ABSTRACT

Title of Dissertation:	COMBINED ROBUST OPTIMAL DESIGN, PATH AND MOTION PLANNING FOR UNMANNED AERIAL VEHICLE SYSTEMS SUBJECT TO UNCERTAINTY
	By Eliot Rudnick-Cohen
Dissertation Directed By:	Professor Shapour Azarm, Mechanical Engineering
	Professor Jeffrey W. Herrmann, Mechanical Engineering

Unmanned system performance depends heavily on both how the system is planned to be operated and the design of the unmanned system, both of which can be heavily impacted by uncertainty. This dissertation presents methods for simultaneously optimizing both of these aspects of an unmanned system when subject to uncertainty. This simultaneous optimization under uncertainty of unmanned system design and planning is demonstrated in the context of optimizing the design and flight path of an unmanned aerial vehicle (UAV) subject to an unknown set of wind conditions. This dissertation explores optimizing the path of the UAV down to the level of determining flight trajectories accounting for the UAVs dynamics (motion planning) while simultaneously optimizing design. Uncertainty is considered from the robust (no probability distribution known) standpoint, with the capability to account for a general set of uncertain parameters that affects the UAVs performance.

New methods are investigated for solving motion planning problems for UAVs, which are applied to the problem of mitigating the risk posed by UAVs flying over inhabited areas. A new approach to solving robust optimization problems is developed, which uses a combination of random sampling and worst case analysis. The new robust optimization approach is shown to efficiently solve robust optimization problems, even when existing robust optimization methods would fail. A new approach for robust optimal motion planning that considers a "black-box" uncertainty model is developed based off the new robust optimization approach. The new robust motion planning approach is shown to perform better under uncertainty than methods which do not use a "black-box" uncertainty model. A new method is developed for solving design and path planning optimization problems for unmanned systems with discrete (graph-based) path representations, which is then extended to work on motion planning problems. This design and motion planning approach is used within the new robust optimization approach to solve a robust design and motion planning optimization problem for a UAV. Results are presented comparing these methods against a design study using a DOE, which show that the proposed methods can be less computationally expensive than existing methods for design and motion planning problems.

COMBINED ROBUST OPTIMAL DESIGN, PATH AND MOTION PLANNING FOR UNMANNED AERIAL VEHICLE SYSTEMS SUBJECT TO UNCERTAINTY

by

Eliot Sylvan Rudnick-Cohen

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2019

ACKNOWLEDGEMENTS

I'd like to thank both of my advisors, Dr. Shapour Azarm and Dr. Jeffrey Herrmann, for putting up with me until this point. Their advice, suggestions, feedback and criticism were immensely helpful in growing and refining my skills as a researcher.

I'd also like to thank the members of my committee, Dr. Steven A. Gabriel, Dr. Huan Xu, Dr. Michael Otte and Dr. Ali Haghani, for setting aside the time to be on my committee for this dissertation.

Additionally, I'd like to thank NAVAIR (cooperative agreement N00421132M006) and AFRL (contract number FA87501820114) which have supported the various research projects that I've been a part of throughout my graduate career and while working on the research composing this dissertation.

I'd also like to thank my fellow peers from the design decision support laboratory (Ruchir, Tianchen, Randall, Xiangxue and Abu), for being around to discuss and share the various things we were all working on. Similarly I'd like to thank everyone on the AFRL project (Ruchir, Estefany, Sharan, Suyash and Khaled) for putting up with me during our group meetings.

I'd like to thank my parents for supporting me throughout this endeavor and for the innumerable times which they lent me a hand with various minor things over the last few years.

Finally, I'd like thank you (the reader), since you're reading this dissertation.

Table of Contents

Chapter 1: Introduction

The design and operation of unmanned systems involves a large number of interacting factors that impact performance. This dissertation presents approaches for finding a "system optimal" solution in terms of simultaneously optimizing the design of an unmanned system and how that unmanned system will be operated, while also considering the effects of uncertainty. In particular, this dissertation focuses on the problem of simultaneously optimizing the design and motions taken by an unmanned aerial vehicle (UAV), in the context of mitigating third party risk posed by the UAV's flight plan to populated areas being flown over. In order to solve this problem, several methods are first presented for solving motion planning problems for UAVs with boundary value problems (BVPs) that have non-trivial computational costs. A new method for solving robust design optimization problems via scenario generation and local robust optimization is then presented. The framework of this robust optimization approach is used to develop an approach for robust optimal motion planning for UAV systems. Several approaches for solving the problem of simultaneous UAV design and path planning optimization on a predefined graph are also developed. One of these approaches developed for dealing with a predefined graph is then extended to work on UAV design and motion planning optimization problems. The resulting UAV design and motion planning optimization approach is then used within the robust optimization approach developed in order to simultaneously optimize the design and motions taken by a UAV subject to uncertainty.

1.1 Research Questions

This dissertation will endeavor to answer the following research questions:

RQ1. How can optimal motion planning be done for UAV systems when the objective being optimized is not time? (Chapter 2)

RQ2. How can robust optimization problems be efficiently solved when non-convex constraints are present or when the optimization problem considered is typically not solved using mathematical optimization techniques? (Chapter 3, Chapter 4, Chapter 7)

RQ3. How can current methods for optimal sampling based motion planning for unmanned systems be extended to account for robustness with respect to uncertainty? (Chapter 4)

RQ4. How can the design and motion of an unmanned system be optimized simultaneously? (Chapter 5, Chapter 6)

RQ5. How can the performance of an unmanned system be optimized with respect to both its design and operation (path planning), while also being able to account for uncertainty (robust optimization) and the dynamics of the unmanned system (motion planning)? (Chapter 6, Chapter 7)

1.2 Literature Review

To understand the relation between the problem's considered in this dissertation related to optimizing the design and path of an unmanned system under uncertainty and existing research, a review is given here of topics related to the design and path planning of unmanned systems, robust optimization under uncertainty, robust path planning methods and techniques for unmanned system motion planning involving risk related objectives. A listing of the related research questions to each of these topics is also given.

1.2.1 Design and Path Planning *(RQ4, RQ5)*

The problem of optimizing the design of a UAV with respect to a path is similar to that of designing any aircraft for a specific mission. Typically, aircraft design optimization problems contain a large number of subsystems [59] and may require complex aerodynamics simulations to model interactions between design variables [4]. This makes it infeasible to exhaustively search the design space of aircraft design optimization problems without some combination of gradient-based search techniques and decomposing the problem into multiple subproblems [59]. Numerous works have investigated instances of UAV design optimization by considering vehicle control [97], uncertainty [53], component selection [77] and using multidisciplinary design optimization (MDO) to optimize the overall UAV system [90]. Many issues important to UAV path planning, such as wind or flight altitude [45], can be mitigated through changes to the UAV's physical design, flight parameter design or through use of reconfigurable systems technologies [114] such as morphing wings, thus there exist clear benefits to integrating the optimization of the UAV's design with the optimization of the path it will take.

The integration of design optimization with path optimization is similar to the problem of co-design, the problem of integrating vehicle's design optimization with controller design optimization, particularly in co-design problems where optimal control is used for trajectory optimization, such as in [20]. Note that design and path optimization occur at the same time scale in these types of approaces, since only one specific motion is being optimized. Rastegar et al. [99] considered a decomposition based approach for a robotic manipulator system, decomposing the problem into independent design and control subproblems managed by a third coordinating subproblem. Nigam and Kroo [87] developed a decomposition-based optimization approach that solves mission planning and design optimization subproblems for a UAV, which considered interactions between the two lower-level problems and used a response surface method to approximate the mission planning sub-problem. Ha et al. [49] tackles the problem of optimizing both the design and gait of a legged robot by using optimal control to determine what the robot's motion would be for a given design.

Several works have considered the integration of vehicle design into traditional operations research transportation problem formulations. Lee and Ahn [70] optimized the layout design of a planetary rover while simultaneously optimizing the rover's exploration routes using a vehicle routing problem. They solved the problem by using a combination of exhaustive search over a single parameter parametrizing all design variations with integer programming techniques. Mufalli et al. [86] considers the combined problem of sensor selection and multi-UAV routing, by formulating the problem as a mixed integer programming problem in terms of a discrete set of available sensors and locations to visit. Taylor and Weck [120] considered the problem of optimizing the design of both aircraft and the air transport system they operate in. They solve the resulting formulation by using simulated annealing for the aircraft design with the transport system being treated as a sub-problem solved via linear programming (LP) at each iteration.

Several works have considered the integration of design into motion planning for unmanned systems. Denarie et al. [33] and Molloy et al. [84] consider a system design and a path planning problem, which explores the feasibility of a fixed set of designs using sampling based motion planning techniques in order to find the best design in the set. Rudnick-Cohen et al. [108] considered the opposite problem, considering methods for using continuous design optimization formulations in conjunction with a discrete path planning graph. Baykal and Alterovitz [9] optimized the design and path of a robotic manipulator for reachability of predefined goal regions. They used simulated annealing to optimize the design of the manipulator by determining the cost of each design iteration by solving a motion planning problem with rapidly-exploring random trees (RRT) [68], via the RRT* algorithm [63]. Glorieux et al. [44] applied an MDO architecture to optimize the design and motions of a group of robots used to handle sheet metal parts. A similar strategy to [9] was used, with an inner trajectory optimizer being used to determine the trajectories and an outer loop global optimization approach being used to optimize designs for subject to those trajectories. All of these approaches can be classified as bilevel optimization [117] methods where either a motion planning problem is nested in a design optimization problem ([9], [44]) or a design optimization problem is nested within a path or motion planning problem. ([33], [84], [108]).

1.2.2 Risk and Safety Based Motion Planning for UAVs *(RQ1, RQ3)*

Operating large unmanned aerial vehicles (UAVs) over inhabited areas poses a risk to third parties on the ground. For example, the mass of a MQ-4C Triton is approximately 14 metric tons, and its wingspan is nearly 40 meters [1]. If it lost power and crashed in an uncontrolled dive, it could easily injure or kill persons caught in its path. This risk is critical after the vehicle takes off; as it climbs towards its cruising altitude, the vehicle's operator or onboard crash mitigation systems may not have enough time to respond to a failure. Thus, it is important to plan takeoff trajectories that minimize the risk to third parties.

Most works on safety based motion planning focus on avoiding collisions with obstacles due to momentum and accounting for moving obstacles. In order to model the feasible region of the configuration space in which a vehicle will not collide with an obstacle, [42] and [92] introduced the concept of an inevitable collision state. The problem of avoiding collisions with dynamically moving obstacles via replanning was discussed in [100], [54] and [10]. When managing third party risk for UAV flight planning, the problem being solved differs from these approaches in that safety is the objective of the problem instead of a constraint. Thus, optimal motion planning techniques are needed, rather than approaches that focus on preventing collisions.

Several works have also explored different optimal motion planning techniques for UAV flight planning. Choudhury [31] developed an approach for planning emergency landings for an unmanned helicopter by extending RRT* to also compute alternate routes respecting a variety of additional planning objectives. The FMT* algorithm [58] has also been applied to the problem of fixed wing UAV motion planning in [110]. RRT* was extended in [30] to also take advantage of CHOMP [131], a gradient based trajectory optimization technique, in order to increase its rate of convergence, with the resulting algorithm being demonstrated in a flight planning problem for an unmanned helicopter.

Although the author is unaware of any previous works on the problem of UAV third party risk management during take-off, prior studies have explored the problem of risk management during UAV emergency landings. Risk-based A* searches have been used for two-dimensional representations of the underlying path planning problem in [121] for a multicopter and in [34] for UAV systems in general. Primatesta et al. [94] considered a two-dimensional (2D) motion planning problem for managing UAS risk posed to third parties. A three-dimensional Dubins curve based approach was developed in [35] to be used in conjunction with a path following controller to land a fixed wing UAV under loss of thrust conditions.

Although RRT* has been used for UAV flight planning problems, Otte [88] noted that RRT* is extremely slow to propagate cost updates when a lower cost region is sampled for the first time, unlike methods that update the shortest paths to every node in such situations, such as RRT$^{\#}$ [7].

The concept of using Dubins curves with sampling based motion planning techniques is not new, dating back to early works on RRT such as [69]. Dubins curves are a set of curves which define the shortest possible paths between two points for a vehicle in 2D with a minimum turning radius, which makes them convenient for modeling aircraft trajectories. There are 6 possible Dubins curves, which are composed of 3 segments (a left or right turn, an opposite direction turn or a straight segment and finally another left or right turn). Most works that consider motion planning for a fixed wing UAV use a three-dimensional extension to the Dubins car model as discussed in [29] and [89], which allows for time optimal motion primitives.

1.2.3 Robust Optimization *(RQ2, RQ3, RQ5)*

The goal of robust optimization is to find a solution to a problem that is feasible under all possible values that any uncertain parameters present in that problem can take and which has the best cost of any solution under the worst possible uncertain parameters for that solution. Problem 1, as shown in in Eq. (1), provides a general formulation for a robust optimization problem based on the formulation given in [14], where $f\ (x,\ u)$ (objective function), $d_l(x)$ (deterministic constraints), $g_i(x,u)$ (constraints containing uncertainty) and $q_j(u)$ (constraints defining domain of uncertain parameters $\mathbf{U}$) are all assumed to be continuously differentiable with respect to both x (design variables) and u (uncertain parameters), which are assumed to be continuous:

Problem 1: Robust Optimization (1.1)

$$\begin{aligned}
&\min_{x,z} z \\
&\textit{subject to} \text{ (or } \textit{s.t.}\text{)} \\
&z \geq f(x,u), \forall u \in \mathbf{U} \\
&d_l(x) \leq 0, \forall l \in \{1,\ldots,L\} \\
&g_i(x,u) \leq 0, \forall i \in \{1,\ldots,I\}, \forall u \in \mathbf{U} \\
&\mathbf{U} = \{u \mid q_j(u) \leq 0, \forall j = 1,\ldots,J\}
\end{aligned}$$

Typical methods for solving robust optimization problems, Eq. (1), represent uncertainty using sets of scenarios (sets of possible values for the uncertain parameters) [14, 18, 11], randomly sampled scenarios [24] or worst-case analysis [128]. However, many of these methods can become impractical for engineering design problems because they might have to deal with highly non-convex constraints, scalability limitations due to the number of uncertain parameters or require too much computational effort to obtain a robust optimal solution.

Most existing methods [14] for solving non-convex robust optimization problems require finding a set of scenarios $\overline{U} = \{u_1,\ldots,u_K\}, \overline{U} \subseteq \mathbf{U}$ that can be used in place of $\mathbf{U}$ in

Problem 2. The resulting formulation can be referred to as being a scenario-based robust optimization problem (SRO), as given in Problem 2 in Eq. (2):

Problem 2: Scenario-based Robust Optimization (SRO) (1.2)

$$
\begin{aligned}
&\min_{x,z} z \\
&s.t. \\
&z \geq f(x, u_k), \forall u_k \in \overline{U} \\
&d_l(x) \leq 0, \forall l \in \{1, ..., L\} \\
&g_i(x, u_k) \leq 0, \forall i \in \{1, ..., I\}, \forall u_k \in \overline{U}
\end{aligned}
$$

An optimal solution to Problem 2 is a robust optimal solution (an optimal solution to Problem 1) if for each constraint $g_i(x,u)$ containing uncertainty, $g_i(x,u_k) \leq 0$ for all scenarios $u_k \in \overline{U}$ implies that $g_i(x,u) \leq 0$ for any possible scenario $u \in \mathbf{U}$.

The most basic approach for constructing the set $\overline{U}$ is to assume that $\overline{U}$ consists of a single "worst-case" scenario u_w and that $g_i(x,u_w) \leq 0$ implies that $g_i(x,u) \leq 0$ for any possible scenario $u \in \mathbf{U}$, commonly referred to as a "worst-case analysis" [37]. Bertsimas et al. [16] use a gradient ascent approach for a worst-case analysis while simultaneously solving an optimization problem. Bertsimas and Nohadani [15] develop a simulated annealing approach which extends the approach of Bertsimas et al. [16] to perform a global search for a robust optimal solution. Li et al. ([71], [72]) develop a measure of robustness around a nominal scenario and use a genetic algorithm for determining the worst-case scenario. Zhou et al. ([128], [129]) develop a sequential quadratic programming robust optimization (SQP-RO) algorithm, where the worst-case scenario for each constraint and the objective function is found via maximization at each SQP iteration. Cheng and Li [28] extend the approach of Zhou et al. [128] to solve problems for global optimality, by using a differential evolution method as an outer optimizer. Similar forms of worst-case analysis are used in the context of reliability based design optimization (RBDO, also called probabilistic or stochastic optimization) by Du and Chen [38], where the most probable point (MPP) is found via an inner optimization problem and used to ensure that constraints are satisfied at a predetermined reliability level. Liang et al. [74] develop a single loop algorithm for RBDO which avoids using the MPP by converting the RBDO problem into a deterministic problem via the first-order Karush-Kuhn-Tucker optimality conditions of the inner optimization problem. In practice, methods which rely on a worst-case or reliability analysis require assumptions that may not hold for non-convex robust optimization, where there can exist multiple "local" worst-case scenarios for a single constraint (see Example 1 in Section 3.3.1 of this dissertation) and where no probability distribution exists for the uncertain parameters.

An alternative to approaches that search for worst-cases is to instead construct the set $\overline{U}$ by using randomly sampled scenarios, an idea first applied by Calafiore and Campi [24] to the problem of robust control design. Chamanbaz et al. [27] and Calafiore ([23], [25]) developed sequential optimization approaches which alternate between checking the feasibility of a candidate solution by sampling further scenarios and finding a new candidate solution when scenarios are found under which the candidate solution is infeasible. Rudnick-Cohen et al. [104] proposed an approach that generates additional scenarios from randomly sampled scenarios through a best and worst-case analysis. Rudnick-Cohen et al. [104] also proposed a method for performing scenario reduction, which can limit the number of scenarios used in a scenario robust optimization problem.

Margellos et al. [79] discuss the tractability and expected number of samples needed for this class of methods. Ramponi [98] introduces the property of “essential robustness” to refer to the conditions under which such methods will asymptotically converge to a robust optimal solution. Because sampling-based approaches converge asymptotically, it is difficult for them to maintain a pre-specified constraint tolerance for feasibility under uncertainty. Note that randomly sampling scenarios help in an inherently global search (every scenario is equally likely to be sampled). Thus the robust optimal solution found by these methods can be considered to be feasible under uncertainty in a global sense, without needing to make any assumption about worst-case scenarios. Sampling based approaches (excluding [104]) make no attempt to minimize the size of $\overline{U}$, this can lead to a much larger optimization problem than would be considered in worst-case analysis based approaches.

1.2.4 Robust Path Planning and Motion Planning Under Uncertainty *(RQ2, RQ3, RQ5)*

A well-studied class of robust optimization problems of interest for unmanned systems path planning is that of robust path planning. The goal of a robust path planning problem is to determine the shortest path between two locations while subject to uncertainty about travel times and whether certain routes can be taken or not.

Robust path planning has been primarily studied through the robust formulation of the shortest path problem [17]. Linear programming (LP) models [17] [19] [55], dynamic programming based approaches [12] and mixed integer programming [127] [26] [111] have all been used for solving Robust shortest path planning problems, depending on the model used for uncertainty. The simplest form of robust shortest path planning problems is to assume that each edge in the graph being considered has its own worst case scenario that it does not share with any other edge in the graph. Thus, the robust shortest path problem simply becomes a conventional shortest path problem with each edge's cost being the cost under the worst case scenario for that edge [26]. However, this ignores any relations between the costs of edges, which is overly pessimistic. Chaerani et al. [26] shows that under an ellipsoidal uncertainty model for edge costs, the robust shortest path problem becomes a mixed integer conic quadratic programming problem (MICQP). Shahabi et al. [111] develops a more general MICQP formulation to model a robust shortest path planning problem that accounts for a set of uncertain parameters independent of the edges, and develops an outer approximation based approach for solving such problems.

Outside of [26] and [111], there are few works on robust shortest path planning problems that account for correlations between edge costs, and none of them account for non-linear effects caused by uncertain parameters. However, there are several papers on the stochastic (expected value) shortest path planning problem [13] that do aim to model more complex correlations between edge costs. Fan et al. [40] and Huang and Gao [57] consider models that can account for congestion like effects in traffic network type problems, with edge costs constraining other nearby related edge costs depending on their values. Ji et al. [60] considers the problem of determining the means and covariance of a multivariate normal distribution for use in a simulation based multiobjective genetic algorithm approach for solving multiobjective stochastic shortest path planning problem. Prakash and Srinivasan [93] develop an algorithm that does not need a predefined probability distribution based off sampling and performing network transformations in order to optimize robustness in a stochastic shortest path problem through a mean variance tradeoff objective.

Most works on robot motion planning under uncertainty cannot be classified as robust optimization, as they rely on knowing the probability distribution for any uncertain parameters that affect the environment or the robot (e.g [5]) and using that distribution to optimize a statistic such as expected cost. Alterovitz et al. [5] created a stochastic variation on PRM [64] using a Markov decision process model to consider probabilistic transitions between configurations in the motion planning graph. Du Toit and Burdick [39] considered the problem of dynamically changing uncertain environments via a stochastic dynamic programming approach.

Singh et al. [116] and Majumdar and Tedrake [78] used the concept of funnels to solve motion planning problems under uncertainty without needing a probability distribution, via methods from robust control. Garimella et al. [43] used sequential nonlinear model predictive control to ensure a robot stays within a "tube" of ellipsoids, in order to guarantee feasibility of motions under uncertainty. These methods consider robust optimal motion planning in terms of ensuring feasibility under uncertainty, but they cannot consider uncertainty in the cost of motions like methods developed for robust path planning can.

1.3 Research Gaps

So far in the literature, no work has considered the system level optimization of an unmanned system while incorporating design optimization, path planning and uncertainty in the same formulation.

Existing works on simultaneously optimizing the design and path of a system are subject to several limitations. Methods relying on co-design formulations ([20], [87], [49], [99]) are subject to both local optima present in the design optimization problem considered and also local optima present in the path planning component of the problem. The combined problem can thus contain combinations of these local optima. This can result in an extremely large number local optima being present, which worsens the quality of solutions found using local search methods. Methods relying on operations research formulations ([70], [86], [120]) can encounter scalability issues with computational cost when dealing with large graphs, such as those used in path planning or motion planning problems. Works which rely on using search based methods (e.g. RRT*) in conjunction with global optimization or discrete design searches (e.g. [9], [68]) can avoid these issues.

While a number of works ([33] [84] [9] [68][44]) have integrated design with trajectory or motion planning, only the RRT* variants used in [9] [33] [84] construct motion planning graphs to find the optimal motion sequence. All of these methods rely on either discrete design domains [33] [84], or global optimization techniques [9], both of which are poorly suited to handling robust optimization, which usually requires repeatedly solving optimization problems under different sets of uncertain scenarios. Additionally, while control based methods such as [44] could theoretically be extended into robust control based methods via the scenario based robust control design methods from [24], these methods would still suffer from issues with local optima.

A gap in the current literature on robust path planning is that the current state of the art cannot handle non-linear effects caused by uncertainty if there are also correlated edge costs. This prevents the consideration of uncertainty from a “black box” standpoint, something that is critical for complex system level design optimization problems, where designers cannot understand in advance how uncertainty will affect the problem. Additionally, it presents an issue for robust motion planning, where nodes in the path planning graph being constructed will become arbitrarily close with sufficient iterations. It is unrealistic to assume that the uncertainty affecting such nodes is uncorrelated, which prevents existing techniques for robust path planning from being used to enable robust motion planning.

The current literature on motion planning also has yet to consider the concept of robustness in the manner considered in the robust path planning literature, with most works considering uncertainty either using some form of risk metric or only attempting to guarantee the feasibility of motions under uncertainty. This relates to the gap in considering "black box" robustness in the robust path planning literature discussed earlier, as sampling based motion planning techniques work by building a graph for path planning, which largely ensures that correlations will be present between the edges in that graph. Thus, robust motion planning requires first solving the currently unsolved problem of "black box" robustness for discrete robust path planning.

There are also a number of minor gaps in current literature on UAV motion planning that are important for considering the integration of design and motion planning for system level objectives. Previous work on motion planning for fixed-wing UAVs used three-dimensional extensions to the Dubins car model, which allows for planning using a set of time optimal motion primitives ([29], [89]). However, risk-based motion planning requires considering a larger configuration space. Additionally, the approaches in [29] and [89] are only optimal for a time objective under very specific assumptions, as they assume that the optimal Dubins curve in two dimensions is still the optimal curve in three dimensions, which will be shown to be false in Chapter 4. Methods for using Dubins curves for UAV motion planning are typically sampling based methods, with the Dubins curves being used as the method for solving the BVPs between different configurations. These methods typically construct highly connected graphs, with each node being connected to a large of neighbors, resulting in large numbers of edges. This poses an issue for integrating robust optimization into motion planning, as all of these edges would have costs affected by uncertainty that would need to be repeatedly computed in different scenarios. This would result in a significant increase in computational cost over deterministic motion planning, unless measures are taken to control the number of edges in the graph and to avoid needing to compute the costs of all edges in every scenario considered.

1.4 Organization

This dissertation is organized into the following chapters which are summarized in this section. Figure 1.1 provides a visual representation as to how each of the chapters in this dissertation build off each other.

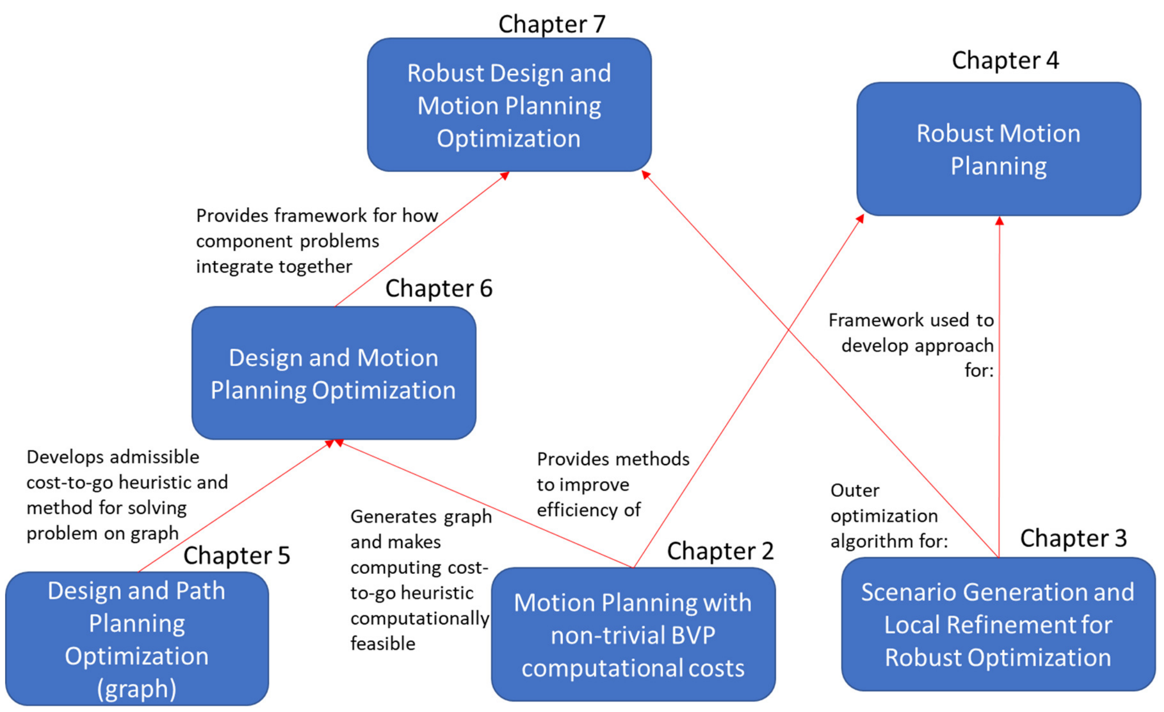

Chapter 7
Robust Design and Motion Planning Optimization
Chapter 4
Robust Motion Planning
Provides framework for how component problems integrate together
Chapter 6
Design and Motion Planning Optimization
Framework used to develop approach for:
Develops admissible cost-to-go heuristic and method for solving problem on graph
Provides methods to improve efficiency of
Outer optimization algorithm for:
Generates graph and makes computing cost-to-go heuristic computationally feasible
Chapter 5
Design and Path Planning Optimization (graph)
Chapter 2
Motion Planning with non-trivial BVP computational costs
Chapter 3
Scenario Generation and Local Refinement for Robust Optimization

Figure 1.1: Visual depiction of how material from each chapter in this dissertation is used in other chapters

1.4.1 Chapter 2 – Risk-based Motion Planning for UAVs

The first problem considered in this dissertation is a UAV motion planning problem that optimizes a UAV's motions in order to minimize a risk objective. In the process of solving this problem, a method for solving motion planning problems for UAVs is developed that reduces the number of boundary value problems (BVPs) that need to be solved to converge to the optimal motion sequence. This method also extends current motion planning techniques by developing a method for utilizing initial conditions in optimal sampling based motion planning and by developing a new method for determining the connection radius when connecting states in the configuration space. Results are demonstrated in the context of managing third party risk for UAVs via motion planning.

1.4.2 Chapter 3 – Feasibility Robust Optimization via Scenario Generation and Scenario Reduction

This chapter presents a new feasibility robust optimization approach involving uncertain parameters defined on continuous domains. The proposed approach is based on an integration of two techniques: (i) a sampling-based scenario generation scheme and (ii) a local robust optimization approach. An analysis of the computational cost of this integrated approach is performed to provide worst-case bounds on its computational cost. The proposed approach is applied to several non-convex engineering test problems and compared against two existing robust optimization approaches. The results show that the proposed approach can efficiently find a robust optimal solution across the test problems, even when existing methods for non-convex robust optimization are unable to find a robust optimal solution. A scalable test problem is solved by the approach, demonstrating that its computational cost scales with problem size as predicted by an analysis of the worst-case computational cost bounds.

1.4.3 Chapter 4 – Cost Robust Motion Planning

The methods developed in Chapter 3 are utilized to develop a method for solving of robust path planning problems on motion planning graphs, where uncertainty affects the costs of edges. Unlike prior literature on robust path planning, the robust path planning problem considered allows for a black-box model of correlations between the uncertain edge costs in the graph. Results are presented comparing the improvements in worst case performance of the new method relative to existing robust and non-robust motion planning methods.

1.4.4 Chapter 5 – UAV Design and Path Planning Optimization

This chapter considers the problem of design and path planning optimization on a graph. An admissible cost-to-go heuristic is developed for the problem of UAV design and path planning on a fixed graph, which is able to heavily reduce computational effort needed for large scale graphs. The proposed cost-to-go heuristic operates by solving the UAV design and path planning problem under the assumption that each edge takes the cost associated with its best possible design, which may not be the same design for each edge. It is shown that the problem of UAV design and path planning on a graph can be solved to local optimality using branch and bound methods if this heuristic is used, however these methods are shown to be computationally inefficient compared to a new algorithm (VDPPA) which is significantly faster and capable of usually finding the same solution that branch and bound would. Results are demonstrated in the context of managing third party risk for UAVs through design and path planning optimization.

1.4.5 Chapter 6 – UAV Design and Motion Planning Optimization

This chapter considers the problem of UAV design and motion planning, using the following approach: The cost-to-go heuristic earlier will be computed by solving the problem using a sampling based motion planner, leveraging the techniques developed for reducing the number of BVPs solved to avoid the computational issues that would occur if current motion planning algorithms were used instead. That solution and the graph created by the motion planner will then be used by VDPPA (see Chapter 5) in order to solve the design and path planning problem in terms of actual motions. Results are demonstrated in the context of managing third party risk for UAVs, using the same example considered in Chapter 2 and Chapter 4.

1.4.6 Chapter 7 – Robust UAV Design and Motion Planning Optimization

This chapter adds uncertainty to the design and motion planning problem considered in Chapter 6, using the same model for uncertainty considered in Chapter 4. Only objective robustness is considered, meaning that uncertainty will not affect the feasibility of solutions to the problem being considered. Results are demonstrated in the context of managing third party risk for UAVs and compared against the results computed in Chapter 6 when not accounting for uncertainty.

Chapter 2: Risk-based Motion Planning for UAVs

This chapter has appeared at the 2019 International Conference on Unmanned Aerial Systems as [102].

This chapter presents a risk-based motion planning approach. Although it is motivated by the problem of fixed-wing UAV takeoff planning, it can be applied to other risk-based motion planning settings for UAVs (e.g. multi-copter UAVs, or planning trajectories for phases other flight than takeoff). This approach extends the risk-based path optimization approach introduced by [103], which planned a path over a discrete graph and used only one crash probability distribution (CPD), which is needed to compute the risk. Because the altitude, orientation, and speed of the UAV affect both the feasibility of a UAV's motions and the location it would hit the ground if it crashed, risk-based motion planning should consider these variables. The risk-based motion planning approach considers two objectives: the third-party risk and the time to reach the goal.

This chapter is organized as follows. Section 2.1 formulates a risk-based UAV motion planning problem. Section 2.2 presents the proposed approach for solving risk-based UAV motion planning problems. Section 2.3 details the example problem used to test the proposed approach in Section 2.2. Section 2.4 presents the results for the example problem. Section 2.5 discusses the results from the example. Section 2.6 summarizes the results of this Chapter and presents some concluding remarks.

2.1 Problem Statement

This section formulates a motion planning problem that requires planning a sequence of motions (a trajectory) in configuration space that minimizes flight time and the risk posed by a UAV flying over inhabited areas. Unlike past work [94], this chapter considers the 3D problem that includes the dynamics constraints for a fixed-wing UAV. Including these constraints requires planning motions in a five-dimensional (5D) configuration space (3D plus yaw and pitch). This expanded configuration space allows for consideration of how different UAV flight states affect the risk it poses, which allows the UAV to mitigate risk through both flight maneuvers and avoiding populated areas.

2.1.1 Notation

Let x, y, and z be the coordinates of the vehicle's location. Let ψ be the vehicle's yaw (heading). Let θ be the vehicle's pitch. Let C be the set of all feasible configurations, where a configuration $c \in C$ consists of (x, y, z, ψ, θ) (the vehicle's roll was treated as a motion dependent variable, see Section 2.2.3). Let c_s be the vehicle's initial configuration; let c_f be the desired final configuration. For c_1 and $c_2 \in C$, let $B(c_1, c_2)$ be the set of all possible solutions to the BVP between configurations c_1 and c_2, where $s \in B(c_1, c_2)$ is a continuous sequence of configurations from c_1 to c_2 that satisfies all of the dynamics constraints.

Let v be the vehicle's speed (which is fixed). Let $f_t(s)$ be the time needed to move along s. Let $f_r(s)$ be the third-party risk [103] created by moving along s. Let w_t and w_r be non-negative weights ($w_t + w_r = 1$) on $f_t(s)$ and $f_r(s)$.

2.1.2 Formulation

The formulation for the risk based UAV motion planning problem proposed is as follows:

Given the set C, the initial and final configurations c_s and c_f, and the weights w_t and w_r, find the trajectory $s \in B(c_s, c_f)$ that minimizes the total cost $w_t f_t(s) + w_r f_r(s)$.

By varying the weights w_t and w_r, different trajectories can be generated, which can be used to construct the best non-dominated set of trajectories that minimize the risk posed by the UAV and its flight time.

2.1.3 Cost Functions

2.1.3.1 Flight Time

To calculate the flight time $f_t(s)$ the UAV needs to travel trajectory *s*, the length of trajectory *s* is divided by the UAV's speed *v*.

2.1.3.2 Third-Party Risk

The risk $f_r(s)$ was determined using a UAS third party risk measure [103]. This risk measure required a probability distribution, called a crash probability distribution (CPD), over the possible locations where the UAS would crash into the ground, which is a function of the UAS's height above ground level, speed, and attitude at the time that it loses power and begins gliding to the ground uncontrollably. Appendix A describes the approach for estimating the crash probability distribution (CPD). Several examples of these CPDs are shown in Figure 2.1.

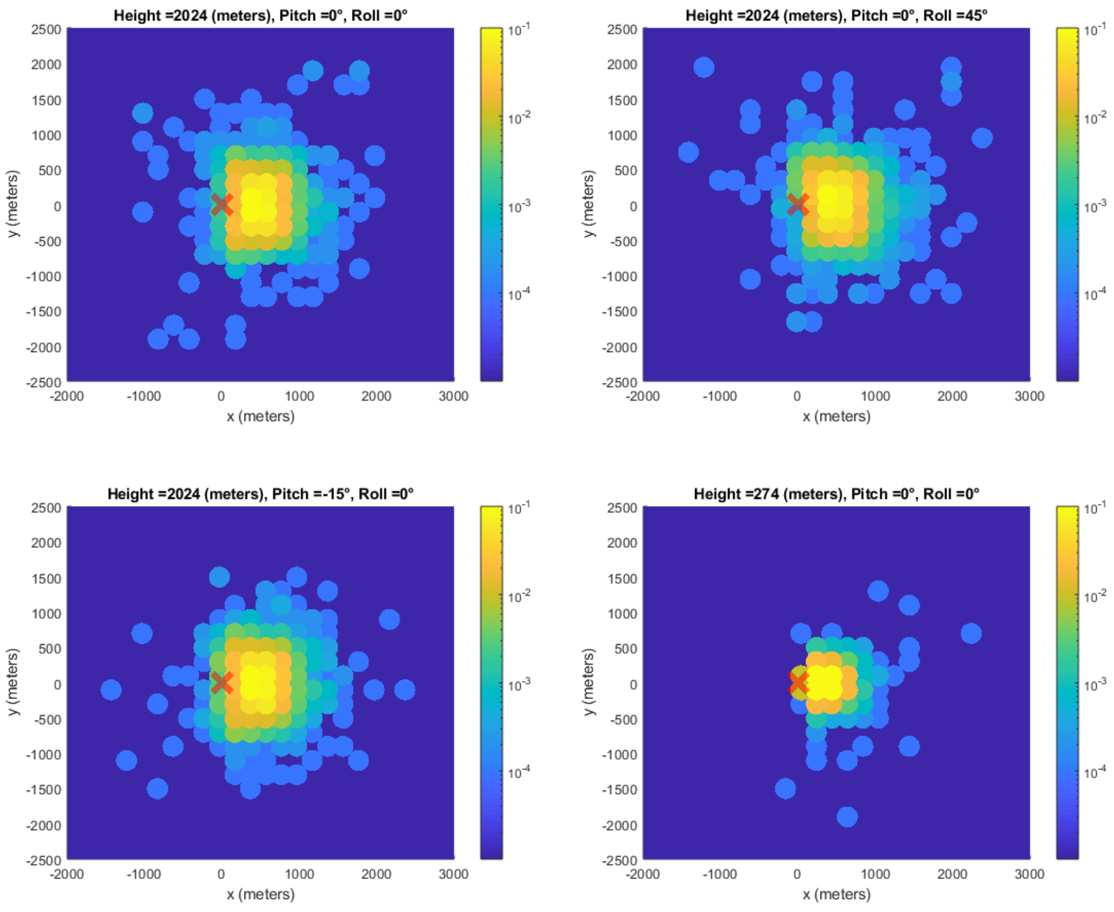
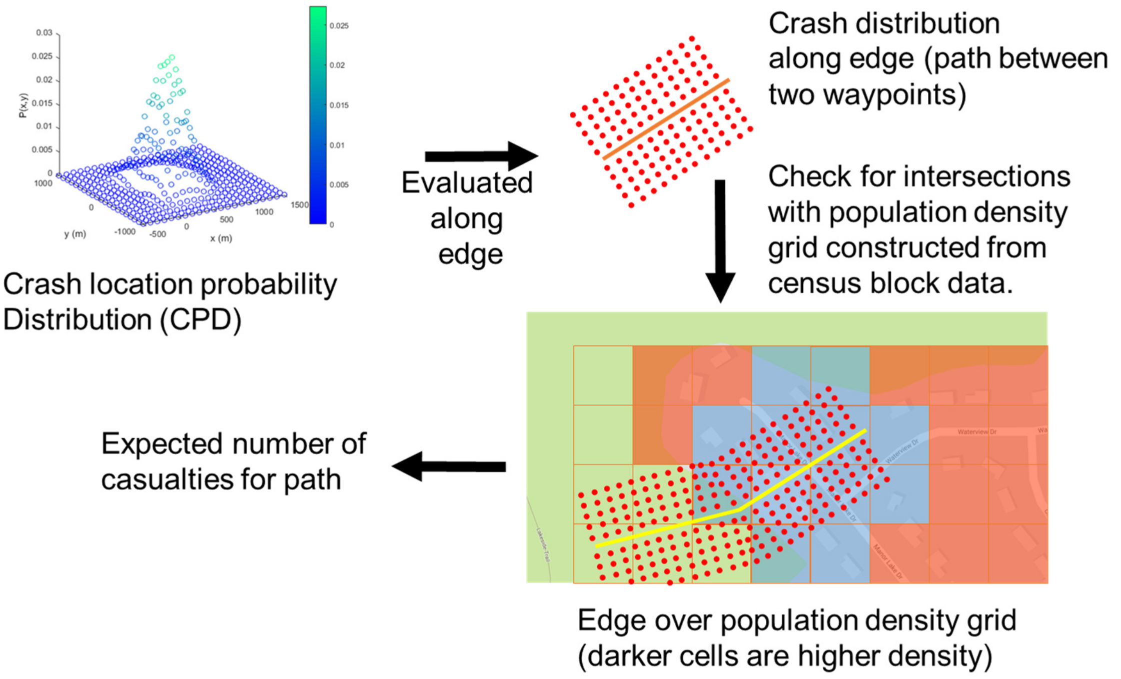


Figure 2.1: (a) Example Crash Distribution for a UAV at a high height while pitched. (b) Example Crash Distribution for a UAV at a high height in level flight. (c) Example Crash Distribution for a UAV at a high height while banking. (d) Example Crash Distribution for a UAV at a low height in level flight.

The risk measure of [103] discretizes a trajectory into a set of legs. In this chapter, a leg is treated as a curve between two configurations (i.e. one edge in the motion planning graph used by RRT$^{\#}$, unlike the straight line paths considered in [103]. Each leg was then discretized into a series of points and evaluated in the same manner as in [103]. For each point along a leg, the CPD is discretized into a set of points and the population is evaluated at each of the CPD's points. These population values are multiplied by the probability of crashing at each of the CPD's points and the sum of these values is combined with an estimate of the UAS's failure rate. The resulting quantity is an estimate of the expected number of fatalities that would be caused by the trajectory in question. Figure 2.2 visually illustrates this process for evaluating 3 points along a leg.



Crash location probability Distribution (CPD)
Evaluated along edge
Crash distribution along edge (path between two waypoints)
Check for intersections with population density grid constructed from census block data.
Expected number of casualties for path
Edge over population density grid (darker cells are higher density)
P(x,y)
x (m)
y (m)

Figure 2.2: Illustration of risk metric from [103]. The crash distribution is evaluated at the red points, which spaced along each leg (yellow lines) of the trajectory. Areas with population that UAS could crash in are highlighted blue, areas with population the UAS avoids are highlighted red, green denotes uninhabited areas.

The number of points at which the CPD was evaluated at was set based on the length of each leg (the distance between the start and end configurations), at a ratio of 50 points per kilometer and with a minimum number of 2 points (the starting point and ending point of the leg). However, when determining the optimal trajectory between two configurations (see Section 2.2.3), a minimum of 10 points were used when assessing different possible trajectories. This ensured that the minimum cost trajectory was always chosen. However, the cost of that trajectory was still assessed normally (minimum of 2 points instead of 10), which ensured that the risk for each trajectory considered during motion planning was computed to the same accuracy.

2.2 Solution Approach

Several modifications were made to the RRT$^{\#}$ algorithm described by Arslan and Tsiotras [8], in order to make it more efficient at solving the risk-based motion planning problem considered.

2.2.1 Notation

Let $x_{init} = c_s$ be the start configuration. Let $x_{goal} = c_f$ be the goal configuration. Let $\mathrm{X} = C$ be the configuration space. Let p_n and p_b be probabilities used in the sampling routine. Let N be the number of iterations. Let $\mathrm{G} = (V, E)$ be the graph that the RRT$^{\#}$ algorithm builds. Let x be a configuration in X . Let r be the connection radius. Let γ_0 and γ_{min} be the bounds on the connection radius.

2.2.2 Algorithm

Table 2.1 lists the modified RRT$^{\#}$ algorithm. For the “Modified Extend” algorithm, Table 2.2 lists the modified lines and provides references to the lines unchanged from the original

"Extend procedure (Algorithm 4 in [8]). Table 2.3 lists the "GET LOCAL RADIUS" procedure. Table 2.4 lists the "RANDOM CONFIG" procedure.

Table: 2.1: Modified RRT# Algorithm

```
 1: function MODIFIED RRT#($x_{init}, x_{goal}, \mathcal{X}, p_n, p_b, N$)
 2:     $V \leftarrow \{x_{init}, x_{goal}\}, E \leftarrow \{(x_{init}, x_{goal})\}$
 3:     PARENT($x_{goal}$) $\leftarrow x_{init}$
 4:     $\mathcal{G} \leftarrow (V, E)$
 5:     for all $k = 1$ to $N$ do
 6:         $x_{sample} \leftarrow$ SAMPLE($k$)
 7:         $r_{local} \leftarrow$ GET LOCAL RADIUS($x_{sample}, \gamma_0, \gamma_{min}$)
 8:         $v_r =$ RANDOMVALUEBETWEEN($0, 1$)
 9:         if $0 \leq v_r < p_n$ then
10:             $x_{rand} \leftarrow x_{sample}$
11:         end if
12:         if $p_n \leq v_r < p_n + p_b$ then
13:             $x_{nearest} \leftarrow$ NEAREST($\mathcal{G}, x_{sample}$)
14:             $x_{rand} \leftarrow$ RANDOM CONFIG IN BALL($x_{nearest}, r_{local}$)
15:         end if
16:         if $p_n + p_b \leq v_r \leq 1$ then
17:             $(x_{rand}, \mathcal{G}) \leftarrow$ SAMPLE ON TRAJECTORY($\mathcal{G}, x_{init}, x_{goal}$)
18:         end if
19:         $\mathcal{G} \leftarrow$ MODIFIED EXTEND($\mathcal{G}, x_{rand}, r_{local}$)
20:         REPLAN($\mathcal{G}, x_{goal}$)
21:     end for
22:     $(V, E) \leftarrow \mathcal{G}, E' \leftarrow \emptyset$
23:     for all $x \in V$ do
24:         $E' \leftarrow E' \cup \{(\text{PARENT}(x), x\}$
25:     end for
26:     return $\mathcal{T} = (V, E')$
27: end function
```

Table: 2.2: Modified Extend Algorithm

```
   1: function MODIFIED EXTEND($\mathcal{G}, x, r$)
   2:     $(V, E) \leftarrow \mathcal{G}, E' \leftarrow \emptyset$
   3:     $\mathcal{X}_{near} \leftarrow$ NEAR($\mathcal{G}, x, radius \leftarrow r$)
4-13:     Lines 8-19 from EXTEND [22]
  14: end function
```

The "RANDOM VALUE BETWEEN(a, b)" procedure randomly selects a value from a uniform distribution between a and b. The "SAMPLE" procedure randomly selects a configuration in X. The "RANDOM CONFIG IN BALL(x, r)" procedure randomly selects a configuration in X within r units of x. Subsection 2.2.3 describes the

"SOLVE BVP" procedure. Other procedures not included here are equivalent to those described by Arslan and Tsiotras [8]. The algorithm used the third cost based vertex inclusion criteria ($\mathrm{RRT}_3^{\#}$) proposed in [8], by only updating the costs of configurations with lower costs than the current cost of the goal configuration.

In each iteration, the modified RRT# algorithm randomly selects a procedure for sampling a new configuration. The probability that it uses "SAMPLE" equals p_n; the probability that it uses "RANDOM CONFIG IN BALL" equals p_b; and the probability that it uses "RANDOM CONFIG" equals $1 - p_n - p_b$.

2.2.3 Modeling of 3-D Dubins curves for multiple objectives

The RANDOM CONFIG procedure requires solving a BVP between two configurations ("SOLVE BVP"). Because the cost function includes both time and risk, the BVP was solved using the following method for generating three-dimensional Dubins curves.

1. The two-dimensional Dubins curves (LSR, LRL, etc.) are computed for going between the x, y coordinates and headings of the start and goal configurations.

2. For each curve (LSR, LRL, etc.), let θ_{max} be the maximum allowable pitch angle, let θ_{min} be the minimum pitch angle for climbing in a helix, let the length of the curve be L_L and let the change in height required to go between the start and goal configurations be h, then the 3-D equivalent of that curve will be:

a. If $\sin^{-1}(h/L_L) \le \theta_{max}$ then the 3-D equivalent curve is just the 2-D curve with a constant increase of L_L/h in height along the curve. Thus the pitch angle for these curves is $\sin^{-1}(h/L_L)$.

b. If $\sin^{-1}(h/L_L) > \theta_{max}$, then several different equivalent 3-D curves can exist. These curves consist of changing either the first or second turn in the 2-D curve into a helix and then making the curve increase in height at a constant slope of $(L_L + 2\pi rn)/h$, where r is the radius of the Dubins curves and n in the number of helix loops. In order to consider different pitch angles for motions, the maximum and minimum number of helix loops are determined from θ_{max} and θ_{min}, and several different numbers of turns between those are used to generate additional possible 3-D curves. Thus the pitch angle for these curves is $\sin^{-1}(h/(L_L + 2\pi rn))$.

3. The lowest cost curve is chosen as the optimal curve.

This process is repeated for several different radii, with each radius corresponds to a different roll angle needed to achieve a banked turn at that radius. For each curve, five different radii and five different numbers of turns were considered in order to determine the minimum cost path between two configurations.

Figure 2.3 show several examples of Dubins curves generated between the same start and end point for different objectives through the proposed methodology. Note that the time optimal case in Figure 2.3a manages to avoid needing to perform a helix turn when going to a higher height by changing the underlying Dubins curve being used, this is an improvement over the approach from [29] as while the LRL motion used in Figure 2.3a is the shortest path between the two points depicted in 2-D, in 3-D at the height considered in Figure 2.4 an RSR motion is actually the true time optimal curve. The approach proposed in [29] assumes that the optimal 2-D motion is still the optimal motion in 3-D, which is not actually the case here.

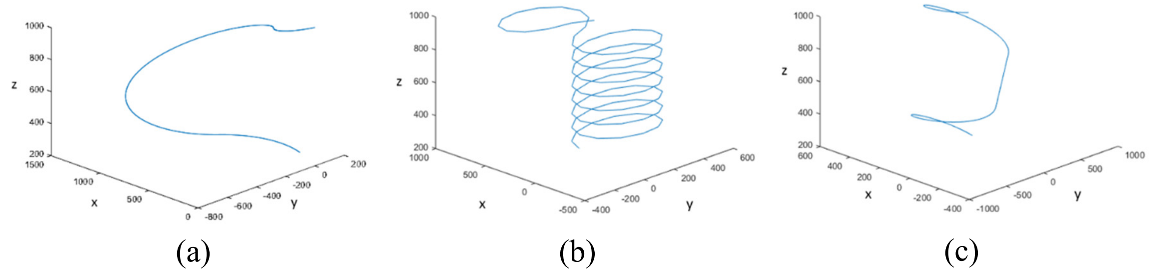


Figure 2.3: Example 3-D Dubins curves for (a) time optimal, (b) risk optimal and (c) half weight on both time and risk objectives. Note that all 3 curves share the same start and end points, but have different underlying 2-D Dubins curves ((a): RSL, (b): LRL, (c): LSL).

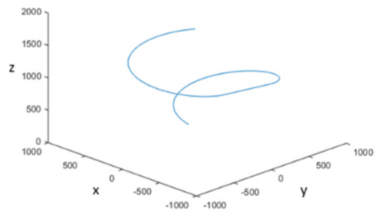


Figure 2.4: Time optimal Dubins curve for end point with same x,y locations as from Figure 2.3a, but with higher destination height. Note that the time optimal 3-D Dubins curve has changed from LRL to RSR due to the height increase.

2.2.4 Determining the Local Connection Radius

Because the total cost includes the risk metric, solving the BVP requires evaluating the risk for a minimum 10 points (see Section 2.1.3.2) on each trajectory considered and the CPDs at those points need to be combined with the population density data in order to compute the risk metric. Additionally, the risk metric has to be recomputed for the solution to the BVP at the specified point density of 50 points for every kilometer of curve length. Consequently, significantly more computational time is needed to evaluate the risk metric than to compute the time needed to traverse a trajectory.

A BVP must be solved every time that the modified RRT$^{\#}$ algorithm attempts to connect an existing configuration to a new configuration, this will be done an excessive number of times if the connection radius is too large. However, the connection radius must still be large enough to ensure that any new configuration can be connected. To avoid this problem and limit the number of times that it must do this, the modified RRT$^{\#}$ algorithm dynamically determines the local connection radius using a new procedure: "GET LOCAL RADIUS") (Table 2.3).

Table 2.3: Get Local Radius Algorithm

```
1: function GET LOCAL RADIUS($x, \gamma_0, \gamma_{min}$)
2:     $nnode \leftarrow$ FIND NEAREST NODE($x$)
3:     $nNeighbors \leftarrow |$NEIGHBORS$(nnode)|)$
4:     if $nNeighbors \leq 2$ then
5:         return $\gamma_0$
6:     end if
7:     $r_{RRT} \leftarrow \gamma_0(\frac{\log(nNeighbors+1)}{nNeighbors+1})^{1/d}$
8:     $rSum \leftarrow 0$
9:     $count \leftarrow 0$
10:    for all $k \in$ NEIGHBORS$(nnode)$ do
11:        if $||nnode - k|| \leq r_{RRT}$ then
12:            $rSum \leftarrow rSum + ||nnode - k||$
13:            $count \leftarrow count + 1$
14:        end if
15:    end for
16:    $r_{avg} \leftarrow rSum/count$
17:    $r_{dist} \leftarrow 2||x - nnode||$
18:    return $\max\{\gamma_{min}, \min\{\gamma_0, \max\{r_{avg}, r_{dist}\}\}\}$
19: end function
```

“GET LOCAL RADIUS” sets the connection radius based on the new configuration's location relative to the existing configurations. When the new configuration is in an unexplored region, the connection radius will be larger, which makes it easier to connect it to the search tree. When the new configuration is close to existing configurations, the connection radius will be smaller to limit the number of existing configurations that must be considered (and the number of BVPs that must be solved). Conceptually, this can be viewed as using the same equation that RRT* and RRT$^{\#}$ use to determine connection radius, but using the number of neighboring nodes (nodes with edges going to a configuration) instead of the total number of nodes in the motion planning graph. The number of neighboring nodes of the new configuration is approximated using the number of neighbors of the configuration closest to the new configuration. Because number of neighbors of a node remains lower than the total number of nodes in the motion planning graph, it is practical to choose a value for *d* lower than that of the dimension of the configuration space. For the risk based motion planning problem in the 5D configuration space considered, $d = 4$ produced better results than using $d = 5$.

2.2.5 Sampling Configurations from a Given Path

Because there are no obstacles to avoid in the risk-based motion planning problem (only high-risk areas), a minimal time solution can be found by solving the BVP between x_{init} and x_{goal}. When $w_t > 0$, adding configurations from this minimal-time solution to the search graph G should help the modified RRT# algorithm find better solutions. To do this, it occasionally uses the "SAMPLE ON TRAJECTORY" procedure (Table 2.4) to get such a configuration. When a configuration is sampled from the current solution, it is used to split an edge in G into two edges, which necessitates the removal of the original edge from G. "SAMPLE ON TRAJECTORY" always splits the longest part of the current solution, which prevents any bias in which configurations on the current solution are sampled.

Table 2.4: Random Config Algorithm

```
function SAMPLE ON TRAJECTORY($\mathcal{G}, x_{init}, x_{goal}$)
    $d_{\max} \leftarrow 0$
    $node \leftarrow x_{goal}$
    $node_{\max} \leftarrow x_{goal}$
    while $node \neq x_{init}$ do
        if SOLVE BVP(PARENT($node$), $node$).$Length \geq d_{\max}$ then
            $node_{\max} \leftarrow node$
            $d_{\max} \leftarrow$ SOLVE BVP(PARENT($node$), $node$).$Length$
        end if
        $node \leftarrow$ PARENT($node$)
    end while
    $path_{\max} \leftarrow$ SOLVE BVP(PARENT($node$), $node$)
    $result \leftarrow path_{\max}$(RANDOMVALUEBETWEEN(0, 1))
    $\mathcal{G} \leftarrow$ ADD EDGES($\mathcal{G}$, [(PARENT($node$), $result$), ($result$, $node$)])
    $\mathcal{G} \leftarrow$ REMOVE EDGE($\mathcal{G}$, (PARENT($node$), $node$))
    return ($result$, $\mathcal{G}$)
end function
```

2.2.6 Locally Biased Sampling

The modified RRT[#] algorithm also occasionally gets a new configuration by sampling the region near the nearest configuration (the one that is closest to the sampled configuration). This increases the likelihood that the newly sampled configuration will be able to be connected to the search tree. The radius of the ball around the nearest configuration equals the connection radius r_{local} determined by "GET LOCAL RADIUS". See lines 10-11 in Table 2.1.

2.3 Example Problem

This section details an example used to evaluate the risk-based motion planning approach. This example problem will also be used later in Chapters 4, 6 and 7.

2.3.1 Problem Instance

This problem instance included the population data for the state of Maryland from the 2010 U.S. Census (United States Census Bureau 2010), which is depicted in Figure 2.5. Table 2.5 lists the start and goal configurations and the bounds on the configuration space. The vehicle's flight speed was fixed at 50 m/s. Eleven combinations of weights were used: $(w_t, w_r) = (0,1),(0.1,0.9),\dots,(1,0)$. The modified RRT[#] algorithm was run with $N = 3000$ iterations for each combination of weights. For selecting a method for randomly sampling a configuration (random configuration, local bias and sampling on the current optimal trajectory), the selection probabilities $p_n = p_b = 0.45$. The initial connection radius

$\gamma_0 = 2000$, and the minimum connection radius $\gamma_{\min} = 200$. A Cessna 182 was used as the UAV model because its flight coefficients are publicly available and documented ([101]).

Table 2.5: The start and goal configurations and the bounds on the configuration variables for the experimental case

	x (m)	y (m)	z (m)	ψ (deg.)	θ (deg.)
Start	74,570	65,770	274	0	0
Goal	56,000	58,000	2,024	0	0
Min	52,500	52,500	274	- π	-15
Max	80,000	70,000	2,024	π	15

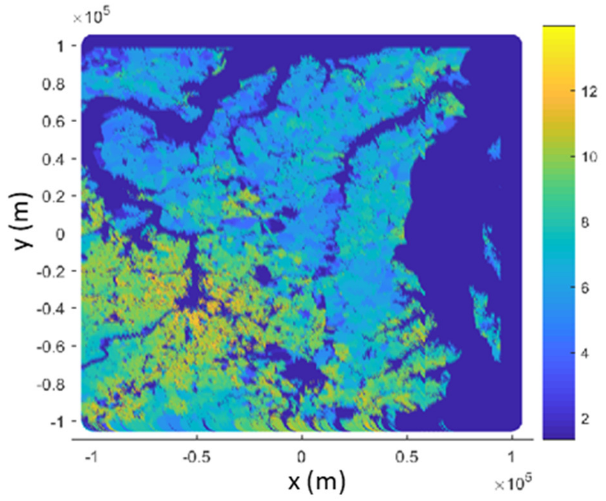


Figure 2.5: Plot of subsection of Maryland region in which experimental case is located in. Note that the region is upside down (south is up, north is down) relative to a normal map. The colorbar denotes the natural logarithm of the population density in the region, computed from 2010 US census block data [122].

2.3.2 Crash Distributions for a UAV Under Varying Orientations

To model how a UAV's configuration affects the CPD, a series of Monte Carlo simulations were conducted to construct CPDs for 125 different design configurations from the combinations of five heights, five roll angles, and five pitch angles. The range for the height was 274 meters to 2024 meters. The range for roll was -45 ° to 45 °. The range for pitch was -15 ° to 15 °. The roll angle range was set to a reasonable amount of roll for safely performing banked turns. The pitch angle range corresponds to the maximum angle of attack for a Cessna 182. Each CPD was represented as a grid of 2,500 bins. (The values of x, y, and ψ affect only the position and orientation of the CPD, not the likelihood of its bins.)

Figures 2.1a, 2.1b, 2.1c and 2.1d show some examples of these CPDs. As shown in Figure 2.1b, at high altitudes, the vehicle's pitch has a strong effect on the shape of the CPD.

As shown in Figures 2.1b and 2.1d, the vehicle's height above ground level affects the spread of the CPD. As shown in Figure 2.1c, the vehicle's roll angle affects the shape of the CPD.

A k-nearest neighbors (KNN) algorithm was then used to determining the CPD for any other configuration (and the corresponding derived roll angle). This algorithm found, from the set of 125 design configurations, the six nearest design configurations and combined the corresponding CPDs by averaging the likelihoods for each bin. Full details on the KNN approach used can be found in Appendix A, Section A.4.3. Equal weight was given to all input parameters and an inverse distance weighting exponent of 1.5 was used.

2.4 Results

Solving the problem with 11 combinations of the weights yielded 11 solutions. Six of these were dominated on the risk and time metrics. Figure 2.6 shows the risk and time performance of the five non-dominated solutions. The extreme solutions perform very well on one metric and very poorly on the other. The remaining solutions also show a tradeoff between risk and time. Figure 9a and 9b shows two dimensional projections of the search trees created by the modified RRT$^{\#}$ algorithm. Note that the tree in Figure 2.7a covers the areas which have low population density, which correspond to lower risk, while the tree 2.7b simply covers the space of configurations reachable in less time than the goal. These differences illustrate the conflict between the risk and time objectives seen in Figure 2.6, as these two search trees cover very different regions.

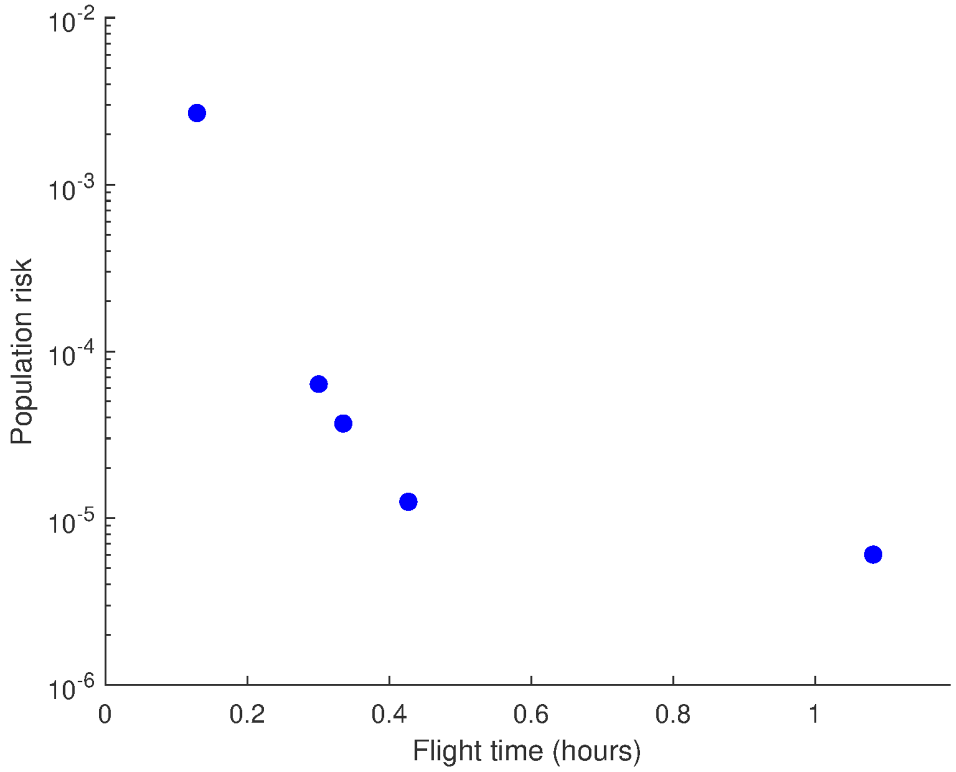
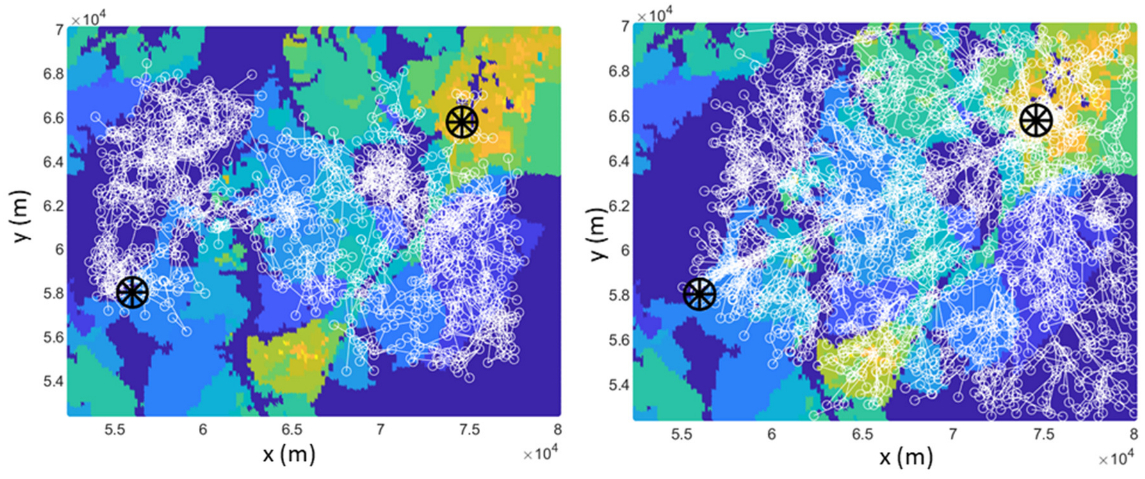


Figure 2.6: Pareto frontier of non-dominated solutions to the example problem for risk and time objectives, note that risk objective is on a log scale



(a) (b)

Figure 2.7: Projection of search tree for all weighting on the (a) risk (b) time objective of RRT# into XY plane, the while circles denote the start and goal configurations, brighter colors indicate higher risk regions

Figure 2.8 shows the trajectory found which posed the least risk; this trajectory reduces risk by flying over uninhabited areas such as bodies of water. It also performs spiraling motions to control the size of the UAS's CPD through changes in height, a more compact distribution can easily avoid populated areas when population is low, whereas a spread out distribution better distributes the risk posed when flying directly over heavily populated areas. A large number of maneuvers are also performed over a lower risk area near the starting point, in order to manipulate the height of the UAS. However, many of these height change maneuvers require a great deal of time to execute (due to the length they add to the flight trajectory), despite the reduction in risk they provide. This occurs because the lowest risk solution comes from when $(w_t, w_r) = (0; 1)$, meaning no weight is being put on minimizing flight time.

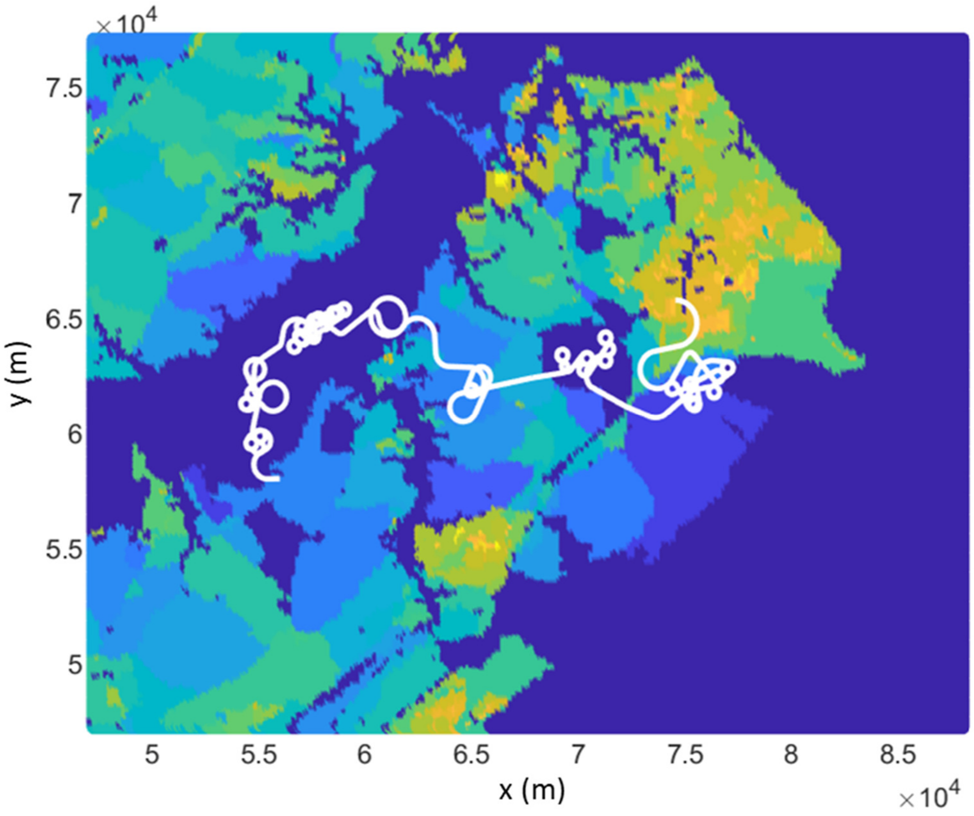


Figure 2.8: Trajectory of the Pareto Optimal solution with the best performance on the risk objective, brighter colors indicate higher risk regions. Note that the distance scale in this figure is 10,000 meters.

Figure 2.9 shows the trajectory found with (w_t, w_r)= (0:8; 0:2); this trajectory avoids high risk areas, while avoiding the time consuming manuevers performed by the risk optimal solution. The trajectory in Figure 11 initially follows a path similar to the time optimal solution (which is the initial solution used at (w_t, w_r) = (0:8; 0:2)), however the trajectory performs several maneuvers the uninhabited area near the start in order to avoid populated areas. However, unlike the time optimal trajectory, the trajectory in Figure 2.9 dives down (see Figure 2.10) in order to lower its height (and reduce the size of its CPD) when flying near several inhabited areas. The trajectory also loops over itself around this point, which shifts the CPD towards the center of the trajectory's turns, keeping the CPD away from an area of high population. The trajectory then moves out over the Patuxent River (which is uninhabited) and climbs to its final altitude at the specified goal configuration. These maneuvers allow the trajectory in Figure 2.9 to perform significantly better on the risk objective than the time optimal solution, while also using significantly less flight time than the risk optimal solution.

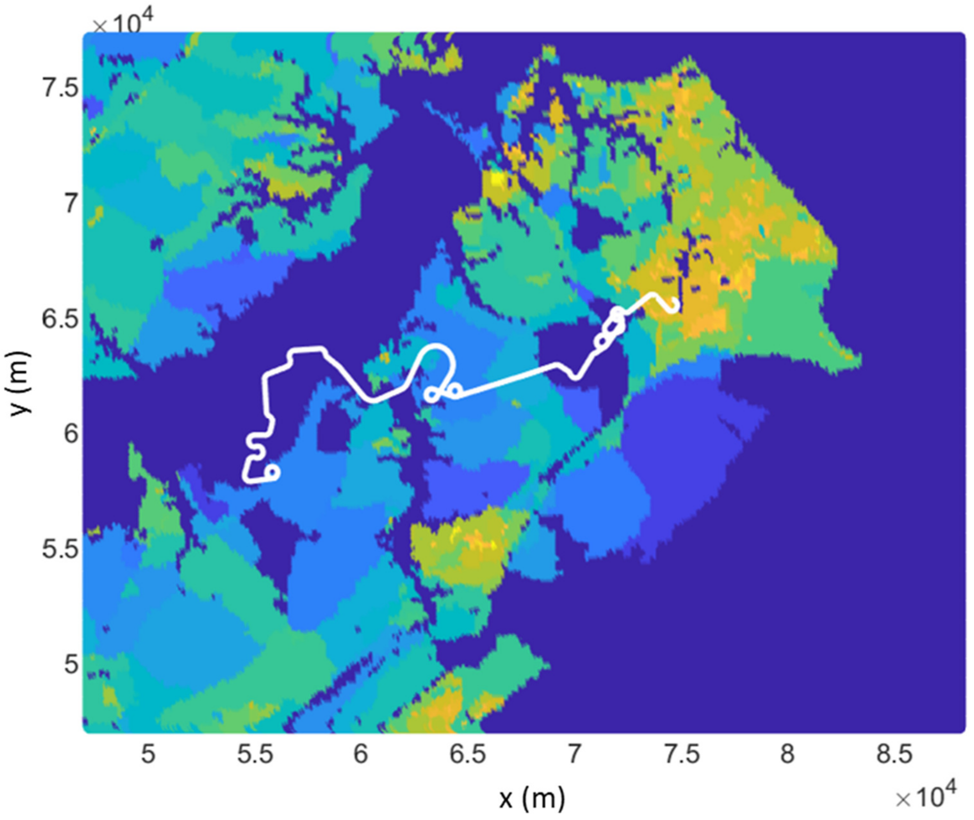


Figure 2.9: Trajectory of the Pareto Optimal solution with the third best performance on the time objective, brighter colors indicate higher risk regions

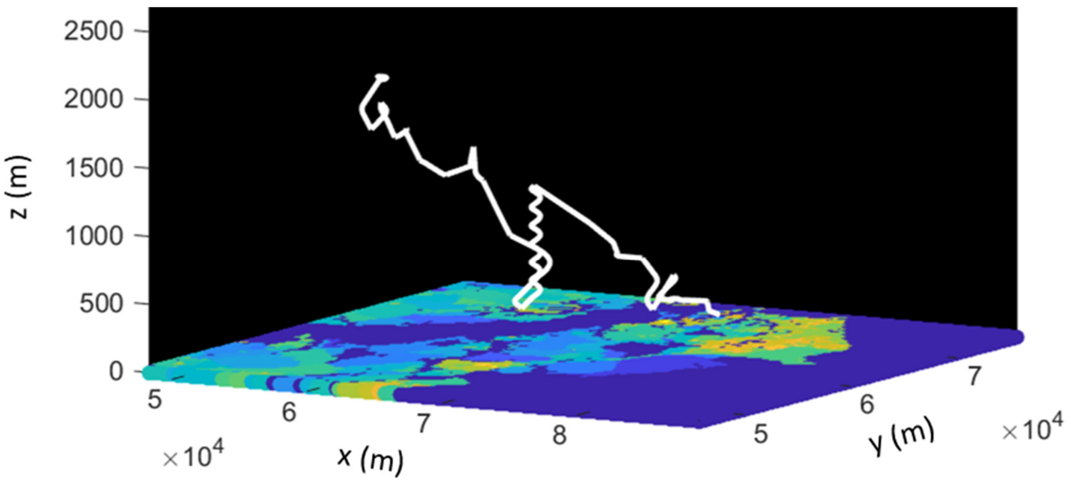
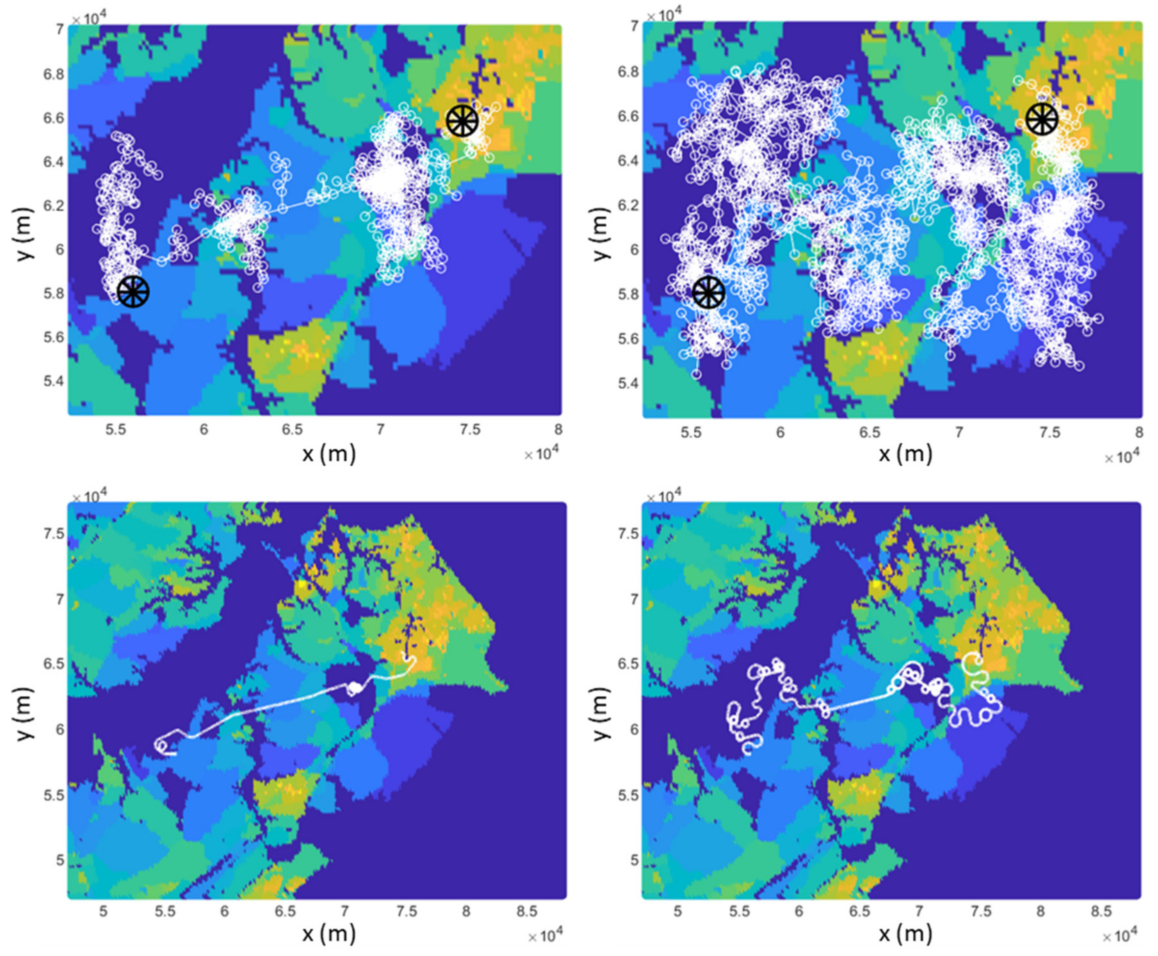


Figure 2.10: 3-Dimensional view of the trajectory from Figure 2.9.

To assess the effect of the proposed alternate connection radius for RRT#, the example problem was also solved using the same sampling scheme proposed in Table 2.1 with all weight put on the risk objective, but using the normal equation used for the connection radius for RRT* and RRT# from [63]. Two different values for the initial connection radius γ_0 (2000 and 3000) were used to illustrate the issues with the normal equation for connection radius. Figure 2.11 shows the search trees computed for these two initial connection radii and the optimal trajectory for each of these trees.



(a) (b)

Figure 2.11: Search trees (top) and optimal trajectories (bottom) for normal RRT$^{\#}$ connection radius for (a) γ_0 = 2000 and (b) γ_0 = 3000

When run with γ_0 = 2000, RRT$^{\#}$ required a total of 18,634,780 leg evaluations (see Section 2.1.3.2) when computing the costs of edges in the motion planning graph. With γ_0 = 3000, RRT$^{\#}$ required 27,750,792 leg evaluations. In comparison, the proposed approach requires 45,675,191 million leg evaluations for this particular instance and objective weights. Figure 2.11 shows that the solutions found by RRT$^{\#}$ without the modified connection radius are noticeably worse than those found using the proposed approach. While the search trees in Figure 2.11 may appear more dense than the one in Figure 2.7a, this is because the search tree in Figure 2.7a has more configurations pruned from it that have higher costs than the goal node. The search trees in Figure 2.11 have gaps where no configurations are present, whereas Figure 2.7a's search tree has a more uniformly spaced distribution of configurations. The effect of this can be seen in the optimal trajectories in Figure 2.11, which spend less time over the uninhabited Patuxent river than the solution depicted in Figure 2.8. Note that the solution and search tree for γ_0 = 2000 is significantly worse than the solution and search tree for γ_0 = 3000, demonstrating that the proposed variable connection radius allows for using a lower initial connection radius than would normally be viable. In order to obtain comparable results to the proposed approach, RRT$^{\#}$ would need to be run with an even larger initial connection radius than γ_0 = 3000, which would require significantly more leg evaluations than the proposed approach.

2.5 Discussion

Figure 8 shows that the risk of the solution found with $(w_t, w_r) = (1; 0)$ can be reduced by over two orders of magnitude by increasing w_r, but this increases the time needed to get to the goal by a factor of over eight. However, compared with the extremely long solution that minimized risk, increasing w_t yields much shorter solutions with slightly higher risk. The trajectories found demonstrated how searching the larger configuration space (instead of merely the two dimensional path along the ground) can yield new maneuvers for mitigating risk, such as the turns and height changes in Figures 2.8, 2.9 and 10.

However, the larger configuration space does still lead to some imperfections in the trajectories found. An example of this can be seen in the section of Figure 2.9 passing over the Patuxent River, the river is uninhabited, but the trajectory taken through it is not time optimal. This is caused by a lack of sampled configurations at appropriate altitudes and headings, causing the trajectory use the nearest configuration possible, which typically requires a change in either altitude or heading. This issue could be addressed by using a larger number of sampled configurations, employing some form of biased sampling, or smoothing the final trajectory using a gradient-based trajectory optimizer.

The search trees in Figures 2.7a and 2.7b, which are appropriately dense near lower cost configurations and sparse around higher-cost configurations, show that the variable connection radius did not reduce solution quality despite the reduction in computational effort it provided. In comparison, the search trees in Figure 2.11 are poorly spaced out, demonstrating the pitfalls of using the normal connection radius for RRT$^{\#}$ in the problem considered. The variable connection radius also reduced the computational cost of RRT$^{\#}$ and allowed for using a lower initial connection radius to further reduce computational cost. Note that using the variable connection radius does not change the computational scalability of RRT$^{\#}$ (see [88] for analysis), since a minimum connection radius is still used.

Sampling configurations from the current optimal solution helped the modified RRT$^{\#}$ algorithm find high-quality solutions when w_t was near 1, such as the trajectory in Figure 2.9. However, floating point error was able to cause the trajectory to deviate from the time optimal trajectory by small amounts. This occurred when using Dubins curves with the minimal turning radius (such as time optimal Dubins curves), as a small shift in a point on these curves could make that point unreachable by the same Dubins curve. Consequently, even when initialized with the minimal time solution as the initial solution, the modified RRT$^{\#}$ was still able to deviate 5-10% from the minimal time solution when (w_t, w_r) = (1; 0). This can be seen in the high densities of configurations present around the start and goal configurations in Figure 2.7b. However, previous experiments which did not sample on the current optimal trajectory typically found solutions over twice this length, so this was still a significant improvement in solution quality relative to the number of iterations which RRT$^{\#}$ was run for. This issue could have been prevented if "SAMPLE ON TRAJECTORY" directly used segments of the original trajectory being split, instead of re-solving the BVPs for the two split segments.

2.6 Concluding Remarks

This chapter presented a risk-based motion planning approach that can be used to minimize the third-party risk associated with fixed-wing UASs that takeoff near inhabited areas. The new risk-based motion planning approach improves over past risk-based motion planning approaches (e.g. [103], [94]), which were limited to planning 2D paths. Additionally, a new approach for computing 3D Dubins curves was presented, which was able to minimize objectives other than time. Several modifications to the RRT$^{\#}$ algorithm were also presented, which made it computationally efficient to use for the risk-based motion planning problem considered. The resulting approach was shown to be capable of planning trajectories which trade-off between the risk they pose to third parties on the ground and the time needed to fly them. Unlike previous works on UAS risk management through path and motion planning, the proposed approach planned trajectories in 3D, which enabled new strategies for mitigating the risk posed by a UAS.

The results indicate that the proposed approach can compute feasible motion plans and optimize a combination of the time and risk metrics. Risk-based motion planning can consider dynamics constraints and factors such the altitude and orientation of the UAS that a two-dimensional graphbased path optimization approach (e.g [2]) cannot. The study described in this chapter did not, however, consider speed or other state variables that could affect the CPD. However, such variables could easily be incorporated in the proposed approach, provided that appropriate dynamics constraints could be defined for them. While this study did not consider obstacles or no fly zones that a UAS would need to avoid, sampling based motion planners such as RRT$^{\#}$ are typically effective at accounting for such obstacles. The proposed approach could thus be easily extended to account for no fly zones and obstacles, however it would likely be difficult to provide it an initial solution in a problem where they were present. However, the proposed approach still works when not provided an initial solution, though it will require more iterations to find solutions of comparable quality to those found by starting with an initial solution.

Using RRT$^{\#}$ with a locally determined connection radius was largely successful, but additional work is needed to improve the approach for sampling configurations on the current best solution. It may be possible to use local optimization to remove loops from low-risk solutions, which would generate new solutions that require less time and have slightly more risk, which would generate a larger set of alternatives to the UAS operators who are planning takeoff trajectories near or in inhabited areas.

The results have also shown that it is possible for UAS to mitigate the risk they pose to third parties by performing maneuvers during a flight. Developing methods for identifying these maneuvers could provide new tools which UAS operators could use in order to mitigate the risk posed by UAS operations in inhabited areas.

This chapter discussed methods for solving risk-based motion planning problems for UAVs without any consideration of uncertainty. The next chapter discusses methods for solving robust optimization problems, a type of optimization problem which considers uncertainty.

Chapter 3: Feasibility Robust Optimization via Scenario Generation and Local Refinement

The work in this chapter is under review at the ASME Journal of Mechanical Design as [107]. Parts of this chapter originally appeared at the 2018 ASME International Design Engineering Technical Conferences as [104].

This chapter presents a new feasibility robust optimization approach that can efficiently solve non-convex robust optimization problems via local search in the design variables space, while maintaining a global search over the uncertain parameters space. This method has two key components. The first component is a new scenario generation method that can both generate scenarios via sampling and refine them with a local optimization method in order to quickly reach a feasibility robust optimal solution. The second component is a new scenario based local robust optimization method, which refines the final solution to ensure that it satisfies the desired constraint tolerance. Computational experiments demonstrate that using the proposed techniques together requires less overall computational effort in some cases than existing robust optimization approaches and that the proposed new method can solve robust optimization problems that cannot be solved with locally robust optimal techniques. While the techniques developed in this chapter are for robust design optimization and not immediately applicable to UAVs, later chapters will make use of these methods to develop approaches for accounting for how uncertainty affects optimal UAV performance.

The new feasibility robust optimization approach presented is based off the framework for sampling based robust optimization presented in [104], with five key differences: (i) the new approach uses a single improved method for scenario generation over the two methods proposed in [104], (ii) the new approach contains a new local robust optimization step for ensuring the feasibility of the final solution found, (iii) the new approach uses a more efficient formulation than Problem 2 that can contain fewer constraints than it, (iv) the new approach avoids solving scenario robust optimization problems twice per iteration as done in [104] and (v) the new approach does not use scenario reduction.

The rest of this chapter is organized as follows. Section 3.1 discusses the formulation for scenario robust optimization used by the proposed new approach. Section 3.2 details the proposed new approach. Section 3.3 demonstrates the new approach on five different examples and compares its performance against existing robust optimization approaches. Section 3.4 summarizes the conclusions of this paper. Appendix C discusses the theoretical computational performance of the proposed new approach relative to existing robust optimization approaches.

3.1 Problem Formulation

Many sampling based approaches, such as [24], [27], [23] and [25], use a reduced form of Eq. (1.2), where scenarios only impose the constraints which they found a design to violate, rather than imposing all constraints containing uncertainty. This reduced scenario robust optimization formulation is given in Problem 3, Eq. (3.1), where $\overline{U}$ is a finite set of scenarios under which the constraints need to be imposed and where $R(u_k)$ is the set of the indices of the constraints $g_i(x,u_k) \le 0$ that should be imposed under scenario $u_k \in \overline{U}$.

Problem 3: Reduced Scenario Robust Optimization (RSRO) (3.1)

$$\begin{aligned}
&\min_x f(x) \\
&s.t. \\
&d_l(x) \le 0, \forall l \in \{1, ..., L\} \\
&g_i(x, u_k) \le 0, \forall u_k \in \overline{U}, \forall i \in R(u_k)
\end{aligned}$$

Problem 3 can consist of fewer constraints than Eq. (1.2) would for the same set of scenarios $\overline{U}$. Thus, the approaches using Problem 3 ([24], [27], [23], [25] and this chapter) should perform better for problems with larger numbers of constraints than those using Eq. (1.2) (such as [104]), since Problem 3 does not need to impose every constraint $g_i(x,u)$ under every scenario $u_k \in \overline{U}$. By using Eq. (3.1) within the robust optimization framework presented in [104] and incorporating several other improvements, an efficient and scalable robust optimization approach can be developed.

Note that Eq. (1.1) contains an infinite number of constraints, unlike Problems 2 and 3, making it impossible to solve as a non-convex optimization problem. All robust optimization approaches that use either Eq. (1.2) or Eq. (3.1) in place of Eq. (1.1) assume that there exists a finite set of scenarios $\overline{U}$ such that the feasible region of Eq. (1.2) or Eq. (3.1) under the scenarios in $\overline{U}$ is the same as the feasible region of Eq. (1.1). When this assumption does not hold, an "infinite" number of scenarios may be necessary to solve a robust optimization problem. This chapter considers problems where this assumption holds.

3.2 Scenario generation with local robust optimization (SGLRO)

The proposed approach, called SGLRO, solves Eq. (1.1) and consists of two components: a scenario generation method (Section 4.1) and a local robust optimization method (Section 4.2). SGLRO starts off by using a sampling based robust optimization approach (see Figure 3.1), using scenario generation in a similar manner to the approach in [104] (subsequently referred to as SGR^{2}O). Each time a scenario is generated, it is added to $\overline{U}$ and R, which are then used to re-solve Eq. (3.1). This process continues for a finite number of iterations, after which SGLRO uses a local robust optimization method to obtain its final solution.

Normally, a sampling based robust optimization approach can return a non-robust optimal solution with very low worst-case constraint violations after being run for a finite number of iterations. However, a solution with very low worst-case constraint violations should be near the boundaries of the feasible region of Eq. (1.1). Thus, locally searching for worst-case scenarios for that solution should yield additional scenarios that could be added to the set $\overline{U}$ in Problems 2 and 3 so that their feasible regions become the same as Eq. (1.1)'s. These additional scenarios will enable Eq. (1.2) or 3 to find the robust optimal solution. From a practical standpoint, this local worst-case search is largely the same as running a robust optimization approach that searches for worst-case scenarios (e.g., [128]).

A simple strategy for mitigating the asymptotic convergence of sampling based methods is thus to use their final solution as the initial conditions for a local "worst-case" based robust optimization approach. SGLRO implements this strategy once it finishes randomly sampling scenarios, by transitioning to a local "worst-case" based robust optimization approach that makes use of both the design and the scenarios generated during random sampling (the “Solve Local Worst Case Robust Optimization using $\overline{U}$” step in Figure 3.1). Thus, SGLRO is able to maintain the same global search over the uncertain parameters as a random sampling based approach to robust optimization, while mitigating the effects of the limitations of asymptotic convergence.

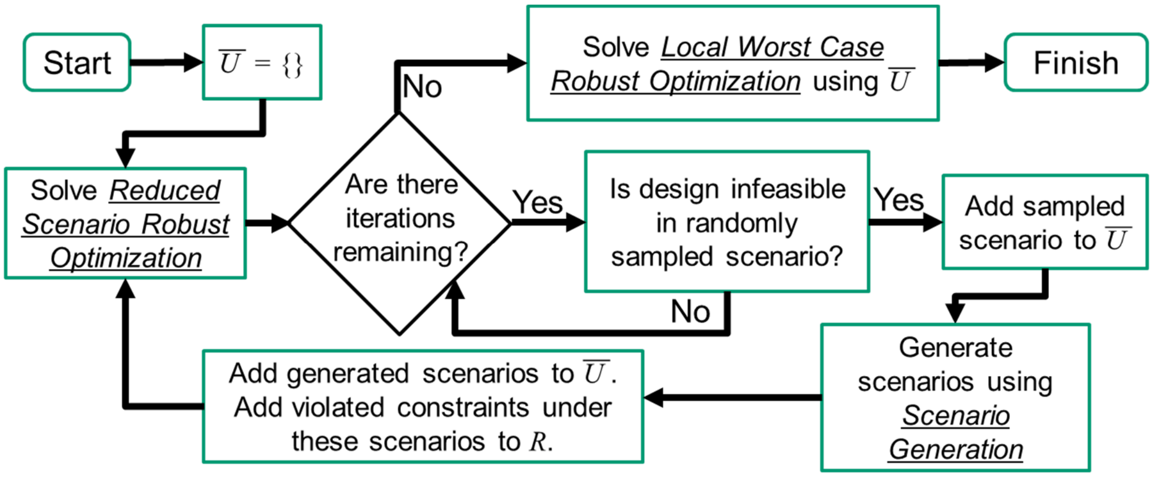

Start
$\overline{U} = \{\}$
Solve *Reduced Scenario Robust Optimization*
Are there iterations remaining?
No
Solve *Local Worst Case Robust Optimization* using $\overline{U}$
Finish
Yes
Is design infeasible in randomly sampled scenario?
Yes
Add sampled scenario to $\overline{U}$
No
Generate scenarios using *Scenario Generation*
Add generated scenarios to $\overline{U}$. Add violated constraints under these scenarios to R.

Figure 3.1: Flowchart Of SGLRO, with Key steps underlined. SGLRO starts by using scenario generation for a fixed number of iterations and then applies a local robust optimization method to obtain its final solution.

An implementation of SGLRO algorithm is shown in Table 3.1. In Table 3.1, the set $\overline{U}$ is a set of scenarios, which is initially empty (cf. line 1). “Solve RSRO” corresponds to solving Problem 3, which should return x_{new}. The function “Sample Possible Scenario” samples a random scenario from $\mathbf{U}$ and returns it. As shown in Table 3.1, SGLRO first solves Problem 3 with no scenarios to generate a candidate design x_B (cf. lines 1-2). Then, it randomly samples scenarios until a scenario is found where the candidate design is infeasible (cf. line 7). It then generates additional scenarios using scenario generation and adds all these scenarios (including the original randomly sampled one) to $\overline{U}$, the current set of scenarios (cf. lines 14-17). SGLRO repeats these steps for a fixed number of iterations (cf. lines 4-5) and then switches to a local robust optimization method (cf. line 19). The number of iterations should be chosen to be sufficiently large such that SGLRO will sample enough scenarios to find the robust optimal solution. When x_B is near (or at) the robust optimal solution after these iterations, the local robust optimization method (cf. line 19) takes care of refining x_B to ensure it is the robust optimal solution. The local robust optimization method is initialized with the set of scenarios $\overline{U}$, it attempts to find new worst-case scenarios which are not in $\overline{U}$ and updates x_B to ensure feasibility in these new worst-case scenarios. When the local robust optimization is done, SGLRO returns its current solution as the robust optimal solution.

Table 3.1: Algorithm 3.1, SGLRO

```
1:  $\overline{U} \leftarrow \{\}$
2:  $x_B \leftarrow$ Solve RSRO
3:  $Reducible \leftarrow True$
4:  While($N_I > 0$)
5:      $N_I \leftarrow N_I - 1$
6:      $Feasible \leftarrow True$
7:      $u_q \leftarrow$ Sample Possible Scenario()
8:      $V \leftarrow \{\}$
9:      For $i \in \{1,...,I\}$
10:         If($g_i(x_B, u_q) > \varepsilon$)
11:             $V \leftarrow V \cup \{i\}$
12:             $Feasible \leftarrow False$
13:     If ($Feasible = False$)
14:         $(\overline{U}, R) \leftarrow$ Scenario
            Generation($x_B, V, u_q, \overline{U}, R$)
15:         $\overline{U} \leftarrow \overline{U} \cup \{u_q\}$
16:         $R(u_q) \leftarrow V$
17:         $x_B \leftarrow$ Solve RSRO
18: $x_B \leftarrow$ Local Robust Optimization($x_B, \overline{U}, R$)
19: Return $x_B$
```

3.2.1 Sampling-Based Scenario Generation

Maximizing constraint violations for a candidate design can be used to find a worst-case scenario, which is more likely to be one of the scenarios in $\overline{U}$ than a randomly sampled scenario. Let *V* be the set of constraints violated by design x_B by a randomly sampled scenario *s* ($i \in V$ if and only if $g_i(x_B, s) \geq \varepsilon$, cf. lines 9-12 in Table 3.). Problem 4 gives the formulation from [104] for finding a new scenario *u* that maximizes the sum of the violated constraints in *V*, where ε is a small positive constraint violation:

Problem 4: Worst-Case Search (3.2)

$$\begin{aligned}
&\underset{u}{\operatorname{argmax}} \sum_{i \in V} g_i(x_B, u) \\
&s.t. \\
&g_i(x_B, u) \geq \varepsilon, \ \forall i \in V \\
&q_j(u) \leq 0, \ \forall j \in \{1, \ldots, J\}
\end{aligned} \tag{3.2}$$

After solving Problem 4, additional constraints may now be violated for x_B, which Problem 4 did not attempt to maximize. Additional scenarios can be generated by solving Problem 4 again for these new violated constraints until solving Problem 4 does not violate any constraint for x_B that has not already been used in the current iteration of scenario generation. Algorithm 2 (Table 3.2) describes this scenario generation process, where "Solve Worst Case Search" refers to solving Problem 4 from initial point *u*. Algorithm 2 works by repeatedly solving Problem 4 for a set of violated constraints (V_{new}) from a given scenario (u_{gen}), and then adding Problem 4's solution as a new scenario (cf. lines 7-10, Table 3.2). The first scenario and set of constraints considered is *V* and u_q (cf. lines 1-3, Table 3.2) which are the randomly sampled scenario from Algorithm 1 (cf. line 14, Table 3.). The next scenario to be considered is the newly generated scenario found by solving Problem 4 (cf. line 7, Table 3.2), with V_{new} being determined from the set of constraints that have yet to be violated by a scenario being generated (cf. lines 11-15, Table 3.2). Algorithm 2 stops when there are no constraints left that are violated (cf. line 5, Table 3.2).

Table 3.2: Algorithm 3.2,
Scenario Generation($x_B, V, u_q, \overline{U}, R$)

```
1:  $V_{check} \leftarrow \{\}$
2:  $u_{gen} \leftarrow u_q$
3:  $V_{new} \leftarrow V$
4:  $c \leftarrow 0$
5:  While($V_{new} \neq \{\}$)
6:      $c \leftarrow c+1$
7:      $u_{gen} \leftarrow$ Solve Worst Case Search
            from $u = u_{gen}$, with $V = V_{new}$
8:      $\overline{U} \leftarrow \overline{U} \cup \{u_{gen}\}$
9:      $R(u_{gen}) \leftarrow V_{new}$
10:     $V_{check} \leftarrow V_{check} \cup V_{new}$
11:     $V_{new} \leftarrow \{\}$
12:     For $\forall i \in \{1...I\}, i \notin V_{check}$
13:         If($g_i(x_B, u_{gen}) \geq \varepsilon$)
14:             $V_{new} \leftarrow V_{new} \cup \{i\}$
```

3.2.2 Scenario-Based Local Robust Optimization

SGLRO uses a simple scenario based local robust optimization method that iteratively performs a local search to find the worst-case scenario for each constraint present. The implementation of the local robust optimization method is given in Table 3.3. In each iteration, the local robust optimization method solves Problem 4 to find the worst-case scenario (cf. line 5, Table 3.3) for each constraint. Any worst-case scenarios that do violate constraints are added to $\overline{U}$ (cf. lines 6-9, Table 3.3). If new scenarios have been added to $\overline{U}$, the scenario robust optimization problem is solved to obtain a new candidate robust optimal solution (c.f. lines 10-11, Table 3.3). This process repeats until no new scenarios are added to $\overline{U}$ (cf. lines 2 and 10, Table 3.3), after which the local robust optimization method stops and returns its current solution as the robust optimal solution.

Table 3.3: Algorithm 3.3,
Local Robust Optimization$(x_B, \overline{U}, R)$

```
1:   $Feasible = False$
2:   While($Feasible = False$)
3:       $Feasible = True$
4:       For($i \in \{1 \ldots I\}$)
5:           $u_{gen} \leftarrow$ Solve Worst Case Search
                 with $V = i$
6:           If ($g_i(x_B, u_{gen}) \geq \varepsilon$)
7:               $\overline{U} \leftarrow \overline{U} \cup \{u_{gen}\}$
8:               $R(u_{gen}) \leftarrow i$
9:               $Feasible = False$
10:      If ($Feasible = False$)
11:          $x_B \leftarrow$ Solve RSRO
12:  Return $x_B$
```

3.3 Examples

SGLRO's performance was compared against a deterministic double loop robust optimization method (see Appendix C) and SGR^{2}O [104] across five different examples of non-convex robust optimization problems. The fifth example problem was a scalable test problem, which was run for increasing numbers of design variables, uncertain parameters and constraints. All examples considered only interval uncertainty.

Because sampling-based robust optimization methods (e.g. SGR^{2}O, SGLRO) are inherently similar to the sampling done in Monte Carlo simulation, Monte Carlo simulation could not be used to verify the robust feasibility of the solutions found. However, in three of the five examples (1, 3, and 4) the set of worst-case scenarios for constraints at the robust optimal solution are known to consist of having uncertain parameters at combinations of their maximum and minimum values. The set of all such scenarios was used to determine the worst-case constraint violations of the approaches compared in these examples. In the remaining two examples, the worst-case constraint violations were determined through alternate analyses (graphically in Example 2, analytically in Example 5).

All examples used the lower bounds for the design variables as the initial conditions for the approaches compared, except where noted otherwise. In all examples, the objective function was treated as being unaffected by uncertainty. In all five examples, SGR^{2}O used $N_S = 12$ scenarios, $N_R = 10$, $N_F = 1$ (number of scenarios sampled per iteration) and $\varepsilon = 10^{-6}$, which is the same as the constraint feasibility tolerance used by the optimization solver. The nominal scenario u_{nom} used by SGLRO was the midpoint of the range for each uncertain parameter. All methods randomly sampled scenarios from a uniform distribution between the lower and upper bounds for each uncertain parameter.

The number of iterations used for each problem was set based on the specific features of the problem. As SGR^{2}O and SGLRO are non-deterministic, they were run 100 times for Examples 1, 2, 3, and 4. Because Example 5 has a single worst-case scenario, SGLRO's performance was deterministic, thus it was run once for each problem size. However, SGR^{2}O was non-deterministic for Example 5, so it was run 10 times for each problem size considered. SGR^{2}O was not run 100 times in Example 5 due to the high computational cost associated with very large problem sizes.

All optimization problems used by SGR^{2}O and SGLRO were solved using MATLAB's fmincon solver with the sequential quadratic programming option [80].

However, the deterministic double loop approach used the interior-point option instead, as it could not find the robust optimal solution in Example 3 when using sequential quadratic programming. When SGR^{2}O solved the scenario reduction refinement problem detailed in [104], fmincon's "OptimalityTolerance" and "StepTolerance" settings were set to 10^{-3}, additionally the "M" parameter from [104] was set to 10^6. When any method compared solved Problem 3, fmincon's "MaxIterations" setting (N_α) was set to 1000 and its "MaxFunctionEvaluations" setting was set to 10^6. All other formulations were solved using fmincon's default parameters. Gradient information was not supplied to fmincon for any of the examples.

3.3.1 Example 1: Basic Circle Problem

Example 1 is an extremely simple non-convex robust optimization problem with a concave objective function and a single convex constraint. The problem is to find the feasible point that is the greatest distance from the origin; a point is feasible if it is inside a circle with a known radius but unknown center. Eq. 3.3 provides the formulation for Example 1.

$$\begin{aligned}
&\min_{x,y} \; -x^2 - y^2 \\
&\text{s.t.} \\
&(x-u_1)^2 + (y-u_2)^2 - 5 \le 0, \forall u_1, u_2 \in [-1,1] \\
&-5 \le x \le 5, -5 \le y \le 5
\end{aligned} \tag{3.3}$$

While Eq. 3.3 is an extremely simple optimization problem, its feasible region requires the constraint $(x+u_1)^2 + (y+u_2)^2 - 5 \le 0$ to be imposed for four different combinations of values for u_1 and u_2 (see Figure 3.2). There are four locally optimal solutions to Eq. 3.3, $(\pm 1, 0)$ and $(0, \pm 1)$, which all share the same globally optimal cost of -1.

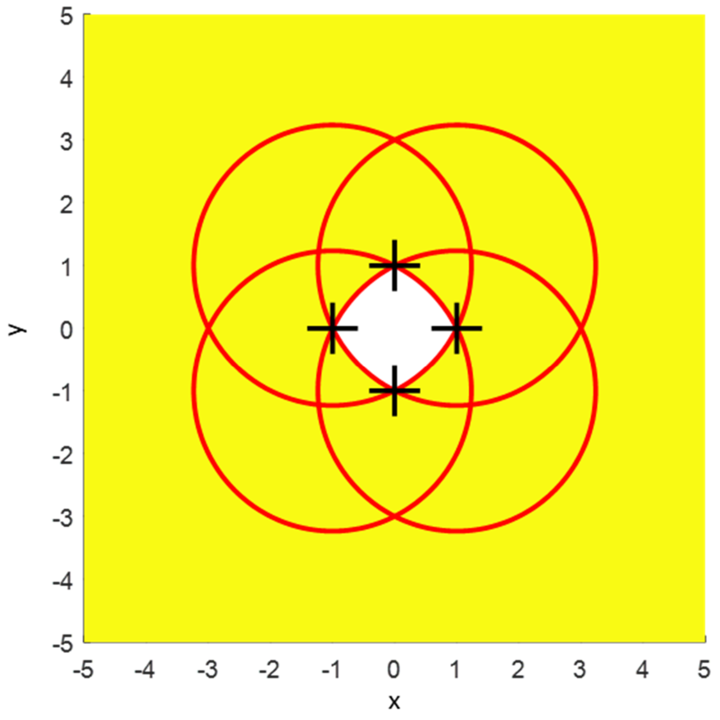


Figure 3.2. Feasible region of example 1, yellow denotes regions which are infeasible. RED denotes the constraint being imposed under different scenarios. The four locally optimal solutions to the problem are Marked with Pluses

All methods compared in Example 1 used $x = 0.5$, $y = 0$ as their initial conditions. SGR^{2}O and SGLRO were run with $N_I = 100$ iterations. Figure 3.3 shows a graphical example of how SGLRO solves Example 1. Note that because SGLRO uses local optimization, it does not need to find all four scenarios which define the feasible region depicted in Figure 3.2.

Figure 3.3: Example Execution of SGLRO on Example 1, Yellow denotes infeasible region, red X denotes infeasible solutions, black X denotes current robust optimal solution. Lines corresponding to Table 3.1 are given for each step in parentheses.

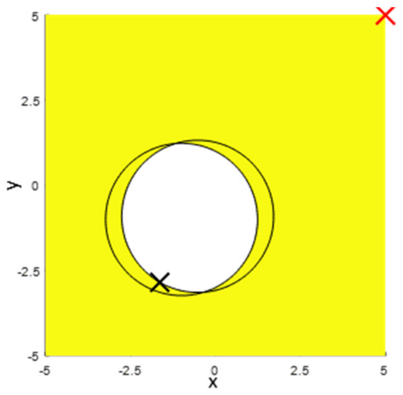
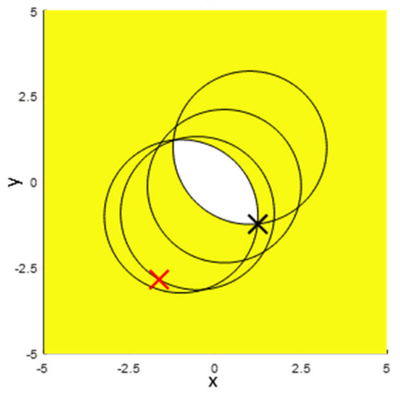
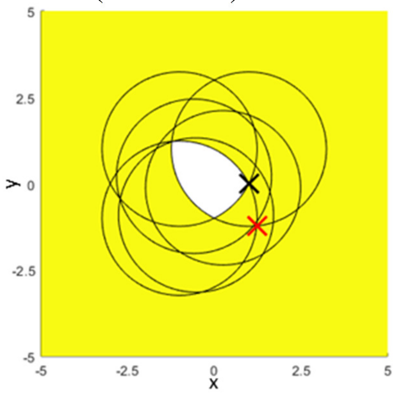
Iteration	Behavior of SGLRO	Feasible Region
0	Eq. (3.1) is solved with $\overline{U}=\{\}$, giving x_B = [5, 5] (lines 1-2)	
1a	Scenario u = [-0.53, -0.91] is sampled. (line 7)	
1b	Design x_B is infeasible under u, $(5+0.53)^2$ + $(5+0.91)^2$ - 5 = 60.5 (lines 8-12)	
1c	Scenario generation occurs, generating the scenario u = [-1, -1] (lines 14-16)	
1d	Eq. (3.1) is re-solved with $\overline{U}$, giving x_B = [-1.64, -2.85]. (line 17)	
2a	Scenario u = [0.26, -0.12] is sampled (line 7)	
2b	Design x_B is infeasible under u, $(-1.64-0.26)^2$ + $(-2.85+0.12)^2$ - 5 = 6.0351 (lines 8-12)	
2c	Scenario generation occurs, generating the scenario u = [1, 1] (lines 14-16)	
2d	Eq. (3.1) is re-solved with $\overline{U}$, giving x_B = [1.23, -1.23]. (line 17)	
3 to15	A scenario u is sampled, but no constraints are violated. (lines 6-12)	
16a	Scenario u = [-0.58, 0.21] is sampled (line 7)	
16b	Design x_B is infeasible under u, $(1.23+0.58)^2$ + $(-1.23-0.21)^2$ - 5 = 0.34 (lines 8-12)	
16c	Scenario generation occurs, generating the scenario u = [-1, 1] (lines 14-17)	
16d	Eq. (3.1) is re-solved with $\overline{U}$, giving x_B = [1, 0]. (line 18)	
17 to 100a	A scenario u is sampled, but no constraints are violated. (lines 6-12)	
100b	Local Robust Optimization is run using $\overline{U}$. No new scenarios are generated on its first iteration, thus x_B = [1, 0] is returned as the robust optimal solution. (lines 18-19)	

Table 3.1 lists the results (mean and standard deviation) of the three methods used to solve Example 1. SGLRO reliably converged to the robust optimal solution of Example 1. SGR^{2}O found the robust optimal solution in 99 of its 100 runs. The one run where SGR^{2}O did not converge was caused by it sampling a scenario (u = [-0.94, 0.86]) that was extremely close to one of the four scenarios defining the feasible region in Figure 3.2. This scenario reduced the probability of sampling a scenario which showed SGR^{2}O's current solution to be infeasible, causing it to run out iterations before finding such a scenario. The deterministic double loop approach did not converge in Example 1, becoming trapped in an infinite loop going between the scenarios shown in Figure 3.2.

Table 3.1: Results for example 1

Approach	Sum of all objective function calls	Sum of all constraint function calls	Largest worst-case constraint violation	Final objective function value	Final number of scenarios
SGR^{2}O (Mean)	137.3	655.4	0.0056	-1.0015	5.97
SGR^{2}O (Standard deviation)	6.338	39.65	0.0557	0.0151	0.30
SGLRO (Mean)	56.9	314.6	0	-1	4.68
SGLRO (Standard deviation)	22.2	115.8	0	$1.6521\times10^{-}$	1.21
Deterministic Double Loop	∞	∞	N/A	N/A	N/A

3.3.2 Example 2: Local Maxima Example

Example 2 (Eq. 3.4) is a very simple problem which contains local maxima with respect to its single uncertain parameter. Unlike Example 1, it does not contain multiple worst-case scenarios, thus it can be used to compare the performance of SGR^{2}O and SGLRO's global searches relative to a local method like the deterministic double loop approach.

$$\begin{aligned}
&\min_{x,y} \; -x \\
&\text{s.t.} \\
&\frac{3}{4}ux\,\mathrm{Cos}\left(\frac{7\pi u(1-x)}{2}\right)^2 - \frac{x}{10} \le 0, \forall u \in [-1,1] \\
&0 \le x \le \frac{8}{10}
\end{aligned} \tag{3.4}$$

There is only one robust feasible solution to Example 2, which is $x = 0$, all other values of *x* are infeasible for at least one value of *u*. SGR^{2}O and SGLRO were run with $N_I = 100$ iterations for Example 2. All approaches in Example 2 used the initial point $x_{IC} = 0.1$. Figure 3.4 shows a plot of the constraint in Example 2 as a function of *x* and *u*, along with the solutions found by the approaches run on Example 2.

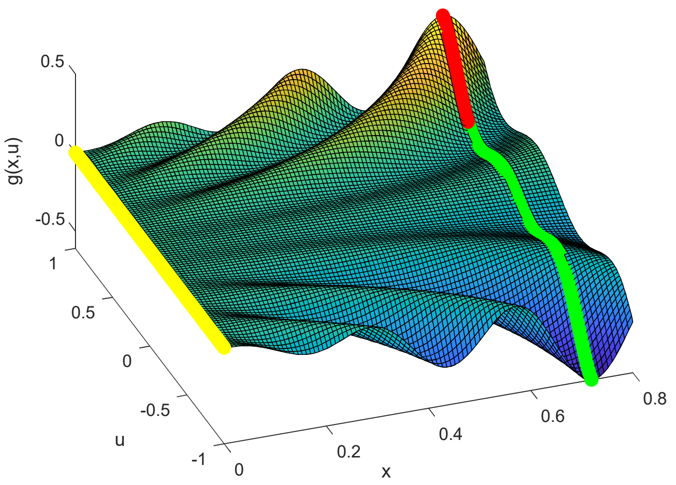


Figure 3.4: Plot of constraint in Example 2. The yellow line highlights the value of the constraint for the robust optimal solution at different uncertain parameter values (which is 0 for all values). The red and green lines denote where the deterministic double loop approach's solution is either feasible (green) or infeasible (red).

Table 3.2 lists the results (mean and standard deviation) of the three approaches compared in Example 2. SGLRO and SGR^{2}O reliably found the robust optimal solution, however the deterministic double loop approach found an infeasible solution (see Figure 3.4). This occurred because the deterministic double loop approach uses a local search to find worst-case scenarios, which caused it to find a scenario that locally maximizes the value of the constraint ($u = 0.215$) instead of the global maximum ($u = 1$). SGLRO was significantly faster than the other two approaches compared in Example 2. The deterministic double loop approach would be fastest if it used sequential quadratic programming as its solver, but it would still find the same infeasible solution shown in Figure 3.4. SGR^{2}O used scenario reduction in 15 of its 100 runs. These 15 runs required many more constraint function calls than the other 85 runs, which caused the large standard deviation in SGR^{2}O's number of constraint function calls.

Table 3.2: Results for example 2

Approach	Sum of all objective function calls	Sum of all constraint function calls	Largest worst-case constraint violation	Final objective function value	Final number of scenarios
SGR^{2}O (Mean)	178	1,865	1.2824×10^{-12}	-2.3544×10^{-12}	6.34
SGR^{2}O (Standard deviation)	214	3,144	8.98711×10^{-12}	1.8123×10^{-11}	2.78
SGLRO (Mean)	14.4	143.5	2.4689×10^{-14}	-4.5330×10^{-14}	2.18
SGLRO (Standard deviation)	4.79	17.24	2.2012×10^{-13}	4.0414×10^{-13}	0.58
Deterministic Double Loop	118	188	0.4659	–0.719	1

3.3.3 Example 3: Robust Welded Beam

Example 3 is a robust optimization variant of the well-known welded beam problem considered by [96], taken from [104] and [85]. The eight uncertain parameters considered were deviations in the values of the problem's four design variables (dimensions of the weld and of the beam) and the length, load and failure stresses of the beam. The objective function is to minimize the cost of the beam without considering uncertainty, accounting for the material cost of the beam and the cost of the weld. Example 1 has six constraints: two require that the beam does not fail under shear and bending stress; the other four limit the deflection of the beam, ensure that the beam does not buckle, require that the weld's thickness is not larger than the beam's width, and limit the weld's thickness. SGR^{2}O and SGLRO were run with $N_I = 100$ iterations.

Table 3.3 lists the results (mean and standard deviation) for all three approaches in Example 3. SGLRO and the deterministic double loop approach reliably converged to the robust optimal solution, but SGR^{2}O found the robust optimal solution in only 99 of its 100 runs. Both SGR^{2}O and SGLRO reached the robust optimal solution after performing scenario generation twice. Note that the number of scenarios sampled by SGR^{2}O was approximately a tenth of those used for this problem in [104], with more iterations all 100 runs of SGR^{2}O would have converged as it did in [104]. The deterministic double loop approach was the fastest approach in Example 3.

Table 3.3: Results for example 3

Approach	Sum of all objective function calls	Sum of all constraint function calls	Largest worst-case constraint violation	Final objective function value	Final number of scenarios
SGR^{2}O (Mean)	536.9	14,509	0.001	2.7859	4.17
SGR^{2}O (Standard deviation)	34.75	1,245.5	0.001	3.9414×10^{-4}	0.40
SGLRO (Mean)	278.2	5,349.4	8.0312×10^{-9}	2.7859	4.2
SGLRO (Standard deviation)	19.35	452.02	3.1742×10^{-8}	1.97×10^{-8}	0.402
Deterministic Double Loop	320	4,679	6.6404×10^{-6}	2.7859	6

3.3.4 Example 4: Enhanced Robust Speed Reducer

Example 4 is a more challenging version of the robust speed reducer design optimization problem first considered in [47], which is detailed in Eq. (3.5). Unlike the formulation considered in other works ([47], [128], [71]), which only considered uncertainty for two design variables, this problem included uncertain deviations for all seven design variables. The new constraint g_{13} constrains the allowable variation of the distance between the two shafts in the speed reducer. This new constraint relaxes constraints g_5 and g_6, which allows a wider range of designs than the original problem did in [47]. The upper and lower bounds for the design variables in the problem have been changed to allow a larger feasible region. The objective function is to minimize the sum of the normal stresses present in the two gears (m_2 and m_3). The volume of the speed reducer (m_1) is constrained by constraint g_{10} . Objective robustness (considered in [128]) is not considered. The initial conditions used are the same as the ones used in [128] ($[x_1, x_2, x_3, x_4, x_5, x_6, x_7]$ = [3.58, 0.71, 18, 8, 8, 3.5, 5.3]). The uncertain deviations of the design variables used in this example were [$u_1, u_2, u_3, u_4, u_5, u_6, u_7$] = [$\Delta x_1, \Delta x_2, \Delta x_3, \Delta x_4, \Delta x_5, \Delta x_6, \Delta x_7$].Unlike the original problem, in Example 4 some constraints have multiple worst-case scenarios, which makes solving the robust optimization problem more challenging. SGR^{2}O and SGLRO were run for $N_I = 100$ iterations. Table 3.4 provides the results (mean and standard deviation) for the approaches tested.

$$\min_x \frac{m_2(x)+m_3(x)}{1000}, \qquad \text{s.t. } g(x+u) \le 0, \forall u \in \mathbf{U} \tag{3.5}$$

Where:

$$m_1(x) = 0.7854 x_1 x_2^2 \left(\frac{10x_3^2}{3} + 14.933x_3 - 43.0934\right) - 1.508x_1(x_6^2 + x_7^2) + 7.477(x_6^3 + x_7^3) + 0.7854(x_4x_6^2 + x_5x_7^2)$$

$$m_2(x) = \frac{\sqrt{\left(\frac{754x_4}{x_2x_3}\right)^2 + 1.69\times10^7}}{0.1x_6^3}$$

$$m_3(x) = \frac{\sqrt{\left(\frac{754x_5}{x_2x_3}\right)^2 + 1.575\times10^7}}{0.1x_7^3}$$

$$g_1(x) = (x_1x_2^2x_3^2)^{-1} - 397.5^{-1}$$
$$g_2(x) = (x_1x_2^2x_3)^{-1} - 27^{-1}$$
$$g_3(x) = x_4^3(x_2x_3x_6^4)^{-1} - 1.93^{-1}$$
$$g_4(x) = x_5^3(x_2x_3x_7^4)^{-1} - 1.93^{-1}$$
$$g_5(x) = -x_4 + x_6 - 0.3$$
$$g_6(x) = -x_5 + x_7 - 0.3$$
$$g_7(x) = 5 - x_1x_2^{-1}$$
$$g_8(x) = x_1x_2^{-1} - 12$$
$$g_9(x) = x_2x_3 - 40$$
$$g_{10}(x) = m_1(x) - 1400$$
$$g_{11}(x) = m_2(x) - 1800$$
$$g_{12}(x) = m_3(x) - 1100$$
$$g_{13}(x) = (x_4 - x_6)^2 + (x_5 - x_7)^2 - 0.3^2$$

$$1 \le x_1 \le 15 \qquad 0.1 \le x_2 \le 1.5 \qquad 8 \le x_3 \le 28$$
$$0.3 \le x_4, x_5 \le 12.3 \qquad 1 \le x_6 \le 8 \qquad 1 \le x_7 \le 8$$
$$-0.01 \le u_1 \le 0.01 \qquad -0.01 \le u_2 \le 0.01 \qquad -1 \le u_3 \le 1$$
$$-0.1 \le u_4, u_5 \le 0.1 \qquad -0.1 \le u_6 \le 0.1 \qquad -0.05 \le u_7 \le 0.05$$

Only SGLRO found the robust optimal solution every time. SGR^{2}O reliably found an infeasible solution that is extremely close to the robust optimal solution but is not robust because small worst-case constraint violations are present in constraints g_{11}, g_{12}, and g_{13}. Note that SGR^{2}O did not perform scenario reduction in Example 4. Like Example 1, Example 4 required multiple worst-case scenarios for one of its constraints (g_{13}), which caused the deterministic double loop approach to enter an infinite loop.

Table 3.4: Results for example 4

Approach	Sum of all objective function calls	Sum of all constraint function calls	Largest worst-case constraint violation	Final objective function value	Final number of scenarios
SGR^{2}O (Mean)	914.3	57,350	0.1668	1.885	9.3
SGR^{2}O (Standard deviation)	95.72	12,500	4.4807×10^{-4}	5.0658×10^{-4}	1.51
SGLRO (Mean)	731.1	11,310	5.3529×10^{-9}	1.886	9.11
SGLRO (Standard deviation)	85.99	1,564	2.822×10^{-8}	3.9391×10^{-6}	1.39
Deterministic Double Loop	∞	∞	N/A	N/A	N/A

3.3.5 Example 5: Robust DTLZ9

Example 5 is a single objective, robust version of the scalable multi-objective DTLZ9 test problem [32], which is given in Eq. (3.6). The objective function is to minimize the sum of the objective functions from the original DTLZ9 problem. Uncertainty is added to the problem by adding an uncertain deviation of ± 0.09 for every design variable and by changing the bounds for each design variable to lie within [0.1, 0.9]. The initial conditions used were the midpoint between the bounds.

$$\begin{aligned}
&\min_{x} \sum_{j\in\{1,\ldots,M\}} f_j \qquad (3.6)\\
&\text{where, } f_j(x) = \sum_{i=\left\lfloor (j-1)\frac{n}{M}\right\rfloor}^{\left\lfloor j\frac{n}{M}\right\rfloor} x_i^{0.1}\\
&\text{s.t.}\\
&g_k(x,u) = f_M^2(x+u) + f_k^2(x+u) - 1 \ge 0,\\
&\forall k \in \{1,\ldots,M-1\}, \forall u \in \mathbf{U}\\
&0.1 \le x_i \le 0.9, -0.09 \le u_i \le 0.09,
\end{aligned}$$

SGR^{2}O [104], SGLRO, and the deterministic double loop approach were run for varying sizes of Example 5, ranging from n = 10 design variables to n = 250 design variables. The parameter M in the DTLZ9 test problem was chosen to be half the number of design variables ($M = n / 2$). Example 5 has a single worst-case scenario, in which every deviation equals -0.09, so the robust optimal solution assigns a value of 0.1 to all but the last two design variables, which instead equal 0.5527. All three approaches found the robust optimal solution to Example 5 for all problem sizes considered. From the computational complexity analysis described in Appendix B, it should be noted that the number of constraints and uncertain parameters in the DTLZ9 problem [32] increases linearly with the number of design variables, so SGR^{2}O [104], SGLRO, and the deterministic double loop approach should all have $O(n^2)$ constraint calls relative to the number of design variables (*n*). Because SGR^{2}O's behavior was not deterministic in Example 5, it was run 10 times for each problem size and the medians of the number of function calls were used for comparison. This non-deterministic behavior was caused by the scenario generation method used by SGR^{2}O, which generates some scenarios by minimizing constraint violations.

As shown in Figure 3.5, the number of objective function calls made by SGR^{2}O, SGLRO, and the deterministic double loop approach increased linearly with problem size, which is expected as the same number of solver iterations is required for most problem sizes, but computing the gradient of the objective function increased linearly in function calls as more design variables were added. SGLRO always found the worst-case on the first iteration, so it needed fewer objective function calls than the deterministic double loop approach.

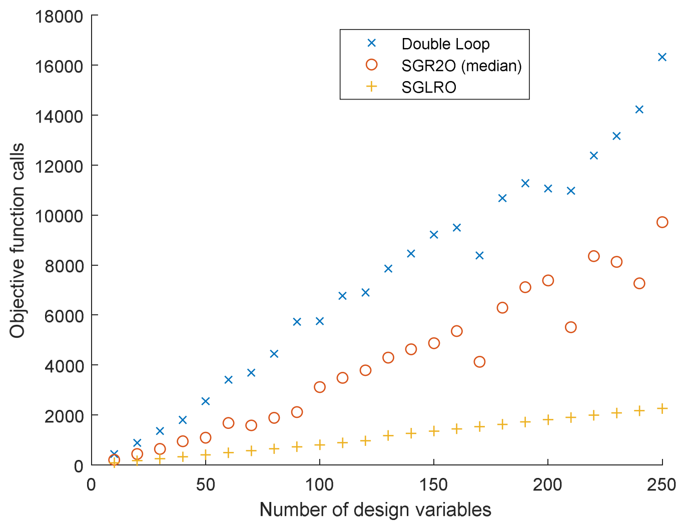


Figure 3.5: Number of objective function calls
relative to problem size for Example 5

As shown in Figure 3.6, the number of constraint function calls that SGR^{2}O, SGLRO, and the deterministic double loop approach made increased quadratically as the size of the problem increased (R^2 value for fitting a quadratic curve is 0.96 for SGR^{2}O, 1 for SGLRO, and 0.99 for the deterministic double loop approach). This result numerically demonstrates that all three approaches have comparable scalability. This relationship also confirms the predicted $O(n^2)$ computation cost and demonstrates the correctness of the computational complexity results presented in Appendix B. SGLRO used fewer constraint function calls than the deterministic double loop approach only when the deterministic double loop approach used MATLAB's interior point solver. When it used sequential quadratic programming as the solver, the deterministic double loop approach required fewer constraint function calls in Example 5 than the other approaches.

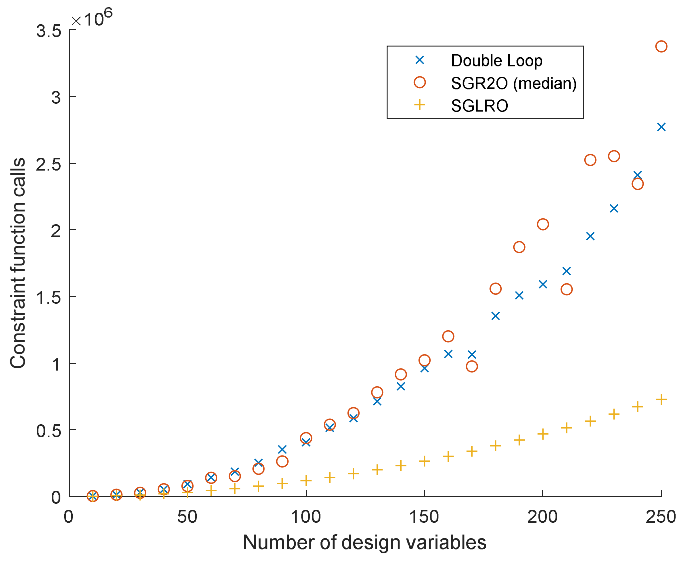


Figure 3.6: Number of constraint function calls
relative to problem size for Example 5

3.3.6 Discussion of Results

For the five examples considered, SGLRO was the only approach that reliably found a robust optimal solution. The deterministic double loop approach found robust optimal solutions for Examples 3 and 5, but it could not do so for Examples 1, 2, and 4. In Example 2, the local maxima present in the constraints prevented the deterministic double loop approach from finding the true worst-case scenarios for the constraints. In Examples 1 and 4, however, the deterministic double loop approach failed because both problems have some constraints with multiple worst-case scenarios. This violates the assumption that each constraint has a single worst-case scenario, which the deterministic double loop approach and other worst-case based approaches to robust optimization (e.g. [128]) require. SGRLO does not require this assumption, which allows it to find all of the scenarios that are needed in order to find the robust optimal solutions to Examples 1 and 4. Additionally, SGRLO uses random sampling when initially searching for worst-case scenarios, which allows it to avoid issues with local maxima such as those present in Example 2.

Although SGR^{2}O almost always found the robust optimal solution in Examples 1 and 3, it reliably failed to find the robust optimal solution in Example 4. The occasional failures in Examples 1 and 3 occurred because SGR^{2}O's number of iterations was set too low. In Example 4, however, increasing the number of iterations would not improve SGR^{2}O's performance. SGR^{2}O failed to find a robust optimal solution in Example 4 because it found an infeasible solution for which the probability of sampling a scenario in which that solution was infeasible was extremely low. Robust optimization methods that rely solely on random sampling, such as SGR^{2}O, are unable to distinguish between this type of solution and a robust optimal solution. SGLRO avoided this problem because its local robust optimization step can easily find a scenario where this solution is infeasible, allowing it to find the robust optimal solution to Example 4. This step also ensured that SGLRO reliably found the robust optimal solution to Examples 1 and 3 using fewer iterations than SGR^{2}O needed.

Curiously, although Example 2 is a very small problem, using scenario reduction actually increased SGR^{2}O's computational cost in Example 2. This occurred because the scenario reduction method proposed in [104] attempts to remove as many scenarios as possible. This causes SGR^{2}O to remove some scenarios that it needs to find the robust optimal solution, so additional function calls are needed to find these scenarios a second time. Thus, SGLRO was faster than SGR^{2}O when finding the robust optimal solution to Example 2.

3.4 Concluding Remarks

This chapter presented SGLRO, a new approach for solving non-convex robust optimization problems. SGLRO extends past work [104] in using sampling based approaches to solve robust optimization problems via a new approach for scenario generation and a new local robust optimization method for refining SGLRO's final solution. The introduction of the local robust optimization method was shown make SGLRO more reliable at finding the robust optimal solution than sampling based approaches and worst-case approaches. It was also demonstrated experimentally that SGLRO was capable of finding the robust optimal solution to several different example problems, even when existing robust optimization methods could not reliably find the robust optimal solution.

The results presented demonstrate that SGLRO can efficiently solve complex nonconvex robust optimization problems with large amounts of uncertainty. However, the results also indicate several areas of potential improvement for SGLRO. SGLRO's performance could potentially be improved by fully integrating the local robust optimization method into the process of sampling scenarios, rather than running it after all scenarios are sampled. A non-uniform scenario sampling approach could make use of existing infeasible scenarios to find new infeasible scenarios more quickly when near the robust optimal solution, speeding up the rate of convergence. Alternate strategies for scenario generation could more quickly obtain useful scenarios, providing a similar benefit. Developing an approach for scenario reduction which avoids the issues observed with the method presented in [104] could also potentially provide an improvement in performance.

All of the approaches discussed require that there exists a finite set $\overline{U}$ that can be used to find the robust optimal solution. It is possible to have a robust optimization problem where Problem 2 requires an infinite number of scenarios (such as a line or other continuous curve of scenarios) in order to reach the robust optimal solution. It may be possible to extend the framework of SGLRO to use robust feasibility cuts, such as in [115], or surrogate modeling based techniques, such as in [130], to handle such problems.

This chapter presented a new approach for robust optimization based off scenario generation and local refinement. The next chapter utilizes the framework for robust optimization developed in this chapter to develop an approach for finding robust optimal solutions to motion planning problems where uncertainty affects motion costs.

Chapter 4: Cost Robust Path and Motion Planning

In this chapter a new approach is presented for robust shortest path planning that is based off the use of a set of scenarios found through random sampling. The new approach is able to handle arbitrary non-linear correlations between edge costs present in a motion planning graph and uncertain parameters. The new approach is thus able to be used for solving the robust shortest path planning problem on motion planning graphs, which enables robust motion planning. A new method for obtaining lower bounds on the worst case costs of paths in the robust shortest path planning problem is also developed. The new method uses a modified variant of the linear programming dual to the proposed scenario based robust shortest path planning problem in order obtain lower bounds on the costs of paths. These lower bounds can be used to minimize the number of edges which need to have their costs computed, which makes it computationally feasible to solve robust motion planning problems. A new method for solving motion planning problems under uncertainty which affects the cost of motions is proposed, making use of the methods developed for robust shortest path planning. The new method uses an existing motion planning graph, generated through methods such as Probabilistic Roadmaps (PRM) [64] or other motion planning techniques which construct a graph (e.g. RRT$^{\#}$ [7]). The new method is demonstrated on a robust version of the risk-based UAV motion-planning problem formulated in [102].

This chapter is organized as follows. Section 4.1 introduces and motivates the problem of cost robust motion planning. Section 4.2 formulates the cost robust motion planning problem using the formulation for the robust shortest path planning problem.

Section 4.3 presents several methods for bounding the costs of paths in robust shortest path planning problems. Section 4.4 presents the proposed algorithm for solving cost robust motion planning problems. Section 4.5 presents an experimental example based off the risk-based motion planning problem from Chapter 2, which has uncertainty added to it. Section 4.6 presents the results from testing the proposed approach on the experimental example. Section 4.7 summarizes the results from this chapter and presents some concluding remarks.

4.1 Introduction to Cost Robust Motion Planning

The performance of unmanned systems in the real world is subject to uncertainty, as it is generally impossible to obtain perfect knowledge of the environment a system will operate in. However, when planning how a system moves, most methods either rely on planning within deterministic environments or across a probability distribution of possible environments [5]. While using a probability distribution of possible environments can be used to account for uncertainty, it also requires a probability distribution to be known in advance. A significant amount of data is needed about an environment in order to determine a probability distribution for uncertainty present in it. Thus, it is often impractical to use probability-based methods for planning under uncertainty in new environments, where an unmanned system is being used in for the first time.

These issues are particularly relevant for UAV systems, as UAVs are subject to various weather conditions such as wind, which impact the performance of a UAV while it is in flight. Weather simulation models (e.g. [52]) can predict possible wind speeds, which can be used when planning routes for a UAV to fly [125]. However, weather forecasts from these models are still imperfect, thus there exists a range of uncertain weather conditions that can occur around their forecasts. Consequently, an approach that does not rely on probability distributions is needed for planning UAV trajectories subject to these uncertain weather conditions.

An alternative to accounting for uncertainty using probability distributions is to use robust optimization [14], where an optimal solution is found for the worst case scenario of uncertain parameters affecting a problem. Such methods have been previously used in the context of motion planning via the concept of funnels [78], for capturing uncertainty in where a robot may end up during motions. These methods only address the problem of feasibility, ensuring that a robot does not collide with obstacles in the environment where uncertainty is present. However, uncertainty also affects the costs of robot motions, particularly in problems where the robot is minimizing an objective other than travel time, such as risk [102]. Considering this type of uncertainty in motion planning creates a "Cost Robust Optimal Motion Planning Problem", the problem of finding a robust optimal solution to a motion planning problem subject to uncertainty affecting the cost of motions. As this problem does not depend on knowledge of a probability distribution, it should be capable of taking into account weather related uncertainty during UAV motion planning.

4.2 Formulation of Robust Motion Planning

Unlike formulations that model cost uncertainty in motion planning using probability distributions, the cost robust motion planning problem formulated here requires a model that provides the cost of moving between two configurations (or vehicle states) under specific uncertain parameter values. This model, denoted as $f(c_1, c_2, u)$, is the cost of moving from configuration c_1 to configuration c_2 under scenario u.

There are two possible definitions for the robust cost of a trajectory when considering a model of the form $f(c_1, c_2, u)$. The first definition (Eq. (4.1)) to treat the robust cost as being the cost under the worst-case scenario for each segment composing trajectory s. The second definition (Eq. (4.2)) treats the robust cost as being the cost under the worst-case scenario for the entire trajectory s.

$$z^1(s) = \sum_{i \in \{1,\ldots,|s|-1\}} \max_{u \in U} f(c_i, c_{i+1}, u) \tag{4.1}$$

$$z^2(p) = \max_{u \in U} \sum_{i \in \{1,\ldots,|p|-1\}} f(c_i, c_{i+1}, u) \tag{4.2}$$

While Eq. (4.1) and Eq. (4.2) may seem similar, Eq. (4.1) is actually significantly more conservative than Eq. (4.2). Consider the scenario involving an unmanned aerial vehicle trying to fly around an obstacle shown in Figure 4.1.

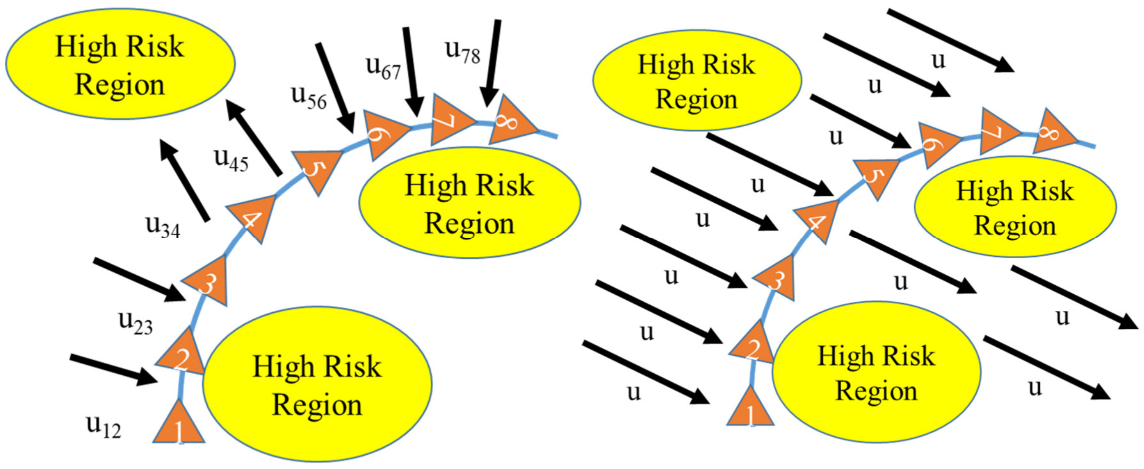


Figure 4.1: Example in which wind blows in an uncertain direction, forcing a trajectory towards a high risk area. The cost is the how well the trajectory avoids the high risk areas.(a) Robust cost model for Eq. (4.1). The worst case direction for the wind changes between each configuration. (b) Robust cost model for Eq. (4.2). The worst case direction for the wind is shared between all configurations.

In the situation depicted in Figure 4.1, Eq. (4.1) would report a higher robust cost than Eq. (4.2) would, as Eq. (4.1) will consider the worst case for each possible segment of the trajectory, which yields a higher cost than considering the worst case for the entire trajectory. While the conservatism built into Eq. (4.1) can be useful in some situations, it is also unrealistic. For example the direction of the wind changes by 180° in Figure 4.1a, whereas an actual wind vector field would vary continuously, as depicted in Figure 4.2.

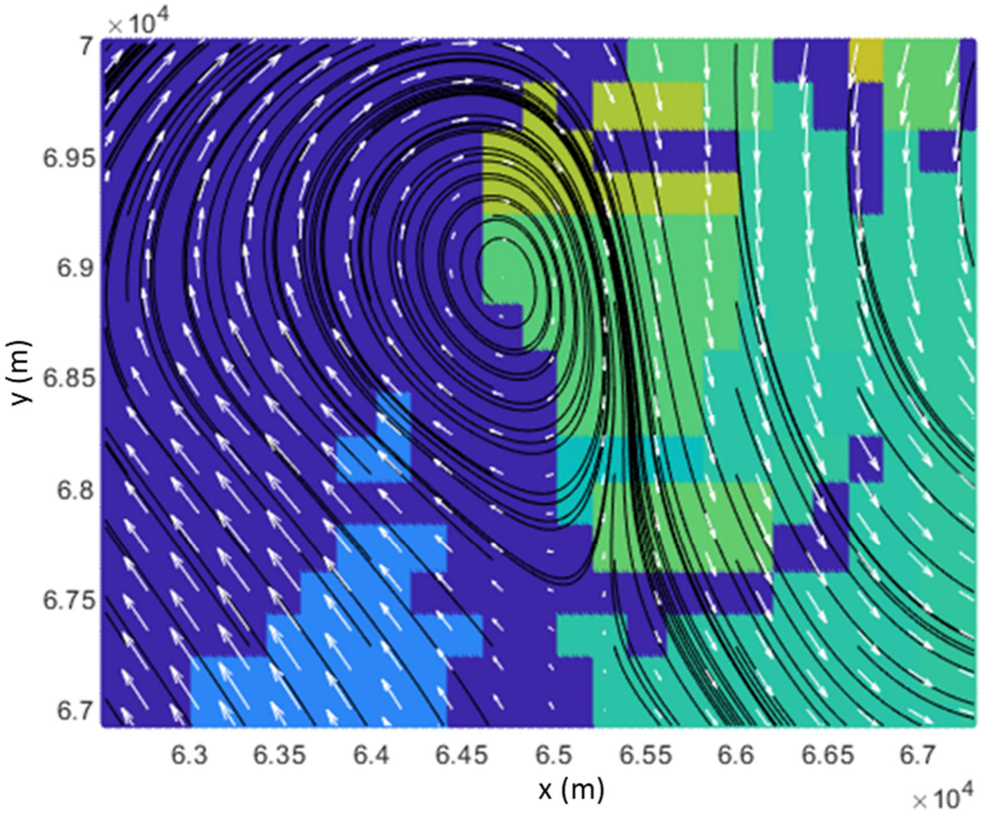


Figure 4.2: Example of a realistic wind vector field, with high risk areas denoted by colors closer to orange and low risk areas denoted by colors closer to blue. The white arrows show the direction of the vector field, while the black lines show streamlines of it. Observe that the vector field varies smoothly, without discontinuous changes in direction.

In Figure 4.2, the high risk regions are small relative to the rate at which the direction and magnitude of the wind vector field can vary spatially. Thus, it would be unrealistic to assume that the wind always blows in the worst possible direction, as such a wind vector field is physically impossible. Because Eq. (4.1) assumes the worst possible case for every trajectory segment, it can end up determining its costs using scenarios (realizations of uncertainty) which are impossible. This will cause Eq. (4.1) to compute higher worst case costs than are actually possible. Eq. (4.2) does not suffer from this issue, since it can use arbitrary model for how uncertainty affects the cost of the entire trajectory. Thus, Eq. (4.2) is more desirable to use for cost robust motion planning, since it can find better performing solutions than Eq. (4.1).

However, Eq. (4.2) is challenging to use as a cost function when solving motion planning problems. Eq. (4.1) (z^1) satisfies the additive relation detailed in Eq. (4.3), meaning that it satisfies the principle of dynamic programing, which is necessary for all existing optimal motion planning techniques (e.g. A* [91], RRT* [63], RRT$^{\#}$ [7] or PRM [64]). However, Eq. (4.2) (z^2) does not, since it is possible that $z^2([c_1,c_2])+z^2([c_2,c_3])>z^2([c_1,c_2,c_3])$. Thus Eq. (4.2) is incompatible with existing methods for both optimal path planning and optimal motion planning.

$$z([c_1,c_2])+z([c_2,c_3])=z([c_1,c_2,c_3]) \qquad (4.3)$$

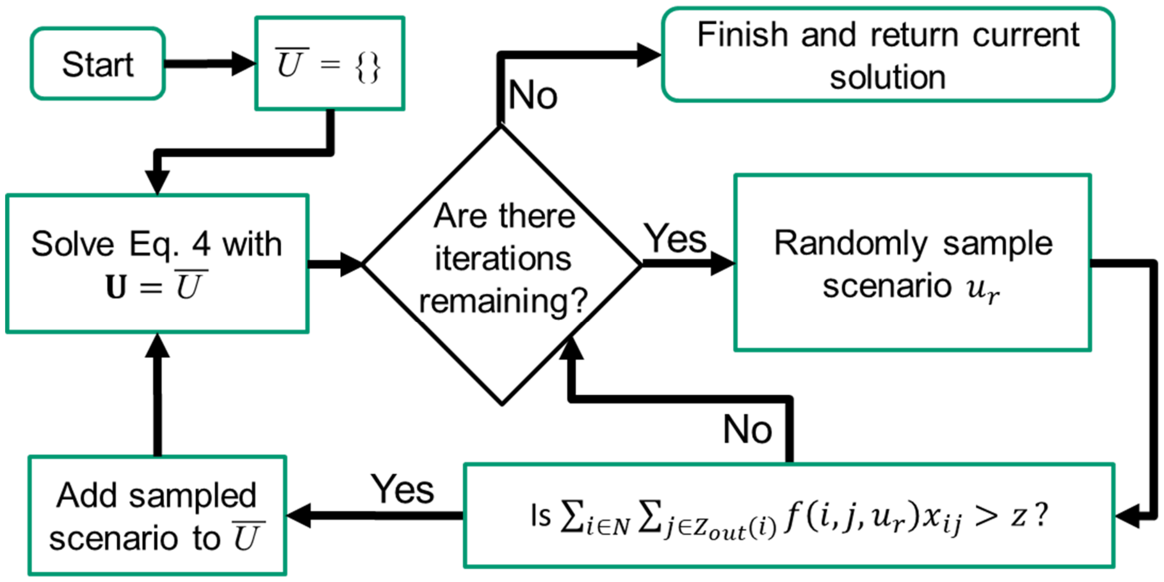
To use Eq. (4.2) as a cost function for motion planning, one must first consider how to use it as a cost function for the shortest path planning problem on a graph. While Eq. (4.2) is incompatible with dynamic programming approaches such as Dijkstra’s algorithm [36] and A* [91], it can still be minimized by formulating a transshipment problem [3]. A robust transshipment problem that minimizes Eq. (4.2) is given below in Eq. (4.4).

1. Let G = (N, E) be a directed graph in which N is the set of configurations (nodes) and E is the set of edges that represent feasible trajectories between these configurations
2. Let A and B be configurations in N, where A is the start configuration, and B is the goal configuration.
3. Let $\mathbf{U}$ be the set of all possible scenarios that affect the cost of all edges in G. $\mathbf{U}$ is a continuous domain when uncertain parameters are defined in a continuous domain.
4. By abuse of notation, let the cost of the edge between configurations $i, j \in N$ under scenario u also be $f(i, j, u)$
5. For any configuration $i \in N$, node *j* is a member of $Z_{out}(i)$, the set of configurations which are incoming neighbors of configuration *i* in graph G = (N, E), if and only if there exists an edge $e_{ij} \in E$, such that e_{ij} starts at configuration *i* and ends at configuration *j*.
6. For any node $i \in N$, node *j* is a member of $Z_{inc}(i)$, the set of configurations which are outgoing neighbors of configuration *i*, if and only if there exists an edge $e_{ji} \in E$, such that e_{ji} starts at configuration *j* and ends at configuration *i*.
7. For all $e_{ij} \in E$, let x_{ij} be a binary integer variable takes a value of 1 when edge e_{ij} is used and 0 otherwise.

$$
\begin{aligned}
&\min_{x,z} z \\
&z \geq \sum_{i \in N} \sum_{j \in Z_{out}(i)} f(i,j,u) x_{ij}, \forall u \in \mathbf{U} \\
&\sum_{j \in Z_{out}(A)} x_{Aj} \geq 1 \\
&\sum_{i \in Z_{inc}(A)} x_{iA} \leq 0 \\
&\sum_{i \in Z_{inc}(B)} x_{iB} \geq 1 \\
&\sum_{j \in Z_{out}(B)} x_{Bj} \leq 0 \\
&\sum_{j \in Z_{inc}(i)} x_{ij} \leq \sum_{j \in Z_{out}(i)} x_{ji}, \forall i \in N, i \neq B \\
&x_{ij} \in \{0,1\}
\end{aligned}
\tag{4.4}
$$

Eq. (4.4) is largely identical to a deterministic transshipment problem, outside of the first constraint in Eq (4.4). The first constraint in Eq (4.4) constrains the optimal cost of Eq. (4.4) to be the maximum possible cost for the current path under a scenario in $\mathbf{U}$. The remaining constraints in Eq. (4.4) are the transport constraints that appear in a typical transshipment problem, which define what a feasible solution is (a connected path from the start to the goal). Thus the remaining constraints in Eq. (4.4) do not contain uncertain parameters in them.

Eq. (4.4) can be used for motion planning as long as a motion planning graph is available, such as the ones constructed by PRM [64] or RRT$^{\#}$ [7]. However, to actually solve Eq. (4.4), a finite set of scenarios needs to be used in place of $\mathbf{U}$ ($\mathbf{U}$ is an infinite set if an uncertain parameter is defined on a continuous domain), so that Eq. (4.4) has a finite number of constraints. In practice this set of scenarios can be generated through randomly sampling scenarios from $\mathbf{U}$, a method which has been successfully applied to a number of robust optimization problems ([24] ,[27] , [23], [25], [105]). Figure 4.3 provides an example of this type of approach, where Eq. (4.4) is re-solved any time a new scenario is sampled in which the cost of Eq. (4.4)'s solution increases.



Start
$\overline{U} = \{\}$
Finish and return current solution
No
Solve Eq. 4 with $\mathbf{U} = \overline{U}$
Are there iterations remaining?
Yes
Randomly sample scenario u_r
No
Add sampled scenario to $\overline{U}$
Yes
Is $\sum_{i\in N}\sum_{j\in Z_{out}(i)} f(i,j,u_r)x_{ij} > z$?

Figure 4.3: Simple example of a sampling based approach being used to solve Eq. (4.4).

While Eq. (4.4) could be directly used on the graph built by a graph based motion planner (e.g. PRM [64], RRT$^{\#}$ [7]) by using a large number of randomly sampled scenarios, there would be a major issue with computationally efficiency. For any scenario $u \in \mathbf{U}$, the function $f(i, j, u)$ needs to be evaluated for every single edge $e_{ij} \in E$. Motion planning problems typically involve high dimensional configuration spaces, meaning the graphs built by PRM and RRT$^{\#}$ typically contain extremely large numbers of edges. This presents an issue when randomly sampling scenarios from $\mathbf{U}$, as each scenario would require computing the cost of the entire motion planning graph. This issue can be partially mitigated by only computing edge costs when a scenario is randomly sampled that yields a new worst case (increases the cost) for the current optimal path. This strategy is demonstrated by the approach in Figure 4.3, which only adds scenarios when this occurs. It can be seen from Figure 4.3 that this strategy only requires computing the cost of the current optimal path under each randomly sampled scenario, which is significantly cheaper than computing the costs of the entire motion planning graph.

4.3 Bounding the Cost of the Solution to the Robust Shortest Path Planning Problem on Large Graphs

A sampling based strategy (e.g. Figure 4.3) can reduce the number of scenarios that need to be considered while solving Eq. (4.4). However, the costs of all edges $e_{ij} \in E$ will still need to be computed for each of those scenarios. Because motion planning graphs can contain a large number of edges, this computational cost will still be very large. However, several methods can be employed to bound the costs of potential solution to the robust shortest path planning problem, which can be used to avoid computing the costs of some of the edges in a motion planning graph.

4.3.1 Duality-based bounds

While Eq. (4.4) cannot determine lower bounds of the cost of including a configuration in the robust optimal path, a lower bound on this cost can be obtained by considering the dual [3] to the LP relaxation of Eq. (4.4). The LP relaxation of Eq. (4.4) drops the constraint that x_{ij} be either 0 or 1, making it a lower bound on the computational cost of Eq. (4.4). The dual to a LP problem shares the same optimal cost as the original LP problem, thus the dual to the LP relaxation of a MILP problem (e.g. Eq. (4.4)) provides a lower bound on the optimal cost of original MILP problem. However, the dual to a LP problem has a different set of design variables than the original LP problem, as the dual will have one design variable (called a dual variable) for each constraint in the original LP problem. The dual to the LP relaxation of Eq. (4.4) is given in Eq. (4.5), where $y_t(i)$ is the dual variable for the transport constraint around node *i* in Eq. (4.4) and $y_s(u)$ is the dual variable to the cost constraint on *z* under scenario *u*.

$$\begin{aligned}
&\max_{y_s, y_t} y_t(B) - y_t(A) \\
&\sum_{u \in U} y_s(u) \leq 1 \\
&y_t(j) - y_t(i) \leq \sum_{u \in U} (f(i,j,u) y_s(u)) \forall i \in N, \forall j \in Z_{out}(i) \\
&y_s \geq 0 \\
&y_t \geq 0
\end{aligned} \tag{4.5}$$

Note that by choosing $y_s(u) = 1$ for one *u* and setting all other $u \in U$ to zero, Eq. (4.5) becomes the dual of the classical (deterministic) unit transshipment problem. In the dual to the deterministic unit transshipment problem, $y_t(i)$ is the cost for the shortest path from the start (supply node) to node *i*. This property will still apply to Eq. (4.5), except that now $y_t(i)$ will be a lower bound on the cost of including node *i* in the robust optimal trajectory. Thus, Eq. (4.5) provides a lower bound on the cost of a path going through node *i*, so if $y_t(i) \geq y_t(B)$, then node *i* cannot be part of the robust optimal trajectory and any edges leaving from node *i* can be ignored.

However, Eq. (4.5) only operates in a forwards direction (all costs are computed relative to the start node), much like a forwards search such as Dijkstra's algorithm or A*. Thus, most nodes will still have costs lower than that of the goal node, meaning they cannot be ruled out. Much like how this issue can be mitigated by using bidirectional search with Dijkstra's algorithm or A*, it is also possible to construct a bidirectional (all costs computed both relative to start node and also relative to goal node) version of Eq. (4.5). Eq. (4.6) formulates a bidirectional form of the dual, where $y_{tf}(i)$ is the forwards direction dual variable for the transport constraint around node *i* and $y_{tb}(i)$ is the backwards direction dual variable for the transport constraint around node *i*.

$$
\begin{aligned}
&\max_{z_d, y_{tf}, y_{tb}, y_s} z_d \\
&\sum_{u \in U} y_s(u) \le 1 \\
&z_d \le (y_{tf}(i) - y_{tf}(A)) + (y_{tb}(i) - y_{tb}(B)), \forall i \in N \\
&y_{tf}(j) - y_{tf}(i) \le \sum_{u \in U} f(i,j,u) y_s(u) \forall i \in N, \forall j \in Z_{out}(i) \\
&y_{tb}(i) - y_{tb}(j) \le \sum_{u \in U} f(i,j,u) y_s(u) \forall i \in N, \forall j \in Z_{out}(i) \\
&y_{tf}, y_{tb}, y_s \ge 0
\end{aligned}
\tag{4.6}
$$

Eq. (4.6) simultaneously computes both the cost going from the start node to each node ($y_{tf}(i)$) and the cost of going from each node to the goal node ($y_{tb}(i)$) using its 3rd and 4th constraints. The second constraint bounds the optimal cost of Eq. (4.6) using the sum of these two costs for each node. Eq, which ensures that $y_{tf}(i)$ and $y_{tb}(i)$ are computed correctly. (4.6) is actually an equivalent formulation to Eq. (4.5), as the constraint on the cost z for the start and the goal configurations constrains z_d to take the same optimal value as in Eq. (4.5). With Eq. (4.6), $y_{tf}(i) + y_{tb}(i)$ is the lower bound on the cost of including node *i* in the optimal path, which provides a much tighter bound than Eq. (4.5) would.

Because Eq. (4.5) and Eq. (4.6) are linear programing (LP) problems, they can be solved much more quickly than Eq. (4.4), which is a MILP problem. However, both Eq. (4.5) and Eq. (4.6) are based off the relaxation of Eq. (4.4), not Eq. (4.4) itself. As additional scenarios are sampled and added to U the cost of the solution to Eq. (4.4) will increase. However, the lower bounds from Eq. (4.5) and Eq. (4.6) do not always increase accordingly. Thus, more nodes will need to be considered as more scenarios are added to U, making Eq. (4.5) and Eq. (4.6) less effective at reducing computational cost as more scenarios are sampled.

4.3.2 Scaling-based bounds

Because Eq. (4.4) is a MILP problem, care must be taken to ensure it is well scaled (no excessively large coefficients present in the A matrix defining the constraints of the MILP problem in the form $Ax \le b$), which must also be done for LP problems such as Eq. (4.6). If this is not done, Eq. (4.4) could fail to produce a feasible solution when solved numerically. When using Eq. (4.4) in a sampling based robust optimization approach, a logical choice for scaling Eq. (4.4) is the new worst case cost found for the current optimal path when a new worst case scenario is sampled. Because this worst-case cost is a feasible solution to Eq. (4.4) when the new scenario is added to U, it is also an upper bound on optimal cost of Eq. (4.4). Thus Eq. (4.4) can be scaled by dividing all edge costs by this worst case cost.

Additionally, because this worst case cost is an upper bound, any edges which can have higher cost than the worst case cost can be ignored. While this may seem obvious, this means that the cost of any such edge can be replaced with the randomly sampled scenario's worst case cost. This allows us to avoid computing the costs of edges which are obviously not part of the robust optimal solution. Additionally, it ensures that Eq. (4.4) does not contain any excessively large edge costs, which could cause issues for MILP solvers if they were present.

4.4 An Algorithm for Solving the Cost Robust Motion Planning Problem Using Sampling Based Robust Optimization

This section presents an algorithm for solving the robust shortest path planning problem using randomly sampled scenarios, which makes use of an already computed motion planning graph G. The proposed algorithm starts with no costs computed under any scenarios for the edges in the motion planning graph G. In addition to the two methods (see Sections 4.3.1 and 4.3.2) for determining which edges can be ignored when solving Eq. (4.4), the proposed algorithms incorporates two additional heuristic strategies, which are described below.

4.4.1 Dual Cost Update Checking Strategy

A heuristic strategy for identifying whether a node's edges need to have their costs computed is to consider whether that node's cost was updated recently in Eq. (4.6) or if that node is part of the current optimal path. This strategy accounts for the fact that Eq. (4.6)'s lower bounds will become less effective (since the duality gap increases) as additional scenarios are sampled. This strategy heavily reduces the number of nodes which need to be considered when computing edge costs. However, it is not guaranteed to identify all the nodes which need to have their edge costs computed in a scenario, which could cause Eq. (4.4) to not yield the robust optimal solution.

4.4.2 Lazy Cost Update Strategy

A second heuristic strategy for reducing the number nodes with edges that need to be recomputed in a scenario is to only update the nodes that are part of the current optimal path. Once this has been done, Eq. (4.4) can be solved under the new randomly sampled scenario. If the current optimal solution does not change, then the new randomly sampled scenario has not changed the optimal solution to Eq. (4.4). If the current optimal solution changes, then this process can be repeated (by updating the costs of the edges going between nodes in the new current optimal path and solving Eq. (4.4) again) until the current optimal solution no longer changes. When the next worst case scenario is sampled, it is practical to update the costs of all nodes that were updated in this manner under the previous worst case scenario. This reduces the number of times that Eq. (4.4) will need to be resolved for a new worst case scenario.

Note that this strategy counteracts the limitation of only using changes in the dual to update costs, since it ensures that the solution to Eq. (4.4) is the robust optimal solution for the current set of scenarios in use.

4.4.3 Proposed Sampling Based Robust Path Planning Algorithm

Algorithm 4.1 (see below in Table 4.1) provides an implementation of the approach proposed, where $m_f(i, j, u)$ is a mapping between an edge $(i, j) \in \mathrm{E}$ and a scenario *u* to the cost of that edge under scenario *u*. Figure 4.4 provides a simplified flowchart for this approach, where the lower corner of each step corresponds to the associated step in Algorithm 4.1.

It can be seen from Figure 4.4 that steps 1, 2, 5, 6, 7, 9, 12 and 15 in Algorithm 4.1 form the same loop as seen in Figure 4.3, which implements a sampling based approach to scenario robust optimization. However, several extra steps are taken to minimize computational cost, by only computing the costs of edges going to and from nodes in the set H. The proposed approach will thus compute the costs of significantly less edges than the approach in Figure 4.3, as it only will compute the costs of edges going to or from nodes in the set H (see Step 12). Thus, the proposed approach is better suited to dealing with very large graphs, where the approach in Figure 4.3 will not be computationally feasible to run. Steps 12 and 18 implement the approach proposed in Section 4.3.1, which uses the dual to Eq. (4.4) to determine which edges can be ignored. Steps 12, 13 and 14 implement the approach proposed in Section 4.3.2, which ignores edges which are too expensive to be part of the robust optimal solution and re-scales their costs accordingly. Note that Step 13 defines m_s so that the values used for scaling are kept separate from the edge costs that have actually been computed (which are tracked in m_f). Steps 12 and 19 implement the heuristic strategy detailed in Section 4.4.1 that updates any nodes with increased cost in the dual. Steps 15, 16 and 17 implement the heuristic strategy detailed in Section 4.4.2 that lazily updates the costs of edges going to and from nodes in the current solution.

Table 4.1: Algorithm 4.1: Proposed Robust Shortest Path Planning Approach

```
1. Initialize $U$ with a nominal scenario $u_n$
2. For all edges $(i,j) \in \mathrm{E}$, $m_f(i,j,u_n) = f(i,j,u_n)$
3. Initialize mapping $D(n)$ to be $D(n) = 0, \forall n \in N$.
4. Set $H_{curent} = \{\}$, $H_{new} = \{\}$.
5. Solve Eq. (4.4) using the set of scenarios $U$ in place of $\mathbf{U}$ and $m_f(i,j,u)$ in
   place of $f(i,j,u)$.
6. If iterations remain, go to step 7 and reduce the number of remaining iterations
   by 1. Otherwise, stop and return the current solution
7. Sample a random scenario $u_w$. If $z < \sum_{i \in N} \sum_{j \in Z_{out}(i)} f(i,j,u_w) x_{ij}$, go to step 8.
   Otherwise, return to step 6.
8. Set $q = \sum_{i \in N} \sum_{j \in Z_{out}(i)} f(i,j,u_w) x_{ij}$.
9. Add $u_w$ to the set $U$.
10. For all edges $(i,j) \in \mathrm{E}$, $m_f(i,j,u_w) = 0$.
11. Set $H_{curent} = H_{new}$, $H_{new} = \{\}$.
12. For all nodes $i \in N$, if $D(i) < q$ and $i \in H_{current}$:
        a. For each scenario $u_s \in U$:
                i. For all nodes $j \in Z_{inc}(i)$, If $\max_{u_{WC} \in U} \mathrm{m}_f(j,i,u_{WC}) < q$:
                        1. If $m_f(j,i,u_s) = 0$ then set $m_f(j,i,u_s) = f(j,i,u_s)$.
                ii. For all nodes $j \in Z_{out}(i)$, If $\max_{u_{WC} \in U} \mathrm{m}_f(i,j,u_{WC}) < q$:
                        1. If $m_f(i,j,u_s) = 0$ then set $m_f(i,j,u_s) = f(i,j,u_s)$.
13. Set $m_s = m_f$.
14. For all edges $(i,j) \in \mathrm{E}$, if $q < \max_{u_{WC} \in U} \sum_{i \in N} m_f(i,j,u_{WC})$, then set
    $m_s(i,j,u_s) = q, \forall u_s \in U$.
15. Set $x_{old} = x$, then re-solve Eq. (4.4) using the set of scenarios $U$ in place of $\mathbf{U}$
    and $m_s(i,j,u)$ in place of $f(i,j,u)$.
16. For all edges $(i,j) \in \mathrm{E}$, if $x_{ij} = 1$ then add node $i$ to both $H_{curent}$ and $H_{new}$.
17. If $x \neq x_{old}$, return to step 12. Otherwise, go to step 18.
18. Solve Eq. (4.6) to obtain the current bidirectional dual solution using the set of
    scenarios $U$ in place of $\mathbf{U}$ and $m_s(i,j,u)$ in place of $f(i,j,u)$.
19. For all nodes $i \in N$:
        a. If $D(i) < y_{tf}(i) + y_{tb}(i)$, add node $i$ to $H_{new}$.
        b. Set $D(i) = y_{tf}(i) + y_{tb}(i)$.
20. Return to step 6.
```

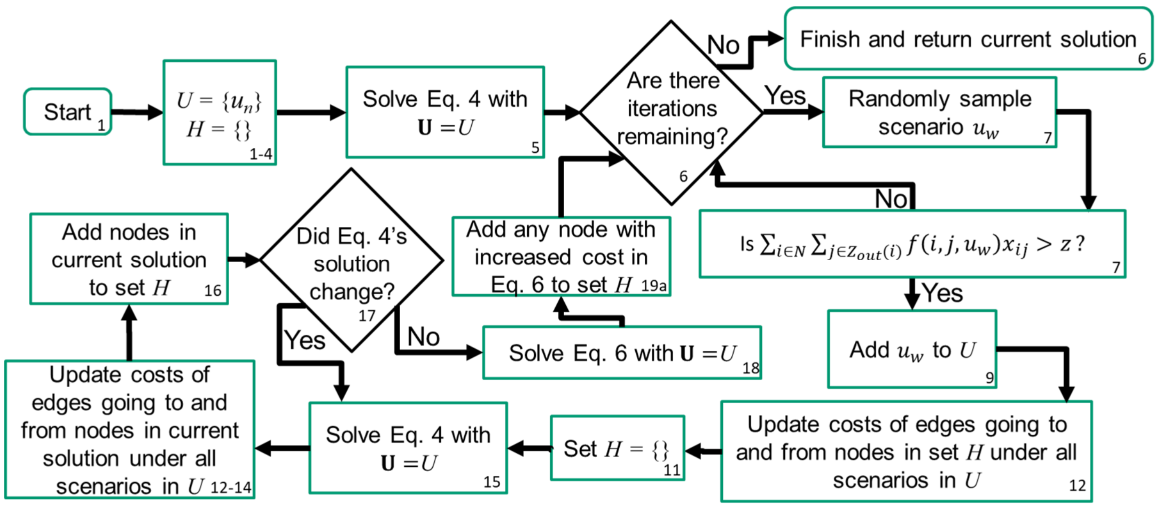

Start 1
$U = \{u_n\}$ $H = \{\}$ 1-4
Solve Eq. 4 with $\mathbf{U} = U$ 5
Are there iterations remaining? 6
No
Finish and return current solution 6
Yes
Randomly sample scenario u_w 7
No
Is $\sum_{i \in N} \sum_{j \in Z_{out}(i)} f(i,j,u_w)x_{ij} > z$? 7
Yes
Add u_w to U 9
Update costs of edges going to and from nodes in set H under all scenarios in U 12
Set $H = \{\}$ 11
Solve Eq. 4 with $\mathbf{U} = U$ 15
Update costs of edges going to and from nodes in current solution under all scenarios in U 12-14
Add nodes in current solution to set H 16
Did Eq. 4's solution change? 17
Yes
No
Solve Eq. 6 with $\mathbf{U} = U$ 18
Add any node with increased cost in Eq. 6 to set H 19a

Figure 4.4: Flowchart of proposed robust path planning algorithm. Numbers in the lower right corner correspond to steps in Algorithm 1.

4.4.4 Cost Robust Motion Planning

While the approach presented above solves a robust path planning problem (on a graph), it can also be applied to motion planning problems by generating a graph using a method such as PRM [64]. It is important to note that Eq. (4.6) is only valid as a lower bound for the graph Eq. (4.6) is solved for, making the proposed approach incompatible with motion planning methods that construct graphs while solving the motion planning problem (e.g. RRT* [63] or RRT$^{\#}$ [7]). Such methods could use a "forward" directional dual, such as Eq. (4.5), however the bounds provided by Eq. (4.5) are much weaker than those provided by Eq. (4.6). Thus, it is more computationally efficient to compute the motion planning graph in advance (using methods such as PRM), so that Eq. (4.6) can be used.

4.5 Experiments

Algorithm 4.1 was tested on a robust version of the risk based motion planning problem proposed in [102], using motion planning graphs generated by the approach used in [102].

The motion planning problem from [102] was solved without an objective function, which generated a motion planning graph which could be used. The motion planning graph used was generated using same motion primitives as in [102], but the time optimal solution was always used as the solution to the BVP problem when finding trajectories between configurations. Additionally, the alternate sampling methods in [102] (sample in ball, sample on optimal trajectory) were not used. However, the variable connection radius proposed in [102] was used with a larger initial connection radius ($\gamma_0 = 3000$) and with $d = 5$. For computational efficiency, a maximum 10 legs were used when computing the costs of any edge in the motion planning graph. Additionally, when each node was first added into the motion planning graph, it was allowed to connect to a maximum of 20 neighbors. However, this limit was not enforced for any nodes already present in the motion planning graph.

4.5.1 Wind Uncertainty Model

The uncertainty considered consisted of an uncertain wind such as the one depicted in Figure 4.3, which was linearly interpolated (using matlab's interp2 function [81]) from the (X,Y) windspeeds (red vectors) at the four nearest points on grid (red circles). The (X,Y) windspeeds were taken from a 20x20 grid of equally spaced points between the bounds of the planning problem used in [102].

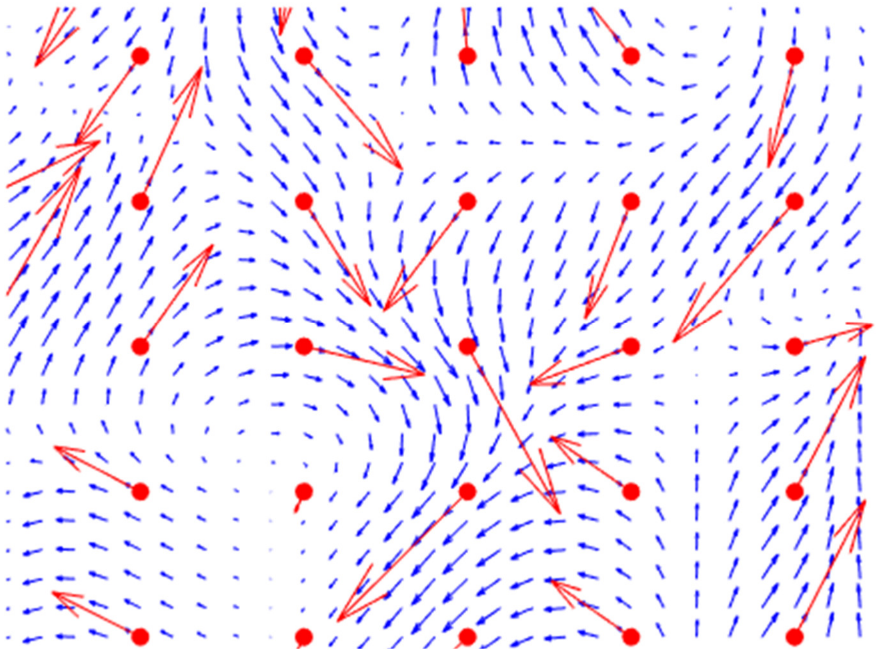


Figure 4.3: Example of randomly sampled wind vectors used in experiments. The wind field is parameterized by the wind vectors (red) at fixed grid points (red), with the wind vectors at points not on the grid (blue) interpolated from the vectors at the nearby grid points

To model the effects of how the wind conditions along a trajectory could affect the risk posed by the trajectory, a model was developed for how different wind conditions can change the crash probability distribution of the UAV. The model uses a proportional linear model (see [109]) to translate the crash distribution of the UAV in the presence of no wind based off how much wind is present based off a slope coefficient for each wind direction. The parameters for the proportional linear model were determined by generating crash distributions for multiple possible wind conditions through the use of monte-carlo simulation approach detailed in [103], using the same procedure as [109]. The slopes used in the linear model were a slope of 0.0728 m per m/s windspeed in the X direction and 0.1207 m per m/s windspeed in the Y direction, the maximum windspeed considered in both the X and Y directions was +-15.1994 m/s (corresponding to 34 mph), which are the same as used in [109]. This model was used in combination with the model for the crash probability distribution from [102], using the current wind vector at each point the crash probability distribution model was evaluated at.

4.5.2 Methods Compared Against

Algorithm 4.1 was compared against the performance of an approach ignoring edge correlations. This approached solved Eq. (4.1) with Djikstra's algorithm, with Matlab's [81] "ga" solver being used to find the worst case scenarios for each edge. Matlab's "ga" solver was run with default settings apart from population size (25) and number of stall iterations (5). It also used only a maximum of 5 legs per edge when assessing risk, in order to further reduce computational cost. Results were also compared against a deterministic approach, which did not consider wind (assumed a windspeed of zero everywhere).

4.5.3 Experiments Conducted

The motion planner from [102] was run 5 times to produce 5 different motion planning graphs, which were used to account for its non-deterministic nature. Because Algorithm 1 is also non-deterministic, it was run under 5 different rng seeds for each of the 5 different motion planning graphs, for a total of 25 different runs. All sampling based approaches were run for 3000 iterations. All MILP problems were solved using Gurobi [48] and all other algorithms considered were implemented in Matlab [81].

When comparing the results from the different approaches, a set of 5000 scenarios was generated using a latin hypercube DOE (DOE), which was used to determine worst case costs. A different DOE was used for each of the 5 motion planning graphs.

4.6 Results

Tables 4.2-4.6 shows all results for each of the 5 different motion planning graphs generated. The optimal cost listed in Tables 4.2-4.6 is the final cost that each approach computes as being its optimal cost, which is the scaled combination of the risk and time objectives. Figures 4.4-4.8 show the cost history of the solutions found by the robust motion planning approach for each motion planning graph.

Table 4.2: Results for motion planning graph #1

Number of Nodes:	2,769		Number of Edges:	33,429
Run	Worst case cost (actual)	Optimal cost (perceived)	Total number of leg evaluations (10^6)	Number of scenarios used
#1	1.11	1.10	3.94	10
#2	1.11	1.10	4.43	12
#3	1.11	1.11	5.03	15
#4	1.11	1.11	2.83	8
#5	1.11	1.11	3.39	10
Worst Case	1.14	1.10	23.9	N/A
Deterministic	1.11	1.04	0.0788	1

Table 4.3: Results for motion planning graph #2

Number of Nodes:	2,755		Number of Edges:	33,237
Run	Worst case cost (actual)	Optimal cost (perceived)	Total number of leg evaluations (10^6)	Number of scenarios used
#1	1.19	1.18	4.73	14
#2	1.19	1.17	5.36	14
#3	1.19	1.18	4.80	15
#4	1.19	1.18	3.91	11
#5	1.19	1.19	5.57	15
Ignoring Edge Correlations	1.51	1.11	25.0	N/A
Deterministic	1.26	1.06	0.0839	1

Table 4.4: Results for motion planning graph #3

Number of Nodes: 2,775			Number of Edges: 33,485	
Run	Worst case cost (actual)	Optimal cost (perceived)	Total number of leg evaluations (10^6)	Number of scenarios used
#1	1.19	1.19	4.35	12
#2	1.18	1.18	3.43	8
#3	1.18	1.18	3.73	9
#4	1.19	1.19	5.04	15
#5	1.19	1.19	2.72	6
Ignoring Edge Correlations	1.18	1.18	24.7	N/A
Deterministic	1.18	1.09	0.0839	1

Table 4.5: Results for motion planning graph #4

Number of Nodes: 2,752			Number of Edges: 32,875	
Run	Worst case cost (actual)	Optimal cost (perceived)	Total number of leg evaluations (10^6)	Number of scenarios used
#1	1.12	1.12	5.51	15
#2	1.12	1.12	4.64	14
#3	1.12	1.13	4.42	14
#4	1.12	1.12	3.31	9
#5	1.12	1.12	4.76	13
Ignoring Edge Correlations	1.12	1.04	23.6	N/A
Deterministic	1.12	0.942	0.0779	1

Table 4.6: Results for motion planning graph #5

Number of Nodes: 2,823			Number of Edges: 34,213	
Run	Worst case cost (actual)	Optimal cost (perceived)	Total number of leg evaluations (10^6)	Number of scenarios used
#1	1.20	1.19	2.84	9
#2	1.20	1.19	4.22	12
#3	1.20	1.20	2.31	9
#4	1.20	1.19	4.02	13
#5	1.20	1.20	2.73	8
Ignoring Edge Correlations	1.45	1.09	25.7	N/A
Deterministic	1.45	1.05	0.0855	1

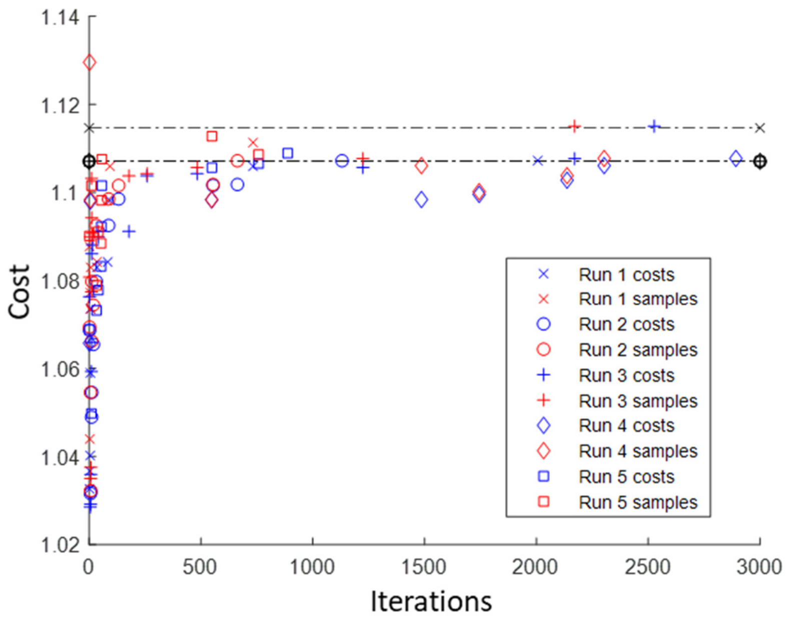


Figure 4.4: Solution history for motion planning graph #1. Red denotes sampled scenarios with new worst case costs, blue denotes the current optimal cost found by the robust motion planning approach. Black lines denote the worst case cost for the final solution found in each run.

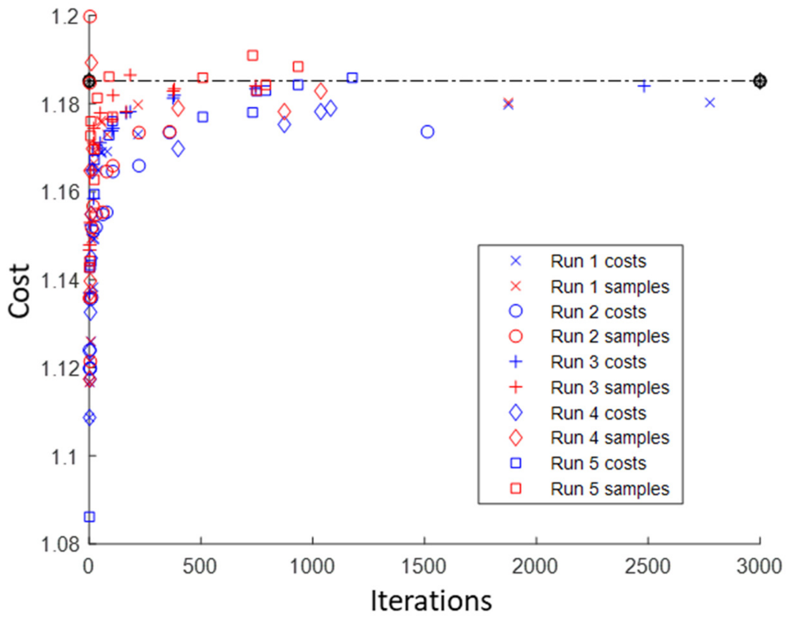


Figure 4.5: Solution history for motion planning graph #2. Red denotes sampled scenarios with new worst case costs, blue denotes the current optimal cost found by the robust motion planning approach. Black lines denote the worst case cost for the final solution found in each run.

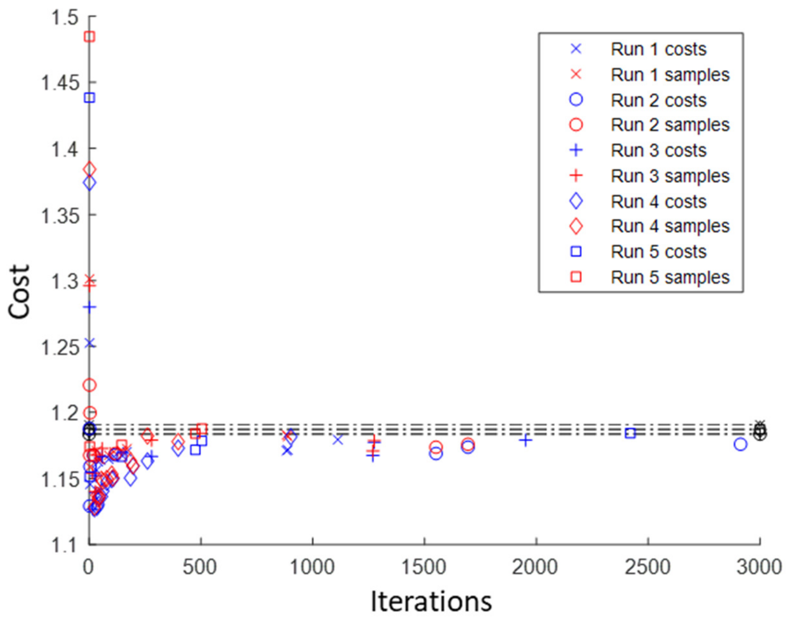


Figure 4.6: Solution history for motion planning graph #3. Red denotes sampled scenarios with new worst case costs, blue denotes the current optimal cost found by the robust motion planning approach. Black lines denote the worst case cost for the final solution found in each run.

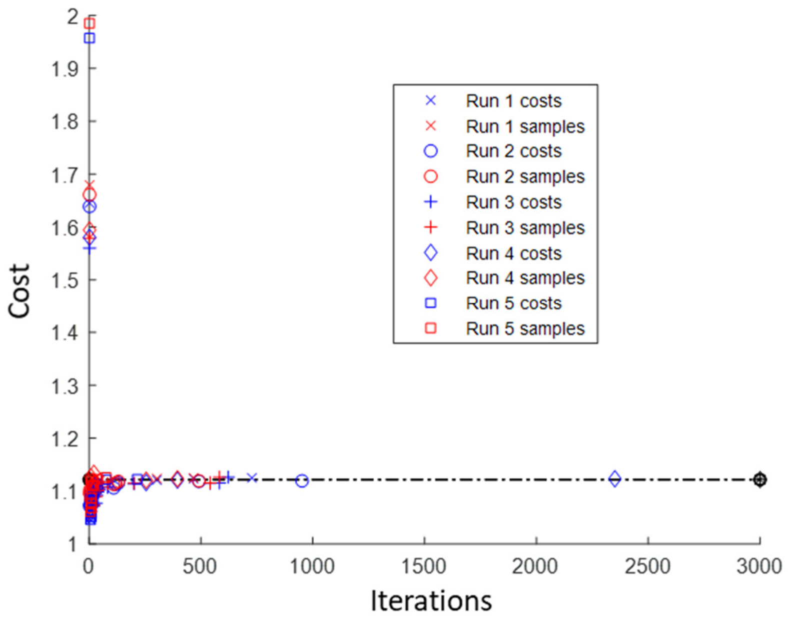


Figure 4.7: Solution history for motion planning graph #4. Red denotes sampled scenarios with new worst case costs, blue denotes the current optimal cost found by the robust motion planning approach. Black lines denote the worst case cost for the final solution found in each run.

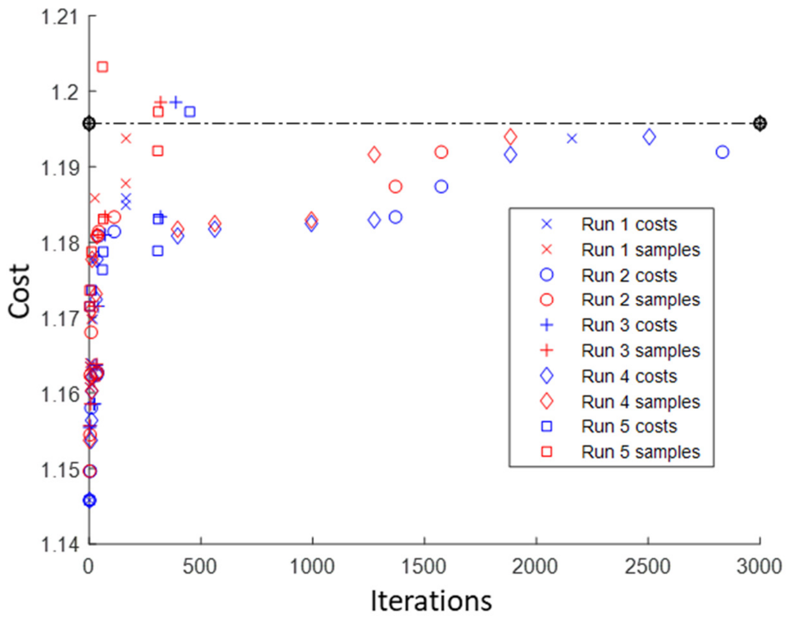


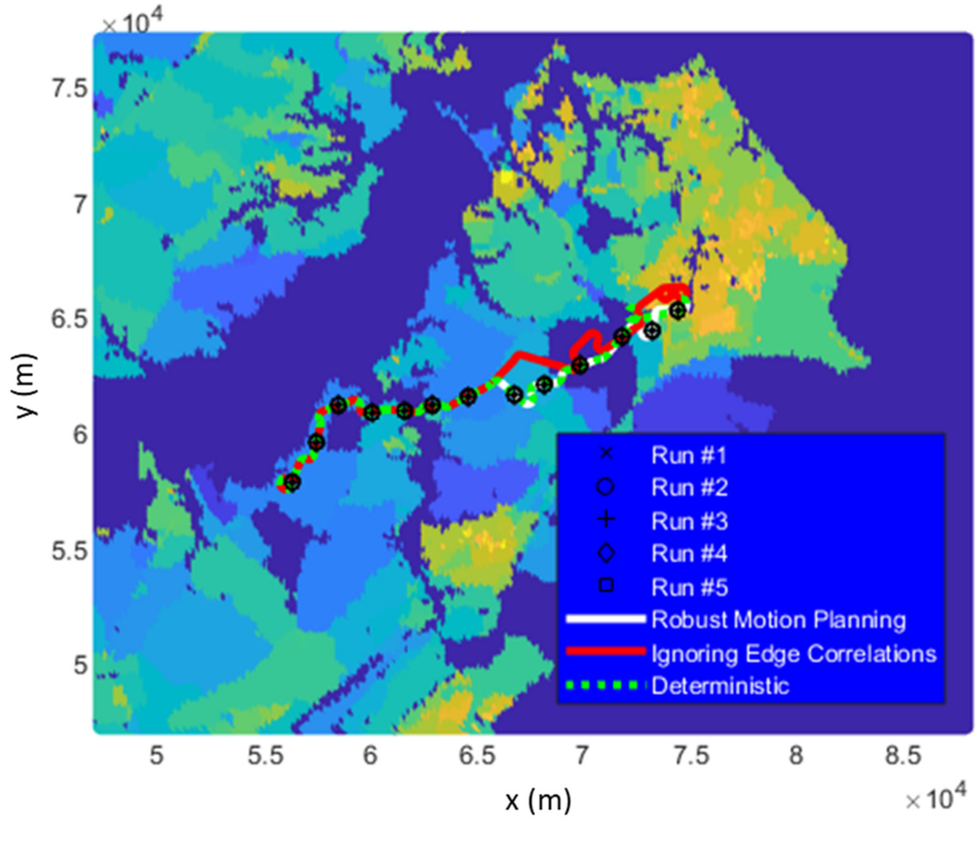
Figure 4.8: Solution history for motion planning graph #5. Red denotes sampled scenarios with new worst case costs, blue denotes the current optimal cost found by the robust motion planning approach. Black lines denote the worst case cost for the final solution found in each run.

As can be seen in Figure 4.4-8, the robust motion planning approach consistently converged to solutions where no new worst case scenarios were likely to be sampled. The gap between the cost in the randomly sampled worst case scenarios (red) and the current robust optimal cost found (blue) converges towards zero as additional scenarios are sampled. Note that in all of the experiments conducted, the largest deviation between the worst case cost found by the robust motion planning approach and the worst case cost assessed using the latin hypercube DOE was 0.02. However, the costs of robust motion planning approach's solutions for the same motion planning graph do vary between runs. This is caused by the robust motion finding different trajectories which have similar costs.

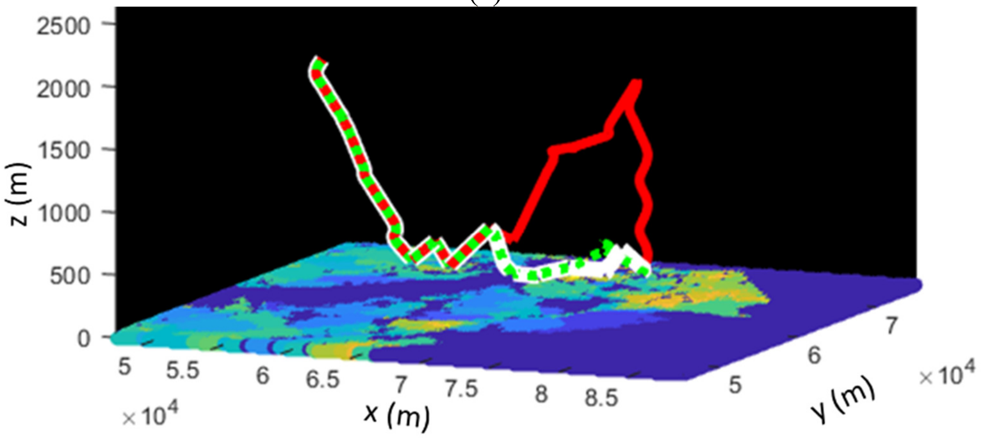
In three of the five motion planning graphs (#1, #3 and #4), the deterministic solution is the robust optimal solution, indicating that there may exists one optimal solution under most scenarios for these three motion planning graphs. However, in the other two motion planning graphs (#2 and #5) there is a significant difference between the worst case costs of the deterministic and robust solutions. Figures 4.9 and 4.10 show the solutions found for these two motion planning graphs.

In both Figure 4.9 and Figure 4.10, the deterministic solution passes near enough to populated areas that the wind can blow the UAV's CPD over those areas. This leads to the deterministic solution having a worse cost than the robust motion planning approaches solutions, which take into account wind. Curiously, the approach that ignored edge correlations also found solutions with poor worst case performance in motion planning graphs #2 and #5. In motion planning graph #2 , the approach that ignored edge correlations chooses to spiral up at the start of its trajectory, which causes it to perform worse than even the determinist solution. This behavior may be partially caused by the smaller number legs used by the approach that ignored edge correlations when assessing risk. However, the deterministic solution shows the same behavior in motion planning graph #5, despite the fact that it used the same number of legs to assess risk as the robust motion planning approach.

In terms of computational cost (number of leg evaluations), the deterministic approach was two orders of magnitude cheaper than the robust motion planning approach. The approach that ignored edge correlations typically required between 5 to 10 times more leg evaluations than the robust motion planning approach. The cost of the robust motion planning approach was roughly proportion to the number of scenarios it used. The number of scenarios used by the robust motion planning approach ranged from as low as 6 to as high as 15.

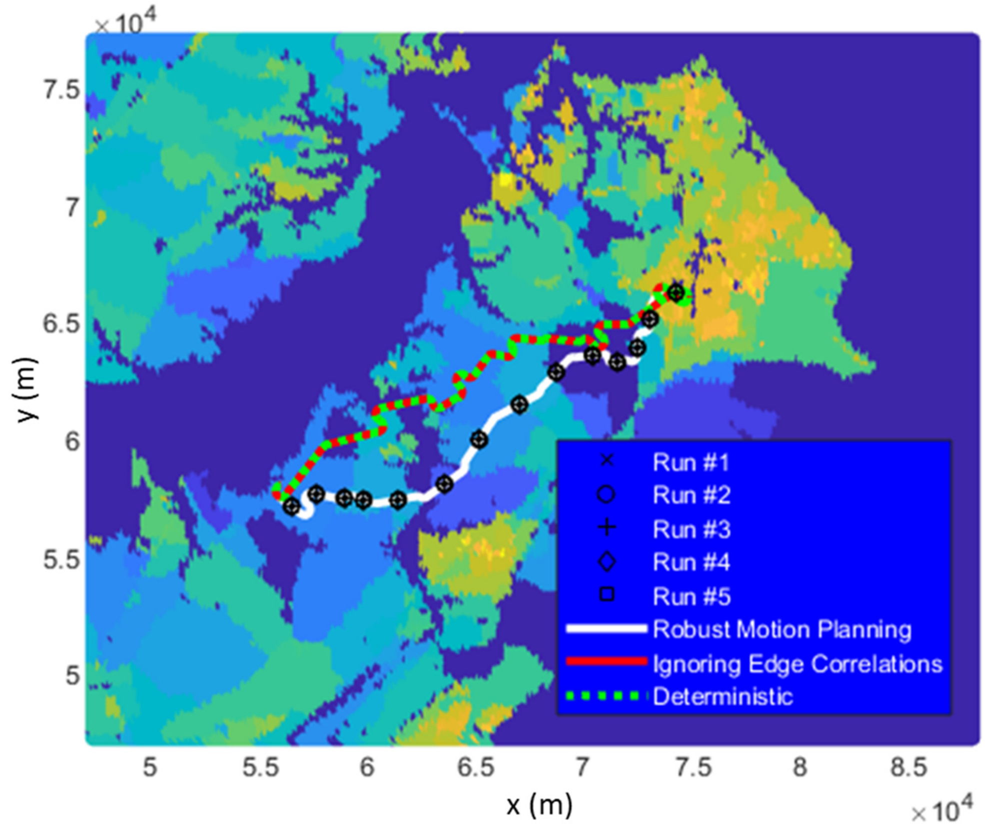


(a)

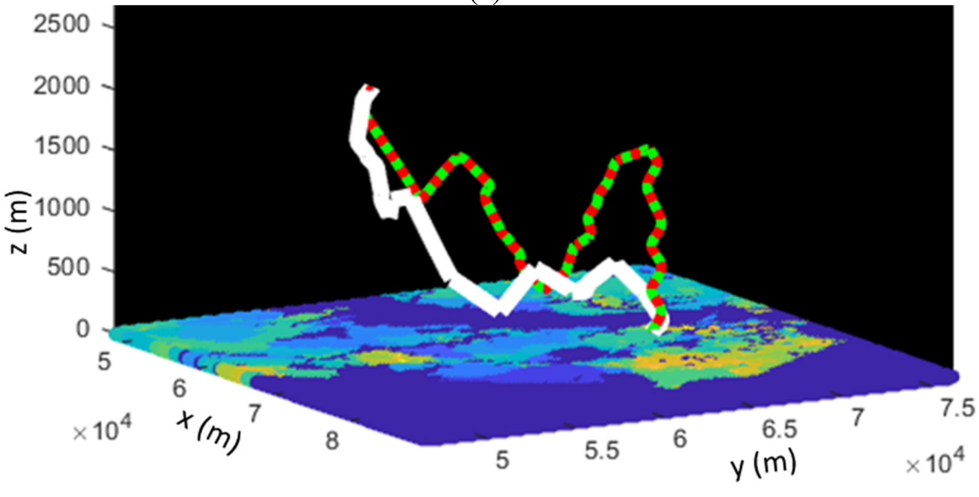


(b)

Figure 4.9: Plot of all optimal trajectories found for motion planning graph #2. (a) 2D view (b) 3-D oblique view (note axes have different scales).



(a)



(b)

Figure 4.10: Plot of all optimal trajectories found for motion planning graph #5. (a) 2D view (b) 3-D oblique view (note axes have different scales).

4.6.1 Discussion

Of the approaches compared, the robust motion planning approach was the only approach that consistently found solutions with costs close to the robust optimal cost. Both the deterministic approach and the approach that ignored edge correlations found solutions with significant deviations in cost when used on motion planning graphs 2 and 5. While this behavior is expected from the deterministic approach (since it ignores uncertainty), it is surprising to also see this behavior from the approach that ignored edge correlations. The large gap in the perceived cost by the approach that ignored edge correlations and its actual worst case cost indicates that it failed to find the true worst case scenarios for several edges in the motion planning graph. Note that the GA was only stopped once it had converged, so this indicates that GA was ineffective at finding worst case scenarios for individual edges. A likely cause for this is that GA's final solution is partly dependent on the quality of its initial population, which by necessity needed to be small so that GA was computationally feasible to run for each edge considered. While GA's initial population is randomly sampled, it is significantly smaller than the total number of scenarios that the robust motion planning approach samples (25 vs. 3000). Thus the purely random search employed by the robust motion planning approach is able to outperform GA at finding worst case scenarios. Notably, the robust motion planning approach also required less leg evaluations than the approach that ignored edge correlations, indicating that a purely random search was also more computationally efficient than using GA.

However, the robust motion planning approaches computational cost was still much higher than that of the deterministic approach, which only accounted for one scenario. Interestingly, the robust motion planning approach was able to use as few as 6 scenarios when finding the robust optimal solution. However, even when using only 6 scenarios (motion planning graph #3, run #5), the number of legs evaluated by the robust motion planning approach is still 32 times higher than that of the deterministic approach. Notably, the deterministic approach did find the robust optimal solution for three of the motion planning graphs generated. This indicates that it would be beneficial to start the robust motion planning approach with a nominal scenario in place of the empty initial scenario set used in the experiments.

4.7 Concluding Remarks

In this chapter, an approach was presented for solving robust motion planning problems. The approach presented was able to consider arbitrary models for how uncertainty affected the costs of edges in a motion planning graph. While such models cannot be used with conventional dynamic programing based motion planning techniques (e.g. PRM, RRT$^{\#}$), the proposed approach bypassed this issue by instead formulating a robust path planning problem using a transshipment formulation. The proposed approach also made use of several methods for bounding solutions costs, which allowed it to reduce the number of times which it needed to compute the costs of edges under different scenarios. The proposed robust motion planning approach was shown experimentally to be able to find solutions with costs near the robust optimal cost.

The proposed robust motion planning approach (which solves Eq (4.2)) was both faster and more effective at finding robust trajectories than the approach that ignored edge correlations (which solves Eq. (4.1)). This indicates that it is possible to develop an efficient random sampling based approach for solving Eq. (4.1) in less computational time than the proposed robust motion planning approach needs to solve Eq. (4.2). Such an approach could be practical for solving robust motion planning problems in real time. The proposed approach could also potentially be used to solve robust motion planning problems in real time, however this would be dependent on using a MILP solver that could be used in real time. State of the art MILP solvers such as Gurobi are typically not designed for use in real time, thus a specialized solver for Eq (4.4) would need to be developed.

The proposed approach's scalability to larger motion planning graphs may also be limited, partially by the computational cost of solving a MILP (exponential worst case computational complexity), but primarily due to the much larger number of nodes (and thus edges) that will be present in the graph. In general, the proposed approach spent significantly more time computing edge costs than it did solving Eq (4.4) or Eq (4.6), indicating that the primary scalability concern for the proposed approach should be the number of edges in the motion planning graph it uses. However, the proposed approach has excellent scalability relative to the number of uncertain parameters present (like any sampling based robust optimization approach), as randomly sampling scenarios is technically unaffected by the number of uncertain parameters being sampled.

While the proposed robust motion planning approach was the faster of the two "robust" approaches compared, its computational cost is still much higher than that of a deterministic motion planning approach that ignores uncertainty. Given that the robust motion planning approach was able to use low number of scenarios on certain motion planning graphs, this indicates that there may be stronger cost bounds possible than the ones discussed in Section 4.3. Additionally, it indicates that it may be beneficial to use some form of scenario generation (as done in Chapter 3), in order to reduce the number of scenarios in which edge costs need to be computed. However, the gradient search based methods used in Chapter 3 performed poorly with the uncertainty model considered in this chapter, being unable to further worsen the cost from the cost in the randomly sampled scenarios used. However, it may be possible to use a heuristic method such as the GA search used in order to do scenario generation, by including the randomly sampled scenario in GA's initial population.

This chapter formulated the cost robust motion planning problem and presented an approach for efficiently solving it. The next chapter formulates an optimization problem for simultaneously optimizing a UAV's design and its path through a graph, and discusses several techniques for solving such problems.

Chapter 5: UAV Design and Path Planning Optimization

This chapter considers a design and path planning optimization problem for unmanned aerial vehicles (UAVs), where the UAV's path planning problem is represented as a shortest path problem on a graph, while its design problem is formulated as a design optimization problem. Such problems are important when optimizing the performance of UAVs, since the shortest path planning problem can be used to represent the UAV's mission planning, while the design optimization problem determines the best possible UAV design configuration for the mission. For example, if a UAV is being operated over inhabited areas, the UAV operator needs to both determine a flight path which minimizes the risk posed to third parties and determine what the most appropriate physical design configuration is for a UAV flying over the inhabited areas in question.

The general form of this problem is the combined optimization problem of the design of a vehicle and determining its path, henceforth referred to as the vehicle design and path planning (VDPP) problem. The path planning part of VDPP problems has a wide range of applications for UAV systems, ranging from finding paths between locations to more abstract planning problems such as mission planning. The goal in all forms of the shortest path problem is to find a path through a graph from a starting node (or start node) to a destination node (or goal node) that yields the lowest total value of a cost function, which is the sum of the costs of the edges in the path. The fastest known approaches for solving shortest path problems using graphs are based on Dijkstra's algorithm [36] and A* search [91]. As shown in this chapter, these methods can be extended and applied to VDPP problems if path costs are determined by solving the vehicle design optimization problem for a fixed path. A* search can quickly solve shortest path problems by using a "cost-to-go" heuristic, which estimates a lower bound cost-to-go from any node in the graph to the goal node in order to bound costs of the nodes that need to be searched. If such a cost-to-go heuristic satisfies the property of admissibility [91], which means the heuristic always provides a lower bound on the true cost to the goal from any node, then A* search will always find an optimal solution to a conventional shortest path problem. This chapter presents an admissible cost-to-go heuristic for the VDPP problem, but it also shows that A* search is not guaranteed to find an optimal solution for VDPP problems, as the edge costs are determined by the vehicle design which must be shared amongst all the edges.

This chapter presents and compares four approaches for solving VDPP problems by constructing the solution using: (1) a proposed cost-to-go heuristic, (2) A*, (3) a new search algorithm, and (4) a branch-and-bound technique. The new search algorithm, called the vehicle design and path planning algorithm (VDPPA), extends the algorithm first developed in [108] to be able to use the proposed cost-to-go heuristic. These approaches were tested on VDPP problem instances in the domain of risk based path planning for UAVs, as discussed in [108] and [103].

This chapter is organized as follows: Section 5.1 formulates the VDPP problem. Section .2 introduces the cost-to-go heuristic for the VDPP problem. Section 5.3 shows via a demonstration example that A* is not guaranteed to find an optimal solution to VDPP problems. Section 5.4 develops a new branch-and-bound algorithm for optimally solving VDPP problems. Section 5.5 presents VDPPA, a new search algorithm which can find more optimal solutions than A*, but without the high cost of an exhaustive search like branch-and-bound. Section 5.6 presents the results of computational experiments used to evaluate the performance of these algorithms. Section 5.7 summarizes the chapter and presents concluding remarks.

5.1 VDPP Problem Formulation

The VDPP problem can be viewed as being a combined formulation of a design optimization problem and a shortest path planning problem. This primarily takes the form of augmenting the formulation of a shortest path problem so that the design optimization problem's formulation is used to determine the cost of each edge in the graph. The goal of the VDPP is to find the lowest cost path between the start and goal nodes, where the cost of a path is a function of design variables (for vehicle design) and path itself, both of which can be altered in order to minimize the total cost function.

5.1.1 Assumptions and Defintions

1. Let G = (N, E) be a directed graph in which N is the set of nodes and E is the set of edges that connects these nodes, such that for any pair of nodes $n_1, n_2 \in N, n_1 \neq n_2$, there is only one edge in E between the nodes n_1 and n_2.
2. Let n_{start} and n_{goal} be two nodes in N, where n_{start} is the start node, and n_{goal} is the goal node.
3. Let path p be an ordered set of m connected edges e_i, i=1,…, m: $[e_1,..., e_i,…, e_m] \in$ E.
4. Let P be the set of all feasible paths from n_{start} to n_{goal}.

5. Let $X \subseteq R^d$ be the feasible domain for the d design variables, such that any design x is only feasible if $x \in X$. For and edge $e_i \in \mathrm{E}$, let $c(e_i, x)$ be the function that describes the cost of the edge e_i with design variables x .
6. Let $f(p,x)$ be the cost function associated with path p and for design variables x, where $f(p,x) = \sum_{e_i \in p} c(e_i, x)$.
7. For any edge $e_i \in \mathrm{E}$ and for any $x \in X$, $c(e_i, x) \geq 0$

5.1.2 Formulation

The general formulation for the VDPP problem is given in Eq. (5.1).

$$
\begin{aligned}
&\min_{x,p} f\left(p,\ x\right) \\
&\text{Subject to:} \\
&p \in P \\
&x \in X
\end{aligned}
\qquad (5.1)
$$

The A* search can be used to solve Eq. (5.1) to determine the path, while the costs of each path A* considers are found by minimizing $f\left(p,\ x\right)$ with respect to x for each path considered.

5.2 Cost-to-go heuristic for VDPP problems

This section presents the cost-to-go heuristic (a lower bound on the cost to get to the goal node from a given node) and an approximation algorithm that uses the cost-to-go heuristic to construct a path that can be used to find a heuristic solution to a VDPP problem.

5.2.1 Cost-to-go Heuristic

The proposed cost-to-go heuristic is based on the idea that associated with each edge e in E is an optimal design $x^*(e)$ in X that minimizes $c(e, x)$, so that $c(e, x^*(e))$ is a lower bound for the cost of edge e in a path containing other edges. The cost of an optimal path from a node to the goal node using these lower bounds as the edge costs is a lower bound on the cost to reach the goal node from that node while using the same design for every edge. Thus, this cost-to-go heuristic satisfies the property of admissibility [91], so it can be used to speed up an A* search without changing the solution that will be found. Note that it is not possible to define a cost-to-go heuristic as a function of a design, while it is possible to evaluate $f(p, x)$ for path that does not reach the goal, $f(p, x)$ cannot be evaluated without specifying values for all of the design variables.

Let $P(n)$ be the set of all possible paths from node n to the goal node, and let $h(n)$ be the cost-to-go for node n, expressed in Eq. (5.2.2), as follows:

$$h(n) = \min_{p \in P(n)} \sum_{e_i \in p} \min_{x \in X} c(e_i, x) \tag{5.2}$$

Determining $h(n)$, the cost-to-go, requires finding a shortest path from node n to the goal node, which can be done using A* (each edge's cost is independent from the other edges). Furthermore, A* can be implemented to determine $h(n)$ using a backwards search [68] from the goal, which can be extended as needed, so that only one A* search is needed to compute $h(n)$ for all the nodes in the graph.

5.2.2 Heuristic Algorithm for Solving VDPP Problems

The cost-to-go heuristic can be used to find a feasible solution to the VDPP using the following two-step heuristic algorithm:

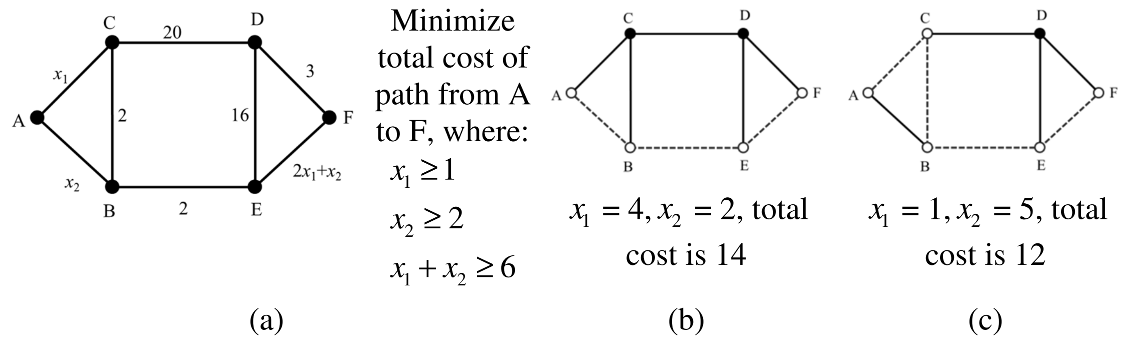
Step 1: Use A* to compute $h(n_{start})$, and let p_H be the corresponding path from n_{start} to the goal node.

Step 2: Find $x_H = \underset{x \in X}{\operatorname{argmin}} f(x, p_H)$.

The solution (x_H, p_H) obtained from the algorithm is a feasible solution to the VDPP problem. Using the cost-to-go heuristic in this manner avoids any additional computational cost that would be incurred if an A* search were to be run using the cost-go-heuristic. However, this heuristic algorithm which only uses the cost-to-go heuristic may find a different solution than an A* search would.

5.3 Suboptimality of A* in VDPP Problems

The fact that the proposed cost-to-go heuristic can also be used to compute a heuristic solution to VDPP problems is important as A* search is not guaranteed to find an optimal solution when used on VDPP problems. To demonstrate this, consider the VDPP example in Figure 5.1(a), where x_1 and x_2 are design variables with the cost for each edge being as given in the graph (Figure 5.1(a)), and the objective function is to minimize the sum of edge costs. The start node is A and the goal node is F. Here, A* will correctly find that the shortest possible path for reaching node B through the edge AB, which leads it to the path in Figure 5.1(b). However, the edge AB is not part of the optimal solution to the VDPP problem, see Figure 5.1(c), even though node B is part of the optimal solution. It can thus be seen that the principle of optimality of dynamic programming [91] does not hold in the context of VDPP, which means that A* and other dynamic programming algorithms are not guaranteed to find an optimal solution to VDPP problems.


A
B
C
D
E
F
20
x_1
x_2
2
16
3
$2x_1+x_2$
2
Minimize total cost of path from A to F, where:
$x_1 \geq 1$
$x_2 \geq 2$
$x_1 + x_2 \geq 6$
$x_1 = 4, x_2 = 2$, total cost is 14
$x_1 = 1, x_2 = 5$, total cost is 12
(a)
(b)
(c)
Figure 5.1: (a) an example of a VDPP with edge costs that are all linear functions of the design variables, (b) the "optimal" solution found via A* search (dashed path), (c) true optimal solution to the problem (dashed path)

It can also be observed that the issue depicted in Figure 5.1 is not something that would be resolved by the proposed cost-to-go heuristic. The heuristic solution that the cost-to-go heuristic would find in Figure 5.1 is the same one that A* search would find with or without using the cost-to-go heuristic. The inherent issue present is that because the principle of dynamic programming [91] does not apply to VDPP problems, the guarantee of finding an optimal solution that it would normally provide to A* no longer holds. Thus, there currently are no known easily usable optimality conditions for identifying optimal solutions to VDPP problems, unlike normal path planning problems, meaning exhaustive search techniques are necessary to guarantee the optimality of the solution found.

5.4 A Branch-and-bound algorithm for Optimally Solving VDPP Problems

Branch-and-bound is an implicit enumeration approach that identifies an optimal solution to an optimization problem. It can be applied to graph search problems, but, for graphs with many nodes, the computational effort required can become enormous due to the huge number of paths that need to be considered to find an optimal path and verify that no better path exists. Better lower bounds can reduce the computational effort. For this study the cost-to-go heuristic was used to generate a feasible initial solution, using the method presented in Section 4. Additionally, the cost-to-go heuristic was used as a lower bound on the cost of the solutions being considered, using the bound detailed in Eq. (5.3):

$$g(p,x)=\begin{pmatrix} h(n)+\sum_{e_i\in p} c(e_i,x) & \text{If } n \text{ is not the goal node} \\ \sum_{e_i\in p} c(e_i,x) & \text{If } n \text{ is the goal node} \end{pmatrix} \tag{5.3}$$

where n is the last node visited by path p.

The most practical way to use branch-and-bound for path planning problems is to directly branch on the paths themselves. This can be achieved by splitting the design optimization problem out of Eq. (5.1) and writing it in terms of an externally determined path with the cost-to-go heuristic being used as the objective function, giving the formulation in Eq. (5.4).

$$F^*(p)=\min_{x\in X} g(x,p) \tag{5.4}$$

Branch-and-bound can use Eq. (5.4) by branching directly on the paths being considered and then evaluating Eq. (5.4) to determine the optimal design for each of the paths. Then

Eq. (5.3) can be used to determine a bound on the optimal cost of a given path. The branch-and-bound algorithm that was used for the VDPP problem consists of the following steps:

Algorithm 5.1: Branch-and-bound for VDPP problems

Step 1: *Compute initial solution.* Compute $h(n_{start})$, the cost-to-go for the starting node, and find (x_H, p_H) using the heuristic algorithm; let this solution be the current best solution, and let $g^* = g(x_H, p_H)$ be the cost of this solution. Create a list of expanded paths T that initially contains only one path (the path that contains only n_{start}); Go to Step 2.

Step 2: *Search.* If T is empty, then return the current best solution. If T is not empty, select the path p in T with the smallest lower bound $F^*(p)$, as determined by Eq. 4, and remove it from T. If $F^*(p) \geq g^*$, then stop and return the current best solution. If the last node in p is the goal node (n_{goal}) and $F^*(p) < g^*$, then make p the selected path the new current best solution, set $g^* = F^*(p)$, and begin Step 2 again. Otherwise, go to Step 3.

Step 3: *Branch.* Let n be the last node in p. For every node r in N such that there exists an edge in E from n to r and r is not in p, add a new path p^+ to T by adding r to the end of p. Return to Step 2.

5.5 Vehicle Design and Path Planning Algorithm (VDPPA)

VDPPA is a forward search extension of the bi-directional search algorithm introduced by Rudnick-Cohen et al. [108] for solving the VDPP problem. As VDPPA is a forward search, it can make use of cost-to-go heuristics to improve its performance. VDPPA is an intermediary approach between using A* search and branch-and-bound for VDPP problems; VDPPA searches through more potentially optimal solutions than A* does, but it avoids the massive set of alternatives that branch-and-bound exhaustively searches. Table 5.1 defines some key terms in explaining the operation of VDPPA.

Table 5.1: Definitions

Start node:	Node representing the point at which the vehicle search starts
Goal node:	Node representing the destination that the vehicle needs to reach
Path:	A sequence of nodes, where each node is connected to the next node in the sequence by an edge
Solution:	A path p and the design variable values $x^*(p)$ that are optimal for that path, with $g(x^*(p), p)$ being the cost of the solution.
Complete Candidate Solution (CCS):	A solution that contains both the start and goal node in its path
Partial Candidate Solution (PCS):	A solution with a path that does not contain the goal node
Subpath:	A sequence of nodes connected by edges within a path
Forward subpath:	The subpath of a path from the start node to a specified node
Backward subpath:	The subpath of a path from a specified node to the goal node
Priority queue [62]:	A queue in which its first element has the lowest cost

5.5.1 Overview

VDPPA extends A* by enabling the search to continue after the goal node has been reached so that multiple candidate solutions can be considered. To do this, VDPPA classifies solutions into two types: a Partial Candidate Solution (PCS) has a path that does not contain the goal node, and a Complete Candidate Solution (CCS) has a path that does contain the goal node. The cost of a PCS should be determined using Eq. (5.3) as it does not contain the goal node, while the cost of a CCS should be determined using Eq. (5.4) as it always ends at the goal node. To accomplish this, VDPPA employs two different algorithms for generating new solutions from existing solutions, one method for PCSs which is largely identical to A* search (see Algorithm 5.2) and one method for building CCSs from existing CCSs (see Algorithm 5.3). These two methods are then integrated with methods for tracking the current best solution for each node to create VDPPA (see Algorithm 5.4).

The following remarks are used to explain how VDPPA works and how it is an extension of A* search.

Remark 1: Best solution for a node - Like A*, in VDPPA, each node stores its current "best" solution (see Remark 1a). In A* search this "best" solution is always a PCS. However, VDPPA searches for the best CCS for each node, thus it attempts to replace the PCSs that A* finds for each node with CCSs which have a path containing that node.

Remark 1a: Until VDPPA finds a CCS that has a path containing a specific node, the best solution for that node is the lowest cost PCS that ends at that node. After a CCS is found, the best solution for any node in that CCS's path is the lowest cost CCS that has a path passing through that node.

Remark 1b: If a node switches its best solution from a PCS to a CCS, any other nodes that have a PCS as their best solution that contain that node in their path should drop their current best solution. This is because the CCS shares part of those PCSs' paths, thus VDPPA needs to find alternate PCS solutions for those nodes with paths that are not a subset of an existing CCS's path.

Remark 1c: A node can have multiple best CCSs with the same cost, so the node must store all such CCSs that are generated.

Remark 1d: VDPPA always assigns only one solution as the best solution for one node at a time. The only exception to this is when a CCS's path contains several nodes, which have yet to receive CCSs, in which case those nodes have their best solution set as the CCS. Like in A*, solutions are assigned in order of their cost, with lower cost solutions (PCS or CCS) being considered first.

Remark 1e: Both PCSs and CCSs must designate a "target" node, which is the node VDPPA should check if the solution is the best for that node. Thus both PCS and CCS can be processed in a best first manner almost identical to A*, by checking if the current lowest cost solution can be assigned as the best solution and moving on to the next lowest cost solution if it cannot.

Remark 2: Generating new CCS - VDPPA also adds a set of procedures for generating new CCSs and PCSs from the solutions stored in neighboring nodes, which allows it to search for more optimal solutions to VDPP problems than A*. Figure 5.2 graphically depicts these procedures, which are described as follows:

Remark 2a: If a neighboring node has a PCS, one new CCS can be created by using the PCS's path to the starting node in place of the original CCSs path to the starting node to create a new CCS. An example of this can be seen in Figure 5.2(a), where the CCS solution (path ABEF) at node B is combined with the PCS (path AC) stored in the neighboring node C, to create a new CCS with the path ACBEF. The procedure for generating new solutions from an existing PCS is as follows:

- Let s be the existing PCS solution from which new PCS solutions should be generated
- Let $e(n_1, n_2)$ be the edge going from node n_1 to node n_2.
- Let $Z(n)$ be the set of neighboring nodes of node n,

  $Z(n) = \{w \mid w \in \mathrm{N} \text{ and } e(n,w) \in \mathrm{E}\}$

- Let $t(s)$ be the node that solution s "targets", which is the node the solution s may be optimal for; $S(t(s))$ is the current best known solution for node $t(s)$ that solution s aims to replace.
- Let $path(s)$ be the path for solution s.

Algorithm 5.2: *Generate solutions from PCS*

For each $n \in Z(t(s))$ where $S(n)$ is empty, generate a new solution s_n as follows:

- If $n \neq n_{goal}$, make a new PCS s_n with $t(s_n) = n, path(s_n) = [path(s), e(t(s), n)]$
- If $n = n_{goal}$, make a new CCS s_n with $t(s_n) = n_{goal}, path(s_n) = [path(s), e(t(s), n_{goal})]$

Remark 2b: If a neighboring node has a CCS, two new CCSs can be created as both nodes have a solution with differing paths to the start node and goal node. One CCS can be created by treating the neighboring CCS as a PCS going up to the neighboring node. This can be seen in Figure 5.2(b), where the CCS (path ABEF) at node B is combined with the path up to node C of the CCS (path ACDF) at the neighboring node C, creating a new CCS with the path ACBEF. The second new CCS can be created by flipping this process, treating the initial CCS as the PCS and the neighboring node's CCS as the initial CCS. This can be seen in Figure 5.2(b), where the path of the CCS (path ABEF) at node B up to node B is combined with the CCS (path ACDF) at the neighboring node C, creating a new CCS with the path ABCDF.

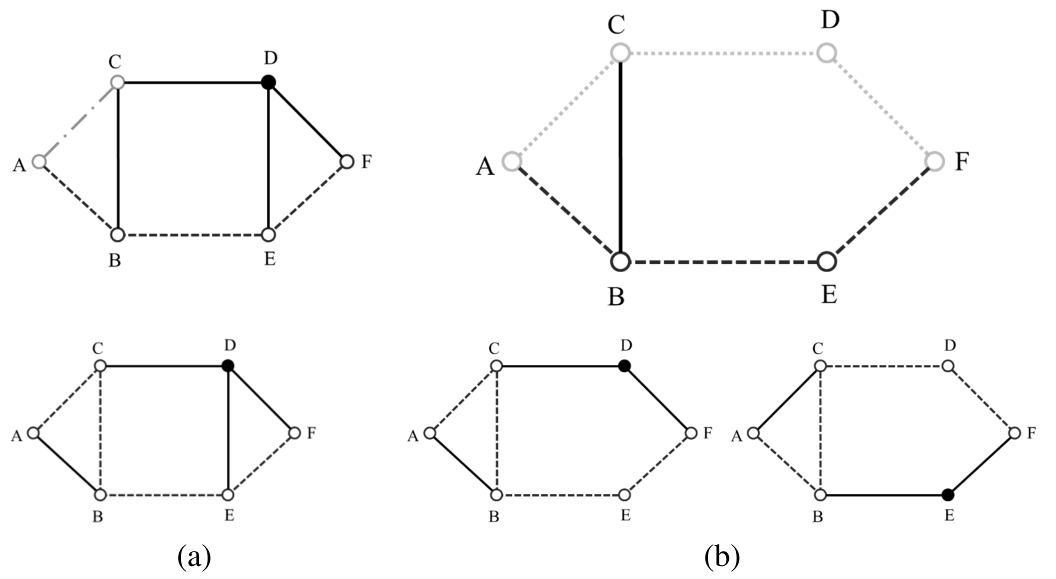


Figure 5.2: Examples of generating new CCSs (lower figures) from neighboring solutions (upper figures) to a CCS (dashed) at node B: (a) generating a new CCS from a neighboring PCS (dash-dot) at node C, (b) generating two new CCSs from a neighboring CCS (dotted) at node C

The procedure for generating new solutions from an existing CCS is as follows:

- In addition to the definitions used for Algorithm 5.2 in Remark 2b:
- Let s be the existing CCS solution from which new CCS solutions should be generated
- Let $S(n)$ be the current best known solution for node n.
- Let $path_f(s,n)$ be the subpath of $path(s)$ from the start node up to node n.
- Let $path_b(s,n)$ be the subpath of $path(s)$ from node n to the goal node.

Algorithm 5.3: *Generate new solutions from CCS*

For each $n \in Z(t(s))$, proceed as follows depending on what $S(n)$ is:

- If $S(n)$ is a PCS, create a CCS s_{bn} with

  $t(s_{bn}) = n, path(s_{bn}) = [path(S(n)), path_b(s,n)]$ and add s_{bn} to Q.

- If *S(n)* is a CCS, For each $s_n \in \{S(n), ties(n)\}$, create the two new CCSs as follows:

    1. Create a new CCS s_{fn}, with

       $t(s_{fn}) = n,\ path(s_{fn}) = [path_f(s,n), path_b(s_n,n)]$ if

       $path(s_{fn}) \neq path(S(t(s_{fn})))$.

    2. Create a new CCS s_{bn}, with

       $t(s_{bn}) = n,\ path(s_{bn}) = [path_f(s_n,n), path_b(s,n)]$ if

       $path(s_{bn}) \neq path(S(t(s_{bn})))$.

Remark 3: Removing dependent PCSs

When VDPPA finds a CCS and assigns it to a node that has a PCS, there may be other nodes which have PCS that build off the PCS getting replaced by the CCS. Some of these PCSs can be said to be PCSs that are “dependent” on the PCS being replaced, they cannot lead to lower cost solutions than the CCS that is being assigned to a node. Thus these “dependent” PCS are unable to be part of the optimal solutions for their “target” nodes. Consequently, whenever a CCS replaces a PCS as the “best” solution for a node, VDPPA removes all PCSs that build off the PCS being replaced from being their target nodes’ “best” solution. This strategy will remove all “dependent” PCS, however it will also remove some PCS that could still be part of the optimal solution. VDPPA handles this by first removing all dependent PCS whenever it adds a CCS and then creating new PCSs by branching off the CCS’s path to the neighboring nodes to the CCS’s path that had their “best” solution removed. This allows VDPPA to get rid of the “dependent” PCSs, while still being able to reconstruct any of the removed PCSs that may still lead to the optimal solution if they are needed.

Additionally, when VDPPA finds a CCS and assigns it to a node that has a PCS, VDPPA replaces all the nodes in the path of that PCS that currently have a PCS as their "best" solution, as the CCS will usually end up becoming the solution for these nodes anyway after further iterations. Thus, the VDPPA gets rid of "dependent" PCSs for all these nodes. For this reason, the start node is treated as being a node that has a CCS for the purposes of keeping PCSs, so that all PCSs are not deleted when this first occurs. Note that this does not change the CCSs that would be generated from the PCSs being deleted, as the PCSs are already the forwards subpaths of the new CCS. This procedure also achieves the goal of Remark 1b, after the "dependent" PCS's are removed, VDPPA will search for alternative PCSs that could have lower costs than the removed PCSs.

Remark 4: Stopping condition - As VDPPA continues to generate CCS solutions, it will reach a point at which the best solution for every reachable node will be a CCS. Once this occurs VDPPA will be unable to find a new best solution for any node. Thus, VDPPA's stopping condition is when it no longer can generate new solutions (PCS or CCS), at which point it must stop and report the lowest cost CCS found.

Remark 5: Speeding up reaching the stopping condition - To accelerate the process of reaching VDPPA's stopping condition without compromising solution quality, it is preferable to identify PCS solutions that cannot lead to the creation of optimal CCS solutions. Like the proposed branch-and-bound approach, VDPPA does this by ignoring PCS solutions for which Eq. (5.3) yields a cost higher than the current best solution. Thus any PCS solution which has a cost determined via Eq. (5.3) to be higher than the cost of the best CCS solution seen so far can be ignored, allowing VDPPA to avoid searching any further paths built off that PCS solution.

5.5.2 VDPPA steps

In this section, a step-by-step description of VDPPA is provided. The following notation is used to describe the VDPPA steps:

- Recall that n_{start} is the start node from which the path starts and that n_{goal} is the goal node where the path should end.
- In addition to the definitions from Algorithms B and C in Remarks 2a and 2b:
- Let $ties(n)$ be the set of CCSs found with the same cost as the current best known solution $S(n)$ for node *n*.
- Let $F(s) = F^{*}(path(s))$ be the cost of solution *s*, defined by either Eq. (5.2) for PCS and CCS or Eq. (5.3) for OPCS, note that computing $F(s)$ means solving the optimization problems in Eq. (5.2) and Eq. (5.3) to find an optimal design for solution s
- Let R be the best CCS obtained so far during the search.
- Let C_R be the cost of R, the best CCS seen so far

Algorithm 5.4: *VDPPA*

Step 1: *Compute initial heuristic solution.* Compute the cost-to-go heuristic for the start node n_{start}. Let the path found by the cost-to-go heuristic for reaching the start node be p_{start}. Create a new CCS s_{init} with $path(s_{init}) = p_{start}$. Set R = s_{init} and $C_R = F(s_{init})$. Go to step 2.

Step 2: *Generate initial PCSs*. Create an empty priority queue Q. For $n \in Z(n_{start})$, create a PCS *s* with $t(s) = n, path(s) = [e(n_{start}, n)]$ and add it to Q. Go to step 3.

Step 3: *Check type of best solution in queue*. If Q is empty, return R and finish. Otherwise, let *s* be the first solution in Q. Remove *s* from Q. Depending of what type of solution *s* is, proceed as follows:

- If *s* is a PCS go to Step 4.
- If *s* is a CCS and $F(s) < C_R$, set R = s and $C_R = F(s)$ and go to Step 5.
- If *s* is a CCS and $F(s) \geq C_R$, go to Step 5.

Step 4: *Process PCS*. If $S(t(s))$ is empty and $F(s) \leq C_R$, set $S(t(s)) = s$ and then generate new solutions by applying Algorithm 5.2 to *s*. Add the newly generated solutions to Q and then return to step 2. If $S(t(s))$ is not empty or $F(s) > C_R$, return to Step 2.

Step 5: *Process CCS*. Proceed as follows:

- If $S(t(s))$ is a PCS, go to Step 6.
- If $F(s) < F(S(t(s))))$, set $S(t(s)) = s$ and $ties(s) = \{\}$, then generate new solutions using using Algorithm 5.3 with *s*. Add the newly generated solutions to Q. Return to Step 2.
- If $S(t(s))$ is a CCS and $F(s) = F(S(t(s)))$, add *s* as a tie solution to $ties(t(s))$ and then generate new solutions using Algorithm 5.3 with *s*. Add the newly generated solutions to Q. Return to Step 2.
- If $F(s) > F(S(t(s)))$, return to step 2.

Step 6: *Remove Dependent Partials*. For each $v \in path_n(s)$, where $S(v)$ is a PCS, perform the following steps in sequence for *v*, then return to step 2 after all $v \in path_n(s)$ have been processed.

a. For any $m \in \mathrm{N}, q = S(m)$, where q is a PCS and $path(S(v))$ is a subpath of q, do the following:
   - Set $S(m) = \{\}$
   - For any $j \in Z(m)$, if $S(j)$ is a PCS or if $j = n_{start}$, generate new solutions using Algorithm 5.2 with $s = S(j)$ and then add the newly generated solutions to Q.

b. For each $u \in Q$ where *u* is a PCS that contains $path(S(v))$ as a subpath, remove *u* from Q.

c. Set $S(v) = s_v$, where s_v is CCS identical to *s* except that $t(s_v) = v$. Generate new solutions from s_v using Algorithm 5.3 and add them to Q.

5.6 Experimental Results

To evaluate the performance of the four solution algorithms (the heuristic algorithm, VDPPA, A*, and branch-and-bound), 504 instances of the VDPP problem were generated. The VDPP problem involved optimizing the design of a UAV and the path it follows from a given start location to a given goal location. The problem instances were set in a region located in Maryland, Virginia, West Virginia, and Washington, D.C. The relevant objectives were reducing the risk to third parties and reducing the time required [103]. The UAV design variables were the flight speed, *x*(1), the wing reference area, *x*(2), and the flight height, *x*(3) (above ground level). All three design variables were continuous variables. Table 5.2 lists their ranges and initial conditions.

Table 5.2: UAV design variables

	Flight Speed *x*(1) (m/s)	Flight Height *x*(2) (m)	Wing Reference Area *x*(3) (m^2)
Lower Bound	30	1,024	12.17
Initial Conditions	50	1,600	16.17
Upper Bound	70	2,024	20.17

Two experiments were performed using these instances. In the first experiment (described in Section 5.6.1), all four algorithms were used to find solutions to one instance with multiple cost functions (combinations of third-party risk and time). In the second experiment (described in Section 5.6.3), the heuristic algorithm, VDPPA, and A* were used to find solutions to all 504 instances with the same objective function (third-party risk). The results for Experiment 1 and 2 are discussed in Section 5.6.2 and 5.6.4, respectively.

5.6.1 Experiment 1

In Experiment 1, the cost function was a combination of the risk to third parties and the time required. A surrogate model was developed to determine how the design variables affect the crash location probability distribution. Monte Carlo simulations were conducted of UAV crashes (using the approach described by Rudnick-Cohen *et al.* [103]) for all 18 combinations of the following design variable values: the lower bound, initial conditions, and upper bound of the flight speed; the lower bound and upper bound of the flight speed; and the lower bound, initial conditions, and upper bound of the wing reference area. When initializing each simulation run, the flight height and wing reference area were set to the given values, and the flight speed was determined by adding a random perturbation to the given value. This perturbation and the perturbations of the other initial conditions were the same as those of Rudnick-Cohen et al. [103].

Each simulation run required solving the system of ODEs in Table A.2 [119], in which the control surfaces were modeled as unactuated surfaces that could move freely between the upper and lower bounds on the control surfaces' angles. Each simulation run ended when the vehicle hit the ground, at which point (p_N, p_E) was taken as the crash location of the vehicle. Except for the three design variables, the vehicle parameters were those of a Cessna 182 [101]. For each configuration of design variables considered, 10,000 simulation runs were conducted in order to construct a crash probability distribution (CPD), representing the probability of the UAV crashing at different locations. Each CPD was represented using a 200 by 200 bin 2-D histogram of the crash locations from the 10,000 runs. For more details on the Monte-Carlo simulation approach used, see Appendix A.

A reference CPD was generated by simulating crashes for the design point with a flight speed, $x(1)$ of 50 m/s, a wing reference area, $x(2)$ of 16.17 m^2, and a flight height $x(3)$ of 1524 m. The other CPDs were modeled as transformations of the reference CPD. A Gaussian Process Regression model using a Matern 5/2 kernel was fit to this reference CPD using MATLAB's Regression Learner tool [82], so that the reference CPD could be treated as a continuously valued 2D probability density function. The final CPD used for design optimization was a transformation of the reference CPD via Eq. (5.6), where x and y are the displacement at which the UAV could crash relative to the UAV's position,

$CPD_{ref}(x,y)$ is the reference CPD and a, b_x, r_x, b_y, r_y and v are the parameters transforming the reference CPD, which are dependent on the design being considered.

$$CPD(x, y, a, b_x, b_y, r_x, r_y, v) = aCPD_{ref}(b_x x + r_x, b_y y + r_y) + v \tag{5.6}$$

Each CPD generated for the conditions in Table 5.2 was split into 64 equally sized pieces with a 25% overlap, least squares was then used to fit a, b_x, r_x, b_y, r_y and v values to each piece, using Eq. (5.6) to fit the probabilities of the bins inside that piece. In order to evaluate Eq. (5.6) for designs for which a, b_x, r_x, b_y, r_y and v were not computed, a K nearest neighbor (KNN) approach was used. The inputs for the KNN were *x*, *y* and the current design variables. The distance for the KNN was an L_2 norm between both the difference between the centroid of a piece and *x* and *y* and the difference between the design being evaluated and the design which a piece was generated from. The KNN chose the parameters for the 6 nearest pieces (k = 6) with a constraint that none of the pieces chosen could come from the same design variables. The KNN then approximated the value of Eq. (5.6) for a design by evaluating Eq. (5.6) with the parameters of the chosen pieces and then combining the resulting probabilities via a weighted sum with weights inversely proportional to the KNN's distance measure. The risk objective considered used the KNN to determine a binned CPD for the current design variables, with the centroid of each bin being used as the x, y inputs to the KNN for that bin. For computational efficiency, the binned CPD constructed for the risk objective had a 21 by 21 bin resolution. Note that the KNN model used in this chapter is not the same as the one detailed in Appendix A.

The risk value for each edge was evaluated using the risk metric from [103], using discrete grid of population data computed from US census block data [122] in place of census tracts. The time to traverse an edge was calculated by the dividing the length of that edge (along the ground) by the flight speed v. Both the risk and time values were scaled as follows: the risk value was divided by the maximum risk value in the graph under the initial design configuration in Table 5.2, the time value was divided by the time needed to travel a straight line path between the start and end locations. The scaled values were then combined using a weighted sum. Six combinations of weights were considered: [1, 0], [0.8, 0.2], [0.6, 0.4], [0.4, 0.6], [0.2, 0.8], and [0, 1].

For solving the design optimization problem, MATLAB's [80] fmincon solver was used with the active set method and objective function and design variable tolerances of 10^{-3}. The design variables were scaled into the range [0, 1]. The initial conditions and upper and lower bound constraints for each design variable in the design optimization problem were those detailed in Table 5.2. Both the VDPPA and A* generated solutions by setting these initial conditions for the first set of solutions generated and then using the design variables from parent solutions as the initial conditions for their child solutions. When computing the cost-to-go heuristic, the initial conditions from Table 5.2 were used for every edge's design optimization problem, to avoid any inconsistencies.

Table 5.3: Relevant parameters for Experiment 1

Start Longitude (deg)	End Longitude (deg)	Start Latitude (deg)	End Latitude (deg)	Number of nodes in graph	Number of edges in graph
-81.283	-76.952	37.945	38.670	560	3,964

5.6.2 Results for Time-risk VDPP in Experiment 1

Table 5.4 lists the results for all four algorithms on the six multi-objective cases considered. The units for the risk objective are expected fatalities. In case 1, when the objective function considers only the risk to third parties, the branch-and-bound algorithm was halted after calling the objective function 1,000,000 times. The heuristic algorithm and A* found solutions with the same path (but different values of the design variables), and the VDPPA and branch-and-bound found solutions with the same path (but different values of the design variables). Figure 5.3 shows the paths for these solutions. The heuristic algorithm required the least number of objective function calls. The VDPPA required slightly more objective function calls than A* did.

Table 5.4: Results from demonstration problem, $f(x)$ is the weighted cost function between the 2 objectives, $x(1)$ is Velocity (m/s), $x(2)$ is Wing Reference Area (m^2) and $x(3)$ is Flight Height (m)

Case #	Weight (Risk)	Weight (Time)	Approach	$f(x)$	Time (hrs)	Risk (10^{-5})	Func. Calls	$x(1)$	$x(2)$	$x(3)$	# of PCS and CCS checked
1	1	0	Heuristic	0.0786	0.886	3.609	9363	70	16.04	1517	N/A
			A*	0.0685	0.886	3.146	16524	70	15.62	1407	58
			VDPPA	0.0683	0.876	3.135	16708	70	15.62	1407	280
			B & B	0.0683	0.876	3.138	$>10^6$	70	15.75	1439	164,133
2	0.8	0.2	Heuristic	2.7434	0.374	9.205	6966	70	16.19	1524	N/A
			A*	2.7426	0.374	9.160	8420	70	16.18	1524	15
			VDPPA	2.7426	0.374	9.160	8420	70	16.18	1524	49
			B & B	2.7426	0.374	9.160	8420	70	16.18	1524	104
3	0.6	0.4	Heuristic	5.2871	0.374	9.261	7383	70	16.18	1524	N/A
			A*	5.2858	0.374	9.159	8776	70	16.18	1524	16
			VDPPA	5.2858	0.374	9.159	8776	70	16.18	1524	50
			B & B	5.2858	0.374	9.159	9304	70	16.18	1524	235
4	0.4	0.6	Heuristic	7.8300	0.374	9.280	6985	70	16.18	1524	N/A
			A*	7.8302	0.374	9.297	8098	70	16.17	1524	16
			VDPPA	7.8300	0.374	9.280	8098	70	16.18	1524	50
			B & B	7.8300	0.374	9.280	8098	70	16.18	1524	104
5	0.2	0.8	Heuristic	10.373	0.374	9.244	6950	70	16.17	1524	N/A
			A*	10.372	0.374	9.104	8030	70	15.46	1524	15
			VDPPA	10.372	0.374	9.104	8030	70	15.46	1524	49
			B & B	10.372	0.374	9.104	8030	70	15.46	1524	104
6	0	1	Heuristic	12.905	0.373	14.68	6932	70	16.17	1524	N/A
			A*	12.905	0.373	14.68	7976	70	16.17	1524	15
			VDPPA	12.905	0.373	14.68	7976	70	16.17	1524	49
			B & B	12.905	0.373	14.68	7976	70	16.17	1524	104

In the other five cases, the four algorithms found solutions with the same path. In cases 3 and 6, these solutions also had the same values of the design variables. In case 4, the solution found by A* had a slightly different value for the wing reference area (variable $x(2)$), which increased the risk to third parties. In case 5, the solution found by the heuristic algorithm had a greater value for the wing reference area (variable $x(2)$), which increased the risk to third parties. In these cases, the VDPPA and A* required the same number of objective function calls. The computational effort (the number of objective function calls and the number of PCS and CCS considered) in these cases was less than the computational effort in case 1. These results indicate that, as the weight on the risk to third parties decreases, it is easier for the algorithms to find high-quality solutions quickly.

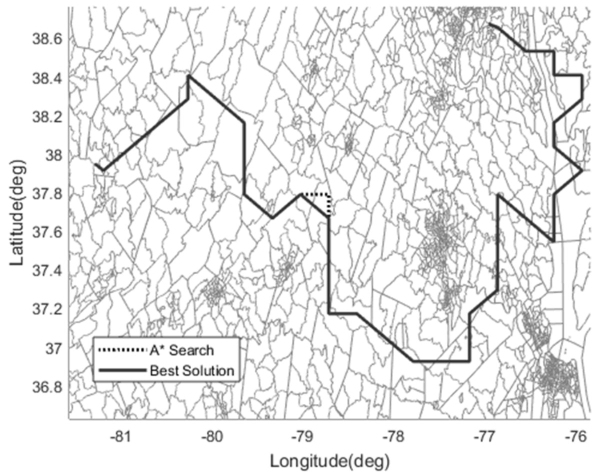


Figure 5.3: Comparison of the path of the solution found by A* and the path of the solution found by branch-and-bound and VDPPA for case 1.

5.6.3 Experiment 2

A set of 504 different instances with randomly determined grid spacing, start points and end points (as detailed in Table 5.5) were generated and used to compare the solutions found by the heuristic algorithm, A* and VDPPA. The demonstration problem instance detailed above was one of the randomly generated cases. All the randomly generated cases were required to have a minimum distance of three degrees between the start and end points. All nodes and edges in the randomly generated cases were constrained to be within one of the three U.S. states of, Maryland, Virginia or West Virginia, or within Washington D.C, any nodes and edges outside this region were removed from the graphs being generated. Each method was only run for the risk objective in each case, due to the risk objective being more heavily affected by the design variables than the time objective. In these instances, the objective function included only the risk to third parties.

Table 5.5: Parameters used to generate cases for Monte Carlo study

	Latitude (deg)	Longitude (deg)	Max number of nodes along x axis in graph	Max number of nodes along y axis in graph
Min	-83	36.5	15	15
Max	-75	39.75	35	35

5.6.4 Results for Experiment 2

For 504 instances generate for Experiment 2, the number of times that each algorithm found the best solution, how often their solutions' paths differed, and the number of objective function calls were tallied. Table 5.6 summarizes these results.

For these instances, compared with the VDPPA, the heuristic algorithm required less computational effort (about 65% of the number of objective function calls), but A* required nearly the same computational effort (99% of the number of objective function calls). The VDPPA found the best solution for every instance. The heuristic algorithm found the best solution in about 40% of the instances, and A* found the best solution in over 50% of the instances. In several instances, A* and the heuristic algorithm found the same optimal path as VDPPA but had worse designs for that path. In 5 of the 33 instances where A* did not find same path as VDPPA neither A* nor the heuristic algorithm were able to find VDPPA's path. In the remaining 28 instances, the heuristic algorithm found the same path as VDPPA, indicating the heuristic algorithm alone would have been sufficient to find the optimal solution. However, it should be noted that A* did better overall at finding the same paths as VDPPA than the heuristic algorithm did.

Table 5.6: Results summary from the 504 randomly generated problems that only considered the risk objective

Method	Frequency of finding best solution	Frequency that best solution involved different path than A*	Frequency that best solution involved different path than Heuristic	Obj. Fun. calls as a percentage of calls made by VDPPA (mean)	Obj. Fun. calls as a percentage of calls made by VDPPA (standard deviation)	Obj. Fun. calls as a percentage of calls made by VDPPA when not best solution (mean)
Heuristic	219	28	N/A	64.83%	1.702%	65.64%
A*	277	N/A	74	99.28%	17.95%	99.10%
VDPPA	504	33	79	N/A	N/A	N/A

5.7 Concluding Remarks

This chapter introduced the VDPP problem, which requires simultaneously optimizing the design of a UAV and determining the path that it should follow to complete a mission. Unlike problems considered in other research, this VDPP problem has continuous design variables and a path that is a connected sequence of nodes and edges in a graph.

This chapter presented VDPPA, a new search algorithm for solving this problem, a new cost-to-go heuristic and a new heuristic algorithm that constructs a feasible solution while computing the cost-to-go at the start node. Results were presented from computational experiments performed to evaluate the performance of these two algorithms and compare them with a version of A*. When used to find solutions for 504 randomly generated instances, the VDPPA always found the lowest cost solution, while the heuristic algorithm and A* did not. The computational effort of the heuristic algorithm was much lower than the computational effort of A* and VDPPA, which required approximately the same computational effort.

Since it can find better solutions with about the same computational effort as A*, VDPPA is a significant improvement in finding solutions to design and planning problems for more general domains. However, VDPPA is limited to problems in which there is at most one edge between any two nodes in the graph, so the algorithm cannot be used to find solutions to problems where there are multiple ways to transition between nodes.

However, the gap between VDPPA and A*'s computational cost may increase when VDPPA is run on graphs containing larger numbers of nodes and edges, since this would increase the number of CCS that VDPPA would need to search. However, VDPPA's computational cost is likely still much better than that of branch and bound, which was observed to be significantly more expensive than VDPPA during Experiment 1 (Section 5.6.1).

Although the VDPP has applications in many domains, the formulation in Eq. (5.1) presumes that the *X*, set of feasible designs, is independent of the path selected. That is, the constraints on the design variables do not depend upon the path. This may not hold in some applications, such as when a path requires UAV maneuvers that make some UAV designs infeasible. Although discretizing the design space is one approach for handling such situations [84], the design space and state space of many systems are both high-dimensional, such as the numerous components and complex dynamics present in a UAV. Thus, additional research is needed to develop methods that can handle a continuous variable design space and a continuous variable state space in order to fully solve optimal design and path planning problems for UAVs and other unmanned systems.

This chapter discussed methods for solving design and path planning optimization problems where path planning occurs on a graph. The next chapter extends these methods to work on motion planning problems, by running them on a motion planning graph.

Chapter 6: UAV Design and Motion Planning Optimization

In this chapter, the problem of design and motion planning optimization is formulated and solved. Unlike the design and path planning problem considered earlier, in a design and motion planning problem both the design variables and the trajectory an unmanned vehicle will travel are defined using continuous spaces. This allows for more sophisticated models of vehicle motion than the simple 2D graphs considered in Chapter 5, which can account for vehicle dynamics and which can involve more than two dimensions. These more sophisticated models of vehicle motion are particularly important when simultaneously optimizing the design and motion of UAVs, since UAVs can have up to 6 degrees of freedom in which they are capable of motion.

This chapter is organized as follows. Section 6.1 presents the problem of design and motion planning optimization. Section 6.2 provides a formulation for the problem of design and motion planning optimization. Section 6.4 presents an approach for solving design and motion planning problems. Section 6.5 tests the proposed approach on the risk-based motion planning problem previously considered in Chapter 2 and compares its results against those of a design study. Section 6.6 summarizes the results of this chapter and presents some concluding remarks.

6.1 Design and Motion Planning Optimization

The problem of design and motion planning optimization can be viewed as being similar to the VDPP problem formulated in Chapter 5, except that the path is now no longer constrained to exist on a graph. In practice a graph structure is still needed to represent the connectivity between different configurations, as done in [9], [33] and [84], though trajectory optimization [44] and optimal control [44] have also been used. The graph based approaches use sampling based motion planners to construct the graphs they use to determine the optimal paths, using either global search [9] or a discrete design set ([33], [84]) to handle the designs.

In this chapter, a new approach to solving design and motion planning problems using sampling based motion planning is proposed. Rather than relying on putting the motion planning problem inside a design optimization formulation, a motion planner is instead used to determine a graph of reachable configurations, which can then be used to formulate a VDPP problem in order to simultaneously optimize the route through the motion planning graph and the optimal design. This differs from all previous approaches to solving design and motion planning problems, which have relied on completely decoupling the problems of design optimization and motion planning into separate optimization problems.

6.2 Formulation of Design and Motion Planning Problems

Below, a formulation for the problem of optimal design and motion planning is given:

1. Let C be the configuration space, the set of all feasible configurations for our vehicle.
2. Let c_s be the vehicle's initial configuration of the vehicle; let c_f be the desired final configuration.

3. For any c_1 and $c_2 \in C$, let $B\left(c_1, c_2\right)$ be the set of all possible solutions to the BVP between configurations c_1 and c_2, where $s \in B\left(c_1, c_2\right)$ is a continuous sequence of configurations from c_1 to c_2 that satisfies all of the dynamics constraints present.
4. Let $X \subseteq R^d$ be the feasible domain for the *d* design variables, such that any design *x* is only feasible if $x \in X$.
5. Let $f\left(x, s\right)$ be the cost of design *x* moving along s. For any configurations c_1, $c_2 \in C$ and for any design $x \in X$, $f\left(x, s\right) \geq 0$ for all $s \in B(c_1, c_2)$.

Then the problem of optimal design and motion planning has the formulation given in Eq. 6.1.

$$\begin{aligned} &\min_{x,s} f(x,s) \\ &\text{Subject To:} \\ &x \in X \\ &s \in B\left(c_s, c_f\right) \end{aligned} \tag{6.1}$$

6.3 Proposed Approach for Design and Motion Planning

When a motion planning graph is available, Eq. 6.1 becomes an instance of a VDPP problem (see Chapter 5). For computational efficiency, it is preferable to attempt to construct the graph for a design and motion planning problem in advance using a method such as PRM, rather online using a method such as RRT#. Because the VDPP problem does not satisfy the principle of dynamic programming (see Chapter 5), the entire VDPP problem will need to be resolved whenever a configuration is added into the graph during RRT#, since the new configuration could be part of the optimal solution. Thus, it is more efficient to first construct the motion planning graph and then solve the VDPP problem on it, as this only requires solving the VDPP problem once.

VDPPA can thus be easily extended to motion planning problems by taking advantage of the fact that it must first compute the optimistic cost-to-go heuristic. The optimistic cost-to-go heuristic (see Chapter 5) satisfies the principle of dynamic programming, thus it can be used as the objective function for RRT$^{\#}$. The graph which RRT$^{\#}$ computes can then be used by VDPPA as it solves the VDPP problem on that graph. A step-by-step description of this process is given below:

1. Use RRT$^{\#}$ to solve a motion planning problem between the goal node (configuration c_f) to the start node (configuration c_s), where the objective is the cost-to-go heuristic proposed in Chapter 5. Note that RRT$^{\#}$ will also compute a search tree connecting each node in the motion planning graph to the start node during this step, which computes the optimistic cost-to-go heuristic for all nodes in the graph.
2. The path RRT$^{\#}$ finds for the cost-to-go heuristic is turned into an actual solution to the design and motion planning problem using the heuristic algorithm from Chapter 5, Section 5.2.2.
3. The initial solution is used with VDPPA (See Chapter 5) to solve for the optimal design and path for the current motion planning graph

6.4 Design and Motion Planning Experiments

The proposed approach was tested for solving design and motion planning problems on the risk based motion planning problem from Chapter 2, with the following design variables added into the problem, wingspan, mass, rudder range and flight speed. The surrogate model for a Cessna 182 crash distribution developed in Appendix A is used to approximate the crash distribution for different configurations, however its resolution was reduced to being a 11 by 11 grid of bins. Note that in the motion planning model, flight speed affects the turning radius of the UAV, with higher flight speeds requiring larger radii for banked turns. Thus, the flight speed design variable is capable of affecting both crash distribution shape and also the shape of the trajectory of the UAV. The risk objective was scaled by a factor of 10^{-4}, while the time objective was unscaled. The objective weights $(w_{t,} w_r) = (0.05; 0.95)$ were used. This weighting puts a majority of weight on the risk objective, while still prioritizing the use of time optimal trajectories when flying over uninhabited or nearly uninhabited areas.

The motion planning graph used was constructed by using the approach detailed in Chapter 2, however configurations were only sampled from a uniform distribution over the configuration space. Only the time optimal Dubins curve was used when solving a BVP between two configurations. An initial connection radius of 3000 (γ_0 = 3000) was used with d = 5. RRT$^{\#}$ was run for 3000 iterations in order to generate the motion planning graph.

For comparison, a Latin hypercube sampling Design of Experiments (DOE) was used to generate 100 different design configurations. The motion planning problem considered in this chapter was solved for each of these 100 design configurations, by running RRT$^{\#}$. This design study provided comparable results to a non-deterministic optimization approach like that of Baykal and Alterovitz [9] or an approach searching a fixed set of design configurations like that of Denarie et al. [84]. Note that a separate motion planning problem was solved for each design configuration, thus each design configuration had a different motion planning graph.

For additional comparison, the A* and heuristic methods proposed in Chapter 5 are also run on the same motion planning graph as VDPPA using the proposed approach. However, the number of legs evaluated by these approaches was not tracked, as they must be less than that of VDPPA's.

6.5 Experiment Results

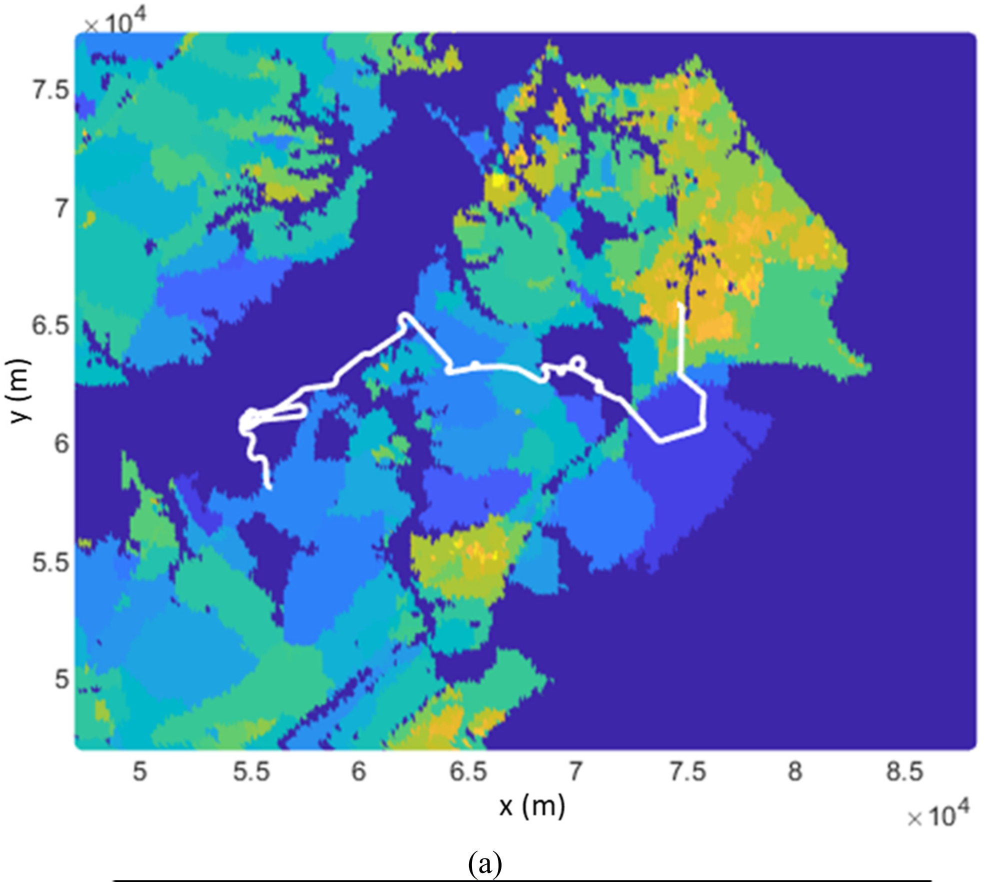
Table 6.1 provides results from the proposed design and motion planning approach and the best solution from the DOE of 100 different design configurations. Only one solution from the 100 design DOE had a lower cost than the solution found by the proposed approach using VDPPA. In total, almost 400 million leg evaluations were required for the 100 designs in the DOE, whereas the proposed approach only required around 50 million leg evaluations. The solution found using VDPPA with the proposed approach is similar to the best solution from the DOE, with a slightly lower flight speed and a slightly higher wingspan.

Table 6.1: Solutions to design and motion planning example problem

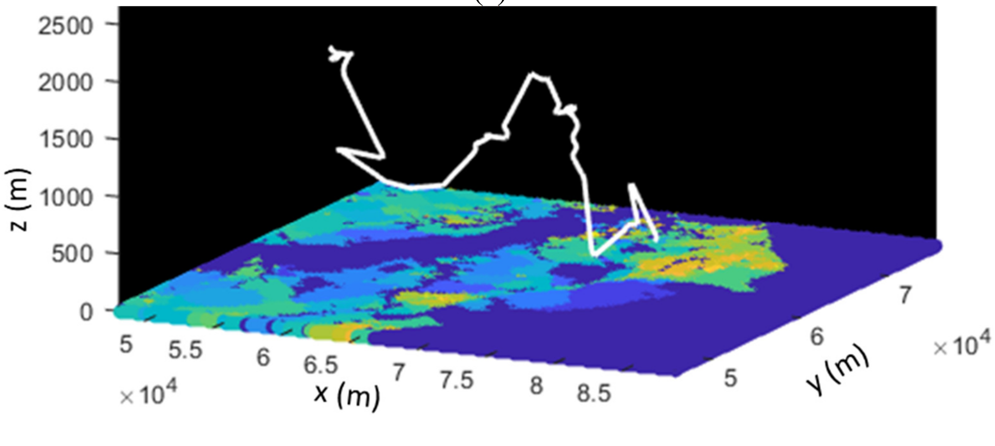
Solution	Speed (m/s)	Wingspan (m)	Rudder Range (deg)	Mass (kg)	Best Cost	Cost for VDPPA's best path	Cost for DOE's best path	Leg evaluations
Heuristic	46.45	13.85	32.21	1,144	2,501	2,496	30,700	Not counted
A*	45.98	16.21	32.58	1,144	2,444	2,455	2,671	Not counted
VDPPA	42.5	10.35	34.52	1,143	2,312	2,312	2,873	54,609,632
DOE	45.19	9.0244	38.14	1,169	2,168	2,373	2,168	395,238,250

Figures 6.1 and 6.2 show the trajectories for these two solutions, which are similar. However, it can be seen that Figure 6.1 (VDPPA's trajectory) comes closer to the edge of

the Patuxent river than Figure 6.2's trajectory does. This is caused by the DOE's solution having a motion planning graph containing edges which provide an efficient route for avoiding the edge of the river, which gives it a lower cost. Additionally, the DOE's motion planning graph also contains edges allowing for a solution that more quickly moves away from the starting configuration, which reduces the risk psoed. Curiously, while VDPPA's design is the lowest cost design observed for VDPPA's trajectory, it's cost becomes significantly higher if its cost is computed for the optimal trajectory of the DOE's best solution. However, the DOE's best design still performs well on VDPPA's trajectory. Similar behavior can also be obvserved in the solutions found using the heuristic approach and A* search. Both A* search and the heuristic approach find different optimal designs and trajectories than VDPPA in the example problem.

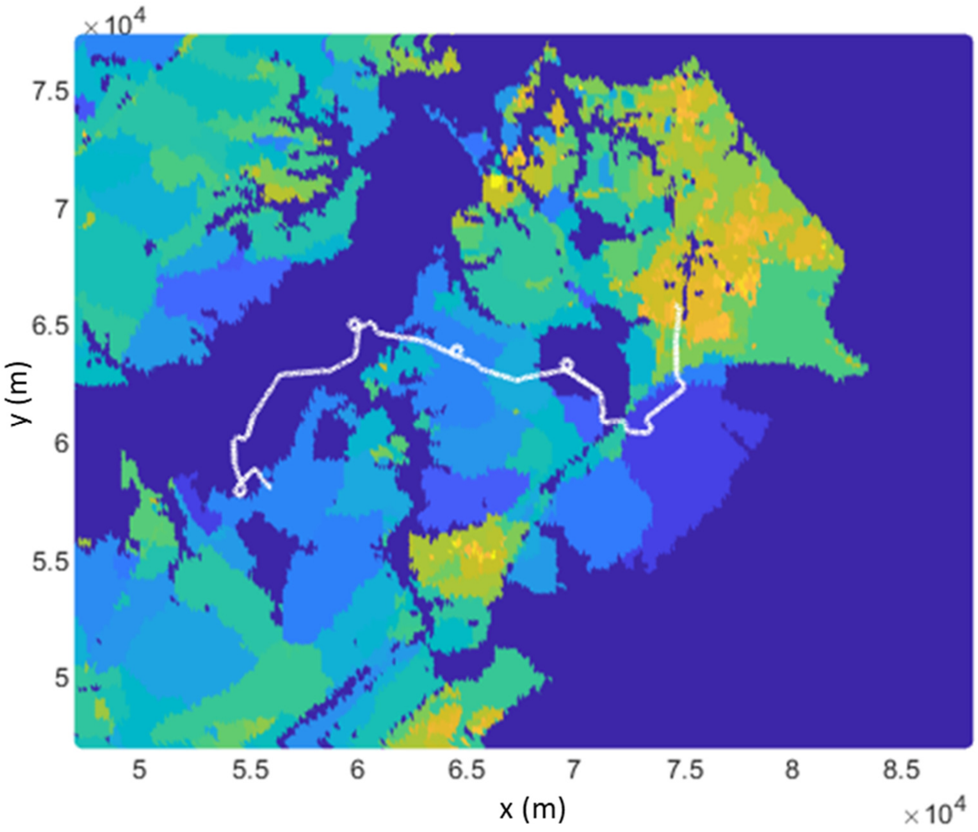


(a)

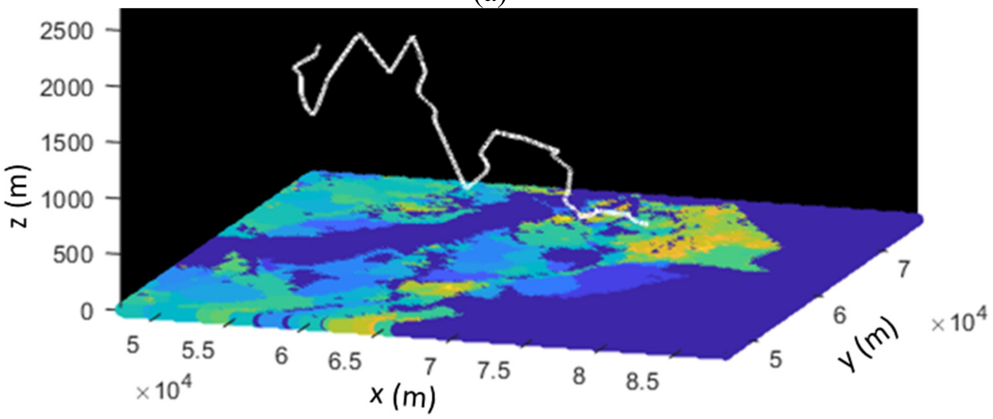


(b)

Figure 6.1: Optimal trajectory for solution found by using VDPPA with proposed design and motion planning approach. (a) 2D view (b) oblique view of trajectory in 3D.



(a)



(b)

Figure 6.2: Optimal trajectory for best design from latin hypercube DOE. (a) 2D view (b) oblique view of trajectory in 3D.

6.6 Concluding Remarks

The proposed approach using VDPPA found the best solution for the motion planning graph used by the proposed approach. However, the solution with the best cost found was one of the solutions computed in the DOE, though the remaining 99 solutions in the DOE all had worse costs than VDPPA's solution. The best solution from the DOE has a similar design to the design found by VDPPA. However, the DOE uses a different motion planning graph for each design, which allows it to find a better trajectory than is possible with the motion planning graph VDPPA used.

However, the DOE was approximately 8 times more computationally expensive than the proposed approach using VDPPA, meaning that the proposed approach was more efficient at solving design and motion planning optimization problems. If more RRT$^{\#}$ iterations were used for motion planning it is possible that this trend might change, as VDPPA would need to search more solutions, while the DOE's computational cost would only scale up in the same manner as RRT$^{\#}$. However, VDPPA's computational cost would only increase linearly if additional design variables were considered (due to extra gradient computations). In comparison, the DOE would require more samples to obtain the same average distance between the samples in its latin hypercube sampling of the design space, which would result in a worse than linear increase in computational cost.

Since the DOE was computed using latin hypercube sampling, each design in the DOE located so that the DOE evenly covers the entire design space. Given that only one solution from the DOE with a lower cost than VDPPA's solution and VDPPA's solution has a similar design to the best design from the DOE, it is reasonable to conclude that the optimal design for the example considered lies in the same region as these two designs. However, one does not see similar solution costs if one swaps the trajectories computed for each of the designs. The DOE's best design still performs well on the trajectory computed by VDPPA, but VDPPA performs significantly worse if it is evaluated on the DOE's best trajectory (see Table 6.1). This indicates that either (1) the solution computed by VDPPA is not robust to changes in the UAV's trajectory or (2) the best solution from the DOE's path is not robust to small changes in the UAV's crash distribution. Given that the designs found by using the proposed approach with the heuristic approach and A* search also do poorly on the DOE's best trajectory, it appears that (2) is more likely to be the case. However, this can only be assessed by solving the example considered as an actual robust optimization problem.

This chapter formulated and presented an approach for solving deterministic design and motion planning problems. The next chapter leverages the robust optimization techniques developed in Chapters 3 and 4 to introduce uncertainty into design and motion planning problems.

Chapter 7: Robust UAV Design and Motion Planning Optimization

This chapter formulates a robust optimization variant on the design and motion planning optimization problem. This robust design and motion planning optimization problem is solved by using a variation on the SGLRO approach presented in Chapter 3, where the inner optimization problem is a scenario based variant on the design and motion planning optimization problem from Chapter 6.

7.1 Motivation

A major limitation of design and motion planning optimization problems is that the optimal design found may only perform well for the exact sequence of motions it was computed for. In the real world, unmanned systems cannot perfectly execute a precomputed trajectory due to environmental factors that cannot be modeled and due to uncertainty affecting the systems performance. Thus, a solution to a design and motion planning problem risks being overspecialized for the optimal trajectory it is associated with. As seen in Chapter 4, uncertain wind conditions can have a significant impact of UAV performance. As UAVs are typically flown outdoors, they cannot avoid this uncertainty. Thus it is critical to account for uncertainty when simultaneously optimizing the design and motion of a UAV, so that a design and trajectory can be found which actually performs well in the real world.

If a design and motion planning optimization problem is formulated as a robust optimization problem, then this issue of overspecialization can be mitigated. By modeling uncertainty and unknown environmental factors as uncertain parameters, a design can be found which performs well under a range of possible conditions.

7.2 Strategy for Solving Robust Design and Motion Planning

The scenario robust optimization problem formulations discussed in Chapter 3 that use discrete scenario sets (e.g. Eq. (1.2), Eq (2.1)) are conventional optimization problems, thus they can easily be used as the "design optimization problem" in a design and motion planning optimization problem. Thus, the concept of scenario robust optimization can be used to formulate and solve robust design and motion planning optimization problems.

Notably, the cost-to-go heuristic only needs to be computed once under a nominal scenario. Since this scenario could be a worst case scenario, any costs computed under it must be a lower bound on actual worst case cost. Thus, the lower bound computed by the cost-to-go heuristic is a lower bound for robust design and motion planning problem's cost, meaning that it is still an admissible heuristic.

7.3 Formulation of Robust Design and Motion Planning Problems

Below a robust formulation of the formulation previously detailed in section 7.1 is given, which extends the definitions given in section 7.1 as follows:

1. Let J be the number of constraints $q_j(u)$ needed to define the set of possible scenarios.
2. Let $\mathbf{U} = \{u \mid q_j(u) \leq 0, \forall j = 1, \ldots, J\}$ be the complete set of possible scenarios.

3. Let $f\left(x,s,u\right)$ be the cost of design x moving along s. For any configurations c_1, $c_2 \in C$, for any design $x \in X$ and for any scenario $u \in \mathbf{U}$, $f\left(x,s,u\right) \geq 0$ for all $s \in B(c_1,c_2)$.

Then the robust optimal design and motion planning problem has the formulation given in Eq. 6.2.

$$\begin{aligned}
&\min_{x,s} \max_{u} f(x,s,u) \\
&\text{Subject To:} \\
&x \in X \\
&s \in B\left(c_s,c_f\right) \\
&u \in \mathbf{U}
\end{aligned} \tag{7.1}$$

The formulation of Eq. 7.1 can then be reformulated as a scenario robust optimization problem with a set of scenarios, which is given in Eq. 7.2. Let $U = \left\{u_1,\ldots,u_K\right\}$ be a finite set of *K* scenarios.

$$\begin{aligned}
&\min_{x,s} z \\
&\text{Subject To:} \\
&z \geq f(x,s,u_k), \forall u_k \in U \\
&x \in X \\
&s \in B\left(c_s,c_f\right)
\end{aligned} \tag{7.2}$$

The scenario robust optimization problem in Eq. 7.2 is a deterministic optimization problem. Thus, it can be used as the inner optimization problem in a VDPP problem, meaning that VDPPA can be used to solve Eq. 7.2. Unlike the robust motion planning approach detailed in Chapter 4, there is no need to solve an integer programming problem.

7.4 Proposed Approach for Robust Design and Motion Planning

Thus, the following approach can be used to solve robust design and motion planning problems:

1. Use RRT# [7] to solve a motion planning problem between the goal node (configuration c_f) to the start node (configuration c_s), where the objective is the cost-to-go heuristic proposed in Chapter 5. Note that RRT# will also compute a search tree connecting each node in the motion planning graph to the start node during this step, which computes the optimistic cost-to-go heuristic for all nodes in the graph.
2. The path RRT# finds for the cost-to-go heuristic is turned into an actual solution to the design and motion planning problem using the heuristic algorithm from Chapter 5, Section 5.2.2.
3. The initial solution is used with VDPPA (See Chapter 5) to solve for the optimal design and path for the current motion planning graph under a nominal scenario
4. The SGLRO algorithm (See Chapter 3) is run to solve Eq. 7.2, by using VDPPA to solve for the optimal design and path for the current motion planning graph under the current set of scenarios. SGLRO's set of scenarios is initialized with the nominal scenario. The following alterations are made to SGRLO and VDPPA:
    a. SGLRO uses it's current solution as the initial solution for VDPPA, in place of a static initial solution. The first initial solution used is the solution computed in Step 3.

b. VDPPA keeps track of the current robust optimal design and worst case cost for each path that it computes a robust optimal design for. VDPPA uses this to avoid solving unnecessary optimization problems, by using the following procedure when solving for the robust optimal design for a path *s*:

    i. The cost of path *s*'s previous optimal design is checked under the most recently added scenario.

    ii. If the worst case cost of the previous robust optimal design is less than its cost in the most recently added scenario, then go to step 4.b.ii.1. Otherwise, go to step 4.b.ii.2.

        1. Return the previous robust optimal design as the current robust optimal design. Otherwise,

        2. Solve the robust design optimization problem for path *s* to get the new robust optimal design for *s*. Update the current optimal design and worst case cost stored for *s*.

Note that steps 1-3 are the same as running the deterministic motion planning approach presented in Section 6.3. The two alterations made to SGLRO and VDPPA help reduce the computational cost of repeatedly solving a VDPP problem within SGLRO's framework. Using the current solution as the initial conditions for VDPPA helps minimize the number of function calls which VDPPA makes when solving its design optimization problem, since the current solution is likely to be closer to the next solution VDPPA will find. Additionally, VDDPA can minimize the number of function calls it makes Using the last computed solution for a path helps avoid repeatedly solving the same scenario robust optimization problems on different iterations of SGLRO.

7.5 Proposed Robust Design and Motion Planning Experiments

The robust design and motion planning approach was also used to solve the risk based motion planning problem presented in Chapter 6, with the uncertain wind vector field from Chapter 4 present. For comparison, a Latin hypercube sampling DOE was used to generate 20 different design configurations. The robust motion planning approach from Chapter 4 was solved for each of these 20 different design configurations. Like in the DOE considered in Chapter 6 a separate motion planning problem was solved for each design configuration, thus each design configuration had a different motion planning graph. The worst case performance of the deterministic design and path planning approaches from Chapter 6 are also compared against the proposed approach and the best solution from the DOE. Worst case scenarios and costs are found for all approaches, using a set of 5000 scenarios generated using latin hypercube sampling.

The proposed robust design and motion planning approach was run for 100 randomly sampled scenarios (100 iterations of SGLRO). The proposed robust design and motion planning approach used the motion planning graph generated by the design and motion planning approach in the example problem considered in Chapter 6.

7.6 Results

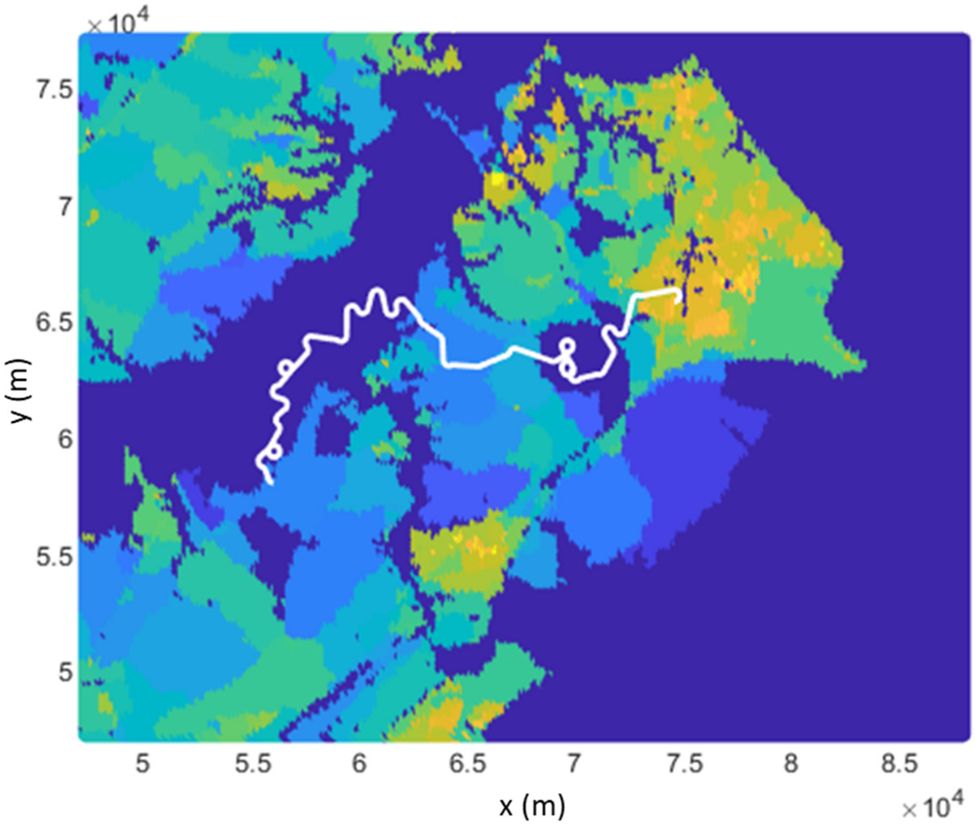
Table 7.1 shows the solutions found by all approaches compared in this chapter. The proposed robust design and motion planning approach converged to its final solution after generating one scenario. After generating that scenario, all remaining scenarios did not produce higher worst case costs. The solution found by the proposed robust design and motion planning approach had the same optimal trajectory as the solution found by VDPPA in Chapter 6. Figure 7.1 shows the best trajectory found by the DOE using the robust motion planning approach.

Table 7.1: Solutions to robust design and motion planning example problem

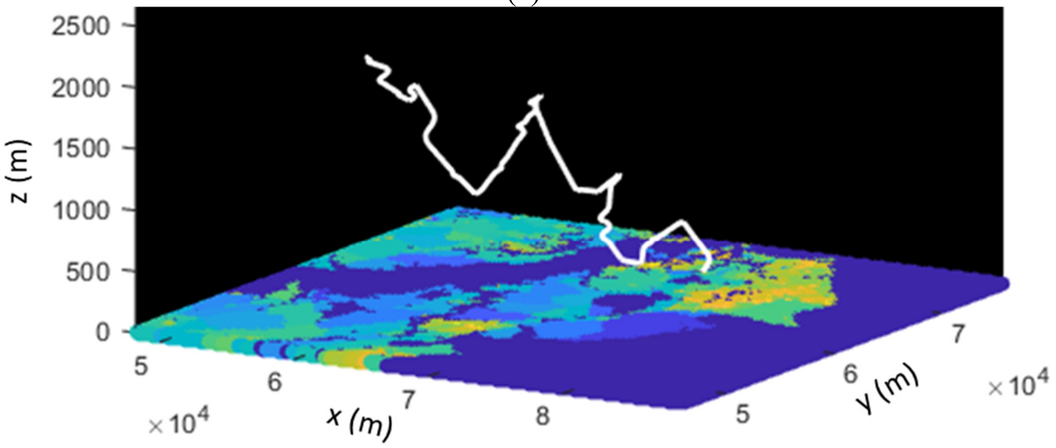
Solution	Speed (m/s)	Wingspan (m)	Rudder Range (deg)	Mass (kg)	Best Cost (computed by approach)	Worst Case Cost	Leg evaluations
VDPPA (deterministic)	42.5	10.35	34.52	1,143	2,312	3,331	54,609,632
DOE (deterministic)	45.19	9.0244	38.14	1,169	2,168	3,229	395,238,250
VDPPA (robust)	42.78	10.27	34.51	1,143	3,241	3,321	571,834,652
DOE (robust)	30.67	10.84	36.90	1,148	2,645	2,637	823,606,666

The solution with the best worst case performance was the best solution from the DOE using the robust motion planning approach, which found a solution with a significantly lower worst case cost than the other approaches. Notably, the worst case cost computed by the robust motion planning approach is higher than the one computed using the 5000 scenarios used to compare results, indicating that the trajectory of the best solution from this chapters DOE is robust optimal for the design it was generated for.

The solution with the next smallest gap between its worst case cost and its computed cost was the proposed motion planning approach, which had a much smaller gap than deterministic case solutions found in Chapter 6. This smaller gap demonstrates that proposed robust design and motion planning approach is able to find a robust optimal solution, which can be different than the deterministic case optimal solution (see Section 4.7 for examples). Despite finding the same optimal trajectory as the deterministic design and motion planning approach, the proposed robust design and motion planning approach required a much higher number of leg evaluations (over half a billion) than the deterministic design and motion planning approach using VDPPA did.



(a)



(b)

Figure 7.1: Plot of robust optimal trajectory for the best design from the DOE conducted in chapter 7. (a) 2D plot (b) oblique view of trajectory in 3D

While the DOE using the robust motion planning approach required more leg evaluations than the proposed design and motion planning approach, it also found lower cost solutions than the solution found by the proposed design and motion planning approach for three of the 20 designs in the DOE. In comparison the only one out of the 100 designs in the DOE from Chapter 6 had a lower cost trajectory than the one found by the deterministic design and motion planning approach from Chapter 6. This indicates that a smaller DOE (e.g. 10 designs instead of 20) is also capable of finding a better solution than the proposed robust design and motion planning approach. The computational cost running the robust motion planning approach for each design in the DOE is proportional to the number of designs in it, thus a smaller DOE would likely be faster than the proposed robust design and motion planning approach.

7.7 Concluding Remarks

Surprisingly, the results obtained indicate that the most effective method for solving robust design and motion planning optimization problems is actually conducting a DOE using a robust motion planning approach. The proposed robust design and motion planning approach had the second smallest gap between its actual worst case cost and the cost it computed as its worst case cost, demonstrating that it was capable of finding a robust optimal solution to a design and motion planning problem. The best solution from the DOE conducted in Chapter 6 had a worst case cost significantly closer to that of the solutions computed using VDPPA. This indicates that the best solution from Chapter 6's DOE was not robust to small changes in its crash distribution (such as the wind field considered in this chapter or the alternate designs it was evaluated for in Chapter 6). However, the optimal trajectory from Chapter 6's DOE is still slightly better than any trajectories possible in the motion planning graph used by the approaches using VDPPA, even when uncertainty is considered.

Curiously, the robust design and motion planning approach found a similar solution to the deterministic design and motion planning approach, using the same optimal trajectory with a very slightly different design. Similar behavior was observed in several of the motion planning graphs in Chapter 4, when the deterministic solution was also the robust optimal solution. Given that the deterministic solution was also not the robust optimal solution for several of the motion planning graphs in Chapter 4, this behavior should not always be expected. When the deterministic optimal solution has a different optimal trajectory than the robust optimal solution, it is likely that the proposed robust design and motion planning approach may need to generate more than the single scenario needed in this chapter's example. Each additional scenario would effectively multiply the number of leg evaluations present in Table 7.1, which would rapidly increase the computational cost of the proposed robust design and motion planning approach

In comparison, the DOE using the robust motion planning approach was able to compute a different motion planning graph for each design it considered and found several solutions with lower costs than the other approaches compared. The poor performance of the best solution from Chapter 6's DOE indicates that a motion planning graph containing trajectories with low worst case costs may not be one which also contains trajectories with low deterministic case costs. Thus, it is necessary to use a robust motion planning approach to determine which motion planning graphs contain solutions with low worst case costs.

It is interesting to note that the computational cost of the robust design and motion planning approach is significantly higher than that of the deterministic design and motion planning approach. In theory, the computational cost of the robust design and motion planning approach should be around 2-3 times that of the deterministic design and motion planning approach, since there are now two scenarios present instead of 1. However, the actual computational cost is over 10 times that of the deterministic design and motion planning approach. This can be attributed to two causes. (1) The scenario based robust optimization problem given in Eq (7.2) is more expensive to solve than Eq. (6.1), due to it being a constrained optimization problem, which requires additional function evaluations to be solved. (2) Because the optimal cost of the robust design and motion planning problem is higher, VDPPA needs to search through a much larger number of PCS than in the deterministic problem, which increases computational cost. It can thus be seen that the robust design and motion planning problem is in general a significantly more computationally expensive optimization problem than a design and motion planning optimization problem that does not consider uncertainty. Consequently, using a DOE with a robust motion planning approach becomes a more practical approach for solving the robust design and motion planning optimization problem. However, If more design variables are present then the robust DOE would likely require a much larger number of sampled designs (and thus a much larger computational cost). Thus neither of the two robust design and motion planning approaches presented will perform well on larger robust design and motion planning optimization problems.

A major limitation of using a DOE for design optimization is that it is effectively just a random search over the design space, thus the solutions it finds are not necessarily the optimal solution, though they may have low costs. In comparison, proposed robust design and motion planning approach is a local optimization based approach, thus while it does not have this issue, it is vulnerable to local optima. The best solution from the DOE using the robust motion planning approach had a noticeably different design, with a much lower flight speed, than the one found by Chapter 6’s DOE and the VDPPA using approaches. This indicates the local optima that the proposed robust design and motion planning approach finds is far from the global optimum for the problem considered. Given that the flight speed of the UAV affects which type of Dubins curve is used when going between two configurations, it is likely that the proposed robust and design motion planning approach might find a lower cost solution if it were to use a different initial design and motion planning graph. However, the only way to find such a design is to use the robust motion planning approach with some form of a DOE. While such a combined approach may be extremely computationally expensive, it also shows the most promise for finding robust optimal solutions to robust design and motion planning optimization problems.

This chapter presented an approach for solving robust design and motion planning optimization problems and compared it against alternative approaches for solving robust design and motion planning optimization problems. the next chapter summarizes the results found in this dissertation and presents the conclusions reached during it.

Chapter 8: Conclusions

In this dissertation, techniques were presented for simultaneously optimizing the design and motion of unmanned aerial systems using robust optimization techniques. Chapter 2 presented methods for planning optimal trajectories for unmanned aerial systems while minimizing the risk posed by trajectory and its flight time. Chapter 3 presented a new method for solving robust design optimization problems, which was shown to be capable of solving robust optimization problems containing large number of uncertain parameters. Chapter 4 used the robust optimization framework developed in Chapter 3 to create an approach for solving robust optimal motion planning problems for unmanned systems subject to uncertainty. Chapter 4 also demonstrated the proposed robust motion planning approach by adding 200 uncertain parameters to the risk based motion planning problem considered in Chapter 2. Chapter 5 presented an approach for simultaneously optimizing the design and 2D path of an unmanned aerial vehicle. Chapter 6 extended the approach presented in Chapter 5 to use a motion planning graph, allowing for simultaneous optimization of an unmanned aerial vehicle's design and flight trajectory. Chapter 7 incorporates uncertainty into the problem considered in Chapter 6, by using the robust optimization approach presented in Chapter 3 and also by using the robust motion planning approach from Chapter 4. This chapter provides answers to the research questions asked in Chapter 1 (8.1), summarizes the key contributions of this dissertation (8.2) and details several avenues for future work (8.3).

8.1 Answers to Research Questions

8.1.1 How can optimal motion planning be done for UAV systems when the objective being optimized is not time?

As seen in Chapter 2, sampling based motion planning techniques can be used to optimize more complex objectives, such as the risk objective considered in Chapter 2. It was also observed that it is important to still account for time when optimizing these objectives, as solely optimizing an objective such as risk can lead to impractical solutions (see Figure 2.8). By varying the radius and pitch of the 3-D Dubins curves presented in Chapter 2, it was possible to optimize objectives other than time when connecting two configurations in a motion planning graph.

It was also observed that objective functions other than time can be significantly more expensive to evaluate. Chapter 2 mitigated this issue by presenting a new method for determining the connection radius used by sampling based motion planning techniques. The new method was able to reduce the number of configurations a new configuration would be connected to, rendering the risk-based motion planning problem in Chapter 2 computationally feasible. Additionally, a new method for branching off an initial solution was presented which reduced the number of sampled configurations needed to get a solution with good performance.

However, the higher computational cost of the risk objective considered in Chapter 2 also made it unviable to use an extremely large number of sampled configurations (e.g. 25,000) during motion planning. Lower numbers of sampled configurations lead to small imperfections in the final trajectory found by sampling based motion planning techniques (see Figures 2.9 and 2.10). Additionally, using Dubins curves to optimize objectives other than time does not guarantee and optimal solution to a BVP, as the true optimal solution may not be a constant radius turn.

8.1.2 How can robust optimization problems be efficiently solved when non-convex constraints are present or when the optimization problem considered is typically not solved using mathematical optimization techniques? (Chapter 3, Chapter 4, Chapter 7)

Chapter 3 showed that scenario based methods, particularly those based off random sampling and scenario generation, are effective at solving non-convex robust optimization problems. Methods which only attempt to account for a single worst case scenario are incapable of solving robust optimization problems where multiple worst cases are present (see Sections 3.3.1 and 3.3.4 for examples). However, methods which rely solely on random sampling were also shown to end up with small worst case constraint violations. The SGLRO algorithm proposed in Chapter 3 resolved this problem, by using sampling and worst case based scenario generation in conjunction with a local robust optimization step run after SGLRO finished sampling scenarios.

While SGLRO was shown to efficiently solve a range of non-convex robust optimization problems, including ones with large number of uncertain parameters, it still has several limitations. As discussed in Section 3.1, SGLRO assumes that it is possible to solve a robust optimization problem using a finite number of scenarios. SGLRO has no guarantees of finding a robust optimal solution when this assumption does not hold. Additionally, SGLRO made use of gradient based optimization when performing scenario generation and worst case analysis. While this was practical for the design optimization problems considered in Chapter 3, it is less practical for problems where gradients do not exist or are near zero for small deviations in the uncertain parameters, as is the case in the problem considered in Chapter 4.

In Chapter 4, an approach for robust motion planning was presented and in Chapter 7 an approach for simultaneous robust optimization of both UAV design and motion planning was presented. Both methods use the same structure as the SGLRO algorithm presented in Chapter 3, by sampling scenarios and then updating their current solution whenever a new worst case is found. However, neither method is able to use a local robust optimization step like SGLRO uses.

By formulating a scenario based variant of a problem, a robust optimization approach for that problem can easily be developed using the using randomly sampled scenarios and scenario generation. However, it is more difficult to incorporate a local robust optimization step into such an approach, since problems which are not gradient based optimization problems (such as motion planning) may be more expensive to solve or may lack features allowing for optimization based scenario generation.

8.1.4 How can current methods for optimal sampling based motion planning for unmanned systems be extended to account for robustness with respect to uncertainty?

Chapter 4 showed that it is possible to solve robust motion planning problems by solving the robust transshipment problem on the motion planning graph generated by sampling based motion planning techniques. It was also shown that this approach to robust motion planning allowed for a more general model for how uncertainty affects the cost of a trajectory than assuming the worst case for every single edge present in a motion planning graph. Chapter 4 also showed that it is still possible to control computational cost by using variants on the dual of the robust transshipment problem, which assigns costs to nodes in a similar manner to sampling based motion planning techniques. Several heuristics were also proposed in Chapter 4 to help manage the cost of robust motion planning problems.

However, it was also shown that the computational cost of the robust motion planning problem was still significantly higher than that of a deterministic (ignoring uncertainty) motion planning problem. Curiously, it was also observed that using global optimization techniques (a genetic algorithm) to find the worst case for each edge was both more expensive than the proposed sampling based approach and less effective at finding worst case scenarios. This indicates that a variation on the proposed robust motion planning approach could be the most efficient algorithm for considering the worst case for every edge present in a motion planning problem.

8.1.5 How can the design and motion of an unmanned system be optimized simultaneously?

Chapters 5 and 6 presented approaches for simultaneously optimizing the design, path planning and motion planning of unmanned aerial vehicles. Chapter 5 formulated the vehicle design and path planning (VDPP) problem and showed that unlike classical shortest path planning problems, the VDPP problem does not satisfy the principle of dynamic programming. Thus, exhaustive search methods are required to find optimal solution to VDPP problems. In light of this, Chapter 5 presented several heuristics for solving VDPP problems in addition to an exhaustive branch and bound search method that took advantage of cost-to-go heuristics. Results in Chapter 5 showed that using the proposed exhaustive branch and bound based search was impractical for solving VDPP problems in a computationally feasible manner. However, Chapter 5 also developed a new heuristic algorithm for solving VDPP problems, the vehicle design and path planning algorithm (VDPPA). VDPPA was shown experimentally to find the same solutions as the exhaustive branch and bound based search, while incurring a significantly lower computational cost.

Chapter 6 extended the VDPP problem of Chapter 5 to account for motion planning. This consisted of using sampling based motion planning techniques to construct a motion planning graph, on which a VDPP problem could be solved. It was observed that while this approach was capable of simultaneously optimizing both the design and motion of a UAV, its performance was limited by what trajectories were possible in the motion planning graph generated. A latin hypercube based design of experiments (DOE) was capable of finding a better solution than VDPPA due to the fact that it used a different motion planning graph for every design considered. However, the computational cost of the DOE was also significantly higher than the VDPPA based approach, due to the large number of motion planning problems that the DOE needed to solve.

8.1.6 How can the performance of an unmanned system be optimized with respect to both its design and operation (path planning), while

also being able to account for uncertainty (robust optimization) and the dynamics of the unmanned system (motion planning)?

Chapter 7 formulated a robust optimization variant of the simultaneous design and motion planning optimization problem considered in Chapter 6. The SGLRO algorithm from Chapter 3 was used as the robust optimization algorithm, with a scenario based variation on VDPPA (Chapter 5, Chapter 6) used as the internal scenario robust optimization problem. This approach was capable of finding robust optimal solutions to design and motion planning optimization problems. However, in the example considered in Chapter 7 the deterministic optimal solution was extremely close to the robust optimal solution. It was also observed that the motion planning graph in use still had a significant impact on performance, as the solution using a different motion planning graph found by performing a DOE with Chapter 4's robust motion planning approach was found to have the best worst case computational cost.

The approach used in Chapter 7 for robust design and motion planning optimization was significantly more computationally expensive than the deterministic approach presented in Chapter 6, despite both approaches finding similar solutions. This demonstrates that robust design and motion planning optimization problems are significantly more difficult than design and motion planning problems that ignore uncertainty. The primary causes for this increased computational cost were the scenario robust design optimization problem being more expensive to solve and the additional trajectories searched by VDPPA due to its final solution having a higher cost.

It should be noted that the example problem in Chapter 7 could be considered an easy robust design and motion planning optimization problem, since it only required generating one worst case scenario. More challenging problems would require additional scenarios to be generated, which would increase computational cost significantly.

The DOE using the robust motion planning approach from Chapter 4 was able to find solutions with substantially lower worst case costs than the proposed robust design and motion planning approach. Based off the results obtained, this approach should be considered the most effective method for solving robust design and motion planning optimization problems. Future work should be able to improve on this approach by using the DOE to determine the motion planning graph and initial conditions for a locally optimal robust design and motion planning approach, such as the combination of SGLRO and VDPPA used in Chapter 7.

8.2 Summary of Key Contributions

This section summarizes and lists the key contributions of this dissertation by chapter.

8.2.1 Chapter 2

1. An approach for minimizing the risk posed by UAV trajectories traveling over inhabited areas was presented and demonstrated on an example problem. The new approach is the first risk-based motion planning approach to plan UAV trajectories in a 5D configuration space, which allows the UAV to mitigate risk through maneuvers performed during a flight.
2. A new method was presented for determining 3-D Dubins curves for a fixed wing UAV, which unlike previous approaches, is able to find time optimal 3-D curves and also optimize non-time objectives such as third-party risk.

3. New techniques for reducing the number of BVP problems that need to be solved by the RRT# algorithm during motion planning were presented. Experimental results showed that the new techniques both reduced the computational cost of RRT# and improved the quality of solutions which it was able to find. The new techniques made it computationally feasible to optimize objectives such third-party risk, which are more computationally expensive to evaluate than flight time.

8.2.2 Chapter 3

1. A new approach for solving non-convex robust optimization problems (SGLRO) was presented, which combined a sampling based robust optimization approach using scenario generation with a local robust optimization step for refining the sampling based approach's final solution. The resulting approach can globally search for worst case scenarios like a sampling based robust optimization approach, while avoiding the small constraint violations that are typically present in the solutions found by such approaches.
2. The new approach was shown experimentally to be capable of solving robust optimization problems where existing local robust optimization techniques fail to find solutions feasible under uncertainty. The new approach was also shown to be more efficient than a sampling based approach that lacks a local robust optimization step.

8.2.3 Chapter 4

1. A new approach for solving robust motion planning problems was presented, which considered uncertainty affecting the costs of motions and which allowed for an

arbitrary “black-box” model for how uncertainty affected the costs of motions. The new approach is theoretically capable of finding lower cost solutions relative to assuming that every edges cost is determined by its worst case, through the use of more accurate models for how uncertainty affects motion costs. Experimental results were presented demonstrating that the proposed new approach was able to find more robust solutions than other approaches for robust and determinist motion planning. Additionally, the proposed new approach was found to be both more computationally efficient than this approach and more effective at finding worst case scenarios than an approach that considered the worst-case scenario for each edge in a motion planning graph.

2. A new algorithm for solving robust shortest path planning problems was presented, which uses a sampling based robust optimization approach with several strategies for minimizing the number of edge costs which need to be computed under different uncertain parameter values. The new approach uses a robust transshipment problem in conjunction with a new “bidirectional” dual to the robust transshipment problem, which allows it to consider a “black-box” model for uncertain edge costs in a computationally efficient manner.

8.2.4 Chapter 5

1. A new cost-to-go heuristic was developed which can be used to provide lower bounds on the costs of incomplete paths in design and path planning optimization problems.

2. Several new algorithms were presented for solving design and path planning optimization problems where the path planning problem considered is on a graph. No previous works in the literature have considered the problem from this standpoint, which avoids the need for using global optimization techniques or a discrete design domain. A new algorithm, VDPPA, was shown to be capable of finding similar solutions to exhaustive search approaches, at significantly reduced computational cost.

8.2.4 Chapter 6

1 A new approach for solving design and motion planning optimization problems was presented, which leverages the VDPPA algorithm developed in Chapter 5, by applying it to the motion planning graph generated using sampling based motion planning methods. The new approach treats its design optimization problem as a gradient based optimization problem, which makes it more computationally efficient than the current global search based methods for design and motion planning optimization. However, results show that the larger number of motion planning graphs searched by methods which globally search the design space can lead to a solution with a lower cost trajectory than is possible in the single motion planning graph used by the proposed approach.

8.2.6 Chapter 7

1. Several methods for solving robust design and motion planning optimization were presented. The problem of robust design and motion planning optimization has not been previously considered in the literature. Experimental results showed that one

of the most effective methods for solving robust design and motion planning optimization problems was to conduct a DOE using a robust motion planning approach.

8.3 Future Work

This section identifies several promising areas for future work related to motion planning, design and motion planning optimization and robust optimization variants of these problems.

8.3.1 Trajectory refinement methods for UAV motion planning with non-time objectives

The performance of a number of the results in this dissertation were limited by the fact that trajectories were planned using a single motion planning graph. This is particularly noticeable in Chapter 6's example problem, where one of the designs from the 100 element DOE outperforms the solution found by VDPPA due it's motion planning graph having a lower cost trajectory present. A common solution to this issue in motion planning problems is to significantly increase the number of sampled configurations used to construct the motion planning graph. However, this would massively increase the number of edges in a motion planning graph, which is not practical for problems with expensive to evaluate objective functions (e.g. the risk objective in Chapter 2), or which involve more computationally expensive operations such as design optimization.

Typically, trajectories computed using sampling based motion planning methods are smoothed using trajectory optimization methods before a vehicle actually executes them. This can remove small imperfections in the asymptotically optimal trajectories found by sampling based motion planning techniques. Such methods have been used previously in grid based 2D risk based path planning problems (e.g. the greedy and local methods in [103]) and could be extended to work in higher dimensional configuration spaces. Because trajectory optimization methods are also a type of optimization problem, they could easily be integrated within equations such as Eq. (5.1) and used directly within a design and path planning optimization problem.

8.3.2 Multi-Objective robust optimization using scenario generation

While Section 3 only discussed feasibility robust optimization (uncertainty only appearing in the constraints), uncertainty in an objective function ($f(x,u)$ instead of $f(x)$) can be dealt with by moving the objective function into the constraints (see Section 3.1 of [14] or Chapter 4 for examples). This concept has also been extended to multi-objective robust optimization (MORO) [46]. As presented, the SGLRO algorithm in Chapter 3 cannot be used for solving MORO problems, as MORO requires accounting for a set of designs (which trade-off between objectives) rather than just one design. Future work will explore methods for using scenario based approaches to solve MORO problems.

8.3.2 Enabling real time cost robust motion planning

While the robust motion planning approach presented in Chapter 4 was the fastest of the two robust motion planning approaches considered, its performance was still significantly higher than that of deterministic motion planning. Developing stronger bounds on the costs of both nodes and edges could help further reduce this cost.

However, part of the reason that so many edge costs need to be evaluated is because the robust transshipment problem requires all costs to be computed before it can be solved. A search based method (e.g. VDPPA or Chapter 5's branch and bound approach) could bypass this issue, as such methods only need to compute costs for the paths that they actually search. However, many search based methods (e.g. A*, VDPPA) are only heuristics for the robust path planning problem in Chapter 4, since it does not satisfy the principle of dynamic programming. However, the robust transshipment problem is already solved using a branch and bound method (Gurobi), however that branch and bound method uses LP relaxations, unlike the approach in Chapter 5 which directly branches on paths. It is likely that a specialized branch and bound (or branch and cut) algorithm can be developed that computes edge costs as it encounters them. This could eliminate the need to initially compute the costs of edges that aren't part of the current solution in a sampling based robust motion planning approach.

From a practical standpoint, performing cost robust motion planning in real time will always require the development of a specialized solver for the robust transshipment problem. Commercial mixed integer programming solvers (e.g. Gurobi [48]) typically have a number of overhead operations meant to assist them in solving many different types of MILP problems. Developing a specialized solver for robust transshipment problems would facilitate running the proposed robust motion planning approach at speeds practical for real time operations.

An alternate strategy for performing cost robust motion planning in real time is to consider the worst cost for each and accept the loss in performance associated with doing so. From a theoretical standpoint, this strategy still satisfies the principle of dynamic programming, meaning that search based methods for real time replanning (e.g. D* [118], RRT^{X} [88]) should be able to efficiently solve the resulting motion planning problem. Thus, it should be possible to solve a robust motion planning problem which ignores correlations between edge costs while computing significantly less edge costs than a robust motion planning approach that considers correlations between edge costs. Chapter 4 showed that sampling based robust optimization methods are relatively efficient at finding worst case scenarios in robust motion planning problems when considering correlations between edge costs. This indicates that it may be possible to develop an extremely efficient sampling based robust motion planning approach that ignores correlations between edges by leveraging the framework and strategies developed in Chapter 4 with a motion planning designed for real time replanning (e.g. RRT^{X} [88]).

8.3.3 Efficient methods for solving robust design and motion planning optimization problems

The results from Chapter 7 indicate that while it is possible to solve robust design and motion planning optimization problems, they are also significantly more expensive than their deterministic (no uncertainty) equivalents. The primary causes of this additional cost are the fact that a robust design optimization problem is more expensive than its deterministic variant and the fact that search based approaches (e.g. VDPPA) need to search additional solutions due to the higher costs caused by considering worst case performance. Using a discrete set of designs can bypass these issues (e.g. the latin hypercube DOE in Chapter 7), however it still incurs a similarly large computational cost. If more efficient methods were developed for solving robust motion planning problems, it could become more efficient to search through a discrete set of designs when solving robust design and motion planning optimization problems than using SGLRO to solve a scenario robust design and motion planning problem.

Alternately, a less computationally expensive search algorithm than VDPPA could be developed. VDPPA spends a significant amount of its computational time computing the costs of partial candidate solutions (PCS) when solving robust design and motion planning problems. This occurs because VDPPA needs to search more PCS (due to its final solution being more expensive), which incurs a large computational cost. A potential heuristic strategy for resolving this issue could be to either compute PCS costs using a fixed set of designs or to use the solutions from the cost-to-go heuristic in place of PCSs. However, further computational experiments are needed to determine how these heuristic strategies would impact the costs of the final solutions found by VDPPA.

8.3.4 Alternate forms of uncertainty which are incompatible with the approaches presented

This dissertation focused on robust optimization that accounts for uncertainty affecting the costs of solutions. Two other important forms of uncertainty affecting the motions of unmanned systems are uncertainty which affects the feasibility of motions and uncertainty about which motions an unmanned system will need to perform.

8.3.4.1 Uncertainty affecting the feasibility of motions

Uncertain conditions can render an unmanned system's trajectory infeasible, thus it is important to plan trajectories for unmanned systems that will not suffer failures. From a robust optimization perspective, if a trajectory segment is infeasible under uncertainty, then that trajectory segment should not be used. For robust motion planning this can be easily implemented, as this consists of removing any edges from the motion planning graph which are found be infeasible under a scenario. However, in a robust design and motion planning optimization problem, it is possible that the feasibility of some edges may depend on the design of the system. This issue could be resolved by converting this robust feasibility problem into an unconstrained problem, by adding any constraints subject to uncertainty to the objective function using a penalty method. Alternately, the constraints in question could be added into the design optimization problem within VDPPA (Eq. (5.1)) whenever the path in Eq. (5.1) contains the edges in question.

8.3.4.2 Uncertainty about which motions an unmanned system will need to perform

The uncertainty model in this dissertation assumed that the start and end configurations of any path and motion planning problem considered are known and are unaffected by uncertain parameters. However, it is possible that the start and end configurations may be uncertain or belonging to a range (which can be modeled as an uncertain parameter). The effect of this is that a range of possible trajectories now need to be planned, instead of just one. This type of problem is particularly important when optimizing the design of an unmanned system, as it can be used to optimize the performance of an unmanned system across the range of motions that it needs to be able to execute. For example, the design of a robotic manipulator might be optimized alongside how it moves between points in its workspace, or an unmanned aerial vehicle’s design might be optimized for the range of flight maneuvers that it needs to be capable of.

This uncertainty model is incompatible with search based approaches such as VDPPA, which only operate between a single start and end point. A possible strategy for solving problems with this type uncertainty present could be to formulate a scenario robust optimization problem, where the cost is constrained by a constraint function $f(x,u)$ representing a motion planner, where x is the system's design and u contains the start and end configurations the motion planner will use. However, $f(x,u)$ would not be an analytical function, making it difficult to use with an algorithm like SGLRO. The resulting problem could technically be solved by only relying on random sampling. However, it would also be computationally expensive since relying solely on random sampling will increase the number of scenarios generated and thus the number of motion planning problems which need to be solved during each iteration of design optimization.

This issue could be bypassed by employing a surrogate model based optimization approach, such as efficient global optimization [61], which could approximate the cost of moving between two configurations using a specific design. For a deterministic design and motion planning problem, this is similar in structure to the approach of Baykal and Alterovitz [9], except that the surrogate replaces the inner motion planning problem, which should significantly improve computational performance. While (to the author's best knowledge) no methods have been developed for directly solving robust optimization problems using efficient global optimization, similar approaches have seen success in solving robust optimization problems involving multiple disciplines (e.g. [56], [73]). This strategy may also show promise in dealing with the cost uncertainty model considered in this dissertation. However, it is limited by the scalability of surrogate modeling techniques, which may encounter scalability issues when dealing with large numbers of uncertain parameters (e.g. the 200 uncertain parameters considered in Chapter 4 and Chapter 7).

Appendix A: Modeling Unmanned Aerial System (UAS) Risks via Monte Carlo Simulation

This appendix has also appeared as [106] at the 2019 International Conference on Unmanned Aerial Systems.

A.1 Overview

Unmanned Aerial Systems (UAS) pose a variety of risks to third parties when operating over populated areas, due to the danger posed if the UAS crashes. Two commonly used metrics for assessing the risk of such crashes are the kinetic energy of the UAS at the time of impact and the probability distribution of locations where the UAS could crash. In this appendix, a Monte Carlo based approach is presented for simulating UAS crashes in order to calculate these metrics. A surrogate modeling approach for UAS safety metrics is also presented, which is built using the results of the Monte Carlo simulations. The surrogate modeling approach is capable of rapidly evaluating UAS safety metrics for arbitrary UAS design and operating parameters. The proposed approach is demonstrated by modeling the kinetic energies at time of impact and crash probability distributions for UAS with dynamics models similar to that of a Cessna 182.

A.2 Introduction

As Unmanned Aerial Systems (UASs) are increasingly adopted across numerous domains, assessing the safety of operating UASs over people is becoming crucial for increasing public acceptance of such operations. This includes assessing and mitigating the risk posed by the UAS if it were to crash in a populated area. Performing this assessment requires understanding where the UAS is likely to crash after a failure and its kinetic energy when it crashes into the ground. Determining these quantities empirically is impractical, as it would require repeatedly crashing a UAS. Predicting these quantities is also challenging because the dynamics model of a UAS is a non-linear model (like that of any aircraft) and is sensitive to its initial conditions. This makes it difficult to estimate the risks posed by a UAS crash analytically without simplifying the dynamics model or ignoring the dynamics in an overly conservative risk assessment.

This appendix presents a simulation-based approach that can generate results and surrogate models to support assessing the safety of UAS operations over populated areas in practical settings. The approach is based on a Monte Carlo simulation of a flight dynamics model that simulates the trajectories of a UAS crashing from different initial flight states. These trajectories produce a probability distribution of the crash location, where the UAS reaches the ground, and can also be used to estimate the UAS's kinetic energy when it impacts the ground. The approach performs simulation experiments to generate the distributions for different values of UAS design and operating parameters. These distributions provide insights into how these parameters affect the Crash location Probability Distribution (CPD) and kinetic energy. Finally, the approach generates surrogate models from these results that can be used to quickly estimate the CPD and kinetic energy for a given set of parameter values. This appendix also discusses the results and surrogate models generated by using the proposed approach to model UAS with dynamics similar to a Cessna 182 [101]

The remainder of this appendix is organized as follows. Section A.3 reviews previous work on assessing UAS safety and positions this appendix with respect to related work.

Section A.4 presents the proposed Monte Carlo simulation-based approach and the approach for generating surrogate models. Section A.5 applies the proposed Monte Carlo simulation approach to perform a parametric study involving design and operating parameters for UAS with similar dynamics to a Cessna 182. Section A.6 presents a statistical analysis of the results from Section A.5 to determine which parameters cause statistically significant effects on the safety of the example UAS model. Section A.7 tests the performance of the surrogate models built using the proposed approach. Section A.8 summarizes the results presented.

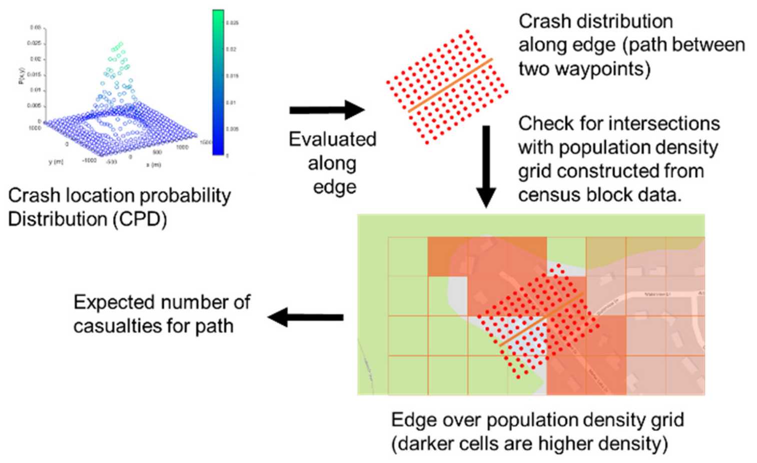

Crash location probability Distribution (CPD)
Evaluated along edge
Crash distribution along edge (path between two waypoints)
Check for intersections with population density grid constructed from census block data.
Expected number of casualties for path
Edge over population density grid (darker cells are higher density)

Figure A.11. Example of how a Crash Probability Distribution (CPD) is used to assess the expected number of fatalities of a UAS operation

A.3 Related work

Multiple methods [2] and frameworks [3] have been proposed for assessing UAS safety. One commonly used quantity in UAS risk assessment is the expected number of fatalities associated with a UAS operation, which requires assessing the probability of failure during an operation and estimating the number of persons struck by a UAS if it were to crash. This estimate requires a CPD (see Figure A.1) to determine how many people could be affected by a UAS crash. The expected fatalities caused by a UAS flight is especially relevant for analyzing the safety of a large UAS, which is likely to cause a fatality if it strikes someone due its high mass and velocity [22]. Approaches for planning a UAS's path to mitigate this risk [103, 65, 95] assess the expected number of fatalities by moving the CPD along a flight path, as depicted in Figure A.1.

There are several ways in which CPDs are constructed. Some approaches [95, 41, 126, 21, 2, 76]] use analytical models to determine the area of all possible locations which a UAS could reach before crashing and then assume that it is equally likely that UAS crashes in any of these locations. La Cour Harbo [66], La Cour Harbo and Schioler [67], and Primatesta et al. [95] used the distance traveled to develop a non-uniform CPD model and also consider the effects of wind. Lum et al. [75] used Monte Carlo simulation to determine the distance traveled by a UAS before crashing, which is used to fit an analytical CPD model. Rudnick-Cohen et al. [103] used Monte Carlo simulation of a UAS crashing to construct the CPD out of the binned frequency distribution of the locations of the simulated crashes.

Prior work has explored how UAS CPDs are affected by either different operating parameters or different design configurations. Wu and Clothier [126] analytically determined the shapes of the regions of all possible crash locations for failures occurring at different altitudes. Rudnick-Cohen et al. [108] constructed a Delaunay triangulation from the results of Monte Carlo simulations to model the effects of design parameters on a CPD to perform design optimization. Haartsen et al. [50] conducted a parameter study on how the operating parameters such as flight speed, altitude, and the vehicle's roll angle affect the CPD of fixed wing and rotorcraft UAS.

Another quantity used to assess UAS safety is the impact kinetic energy of the UAS when it crashes into the ground, which determines whether a UAS crash can cause fatalities. The U.S. Federal Aviation Administration (FAA) has proposed preliminary requirements for small UAS operating over people that require the kinetic energy at time of UAS impact be below either 11 or 25 ft-lbs, depending on where the UAS is being flown [123]. These limits on kinetic energy originated from a U.S. Army Range Commander Council specification [22, 124]. The kinetic energy depends upon the mass and the velocity of the UAS. Estimating the UAS's velocity as it crashes requires a model of the UAS's flight dynamics, which are affected by the UAS's physical design and its flight state at time of failure. This makes it difficult to directly compute the kinetic energy of a UAS at the time it crashes into the ground.

The approach presented in this chapter expands upon the Monte Carlo approach introduced in [103] with the following extensions: (1) an integrated simulation-based approach for estimating the CPD and kinetic energy of a crashing UAS; (2) insights into how changes to design and operating parameters affect the CPD and kinetic energy; and (3) surrogate models that can be used to quickly estimate these safety metrics for UAS.

A.4 Monte Carlo Simulation Of UAS crashes and Surrogate Modeling of UAS safety metrics

A.4.1 Simulation Model

Table A.1 defines the variables in our model of the dynamics of an air vehicle in steady state flight, which can be described using the ordinary differential equations (ODEs) in

Table A.2 [101]. By solving these ODEs for different initial conditions, different crash trajectories will be obtained. By changing these initial conditions and performing Monte Carlo simulation, a range of potential crash trajectories will be generated. Note that the forces and moments present in Table A.1 will have different values and equations depending on the type of UAS being modeled. This can require modeling additional state variables to those in Table A.1, such as control surfaces (fixed wing UAS) or rotors (multicopter UAS).

In order to keep the Monte Carlo simulations computationally tractable, we ignored any effects on drag coefficients caused by angular velocities (P,Q,R) and the time derivatives of the UAS's angle of attack and sideslip angle. We found that this does not significantly affect the location where the UAS crashes, but it provides a large reduction in the time needed to simulate the UAS crashing.

TABLE A.1: Variables used in Table A.2

U,V,W	Velocities (Body Frame) (m/s)
θ,ϕ,ψ	Orientation (Euler angles) (rad)
P,Q,R	Angular velocities (Body Frame) (rad/s)
p_N, p_E, h	Position of vehicle (NED Frame) (m)
F_x, F_y, F_z	Drag forces on vehicle (Body Frame) (N)
$\bar{L}, M, N$	Drag moments on vehicle (Body Frame) (N)
$c_1, c_{2,} \ldots, c_9$	Moment of inertia constants (kg/m^2)
g	Gravitational constant (m/s^2)

TABLE A.2: Flight dynamics equations for Monte Carlo crash simulations, overdots denote time derivatives

$$\dot{U} = RV - QW - g\sin(\theta) + \frac{F_x}{m}$$

$$\dot{V} = -RU + PW + g\sin(\phi)\cos(\theta) + \frac{F_y}{m}$$

$$\dot{W} = QU - PV + g\cos(\phi)\cos(\theta) + \frac{F_z}{m}$$

$$\dot{\phi} = P + \tan(\theta)(Q\sin(\theta) + R\cos(\phi))$$

$$\dot{\theta} = Q\cos(\phi) - R\sin(\phi)$$

$$\dot{\psi} = \frac{Q\cos(\phi) + R\sin(\phi)}{\cos(\theta)}$$

$$\dot{P} = (c_1R + c_2P)Q + c_3\overline{L} + c_4N$$

$$\dot{Q} = c_5PR - c_6(P^2 - R^2) + c_7M$$

$$\dot{R} = (c_8P - c_2R)Q + c_4\overline{L} + c_9N$$

$$\dot{p}_N = U\cos(\theta)\cos(\psi) + V(-\cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)$$
$$\cos(\psi)) + W(\sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\psi))$$

$$\dot{p}_E = U\cos(\theta)\sin(\psi) + V(\cos(\phi)\cos(\psi) + \sin(\phi)\sin(\theta)$$
$$\cos(\psi)) + W(-\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi))$$

$$\dot{h} = U\sin(\theta) - V\sin(\phi)\cos(\theta) - W\cos(\phi)\cos(\theta)$$

A.4.2 Monte Carlo Approach

Let $x(t)$ be the state of the UAS as a function of time t, and let $\vec{x}^r$ be the initial state. Let $f(x)$ be the equation of the flight dynamics such that $\dot{x} = f(x(t))$. Let X^d be the range of the random perturbations. Let t_{fi} be the initial amount of integration time the ODE is solved for. Let $\mathrm{ME}(x)$ be a function that returns the total mechanical energy (kinetic energy plus potential energy) of state x. Let N be the number of iterations. Table A.3 details the algorithm used to perform the Monte Carlo simulation. Although the flight dynamics model $f(x)$ is deterministic, each iteration of the simulation uses randomly perturbed initial conditions (see Table A.1II, steps 1.a and 1.b). The equations in Table A.1 can be solved for specific initial conditions using an ODE solver (see Table A.1II, step 1.d.ii). However, most efficient ODE solvers (e.g., MATLAB's ode45 [112]) make use of adaptive step sizes, which complicates identifying the exact time and location at which a vehicle crashes. Additionally, the amount of time which the UAS remains in the air before crashing could vary significantly. To account for this, we ran the ODE solver for a fixed amount of integration time before checking for a crash (see Table A.1II, step 1.d). If the vehicle's altitude is still above the ground, then the final solution from the previous ODE solution is used as the new initial conditions and the ODE solver's integration time is set to be the total amount of integration time used (see Table A.1II, step 1.d.i). Repeating this process allows for using an appropriate amount of integration time as needed during the crash simulation. When the UAS's altitude is below the ground, we extract the first time in the ODE solution where the UAS is under the ground and use the state at that time as the state when the UAS crashed (see Table A.1II, step 1.e).

The ODE solver also needs to be stopped if an event that changes the system's dynamics occurs. For example, if a UAS with unpowered (free to move while crashing) control surfaces is simulated, the ODE solver needs to be stopped any time the control surfaces hit a hard stop, as the dynamics of the UAS will have changed. In this case the ODE solver is restarted with the state at the time which the event occurs and the integration time for the next iteration is set as it normally would be. This ensures that the crash simulation avoids any instability that could be caused by the UAS's dynamics changing. From a practical standpoint, an event should be generated once the UAS's altitude reaches a fixed height beneath the ground. This allows the ODE solver to be stopped early, reducing computational time. However, the event should not be generated when the UAS's altitude is zero, as this would interfere with ODE solvers that move forwards and backwards in time (e.g., MATLAB's ode45 [112]).

TABLE A.3: Monte Carlo Crash simulation approach

```
1. While $N > 0$
    a. Let $\vec{x}^d$ be a random perturbation drawn from $X^d$
    b. $\vec{x}^i = \vec{x}^r + \vec{x}^d$
    c. $t_i = 0, t_f = t_{fi}, t_{end} = 0$
    d. While $\vec{x}^i$ corresponds to a state that hasn't crashed
        i. Set $t_f = t_{end} + t_f, t_i = t_{end}$
        ii. Solve the ODE system $\dot{x} = f(x(t))$ on time interval $[t_i,\ t_f]$ , with initial conditions $x(t_i) = \vec{x}^i$ , let $\vec{x}^f$ be the final state in the solution and let $t_{end}$ be the time the solver stopped at. The solver should stop before reaching $t_f$ if an event occurs
        iii. Set $\vec{x}^i = \vec{x}^f$
    e. Let $t_{crash}$ the first time in the solution for $x(t)$ where the vehicles height is less than zero. $\vec{x}_{crash} = x(t_{crash})$
    f. If $\mathrm{ME}(\vec{x}^f) < \mathrm{ME}(\vec{x}^r + \vec{x}^d)$
        i. Store $\vec{x}_{crash}$ in the list of valid Monte Carlo simulation results
    g. Set $N = N - 1$
2. Return the list of valid Monte Carlo simulation results and finish
```

Because the Monte Carlo simulation randomly perturbs the initial conditions of the UAS, many different trajectories are generated for a single initial state. The CPD generated from this initial state is the 2-D binned histogram of the locations where the UAS reached the ground. The maximum kinetic energy at time of impact is determined from the greatest impact velocity observed over the *N* iterations.

However, care needs to be taken to account for degenerate results, which are caused by initial conditions for which the ODE solver is unstable. Fortunately, such cases are easy to identify, as the final kinetic energy of the UAS will often increase to a quantity larger than the initial mechanical energy in the system. Although the initial velocities of the UAS are typically variables perturbed in the Monte Carlo simulation, this check (Table A.1II, step 1.f) can be done against the unperturbed velocity, since drag forces should cause a significant loss in the UAS's mechanical energy.

A.4.3 Using parameter studies to construct surrogate models

In order to investigate the effects of various UAS design or operation parameters, the Monte Carlo approach from Section III.B can be run for multiple parameter configurations. A large-scale parameter study with an appropriate DOE (DOE), such as a Full Factorial or Latin Hypercube design, will generate enough CPDs such that CPDs for other parameter configurations can be interpolated from them.

A k-Nearest Neighbors (KNN) model can be used to interpolate the results of multiple Monte Carlo simulations to obtain the kinetic energy at time of crashing and the CPD of a UAS for a queried set of UAS parameters. The KNN model performs an inverse distance weighted average of the Monte Carlo simulation results for the *k* design points (parameter configurations in the DOE) nearest to the queried parameters, where the distance $d(x, y)$ is determined by the weighted distance measure given in Eq. (1). In this distance measure, *w* denotes the weight for each parameter, *x* denotes the input queried parameters, and *y* denotes the parameters of one of the design points.

$$\begin{aligned} d(x,y) = {} & w_{height}(x_{height} - y_{height})^2 + w_{roll}(x_{roll} - y_{roll})^2 + \\ & w_{pitch}(x_{pitch} - y_{pitch})^2 + w_{speed}(x_{speed} - y_{speed})^2 + \\ & w_{wingspan}(x_{wingspan} - y_{wingspan})^2 + \\ & w_{rudder}(x_{rudder} - y_{rudder})^2 + w_{mass}(x_{mass} - y_{mass})^2 \end{aligned} \tag{A.1}$$

Eq. (A.1) is used with an inverse distance weighting [113] in order to combine together the results from the k-nearest neighbors to the queried parameters. The inverse distance weighting exponent *u*, the weights w and the number of neighbors *k* should be separately determined for each UAS safety metric being modeled. The performance of the KNN model is tuned by adjusting the weights *w*, the number of neighbors *k* and the inverse distance exponent *u*.

A.5 Example: UAS based off Cessna 182

To demonstrate the Monte Carlo crash simulation approach, we tested it on a UAS with the dynamics of a Cessna 182 suffering from a total loss of power, meaning that no systems on the UAS would be actuated during the crash. The Cessna 182's dynamics model was chosen because its aerodynamic coefficients are publicly available [101]. A 250 element Latin Hypercube DOE (DOE) was created using MATLAB [81] and Monte Carlo simulations were run for each of the 250 experiments in the DOE. 7 parameters were used for the DOE, the UAS's height or elevation from the ground level, roll, pitch and forwards velocity at time of failure and its wingspan, allowable rudder movement range and mass. The ranges of these parameters are listed in Table A.4.

A.5.1 Cessna 182 based UAS dynamics model including unactuated control surfaces

The UAS dynamics considered were those of a Cessna 182 with unactuated control surfaces, which were modeled as simple masses with a mass of 1 kg subject to the drag forces caused by the control surfaces. The Cessna 182 model has three types of control surfaces, ailerons, elevators, and a rudder. We assume that the elevator control surfaces move together and that the aileron control surfaces always move in opposite directions to each other. Each control surface has a maximum movement range; a constraint in the ODE dynamics model prevents it moving outside this range. If a control surface reaches this limit, the ODE solver is stopped and restarted to account for the change in the dynamics.

TABLE A.4: Parameter ranges used for Latin Hypercube DOE

	Height (m)	Roll	Pitch	Speed (m/s)	Wingspan (m)	Rudder Range	Mass (kg)
Min	512	-45°	-15°	30	7.45	±24°	887
Max	2024	45°	15°	70	19.47	±40°	1400

A.5.2 Parameter settings for example

The random perturbations used in the Monte Carlo simulations are detailed in Table A.5. The values for U (speed), h (height), Φ (roll) and Θ (pitch) were perturbed from the values specified for each Monte Carlo simulation by the DOE. All other parameters detailed in Table A.5 were perturbed from a value of zero. Perturbations were applied in both positive and negative directions.

The Monte Carlo simulations used $t_{fi} = 0.25$ seconds, additionally a minimum time step of 10^{-5} seconds was imposed on the ODE solver used. Any time the ODE solver's timestep went below 10^{-5} seconds, an event was generated to stop the ODE solver, to avoid wasting computational time in degenerate cases. At each design point, we ran $N = 10,000$ crash simulations, so a total of 2.5 million crash simulations were performed for the entire DOE considered. The ODE solver used was MATLAB's ode45 [112] with absolute and relative tolerances set at 10^{-3}. An event to stop the ODE solver was generated whenever the UAS fell more than 100 meters beneath the ground (altitude of 0 meter). Additionally, to avoid wasting computational time in cases where the ODE solver became unstable, a time limit of 30 seconds was imposed for each iteration (Step 1, Table A.3) of the Monte Carlo simulations.

TABLE A.5: INITIAL CONDITIONS FOR MONTE CARLO SIMULATIONS

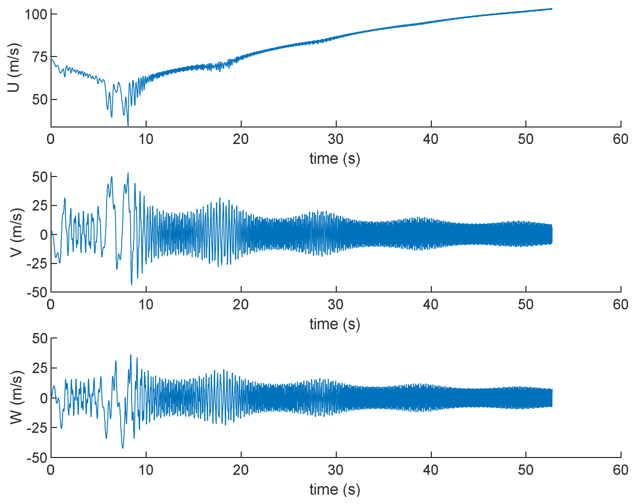
Velocity (m/s)	Perturbation
U	10
V	10
W	10
Position (m)	
p_N	0
p_E	0
h	0
Orientation, Euler angles (degrees)	
Φ (Roll)	11.25
Θ (Pitch)	11.25
Ψ (Yaw)	11.25
Angular Velocity (degrees/s)	
P	11.25
Q	11.25
R	11.25
Control surface deflection (degrees)	
Elevator Deflection (δ_E)	11.25
Rudder Deflection (δ_R)	11.25
Aileron Deflection (δ_A)	11.25
Control surface deflection rates (degrees/s)	
Elevator deflection rate ($\dot{\delta}_E$)	0

Rudder deflection rate ($\dot{\delta}_R$)	0
Aileron deflection rate ($\dot{\delta}_A$)	0

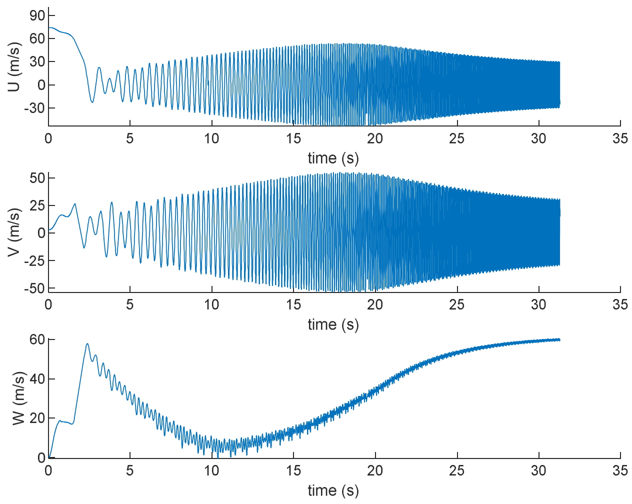
A.5.3 Results from example

Based on the kinetic energy at the time of crashing, the crash trajectories produced by the Monte Carlo simulations can be grouped into two different types. Figure A.2 shows the history of the UAS's velocities for examples of these two types of trajectories. Note that the high frequency of oscillations in Figure A.2 occurs because the model ignores rate terms for angular rate; moreover, Figure A.2 depicts body frame velocity. The speed of the vehicle does not oscillate at the magnitudes or frequencies of its components depicted in Figure A.2. Trajectory (a) corresponds to the UAS crashing by flying along a stable trajectory to the ground, as evidenced by the lack of oscillations in *U*, the body frame velocity component moving directly against the UAS's drag. Trajectory (b) corresponds to the UAS crashing by falling, as indicated by lack of oscillations in W, the velocity component moving directly against the UAS's lift. Because fixed wing aircraft (such as the Cessna 182) are designed to have high L/D (lift over drag) ratios, Trajectory (a) incurs significantly less loss in speed that Trajectory (b), causing Trajectory (a) to have a much higher kinetic energy when it crashes into the ground. This implies that design changes that increase drag (such as increasing wingspan) should also provide reductions in the kinetic energy of a fixed wing UAS when it crashes.

Figure A.3 and Table A.6 describe several of the CPDs that we generated and their associated max kinetic energy at time of crash for different sets of design and operating parameters. Figure A.3 shows several expected trends: failures that occur at higher altitudes lead to a larger and more spread out CPD and higher speeds moved the centroid of the CPD forwards (a, b) and caused a more spread out CPD (a, b, d, f). The roll of the UAS produced a bias towards the direction of the banked turn that the roll would cause (see d). Negative pitch angles (b, c) shifted the CPD's centroid away from the point where the UAS starts crashing, while positive and near horizontal pitch angles (a, e) moved the centroid the other way. Low masses increased the size of the CPD (e, f), which blurred some of the trends present in CPDs with higher mass (a, b, c, d). The CPDs with the lowest kinetic energy at the time of crashing (c, e) correlated to either a low height before crashing (c) or a low mass (e). Apart from mass's already known effect on kinetic energy, the height, speed, and wingspan appear to affect the UAS's kinetic energy when it crashes.



(a)



(b)

Figure A.2. Velocity history from two trajectories from the same Monte Carlo simulation with similar initial speeds and the same design parameters and initial height.

(a) Initial Speed: 73.7 m/s, Final Kinetic Energy at time of crash: 5.65 Megajoules

(b) Initial Speed: 74.1 m/s, Final Kinetic Energy at time of crash: 2.37 Megajoules

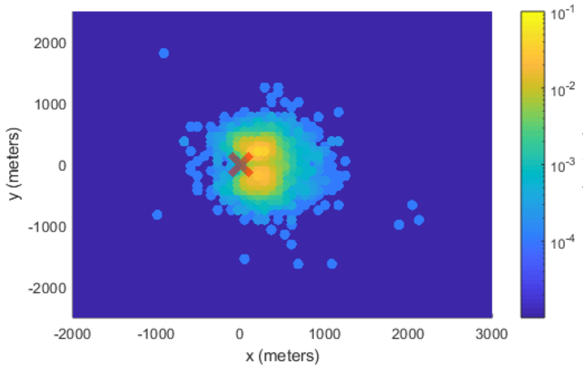
However, the results presented in Figures 2 and 3 and Table A.6 are only a snapshot of the 250 Monte Carlo simulations run for different configurations of design and operating parameters. In order to draw more accurate conclusions about how these parameters affect the safety of a UAS, a statistical analysis is needed that can be used for analyzing the Monte Carlo simulation results.

A.6 Statistical analysis of Monte Carlo results

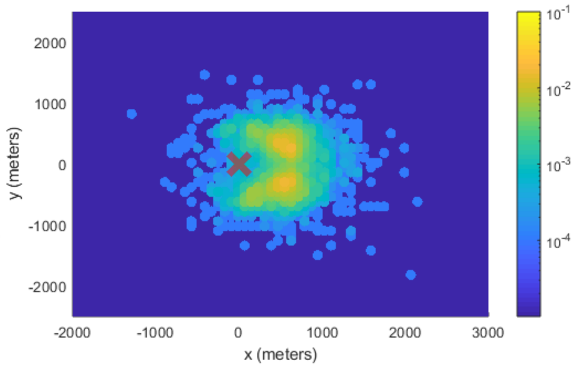
Statistical tests were performed on the Monte Carlo results to verify that the parameters varied affected the CPD and the kinetic energy at time of crash of the example UAS.

A.6.1 Kinetic energy at time of crash

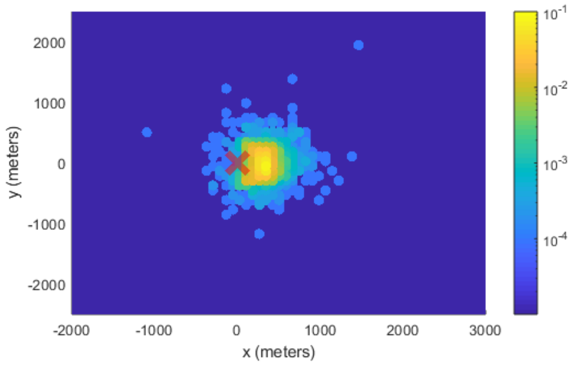
Because a UAS's kinetic energy at the time of crashing is a single value it was possible to test which parameters caused statistically significant effects on it by binning the data based on parameter values. For all possible pairs of input variables, analysis of variance (ANOVA) tests were conducted, to test for first and second order correlation effects between the parameters. Three equally spaced bins for each parameter were used to bin the maximum kinetic energy from each of the 250 Monte Carlo simulations. Table A.7 provides the p-values from these tests, entries between the same variable denote first order correlation effects being present. The p-values in Table A.7 are the probability that the null hypothesis that there are no differences between the bins being compared is true. From the analysis in Table A.7, we see that Height, Speed, Wingspan and Vehicle Mass can reject this null hypothesis (at a significance level of 0.05) and have statistically significant effects on the kinetic energy of the example UAS. Analytically this makes sense, height, speed and mass affect the initial potential and kinetic energy of the UAS before it begins crashing and wingspan affects the lift and drag on the UAS while it crashes.



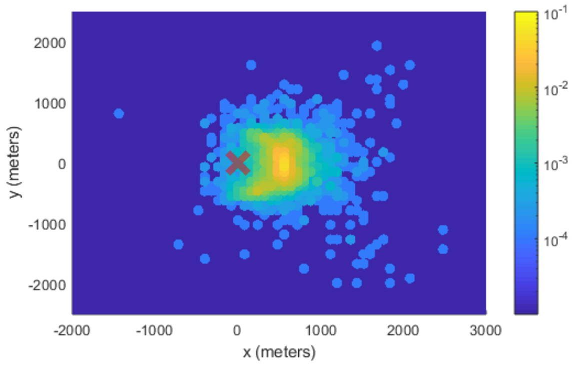
(a)



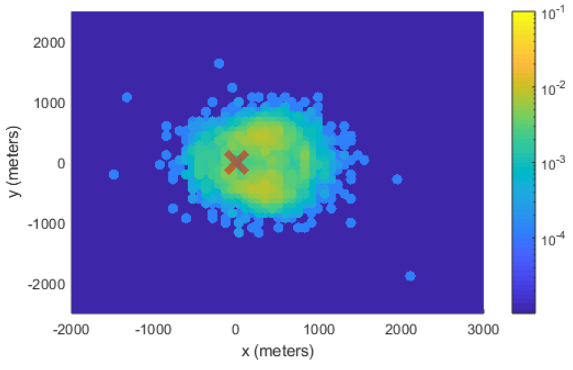
(b)



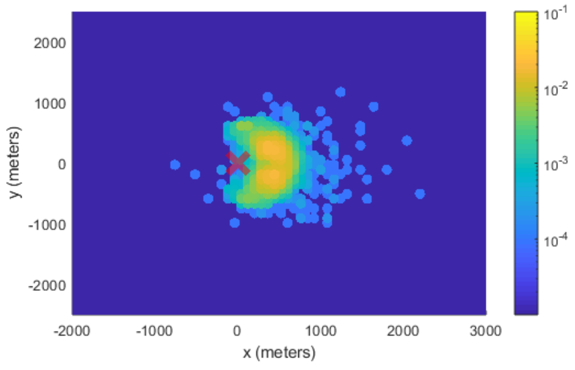
(c)



(d)



(e)



(f)

Figure A.3: Selected CPDs generated from Monte Carlo simulations in the example, see Table A.6 for parameters. The red X shows the point where the UAS begins crashing, the vehicle's velocity prior to crashing is in the same direction as the x-axis.

TABLE A.6: Kinetic Energy at time of crash and parameters of selected example Monte Carlo simulations

Label	Kinetic Energy at time of crash (MJ)	Height (m)	Roll (deg.)	Pitch (deg.)	Speed (m/s)	Wingspan (m)	Rudder Range (deg.)	UAS Mass (kg)
(a)	15.6	1,898	26.8	0.89	37.5	15.1	36.0	1,356
(b)	27.0	1,969	-18.8	-8.84	60.6	12.9	33.5	1,378
(c)	8.54	658.7	-23.5	-9.25	41.7	16.8	31.6	1,166
(d)	14.5	871.2	-37.9	2.69	66.7	14.88	34.6	1,350
(e)	6.89	1,987	3.77	13.6	66.1	15.5	32.5	897.8
(f)	12.2	1,578	6.74	-1.82	42.3	7.52	37.5	995.0

A.6.2 Crash Probability Distribution

Because a CPD is a probability distribution, we analyzed the CPDs that we generated by calculating the sample means, covariances, coskewness [83] and cokurtosis [83]. The sample mean provides a statistic of where the centroid of the distribution is. The sample covariance provides a statistic describing the width and shape of the distribution. The sample coskewness provides a statistic describing the directional bias of the crash distribution. The sample cokurtosis provides a statistic capturing how spread out the distribution is. These four multivariate statistics were computed for each of the 250 CPDs generated. Note that the sample cokurtosis and coskewness varied noticeably between the 250 crash distributions, meaning that it would be inaccurate to use multivariate normal distributions to approximate the CPDs.

For all possible pairs of input variables, multiple analysis of variance (MANOVA) tests were conducted, in order to test for first and second order correlation effects between the input variables. The tests were conducted with a p-value of 5%, three equally spaced bins for each input variable were used and the output variables were the statistics computed for each of the 250 crash distributions. The results from these tests showed that there were statistically significantly differences between all possible pairs of input variables, indicating that there were statistically significant differences in the CPDs caused by the parameters considered. However, this analysis provides no information about how these parameters affect the CPDs.

A correlation analysis was performed to gain a basic understanding of how the input parameters affect the shape of the crash distribution. Correlation coefficients [51] were computed for the input variables relative to the computed statistics, using MATLAB's [81] corrcoeff function. The computed correlation coefficients were tested against the null hypothesis of the coefficients occurring due to random chance, using the corrcoeff function [81]. The probabilities of a correlation being present were identified between the input variables and the statistics computed for crash distributions, which are displayed in Table A.8.

Table A.7: Statistical significance (p- values) of combinations of parameters affecting the UAS's kinetic energy at time of crashing, computed using ANOVA. Correlations between a parameter and itself indicate the parameter has a statistically significant effect when considered on its own. Bolded p-values have statistically significant effects on kinetic energy.

	Height	Roll	Pitch	Speed	Wingspan	Rudder Range	Vehicle Mass
Height	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Roll	**0.00**	0.57	0.24	0.08	**0.02**	0.58	**0.00**
Pitch	**0.00**	0.24	0.08	**0.01**	**0.00**	0.29	**0.00**
Speed	**0.00**	0.08	**0.01**	**0.00**	**0.00**	**0.01**	**0.00**
Wingspan	**0.00**	**0.02**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Rudder Range	**0.00**	0.58	0.29	**0.01**	**0.00**	0.15	**0.00**
Mass	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**

From the parameters affecting the mean we see that the centroid of the CPD in the direction the UAS travels in is affected by pitch, speed, wingspan and mass. The shift in the CPD's centroid perpendicular to this direction is affected by roll and wingspan.

From the parameters affecting the variance statistics, we see that the size of the CPD is affected by height, pitch and speed, though pitch only affects the width of the CPD. Additionally roll affects the orientation of the CPD.

From the parameters affecting skewness, we see that the overall bias of the CPD in the direction the UAS travels in is affected by height, wingspan and mass. However, the bias perpendicular to this direction is unaffected by the parameters considered in this study. Speed, wingspan and mass affect the directionality of the bias of the CPD.

From the parameters affecting kurtosis we see that similar parameters affect the spread of the CPD as those that affect the size of the CPD. However, rudder range affects the spread of the CPD, but not its size.

Table A.8: p-values for null hypothesis that no correlation exists between input variables and distribution statistics. Bolded p-values are statistically significant (rejecting the null hypothesis), indicating that the input variable affects the distribution statistic in question.

Statistic	Mean		Covariance			Coskewness				Cokurtosis				
Variable	X	Y	X	Y	XY	X	Y	XXY	XYY	X	Y	XXXY	XXYY	XYYY
Height	0.33	0.70	**0.00**	**0.00**	0.79	**0.00**	0.74	0.27	0.20	**0.00**	**0.00**	0.97	**0.00**	0.89
Roll	0.06	**0.00**	0.62	0.53	**0.00**	0.21	0.31	0.34	0.67	**0.03**	0.75	**0.01**	0.40	**0.02**
Pitch	**0.00**	0.83	1.00	**0.02**	0.67	0.66	0.45	0.98	0.44	0.08	**0.00**	0.20	**0.00**	0.31
Speed	**0.00**	0.34	**0.00**	**0.00**	0.31	0.72	0.41	0.35	**0.00**	**0.00**	0.13	0.24	**0.01**	0.15
Wing-span	**0.00**	**0.03**	**0.00**	0.09	0.23	**0.00**	0.08	**0.02**	**0.00**	**0.01**	**0.00**	0.62	0.61	0.72
Rudder Range	0.09	0.46	0.32	0.70	1.00	0.32	0.57	0.21	0.46	**0.00**	**0.00**	0.38	**0.00**	0.82
Mass	**0.00**	0.72	**0.00**	0.27	0.15	**0.04**	0.11	0.09	**0.05**	0.07	0.11	0.71	0.20	0.61

Although the statistics in Tables VII and VIII are useful for understanding which aspects of a UAS's CPD and kinetic energy at time of crashing are affected by parameters, they do not allow for actually predicting the CPD of the UAS for a given set of parameters. Additionally, they are computed using a large number of computationally expensive Monte Carlo simulations. Having a CPD model that can be evaluated quickly is important when assessing safety during UAS operations or when UAS safety is being maximized using a numerical optimization method. To enable this type of analysis, we used the proposed approach to construct surrogate models for the CPD and kinetic energy of a UAS that could be evaluated quickly.

A.7 Surrogate modeling UAS safety using a k-nearest neighbors (KNN) model

A.7.1 Testing UAS surrogate models

In order to test the surrogate models developed, a test set of 50 random sets of parameters were drawn from a uniform distribution and the Monte Carlo simulations were run for these parameters. The weights, number of neighbors and inverse weighting exponent for the KNN were determined empirically without considering the test data set. The test set was not used as input data to the KNN, thus it could be used to assess the accuracy of the KNN model by comparing the results from the KNN model against those in the test set.

A.7.2 Modeling kinetic energy at time of crashing

The statistical analysis of Section V showed that the kinetic energy at time of crashing for the example UAS model was only affected by height, speed, wingspan and mass. Thus $w_{roll} = w_{pitch} = w_{rudder} = 0$. Additionally, we know part of the relation between mass and the kinetic energy at time of crashing ($1/2mv^2$), thus we use the KNN model to approximate the speed (v^2) of the UAS at the time of crashing and then compute the kinetic energy as $1/2mv^2$.

Empirically, it was found that $w_{height} =$ 0.71, $w_{speed} =$ 0.24, $w_{wingspan} = 0.99$, $w_{mass} = 0.86$, $u =$ 1.7, $k = 13$ yielded an acceptable KNN model. With these parameters, the KNN had a root mean squared error (RMSE) of 2.0 Megajoules on the test dataset, with a maximum error magnitude of 4.15 Megajoules. The highest maximum kinetic energy at time of crashing in the test data set was 26 Megajoules, the lowest maximum was 5.6 Megajoules.

A.7.3 Modeling the UAS's CPD

Because all parameters showed evidence earlier of affecting the CPD of the example UAS, they were all considered in the KNN model for the CPD. When modeling the CPD, we separate the CPD into a 2-D binned frequency distribution of a fixed size and the centroid of the distribution. This made it easier for the KNN to model effects that changed the shape of the CPD.

Table A.9: Quality of fit of KNN CPD model in terms of scaled root mean square error (RMSE) and scale maximum magnitude of error (Max Err.). Values scaled based off range of statistics present in the Monte Carlo simulations from Section V

Statistic	Mean		Covariance			Coskewness				Cokurtosis				
	X	Y	X	Y	XY	X	Y	XXY	XYY	X	Y	XXXY	XXYY	XYYY
RMSE	0.11	0.10	0.18	0.19	0.10	0.12	0.15	0.12	0.15	0.13	0.19	0.08	0.15	0.1
Max Err.	0.27	0.3	0.42	1.1	0.45	0.40	0.45	0.38	0.34	0.56	0.76	0.21	0.67	0.42

Empirically, it was found that $w_{height} = 0.2$, $w_{roll} = 0.76$, $w_{pitch} = 0.34$, $w_{speed} = 0.97$, $w_{wingspan} = 0.77$, $w_{rudder} = 0.19$, $w_{mass} = 0.17$, $u = 1.50$, $k = 2$ yielded an acceptable KNN model. Table A.1X details the RMSE and maximum error observed in each of the statistics discussed earlier between the CPDs produced by the KNN and those computed from the results of the test set.

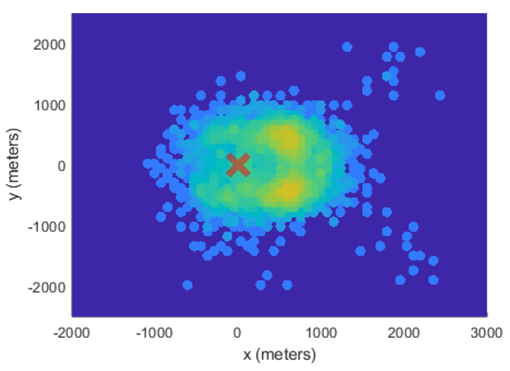
A.7.4 Computational Time

The KNN models for both kinetic energy and the CPD required no more than 2 milliseconds to generate a result. In comparison the times needed to run one of the Monte Carlo simulations to generate a CPD in the test data set ranged from 1.8 to 32 hours.

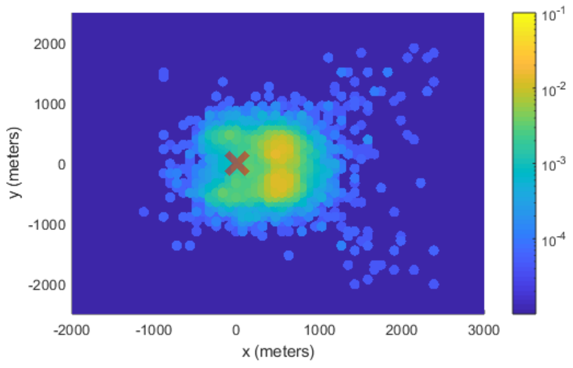
A.7.5 Discussion

Figure A.4 shows a comparison of several CPDs in the test set against the CPDs produced by the KNN model for the same parameters (Table A.10). The CPD KNN model's performance is more difficult to quantify, while its RMSE is low for the statistics considered, the maximum errors for several statistics are significantly higher. Part of this can be attributed to the KNN model's tendency to approximate larger CPDs than the CPD produced via Monte Carlo simulation, as seen in Figure A.4b. Additionally, the KNN model sometimes misses behavior that may not have been present in the CPDs from the parameter study, which the KNN uses, such as the behavior in Figure A.4c. However, the KNN also obtains slightly smoother CPDs that the Monte Carlo simulations, as seen in Figure A.4b. This can lead to behavior where the KNN model misses a part of the CPD, like in Figure A.4a.

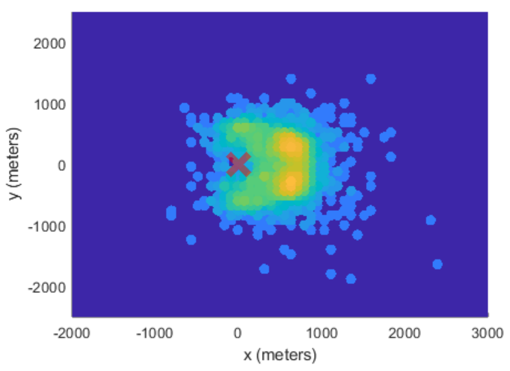
However, the KNN model still produces CPDs that are visually close to the more accurate CPDs produced using Monte Carlo simulation, while using six orders of magnitude less computational time than a Monte Carlo simulation.



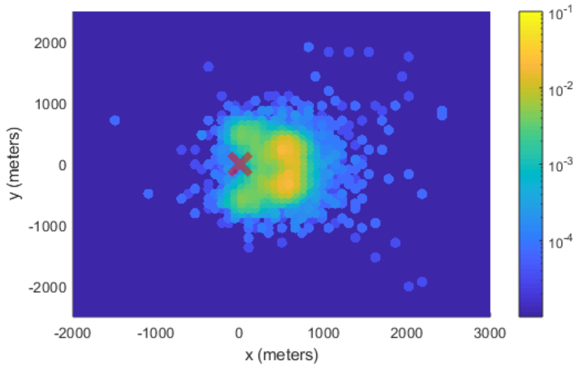
(a)



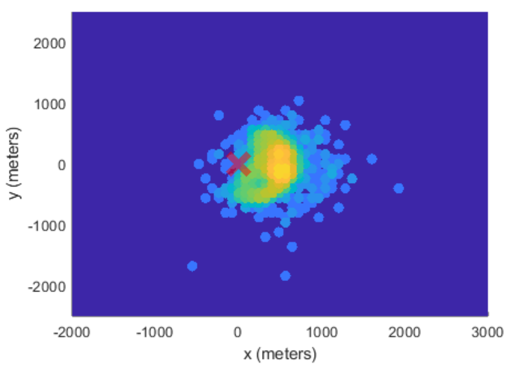
(b)



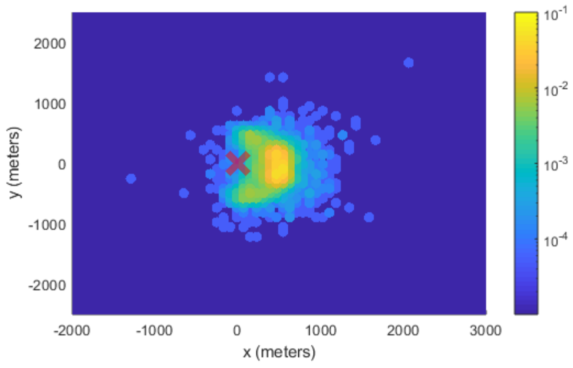
(c)



(d)



(e)



(f)

Figure A.4: Left (a, c, e): CPDs produced by the KNN model. Right (b, d, f): CPDs produced via Monte Carlo simulation

Table X: parameters of selected examples from test set used for comparison

Label	Height (m)	Roll (deg.)	Pitch (deg.)	Speed (m/s)	Wingspan (m)	Rudder Range (deg.)	UAS Mass (kg)
(a)	1,806	-19.3	1.83	69.8	12.2	36.0	898.2
(b)	1,307	-0.48	4.97	63.7	8.7	37.6	928.4
(c)	872.5	-40.5	-13.4	52.6	7.56	38.5	891.3

A.8 Conclusions

This appendix presented an approach for assessing UAS safety metrics by using Monte Carlo simulation to simulate the UAS crashing. A surrogate modeling approach was also presented, which can use the results of the Monte Carlo simulations to rapidly estimate UAS safety metrics for different UAS parameters. These approaches were demonstrated on an example UAS model based off a Cessna 182 in order to estimate two safety metrics for different design and operating parameters, the kinetic energy when impacting the ground and the UAS's CPD. A statistical analysis was conducted to assess the effects of these parameters on the two safety metrics. Results were presented comparing the results from the surrogate modeling approach against the Monte Carlo simulation approach for the two safety metrics considered.

The results indicate that the proposed approach was more accurate at modeling kinetic energy than it was at modeling CPDs. This can be partially attributed to the fact that the CPD KNN model used a k value of 2, which generates CPDs using only two reference CPDs. However, the performance of the CPD KNN could not be improved by increasing the k value. This indicates that there may more effective surrogate approaches for estimating CPDs than a KNN model, or that increasing the number of experiments in the DOE would have been beneficial. The k value could have been set higher had a Full Factorial DOE been used instead of a Latin Hypercube DOE. However, a Full Factorial

DOE requires an exponential number of experiments relative to the number of parameters being varied, which is only computationally feasible when considering a small number of parameters.

The KNN models were several orders of magnitude faster than the Monte Carlo simulations, which is an acceptable tradeoff for the loss in accuracy relative to them. Critically, the KNN models are fast enough that they could be potentially evaluated in real time. Such models can facilitate UAS performing online risk management and make it computationally feasible to optimize UAS safety. Thus, the proposed modeling approach shows promise for enabling new approaches to managing UAS safety.

Appendix B: Computational Complexity of SGLRO and Other Robust Optimization Approaches

Table B.1 details the computational complexity of each of the steps within SGLRO in terms of the total number of constraint function calls (total number of times that any of the constraints $d_l(x)$, $g_i(x,u)$ and $q_j(u)$ are evaluated). It is assumed that all optimization solvers use a central difference method to estimate derivatives and that the BFGS algorithm [6] is used to estimate the Hessian of $f(x)$ for both the local robust optimization method and any optimization solvers which make use of Hessians (e.g. MATLAB's fmincon [80]), as it is more efficient than numerically computing the Hessian at every iteration via finite differences. Each entry in Table B.1 is computed assuming that the step in question occurs on every iteration of SGLRO, which is why all steps except "Local Robust Optimization" are multiplied by N_I. "Reduced Scenario Robust Optimization" requires at most $N_\alpha \times D \times (I \times N_I + L)$ function calls, as Eq. (3.1) has at most $I \times N_I + L$ constraints (if a scenario is generated on every single iteration), which require *D* function calls to evaluate the gradient of, to a maximum of N_α times. A similar expression exists for "Worst Case Search" , except using *P* instead of *D* and *J* instead of *L*, however the maximum number of constraints used during "Worst Case Search" will never increase. The cost of "Local Robust Optimization" is the sum of the costs of "Worst Case Search" and "Reduced Scenario Robust Optimization", except that N_Q (the number of iterations "Local Robust Optimization" needs to converge) replaces N_I.

Table B.1: Breakdown of Computational costs in SGLRO

Step	Upper bound on number of function calls
Reduced Scenario Robust Optimization	$O(N_I^2 \times N_\alpha \times D \times I + N_I \times N_\alpha \times D \times L))$
Worst Case Search	$O(N_I \times N_\alpha \times P \times I \times J \times N_S)$
Feasibility checking (lines 8-13 of Algorithm 1)	$O(N_I \times I)$
Local Robust Optimization	$O(N_Q \times N_\alpha \times P \times I \times J + N_Q^2 \times N_\alpha \times D \times I + N_Q \times N_\alpha \times D \times L)$

The term $N_\Omega = N_I + N_Q$ can be used to represent the total number of iterations used by SGLRO, which simplifies its worst case computational cost to the expression given in Table B.2, which is the sum of the terms in Table B.1. Table B.2 also provides a comparison of SGLRO's computational cost against a basic deterministic double loop approach (see Appendix C for implementation) and SGR^{2}O [104]. N_S is the maximum limit on the number of scenarios used by SGR^{2}O.

Table B.2: Computational Costs of Methods Compared

Approach	Theoretical worst case computational cost
SGR^{2}O	$O(N_\Omega \times N_\alpha \times (N_S \times I + L) \times D + N_\Omega \times N_\alpha \times (N_S \times I + J) \times P)$
Deterministic Double Loop	$O(N_\Omega \times N_\alpha \times P \times I \times J + N_\Omega \times N_\alpha \times D \times I + N_\Omega \times N_\alpha \times D \times L))$
SGLRO	$O(N_\Omega^2 \times N_\alpha \times P \times I \times J + N_\Omega^2 \times N_\alpha \times D \times I + N_\Omega \times N_\alpha \times D \times L)$

From a theoretical standpoint, both SGR^{2}O [104] and a deterministic double loop approach should be faster than SGLRO, as SGLRO has N_Ω^2 terms present. SGR^{2}O appears faster because SGR^{2}O uses scenario reduction to limit the maximum number of scenarios in use, which changes the cost of solving Eq. (3.2) or Eq. (3.3) to be $O(N_I \times N_S \times N_\alpha \times D \times I + N_I \times N_\alpha \times D \times L)$. The deterministic double loop optimization approach only considers one scenario per constraint, which provides a similar benefit.

However, there exist robust optimization problems where the robust optimal solution cannot be found by only considering one scenario per constraint. Additionally, the use of scenario reduction may require additional scenarios to be generated, which can result in SGR^{2}O requiring more constraint function calls than SGLRO.

Appendix C: Deterministic Double Loop Robust Optimization Method

Table C.1: Deterministic Double Loop Robust Optimization Algorithm

```
1:  $Feasible = False$
2:  $\overline{U} \leftarrow \{u_{nom}\}$
3:  $R(u_{nom}) \leftarrow \{1,...,I\}$
4:  While($Feasible = False$)
5:      $x_B \leftarrow$ Solve RSRO
6:      $Feasible = True$
7:      $\overline{U} \leftarrow \{\}$
8:      For($i \in \{1...I\}$)
9:          $u_{gen} \leftarrow$ Solve Worst Case Search
                with $V = i$
10:         $\overline{U} \leftarrow \overline{U} \cup \{u_{gen}\}$
11:         $R(u_{gen}) \leftarrow i$
12:         If ($g_i(x_B, u_{gen}) \geq \varepsilon$)
13:             $Feasible = False$
14: Return $x_B$
```

Appendix D: Nomenclature and Glossary

ABREVIATIONS AND DEFINITIONS

Backward subpath:	The subpath of a path from a specified node to the goal node
BVP	Boundary Value Problem
CCS	Complete Candidate Solution: A solution that contains both the start and goal node in its path
Configuration	A set of values which belong the configuration space, a space which describes all feasible states a system can be in
CPD	Crash Probability Distribution
DOE	Design of experiments
Forward subpath:	The subpath of a path from the start node to a specified node
LP	Linear Programming
PCS	Partial Candidate Solution: A solution with a path that does not contain the goal node
Priority queue [62]:	A queue in which its first element has the lowest cost
Scenario	A scenario assigns a value to all uncertain parameters present in a problem
SGLRO	Scenario Generation with Local Robust Optimization

SGR^{2}O	Scenario Generation and Reduction Robust Optimization, robust optimization approach of Rudnick-Cohen et al. [104]
Subpath:	A sequence of nodes connected by edges within a path

NOTATION

$B(c_1, c_2)$	The set of all possible solutions to the BVP between configurations c_1 and c_2.
c_s	The vehicle's initial configuration
c_f	The vehicle's desired final configuration
D	Number of design variables
$d_l(x)$	lth constraint without uncertainty
e_{ij}	An edge going from node i to node j
$f(c_1, c_2, u)$	Objective function of Chapter 4. Outputs the cost of moving from configuration c_1 to configuration c_2 under scenario u
$f_t(s)$	The time needed to move along trajectory s
$f_r(s)$	The risk caused by moving along trajectory s
$f(x)$	Objective function
$g_i(x,u)$	ith constraint subject to uncertainty

I, J, L	Number of constraints on design containing uncertainty, on the domain of uncertain parameters and on design not containing uncertainty, respectively
L_L	Length of a curve
N_α	Maximum number of optimization solver iterations
N_I	Number of iterations to run SGLRO algorithm
N_Q	Number of iterations used by a local robust optimization method
P	Number of uncertain parameters
p_n	Probability that modified RRT# algorithm uses "SAMPLE" procedure
p_b	Probability that modified RRT# algorithm uses "RANDOM CONFIG IN BALL" procedure
$q_j(u)$	*j*th constraint defining the domain of uncertain parameters
$R(u)$	The set of the indices of the constraints which *u* should impose in a reduced scenario robust optimization problem
s	A trajectory or path, defined by a continuous sequence of configurations
u	Vector of all uncertain parameters present in optimization problem
$\overline{U}$	Set of scenarios used to solve scenario robust optimization problem
U	Set of all possible combinations of uncertain parameters (domain of uncertain parameters)

V	Set of violated constraints
w_r	Weighting coefficient for risk objective
w_t	Weighting coefficient for time objective
x	Vector of all design variables present in optimization problem
x_B	Current best solution for design variables
x_{ij}	Design variable in a transshipment problem which takes a value of 1 if edge e_{ij} is a part of the current optimal path and 0 otherwise
$Z(n)$	Set of nodes neighboring node *n*
$Z_{inc}(i)$	Set of nodes neighboring node *i* with edges going into node *i*
$Z_{out}(i)$	Set of nodes neighboring node *i* with edges leaving from node *i*
ε	User specified constraint tolerance
γ_0	Initial connection radius of RRT#
γ_{min}	Minimum connection radius of RRT#

References

[1] Mq-4c triton. Online, August 2018. Available at http://www.navair.navy.mil/index.cfm?fuseaction=home.displayPlatform&key=C86C2E45-74DD-42A8-94E4-9238D184237F.

[2] R Aalmoes, YS Cheung, E Sunil, JM Hoekstra, and F Bussink. A conceptual third party risk model for personal and unmanned aerial vehicles. In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pages 1301–1309. IEEE, 2015.

[3] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993. Google-Books-ID: WnZRAAAAMAAJ.

[4] Juan J Alonso, Patrick LeGresley, and Vctor Pereyra. Áircraft design optimization. *Mathematics and Computers in Simulation*, 79(6):1948–1958, 2009.

[5] Ron Alterovitz, Thierry Siméon, and Ken Goldberg. The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty.

[6] Jasbir Singh Arora. *Introduction to Optimum Design*. Elsevier, 2004.

[7] Oktay Arslan and Panagiotis Tsiotras. Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2421–2428. IEEE, 2013.

[8] Oktay Arslan and Panagiotis Tsiotras. Dynamic programming guided exploration for sampling-based motion planning algorithms. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4819–4826. IEEE, 2015.

[9] Cenk Baykal and Ron Alterovitz. Asymptotically optimal design of piecewise cylindrical robots using motion planning. In *Robotics: Science and Systems 2017*, 2017.

[10] Kostas E Bekris and Lydia E Kavraki. Greedy but safe replanning under kinodynamic constraints. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 704–710. IEEE, 2007.

[11] Aharon Ben-Tal and Arkadi Nemirovski. Robust optimization–methodology and applications. *Mathematical Programming*, 92(3):453–480, 2002.

[12] Dimitri P Bertsekas. Robust shortest path planning and semicontractive dynamic programming. *Naval Research Logistics (NRL)*, 2016.

[13] Dimitri P Bertsekas and John N Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

[14] Dimitris Bertsimas, David B Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM review*, 53(3):464–501, 2011.

[15] Dimitris Bertsimas and Omid Nohadani. Robust optimization with simulated annealing. *Journal of Global Optimization*, 48(2):323–334, 2010.

[16] Dimitris Bertsimas, Omid Nohadani, and Kwong Meng Teo. Nonconvex robust optimization for problems with constraints. *INFORMS Journal on Computing*, 22(1):44–58, 2010.

[17] Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical programming*, 98(1):49–71, 2003.

[18] Hans-Georg Beyer and Bernhard Sendhoff. Robust optimization–a comprehensive survey. *Computer methods in applied mechanics and engineering*, 196(33-34):3190–3218, 2007.

[19] Lars Blackmore, Hui Li, and Brian Williams. A probabilistic approach to optimal robust path planning with obstacles. In *American Control Conference, 2006*, pages 7–pp. IEEE, 2006.

[20] Justin M Bradley and Ella M Atkins. Optimization and control of cyber-physical vehicle systems. *Sensors*, 15(9):23020–23049, 2015.

[21] Jeff Breunig and Shereef Sayed. Modeling ground collision severity of small unmanned aircraft systems. In *2018 Aviation Technology, Integration, and Operations Conference*, page 3349, 2018.

[22] David Alexander Burke. *System Level Airworthiness Tool: A Comprehensive Approach to Small Unmanned Aircraft System Airworthiness.* PhD thesis, North Carolina State University, 2010.

[23] Giuseppe C Calafiore. Repetitive scenario design. *IEEE Transactions on Automatic Control*, 62(3):1125–1137, 2017.

[24] Giuseppe C Calafiore and Marco C Campi. The scenario approach to robust control design. *IEEE Transactions on Automatic Control*, 51(5):742–753, 2006.

[25] Giuseppe Carlo Calafiore. Random convex programs. *SIAM Journal on Optimization*, 20(6):3427–3464, 2010.

[26] Diah Chaerani, Cornelis Roos, and A Aman. The robust shortest path problem by means of robust linear optimization. In *Operations Research Proceedings 2004*, pages 335–342. Springer, 2005.

[27] Mohammadreza Chamanbaz, Fabrizio Dabbene, Roberto Tempo, Venkatakrishnan Venkataramanan, and Qing-Guo Wang. Sequential randomized algorithms for convex

optimization in the presence of uncertainty. *IEEE Transactions on Automatic Control*, 61(9):2565–2571, 2016.

[28] Shuo Cheng and Mian Li. Robust optimization using hybrid differential evolution and sequential quadratic programming. uses the max approach for constraints, but when used inside a global optimization approach this is similar to sampling. *Engineering Optimization*, 47(1):87–106, 2015.

[29] Hamidreza Chitsaz and Steven M LaValle. Time-optimal paths for a dubins airplane. In *Decision and Control, 2007 46th IEEE Conference on*, pages 2379–2384. IEEE, 2007.

[30] Sanjiban Choudhury, Jonathan D Gammell, Timothy D Barfoot, Siddhartha S Srinivasa, and Sebastian Scherer. Regionally accelerated batch informed trees (rabit*): A framework to integrate local information into optimal path planning. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 4207–4214. IEEE, 2016.

[31] Sanjiban Choudhury, Sebastian Scherer, and Sanjiv Singh. Rrt*-ar: Sampling-based alternate routes planning with applications to autonomous emergency landing of a helicopter. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3947–3952. IEEE, 2013.

[32] Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable test problems for evolutionary multiobjective optimization. In *Evolutionary multiobjective optimization*, pages 105–145. Springer, 2005.

[33] Laurent Denarie, Kevin Molloy, Marc Vaisset, Thierry Siméon, and Juan Cortés. Combining system design and path planning. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.

[34] Pedro FA Di Donato and Ella M Atkins. Evaluating risk to people and property for aircraft emergency landing planning. *Journal of Aerospace Information Systems*, pages 259–278, 2017.

[35] Pedro FA Di Donato and Ella M Atkins. Three-dimensional dubins path generation and following for a uas glider. In *Unmanned Aircraft Systems (ICUAS), 2017 International Conference on*, pages 294–303. IEEE, 2017.

[36] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[37] Xiaoping Du and Wei Chen. Towards a better understanding of modeling feasibility robustness in engineering design. *Journal of Mechanical Design*, 122(4):385–394, 2000.

[38] Xiaoping Du and Wei Chen. Sequential optimization and reliability assessment method for efficient probabilistic design. *Journal of Mechanical Design*, 126(2):225–233, 2004.

[39] Noel E Du Toit and Joel W Burdick. Robot motion planning in dynamic, uncertain environments. *IEEE Transactions on Robotics*, 28(1):101–115, 2012.

[40] YY Fan, RE Kalaba, and JE Moore. Shortest paths in stochastic networks with correlated link costs. *Computers & Mathematics with Applications*, 49(9):1549–1564, 2005.

[41] Andrew Ford and Kevin McEntee. Assessment of the risk to ground population due to an unmanned aircraft in-flight failure. In *10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, page 9056, 2010.

[42] Thierry Fraichard and Hajime Asama. Inevitable collision statesâ€”a step towards safer robots? *Advanced Robotics*, 18(10):1001–1024, 2004.

[43] Gowtham Garimella, Matthew Sheckells, Joseph Moore, and Marin Kobilarov. Robust obstacle avoidance using tube nmpc.

[44] Emile Glorieux, Pasquale Franciosa, and Darek Ceglarek. End-effector design optimisation and multi-robot motion planning for handling compliant parts. *Structural and Multidisciplinary Optimization*, 57(3):1377–1390, 2018.

[45] Chad Goerzen, Zhaodan Kong, and Bernard Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *Journal of Intelligent and Robotic Systems*, 57(1-4):65–100, 2010.

[46] S Gunawan and S Azarm. Multi-objective robust optimization using a sensitivity region concept. *Structural and Multidisciplinary Optimization*, 29(1):50–60, 2005.

[47] Subroto Gunawan. *Parameter sensitivity measures for single objective, multi-objective, and feasibility robust design optimization*. PhD thesis, Univeristy of Maryland, 2004.

[48] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2016.

[49] Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. Computational co-optimization of design parameters and motion trajectories for robotic systems. *The International Journal of Robotics Research*, page 0278364918771172, 2018.

[50] Y Haartsen, R Aalmoes, and YS Cheung. Simulation of unmanned aerial vehicles in the determination of accident locations. In *Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*, pages 993–1002. IEEE, 2016.

[51] Wolfgang Härdle and Léopold Simar. *Applied multivariate statistical analysis*, volume 22007. Springer, 2007.

[52] Timothy F Hogan, Ming Liu, James A Ridout, Melinda S Peng, Timothy R Whitcomb, Benjamin C Ruston, Carolyn A Reynolds, Stephen D Eckermann, Jon R Moskaitis, Nancy L Baker, et al. The navy global environmental model. *Oceanography*, 27(3):116–125, 2014.

[53] Majid Hosseini, Mehran Nosratollahi, and Hossein Sadati. Multidisciplinary design optimization of UAV under uncertainty. *Journal of Aerospace Technology and Management*, 9(2):169–178, 2017.

[54] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255, 2002.

[55] TC Hu, Andrew B Kahng, and Gabriel Robins. Optimal robust path planning in general environments. *IEEE Transactions on Robotics and Automation*, 9(6):775–784, 1993.

[56] Weiwei Hu, Shapour Azarm, and Ali Almansoori. New approximation assisted multi-objective collaborative robust optimization (new aa-mcro) under interval uncertainty. *Structural and Multidisciplinary Optimization*, 47(1):19–35, 2013.

[57] He Huang and Song Gao. Optimal paths in dynamic networks with dependent random link travel times. *Transportation Research Part B: Methodological*, 46(5):579–598, 2012.

[58] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International journal of robotics research*, 34(7):883–921, 2015.

[59] Raphael Haftka Jaroslaw Sobieszczanski-Sobieski. Multidisciplinary design optimization: an emerging new engineering discipline. In *Advances in Structural Optimization*, pages 483–496. Springer, 1995.

[60] Zhaowang Ji, Yong Seog Kim, and Anthony Chen. Multi-objective α-reliable path finding in stochastic networks with correlated link costs: A simulation-based multi-objective genetic algorithm approach (smoga). *Expert Systems with Applications*, 38(3):1515–1528, 2011.

[61] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[62] Douglas W Jones. An empirical comparison of priority-queue and event-set implementations. *Communications of the ACM*, 29(4):300–311, 1986.

[63] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[64] Lydia Kavraki, Petr Svestka, and Mark H Overmars. *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, volume 1994. Unknown Publisher, 1994.

[65] Anders La Cour-Harbo. Quantifying risk of ground impact fatalities of power line inspection bvlos flight with small unmanned aircraft. In *Unmanned Aircraft Systems (ICUAS), 2017 International Conference on*, pages 1352–1360. IEEE, 2017.

[66] Anders la Cour-Harbo. Quantifying risk of ground impact fatalities for small unmanned aircraft. *Journal of Intelligent & Robotic Systems*, pages 1–18, 2018.

[67] Anders La Cour-Harbo and H Schioler. Ground impact probability distribution for small unmanned aircraft in ballistic descent. *Reliability engineering and system safety*, 2017.

[68] Steven M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[69] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

[70] Dongoo Lee and Jaemyung Ahn. Integrated optimization of planetary rover layout and exploration routes. *Engineering Optimization*, 50(1):164–182, 2018.

[71] Mian Li, Shapour Azarm, and Vikrant Aute. A multi-objective genetic algorithm for robust design optimization. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 771–778. ACM, 2005.

[72] Mian Li, Shapour Azarm, and Art Boyars. A new deterministic approach using sensitivity region measures for multi-objective robust and feasibility robust design optimization. *Journal of mechanical design*, 128(4):874–883, 2006.

[73] Wei Li, Mi Xiao, and Liang Gao. Improved collaboration pursuing method for multidisciplinary robust design optimization. *Structural and Multidisciplinary Optimization*, 59(6):1949–1968, 2019.

[74] Jinghong Liang, Zissimos P Mourelatos, and Jian Tu. A single-loop method for reliability-based design optimization. In *ASME 2004 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 419–430. American Society of Mechanical Engineers, 2004.

[75] Christopher Lum, Kristoffer Gauksheim, Chris Deseure, Juris Vagners, and Tad McGeer. Assessing and estimating risk of operating unmanned aerial systems in populated areas. In *11th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference, including the AIAA Balloon Systems Conference and 19th AIAA Lighter-Than*, page 6918, 2011.

[76] Christopher W Lum and Dai A Tsukada. Uas reliability and risk analysis. *Encyclopedia of Aerospace Engineering*, pages 1–12, 2016.

[77] Øyvind Magnussen, Geir Hovland, and Morten Ottestad. Multicopter uav design optimization. In *Mechatronic and Embedded Systems and Applications (MESA), 2014 IEEE/ASME 10th International Conference on*, pages 1–6. IEEE, 2014.

[78] Anirudha Majumdar and Russ Tedrake. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research*, 36(8):947–982, 2017.

[79] Kostas Margellos, Paul Goulart, and John Lygeros. On the road between robust optimization and the scenario approach for chance constrained optimization problems. *IEEE Transactions on Automatic Control*, 59(8):2258–2263, 2014.

[80] Matlab optimization toolbox, 2017a. The MathWorks, Natick, MA, USA.

[81] Matlab, 2018a. The MathWorks, Natick, MA, USA.

[82] Matlab statistics and machine learning toolbox, 2017a. The MathWorks, Natick, MA, USA.

[83] Michael B Miller. *Mathematics and Statistics for Financial Risk Management*. John Wiley & Sons, 2013.

[84] Kevin Molloy, Laurent Denarie, Marc Vaisset, Thierry Siméon, and Juan Cortés. Simultaneous system design and path planning: A sampling-based algorithm. *The International Journal of Robotics Research*, page 0278364918783054, 2018.

[85] A Mortazavi, S Azarm, and SA Gabriel. Adaptive gradient-assisted robust design optimization under interval uncertainty. *Engineering Optimization*, 45(11):1287–1307, 2013.

[86] Frank Mufalli, Rajan Batta, and Rakesh Nagi. Simultaneous sensor selection and routing of unmanned aerial vehicles for complex mission plans. *Computers & Operations Research*, 39(11):2787–2799, 2012.

[87] Nikhil Nigam and Ilan Kroo. Control and design of multiple unmanned air vehicles for a persistent surveillance task. In *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, page 5913, 2008.

[88] Michael Otte and Emilio Frazzoli. Rrtx: Asymptotically optimal single-query sampling-based motion planning with quick replanning. *The International Journal of Robotics Research*, 35(7):797–822, 2016.

[89] Mark Owen, Randal W Beard, and Timothy W McLain. Implementing dubins airplane paths on fixed-wing uavs. In *Handbook of Unmanned Aerial Vehicles*, pages 1677–1701. Springer, 2015.

[90] Athanasios Papageorgiou. Development of a multidisciplinary design optimization framework applied on uav design. Master's thesis, Linköping University, 2015.

[91] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[92] Stéphane Petti and Thierry Fraichard. Safe motion planning in dynamic environments. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 2210–2215. IEEE, 2005.

[93] Arun Prakash and Karthik Srinivasan. Sample-based algorithm to determine minimum robust cost path with correlated link travel times. *Transportation Research Record: Journal of the Transportation Research Board*, (2467):110–119, 2014.

[94] Stefano Primatesta, Luca Spanò Cuomo, Giorgio Guglieri, and Alessandro Rizzo. An innovative algorithm to estimate risk optimum path for unmanned aerial vehicles in urban environments. *Transportation Research Procedia*, 35:44–53, 2018.

[95] S Primatestaa, A Rizzoa, and A la Cour-Harbob. Ground risk map for unmanned aircraft in urban environments. *Reliability Engineering and System Safety*, 2018.

[96] KM Ragsdell and DT Phillips. Optimal design of a class of welded structures using geometric programming. *Journal of Engineering for Industry*, 98(3):1021–1025, 1976.

[97] Sujit Rajappa, Markus Ryll, Heinrich H Bülthoff, and Antonio Franchi. Modeling, control and design optimization for a fully-actuated hexarotor aerial vehicle with tilted propellers. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4006–4013. IEEE, 2015.

[98] Federico Alessandro Ramponi. Consistency of the scenario approach. *SIAM Journal on Optimization*, 28(1):135–162, 2018.

[99] Jahangir S Rastegar, Lidong Liu, and Dan Yin. Task-specific optimal simultaneous kinematic, dynamic, and control design of high-performance robotic systems. *IEEE/ASME transactions on mechatronics*, 4(4):387–395, 1999.

[100] John Reif and Micha Sharir. Motion planning in the presence of moving obstacles. *Journal of the ACM (JACM)*, 41(4):764–790, 1994.

[101] Jan Roskam. *Airplane Flight Dynamics and Automatic Flight Controls*. DARcorporation, 1998.

[102] Eliot Rudnick-Cohen, Shapour Azarm, and Jeffrey W. Herrmann. Planning unmanned aerial vehicle takeoff trajectories to minimize third-party risk. In *International Conference on Unmanned Aerial Systems*, 2019.

[103] Eliot Rudnick-Cohen, Jeffrey W. Herrmann, and Shapour Azarm. Risk-based path planning optimization methods for unmanned aerial vehicles over inhabited areas. *Journal of Computing and Information Science in Engineering*, 16(2):021004, 2016.

[104] Eliot Rudnick-Cohen, Jeffrey W Herrmann, and Shapour Azarm. Feasibility robust optimization via scenario generation and reduction. In *ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages V02BT03A059–V02BT03A059. American Society of Mechanical Engineers, 2018.

[105] Eliot Rudnick-Cohen, Jeffrey W. Herrmann, and Shapour Azarm. Feasibility robust optimization via scenario generation and reduction. In *ASME 2018 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*. ASME, 2018.

[106] Eliot Rudnick-Cohen, Jeffrey W. Herrmann, and Shapour Azarm. Modeling unmanned aerial system (uas) risks via monte-carlo simulation. In *International Conference on Unmanned Aerial Systems*, 2019.

[107] Eliot Rudnick-Cohen, Jeffrey W Herrmann, and Shapour Azarm. Non-convex feasibility robust optimization via scenario generation, scenario reduction and local refinement. *Under Review at ASME Journal of Mechanical Design*, pages V02BT03A059–V02BT03A059, 2019.

[108] Eliot S. Rudnick-Cohen, Shapour Azarm, and Jeffrey Herrmann. Multi-objective design and path planning optimization of unmanned aerial vehicles (uavs). In *16th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, page 2322, 2015.

[109] Eliot Sylvan Rudnick-Cohen. Risk-based multiobjective path planning and design optimization for unmanned aerial vehicles. Master's thesis, University of Maryland, 2016.

[110] Daniel Schneider. Path planning for fixed-wing unmanned aerial vehicles. Master's thesis, ETH Zürich, 2016.

[111] Mehrdad Shahabi. *Robust Shortest Path Problem: Models and Solution Algorithms*. PhD thesis, West Virginia University, 2015.

[112] Lawrence F Shampine and Mark W Reichelt. The matlab ode suite. *SIAM journal on scientific computing*, 18(1):1–22, 1997.

[113] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524. ACM, 1968.

[114] Afreen Siddiqi and Olivier L de Weck. Modeling methods and conceptual design principles for reconfigurable systems. *Journal of Mechanical Design*, 130(10):101102, 2008.

[115] Sauleh Siddiqui, Shapour Azarm, and Steven Gabriel. A modified benders decomposition method for efficient robust optimization under interval uncertainty. *Structural and Multidisciplinary Optimization*, 44(2):259–275, 2011.

[116] Sumeet Singh, Anirudha Majumdar, Jean-Jacques Slotine, and Marco Pavone. Robust online motion planning via contraction theory and convex optimization. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 5883–5890. IEEE, 2017.

[117] Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. A review on bilevel optimization: From classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295, 2017.

[118] Anthony Stentz et al. The focussed d* algorithm for real-time replanning. În *IJCAI*, volume 95, pages 1652–1659, 1995.

[119] Brian L. Stevens and Frank L. Lewis. *Aircraft Control and Simulation*. Wiley-Interscience, 1992.

[120] Christine Taylor and Olivier De Weck. Coupled vehicle design and network flow optimization for air transportation systems. *Journal of Aircraft*, 44(5):1479–1486, 2007.

[121] Alec J Ten Harmsel, Isaac J Olson, and Ella M Atkins. Emergency flight planning for an energy-constrained multicopter. *Journal of Intelligent & Robotic Systems*, 85(1):145–165, 2017.

[122] United States Census Bureau. TIGER/Line with selected demographic and economic data, 2010 census, 2010. United States Census Bureau.

[123] United States Federal Aviation Administration. Draft-nprm operation of small unmanned aircraft systems over people. Proposed changes to 14 CFR part 107, January 2019. https://www.faa.gov/uas/programs_partnerships/DOT_initiatives/media/2120-AK85_NPRM_Operations_of_Small_UAS_Over_People.pdf.

[124] U.S.A Department of Defense. Range commanders council (rcc) standard 321-07 "common risk criteria standards for national test ranges: Supplment", 2007.

[125] Lukas Wirth, Philipp Oettershagen, Jacques Ambühl, and Roland Siegwart. Meteorological path planning using dynamic programming for a solar-powered uav. In *2015 IEEE Aerospace Conference*, pages 1–11. IEEE, 2015.

[126] Paul P Wu and Reece A Clothier. The development of ground impact models for the analysis of the risks associated with unmanned aircraft operations over inhabited areas. In *Proceedings of the 11th probabilistic safety assessment and management conference (PSAM11) and the annual European safety and reliability conference (ESREL 2012)*, 2012.

[127] Yuli Zhang, Shiji Song, Zuo-Jun Max Shen, and Cheng Wu. Robust shortest path problem with distributional uncertainty. *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[128] Jianhua Zhou, Shuo Cheng, and Mian Li. Sequential quadratic programming for robust optimization with interval uncertainty. *Journal of Mechanical Design*, 134(10):100913, 2012.

[129] Jianhua Zhou and Mian Li. Advanced robust optimization with interval uncertainty using a single-looped structure and sequential quadratic programming. *Journal of Mechanical Design*, 136(2):021008, 2014.

[130] Qi Zhou, Ping Jiang, Xiang Huang, Feng Zhang, and Taotao Zhou. A multi-objective robust optimization approach based on gaussian process model. *Structural and Multidisciplinary Optimization*, 57(1):213–233, 2018.

[131] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.