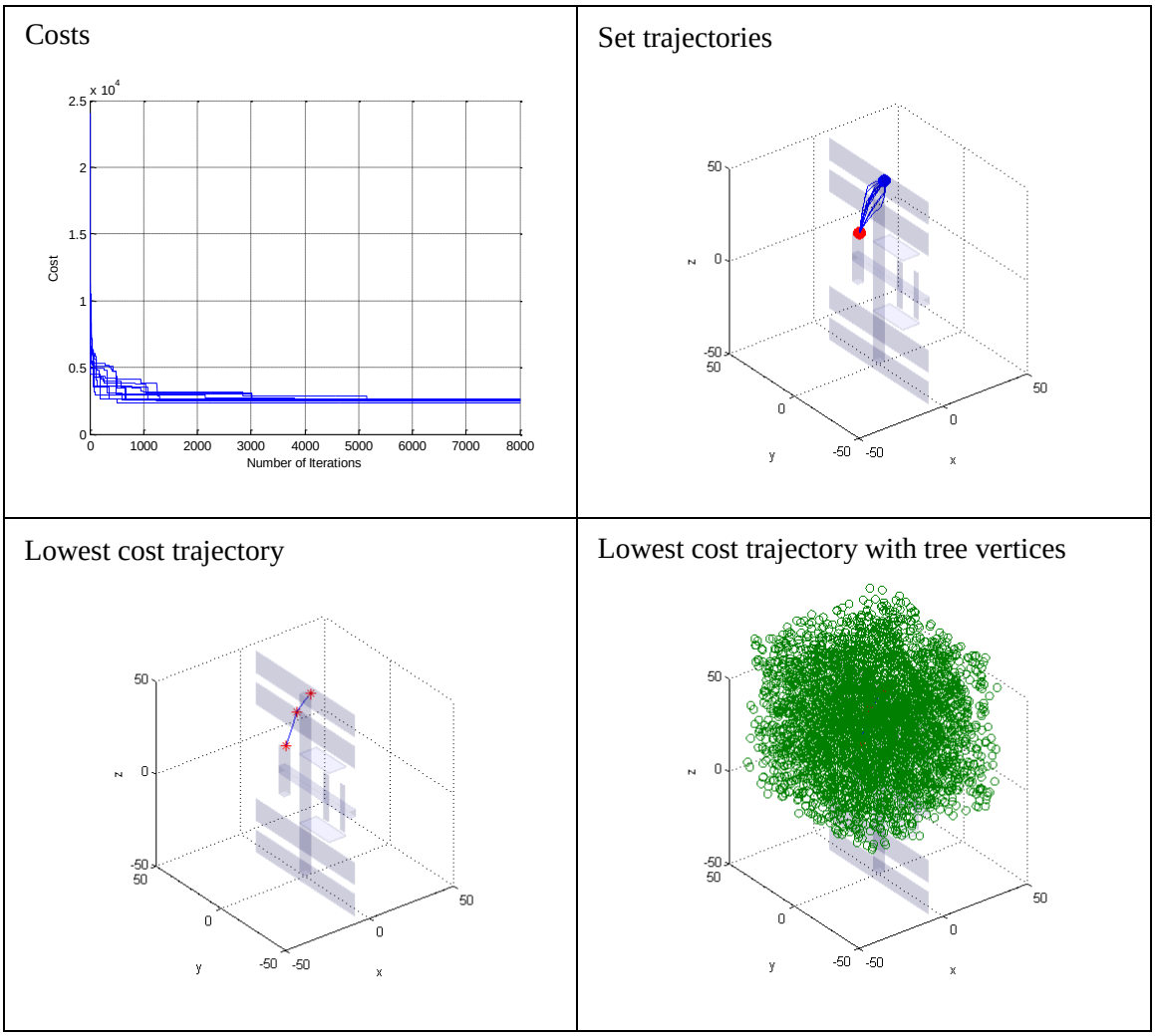


Figure D-8 Run set 8

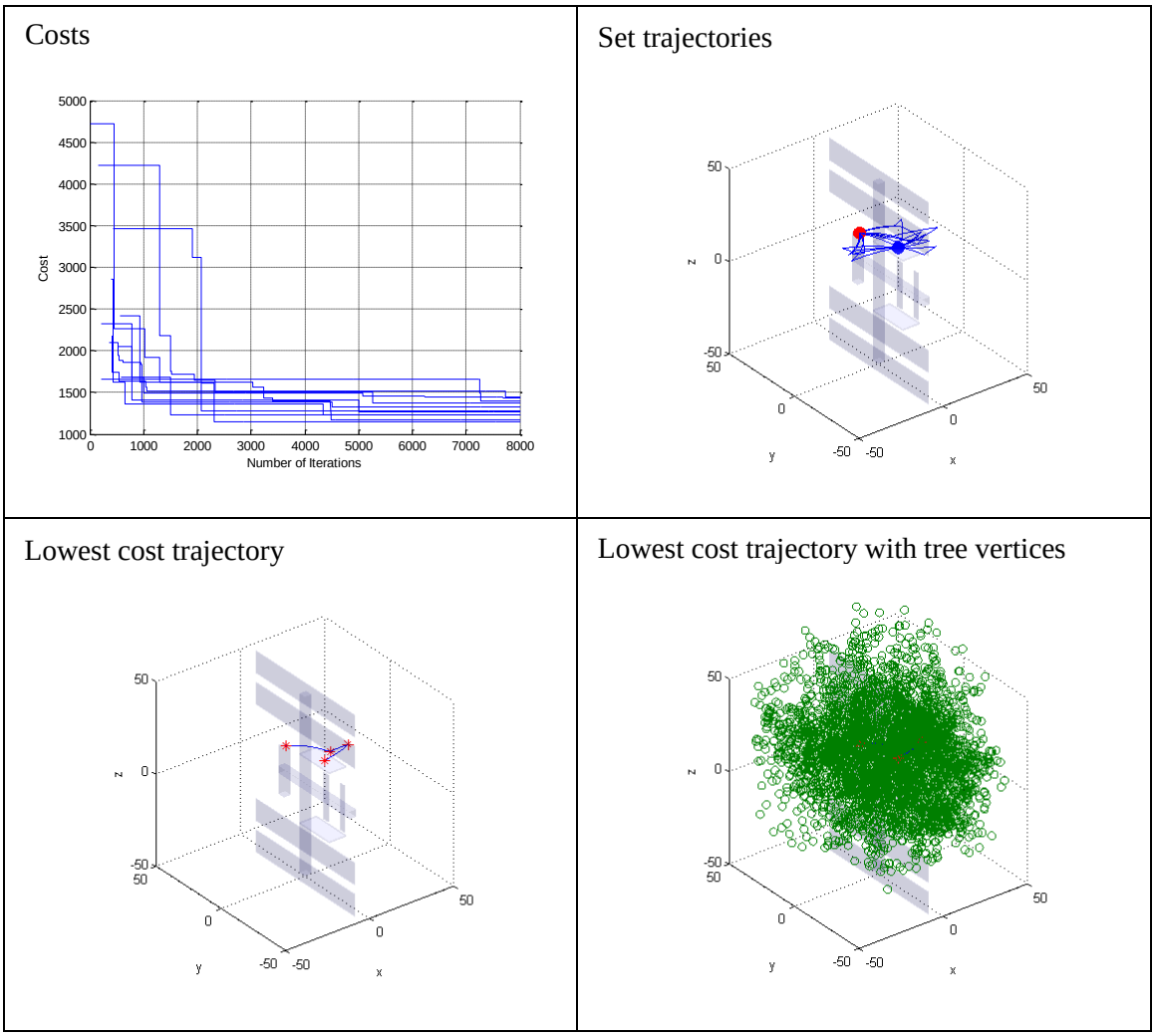


Figure D-9 Run set 9

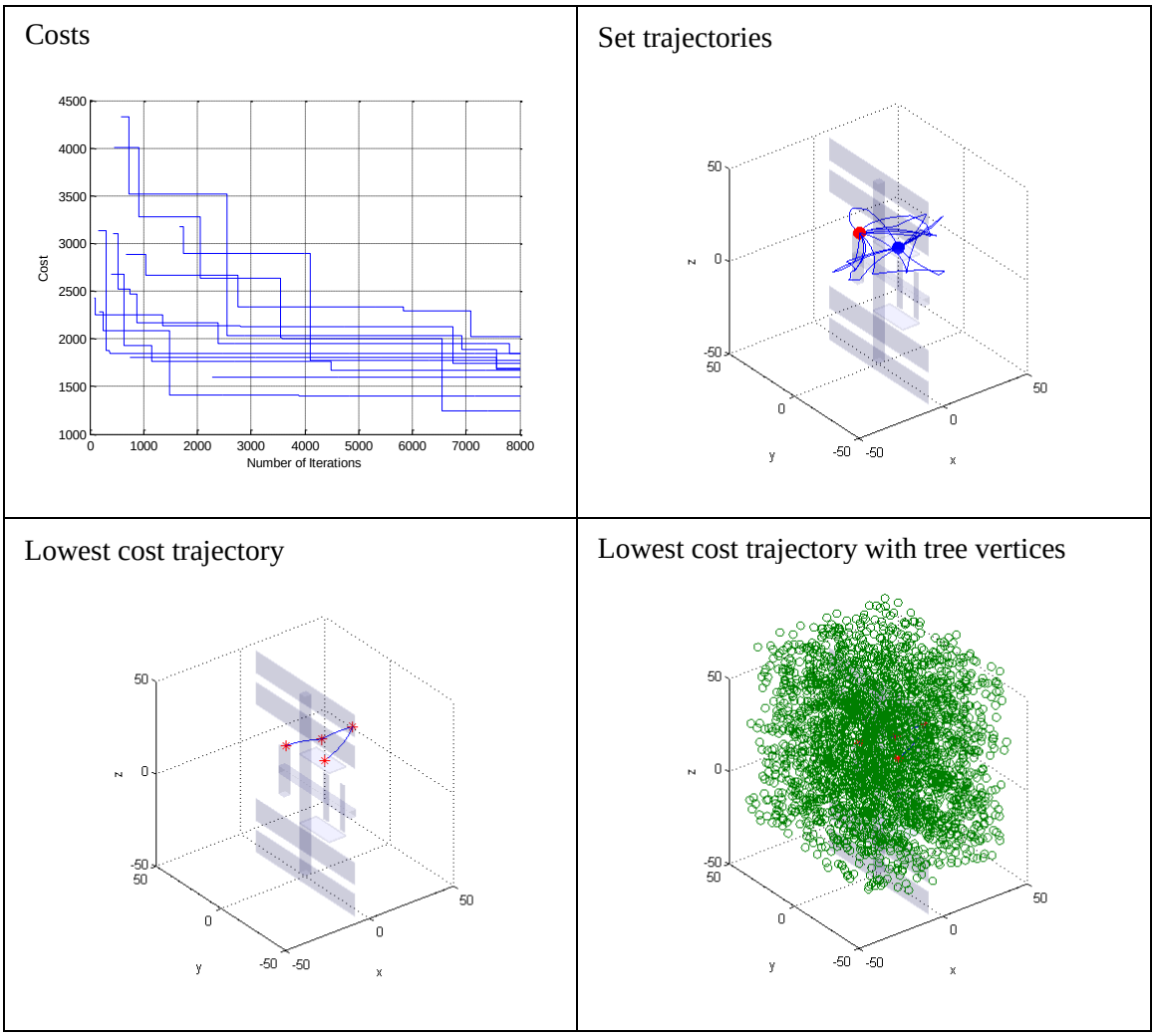


Figure D-10 Run set 10

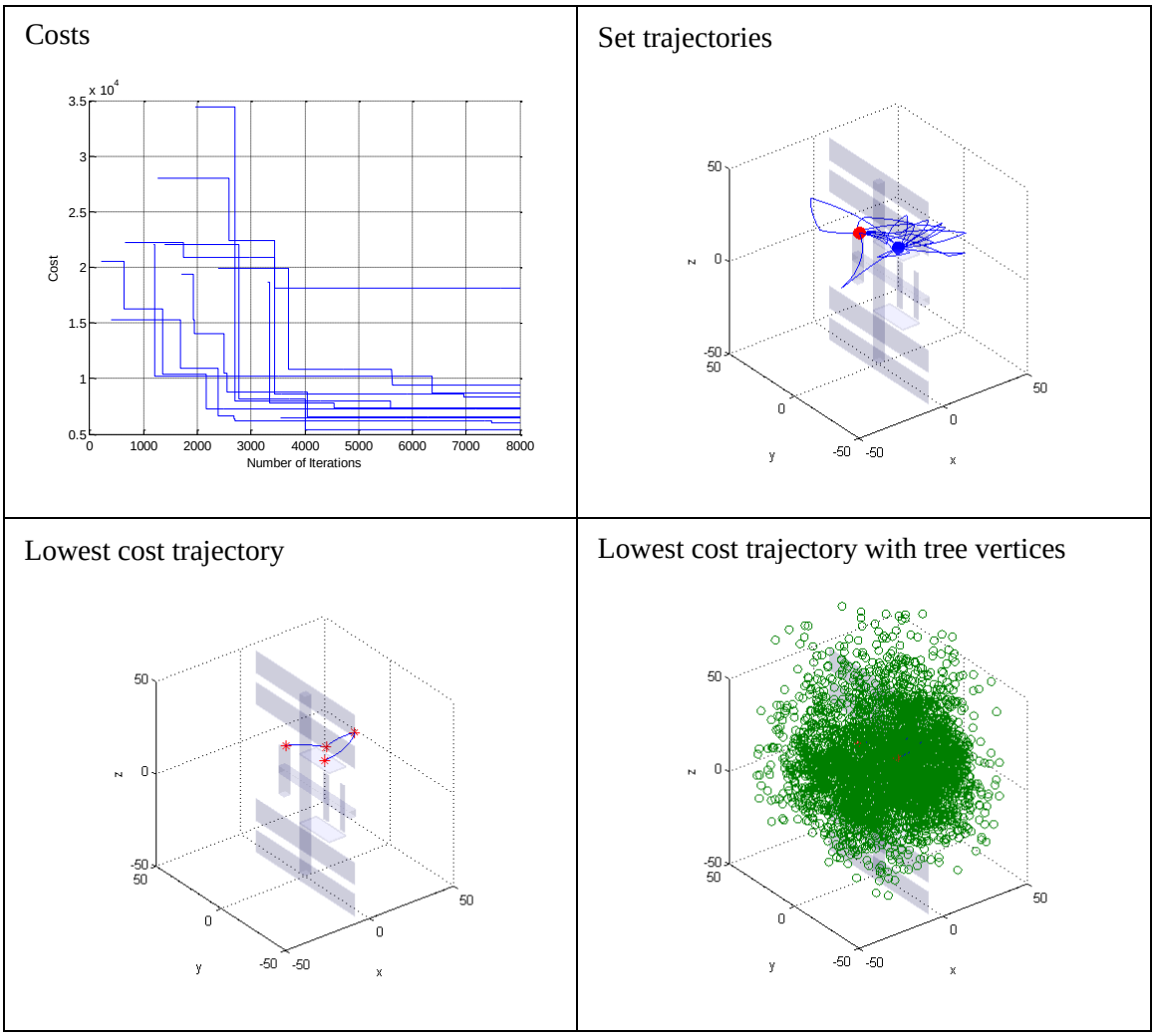


Figure D-11 Run set 11

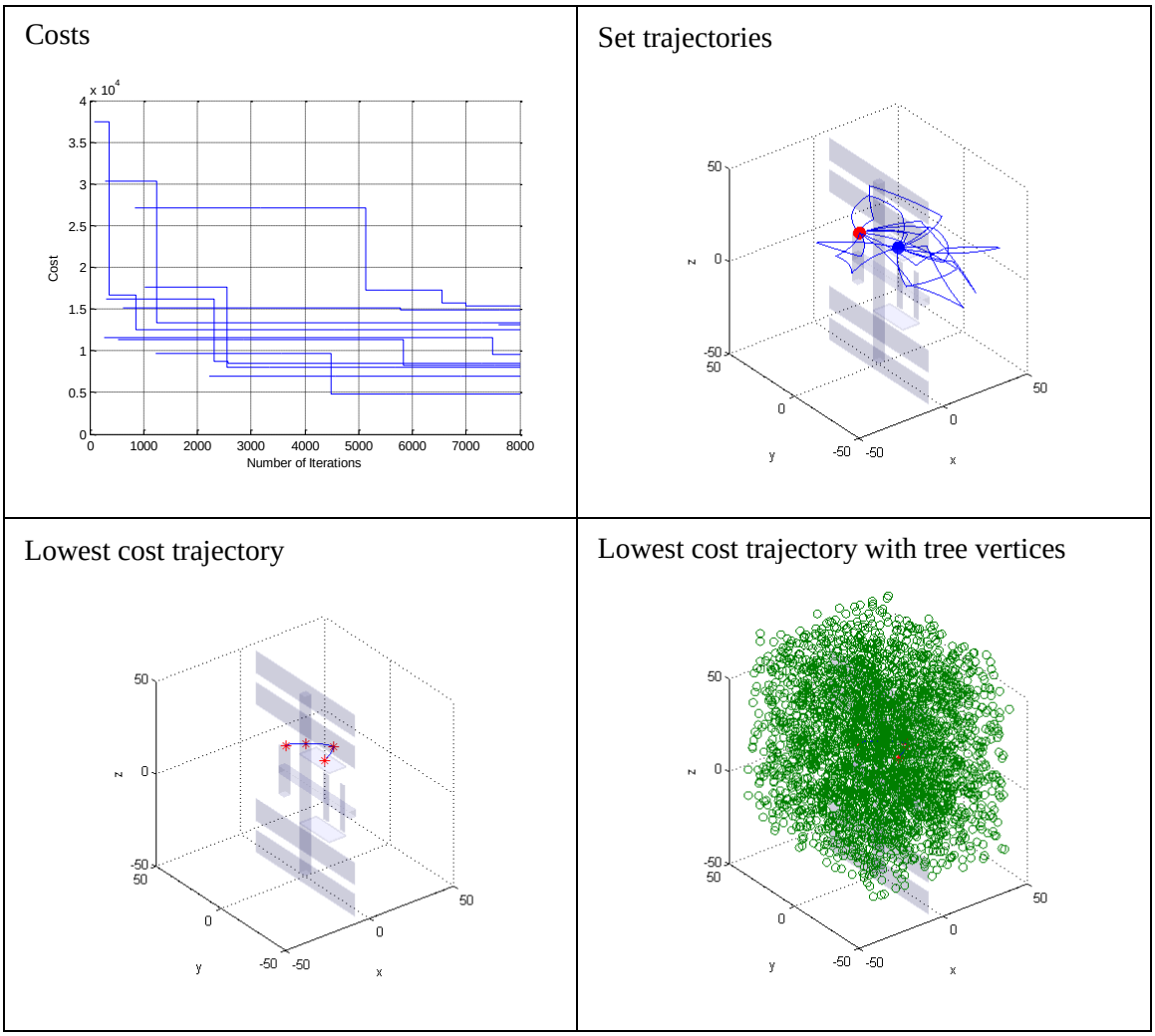


Figure D-12 Run set 12

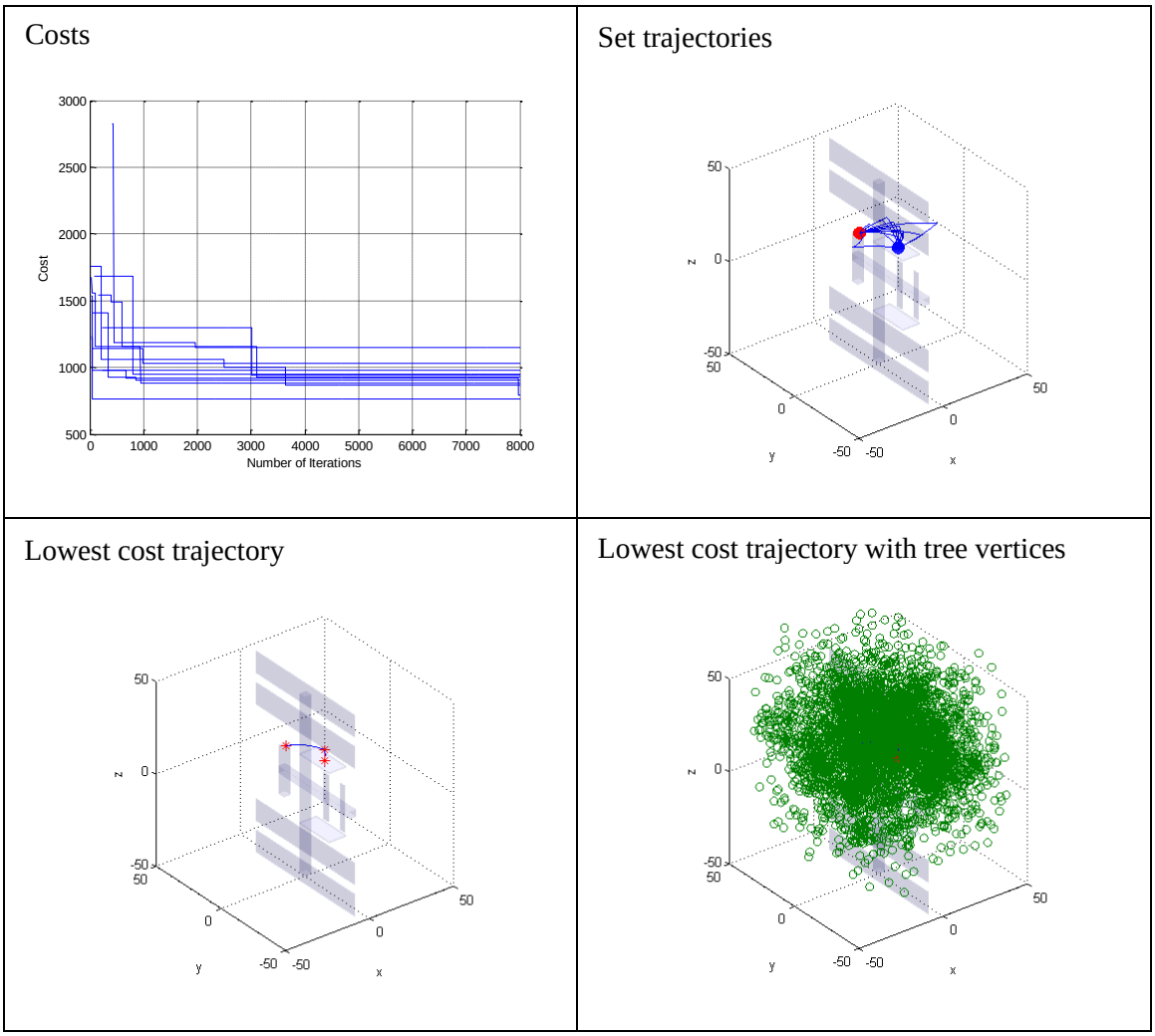


Figure D-13 Run set 13

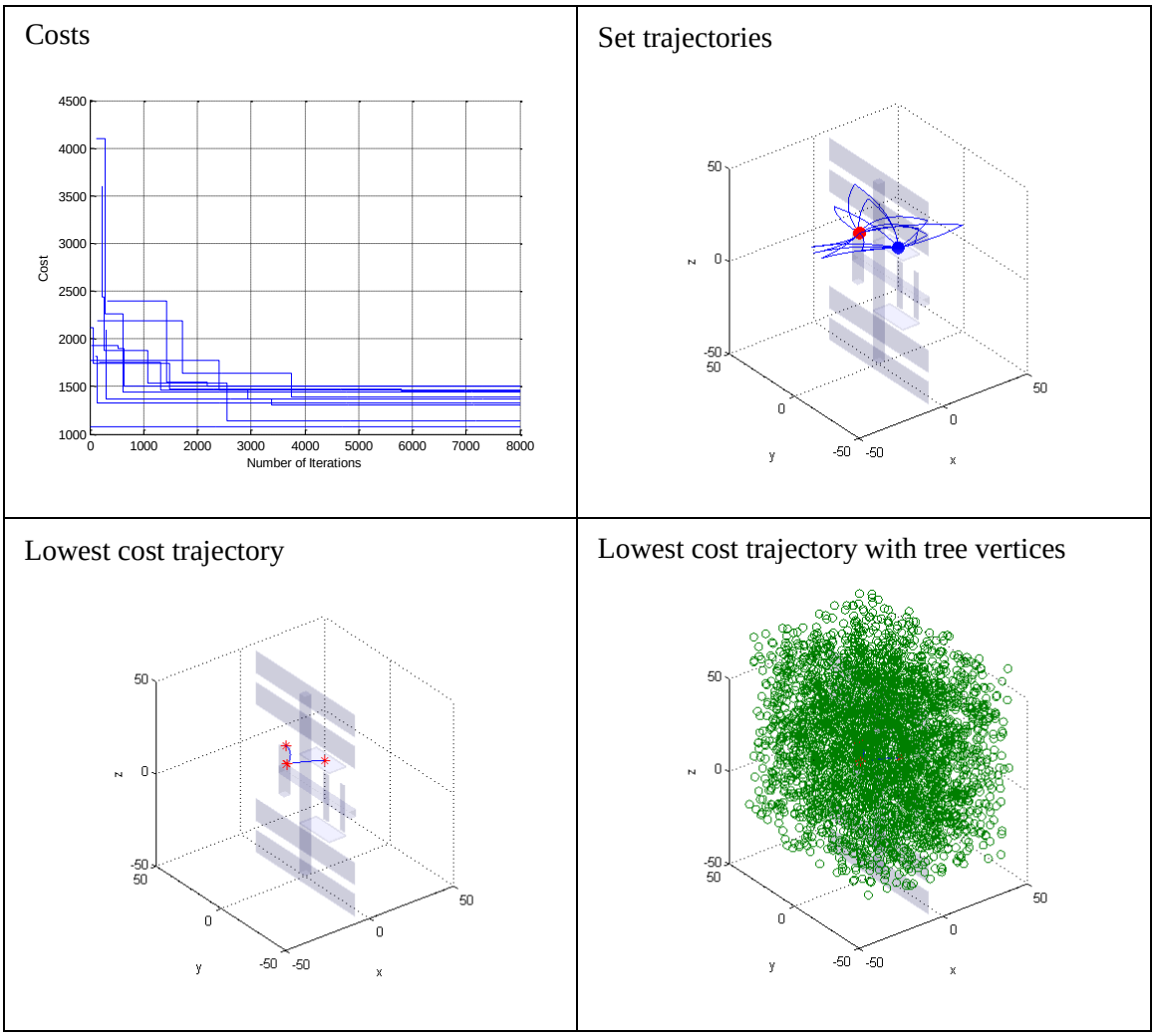


Figure D-14 Run set 14

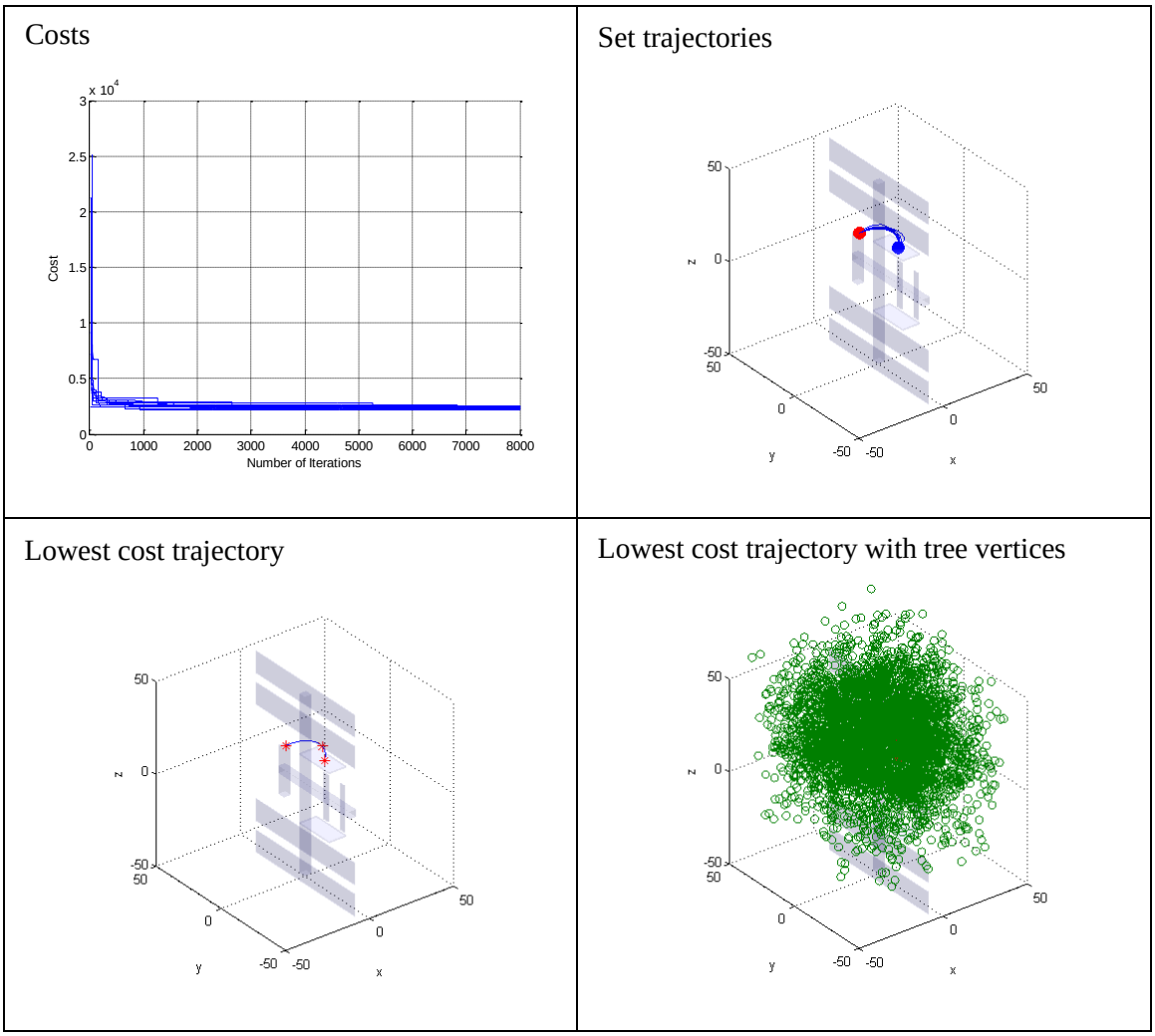


Figure D-15 Run set 15

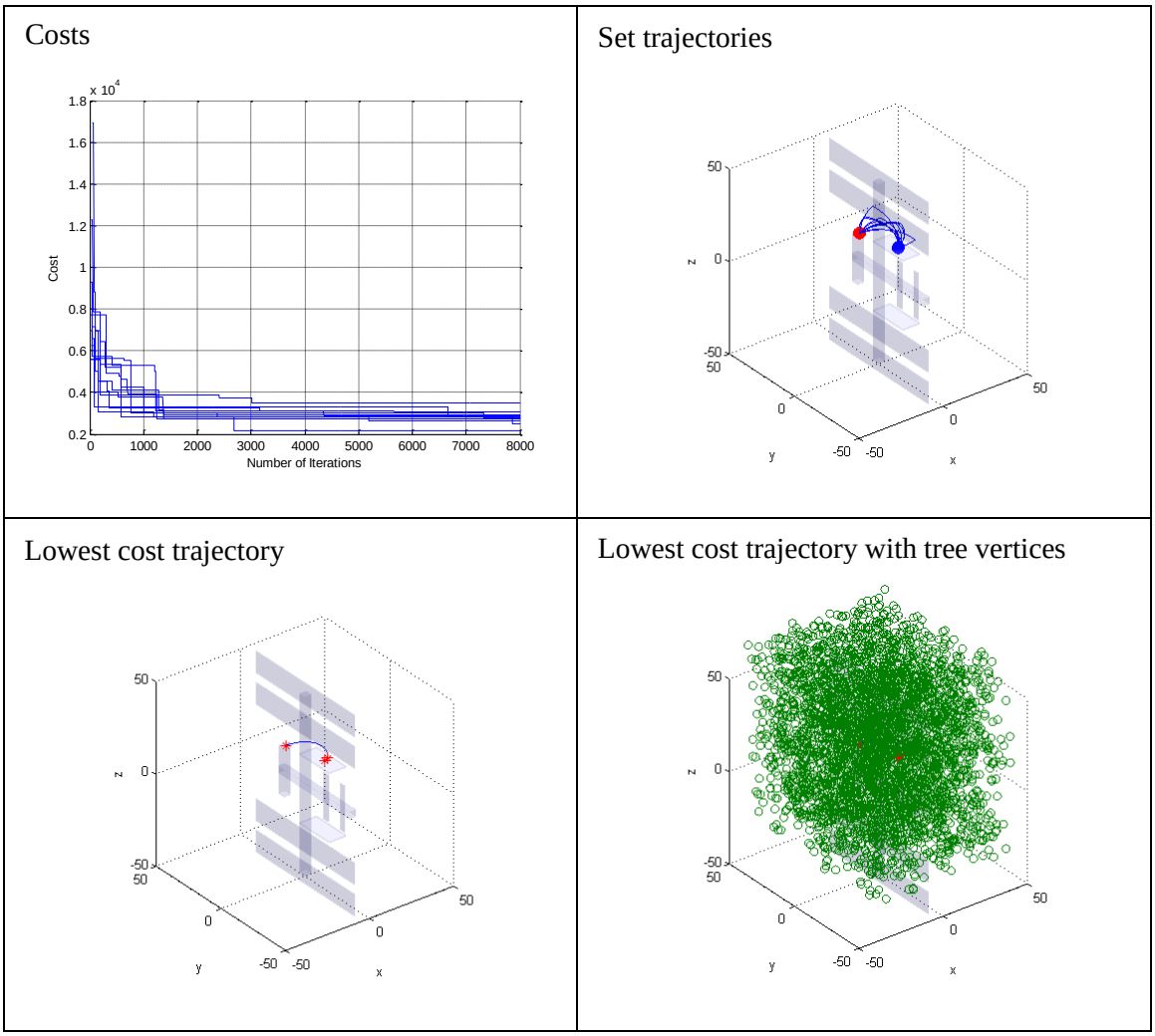


Figure D-16 Run set 16

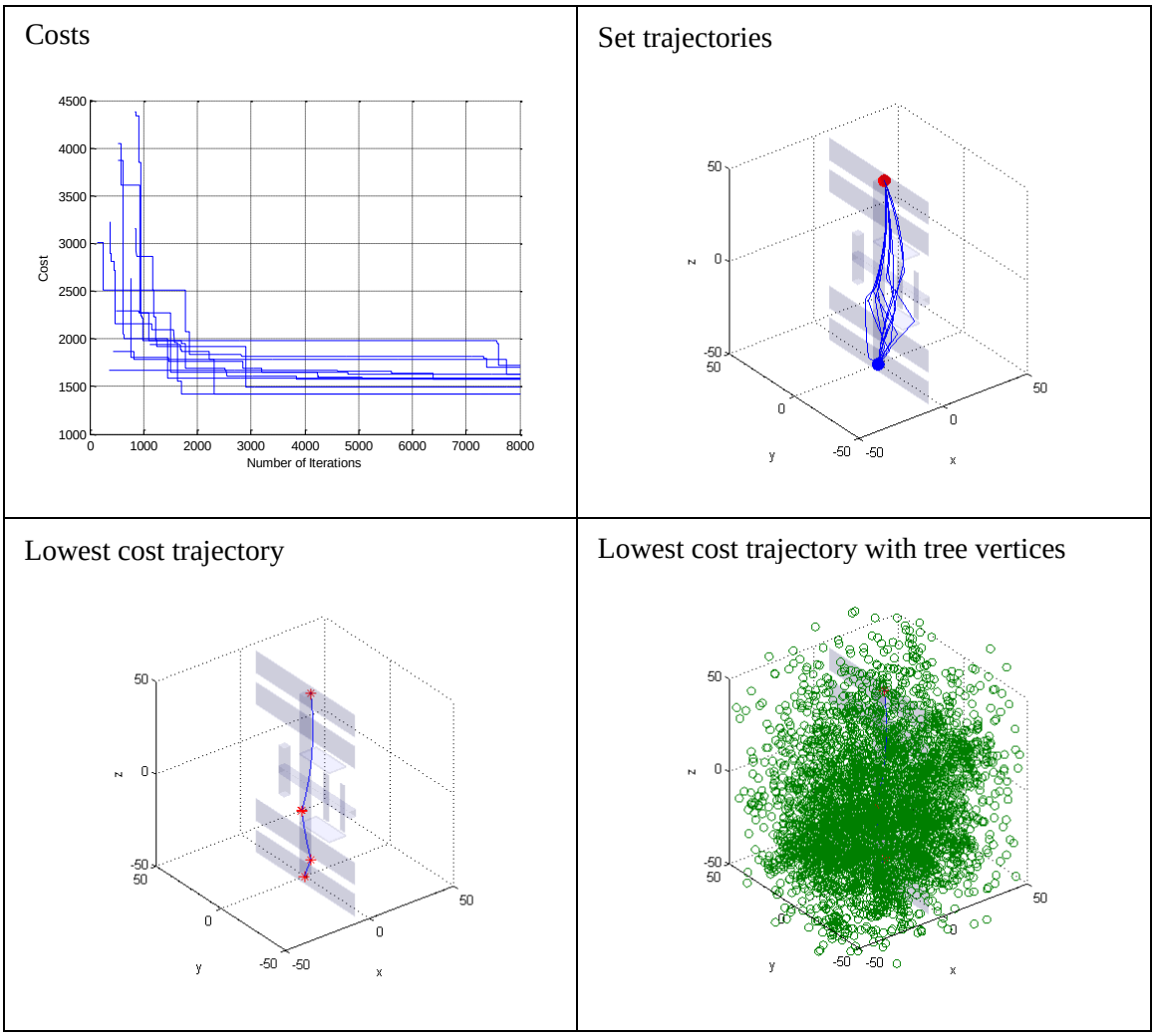


Figure D-17 Run set 17

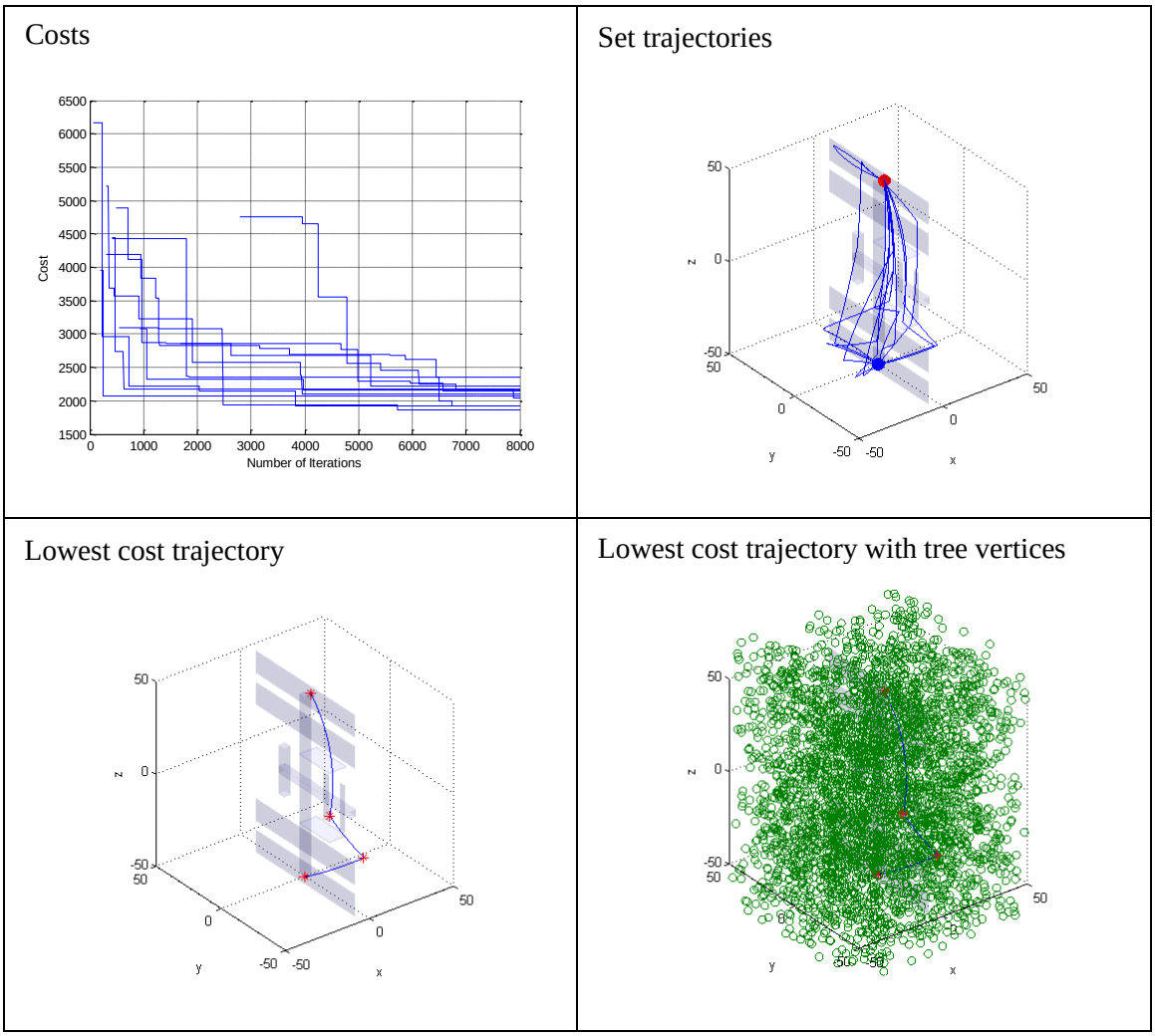


Figure D-18 Run set 18

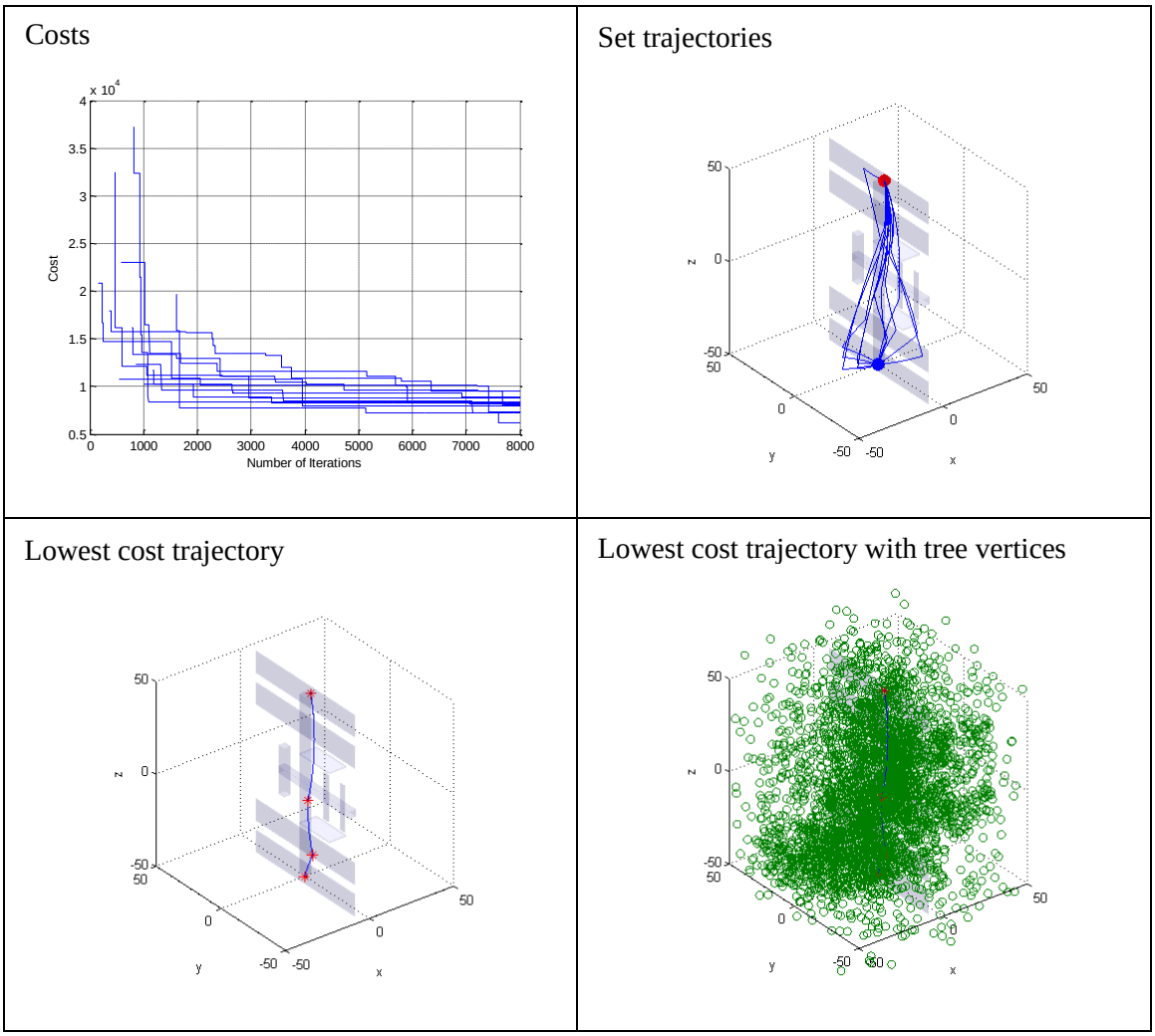


Figure D-19 Run set 19

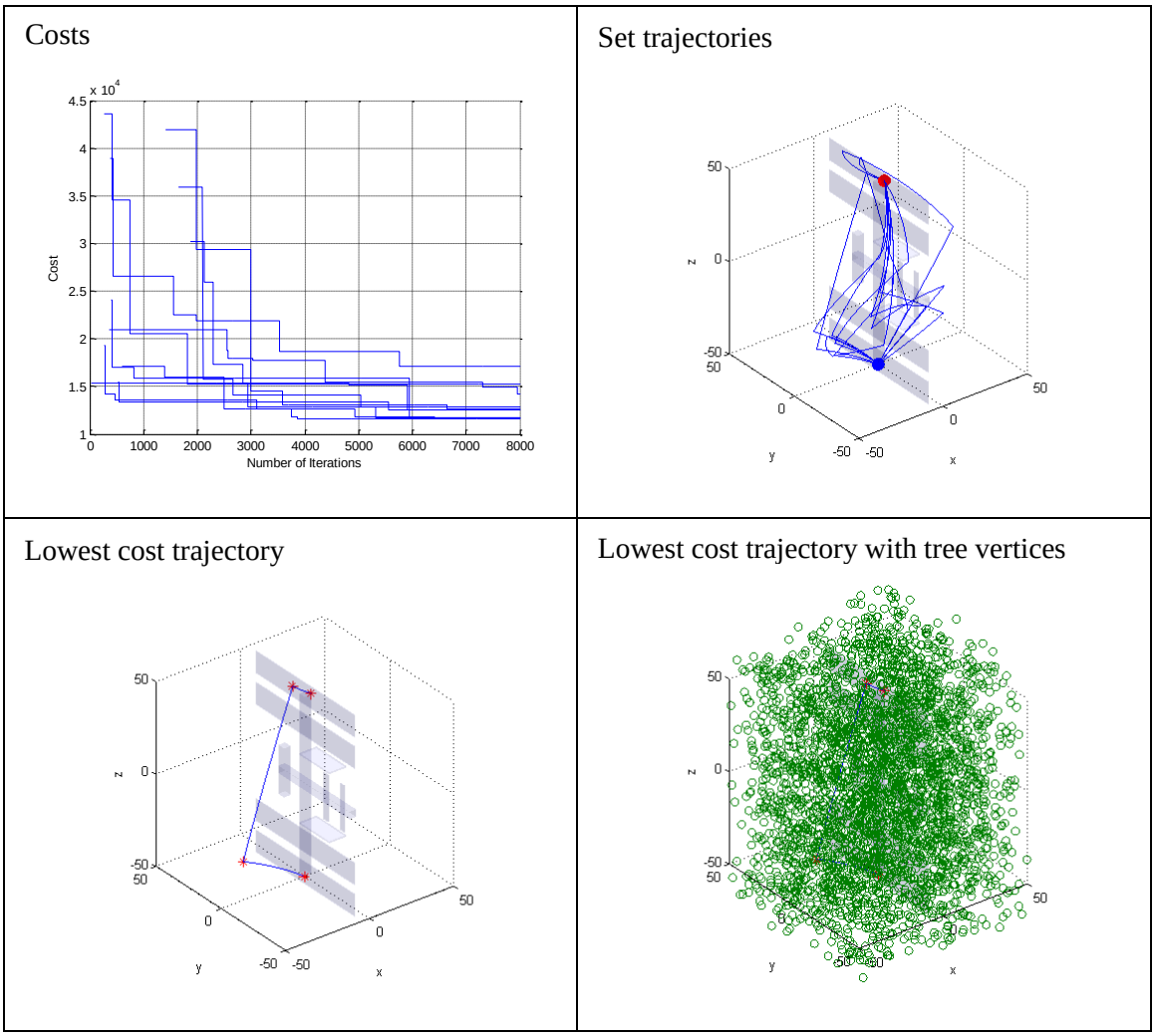


Figure D-20 Run set 20

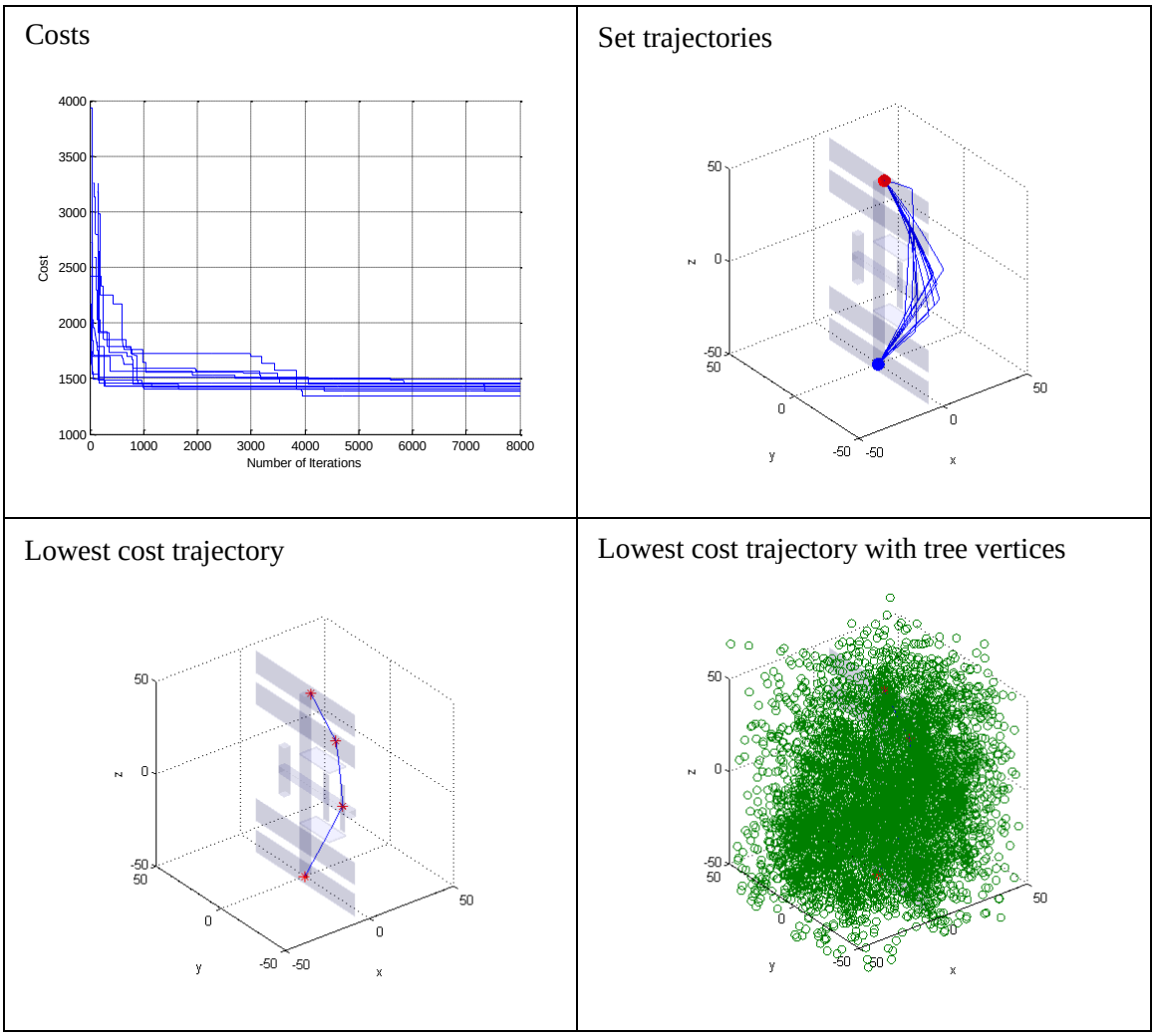


Figure D-21 Run set 21

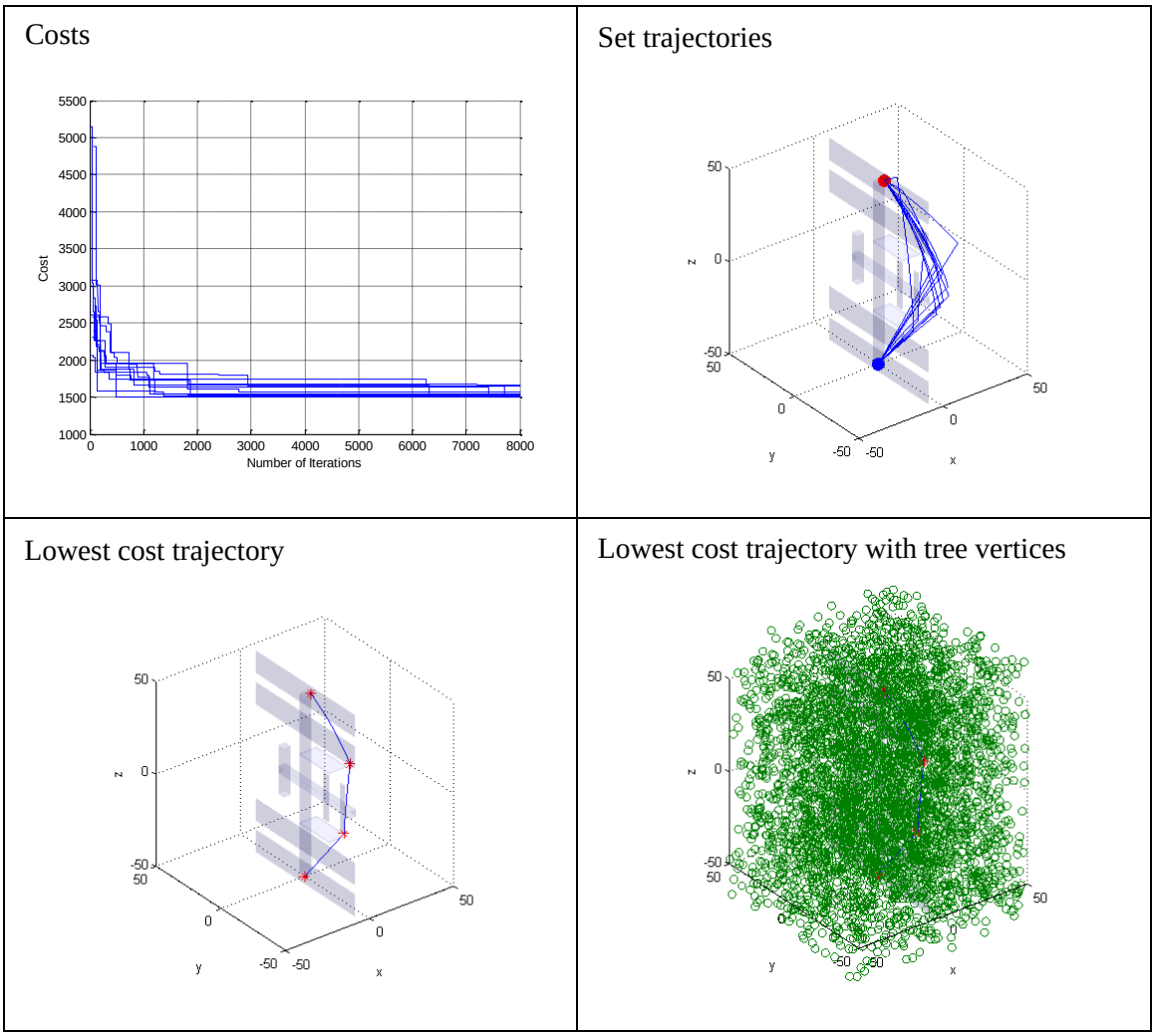
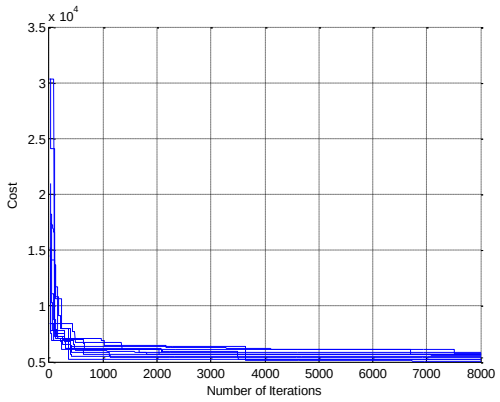
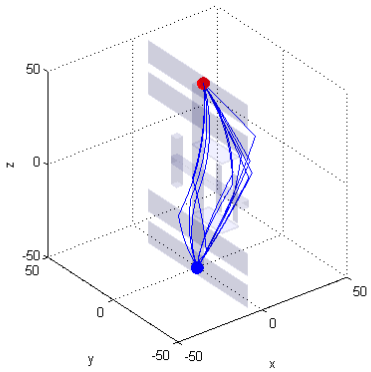


Figure D-22 Run set 22

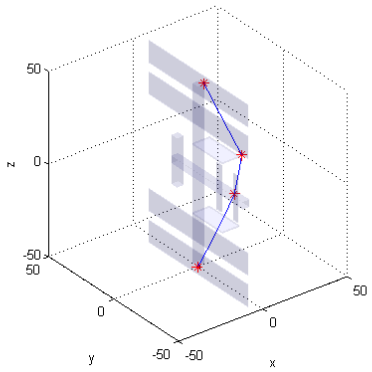
Costs



Set trajectories



Lowest cost trajectory



Lowest cost trajectory with tree vertices

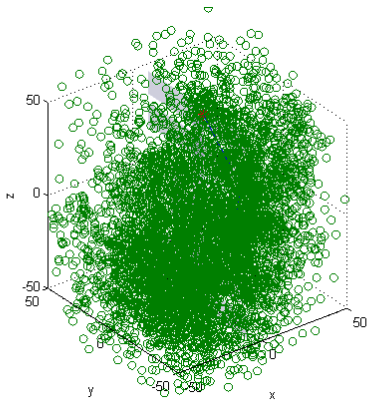
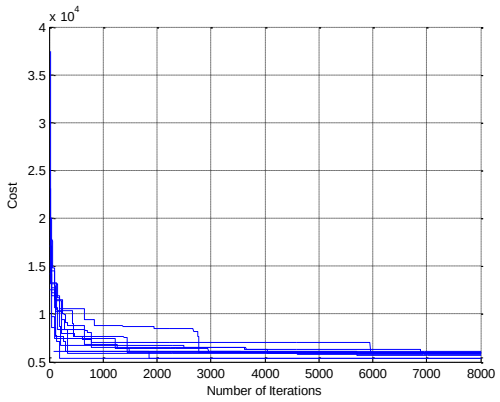
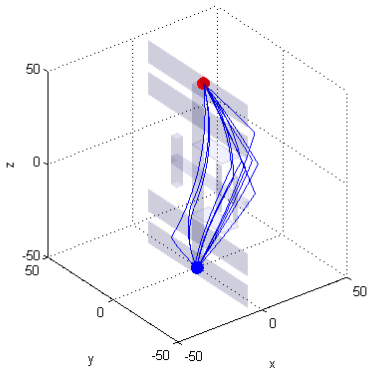


Figure D-23 Run set 23

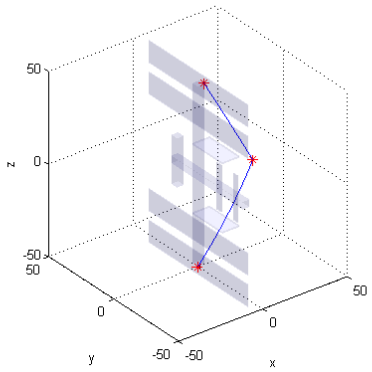
Costs



Set trajectories



Lowest cost trajectory



Lowest cost trajectory with tree vertices

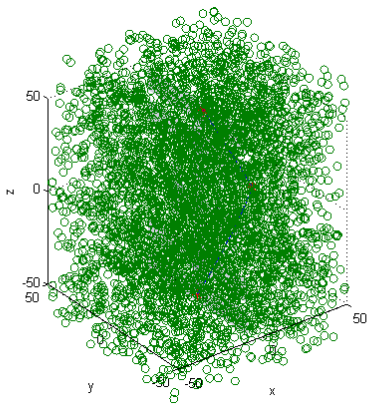


Figure D-24 Run set 24

Bibliography

[1] Auger, A., and Hansen, N., "Tutorial CMA-ES: Evolution strategies and covariance matrix adaptation," *The Genetic and Evolutionary Computation Conference (GECC))*, 2012, pp. 827-848.

[2] Bibby, J., and Necessary, Ryan, "Mini AERCam," NASA, March 2009, [website], URL: http://aercam.jsc.nasa.gov/sprint/ [cited 10 July 2014].

[3] Breger, L. S., and How, J. P., "Safe Trajectories for Autonomous Rendezvous of Spacecraft," *Journal of Guidance, Control, and Dynamics*, vol. 31, 2008, pp. 1478–1489.

[4] Chen, A., "Propulsion System Characterization for the Spheres Formation Flight and Docking Testbed," Master of Science, Massachusetts Institute of Technology, 2002.

[5] Clohessy, W. H., and Wiltshire, R. S., "Terminal Guidance System for Satellite Rendezvous," *Journal of the Aerospace Sciences*, vol. 27, 1960, pp. 653–658.

[6] Curtis, H., *Orbital Mechanics for Engineering Students*, Butterworth-Heinemann, 2013.

[7] Devore, J. L., *Probability and Statistics for Engineering and the Sciences*, 5th ed., Brooks/Cole, Pacific Grove, CA, 2000, Chaps. 10, 11.

[8] DiGirolamo, L.J., Hoskins, A.H., Hacker, K.A., and Spencer, D.B., "A Hybrid Motion Planning Algorithm for Safe and Efficient Close Proximity, Autonomous Spacecraft Missions," *2014 AIAA SPACE Conference*, San Diego, CA, 2014.

[9] Fredrickson, S. E., Abbott, L. W., Duran, S., Jochim, J. D., Studak, J. W., Wagenknecht, J. D., and Williams, N. M., "Mini AERCam: development of a free-flying nanosatellite inspection robot," *SPIE AEROSense 2003 Conference*, Orlando, FL, 2003, pp. 97–111.

[10] Gavin, Multiple Rapidly-exploring Random Tree (RRT), MATLAB Code, MATLAB Central File Exchange, 15 Sep 2013.

[11] Goretkin, G., Perez, A., Platt, R., and Konidaris, G., "Optimal sampling-based planning for linear-quadratic kinodynamic systems," *2013 IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 2429–2436.

[12] Hansen, N., and Ostermeier, A., "Completely Derandomized Self-Adaptation in Evolution Strategies," *Evolutionary Computation*, vol. 9, Jun. 2001, pp. 159–195.

[13] Hansen, N., "The CMA Evolution Strategy: A Comparing Review," *Towards a New Evolutionary Computation*, J.A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, eds., Springer Berlin Heidelberg, 2006, pp. 75–102

[14] Hansen, N., "The CMA Evolution Strategy: A Tutorial," Jun. 2011, [website], URL: https://www.lri.fr/~hansen/cmatutorial.pdf [cited 20 July 2014].

[15] Jastrebski, G. ., and Arnold, D. V., "Improving Evolution Strategies through Active Covariance Matrix Adaptation," *IEEE Congress on Evolutionary Computation, 2006. CEC 2006*, 2006, pp. 2814–2821.

[16] Karaman, S., and Frazzoli, E., "Optimal kinodynamic motion planning using incremental sampling-based methods," *2010 49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 7681–7687.

[17] Karaman, S., and Frazzoli, E., "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, Jun. 2011, pp. 846–894.

[18] Koren, Y., and Borenstein, J., "Potential field methods and their inherent limitations for mobile robot navigation," *1991 IEEE International Conference on Robotics and Automation*, Sacramento, CA: 1991, pp. 1398–1404 vol.2.

[19] LaValle, S. M., and Kuffner, J. J., "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, vol. 20, May 2001, pp. 378–400.

[20] McInnes, C. R., "Autonomous path planning for on-orbit servicing vehicles," *Journal of the British Interplanetary Society*, vol. 53, 2000, pp. 26–38.

[21] Nilsson, N. J., *Principles of Artificial Intelligence*, Springer Science & Business Media, 1982.

[22] Richards, A., Schouwenaars, T., How, J. P., and Feron, E., "Spacecraft Trajectory Planning with Avoidance Constraints Using Mixed-Integer Linear Programming," *Journal of Guidance, Control, and Dynamics*, vol. 25, 2002, pp. 755–764.

[23] Robbins, H. M., "An analytical study of the impulsive approximation," *AIAA Journal*, vol. 4, Aug. 1966, pp. 1417–1423.

[24] Stentz, A., "Optimal and efficient path planning for partially-known environments," *1994 IEEE International Conference on Robotics and Automation, 1994. Proceedings*, 1994, pp. 3310–3317 vol.4.

[25] Webb, D. J., and van den Berg, J., "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," *2013 IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 5054–5061.