METAHEURISTIC PLANNING FOR VEHICLE FLEETS WITH KINEMATIC, UNCERTAIN, AND BATTERY CONSTRAINTS

by

James Swedeen

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Computer Engineering

Approved:

Greg Droge, Ph.D. Major Professor	Burak Sarsılmaz, Ph.D. Committee Member
Mario Harper, Ph.D. Committee Member	Matt Harris, Ph.D. Committee Member
Todd Moon, Ph.D. Committee Member	David F. Feldon, Ph.D. Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2025

ABSTRACT

Metaheuristic Planning for Vehicle Fleets with Kinematic, Uncertain, and Battery Constraints

by

James Swedeen, Doctor of Philosophy

Utah State University, 2025

Major Professor: Greg Droge, Ph.D.
Department: Electrical and Computer Engineering

The field of vehicle mission planning grows more diverse and sophisticated as the technologies employed expand in capabilities and become increasingly widely used. This research contributes to three specializations in the field that have become more relevant and important in recent years. Namely, the fields to which this work contributes are motion planning under kinematic constraints, motion planning under integrated and probabilistic constraints, and electric vehicle patrol routing problems (EVPRPs).

The field of motion planning under kinematic constraints is of importance because nearly every vehicle used today is subject to the kinematic constraint of respecting maximum curvature limitations, such as cars, airplanes, and boats. This research augments existing state-of-the-art algorithms to respect these constraints, producing Fillet-Based Rapidly-exploring Random Tree (FB-RRT*) and Fillet-Based Batch Informed Trees (FB-BIT*). These algorithms respect common kinematic constraints without sacrificing algorithmic performance compared to the unconstrained versions of the algorithms. Next, FB-BIT* is augmented for the field of motion planning under integrated and probabilistic constraints. The example application of planning paths for fixed-wing unmanned aerial vehicles (UAVs), through global positioning system (GPS)-denied regions, while avoiding

detection by ground-based radar systems, is used to show the effectiveness of the developed algorithm. This involves implementing a closed-loop linear covariance (LinCov) analysis of a UAV model being localized with an inertial extended Kalman filter (EKF), with Monte Carlo simulations for validation. The augmented FB-BIT* algorithm is shown to far outperform existing methods in both reliably producing a constraint-satisfying path and the optimality of the resulting path.

Finally, this research contributes to the field of EVPRPs. These problems arise when a fleet of battery electric vehicles must monitor an area for possible problems. The literature on the patrol routing problem (PRP) is lacking works that consider electric vehicle (EV) constraints and the ones that do use simplistic battery models. This research formulates the EV-related constraints more thoroughly than the previous work on the EVPRP. Several population-based metaheuristic algorithms are augmented with a novel application of the Levenshtein edit-distance so that they can be used for the PRP. A comparison is made among them, showing that the path relinking (PR) heuristic is very effective for solving the PRP.

(170 pages)

PUBLIC ABSTRACT

Metaheuristic Planning for Vehicle Fleets with Kinematic, Uncertain, and Battery Constraints

James Swedeen

The planning of vehicle actions and movements is becoming increasingly important in daily life. With the growing list of potential and current applications of autonomous planning, the field is faced with an ever-growing list of complications. These complications make the task of producing efficient plans increasingly difficult. This research studies and develops solutions for a few of the most important vehicle planning applications in recent years.

First, there is the planning of vehicle motion while considering the physical limitations of the vehicle. For many vehicles, for example, cars, airplanes, and boats, the biggest limitation is the inability to pan directly to the left or right. Although so many vehicles have this limitation, motion planning under this constraint is a difficult problem to solve efficiently. This research augments several state-of-the-art motion planning algorithms with the constraints imposed by these vehicles without sacrificing any performance penalty compared to the original algorithms.

Next, this dissertation studies mission planning under vehicle constraints and when the location of the vehicle is not known exactly. This problem arises when an autonomous vehicle must pass through a region where global positioning system (GPS) is denied or unreliable, for example, urban environments where large buildings interfere with GPS. Not knowing exactly the position of the vehicle leads to complications when attempting to ensure that the vehicle will avoid hitting walls or triggering other undesirable outcomes. This research further augments the mission planning algorithms from before to solve this

problem. The resulting algorithms far exceed previous work in terms of reliability and mission optimality.

The final development given in this dissertation is the design of routes for fleets of battery electric vehicles with the goal of monitoring a given region. Examples of when this application arises are monitoring a protected building with autonomous robots and law enforcement patrolling while driving electric vehicles. This research derives a formulation of the electric vehicle constraints that considers more aspects of the problem than previous works on the topic. Several algorithms are derived to solve the problem and show the benefits of the proposed constraint formulation.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

ACO	Ant Colony Optimization
AEX	alternate edge crossover
ALNS	adaptive large neighborhood search
BAPS	Bayesian Ant-based Patrolling Strategy
BIT*	Batch Informed Trees
CE	cross entropy
CVRP	capacitated vehicle routing problem
CX	cycle crossover
DVRP	distance-constrained vehicle routing problem
EDR	expected disturbance rate
EKF	extended Kalman filter
EV	electric vehicle
EVPRP	electric vehicle patrol routing problem
EVRP	electric vehicle routing problem
FA	Firefly Algorithm
FB-BIT*	Fillet-Based Batch Informed Trees
FB-RRT*	Fillet-Based Rapidly-exploring Random Tree
FMT*	Fast Marching Tree
GA	Genetic Algorithm
GPS	global positioning system
HFA	hybrid firefly algorithm
HGA	Hybrid Genetic Algorithm
HGPRA	hybrid genetic path relinking algorithm
HPSA	hybrid population-based simulated annealing
I-RRT*	Informed Optimal Rapidly-exploring Random Tree
IP	integer program

LinCov	linear covariance
MCTS	Monte-Carlo Tree Search
MICP	mixed integer convex program
MILP	mixed integer linear program
MIP	mixed integer program
NFP	network flow problem
PMX	partially mapped crossover
POD	probability of detection
POMDP	partially observable Markov decision process
PR	path relinking
PRP	patrol routing problem
PSO	Particle Swarm Optimization
RCS	radar cross section
RRT	Rapidly-exploring Random Tree
RRT*	Optimal Rapidly-exploring Random Tree
S-RRT*	Smart Optimal Rapidly-exploring Random Tree
SA	Simulated Annealing
SCX	sequential constructive crossover
SOC	state of charge
UAV	unmanned aerial vehicle
UCT	upper confidence bounds applied to trees
UGV	unmanned ground vehicle
VRP	vehicle routing problem
VRPTW	vehicle routing problem with time windows

CHAPTER 1

INTRODUCTION

The field of vehicle mission planning is rich with a diverse set of potential and active research applications. Examples of applications include search and rescue operations [1], satellite maneuver planning [2], bus scheduling [3], law informant officer route planning [4], and motion planning for manipulators or mobile vehicles [5], to name just a few. Over time, the complexity of many of these applications has grown exponentially. As the number of vehicles taken into account during planning increases, the search space also grows. The increasing use of electric vehicles (EVs) for personal, public, and other travel has added new restrictions to problems such as bus scheduling and route planning. As the complexity and size of the planning problems grow, so must the planning algorithms used to solve them.

This work develops metaheuristic algorithms to solve two common varieties of mission planning problems. Metaheuristic algorithms can be advantageous for several reasons. They typically produce initial solutions faster and with less computational effort than traditional optimization. This makes them useful in applications where an algorithm must run on a resource-limited platform, e.g., mobile robots. Metaheuristic algorithms that consider a cost function iteratively improve the initial solution to minimize its cost. As a result, these algorithms are well suited for large problems where finding an optimal solution is infeasible due to computational restrictions. This is because the algorithm can be stopped at any time and still produce a valid solution.

The first research problem studied is within the field of path planning in uncertain and stochastic environments [6–13]. This field arises commonly in real-world applications, e.g., the position of some obstacles might be unknown or dynamic [7]. However, there are few computationally tractable algorithms for this problem because of its complexity. Most works that plan for these applications resort to simple heuristics that can be solved in a tractable length of time but have little to no optimality guarantees, e.g., [14–19]. This work

contributes an efficient algorithm to the field and uses it to plan the trajectories of fixed-wing unmanned aerial vehicles (UAVs) through adversarial environments where detection by ground-based radar stations results in mission failure [20, 21].

In addition to the detection constraints imposed by the radar stations, the first problem studied requires a fast path-planning algorithm that is capable of respecting the kinematic constraints of a fixed-wing UAV. Many earlier works on path planning have focused solely on finding any feasible path [10, 11, 17, 18, 22–27]. However, the paths generated by these planners are excessively long, leading the field to devise path planners with some notion of optimality [5, 20, 22, 28–39]. Most path planners that produce close to optimal solutions do not consider the constraints of the vehicle being planned for. One common constraint is the inability to turn the vehicle on a point, e.g., a fixed-wing UAV must maintain forward velocity while turning. When it comes to having these vehicles follow the unconstrained paths generated, a typical approach is to apply a path smoother that transforms the path into one that is followable [40, 41]. However, there is no guarantee that such methods will work or provide optimal solutions. The need to generate paths that common mobile vehicles can follow led to the advent of path planners that consider the constraints of the vehicle being planned for during the planning process [42–48]. Previous works that respect these constraints show a significant slowdown in the algorithm due to the constraints. However, as part of this work, a sampling-based path planner is devised that respects kinematic constraints such as maximum curvature and maximum curvature rate without significantly slowing the convergence of the algorithm when compared to the unconstrained version of the algorithm.

With a fast way to generate UAV followable paths, it remains to develop the probabilistic constraints related to radar detection events and incorporate those constraints into the path planner. Many previous works in the field of UAV reconnaissance ignore the effects of UAV position uncertainty when evaluating the probability of detection (POD) by radar stations. However, when passing through global positioning system (GPS)-denied areas, the variance in UAV position can lead to large deviations in the POD [9]. The only

planning work that takes these deviations into account ignores the effects of UAV truth dynamics and closed-loop control when evaluating these uncertainties [11]. Bridging this gap in the literature, this work uses the kinematically constrained path generated via the sampling-based planner as a reference trajectory. A closed-loop linear covariance (LinCov) method is used to rapidly generate UAV state covariance estimates. These estimates are used while evaluating the POD constraints in the path planner to ensure that the given probabilistic constraints are satisfied. The resulting planner is more reliable at finding constraint-satisfying solutions than previous works, and the resulting solutions are closer to optimal than those made by previous planners.

The second research problem studied changes the role of the agent being planned for. Instead of one agent avoiding detection by adversaries while reaching a location, in the patrol routing problem (PRP) problem, routes are planned for a fleet of vehicles with the goal of monitoring or patrolling an area for the stochastic arrival of a given event. The PRP is a specialization of the vehicle routing problem (VRP) [4, 49–54]. Many of the works on VRPs use metaheuristic algorithms such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Firefly Algorithm (FA) [52–62]. Some works propose exact solution strategies that usually take prohibitively long to run or simple heuristic approaches with no optimality guarantees [1, 51, 63, 64].

The increasing use of EVs has led to the need to consider battery-related constraints when developing patrol routes, leading to the electric vehicle patrol routing problem (EVPRP). Much of the research that addresses the EVPRP problem abstracts the battery constraints away from the planner by instead restricting the path length that can be covered between depot or home base visits, where charging can be performed [52–54, 57, 60]. Even among planners in this field that consider battery constraints directly [56, 61, 62, 65], the author is aware of none that consider regenerative braking or uses anything but the simplest battery models that consider the battery discharge per unit length of all streets to be equivalent regardless of road grade, speed, condition, etc. This dissertation contributes to the field of VRPs with a problem formulation that considers regenerative breaking. Additionally, this

dissertation brings EV constraints to the field of solving the PRP. Several population-based metaheuristic algorithms are developed to solve the subsequent problem, which make use of a novel application of Levenshtein edit-distances [66].

The remainder of this work proceeds as follows. The relevant literature is discussed in more detail in Chapter 2. Chapter 3 lists the contributions to the literature that this work provides. Fillet-Based Rapidly-exploring Random Tree (FB-RRT*) is discussed in Chapter 4. The fillet idea is applied to Batch Informed Trees (BIT*) in Chapter 5. The resulting Fillet-Based Batch Informed Trees (FB-BIT*) algorithm is augmented for the field of stochastic path planning in Chapter 6. The EVPRP is studied in Chapter 7. Concluding remarks on this work are given in Chapter 8. In support of the main document, several appendices are included. Appendix A details the BIT* algorithm without the added complication of using fillet-based motion primitives. Appendix B provides a thorough derivation of LinCov analysis, which is used in Chapter 6.

CHAPTER 2

LITERATURE REVIEW

This chapter describes some of the various algorithms that already exist in the literature for solving problems similar to this research. Mission planning for unmanned aerial vehicles (UAVs) through contested environments requires a computationally efficient path planner that respects the vehicle constraints of UAVs. Section 2.1 describes some of the most common and existing path planners that are highly efficient but do not consider vehicle constraints. Section 2.2 describes existing path planners that consider vehicle constraints, but are computationally inefficient. Probability of detection (POD) models, which are needed to constrain the UAV mission planning, are considered in Section 2.3. The literature on motion planning in uncertain and contested environments is discussed in Section 2.4. Section 2.5 studies the literature on the vehicle routing problem (VRP) and the electric vehicle routing problem (EVRP). Finally, what literature exists on the patrol routing problem (PRP) is considered in Section 2.6.

2.1 Holonomic Path Planning

This section discusses the literature related to the holonomic path planning problem. A vehicle for which all motion constraints can be expressed in the form $h(x) = 0$ where x is the generalized coordinates of the vehicle, i.e., position and orientation, is called holonomic [67]. A path planner that can treat the vehicle being planned for as holonomic does not need to consider any dynamic constraints, and typically will plan paths that are first-order continuous in position. Even without dynamic constraints, path planning is an NP-complete problem in general [22]. Early attempts at solving this problem include Dijkstra's path planning algorithm and A* [35]. These algorithms discretize the search space into a graph of connected states and then solve for the shortest route through the graph. Discretizing the search space introduces potential issues when the discretization is too fine

or too coarse. These issues are avoided in sampling-based motion planning algorithms, where the search space is sampled iteratively to produce a graph [5,28–30,42,43,68]. These algorithms operate on the principles of dynamic programming, breaking the problem into many smaller problems that can each be solved individually and then combined to make an overall solution.

Of particular interest and popularity are sampling-based motion planners based on Rapidly-exploring Random Tree (RRT) [23]. In its simplest form, RRT iteratively builds a rooted search tree from a given initial location that branches out to every reachable location in the search space. Once the tree finds the goal location, a path to that location can be generated by tracing the tree branches back to the initial location. In each iteration of building the search tree, a sample point is randomly selected from the search space, and primitive motions are used to extend the tree in the direction of the sampled point. The use of general motion primitives enables the application of RRT to a wide range of problems with guarantees of probabilistic completeness [22]. However, the path that RRT finds is typically far from optimal, and the probability that RRT produces the optimal solution has been shown to be zero [5].

Optimal Rapidly-exploring Random Tree (RRT*) is a sampling-based planning algorithm that was derived by modifying RRT such that the resulting algorithm has asymptotic optimality guarantees [5]. RRT* is similar to RRT, building a rooted search tree from the initial location by randomly sampling the search space and connecting the sample to the tree. Unlike RRT, as RRT* searches, local optimizations are performed on the search tree that transform the tree into a set of optimal paths to every reachable location in the search space. These local optimizations work to add and remove edges between existing nodes, requiring the motion primitives to be able to find a continuous path to connect the corresponding states represented by the nodes. This exact connection requirement is not required by the original RRT algorithm, so the RRT* changes are generally not applicable to all applications of RRT. Furthermore, the same random sampling that ensures that RRT finds a solution can cause slow convergence to the optimal solution by RRT* [28–33].

Once an asymptotically optimal variant of RRT was known, the literature exploded with variants that seek to improve the convergence rate of RRT* [28, 29, 31, 32, 34]. Most variants modify the way RRT* samples the search space, trying to sample in locations that are more likely to improve the current solution [35]. For example, Informed Optimal Rapidly-exploring Random Tree (I-RRT*) attempts to reduce the sampling space by providing a conservative estimate of the area that will contain the optimal solution [28]. I-RRT* takes advantage of cost-to-come and cost-to-go heuristics in a way similar to A* [36]. Using these heuristics, I-RRT* defines an "informed ellipse" such that any point in the state space that falls outside the ellipse cannot be a part of the optimal solution. Given that the optimal solution is contained in the informed ellipse, I-RRT* only samples within the ellipse. This reduces the volume of search space that I-RRT* searches over, and hence the convergence rate is improved.

Some variants of RRT* overcome slow convergence rates with path-refinement procedures that run periodically on the solution path. In [29], Smart Optimal Rapidly-exploring Random Tree (S-RRT*) provides an alternative sampling heuristic as well as a path refinement procedure to avoid waiting for probabilistic sampling to straighten the path. In addition, B-splines are used to smooth the resulting path as a postprocessing step in [41].

Fast Marching Tree (FMT*) is another sampling-based planning algorithm that uses the principles of dynamic programming to avoid some of the unnecessary calculations that RRT* performs [37]. FMT* builds a rooted search tree, similar to RRT*. However, instead of iteratively sampling the search space as the algorithm goes, FMT* samples a fixed number of times at startup. A search tree is formed from these sampled points. With knowledge of where samples are from the start, FMT* avoids many of the iterative calculations that RRT* performs while still producing the same solution. This makes FMT* faster than RRT* at generating a solution from a fixed number of samples. However, it loses the ability of RRT* to refine the solution indefinitely.

Batch Informed Trees (BIT*) merges the iterative nature of RRT* with the efficient graph searching techniques of FMT* [38, 39]. BIT* iteratively generates batches of samples

from the search space, then uses a procedure similar to FMT* to incorporate the new batch of samples into the preexisting search tree. The characteristics of BIT* vary with the size of each batch of samples. When the batch size is only one, BIT* is nearly equivalent to RRT*. When the batch size is very large, BIT* is nearly equivalent to FMT*, except for considering subsequent batches. This allows the user of the algorithm to tune BIT* to have the desired performance characteristics.

2.2 Kinematically Constrained Path Planning

The planners described above are highly effective at solving the holonomic path planning problem quickly, optimally, and with minimal computational effort. However, many problems exist that cannot be formulated with holonomic constraints. To solve these non-holonomic problems, the algorithms above must be augmented with the corresponding problem constraints. In the case of this work, the problem constraints include a maximum allowed path curvature and sometimes a maximum allowed path curvature rate. These kinematic constraints allow the vehicle models being planned for to follow the resulting trajectory.

In recent years, the problem of planning paths for kinematically constrained vehicles has received increasing attention [43–47, 69]. A kinematically constrained vehicle is one in which all vehicle motion constraints can be expressed in the form $h(x, \dot{x}) = 0$, where x and $\dot{x}$ are the generalized coordinates and velocities of the vehicle, respectively. One kinematic constraint of particular interest is that of maximum curvature. Many vehicles have a maximum curvature restriction: cars, fixed-wing UAVs, and boats. In the case of this work, fixed-wing UAVs have a maximum curvature constraint because the vehicle must maintain forward velocity while turning. Curvature constraints can be trivially satisfied in nearly open environments by smoothing the straight-line path generated by one of the planners described above [25, 41]. However, in tight corridors, this smoothing approach often results in obstacle collisions or requires turns that violate the curvature constraints. Furthermore, the optimality of the resulting solution cannot be guaranteed without considering the constraints during path planning.

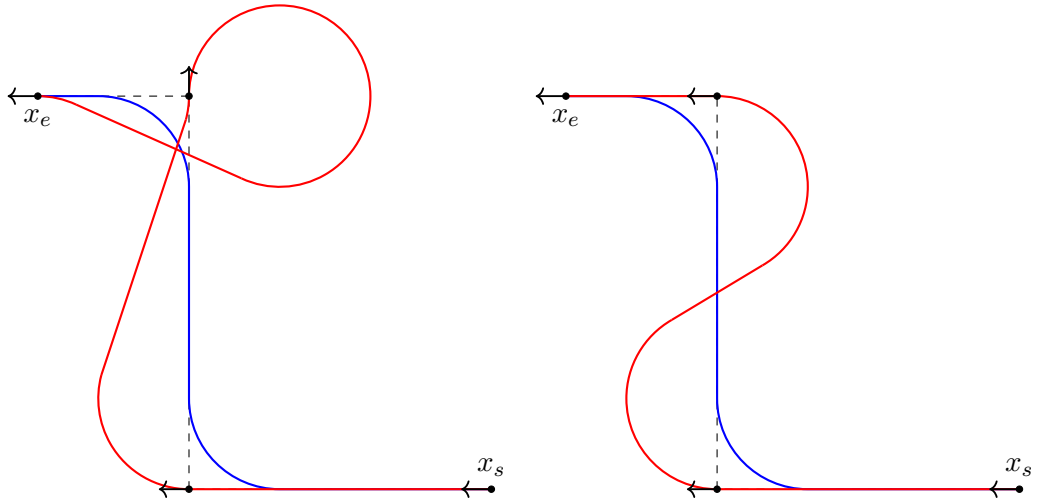


Fig. 2.1: Given the set of nodes that start with x_s and end with x_e, their respective orientations are denoted with arrows pointing from them. The blue path is the path that is made by the arc-fillet path generation, and the red path is the path that is made by Dubin's paths.

Many have found that curvature constraints are difficult to respect in RRT*-based algorithms without significantly slowing the convergence rate of the algorithms they are using [46, 47]. When applying the principles of dynamic programming, difficulty arises when attempting to connect subproblem solutions while respecting curvature constraints. One common method for considering curvature constraints is to plan with straight lines and arcs using techniques such as Dubin's and Reed-Shepp's paths [22, 25]. Although these techniques provide the shortest paths between oriented waypoints, they are not well suited for the local optimization procedures in RRT* due to the inclusion of orientation. This is because the shortest path connecting two oriented points can vary significantly with small orientation changes; see Figure 2.1.

In [42, 48], an alternative motion primitive is used in the form of a fillet. Instead of connecting two points, a fillet connects two straight-line segments with a curve that starts on the first segment and ends on the second segment. Minor changes in each line produce small changes in the path length, making the fillet approach amenable to the local optimizations required by RRT*. In the case of [42, 48], Bézier curves are used to connect line segments, with the added benefit that continuous change in curvature is guaranteed [40]. However,

their work lacks a formal analysis of the additional constraints needed in RRT* to produce valid paths. Instead, they overestimate the constraints with a heuristic, which does not guarantee solution validity in all situations and significantly impairs the convergence rate.

2.3 Probability of Detection Models

When mission planning for UAVs traversing adversarial environments, a model of the probability of the UAVs being detected is necessary so that the mission can be planned to minimize this risk. The POD of a fixed-wing UAV is a function of the physical characteristics of the aircraft, the position and orientation of the aircraft relative to the radar, and the radar system parameters [19]. Calculating it using high-fidelity simulation models takes a prohibitively long time when using the POD in mission planning algorithms. Many planning techniques utilize functional estimates of the POD because they can be calculated rapidly. Common POD models include models based on the inverse of the radar to UAV range [14, 17, 21, 26], models based on the peak or aggregate radar cross section (RCS) [15], and models that consider both [13, 16, 20, 70].

The mission planning literature frequently makes the assumption when calculating the POD that the aircraft and radar positions/parameters are deterministic and known [10, 14–18, 20, 21, 26, 27, 70]. However, in practice, these values always have some inherent uncertainty. Radar parameters are estimated from any possible intelligence gathering, and the positions of radars and aircrafts are estimated through sensor measurements that are always noisy and biased [5]. Furthermore, many missions where the threat of detection must be minimized will have to enter global positioning system (GPS)-denied areas where aircraft position estimates are poor. Without GPS, accurately tracking a planned trajectory becomes impossible, resulting in large deviations from the planned route, possibly bringing the UAV closer to the radars than intended. It has been shown, see [9, 12], that these deviations in aircraft position and radar parameters cause significant variations in the POD. Therefore, a mission planner must consider these uncertainties to ensure that the resulting plan keeps the POD below an acceptable threshold.

The author is aware of only one work that develops a mission planner that considers

these uncertainties [11]. Unfortunately, it has some shortcomings in its methodology. First, the mission planner proposed has no guarantees of finding a solution when feasible solutions exist. Second, a closed-loop simulation model is never defined, and as a result, when evaluating the POD constraints, the navigation state uncertainties are used instead of the uncertainties of the true state of the aircraft. This work overcomes both of these shortcomings with advanced planning techniques and a complete closed-loop linear covariance (LinCov) covariance evaluation method.

2.4 Path Planning with Uncertainty

The mission planning literature is full of works that plan paths while respecting probabilistic constraints, such as those that arise in Chapter 6 due to uncertainty in the position of the UAV. For example, [6] uses a RRT-based algorithm where LinCov analysis is used to estimate the probability of collision with uncertain obstacles in GPS-denied areas. A similar problem is studied in [7] where environmental obstacles move in random ways over time. They develop a two-stage path generation scheme. In the first stage, RRT* is used to generate a reference trajectory assuming that there is no obstacle movement. In the second stage, a model predictive controller repeatedly solves a mixed integer convex program (MICP) that considers the uncertainty in the position of the obstacles to generate the control signal. As was the case with the path smoothing approaches discussed in Section 2.2, there is no guarantee of optimality.

The route planning of unmanned ground vehicles (UGVs) to ensure the traversability of the path through treacherous terrain is considered in [41]. A RRT*-based algorithm is developed in which planning constraints are defined to ensure that the probability of getting stuck at a location is kept below a threshold. Additionally, B-splines are used to smooth the resulting path as a post-processing step.

Decision making for a general partially observable Markov decision process (POMDP) is considered in [8]. They develop a Monte-Carlo Tree Search (MCTS) algorithm to simultaneously model the environment and plan optimal actions. Upper confidence bounds applied to trees (UCT) is used to guide the search effort in the directions that are most likely to

produce optimal results. Even with UCT to guide the search, the amount of computational effort required to produce a solution with a MCTS is typically much greater than that of more specialized algorithms.

Most of the path planning under threat of detection literature can be categorized into heuristic-based planners, with no optimality considerations, and direct optimization-based planners, which are typically computationally expensive. Within the category of heuristic-based planners, most have a shared structure. A set of waypoints is generated for the vehicle to follow, the POD is calculated along the resulting trajectory, and the waypoints are iteratively adjusted until the POD constraints are satisfied. For example, [11] defines a polygon of waypoints around each radar, solves for the shortest path with Dijkstra's algorithm, and moves the waypoint away from the radars when the POD constraints are violated. Some works avoid the need to iteratively refine their waypoints by using a POD model that is simple enough to allow them to define waypoints that are known to satisfy constraints. In [14], Delaunay triangulations are used to generate a waypoint graph that is known to satisfy constraints; then Dijkstra's algorithm is used to find the shortest path through the graph. In [26], a Voronoi tessellation is proposed that has been adjusted to keep the POD at a minimum to generate the waypoint graph. In [15], a greedy waypoint selection process is used, in which each waypoint is chosen to maximize its distance to the nearest radar. Other works avoid the need to define waypoints, instead using other control techniques; for example, potential field descent is used in [16].

Direct optimization-based planners typically formulate the problem as a mixed integer program (MIP) and then solve the program with a third-party optimization library. For example, [20] and [17] both use this approach. Some works use metaheuristics such as Particle Swarm Optimization (PSO) to solve the program [18]. Others use POD and vehicle models that are simple enough to allow an analytically optimal solution to be found [21]. However, all of these works lack important aspects of the problem being solved in Chapter 6. This simplification makes it practical to solve the problems they solve with the algorithms they use, but it is unclear whether these same algorithms would solve more advanced problems.

2.5 Battery Constrained Vehicle Routing Problem

The PRP studied in Chapter 7 has challenges similar to the VRP. The VRP arises when a fleet of vehicles in a central depot location is tasked with achieving a set of geographically distributed goals with minimal cost. Common situations where the VRP applies are mail delivery, product distribution [62], and watering dusty roads [58]. Chapter 7 solves a variant of the VRP that considers the constraints of electric vehicles (EVs), which is known as the EVRP. The EVRP arises in this work when a fleet of patrolling vehicles are charged with enforcing laws throughout a city while maintaining a safe battery charge level.

The VRP in an urban setting with time-dependent travel times is studied in [55]. In this problem, each vehicle is loaded with goods that must be delivered to customers while complying with the delivery time window and the vehicle load capacity constraints. Another article studies the VRP in an urban setting with limitations on load and battery capacities, time-dependent travel times, and road tolls [56]. In [58], a version of the VRP is considered, in which the goal to be achieved is to water dirt roads to reduce dust. The humidity of the roads changes over time, and each truck has a maximum water capacity. In all three works, the problem is formulated as a mixed integer linear program (MILP) by breaking the planning time horizon into regions and formulating the constraint of the problem within each of those windows. A generalized procedure is provided by [55] that picks the time windows such that the start and end of each correspond to every decision boundary in the problem. Zhang and Riquelme in [56,58] split the time horizon into fixed-length intervals and incur a potential loss of accuracy as a result. To improve their solving times, they develop versions of the adaptive large neighborhood search (ALNS) algorithm that produce solutions faster but have weak optimality guarantees, resulting in a reduction in the quality of the objective value of approximately 10%. The vehicle routing problem with time windows (VRPTW) on when goods must be delivered in an urban environment is studied in [63]. They formulate the problem as a MILP, similar to [55,56,58], and define lower and upper bounds on the objective function to speed up solve times.

In [57], the distance-constrained vehicle routing problem (DVRP) is considered where

the objective function is to minimize the distance traveled. They develop a few Hybrid Genetic Algorithms (HGAs), i.e., Genetic Algorithms (GAs) that have been augmented with occasional greedy local searches to speed up convergence time, to solve the problem. In [59], they develop a population-based improved simulated annealing algorithm with a crossover operator that they call ISA-CO. This algorithm is almost exactly like a HGA except for two things. First, instead of performing crossover, mutation, and then local search, this algorithm performs local search, crossover, and then population trimming. Additionally, they use the Simulated Annealing (SA) heuristic when deciding to keep a less optimal solution after the local search. The effectiveness of four common GA crossover operations is studied in the context of the DVRP in [60]. Those operators are sequential constructive crossover (SCX), cycle crossover (CX), partially mapped crossover (PMX), and alternate edge crossover (AEX). They find that SCX works best for the DVRP.

Yang [71] proposed a new metaheuristic algorithm in 2009 called the Firefly Algorithm (FA). FA is a nature-inspired algorithm that is roughly modeled after the behavior of fireflies to flash when attracting a mate. Yang shows that the PSO algorithm can have characteristics similar to those of the FA, but the FA is more adaptable to different problems. FA in its original formulation is designed for continuous search spaces. Unlike PSO, in FA each element of its population is attracted to all other elements with better fitness, instead of just the global optimal and the historic best of the particle.

The asymmetric capacitated vehicle routing problem (CVRP) with pickups, deliveries, and variable transport time is considered in [72]. They modify the FA for use in this discrete setting. They use the Hamming distance to define the distances between fireflies. The movement is performed by randomly switching the visitation order of one vertex. This is repeated the number of times drawn, and each is considered a different solution. The lowest cost one becomes the new firefly.

A FA is modified to solve the CVRP in [61]. They use the Hamming distance to define inter-firefly distances and crossover operations to move less fit solutions toward more fit solutions. In addition, they hybridize the algorithm with the improved 2-opt and 2-h-opt

procedures. Finally, they include two mutation operators for diversification. However, as their follow-up paper [62] discusses, this algorithm tends to get stuck in local minima. They attribute this to hybridization; however, the author thinks it is because they steer all solutions towards the global optimal instead of each pairwise optimal, as is done in the original FA. In [62], they suggest a cooperative FA algorithm using concepts from the cooperative island model. In this algorithm, multiple hybrid FA instances are run independently. Periodically, some of the population of each instance is mixed with the other FA instances. This helps prevent the algorithm from getting stuck in local minima.

2.6 Patrol Routing Problem

The field of PRPs is a specialized subset of the field of VRPs where agents are continuously patrolling an area. Patrolling, in general, is the act of periodically traveling around an area with the goal of protecting or monitoring. Law enforcement officers employ a wide range of patrolling strategies depending on the geographic region and situation. The strategy used in Chapter 7 is hotspot patrolling, an effective and promising method for preventing crime. This is the practice of increasing patrol activity in geographical areas with high crime intensity. The increased presence of police deters would-be criminals before a crime is committed [50].

In [1], the problem of searching an open area with a fleet of aircraft is considered. They design a potential field that pushes agents to unobserved regions of the search space and away from other agents in the fleet. While [1] has promising results, it does not consider a street graph to plan over but a continuous, obstacle-free space.

The PRP and the metrics that are the most important for the patrolling problem are studied in [54]. Specifically, Chen argues that these metrics are efficiency, flexibility, scalability, unpredictability, and robustness. They develop a solver called Bayesian Ant-based Patrolling Strategy (BAPS) that is similar to the potential field accent approach described in [1] except that BAPS operates on a discrete graph. Each hotspot is associated with a "pheromone" level that increases when the hotspot is visited and decays exponentially when a patrolling agent is present at the hotspot. Patrolling agents are drawn to close hotspots

with low levels of pheromone. They develop a simulation with hotspots and emergency calls to evaluate BAPS.

In [53], the PRP is considered with a fixed number of hotspots. Each hotspot has an expected disturbance rate (EDR), an estimated model of how much crime that area will see that decreases when an agent monitors it and increases otherwise. The rate of increase is a time-dependent function, as they argue is the case in reality. Agent charge and time constraints are not considered. They first formulate an integer program (IP), and then a cross entropy (CE) solver is derived to speed up the solve times. The CE method lends itself well to producing unpredictable paths, though it is unclear whether it would be amenable to complicated cost functions and large numbers of graph vertices.

The PRP in which the objective is to maximize the time spent monitoring the hotspots is studied in [51]. Each hotspot is only active during certain time windows. They analyze the MIP formulation in [64], which is NP-hard. They show that the same problem can be formulated as a network flow problem (NFP) which can be solved in polynomial time. They then extend the problem formulation to allow multiple shift start and end locations, which results in an NP-hard problem regardless of the formulation.

Two metaheuristic algorithms are developed for the PRP in [52] and [49]. A GA is developed to solve a patrolling problem where a crime mapping algorithm predicts crime hotspots and sets the priority of patrolling each hotspot in [49]. An actor-critic machine learning method is used to find paths that monitor a two-dimensional grid of locations where the input of the actor-critic block comes from a graph attention network fed with the observations of each patrolling agent in [52]. Overall, these machine learning based algorithms seem to need more work before they can produce quality results reliably.

CHAPTER 3

RESEARCH CONTRIBUTIONS

As discussed in Chapter 1 and further explored in Chapter 2, there are currently gaps in the state of the field of vehicle mission planning. This research contributes to several subfields within vehicle mission planning, filling gaps in the literature. The contributions are relevant and important to the current state of the art within the fields they contribute to.

The first contribution is in the field of kinematically constrained vehicle motion planning. This is an important field because so many modern vehicles have kinematic constraints, e.g., boats, airplanes, and cars. While the literature contains motion planners that consider such kinematic constraints, most either produce very long and sub-optimal paths or take a prohibitively long time to run. The literature also contains path planners that rapidly converge to highly optimal solutions, e.g., Optimal Rapidly-exploring Random Tree (RRT*), Batch Informed Trees (BIT*); however, an efficient way to consider kinematic constraints within them is lacking. Chapters 4 and 5 fill this gap in the literature by developing the Fillet-Based Rapidly-exploring Random Tree (FB-RRT*) and Fillet-Based Batch Informed Trees (FB-BIT*) algorithms, which respect kinematic constraints and converge rapidly. These algorithms use fillets and motion primitives within the formulation of RRT* and BIT*. The paths generated by these algorithms respect any kinematic constraints that the fillets used respect. Furthermore, they produce paths of approximately the same length as RRT* and BIT* in the same amount of time, while simultaneously respecting additional constraints. Additionally, Chapter 4 develops a sampling heuristic that accelerates the convergence of general sampling-based motion planners.

The next two contributions are within the field of mission planning under probabilistic constraints. Specifically, the field of mission planning for fixed-wing unmanned aerial vehicle (UAV) through adversarial environments with unreliable global positioning system (GPS)

support while under the threat of detection by ground-based radar systems. Within the field of adversarial mission planning, few works consider the fact that these missions often must enter GPS-denied regions. The ones that do use simplistic open-loop UAV models. Especially when GPS is denied, the effects of a closed-loop controller can have significant effects on the uncertainty of the UAV over time. Chapter 6 fills this gap by developing a representative closed-loop UAV simulation model and using it while defining constraints related to the probability of detection (POD) by the adversarial radar systems. This involves defining a full-state inertial extended Kalman filter (EKF) and deriving the closed-loop linear covariance (LinCov) analysis equations to accelerate evaluation times over Monte Carlo simulations.

The second contribution in Chapter 6 is within the generalized field of GPS-denied UAV path planning. In the existing literature, mission planners rarely produce optimized and efficient paths. Chapter 6 augments FB-BIT* to be able to respect the constraints that arise when a UAV must plan through GPS-denied areas. The resulting planner outperforms existing mission planners in terms of both reliably producing a solution and the optimality of the solutions produced.

Chapter 7 also contains two contributions to closely related fields. The first is with the field of solving the electric vehicle routing problem (EVRP). Of the works relating to the EVRP, few consider battery constraints directly, instead approximating them as a maximum trip distance constraint. The works that consider battery constraints use simple versions of a battery model, and none consider regenerative braking. Chapter 7 formulates the electric vehicle (EV) constraints while considering regenerative breaking and shows that the resulting problem can produce higher quality results than a trip distance-based constraint.

The works relating to the patrol routing problem (PRP) are relatively sparse and, to the knowledge of the author, none consider EV constraints. Chapter 7 contributes to the field of solving the electric vehicle patrol routing problem (EVPRP). Additionally, the Levenshtein edit-distance calculation is used to propose variants of the path relinking (PR)

and Firefly Algorithm (FA) heuristics that can be used for the EVPRP. These heuristics and that of Simulated Annealing (SA) are used to define population-based metaheuristic algorithms. The algorithms are compared on several problem instances to show which are helpful for solving the EVPRP.

CHAPTER 4

FILLET-BASED RRT*: A RAPID CONVERGENCE IMPLEMENTATION OF RRT* FOR CURVATURE CONSTRAINED VEHICLES

Published in the *Journal of Intelligent & Robotic Systems* [73]

Journal of Intelligent & Robotic Systems (2023) 108:68
https://doi.org/10.1007/s10846-023-01846-x

REGULAR PAPER

Fillet-based RRT*: A Rapid Convergence Implementation of RRT* for Curvature Constrained Vehicles

James Swedeen[1] · Greg Droge[1] · Randall Christensen[1]

Received: 10 March 2022 / Accepted: 1 March 2023 / Published online: 20 July 2023

Abstract
Rapidly exploring random trees (RRTs) have proven effective in quickly finding feasible solutions to complex motion planning problems. RRT* is an extension of the RRT algorithm that provides probabilistic asymptotic optimality guarantees when using straight-line motion primitives. This work provides extensions to RRT and RRT* that employ fillets as motion primitives, allowing path curvature constraints to be considered when planning. Two fillets are developed, an arc-based fillet that uses circular arcs to generate paths that respect maximum curvature constraints and a spline-based fillet that uses Bézier curves to additionally respect curvature continuity requirements. Planning with these fillets is shown to far exceed the performance of RRT* using Dubin's path motion primitives, approaching the performance of planning with straight-line path primitives. Path sampling heuristics are also introduced to accelerate convergence for nonholonomic motion planning. Comparisons to established RRT* approaches are made using the Open Motion Planning Library (OMPL).

Keywords Motion planning · Sample-based algorithms · Rapidly-exploring random trees · RRT*

1 Introduction

The ability to plan paths through complex obstacles is a fundamental requirement of many mobile robot applications and has been shown to be an NP-complete problem in general [11]. A variety of methods exist to decompose this NP-complete problem into manageable subproblems. One class of methods that has seen explosive growth in recent years is sample-based motion planning techniques [4, 6, 15–18, 23]. Of particular note is the Rapidly-exploring Random Tree (RRT) and its optimal variant, RRT* [6]. RRT quickly plans obstacle free paths for systems with arbitrary motion primitives. RRT* provides probabilistic guarantees for asymptotically converging to the optimal path, although it is not well-suited for nonholonomic motion constraints. This work contributes to the RRT* literature by developing techniques that can naturally consider the curvature constraints of a mobile robot with convergence times similar to that of straight-line motion primitives.

In its most simple form, RRT iteratively builds a search tree to randomly explore the environment. A sample point is randomly selected at each iteration and primitive motions are used to extend the tree in the direction of the sampled point. The use of general primitive motions enables the application of RRT to a wide range of problems with guarantees of probabilistic completeness [11]. However, the path that RRT finds is typically far from optimal. Karaman and Frazzoli [6] developed RRT*, which makes two modifications to the RRT algorithm that probabilistically result in asymptotic optimality. Both modifications perform local optimizations to the tree using a neighborhood of nodes around each new point being added to the tree. These local optimizations work to add and remove edges between existing nodes, requiring the motion primitives to be able to find a continuous path to connect the corresponding states represented by the nodes. This exact connection requirement is not required by the original RRT algorithm, so the RRT* changes are not generally applicable to all applications of RRT. Moreover, the same random sampling that ensures that RRT finds a solution causes the asymptotic convergence to the optimal solution by RRT* to be quite slow [1, 4, 7, 16, 17].

James Swedeen
james.swedeen@usu.edu

Greg Droge
greg.droge@usu.edu

Randall Christensen
rchristensen@blueorigin.com

[1] Department of Electrical and Computer Engineering, Utah State University, Old Main Hill, Logan 84322, UT, USA

Exactly connecting two states can become difficult when considering the motion constraints of wheeled vehicles, such as path curvature. One common method for considering curvature constraints is to plan with straight-lines and arcs using techniques such as Dubin's and Reed-Shepp paths [2, 11]. While these techniques provide the shortest paths between oriented waypoints, they are not well-suited for the local optimization procedures in RRT* due to the inclusion of orientation [3]. The path exactly connecting two oriented points can vary significantly with small changes in orientation. An alternative motion primitive in the form of a fillet was used in [21, 23]. Instead of connecting two points, a fillet connects two straight-line segments with a curve that starts on the first segment and ends on the second segment. Small changes in each line will produce small changes in the path length, making the fillet approach amenable to the local optimizations required by RRT*. In the case of [21, 23], Bézier curves were used to connect the line segments, with the added benefit that continuous change in curvature is guaranteed.

While fillets enable the use of local optimization techniques, convergence to the optimal solution is naturally rather slow in RRT*. To overcome slow convergence rates, many alternative sampling and path refinement procedures have been introduced [1, 4, 7, 16, 19]. This work utilizes two such approaches. In [4], Informed RRT* (I-RRT*) attempts to reduce the sampling space by providing a conservative estimate of the area that will contain the optimal solution. In [16], Smart RRT* (S-RRT*) provides an alternative sampling heuristic as well as a path refinement procedure to avoid waiting for the probabilistic sampling to straighten the path. We then combine ideas from [4] and [16] to develop the novel Smart and Informed RRT* (SI-RRT*) which provides greedy refinement of the solution without as many parameters to tune as S-RRT*.

This paper develops a Fillet-based RRT* (FB-RRT*) algorithm for curvature constrained path planning. Similar to [21, 23], a fillet approach is used to locally connect points. Contributions to [21, 23] include the generalization of the fillet structure for RRT planning, a relaxation of connection assumptions that increases flexibility in growing the tree, and a newly developed rewiring procedure to ensure continuity and cost improvement in the resulting path. Established sampling and path refinement procedures are also extended to the fillet structure. A minor contribution of this work is the combination of two sampling heuristics [4, 16] within the fillet framework.

The remainder of the paper proceeds as follows. In Section 2, the basics of RRT and RRT* are introduced. Section 3 then introduces the fillet approach to local planning. Section 4 develops procedures for incorporating the fillet approach into RRT* with a brief description of reverse fillet considerations given in Appendix. Section 5 then develops the smart-and-informed sampling and path refinement procedures and presents the Fillet-based RRT* algorithm. Results are presented in Section 6 using the Open Motion Planning Library (OMPL) [14] to benchmark the performance of RRT* using a straight-line motion primitive, an arc-based fillet, a Bézier curve fillet, Dubin's paths, and various sampling techniques. Concluding remarks are given in Section 7.

2 The Rapidly-exploring Random Tree

RRT-based algorithms are commonly broken into a series of generalized space sampling and tree growing procedures. RRT's variants (including RRT*) refine and augment these procedures. This section defines several basic procedures, giving them context within RRT and RRT*. The procedures in this section are found in [6, 22] with variations in notation. They are included for the sake of completeness in presenting the Fillet-Based RRT* (FB-RRT*) formulation and sampling heuristics in Sections 3 through 5.

2.1 Notation

RRT-based algorithms iteratively construct a rooted, out-branching tree to find a path through the state space. The tree is an acyclic directed graph denoted as $T = \{V, E\}$, where V is the set of nodes or vertices within the tree and $E \subset V \times V$ denotes the set of edges between vertices. If an edge points from v_1 to v_2, v_2 is referred to as the child of v_1 and v_1 as the parent of v_2. The root has no parent while all other vertices have exactly one parent. Each vertex can have multiple children. Each vertex within V corresponds to a state in the d-dimensional state space denoted as $X \subset \mathbb{R}^d$. Note that we will assume that $X \subset \mathbb{R}^2$, although that is certainly not the case for general RRT formulations. The tree is initialized with solely the root node, i.e. $V = \{x_r\}$, $E = \varnothing$. Nodes are added to the tree to find a path to the target set, $X_t \subset X$. The path must avoid the space blocked by obstacles, $X_{obs} \subset X$, staying within the free space, $X_{free} = X \setminus X_{obs}$.

Paths through the state space are written as an ordered subset of X. In RRT*, the paths are assigned a cost, typically the path length. Allowing $P(X)$ to denote the power set of X, the cost is a mapping $c : P(X) \to \mathbb{R}_+$. The closed ball of radius $r \in \mathbb{R}_+$ centered at $x \in \mathbb{R}^d$ is denoted as $\mathcal{B}_{x,r} = \{y \in X : \|y - x\| \leq r\}$. $X_s \subset X_{free}$ is the set of states from which the additional nodes will be sampled. Additional notation is summarized in Table 1.

2.2 Common Sampling-based Planning Procedures

The literature on sample-based planning defines planning algorithms using a number of procedures. We now define several generic procedures that can be found in [6] with notation changed to match the sequel.

Table 1 Notation used throughout the paper

Name	Description	Name	Description
General sets			
$\mathbb{Z}$	The set of all real-valued integers	$\mathbb{R}_+$	The set of all positive real numbers
$\mathbb{R}$	The set of all real-valued numbers	$\mathbb{R}^d$	The set of all real d-dimensional vectors
Configuration space			
X	The full configuration space, $X \subset \mathbb{R}^2$	X_{obs}	Obstacle filled configuration space
X_{free}	Obstacle free configuration space	X_t	The target set
X_s	The sampling set	X_b	The beacon set
X_{path}	An ordered set of states	X_{near}	A set of nodes near a given node
X_i	The set that, if sampled, will improve the path	X_{sol}	The solution path
x	An element of X, a 2D position	ψ	Used to denote orientation
Special sets			
$\mathcal{E}_{x_a,x_b}$	A subset of X in an ellipse with focal points x_a and x_b	$\mathcal{C}^y$	The set of all $y \in \mathbb{Z}_+$ times continuously differentiable paths
$\mathcal{B}_{x,r}$	The closed ball of radius r and centered at x		
Operators			
$R(\theta)$	The right-handed rotation matrix parameterized by angle θ	$a \vdots b$	Tests if a is divisible by b (i.e., true if $a \bmod b == 0$)
$c(X_{path})$	Cost of a path through the state space	$\|x\|$	The 2-norm of x
$\wedge, \vee$	Logical "and" and "or' operators	$\binom{n}{k}$	N choose k, i.e. $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
Tree notation			
V	A set of vertices	T	An acyclic directed graph, $T = \{V, E\}$
E	A set of edges that connect vertices	x_r	The root node
Planning Parameters			
α	The max number of neighbors to consider	η	The max distance to travel when steering a point
ρ	The radius for searching for nearest neighbors	b_t	Determines how frequently X_t will be sampled
γ	The angular displacement between two vectors	b_b	Determines how frequently X_b will be sampled
Tree Search Variables			
x_p	A parent node	x_c	A child node
x_{gp}	A parent of a parent (i.e., grandparent node)	x_{gc}	A child node of a child (i.e., grandchild node)
x_n	A new node to be added to the tree	x_{rand}	A randomly sampled state
c_y	The cost of the node x_y	$x_{nearest}$	The point that is closest to x_n in V
x_{best}	The last node in the best X_{sol} set found so far	x_{near}	A point that is close to x_n, $x_{near} \in X_{near}$
Fillets			
κ_{max}	The maximum path curvature allowed	s	A spatial indexing
x_1, x_2, x_3	The point where a fillet begins, the center point, and the ending point	s_0, s_1, s_2, s_3	The index of the fillet at x_1, x_s, x_e, and x_3 respectively
x_s, x_e	Points where fillet curve starts and ends	$\mathcal{F}_i$	Abbreviation for the full fillet length
$\Psi(s)$	The spatially indexed curve of the fillet	Ψ_i	The path length of the curve centered at x_i

Table 1 continued

Name	Description	Name	Description
b_i	Distance from x_{i-1} to the beginning of the fillet curve centered at x_i	e_i	Distance from x_{i+1} to the end of the fillet curve centered at x_i
$d(\gamma)$	The distance from x_s or x_e to x_2 given the angular displacement between fillet lines	$m_{i,j}$	Distance on the line $\overline{x_i x_j}$ that isn't replaced by a fillet curve, i.e. $\|x_i - x_j\| - d(\gamma_i) - d(\gamma_j)$
Arcs			
r	The radius of the circle made from executing κ_{max}	ζ	Distinguishes clockwise and counterclockwise arcs
θ	An angular variable ranging from $[0, \gamma)$		
Bézier fillets			
$B_{n,i}(\tau)$	A Berstein polynomial of degree n and iteration i	τ	A path parameterization index ranging from 0 to 1
$P_n(\tau)$	A Bézier curve of degree n	p_i	Control points for the Bézier curve
ν_{1-4}	Constant scalars	h,g,k	Weights that are dependent on $d(\gamma)$
Vectors and lines			
u_{ab}	The unit vector formed from $(x_b - x_a)/\|x_b - x_a\|$	$\overline{x_a x_b}$	The line that intersects points x_a and x_b
$\overrightarrow{x_a x_b}$	The vector that goes from x_a to x_b	$x_{a,i}$	Vector a's ith element

Procedure 1 *$(T \leftarrow Initialize(x_r))$ Returns an initialized tree with $x_r \in X$ as the root node and no edges, i.e. $T \leftarrow \{V, E\}$, $V \leftarrow \{x_r\}$, and $E \leftarrow \varnothing$.*

Procedure 2 *$(x_{rand} \leftarrow Sample(X_s))$ Returns a random state from the set $X_s \subset X$.*

Procedure 3 *$(x_{nearest} \leftarrow Nearest(x_{rand}, T))$ Finds the nearest vertex in the set V to the state $x_{rand} \in X$ using the 2-norm as a distance.*

Procedure 4 *$(X_{near} \leftarrow Near_{\rho,\alpha}(x_{rand}, T))$ Finds the nearest $\alpha \in \mathbb{Z}_+$ vertices that are within a given radius[1], $\rho \in \mathbb{R}_+$, of the point $x_{rand} \in X$.*

The constant α is used to prevent too many connections from being attempted in a given iteration [11].

Procedure 5 *$(x_n \leftarrow Steer_\eta(x, y))$ Returns a point that is within a predefined distance $\eta \in \mathbb{R}_+$ from $x \in X$ in the direction of $y \in X$, i.e.*

$$Steer_\eta(x, y) = argmin_{\{z \in X : \|z-x\| \le \eta\}} \|z - y\|.$$

The Steer function prevents long edges from being added to RRT search trees. This is important because it reduces the expected extension length at each iteration and likewise reduces the likelihood that a given iteration will fail to expand the search tree due to its edge being blocked by an obstacle [10].

[1] In this work ρ is held constant, but many RRT* based algorithms vary ρ [6].

Procedure 6 *$(T \leftarrow InsertNode(x_n, x_p, T))$ Adds the node $x_n \in X_{free}$ to the tree with $x_p \in V$ as the new node's parent, i.e. $V \leftarrow V \cup \{x_n\}$; $E \leftarrow E \cup \{(x_p, x_n)\}$.*

Procedure 7 *$(X_{sol} \leftarrow Solution(x_v, T))$ Finds the path through T, $X_{sol} \subset V$, that leads from the root node to x_v.*

Procedure 8 *$(X_{path} \leftarrow Path(x_{start}, x_{end}))$ Builds an ordered set of states that connect the state $x_{start} \in X$ to $x_{end} \in X$ without considering obstacles.*

Note that *Solution* is used to search the tree while *Path* is used to search X in an attempt to grow the tree.

Procedure 9 *$(bool \leftarrow CollisionFree(X_{path}))$ Returns true if and only if X_{path} is obstacle free, i.e. $X_{path} \subset X_{free}$.*

Procedure 10 *$(x_p \leftarrow Parent(x_c, T))$ Returns the parent node of $x_c \in V$ in the tree T, or $\varnothing$ if x_c is the root node.*

Procedure 11 *$(X_{children} \leftarrow Children(x_p, T))$ Returns every node from the set V in T that has x_p as its parent.*

Procedure 12 *$(c_v \leftarrow Cost(x_v, T))$ Returns the cost of $x_v \in V$. The cost of a vertex is defined as the path length traveled from x_r to x_v along the tree, i.e.*

$$Cost(x_v, T) = c(Solution(x_v, T)).$$

Procedure 13 *$(c_n \leftarrow CostToCome(x_n, x_p, T))$ Calculates the cost of $x_n \in X$ if it were connected to the tree through $x_p \in V$, returning an infinite cost if the path is not obstacle free. It is defined in Algorithm 1.*

Algorithm 1 $c_n \leftarrow CostToCome(x_n, x_p, T)$.

1: $X_{path} \leftarrow Path(x_p, x_n)$
2: **if** $CollisionFree(X_{path})$ **then**
3: **return** $Cost(x_p, T) + c(X_{path})$ ▷ Path length calculation
4: **else**
5: **return** ∞
6: **end if**

2.3 RRT

RRT quickly searches X_{free} to find a feasible (not optimal), obstacle free solution and can be used with complex motion primitives while maintaining probabilistic completeness [13]. The RRT algorithm is composed of two main steps that are repeatedly performed until a solution is found. The first is taking a biased sample from X. The second is growing the search tree toward the random sample using the $Extend$ procedure. These two procedures are now stated.

Procedure 14 *($x_{rand} \leftarrow Biased\text{-}Sample(i, b_t, X_t)$) Returns a random point at iteration $i \in \mathbb{Z}_+$ given the sampling bias, $b_t \in \mathbb{Z}_+$, and the target set, X_t, as described in Algorithm 2.*

Algorithm 2 $x_{rand} \leftarrow Biased\text{-}Sample(i, b_t, X_t)$.

1: **if** $i \vdots b_t$ **then** ▷ Bias towards target set
2: $x_{rand} \leftarrow Sample(X_t)$
3: **else** ▷ Otherwise sample a random point
4: $x_{rand} \leftarrow Sample(X)$
5: **end if**
6: **return** x_{rand}

The sample is biased towards the target set by selecting the sample from X_t every b_t iterations. b_t is a design parameter affecting exploration and exploitation. A small b_t will attempt to connect the tree to the target set more frequently.

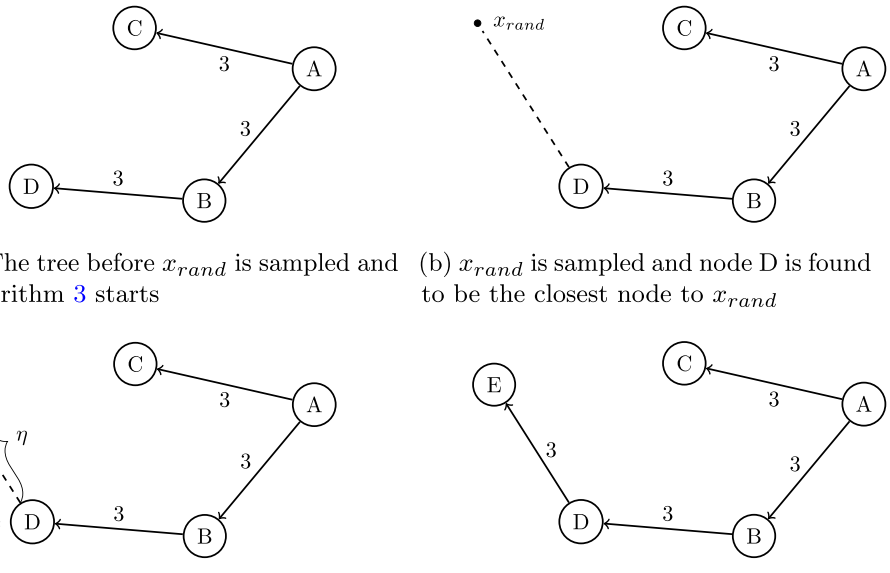
Procedure 15 *($\{x_n, x_p, c_n\} \leftarrow Extend(x_{rand}, T)$) Given a sample, $x_{rand} \in X$, and tree, $T = \{V, E\}$, the $Extend$ procedure finds the closest vertex to x_{rand} that is already in V and checks if a valid extension can be made from the tree towards x_{rand}, see Algorithm 3.*

The $Extend$ procedure is illustrated in Fig. 1. Note that RRT does not make use of the extension cost c_n; it is included for use in RRT*.

Algorithm 3 $\{x_n, x_p, c_n\} \leftarrow Extend(x_{rand}, T)$.

1: $x_{nearest} \leftarrow Nearest(x_{rand}, T)$
2: $x_n \leftarrow Steer_\eta(x_{nearest}, x_{rand})$ ▷ Steer towards the nearest point
3: $c_n \leftarrow CostToCome(x_n, x_{nearest}, T)$ ▷ Evaluate cost of resulting path
4: **if** $\infty \neq c_n$ **then**
5: **return** $\{x_n, x_{nearest}, c_n\}$
6: **else**
7: **return** $\{\varnothing, \varnothing, \infty\}$
8: **end if**

The RRT algorithm can now be described. First, a random point is sampled from the configuration space. If the tree can be extended, the new point is added to the tree. If the new point is in the target set then RRT returns a solution, as shown in Algorithm 4. RRT is known to quickly find solutions for complex problems as it naturally explores unexplored areas of the state space, a property called the Voronoi property [8]. When using the vertices of the tree to create a Voronoi diagram, unexplored regions correspond to larger Voronoi cells. The probability that a Voronoi cell is sampled is proportional



(a) The tree before x_{rand} is sampled and Algorithm 3 starts

(b) x_{rand} is sampled and node D is found to be the closest node to x_{rand}

(c) x_{rand} is steered toward node D resulting in the new node, node E

(d) After checking for obstacles, node E is added to the tree

Fig. 1 An illustration of the $Extend$ procedure

Algorithm 4 $X_{sol} \leftarrow RRT(x_r, X_t, b_t)$.

$T \leftarrow Initialize(x_r)$
for $i = 1, \cdots,$ inf **do**
  $x_{rand} \leftarrow Biased\text{-}Sample(i, b_t, X_t)$ ▷ Biased configuration sampling
  $\{x_n, x_p, c_n\} \leftarrow Extend(x_{rand}, T)$ ▷ Extend tree towards new point
  if $x_n \neq \varnothing$ **then**
    $T \leftarrow InsertNode(x_n, x_p, T)$ ▷ Add point to tree
    if $x_n \in X_t$ **then**
      return $Solution(x_n, T)$ ▷ Return solution if found
    end if
  end if
end for

to the size of that cell. Thus, the RRT tree naturally extends towards regions that have not yet been explored, avoiding problems with local minima and nonconvex obstacles.

2.4 RRT*

RRT* is an extension of RRT that adds probabilistic guarantees for asymptotic optimality to the probabilistic completeness guarantees of RRT [6]. RRT* does so by performing local optimizations on the edges in the tree whenever a new node is added. As the number of iterations goes to infinity, the repeated local optimization transforms the tree into a set of globally optimal paths from the root node to every reachable point in the obstacle free configuration space.

RRT* includes two significant changes to RRT, both of which concern the neighborhood set of the node being added to the tree, i.e. $X_{near} = Near_{\rho,\alpha}(x_n, T)$. The first modification is replacing the $Extend$ procedure with $Extend^*$. As illustrated in Fig. 2, the $Extend^*$ procedure selects a parent from V within a specified distance of the new point that minimizes the cost of the new node.

The second modification happens after x_n is added to the tree in a new procedure called $Rewire$. Each node in a local neighborhood is tested to see if its cost would be improved by going through the new node instead of its current parent node. If so, the edges are changed so that x_n becomes the node's new parent as illustrated in Fig. 3.

Procedure 16 *($\{x_n, x_p\} \leftarrow Extend^*(x_{rand}, T)$) Given a tree, $T = \{V, E\}$, the $Extend^*$ procedure finds the best "local" connection for extending the tree in the direction of $x_{rand} \in X$. It returns a new point to be added to the tree, x_n, and the parent, $x_p \in V$, as defined in Algorithm 5.*

Algorithm 5 $\{x_n, x_p\} \leftarrow Extend^*(x_{rand}, T)$.

$\{x_n, x_p, c_{min}\} \leftarrow Extend(x_{rand}, T)$
if $x_n \neq \varnothing$ **then** ▷ If an extension is possible
  $X_{near} \leftarrow Near_{\rho,\alpha}(x_n, T)$
  for all $x_{near} \in X_{near}$ **do** ▷ Check for a lower cost connection
    $c_{tmp} \leftarrow CostToCome(x_n, x_{near}, T)$
    if $c_{tmp} < c_{min}$ **then**
      $x_p \leftarrow x_{near}$
      $c_{min} \leftarrow c_{tmp}$
    end if
  end for
  return $\{x_n, x_p\}$
end if
return $\{\varnothing, \varnothing\}$;

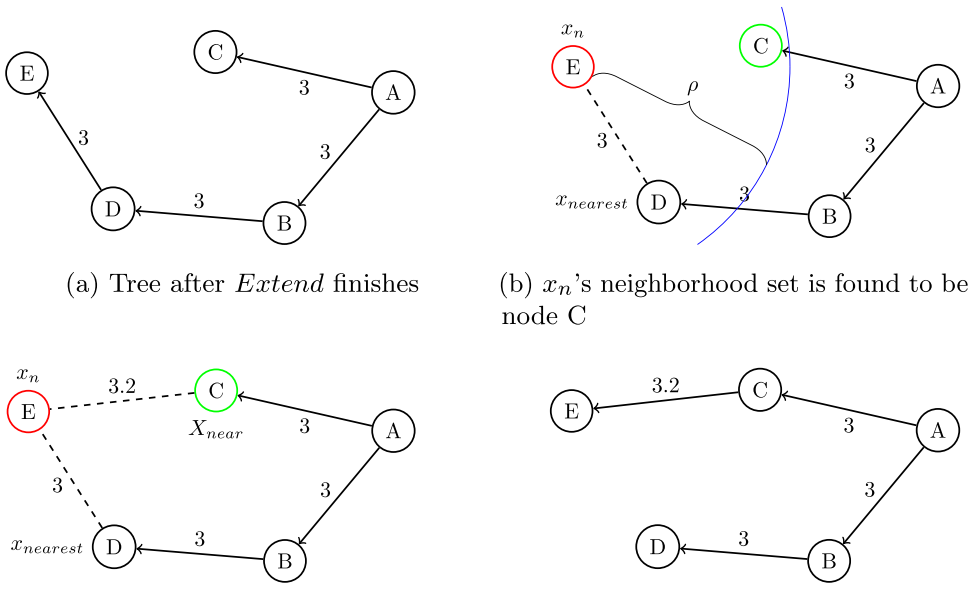


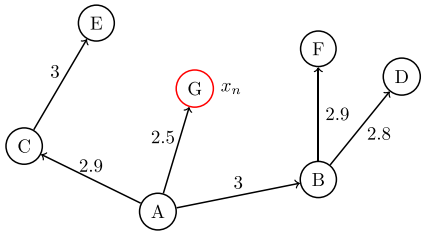
Fig. 2 An illustration of the $Extend^*$ procedure

(a) Tree after $Extend$ finishes

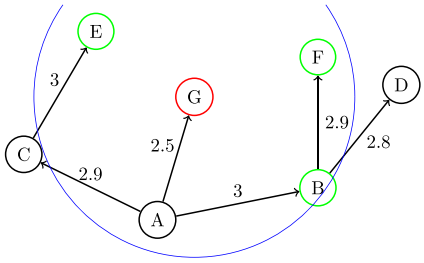
(b) x_n's neighborhood set is found to be node C

(c) Connecting through nodes C and D result in total costs of 6.2 and 9 respectively

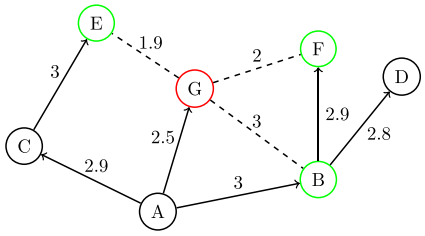
(d) Because connecting through node C yields a lower total cost, node E is connected to node C



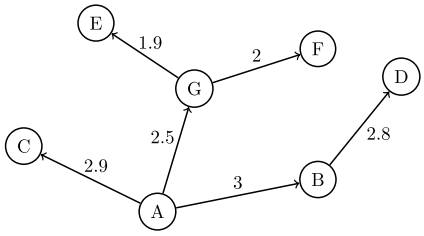
(a) The tree after $Extend^*$ adds node G to the tree



(b) Nodes B, E, and F are found to be G's neighborhood set



(c) The original costs of nodes B, E, and F are 3, 5.9, and 5.9 and their potential costs are 5.5, 4.4, and 4.5 respectively



(d) Node B is not rewired because its cost would not be lowered by the operation, however, nodes E and F are rewired

Fig. 3 An illustration of the *Rewire* procedure

Procedure 17 *($T \leftarrow Rewire(x_n, X_{near}, T)$) Given a tree, $T = \{V, E\}$, with node $x_n \in V$ and set $X_{near} \subset V$, Rewire returns a tree with a modified edge set such that x_n is made the parent of elements in X_{near} if it results in a lower cost for the elements of X_{near}. It is defined in Algorithm 6.*

Algorithm 6 $T \leftarrow Rewire(x_n, X_{near}, T)$.

for all $x_{near} \in X_{near}$ **do**
  $c_{near} \leftarrow CostToCome(x_{near}, x_n, T)$
  ▷ Check x_{near}'s feasibility and that its cost is reduced
  if $c_{near} < Cost(x_{near}, T)$ **then**
    $x_p \leftarrow Parent(x_{near}, T)$ ▷ Rewire around x_{near}
    $E \leftarrow \left(E \setminus \{x_p, x_{near}\}\right) \cup \{x_n, x_{near}\}$
  end if
end for
return T

The RRT* algorithm is shown in Algorithm 7. Note that RRT and RRT* are very similar with the main difference being the addition of the $Extend^*$ and $Rewire$ procedures. Additionally, RRT* runs for a specific number of iterations, $n \in \mathbb{N}_+$, instead of stopping when the first solution is found.

RRT* is both probabilistically complete and asymptotically optimal [6]. However, RRT* tends to converge slowly because of the Voronoi property. The Voronoi property helps RRT-based algorithms find valid solutions by encouraging exploration. As the solution improves in RRT*, the Voronoi regions around the solution get smaller, resulting in a diminishing probability that a given sample will improve the solution [1].

Algorithm 7 $X_{sol} \leftarrow RRT^*(x_r, X_t, b_t, n)$.

$T \leftarrow Initialize(x_r)$
$x_{best} \leftarrow \varnothing$
for $i = 1, \ldots, n$ **do**
  $x_{rand} \leftarrow Biased\text{-}Sample(i, b_t, X_t)$ ▷ Biased configuration sampling
  $\{x_n, x_p\} \leftarrow Extend^*(x_{rand}, T)$ ▷ Extend tree towards new point
  if $x_n \neq \varnothing$ **then**
    $T \leftarrow InsertNode(x_n, x_p, T)$ ▷ Add point to tree
    $T \leftarrow Rewire(x_n, Near_{\rho,\alpha}(x_n, T), T)$ ▷ Rewire edges around new point
    if $x_n \in X_t \wedge (x_{best} = \varnothing \vee Cost(x_n, T) < Cost(x_{best}, T))$ **then**
      $x_{best} \leftarrow x_n$ ▷ Update best path found
    end if
  end if
end for
if $x_{best} \neq \varnothing$ **then** ▷ Return best solution found
  return $Solution(x_{best}, T)$
else
  return $\{\varnothing\}$
end if

3 The Fillet Approach for Local Planning

In many cases, planned paths must obey nonholonomic constraints, e.g. [3, 10, 11]. The $Extend$ and $Path$ procedures can be modified to use basic atomic motions that satisfy such constraints during RRT-based planning, enabling RRT to be used with virtually any set of dynamics. RRT* variants, however, have no additional benefit if the underlying dynamics or primitive motions do not allow the connection of any two states using a single edge in open space [12]. The reason

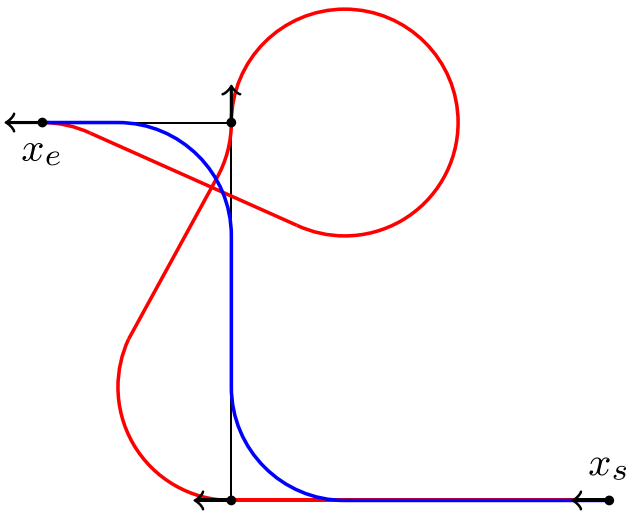


Fig. 4 Given the set of nodes that start with x_s and end with x_e their respective orientations are denoted with arrows pointing from them. The blue path is the path that is made by the arc-fillet path generation and the red path is the path that is made by Dubin's paths

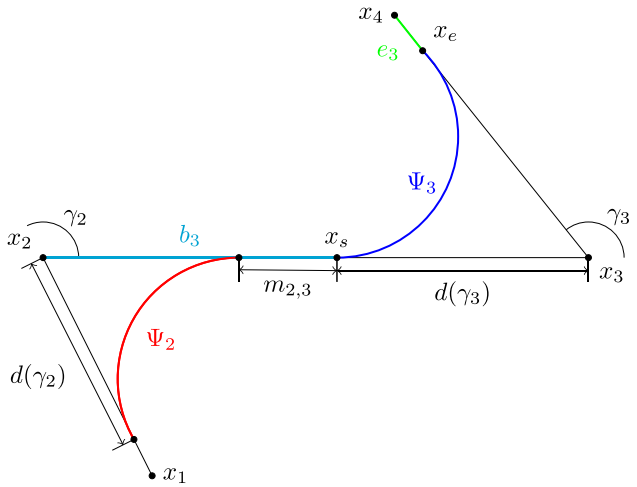


Fig. 5 The fillet generated to connect x_2 and x_4 is shown in blue with the fillet that comes before it shown in red. Note that $d(\gamma_2) + d(\gamma_3) \leq \|x_2 - x_3\|$, providing a continuous path

being that the *Rewire* procedure cannot be performed if the nodes in the neighborhood set cannot be exactly connected to each other.

This constraint is detrimental when planning with motion primitives that enforce dynamic path constraints, such as maximum curvature. A common technique for considering maximum curvature constraints is to use Dubin's paths. Dubin's paths connect orientated points with the shortest path while considering maximum curvature constraints [11]. The issue with using Dubin's paths in a sample-based path planner is that a poor choice in the orientation of the points along the solution can cause a significant increase in path length, as shown in Fig. 4. Furthermore, the orientation that minimizes overall path length changes as the solution converges to optimality.

Instead of attempting to connect two oriented points, fillets connect two line segments (defined with three unoriented points) with a curve transitioning smoothly between them, as shown in Fig. 5. Without the addition of orientation, fillets naturally allow incremental improvements to the solution. The result is a path that is continuous in position and orientation. Additional path qualities may be achieved depending on the choice of fillet. This section will introduce the general fillet concept and requirements for creating a path using fillets. Two fillets are then defined, one using an arc and one using Bézier curves. Section 4 utilizes these fillets as motion primitives in RRT-based algorithms.

3.1 General Fillets

Given three input points $x_1, x_2, x_3 \in X$, a fillet connects x_1 to x_3 with the combination of two straight-line segments and a curve. The line segments constitute portions of the lines $\overline{x_1x_2}$ and $\overline{x_2x_3}$. The curve intersects line $\overline{x_1x_2}$ at point x_s and line $\overline{x_2x_3}$ at x_e. The resulting fillet moves in a straight line from x_1 to x_s, along the curve from x_s to x_e, and then along the straight line from x_e to x_3, as depicted in Fig. 5. The major differentiator between the different fillets is the definition of the curve portion, which affects the placement of x_s and x_e.

This work assumes symmetric fillets, resulting in an equivalent distance between the node that the fillet is centered at and the two ends of the fillet curve, x_s and x_e. This distance is a function of the fillet curve type as well as the change in orientation between $\overrightarrow{x_1x_2}$ and $\overrightarrow{x_2x_3}$.

The position along the fillet can be described using a spatial index s. Allow s_0 to be where the fillet meets x_1, s_1 to be the index where the fillet's curve begins, s_2 to be the index of where the fillet's curve ends, and s_3 to be the index of where the fillet reaches x_3. Allow $\Psi(s)$ to be the fillet's curve such that $\Psi(0) = x_s$ and $\Psi(s_2 - s_1) = x_e$. Also, define the unit vector from x_i to x_j as u_{ij} (see Table 1). The position along the fillet can be written in a piecewise form as

$$x(s) = \begin{cases} x_1 + su_{12} & s_0 \leq s \leq s_1 \\ \Psi(s - s_1) & s_1 < s \leq s_2 \\ x_e + (s - s_2)u_{23} & s_2 < s \leq s_3 \end{cases}. \tag{1}$$

Defining $\gamma_i \in [0, \pi)$ as the angle measured from $\overrightarrow{x_{i-1}x_i}$ to $\overrightarrow{x_ix_{i+1}}$, the fillet distance for x_1, x_2, x_3 can be defined as $d(\gamma_2) = \|x_s - x_2\| = \|x_e - x_2\|$. Once $d(\gamma_2)$ is found, the start and end points of the curve can be written as

$$\begin{aligned} x_s &= x_2 + d(\gamma_2)u_{21} \\ x_e &= x_2 + d(\gamma_2)u_{23} \end{aligned}. \tag{2}$$

3.2 Fillet Paths

A smooth path to a destination node can be created using a sequence of points where fillets are formed from point triplets and then combined, as shown in Fig. 5. Without loss of generality, it is assumed that the path starts at node 1 and moves to node n using the sequence $x_1, x_2, ..., x_n$. The path is thus made from n nodes, using $n-2$ fillets to arrive at x_n. There are two major concerns when formulating the path. The first is the path continuity; not every sequence of points can be combined using fillets to create a continuous path. The second major consideration is path length; the $Extend^*$ and $Rewire$ procedures depend upon path length for local optimizations.

3.2.1 Path Continuity

The first key to using the fillets for planning purposes is to ensure that the path resulting from joining multiple fillets is continuous. There are two conditions to ensure feasibility: one to ensure that the fillet curve ends before the final point in the fillet and one condition to ensure that all fillets end before the next one begins. Assuming x_i is the middle node, these conditions can be expressed as

$$\begin{aligned} d(\gamma_i) &\leq \|x_i - x_{i+1}\| \\ d(\gamma_{i-1}) + d(\gamma_i) &\leq \|x_i - x_{i-1}\| \end{aligned}. \tag{3}$$

3.2.2 Length of Fillet Paths

In RRT*'s $Extend^*$ and $Rewire$ procedures, local optimizations are made to the search tree to find the shortest path. These procedures are defined for straight-line paths where the path length can be calculated as the distance between nodes in the path. This is not the case for paths created from fillets. Thus, the path length to a particular node and the length relation to other nodes are now evaluated.

Each fillet consists of two line segments and a fillet curve. The length of a single fillet with x_i as the middle node of the fillet can be expressed as

$$\mathcal{F}_i = b_i + \Psi_i + e_i \tag{4}$$

where b_i, Ψ_i, and e_i are the beginning component length, the curve length, and the end component length (see Fig. 5). Given (3), the two straight-line lengths can be expressed as

$$\begin{aligned} b_i &= \|x_{i-1} - x_i\| - d(\gamma_i) \\ e_i &= \|x_{i+1} - x_i\| - d(\gamma_i) \end{aligned}. \tag{5}$$

This definition of fillet length allows for the expression of a recursive relationship for calculating the path length in the following lemma.

Lemma 1 *Assume an ordered sequence of nodes is used to create a path using fillets with (3) satisfied for every intermediate node. Given a resulting path length of c_i to arrive at node x_i, $i \geq 3$, the path length to arrive at node x_{i+1} can be expressed as*

$$c_{i+1} = c_i + \mathcal{F}_i - \|x_i - x_{i-1}\|. \tag{6}$$

Proof The path length to node x_i along a sequence of curves and lines can be written as a summation of individual parts. The path to x_i contains $i-2$ curves of length Ψ_2 through Ψ_{i-1}. Let $m_{j,j+1}$ be the length of the straight-line segment that connects curve j to curve $j+1$, i.e.

$$m_{j,j+1} = \|x_j - x_{j+1}\| - d(\gamma_j) - d(\gamma_{j+1}), \tag{7}$$

which is positive assuming (3) is satisfied for all fillets. The length of the path to arrive at x_i is

$$c_i = b_2 + \sum_{k=2}^{i-1} \Psi_k + \sum_{k=2}^{i-2} m_{k,k+1} + e_{i-1}. \tag{8}$$

Note that the path connecting to node x_{i+1} could be expressed similarly with a length of

$$c_{i+1} = b_2 + \sum_{k=2}^{i} \Psi_k + \sum_{k=2}^{i-1} m_{k,k+1} + e_i,$$

which can be written in terms of c_i as

$$c_{i+1} = c_i + \Psi_i + m_{i-1,i} + e_i - e_{i-1}. \tag{9}$$

Given (5) and (7), (9) becomes

$$c_{i+1} = c_i + b_i + \Psi_i + e_i - \|x_i - x_{i-1}\|. \tag{10}$$

Given the definition of $\mathcal{F}_i$ in (4), (10) simplifies to (6). □

A few properties can now be stated using the recursive relationship in Lemma 1, beginning with the relationship between the path length to a node and the path length to one of its descendants.

Corollary 1 *The path length to a node is not dependent upon the choice of any nodes that come after it.*

Proof This can be seen by examining the summation form of the path length in (8) and noting that none of the variables depend upon any node after node x_i. □

The recursive path length calculation using fillets depends on the parent as well as the grandparent node. This is different from the straight-line motion primitive where the path length

to a node can be calculated using knowledge of solely the parent node. This leads to the following lemma about path length, which has significant implications for rewiring a tree connected with fillets.

Lemma 2 *Given a node x_i with a child node x_{i+1} and multiple possible parent nodes, choosing the parent for x_i to minimize c_i will not necessarily result in the smallest possible value for c_{i+1}.*

Proof The proof is given through a simple example where the shorter path to x_i results in a longer path to x_{i+1}. Consider Fig. 6. Let the path from x_r to x_6 have the same length as the path from x_r to x_1 and the same be true of x_5 and x_2, i.e.

$$\begin{aligned} Cost\,(x_1, T) &= Cost\,(x_6, T) \\ Cost\,(x_2, T) &= Cost\,(x_5, T) \end{aligned}.$$

Path A, shown in red, is the shortest path to x_3 with a path length of 1.9. Path A also results in a path length to x_4 of 2.9. On path B, shown in blue, let the fillet that connects x_5 to x_4 be an arc-fillet with $r = 0.5$, $d(\pi/2) = 1/2$, and a resulting arc length of $\pi/4$. Starting at x_6 and following path B results in a path length to x_3 of 2, which is greater than the path length to get to x_3 on path A. To minimize the path length to x_3 path A is chosen. However, the length of the path from x_6 to x_4 is $1 + 1/2 + \pi/4 + 1/2 \approx 2.785$. Thus, minimizing path length to x_3 does not minimize path length to x_4. □

Therefore, in a *Rewire* procedure, it is not sufficient to solely check the path to the node being rewired, but the path lengths to descendants must also be evaluated. As the tree grows larger, checking all descendants would be overly cumbersome. The following corollary establishes that the only descendants that need to be checked are the children nodes.

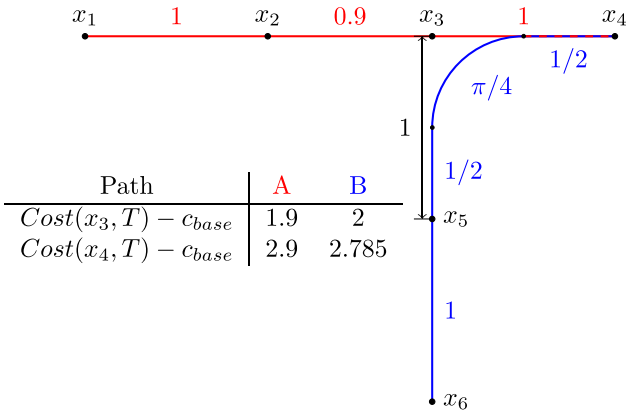


Fig. 6 Let $c_{base} = Cost(x_1, T) = Cost(x_6, T)$ and $Cost(x_2, T) = Cost(x_5, T)$. Path A, shown in red, yields a shorter path length for x_3 than path B, shown in blue. However, path B yields a shorter path length to x_4 than path A

Corollary 2 *If a node's parent and grandparent remain unchanged, but the tree is rewired such that the cost of the node's parent is lowered, then the cost of the node will be lowered by the same amount as its parent.*

Proof If a node's parent and grandparent remain the same then an examination of (6) shows that the only portion of its path length that will change is the length to the parent's node, c_i, since both $\mathcal{F}_i$ and $\|x_i - x_{i-1}\|$ remain unchanged. Thus, the same change in cost seen by a parent will be reflected in the cost of the node in question if the grandparent node remains unchanged. □

3.3 The Arc Fillet

The arc-fillet formulates the curve portion of the fillet using a circle of radius $r \in \mathbb{R}_+$ that is tangential to the two line segments at x_s and x_e, see Fig. 7. To respect curvature constraints, the radius can be chosen as the inverse of the maximum curvature, i.e. $r = 1/\kappa_{max}$. The distance between the curve intersection points and the intermediary point of the fillet is expressed in the following Lemma:

Lemma 3 *The arc-fillet distance, $d(\gamma)$, for a circular curve of radius r is*

$$d(\gamma) = \frac{r\big(1 - \cos(\gamma)\big)}{\sin(\gamma)}. \tag{11}$$

Proof Assume that the coordinate frame f is defined such that x_s is at the origin and the x-axis is pointing along $\overrightarrow{x_1 x_2}$. Using the pre-subscript f to denote a point expressed in frame f then ${}_f x_s = [0\ 0]^T$. As $d(\gamma)$ represents the distance along $\overline{x_1 x_2}$ from x_s to x_2, the value of x_2 in frame f is

$${}_f x_2 = \begin{bmatrix} d(\gamma) \\ 0 \end{bmatrix}. \tag{12}$$

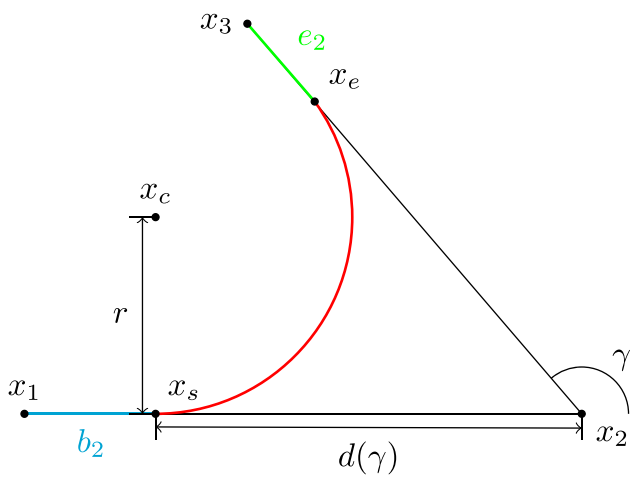


Fig. 7 The arc-fillet generated to connect x_1 and x_3

As the orientation of the path must be tangential to the circle at x_s for continuity, the center point of the circle will lie upon the y-axis at a distance r from x_s, i.e.,

$$ {}_f x_c = \begin{bmatrix} 0 \\ r\zeta \end{bmatrix}, \zeta \in \{-1, 1\}, $$

where $\zeta = 1$ corresponds to a counter-clockwise arc and $\zeta = -1$ to a clockwise arc. Parameterizing the arc by its tangent angle, the arc can be expressed in frame f as

$$ {}_f\Psi_{arc}(\theta) = r \begin{bmatrix} \sin(\theta) \\ \zeta\,(1 - \cos(\theta)) \end{bmatrix} $$

where ${}_f x_s = {}_f\Psi_{arc}(0)$. The parameter $\theta \in [0, \gamma)$ represents the orientation of the path in frame f. The tangent of ${}_f\Psi_{arc}(\gamma)$ should be parallel to $\overline{x_2 x_3}$. The task is to solve for $d(\gamma)$ such that ${}_f x_e = {}_f\Psi_{arc}(\gamma)$.

The unit vector from ${}_f x_3$ to ${}_f x_2$ can be written in terms of γ as $-[\cos(\gamma)\ \sin(\gamma)]^T$. The point ${}_f x_2$ can be expressed as a combination of this unit vector and ${}_f x_e$ as

$$ \begin{aligned} {}_f x_2 &= {}_f x_e - d(\gamma) \begin{bmatrix} \cos(\gamma) \\ \sin(\gamma) \end{bmatrix} \\ &= r \begin{bmatrix} \sin(\gamma) \\ \zeta\,(1 - \cos(\gamma)) \end{bmatrix} - d(\gamma) \begin{bmatrix} \cos(\gamma) \\ \sin(\gamma) \end{bmatrix} \end{aligned} \tag{13} $$

Equation (11) is found by setting (12) equal to (13) and solving for $d(\gamma)$. □

Remark 1 It is important to note the singularities of $d(\gamma)$ and their corresponding significance. A change of direction of $\gamma = 0$ corresponds to executing a straight line. A value of $\gamma = \pm\pi$ would correspond to reversing direction.

One advantage of the arc-fillet is that the fillet curve can be directly expressed using the spatial index s. The orientation of $\overrightarrow{x_1 x_2}$ can be written as

$$ \psi = atan2(u_{12,2}, u_{12,1}). $$

The point along the fillet curve can then be expressed in the inertial frame as

$$ \Psi_{arc}(s) = R(\psi) \cdot {}_f\Psi_{arc}\left(\frac{s}{r}\right) + x_s, \quad R(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{bmatrix}, \tag{14} $$

where θ has been replaced with $\frac{s}{r}$ as the arc-length of a circle is $s = \theta r$. Note that $\Psi(0) = x_s$ and $\Psi(\gamma r) = x_e$, as expected.

Finally, the spatial switching indices from (1) are

$$ \begin{aligned} s_1 &= s_0 + \|x_s - x_1\| \\ s_2 &= s_1 + \gamma r \\ s_3 &= s_2 + \|x_3 - x_e\| \end{aligned} . $$

The development of the arc-fillet enables a definition of a new procedure for creating a path between three points:

Procedure 18 *($X_{fillet} \leftarrow ArcFillet\,(x_0, x_1, x_2, x_3)$) The $ArcFillet$ procedure uses (2), (11), and (14) to make an arc that connects $\overline{x_1 x_2}$ to $\overline{x_2 x_3}$. Feasibility checks are made with respect to the arc that connects $\overline{x_0 x_1}$ to $\overline{x_1 x_2}$ using (11) to define $d(\gamma)$ in (3). If the checks fail the null path is returned.*

Remark 2 The arc-fillet is made from straight-line segments and arcs assuming forward motion as is a Dubin's path. As a result, a path made using arc-fillets will have the same continuity properties as a Dubin's path, i.e. $\mathcal{C}^1$ in position and orientation with curvature assuming instantaneous changes between zero and maximum curvature [11].

3.4 The Bézier Fillet

The arc-fillet assumes an instantaneous change in curvature which may be inappropriate for some scenarios that require higher levels of smoothness. In this section, Bézier curves are used to generate the fillet, resulting in paths that are $\mathcal{C}^2$ continuous in position and orientation, and $\mathcal{C}^1$ continuous in curvature. A detailed description of the curve is left to [23] and the references therein. The definition of the curve is shown as follows for the sake of completeness.

A Bézier curve connects two points and is defined as,

$$ P_n(\tau) = \sum_{i=0}^{n} p_i B_{n,i}(\tau), \tag{15} $$

where $p_i \in \mathbb{R}^2$ are control points, $n \in \mathbb{Z}_+$ is the degree of the polynomial, τ is a path parameterization index such that $0 \le \tau \le 1$, $P_n(0) = p_0$, and $P_n(1) = p_n$. Note that there is no direct relationship between changes in τ and path length [5]. The functions $B_{n,i}(\tau)$ are Berstein polynomials defined as

$$ B_{n,i}(\tau) = \binom{i}{n} \tau^i (1 - \tau)^{n-i}. \tag{16} $$

[23] combines two cubic Bézier curves to generate the curve of the fillet. In Fig. 8, the curves are denoted as ${}_0P$ and ${}_1P$ with the connecting lines denoted as b_2 and e_2. The fillet distance can now be stated.

Lemma 4 *The distance between the intermediary point, x_2, and the curve's start and end points can be expressed as*

$$ d(\gamma) = \frac{\nu_4 \sin\left(\frac{\gamma}{2}\right)}{\kappa_{max} \cos^2\left(\frac{\gamma}{2}\right)}, \tag{17} $$

where

$$ \nu_1 = 7.2364\ \nu_2 = \tfrac{2}{5}\left(\sqrt{6} - 1\right)\ \nu_3 = \tfrac{\nu_2 + 4}{\nu_1 + 6}\ \nu_4 = \tfrac{(\nu_2 + 4)^2}{54\nu_3}. $$

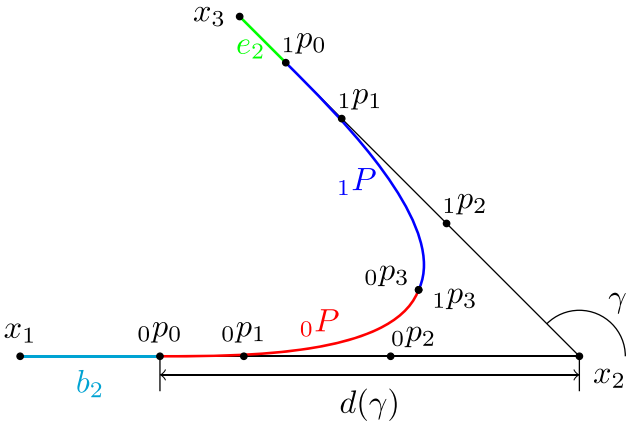


Fig. 8 The fillet generated to connect x_1 and x_3. Note that sets ${}_0p_i$ and ${}_1p_i$ are control points that replace p_i in (15) and in analogy to Fig. 7, ${}_0p_0 = x_s$ and ${}_1p_0 = x_e$

Proof See Section 2.1 of [23]. □

To define the fillet curve, four control points are needed for each of the two Bézier curves. The control points for ${}_0P$ and ${}_1P$ are denoted as ${}_0p_i$ and ${}_1p_i$, respectively, and can be expressed as

$$\begin{aligned} {}_0p_0 &= x_2 + d \cdot u_{12} & {}_1p_0 &= x_2 + d \cdot u_{32} \\ {}_0p_1 &= {}_0p_0 - g \cdot u_{12} & {}_1p_1 &= {}_1p_0 - g \cdot u_{32} \\ {}_0p_2 &= {}_0p_1 - h \cdot u_{12} & {}_1p_2 &= {}_1p_1 - h \cdot u_{32} \\ {}_0p_3 &= {}_0p_2 + k \cdot u_d & {}_1p_3 &= {}_1p_2 - k \cdot u_d \end{aligned}, \tag{18}$$

where u_d is the unit vector pointing from ${}_0p_2$ to ${}_1p_2$ and the weights, h, g, and k are defined as

$$h = v_3 d \;\; g = v_2 v_3 d \;\; k = \frac{6v_3 \cos(\frac{\gamma}{2})}{v_2+4} d\,. \tag{19}$$

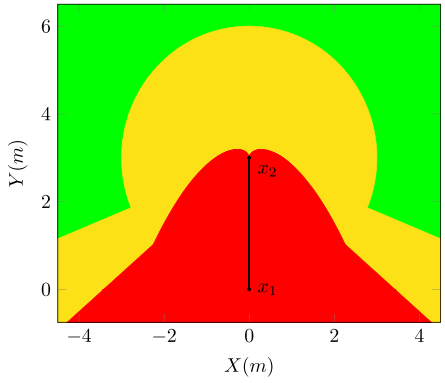
Table 2 Three points, x_1, x_2, x_3, were randomly sampled 1 million times with each x, y component bounded between 0 and 10 at each sample. For Dubin's paths, the orientation of a point was set to be tangential to the vector pointing to it from its parent. The average length of the resulting paths and time it took to find them is presented with their respective standard deviations

Motion Primitive	Length (m)	Computation Time (μs)
Straight-line	20.845 ± 7.417	3.371 ± 1.542
Dubin's path	23.797 ± 7.729	19.087 ± 8.670
Arc-fillet	18.838 ± 6.811	7.965 ± 2.762
Bézier-fillet	18.909 ± 6.242	15.568 ± 12.517

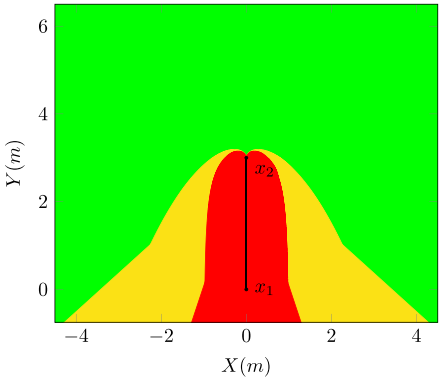
A procedure that differs from what is found in [23] is now defined to generate curves. In [23], a maximum curve angle, γ_{max}, is employed with an associated distance $d_{min} = d(\gamma_{max})$. To ensure subsequent fillet curves do not overlap, connecting points are forced to be $2d_{min}$ apart. We found the $2d_{min}$ node separation to be overly restrictive as small path refinements are not allowed under such a constraint. These small path refinements prove necessary, especially around curves in the obstacles. This limitation significantly reduces the ability to rewire, during which small refinements to the tree are common.

The underlying desired constraint enforced with the $2d_{min}$ spacing is path continuity. The conditions in (3) give the path generation process more flexibility. An example is given in Fig. 9a showing the points that could be considered under both the conditions in (3) and under a γ_{max} and $2d_{min}$ constraint. The curve generating procedure can now be stated.

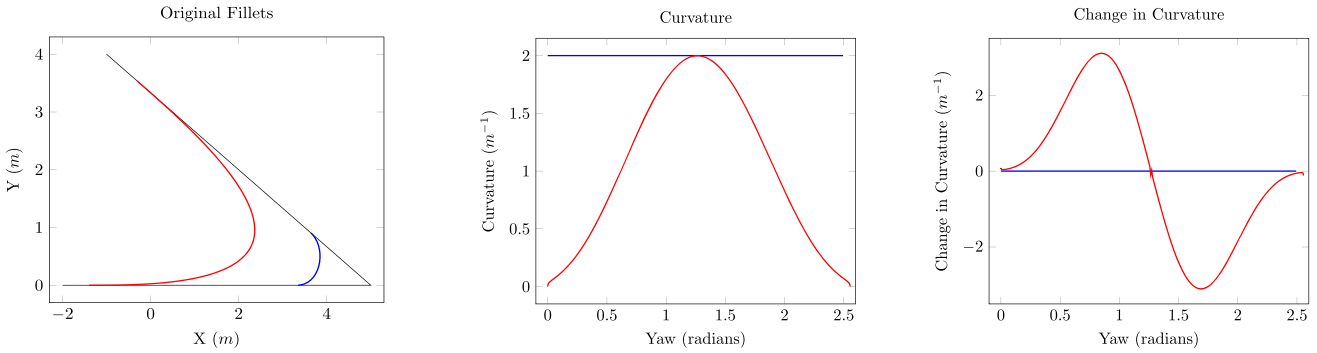
Fig. 9 Comparisons of different constraints on the position of x_3 if a fillet was made starting from x_1 through x_2 and to x_3. On the left, the constraints in [23] are compared to those found in (3) for the Bézier curve fillets. On the right, the difference in the definition of $d(\gamma)$ between arc-fillets and Bézier-fillets is expressed in terms of where (3) is satisfied. Both figures assume x_1 is at the origin, $x_2 = \begin{bmatrix} 0 & 3 \end{bmatrix}^T$, $\kappa_{max} = 2m^{-1}$, $d(\gamma_1) = 0$, $d_{min} = 1.5m$, and $\gamma_{max} = 0.624\pi$ radians



(a) Comparison of the constraints in [22] to those found in (3) for Bézier curve fillets. The red shows the area that both sets of feasibility conditions deem invalid for x_3. The yellow is area that only (3) deems valid. The green is area that both sets of conditions deem valid.



(b) Visualizes the reachability regions of the arc-fillet and Bézier-fillet generation. The red shows the area that both fillets cannot reach. The yellow is area that arc-fillets can reach but Bézier-fillets cannot. The green is area that both fillets can reach.



(a) Fillets generated between the points $\begin{bmatrix}-2 & 0\end{bmatrix}^T$, $\begin{bmatrix}5 & 0\end{bmatrix}^T$, and $\begin{bmatrix}-1 & 4\end{bmatrix}^T$

(b) The curvature of each fillet over the fillet curve

(c) The first derivative of the curvature of each fillet type

Fig. 10 An example of the arc and Bézier fillets with the corresponding curvature and curvature rate. The arc-fillet is shown in blue and the Bézier-fillet is shown in red. Note that the arc has zero curvature change as there is an instantaneous jump from zero to maximum curvature

Procedure 19 *($X_{fillet} \leftarrow BezierFillet(x_0, x_1, x_2, x_3)$) $BezierFillet$ uses the cubic Bézier spline to generate a $\mathcal{C}^2$ continuous curve from x_1 to x_3 using (15) through (17). Feasibility checks are made with respect to the curve that starts at x_0 and goes to x_2 as is described by the combination of (3) and (17). If the checks fail the null path is returned.*

3.5 A Comparison of Arc and Bézier Fillets

Arc and Bézier fillets provide different advantages and disadvantages. A big advantage of the arc-fillet is its simplicity and speed. As can be seen in Table 2, an arc-fillet can be generated in about half the time it takes to make a Bézier-fillet. Another benefit of the arc-fillet is that it is less constrained than the Bézier-fillet, resulting in a larger reachability set for connecting points, as shown in Fig. 9b. This fact is critical to RRT because it directly affects exploration and convergence.

The major advantage of the Bézier-fillet is the smoothness of the resulting path. The arc-fillet, like Dubin's paths, guarantees only $\mathcal{C}^1$ continuity of pose. The Bézir-fillet guaranties $\mathcal{C}^2$ continuity of pose and $\mathcal{C}^1$ continuity of curvature, as is shown in Fig. 10.

4 Fillet-based RRT*

Fillet-based variants of the RRT and RRT* algorithms are proposed in this section. The overall structure of the fillet-based variants is the same as their standard counterparts. This section will only cover the procedures that change, namely: $Initialize$, $CostToCome$, $Extend$, $Extend$*, and $Rewire$. To construct the fillet-based variants, each of these procedures is redefined with a procedure of the same name but with the prefix "FB," i.e. $Extend$ becomes $FB\text{-}Extend$. The first four procedures have small changes to their standard counterparts and are presented first. The $FB\text{-}Rewire$ procedure is then discussed in detail. This section ends with a discussion of FB-RRT*'s advantages. The FB-RRT* algorithm is not presented until Section 5 where a modified sampling procedure is discussed.

4.1 Procedures with Minor Changes

The first procedure to be modified is the $Initialize$ procedure. As defined, the $Initialize$ procedure has no consideration of the vehicle orientation, which must be respected to generate an executable path for the vehicle. $FB\text{-}Initialize$ differs from $Initialize$ in that it initializes the search tree to have an edge of length $d_{init} \in \mathbb{R}_+$ extending from the root, x_r, to a point in the direction of the initial orientation of the robot, $\psi_r \in [-\pi, \pi)$. Recall that connecting to a new point using a fillet requires both a parent and a grandparent node. If x_r is returned from $Nearest$ or $Near$ any attempt to make an edge with x_r is ignored as it has no parent, ensuring that the starting orientation is respected[2]. The new initialization procedure is defined as follows.

Procedure 20 *($T \leftarrow FB\text{-}Initialize_{d_{init}}(x_r, \psi_r)$) Returns a tree with two nodes and an edge of length d_{init} based on the root node, x_r, and initial orientation, ψ_r, as defined in Algorithm 8.*

The $CostToCome$ procedure is redefined to accommodate the new fillet-based path generation. Recalling Lemma 1, this depends upon both the parent and grandparent nodes of the node in question. The new procedure is given as follows.

[2] In our implementation, x_r is left out of the search in $Nearest$ and $Near$.

Algorithm 8 $T \leftarrow FB\text{-}Initialize_{d_{init}}(x_r, \psi_r)$.

$x_n \leftarrow x_r + d_{init}\left[\cos(\psi_r)\ \sin(\psi_r)\right]^T$
$V \leftarrow \{x_r, x_n\}$
$E \leftarrow \{(x_r, x_n)\}$
$T \leftarrow \{V, E\}$
return T

Algorithm 9 $c_n \leftarrow FB\text{-}CostToCome(x_n, x_p, x_{gp}, T)$.

$X_{fillet} \leftarrow Fillet(Parent(x_{gp}, T), x_{gp}, x_p, x_n)$
if $X_{fillet} \neq \varnothing$ & $CollisionFree(X_{fillet})$ **then**
    return $Cost(x_p, T) + c(X_{fillet}) - \left\|x_p - x_{gp}\right\|$ ▷ Path length calculation
else
    return ∞
end if

Procedure 21 *($c_n \leftarrow FB\text{-}CostToCome(x_n, x_p, x_{gp}, T)$) Calculates the cost of x_n if it is connected to the tree, T, through the parent, x_p, and the grandparent, x_{gp}, as in Algorithm 9. Note that the Fillet procedure in Algorithm 9 can be replaced by ArcFillet or BezierFillet depending upon the fillet being used.*

The *Extend* procedure is updated in two substantial ways. The first is the use of *FB-CostToCome*. The second is the use of a "node orientation" when performing the nearest neighbor search. By incorporating a sense of "node orientation", infeasible sharp turns can be avoided in the nearest neighbor searching. There is no actual "orientation' of a node as the nodes are 2D points that guide the creation of the path. However, we can bias the nearest neighbor search to penalize turns by noting that a fillet ending at a node will be oriented with the line extending from the node's parent to the node. After the fillet curve has been executed,

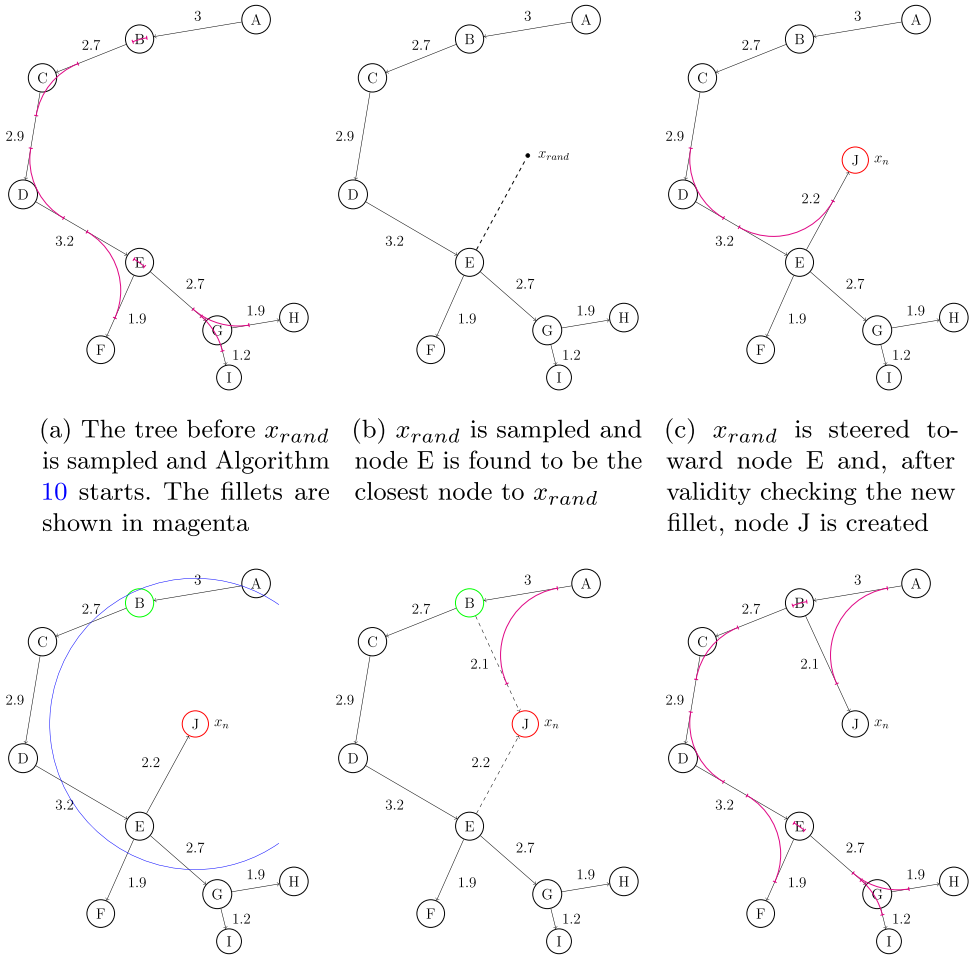


Fig. 11 An illustration of the *FB-Extend* and *FB-Extend** procedures

(a) The tree before x_{rand} is sampled and Algorithm 10 starts. The fillets are shown in magenta

(b) x_{rand} is sampled and node E is found to be the closest node to x_{rand}

(c) x_{rand} is steered toward node E and, after validity checking the new fillet, node J is created

(d) x_n's neighborhood set is found to be the singleton set of just node B

(e) The correctness of connecting to B is verified. Connecting through B and E results in costs of 5.1 and 14 respectively

(f) Because connecting through node B yields a lower total cost, node J is connected to node B. The fillets are shown in magenta

the robot will be aligned with that orientation. Given node x_i and its parent x_{i-1}, the orientation of x_i is defined as $\psi_i = atan2(u_{i-1\,i,2}, u_{i-1\,i,1})$. The nearest neighbor search is performed over $[\,x_{i,1}\ x_{i,2}\ \cos(\psi_i)\ \sin(\psi_i)\,]^T$ instead of $[\,x_{i,1}\ x_{i,2}\,]^T$. The inclusion of a pseudo "node orientation" combined with the relaxed continuity constraints, depicted in Fig. 9a, enables us to forgo the k-nearest-neighbor search that [21] uses to aid in the success of $Extend$. The updated procedure is now defined.

Algorithm 10 $\{x_n, x_p, c_n\} \leftarrow FB\text{-}Extend(x_{rand}, T)$.

$x_{nearest} \leftarrow Nearest(x_{rand}, T)$
$x_n \leftarrow Steer_\eta(x_{nearest}, x_{rand})$ ▷ Steer towards the nearest point
$x_{gp} \leftarrow Parent(x_{nearest}, T)$
$c_n \leftarrow FB\text{-}CostToCome(x_n, x_{nearest}, x_{gp}, T)$ ▷ Evaluate cost of path
if $\infty \neq c_n$ **then**
  return $\{x_n, x_{nearest}, c_n\}$
end if
return $\{\varnothing, \varnothing, \text{inf}\}$

Procedure 22 *($\{x_n, x_p, c_n\} \leftarrow FB\text{-}Extend(x_{rand}, T)$) Given $x_{rand} \in X$ and a tree T, the $FB\text{-}Extend$ procedure finds the closest vertex to x_{rand} in terms of a combined position and orientation metric and attempts to extend the tree in the direction of x_{rand}, as defined in Algorithm 10.*

The $FB\text{-}Extend^*$ procedure is identical to $Extend^*$ except for the use of $FB\text{-}Extend$ and $FB\text{-}CostToCome$. The $FB\text{-}Extend$ and $FB\text{-}Extend^*$ procedures are illustrated in Fig. 11. Note that the numbers shown in Fig. 11 are the edge costs of the edges they are near. The same is

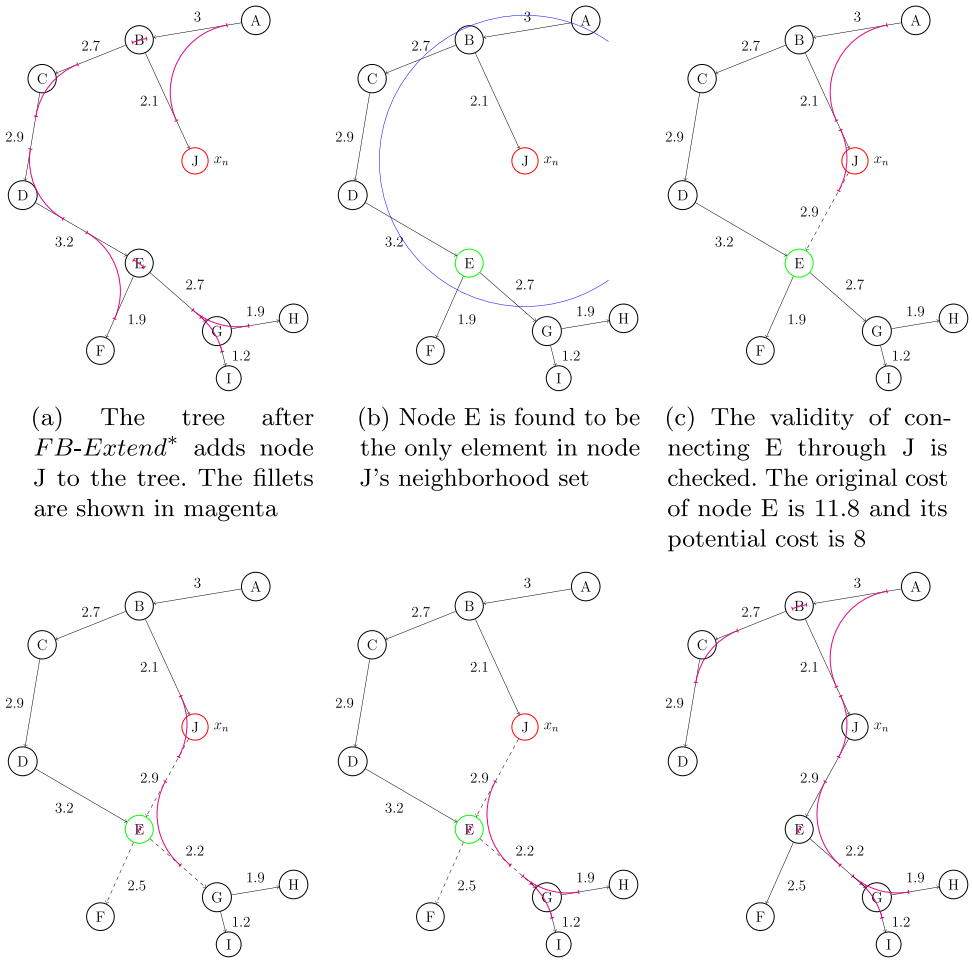


Fig. 12 An illustration of the $FB\text{-}Rewire$ procedure

(a) The tree after $FB\text{-}Extend^*$ adds node J to the tree. The fillets are shown in magenta

(b) Node E is found to be the only element in node J's neighborhood set

(c) The validity of connecting E through J is checked. The original cost of node E is 11.8 and its potential cost is 8

(d) The validity of E's children is checked. The original costs of F and G are 13.7 and 14.5 and their potential costs are 10.5 and 10.2 respectively

(e) The validity of E's grandchildren, nodes H and I, is checked with respect to connecting node E through node J

(f) Node E is rewired to have node J as its parent. The fillets are shown in magenta

true of Figs. 1, 2, and 3 except Fig. 11 uses the fillet cost calculation described in Lemma 1.

Procedure 23 *($\{x_n, x_p\} \leftarrow FB\text{-}Extend^*(x_{rand}, T)$) Given $x_{rand} \in X$ and a tree $T = \{V, E\}$, the $FB\text{-}Extend^*$ uses $FB\text{-}Extend$ to find a node for extending the tree and then finds the locally optimal path in V for connecting to the new point. The procedure is given in Algorithm 11.*

Algorithm 11 $\{x_n, x_p\} \leftarrow FB\text{-}Extend^*(x_{rand}, T)$.

$\{x_n, x_p, c_{min}\} \leftarrow FB\text{-}Extend(x_{rand}, T)$
if $x_n \neq \varnothing$ **then** ▷ If an extension is possible
  $X_{near} \leftarrow Near_{\rho,\alpha}(x_n, T)$
  for all $x_{near} \in X_{near}$ **do** ▷ Check for a lower cost connection
    $x_{gp} \leftarrow Parent(x_{near}, T)$
    $c_{tmp} \leftarrow FB\text{-}CostToCome(x_n, x_{near}, x_{gp}, T)$
    if $c_{tmp} < c_{min}$ **then**
      $x_p \leftarrow x_{near}$
      $c_{min} \leftarrow c_{tmp}$
    end if
  end for
  return $\{x_n, x_p\}$
end if
return $\{\varnothing, \varnothing\}$;

4.2 The Fillet-based Rewire Procedure

In the FB-RRT* framework, care must be taken to ensure both path feasibility and cost improvement when rewiring. Unlike its straight-line counterpart, it is not sufficient to choose a parent based purely on the path length to the node. The following lemma presents a set of sufficient conditions to ensure that a rewiring will not be detrimental to the tree.

Lemma 5 *Assume that a tree $T = \{V, E\}$ is given such that (3) is satisfied for all consecutive nodes. Rewiring E to make $x_n \in V$ the new parent of $x_{near} \in V$ will result in a continuous path with all node costs unchanged or lowered if the following three conditions are met:*

1. *The resulting path to x_{near} using x_n as its parent is obstacle free, does not violate (3), and the cost of x_{near} is improved, see* Fig. 12c.
2. *The resulting path to each child of x_{near} is obstacle free, does not violate (3), and the cost of the child is not increased, see* Fig. 12d.
3. *The resulting path to each grandchild of x_{near} does not violate (3), see* Fig. 12e.

Proof The only paths that will be affected by changing the parent of x_{near} will be the fillet connecting x_{near} to its grandparent and the fillets connecting x_n to the children of x_{near}. Conditions 1, 2, and 3 employ obstacle checking and (3) to ensure that fillet curves do not overlap in the new section of path nor with the preceding or subsequent sections of the path.

The path cost to x_{near} will be improved due to 1. The path costs to the children are not increased per 2. As all other parent and grandparent nodes remain unchanged, their respective path costs will not increase due to Corollary 2. □

The $FB\text{-}Rewire$ procedure is now stated and illustrated in Fig. 12.

Procedure 24 *($E \leftarrow FB\text{-}Rewire(x_n, X_{near}, T)$) Given a tree, $T = \{V, E\}$, with node $x_n \in V$ and set $X_{near} \subset V$, $FB\text{-}Rewire$ returns a modified tree with E changed to have x_n be the parent to elements of X_{near} if conditions in Lemma 5 are satisfied. The procedure is given in Algorithm 12.*

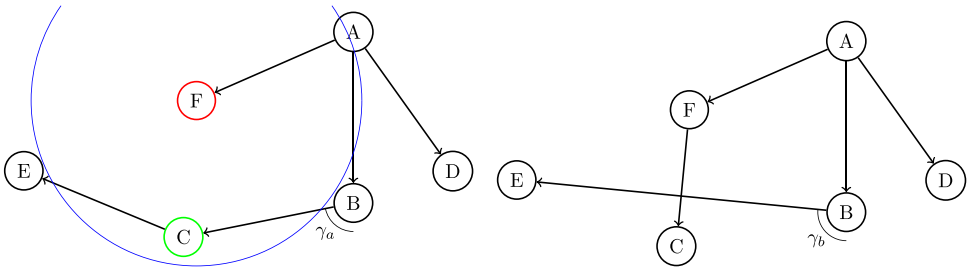
Algorithm 12 $T \leftarrow FB\text{-}Rewire(x_n, X_{near}, T)$.

for all $x_{near} \in X_{near}$ **do**
  $x_p \leftarrow Parent(x_n, T)$
  $c_{near} \leftarrow FB\text{-}CostToCome(x_{near}, x_n, x_p, T)$
  ▷ Check x_{near}'s feasibility and that its cost is reduced
  if $c_{near} \geq Cost(x_{near}, T)$ **then**
    goto Continue
  end if
  for all $x_c \in Children(x_{near}, T)$ **do**
    $c_c \leftarrow FB\text{-}CostToCome(x_c, x_{near}, x_n, T)$
    ▷ Check the feasibility and cost of x_{near}'s children
    if $c_c > Cost(x_c, T)$ **then**
      goto Continue
    end if
    for all $x_{gc} \in Children(x_c, T)$ **do**
      $X_{fillet} \leftarrow Fillet(x_n, x_{near}, x_c, x_{gc})$
      if $\varnothing = X_{fillet}$ **then** ▷ Check the feasibility of x_{near}'s grandchildren
        goto Continue
      end if
    end for
  end for
  $x_p \leftarrow Parent(x_{near}, T)$
  $E \leftarrow \left(E \setminus \{x_p, x_{near}\}\right) \cup \{x_n, x_{near}\}$ ▷ Rewire around x_{near}
  Continue:
end for
return T

Note that the $Rewire$ procedure described above is different than that in [21]. In [21], the neighborhood set of x_n, X_{near}, is checked to ensure that:

a. Connecting x_n and x_{near} will not violate their max angle and distance conditions.
b. The curve formed between x_n and x_{near} is obstacle free.
c. The cost of x_{near} will be improved by the rewire operation.

Thus, condition 1 is met (with a conservative continuity condition), but conditions 2 and 3 are not considered. If [21]'s conditions hold, the children of x_{near} are set to be the children of the parent of x_{near} and the parent of x_{near} is set to be x_n, see Fig. 13b. It is important to note that the $Rewire$



(a) The Node F is being rewired around the preexisting tree, and node C is F's neighborhood set

(b) Node C is rewired to have node F as its parent, and node E becomes a child of node B

Fig. 13 After node C has been rewired to node F the angle formed between nodes A, B, and B's child has increased, i.e. $\gamma_b > \gamma_a$. Without checking, there is no way to know if γ_b is less then the max angle allowed

operation in [21] could result in discontinuous paths due to not checking feasibility for all affected nodes. There is no guarantee that connecting the parent of x_{near} directly to the children of x_{near} will result in a valid tree. The angles and distances formed by that connection must first be checked as illustrated in Fig. 13. Moreover, due to not checking costs on all affected nodes, the $Rewire$ operation in [21] may actually increase costs to some nodes as shown in Lemma 2.

Note that the FB-RRT and FB-RRT* algorithms are not yet stated as additional improvements to the sampling and smoothing are first discussed in the following section. Also note that reverse motion considerations are discussed in Appendix.

5 Overcoming the Voronoi Property

Nonholonomic constraints exacerbate the slow convergence of RRT*. This section introduces two refinements to FB-RRT* designed to reduce convergence time. There are multiple variants of RRT* that aim to improve convergence by improving sampling [1, 4, 9, 16]. Two such variants are used as a basis for design herein: Informed RRT* (I-RRT*) [4] and Smart RRT* (S-RRT*) [16]. These algorithms are identical to RRT* before the first solution is found. However, after the first path to the goal is found, they use information about the solution to guide sampling toward space that will improve the final solution. In addition, S-RRT* introduces a path smoothing procedure that continuously refines the solution path, further improving convergence. This section first presents I-RRT* and S-RRT*. These techniques are then combined in the Fillet-based Smart and Informed RRT* (FB-SI-RRT*) formulation.

5.1 Informed RRT*

Informed RRT* (I-RRT*) is an extension of RRT* that aims to reduce the amount of time spent sampling space that will not improve the final path. It is observed that any time spent sampling outside of the set that will improve the path is wasted time. This set is referred to as the "informed" set and denoted as $X_i \subset X_{free}$. Naturally, it would be best to directly sample X_i. However, calculating X_i can be difficult, if not impossible.

A conservative approximation of X_i, denoted as X_i', is defined in [4] based upon an approximation of the problem's optimal cost and the best cost found, c_{best}. The optimal cost is denoted as c_{min} with its approximation denoted as c'_{min}. The approximation for c'_{min} is calculated as the distance between the root node and final state in the best-found path, i.e. $c'_{min} = \|x_r - x_t\|$. X_i' is defined as

$$X_i' = \mathcal{E}_{x_r,x_t} \tag{20}$$

where $\mathcal{E}_{x_r,x_t}$ is the open set of states in an ellipse with the focal points set at x_r and x_t. The length of the major axis of the ellipse is c_{best} and the length of the minor axis is $\sqrt{c_{best}^2 - c_{min}'^2}$, see Fig. 14. With this definition of X_i', it is guaranteed that the best solution found so far is entirely inside the informed set.

I-RRT* samples solely within X_i' once the first solution is found, reducing the configuration space sampled and improving the probability that a given sample will improve the solution. Note that I-RRT* is identical to RRT* with the exception of using X_i' instead of X_s for sampling new points

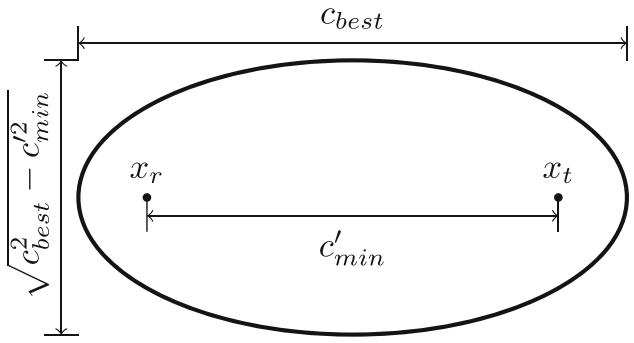


Fig. 14 The ellipse that defines X_i'

after the first solution is found. As long as $c'_{min} \leq c_{min}$, it is shown in [4] that I-RRT* maintains asymptotic optimality.

In an environment with small convex obstacles, I-RRT* is shown to converge much faster than RRT* [4]. If there are larger nonconvex obstacles then I-RRT* degrades in performance to RRT*. This is because the optimal cost, c_{min}, is much more than c'_{min}, causing the informed subset to be very large.

5.2 Smart RRT*

Smart RRT* (S-RRT*) proposes two alternative approaches to improving convergence [16]. First, similar to I-RRT*, the sampling set is reduced once a solution is found. Second, S-RRT* proposes a path smoothing procedure that is used as an integral part of the algorithm to further improve convergence.

The sampling set is reduced by biasing sampling around the nodes that form the shortest found path. The idea being that improvements near the path can help to refine the chosen route around obstacles. These nodes are referred to as the beacon set, X_b, and can be defined as

$$X_b = Solution(x_t, T)$$

where $x_t \in X_t$ is the end of the shortest path to the target set. Subsequent iterations then bias the sampling towards the union of balls of radius r_b around each beacon, i.e.

$$X_s = \bigcup_{x_b \in X_b} \mathcal{B}_{x_b, r_b}$$

A major philosophical difference between S-RRT* and I-RRT* is that S-RRT* biases the sampling around the beacon set whereas I-RRT* seeks to reduce the size of the sampling space. The biasing encourages local path refinement with a continued sampling of X for exploration.

Significant improvements are also made through the addition of a path smoothing procedure, referred to as $OptimizePath$ in [16]. $OptimizePath$ seeks to straighten out the path each time a better path is found. This avoids waiting for the sampling to straighten the path, something that becomes decreasingly probable as the number of samples increases due to the Voronoi property. The $OptimizePath$ procedure does this by performing rewire operations on each beacon with every other beacon, as detailed in Algorithm 13. Note that the $RemoveBetween$ procedure removes all of the nodes in X_b that lie between x_{near} and x_{ittr} not including x_{near} and x_{ittr}. This removes nodes from the solution/beacon set that are not needed. Which straightens out the solution and reduces the number of beacons that have to be sampled.

Before the first solution is found, S-RRT* performs identically to RRT*. However, after the first solution is found, the convergence of S-RRT* is much faster than RRT*, especially for straight-line motion primitives. It is important to note that S-RRT* has a tendency to spend more time converging on local minima than RRT*. In fact, if only the space near the beacon set were to be sampled after the first solution was found, then S-RRT* would lose its asymptotic optimality because it would only refine the first path found. We found that the beacon radius could be relatively small using straight-line paths, but needed to be increased significantly for curvature constrained paths.

Algorithm 13 $X'_b \leftarrow OptimizePath(X_b, T)$.

```
X'_b ← ∅
for all x_ittr ∈ X_b do
    for all x_near ∈ X_b ∩ X'_b do   ▷ Beacons that have not been used
        T ← Rewire(x_ittr, {x_near}, T)
        if Parent(x_near, T) = x_ittr then   ▷ Remove unneeded beacons
            X_b ← RemoveBetween(x_near, x_ittr, X_b)
        end if
    end for
    X'_b ← X'_b ∪ {x_ittr}
end for
return X'_b
```

5.3 Smart and Informed Sampling

I-RRT* performs well in small path planning problems where the obstacles are convex and c'_{min} is a good approximation of c_{min}. S-RRT* converges impressively fast, especially when planning using straight-line paths. Both techniques add parameters that need to be determined – an estimate of the best cost for informed and the beacon size for smart sampling. This work develops smart-and-informed sampling that combines S-RRT*'s fast convergence with an adaptive sample

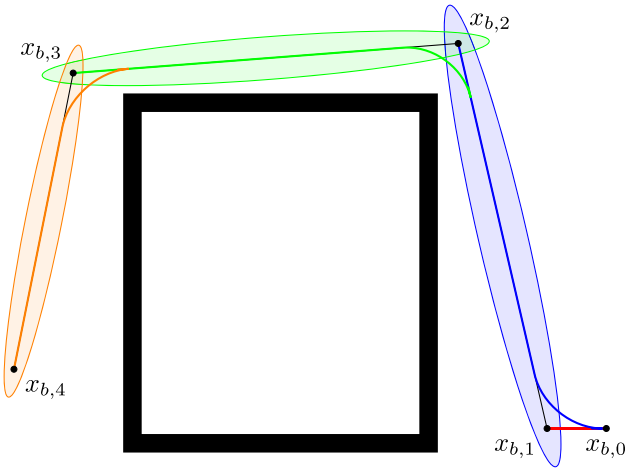


Fig. 15 The four nodes, $x_{b,0}$ through $x_{b,4}$, are connected with arc-fillets and each sampling ellipse, $\mathcal{E}_{x_{b,0},x_{b,1}}$, $\mathcal{E}_{x_{b,1},x_{b,2}}$, $\mathcal{E}_{x_{b,2},x_{b,3}}$, and $\mathcal{E}_{x_{b,3},x_{b,4}}$, is shown in red, blue, green, and orange respectively. Note that the volume of $\mathcal{E}_{x_{b,0},x_{b,1}}$ is 0, because $c_{best,0} = c_{min,0}$

set similar to I-RRT*. It is designed specifically for the non-holonomic motion primitives to provide a sampling heuristic that does not require fine tuning of additional parameters.

As is the case with both of its predecessors, the sampling will be identical to RRT* until the first solution is found. At that point, the $OptimizePath$ procedure, Algorithm 13, is called on the initial solution. Instead of using a constant radius around each beacon (S-RRT*), or an adaptive set based upon the optimality of the entire path (I-RRT*), the beacon set, $X_b \subset V$, is used to generate ellipses around each adjacent pair of beacons along the path that leads from x_r to X_t, as illustrated in Fig. 15. The axes of the ellipse are adapted based on the local optimality of the path, producing a larger sampling space when path refinement is needed and a smaller space when the local path is near optimal. Similar to the sampling in S-RRT*, the sampling after the beacons are found is biased towards the set formed by these ellipses,

$$\mathcal{E}_{X_b} = \bigcup_{i=0}^{|X_b|-1} \mathcal{E}_{x_{b,i},x_{b,i+1}} \subset X.$$

This sampling bias encourages path refinement. The full configuration space is still sampled periodically for sake of exploration.

For each beacon ellipse, $c'_{min,i}$ is defined as the distance between the two adjacent beacons that act as the focal points for that ellipse

$$c'_{min,i} = \|x_{b,i+1} - x_{b,i}\|$$

and $c_{best,i}$ is the cost differential across the two beacons:

$$c_{best,i} = Cost(x_{b,i+1}) - Cost(x_{b,i})$$

Note that for straight-line primitives $c_{best,i} = c'_{min,i}$, causing each ellipse to degenerate to the line between $x_{b,i}$ and $x_{b,i+1}$. While sampling along this line can be beneficial, we show an example where it slows convergence because it neglects the exploration half of the exploration-exploitation paradigm. With the fillets, the ellipse rarely degenerates to a straight-line. Even when it does the $OptimizePath$ procedure removes the redundant intermediary points. The smart-and-informed sampling procedure is now presented.

Procedure 25 *($x_{rand} \leftarrow SI\text{-}Sample(i, b_t, b_b, X_b, X_t)$) Returns a random point at iteration $i \in \mathbb{Z}_+$ given the sampling biases $b_t \in \mathbb{Z}_+$ and $b_b \in \mathbb{Z}_+$, the beacon set X_b, and the target set X_t as described in Algorithm 14.*

The $SI\text{-}Sample$ procedure includes a biasing towards the target set and a random sampling of the configuration space for sake of exploration. Additionally, as is the case with S-RRT*'s sampling, some percentage of the samples must be drawn from the full configuration space to maintain asymptotic optimality. This is because SI-RRT* makes no guaranties about its ellipses containing the optimal solution as I-RRT* does. Sampling within the ellipses of SI-RRT* and the beacons of S-RRT* has the effect of biasing the search to locally refine the current best path. The sampling of the beacon ellipses works in conjunction with the $OptimizePath$ procedure to seek improvements to the current best path found. The $OptimizePath$ procedure works to straighten paths while the sampling of the beacon ellipses works to see if local perturbations will improve path length.

Algorithm 14 $x_{rand} \leftarrow SI\text{-}Sample(i, b_t, b_b, X_b, X_t)$.

1: **if** $i \vdots b_t$ **then** ▷ Bias towards target set
2: $x_{rand} \leftarrow Sample(X_t)$
3: **else if** $X_b \neq \varnothing \wedge i \vdots b_b$ **then** ▷ Bias towards beacon ellipses
4: $x_{rand} \leftarrow Sample(\mathcal{E}_{X_b})$
5: **else** ▷ Otherwise sample a random point
6: $x_{rand} \leftarrow Sample(X)$
7: **end if**
8: **return** x_{rand}

5.4 Fillet-based Smart and Informed RRT*

With the $SI\text{-}Sample$ procedure in hand, all of the components are in place for presenting the Fillet-Based Smart and Informed RRT* (FB-SI-RRT*) algorithm. The FB-SI-RRT* algorithm is defined in Algorithm 15. The straight-line counterpart can be expressed by removing the FB prefix in all of the procedures. Furthermore, the fillet generation procedure used in $FB\text{-}CostToCome$ can be replaced with either the arc or Bézier fillets.

The FB-SI-RRT* algorithm is very similar to the traditional RRT* algorithm as defined in Algorithm 7. The major differences are the use of the fillet-based procedures in place of the straight-line counterparts and the $SI\text{-}Sample$ procedure in place of the $Biased\text{-}Sample$. The path optimization procedure from S-RRT* is also included in SI-RRT* but is absent in traditional RRT*.

The FB-SI-RRT* algorithm begins on line 1 by initializing the tree to consider the initial orientation of the vehicle and setting X_b and c_b to the empty set. As with RRT*, a prefixed number of iterations is specified for refining the path. Each iteration begins with the sampling of a new point using the $SI\text{-}Sample$ procedure. This balances biasing towards the target set to find the goal, biasing towards the beacon set to refine the best path found, and sampling from the general obstacle free space for exploration. The sampled point is then used within $FB\text{-}Extend^*$ in line 6 to grow the tree in the direction of the new sample while respecting the fillet continuity constraints. If a connection to the tree is found, a new node is then inserted into the tree on line 8. The edge set is then rewired

around the new node on line 9 using the $FB\text{-}Rewire$ to consider the fillet continuity and cost improvement requirements. If the first path or a shorter path to the target has been found, then the beacon set is updated in line 11. If c_b has changed, and thus X_b has changed, a $FB\text{-}OptimizePath$ procedure is used on line 14 in an attempt to refine the beacon set. Note that the $FB\text{-}OptimizePath$ has not been defined, but it can be expressed by changing line 4 of Algorithm 13 to use the $FB\text{-}Rewire$ procedure.

Algorithm 15 $X_{sol} \leftarrow FB\text{-}SI\text{-}RRT^*(x_r, \psi_r, X_t, b_t, b_b, n)$.

1: $T \leftarrow FB\text{-}Initialize_{d_{init}}(x_r, \psi_r)$
2: $X_b \leftarrow \varnothing$
3: $c_b \leftarrow \varnothing$
4: **for** $i = 1, \ldots, n$ **do**
5: $x_{rand} \leftarrow SI\text{-}Sample(i, b_t, b_b, X_b, X_t)$ ▷ Smart-and-informed sampling
6: $\{x_n, x_p, c_n\} \leftarrow FB\text{-}Extend^*(x_{rand}, T)$ ▷ Extend tree towards new point
7: **if** $x_n \neq \varnothing$ **then**
8: $T \leftarrow InsertNode(x_n, x_p, T)$ ▷ Add point to tree
9: $T \leftarrow FB\text{-}Rewire(x_n, Near_{\rho,\alpha}(x_n, T), T)$ ▷ Rewire edges around x_n
10: **if** $x_n \in X_t \wedge (X_b = \varnothing \vee Cost(x_n, T) < c(X_b))$ **then**
11: $X_b \leftarrow Solution(x_n, T)$ ▷ Update best path found
12: **end if**
13: **if** $X_b \neq \varnothing \wedge c_b \neq c(X_b)$ **then** ▷ If the best path has changed
14: $X_b \leftarrow FB\text{-}OptimizePath(X_b, T)$
15: $c_b \leftarrow c(X_b)$
16: **end if**
17: **end if**
18: **end for**
19: **return** X_b

Both an informed variation and smart variation to the FB-RRT* algorithm can be created with small variations to Algorithm 15. The informed algorithm can be formed by replacing line 5 with informed sampling and removing line 14. The smart algorithm can be formed by replacing line 5 with smart sampling.

6 Examples

A series of examples are now shown to demonstrate the Fillet-based RRT* approach. Three environments were chosen to illustrate the performance of various RRT*-based planners. Within each environment, 16 series of simulations were conducted to illustrate and evaluate the sampling and motion primitive variations. RRT*, I-RRT*, S-RRT*, and SI-RRT* planned using straight-line, arc-fillet, Bézier-fillet, and Dubin's path motion primitives.

Note that a comparison between motion primitive types is not meant to show that one motion primitive is better than another. Planning with straight-line paths is going to have better convergence characteristics due to their simplicity and lack of dynamic constraints. In fact, the only primitives that can be directly compared in this fashion are arc-fillets and the Dubin's paths as both assume the same dynamic constraints. The straight-line primitive is included to show a best-case scenario, providing a pseudo cost of considering the additional dynamic constraints. This section proceeds with details on the simulations followed by a description of the different environments. A comparison is made between the fillet formulation presented herein and that of [21] followed by an example that justifies the need to consider curvature constraints while path planning. Finally, the section ends with a discussion of the results.

6.1 Simulation Details

The obstacles are represented with an occupancy grid with each pixel corresponding to one square millimeter. When performing obstacle collision checks clearance of $0.5m$ is required on all sides. As orientation is well defined in the case of Dubin's paths and fillets, the points that are $0.5m$ to the right and left of the paths are checked. In the case of straight-line paths, orientation is not well-defined at the nodes so points are checked $0.5m$ in every cardinal direction. Paths are generated and checked at one-centimeter resolution.

The steering constant, η, and neighbor search radius, ρ, are both $3m$. The max number of neighbors to search, α, is 100. The check target period, b_t, is 50 and the target set, X_t, is a circle of radius $0.1m$. The Dubin's radius is set to $0.5m$ and likewise, the maximum curvature constraint imposed on the fillet generation is $2m^{-1}$. The root node to first node distance, d_{init}, is $1m$. Note that these values are somewhat aggressive for some curvature constrained applications, but they allow the straight-line primitive to provide a tighter bound on the possible performance characteristics of the curvature constrained planners. For S-RRT*, the beacon radius is $3m$ unless otherwise stated. Note that this is not necessarily the best choice of beacon radius, as shown in Fig. 21. However, smaller radius values are very detrimental to the Dubin's path results. The beacon bias, b_b, is 3 for both S-RRT* and SI-RRT*.

Results are gathered using OMPL [14]. Simulation code can be found in our open-source repository https://gitlab.com/utahstate/robotics/fillet-rrt-star. As the sampling is random, each simulation series consists of 100 individual simulations with the average results being presented. The results were gathered on an AMD Ryzen™Threadripper™2990WX processor. Convergence plots were made by fitting a 10^{th} order polynomial using a least-squares fitting algorithm as described in [20].

A least-squares approach is used as the sampling times for path length are not uniform across all simulations and not all simulations find the initial path at the same time. Note that while the path length for any one run will be monotonically decreasing with time, the least squares fitted plot does not

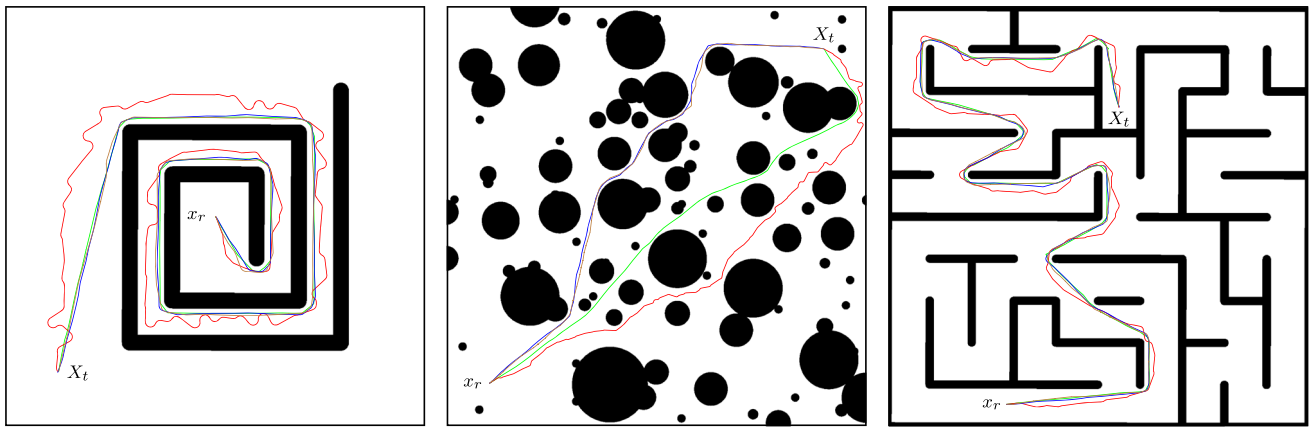


Fig. 16 The resulting paths from running RRT* for 30 seconds in the Spiral world (left), Cluttered world (middle), and the Maze world (right). Straight-line, Dubin's path, Arc-fillet, and Bézeir-fillet paths are shown in green, red, blue, and brown respectively

always have the same monotonic property. The reason is that a particular run may not find a solution until well after other runs and the initial solution it finds may be much larger than the current solution of the other runs, effectively causing the average to increase at the time the run first produces path length data.

6.2 Environments

The three environments shown in Fig. 16 were chosen to present and evaluate the performance of differing RRT* approaches. The environments are referred to as the Spiral world, the Cluttered world, and the Maze world.

The Spiral world is made up of one narrow passage that twists around the starting point. The world is $40m$ by $40m$, $x_r = [\,0\;0\,]^T$, and the center of X_t is at $[\,-15\;-15\,]^T$. The only path from x_r to X_t is through a narrow hallway forming a "bug trap" like set of obstacles. The Spiral world tests planners' abilities to find a way out of the confined starting area and then converge through all of the passageways. The "bug trap" like design makes it difficult for Dubin's paths based planners to find an initial solution, and I-RRT*'s cost heuristic is a poor estimate of c_{min} in this environment. The environment is well suited for smart sampling as there is only one path and refinements are beneficial at each turn.

The Cluttered world is composed of many overlapping circular obstacles. The world is $100m$ by $100m$, $x_r = [\,-40\;-40\,]^T$, and the center of X_t is at $[\,40\;40\,]^T$. The abundance of small obstacles results in many small local minima, but there are still large open areas for exploration. The Cluttered world tests the planners' ability to break out of local minima. I-RRT* based sampling is well-suited in the environment as the I-RRT* heuristic is a good estimate of the optimal path length.

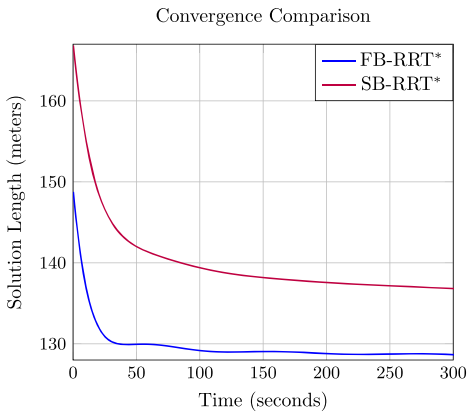
The Maze world features a series of narrow passages and dead ends. The world is $50m$ by $50m$, $x_r = [\,-11\;-22.5\,]^T$, and the center of X_t is at $[\,2.5\;12.5\,]^T$. The Maze world has fewer local minima than the Cluttered world and consists of long narrow corridors. This world tests the planners' ability to find high-quality initial solutions quickly and then converge past those initial solutions. While the local minima can be detrimental to beacon-based sampling, the optimal cost heuristic in I-RRT*'s sampling is poor in this case, proving detrimental to I-RRT*'s convergence.

6.3 Comparison with Previous Work

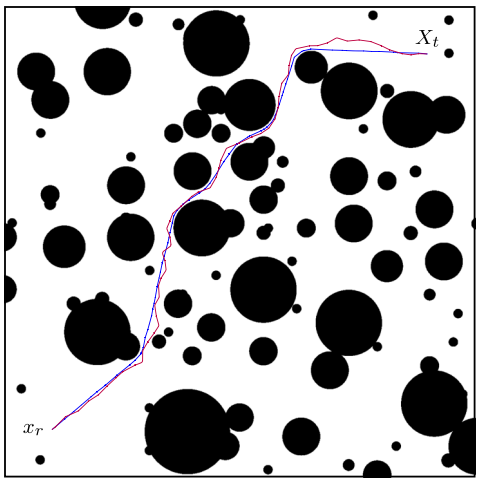
Note that this work's FB-RRT* differs from what is given in [21, 23] in the following ways:

- The generalization of [21, 23]'s γ_{max} and d_{min} based path continuity constraints to the less restrictive form given in (3). See Fig. 9a for an illustration of how (3) is less restrictive than using the γ_{max}/d_{min} constraints.
- The addition of a pseudo "node orientation" in the nearest neighbor search heuristically penalizes turns and enables us to forgo the k-nearest-neighbor search that [21] uses. See the explanation of $FB\text{-}Extend$ in Section 4.1 for more information.
- A newly developed rewiring procedure that ensures continuity and cost improvement in the resulting path. See Section 4.2 for more information on $FB\text{-}Rewire$.
- The generalization of the fillet-based planner structure to make use of any fillet type instead of just Bézeir-fillets.

This section uses convergence plots to compare the formulation of the fillet constraints in this work to the formulation given in [21, 23]. Specifically, instead of constraining node addition in the tree with (3) we use the constants given in [23]. [23] defines a max node-to-node angle, γ_{max}, and then uses that angle to define a minimum node-to-node distance, $d_{min} = d(\gamma_{max})$. Any nodes that form an angle greater



(a) Convergence of solution cost in the Cluttered world over 5 minutes. FB-RRT* is shown in blue and SB-RRT* is shown in purple



(b) Example solutions found in the Cluttered world after 5 minutes. FB-RRT* is shown in blue and SB-RRT* is shown in purple

Fig. 17 A comparison of the solution quality found by FB-RRT* and SB-RRT*. FB-RRT* converges faster and to a shorter solution then SB-RRT*. FB-RRT* is shown in blue and SB-RRT* is shown in purple

than γ_{max} or are closer together than $2d_{min}$ are deemed invalid. This is a conservative approximation of the constraints defined in (3), see Section 3.4 for more details. We refer to the version of FB-RRT* that uses [23]'s constraints as SB-RRT*.

Note that SB-RRT* differs from what is given in [21] because the $FB\text{-}Rewire$ procedure is used to avoid the invalid tree configurations that result from the $Rewire$ procedure in [21], see Fig. 13. The Bézeir-fillet is used for comparison because that is the fillet used in [21, 23]. The Cluttered world was chosen for this simulation because it is similar to the simulated environment used in [23]. The max allowed curvature is kept at $2m^{-1}$, $\gamma_{max} = \frac{\pi}{2} rad.$, and $d_{min} = 0.7938m$.

Figure 17a shows the convergence of SB-RRT* and FB-RRT* in the scenario described. FB-RRT* far outperforms SB-RRT* in both initial convergence speed and the solution to which it settles over time. FB-RRT* converges to a much shorter path then SB-RRT* as the constraints given in [23] force each node along the solution to be at least $2d_{min}$ distance away from each other. As the solution converges, it becomes difficult to shorten the path further without reducing the number of nodes that make up the solution path. We emphasize that this simulation only shows the benefit of using the constraints in (3). A significant benefit is also received from the updated rewiring procedure that ensures continuous paths are produced.

6.4 Curvature Constrained Paths

This section provides an example of when considering curvature constraints during path planning is advantageous. A common approach to curvature constrained path planning is to plan using straight-line motion primitives and then smooth the path after planning; see, for example, Chapter 11 of [2]. However, this can lead to invalid paths.

One of the scenarios where this would be a problem is shown in Fig. 18. Path planning using straight-line primitives converges to the solution shown in green. This is the shortest path between the start and goal locations while ensuring a minimum distance is maintained between the path and obstacles.

However, a path that goes through this hallway can not satisfy the curvature constraints of the problem without hitting the walls or performing a complex, multi-turn maneuver. See Appendix for details on generating a multi-turn maneuver with FB-RRT*. If curvature constants are considered during

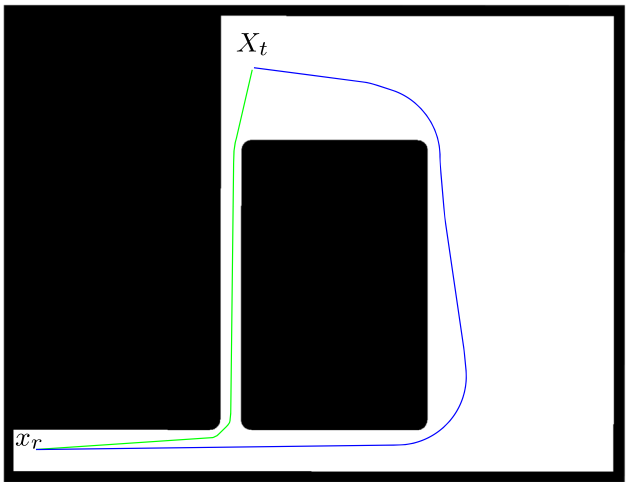


Fig. 18 The resulting paths from running RRT* in an environment where the straight-line path turns sharply down a corridor. Straight-line and arc-fillet paths are shown in green and blue respectively

Table 3 Initial solution results

Motion Primitive	Spiral	Cluttered	Maze
(a) The means and standard deviations of the time in seconds it takes for RRT* to find an initial solution.			
Straight-line	4.328 ± 1.158	0.498 ± 0.196	11.183 ± 3.455
Dubin's path	96.792 ± 38.124	1.585 ± 1.357	40.849 ± 17.587
Arc-fillet	6.373 ± 2.124	0.586 ± 0.527	7.473 ± 2.057
Bézeir-fillet	10.288 ± 3.292	1.081 ± 1.966	14.002 ± 4.04
(b) The means and standard deviations of the initial path length in meters after RRT* has found an initial solution.			
Straight-line	136.05 ± 3.91	167.42 ± 12.14	142.25 ± 7.84
Dubin's path	161.68 ± 7.73	275.07 ± 39.55	162.65 ± 9.86
Arc-fillet	144.45 ± 6.47	184.54 ± 21.81	148.56 ± 8.4
Bézeir-fillet	137.76 ± 5.45	181.45 ± 23.76	147.2 ± 8.03

path planning, as is the case for FB-RRT*, then the path planner will return a solution through the second corridor, which is wide enough to make the turn.

6.5 Results

The different motion primitives are now compared in terms of the initial solution, the convergence, and the effects of the various sampling techniques.

6.5.1 Initial Solution Comparison

Table 3 shows the performance of each motion primitive in finding the initial solution in terms of the mean time and length of the initial solution for each environment. The results in Table 3 apply regardless of the sampling technique used as each sampling approach behaves identically to RRT* before the first solution is found.

The Spiral and Maze worlds take longer on average to find an initial solution than the Cluttered world. There are a couple of reasons that this may occur. Both worlds have more narrow passages, resulting in a large number of iterations that fail to extend the tree because their paths become invalidated by obstacles. Furthermore, the Maze world has many dead-ends, allowing the tree to spend time growing in a direction that will not lead to the target set.

Planners using the arc-fillet primitive find an initial solution in a time comparable to that of the planners using straight-line paths in all of the environments considered, performing slightly faster in the Maze world. Similarly, the path lengths found are comparable, with the straight-line based paths being slightly shorter in all cases. The added cost of considering continuity in curvature is seen as the planners using Bézier-fillets require roughly twice the time to find an initial solution as their arc-fillet counterparts in all three environments. Table 3b shows the Bézeir-fillet based planners finding a slightly shorter initial solution than their arc-fillet counterparts. This may be due to the fact that the Bézier-fillet planners take more time and iterations to find an initial solution. During that same time, the planners using arc-fillets are refining their solutions, always having a shorter path than the Bézier-fillet planners.

It takes significantly longer for planners using Dubin's paths to find an initial solution in each of the worlds tested here. The only world where Dubin's paths based planners found an initial solution in a comparable amount of time is the Cluttered world, although the initial path is also significantly longer. This is, in part, due to loops in the path, similar to that depicted in Fig. 4. In the Cluttered world, the first path found has many loops and bends, resulting in a longer path length on average. These loops prove even more detrimental in the Spiral and Maze worlds with their narrow passageways. The path length does not increase as dramatically for Dubin's paths in the Spiral and Maze worlds as it does in the Cluttered world because there is not as much room for looping paths. However, there is a significant impact to the initial solution time as planning with Dubin's paths requires over 2 to 9 times as long as Bèzier-fillets and 5 to 15 times as long as arc-fillets.

It is important to note that the least squares fitting distorts the average transients plots in Fig. 19. The average initial solution length for each planner is equivalent for any particular motion primitive. However, the rapid convergence of some planners cause the least squares solution to appear lower at the initial solution time.

6.5.2 Convergence Times of Motion Primitives

Once an initial solution is found, the algorithms work to converge on the shortest path. In this section, the resulting path lengths generated from planning using different motion primitives are directly compared. Note that this is inherently an unfair comparison as the primitives have different kinematic constraints. The only two motion primitives that use the same kinematic constraints are Dubin's paths and arc-fillets. The

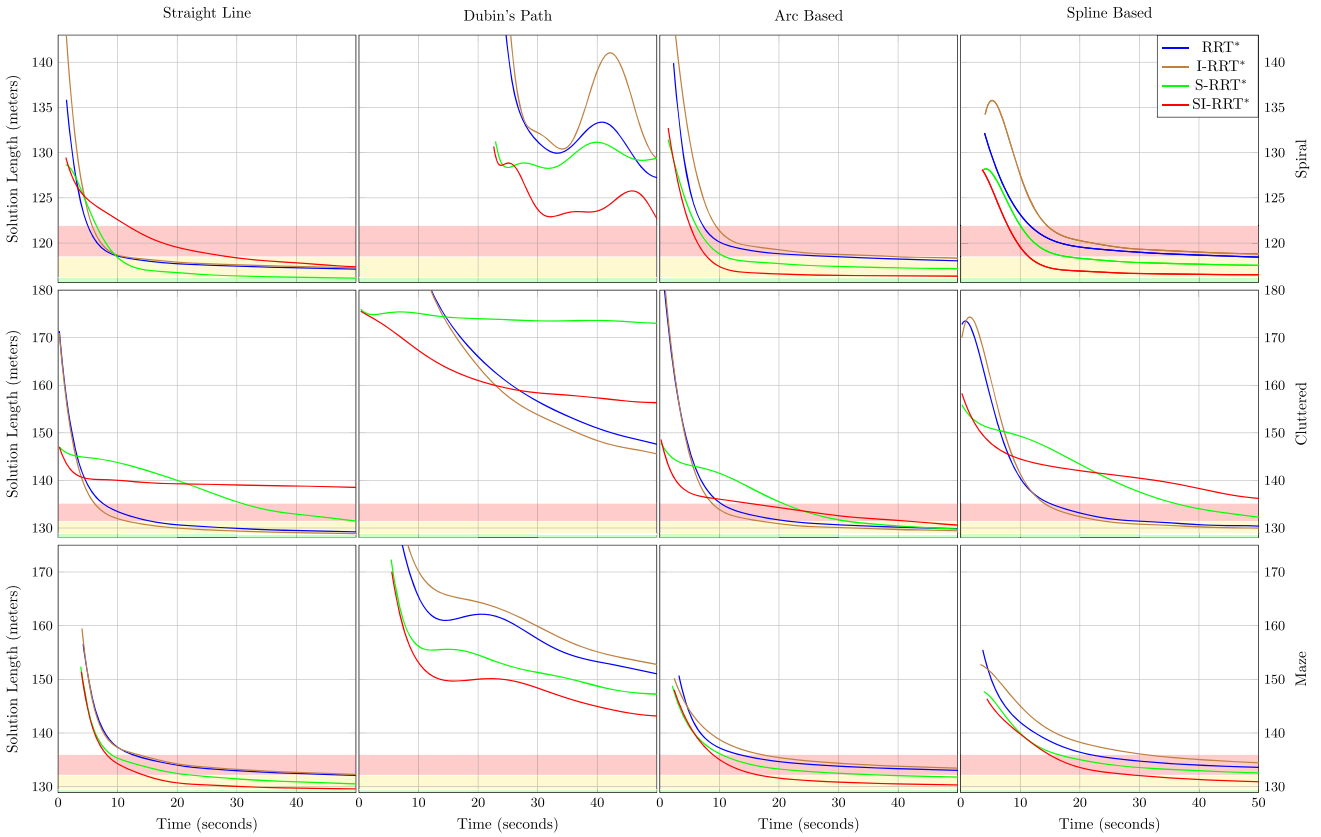


Fig. 19 The convergence plots of the Spiral world on top, Cluttered world in the middle, and the Maze world on bottom over 50 seconds. Images from left to right show the transients of the straight-line, Dubin's path, arc-fillet, and Bézeir-fillet motion primitives. RRT* is shown in blue, I-RRT* in brown, S-RRT* in green, and SI-RRT* in red. From the top going down, the red shaded area starts once the solution length is within 5 percent of the best averaged path found after 50 seconds. The yellow shading starts withing 2 percent, and the green shading starts at the best solution length found

straight-line motion primitive does not respect curvature constraints while Bézeir-fillet primitive considers a curvature rate constraint in addition to the curvature constraint considered by the arc-fillet primitive. Each scenario has been designed such that the optimal path for each motion primitive will be similar, unlike the scenario in Fig. 18. This allows the straight-line convergence rate to be a pseudo best-case solution. The results will show that the fillet-based planners have comparable results to the straight-line planners.

Figure 19 shows the performance of each motion primitive with the four different sampling techniques. The green shaded areas start at the path length of the best averaged solution found for that world in 50 seconds. The yellow shaded area starts once the path length is within 2 percent of the best averaged solution for the respective environment. Similarly, the red shaded area starts once the path length is within 5 percent of the best averaged solution for the respective environment.

Table 4 shows the best performing planners in each world and type of motion primitive. In the Spiral world, the best averaged solution found within 50 seconds was found by S-RRT* planning with straight-lines with a path length of $116.14m$. For the Cluttered world, the best was found by I-RRT* planning with straight-lines with a path length of

Table 4 The best performing planner and associated average path length for that planner in each environment and motion primitive combination

Motion Primitive	Spiral		Cluttered		Maze	
Straight-line	S-RRT*	116.14	I-RRT*	128.79	SI-RRT*	129.54
Dubin's path	SI-RRT*	122.6	I-RRT*	145.54	SI-RRT*	143.17
Arc-fillet	SI-RRT*	116.34	I-RRT*	129.46	SI-RRT*	130.31
Bézeir-fillet	SI-RRT*	116.49	I-RRT*	129.98	SI-RRT*	130.92

Results are calculated after running the planners for 50 seconds

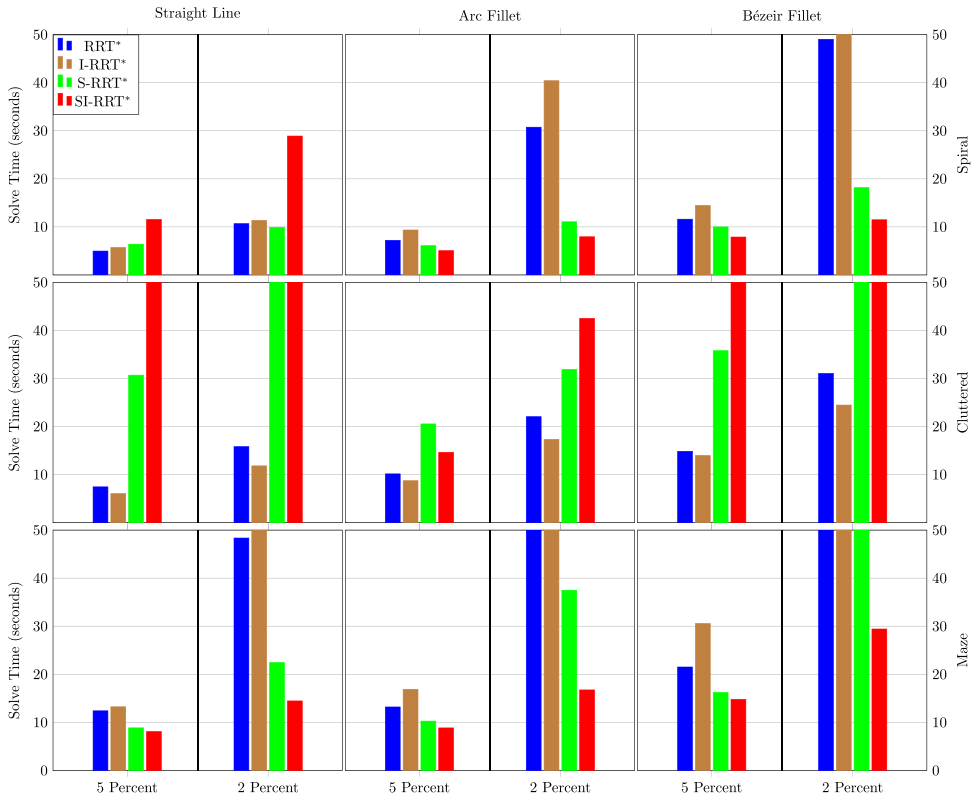


Fig. 20 The average time it took each planner to find a solution that was within 5 and 2 percent of the best averaged solution found after 50 seconds. Images from left to right show the results of the straight-line, arc-fillet, and Bézeir-fillet motion primitives. RRT* is shown in blue, I-RRT* in brown, S-RRT* in green, and SI-RRT* in red

128.79m. For the Maze world, the best was found by SI-RRT* planning with straight-lines with a path length of 129.54m. Note that in each case, the planner that found the shortest averaged solution was planning with straight-line motion primitives. This is expected because the straight-line primitive is the least constrained.

Figure 20 shows the 5 and 2 percent convergence times in a bar graph for a quick comparison of results. It is worth noting that planners using the Dubin's path primitive struggle to approach the 5 percent convergence region and are subsequently left out of Fig. 20. On the other hand, the arc-fillet planners perform quite well despite assuming the same motion constraints as the Dubin's primitive. The arc-fillet based planners perform comparably, and in some cases better, than the straight-line primitives in converging to the 5 percent threshold. Convergence begins to suffer for the 2 percent threshold, with some arc-fillet planners unable to cross that threshold in the Maze world.

The Bézier-fillet planners show an increase in convergence time, underscoring the cost of requiring continuity in curvature. SI-RRT* is the only planner that is able to cross the 2 percent threshold when planning with Bèzier-fillets in the Maze world. However, when planning with Bézier-fillets in the Cluttered world, the smart sampling techniques (S-RRT* and SI-RRT*) fail to cross the 2 percent threshold while both RRT* and I-RRT* are able.

6.5.3 The Effect of Sampling Techniques

The environments have little effect on the trends of ranking the performance of the motion primitives. The planning with straight-lines typically outperforms the arc-fillets, which outperforms the Bèzier-fillets, which in turn outperforms planning with Dubin's paths. However, the environment has a significant effect on the performance of the sampling procedures.

Table 4 shows that each environment has a sampling procedure that works best in that environment. For the Cluttered world, I-RRT* performs the best across all motion primitives. Whereas in the Spiral and Maze worlds, the greedier sampling heuristics perform better. In the Maze world, SI-RRT* performs the best for all motion primitives. In the Spiral world, S-RRT* performs the best when planning with straight-lines but SI-RRT* performs best for all other motion primitives. This shows that S-RRT*'s sampling works very well when planning with straight-lines but tends to struggle more when planning with kinematic constraints.

As mentioned, the Cluttered world is ideal for I-RRT*'s sampling approach and, as expected, the I-RRT* based planners perform the best in terms of 5 and 2 percent convergence. However, the transient plots in Fig. 19 show a very interesting trend. The smart approaches (S-RRT* and SI-RRT* based planners) show significantly faster initial convergence.

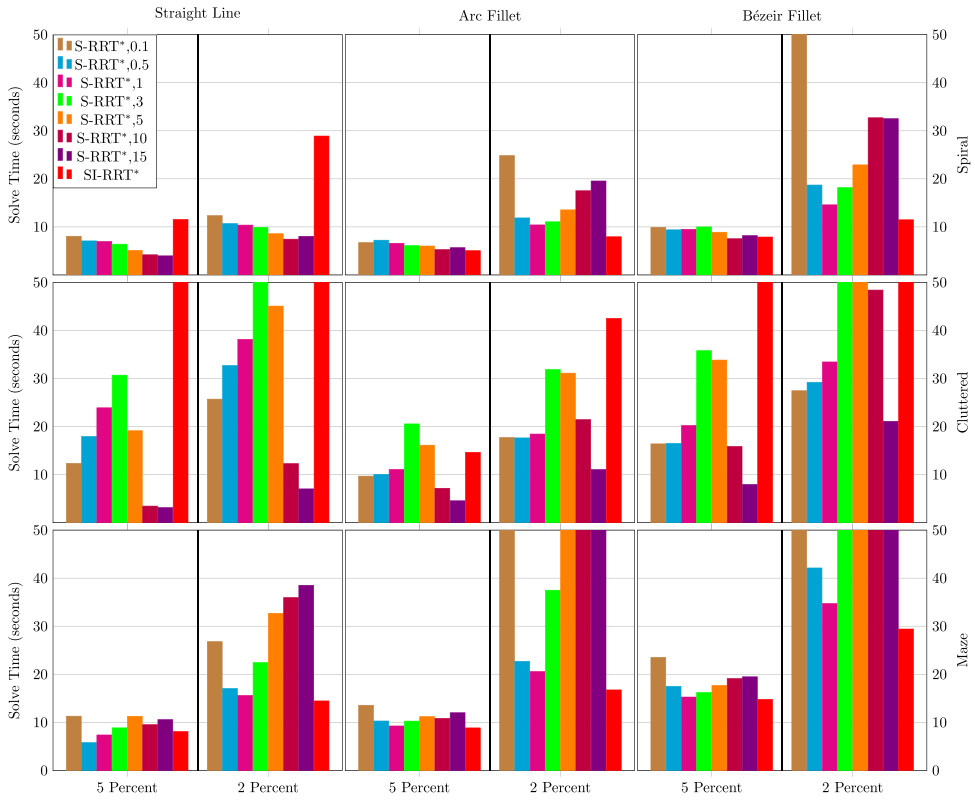


Fig. 21 The average time it took each planner to find a solution what was within 5 and 2 percent of the best averaged solution found after 50 seconds. Images from left to right show the results of the straight-line, arc-fillet, and Bézeir-fillet motion primitives. S-RRT* with a beacon sizes of $0.1m$, $0.5m$, $1m$, $3m$, $5m$, $10m$, and $15m$ are shown in brown, cyan, magenta, green, orange, purple, and violet respectfully. SI-RRT* is shown in red

This is due to the fact that the smart approaches focus on refining the best solution instead of searching the environment, which also means that they tend to spend more time in local minima. This is particularly noticeable in the Cluttered world where the smart approaches plateau for a time before dropping again. The Spiral and Maze worlds present environments in which there are fewer local minima and as such the smart approaches continue rapidly refining the solution until the 5 and 2 percent thresholds.

Figures 19 and 20 show that the SI-RRT* based planners focus heavily on refining the current shortest solution. In the Cluttered world, this proves detrimental as it causes the planner to focus on local minima instead of searching for other paths through the obstacle topology. SI-RRT*'s greedy convergence to local solutions is why SI-RRT*'s solution cost plateaus in the Cluttered world, see Fig. 19. In the Maze and Spiral worlds, it proves beneficial and results in the fastest convergence, both initially and to the 5 and 2 percent thresholds for all but the straight-line motion primitives. This may be because there are fewer local minima in the obstacle topology of these worlds. As expected, the smart-and-informed sampling does quite poorly for straight-line primitives.

Figure 21 shows that the SI-RRT* approach provides greedy refinement for fillet-based primitives without requiring the tuning of an extra beacon-size parameter. Figure 21 compares the results from S-RRT* based planners over various beacon sizes to the results of the respective SI-RRT* based planner. In the Cluttered world, SI-RRT* spends most of its time refining local minima and results in the worst convergence times. In both the Maze and the Spiral worlds, the smart-and-informed sampling performs near the best, except when considering straight-line primitives. While an iterative search over beacon sizes for S-RRT* can produce similar convergence results to SI-RRT*, the smart-and-informed sampling does not require additional tuning to find the best beacon size. Note that the resulting best beacon size for S-RRT* sampling is dependent upon both the environment and the motion primitive, making such a search difficult prior to execution.

7 Conclusion

In this work, an RRT-based path planning algorithm is proposed that uses general fillets as motion primitives. An arc-fillet is designed to provide path continuity similar to a Dubin's path while a Bèzier-fillet is developed to provide continuity in path curvature. RRT*-like procedures are developed to accommodate fillet-based motion primitives. Simulation results show that planning with arc-fillets significantly outperforms the use of Dubin's paths as a motion primitive. Planning with arc-fillets is shown to perform almost as well as straight-line motion primitives. Planning with Bèzier-fillets exhibitsslightly worse performance

than arc-fillets, although it far outperformed planning with Dubin's despite considering more complex dynamic constraints. A comparison is made between informed sampling, smart sampling, and a new smart-and-informed sampling technique. Like their straight-line counterparts, the fillet-based planners perform better with informed sampling when the heuristic for the shortest path is valid and better with smart sampling when there are fewer local minima. The smart-and-informed sampling performed well for fillet-based motion primitives and was found to be most applicable to environments with fewer local minima. In such environments, it performed on par with the best smart beacon size without the need for an iterative search for that beacon size.

Author Contributions All authors contributed to the work's conception and design. Software development, material preparation, and data collection were performed by James Swedeen. Data analyses were performed by James Swedeen and Dr. Greg Droge. James Swedeen is the primary author with major editorial and conceptual contributions made by Dr. Greg Droge and Dr. Randall Christensen at various stages of the writing process. All authors read and approved the final manuscript.

Funding The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Code Availability Simulation code can be found in our open-source repository https://gitlab.com/utahstate/robotics/fillet-rrt-star.

Declarations

Competing Interests The authors have no relevant financial or non-financial interests to disclose.

Appendix: Reverse Fillet

This section describes a novel reverse fillet formulation that enables the ability to plan paths that go forward and backward. The reverse fillet formulation uses a generic one-directional fillet internally that can be replaced by any fillet primitive desired. This section ends with an example planning problem that necessitates reverse and forward motion to follow the shortest path possible.

When using the reverse fillet motion primitive the direction state, d, is added to the state space. $d = 1$ when the vehicle is moving forward and $d = -1$ when the vehicle is moving backward. It can be determined if a fillet should keep the direction of travel the same or change it by comparing the d values of the nodes that the fillet connect. When using FB-RRT* with the reverse fillet formulation, d is randomly sampled from a uniform distribution when the state space is sampled.

Before the $ReverseFillet$ procedure can be given a few pieces of notation must be defined. We use the notion $x_{0,d}$ to denote the d value of node x_0 and $x_{0,xy}$ to denote the position vector of node x_0. The function $R(\cdot)$ consumes an angle and produces a two-by-two right handed rotation matrix.

Algorithm 16 $X_{fillet} \leftarrow ReverseFillet(x_0, x_1, x_2, x_3)$.

1: **if** $x_{2,d} = x_{3,d}$ **then** ▷ Fillet with uniform path direction
2: $X_{unidir} \leftarrow Fillet\left(x_{0,xy}, x_{1,xy}, x_{2,xy}, x_{3,xy}\right)$
3: $X_{fillet} \leftarrow \left\{\begin{bmatrix} x_{u,x} & x_{u,y} & x_{3,d}\end{bmatrix}^T, x_u \in X_{unidir}\right\}$
4: **else** ▷ Fillet with switching path direction
5: ▷ Rotation matrix for the orientation of x_2
6: $R_2 \leftarrow R\left(-atan2\left(x_{2,y} - x_{1,y}, x_{2,x} - x_{1,x}\right)\right)$
7: ▷ Put each point in the frame with x_2 at the origin and x_1 on the x-axis
8: ${}_2x_0 \leftarrow R_2 \cdot \begin{bmatrix} x_{0,xy} - x_{2,xy}\end{bmatrix}$
9: ${}_2x_1 \leftarrow R_2 \cdot \begin{bmatrix} x_{1,xy} - x_{2,xy}\end{bmatrix}$
10: ${}_2x_2 \leftarrow \begin{bmatrix} 0 & 0\end{bmatrix}^T$
11: ${}_2x_3 \leftarrow R_2 \cdot \begin{bmatrix} x_{3,xy} - x_{2,xy}\end{bmatrix}$
12: ${}_2x_{3f} \leftarrow \begin{bmatrix} -{}_2x_{3,x} & {}_2x_{3,y}\end{bmatrix}^T$ ▷ Flip ${}_2x_3$ across y-axis
13: $X_{unidir} \leftarrow Fillet\left({}_2x_0, {}_2x_1, {}_2x_2, {}_2x_{3f}\right)$ ▷ Make unidirectional fillet
14: ▷ Flip any part of the fillet that is in front of x_2 over the y-axis
15: $X_{fillet} \leftarrow \left\{\begin{cases} \begin{bmatrix} x_{u,x} & x_{u,y} & x_{2,d}\end{bmatrix}^T & \text{for } x_{u,x} \le 0 \\ \begin{bmatrix} -x_{u,x} & x_{u,y} & x_{3,d}\end{bmatrix}^T & \text{for } x_{u,x} > 0 \end{cases}, x_u \in X_{unidir}\right\}$
16: ▷ Move the fillet back to the original coordinate frame
17: $X_{fillet} \leftarrow \left\{\begin{bmatrix} R_2^T x_{f,xy} + x_{2,xy} \\ x_{f,d}\end{bmatrix}, x_f \in X_{fillet}\right\}$
18: **end if**
19: **return** X_{fillet}

Algorithm 16 gives the procedure for generating a reverse fillet and Fig. 22 illustrates the process. If the direction of travel for the two points being connected, x_2 and x_3, are the same then a normal fillet can be made to connect them, see lines 2 and 3. If the direction of travel changes, more logic is needed to make use of a unidirectional fillet primitive to make a fillet that changes direction. The process, shown on lines 6 through 17, involves flipping x_3 over the plane that intersects x_2 and is perpendicular to $\overline{x_1 x_2}$ to produce a new point, x_{3f}. An unidirectional fillet can be designed using x_{3f}. The fillet to execute is obtained by flipping the portion of the unidirectional fillet from x_2 to x_{3f} back so that the path ends at x_3. The portion from x_2 to x_3 will then be executed in the opposite direction of that from x_1 to x_2.

First x_0, x_1, and x_3 are transformed into a coordinate frame with x_2 at the origin and x_1 on the x-axis, see lines 6 through 11 and Fig. 22b. This frame is defined to make flipping x_3 easier. The prescript 2 is used to denote a point in this frame, i.e., ${}_2x_0$ is the point $x_{0,xy}$ in the new frame. On line 12, ${}_2x_3$ is flipped across the y-axis, see Fig. 22c. The transformed set of points ${}_2x_0$, ${}_2x_1$, ${}_2x_2$, and ${}_2x_{3f}$ form a chain of nodes that do not change direction. Line 13 generates a unidirectional fillet called X_{unidir} with these modified points, see Fig. 22d. Line 15 fills X_{fillet} with a fillet that starts moving in the same direction at x_2 while the x component of X_{unidir} is negative. When the x component of X_{unidir} hits the y-axis

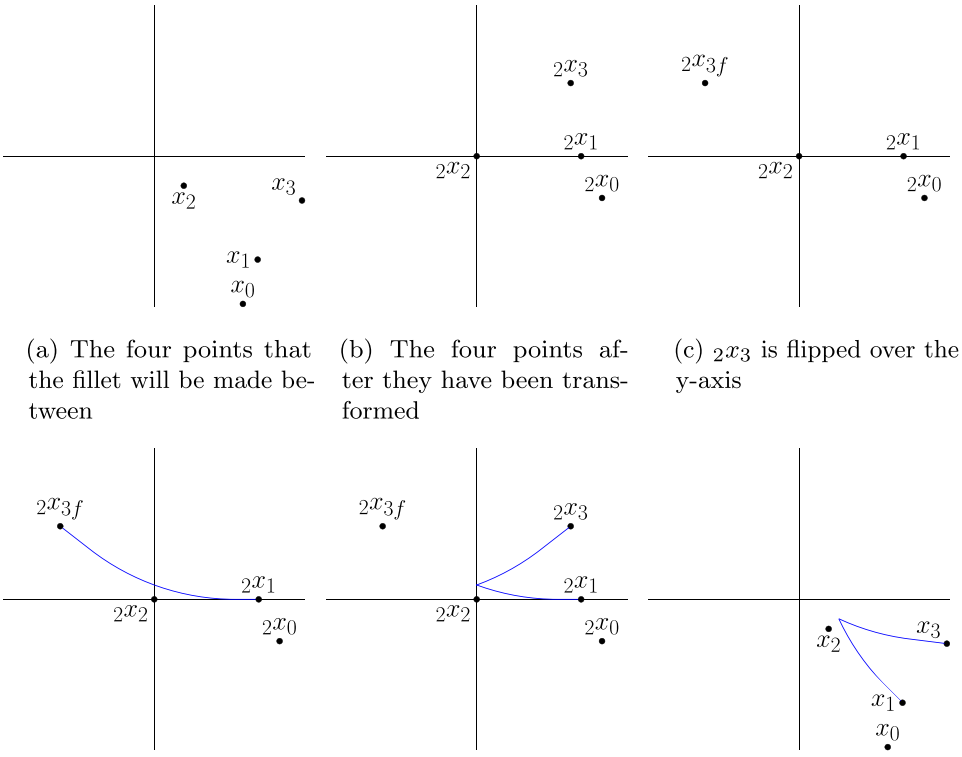


Fig. 22 An illustration of the *ReverseFillet* procedure

(a) The four points that the fillet will be made between

(b) The four points after they have been transformed

(c) $_2x_3$ is flipped over the y-axis

(d) The unidirectional fillet is generated

(e) The second half of the unidirectional fillet is flipped to reverse it

(f) The fillet is transformed back into the original coordinate frame

the fillet switches to the direction of travel of x_3 and flips the fillet across the y-axis. The result is a fillet that comes to a point on the y-axis and switches direction at that point, see Fig. 22e. Line 17 transforms fillet back to the original coordinate frame, as shown in Fig. 22f.

One scenario where the ability to plan forward and reverse fillets is beneficial is shown in Fig. 23. Figure 23 uses the same planning configuration as Fig. 18 from Section 6.4. The only difference between Figs. 23 and 18 is that Fig. 23 shows the result of path planning using the *ReverseFillet* procedure in green and red instead of the solution found with straight-line primitives. The solution found from planning without the *ReverseFillet* procedure is shown in blue. Both planners are using arc-fillets with the same maximum curvature constraint, but the red path makes use of the *ReverseFillet* procedure.

As is described in Section 6.4, a path that goes through the narrow hallway cannot satisfy the curvature constraints of the problem without hitting walls when solely forward motion is considered. Figure 23 shows that it is possible using the *ReverseFillet* procedure. Following the green and red solution, generated with forward and reverse motion, the path turns partially into the narrow hallway. When it nears the wall, the path stops and continues the turn in reverse. The path follows the hallway in reverse until it gets to X_t. Without the functionality added with the *ReverseFillet* procedure, the blue solution is unable to follow the hallway and instead must plan a significantly longer path that goes around the obstacles. Note that the inclusion of the reverse motion causes

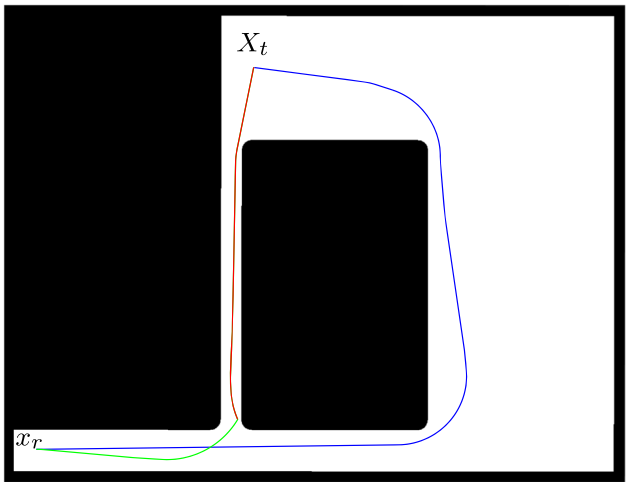


Fig. 23 The resulting paths from running FB-RRT* in an environment where the shortest path turns sharply down a corridor. Solutions from planning with arc-fillet paths are shown in blue. Solutions from planning with reverse-arc-fillet paths are shown in green when $d = 1$, forward, and red when $d = -1$, reverse

an increase in convergence time due to the added dimension in the sampling space. Future work could include methods to reduce this complexity and also to penalize long stretches of reverse motion.

References

1. Akgun, B. Stilman, M.: Sampling Heuristics for Optimal Motion Planning in High Dimensions. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2640–2645. (2011). https://doi.org/10.1109/IROS.2011.6095077
2. Beard, R., McLain, T.: Small Unmanned Aircraft: Theory and Practice. Princeton University Press, USA (2012)
3. Cui, P., Yan, W., Guo, X.: Path Planning for Robot with Pose Constraints Using Dubins-RRT. In: 2018 3rd International Conference on Advanced Robotics and Mechatronics (ICARM), pp. 560–565. (2018). https://doi.org/10.1109/ICARM.2018.8610812
4. Gammell, J., Srinivasa, S., Barfoot, T.: Informed rrt*: Optimal Sampling-Based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic. IEEE/RSJ Int. Conf. Intell. Robot. Syst. (2014). https://doi.org/10.1109/IROS.2014.6942976
5. Gravesen, J.: Adaptive subdivision and the length and energy of bézier curves. Comput. Geom. **8**(1), 13–31 (1997). https://doi.org/10.1016/0925-7721(95)00054-2
6. Karaman, S., Frazzoli, E.: Sampling-Based Algorithms for Optimal Motion Planning. Int. J. Robot. Res. **30**(7), 846–894 (2011). https://doi.org/10.1177/0278364911406761
7. Kobilarov, M.: Cross-Entropy Motion Planning. Int. J. Robot. Res. **31**(7), 855–871 (2012). https://doi.org/10.1177/0278364912444543
8. Kuffner, J.J., LaValle, S.M.: RRT-connect: An Efficient Approach To Single-Query Path Planning. In: Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), vol. 2, pp. 995–1001. (2000)
9. Kuwata, Y., Teo, J., Fiore, G., Karaman, S., Frazzoli, E., How, J.P.: Real-Time Motion Planning With Applications to Autonomous Urban Driving. IEEE Trans. Control. Syst. Technol. **17**(5), 1105–1118 (2009). https://doi.org/10.1109/TCST.2008.2012116
10. Lan, X., Di Cairano, S.: Continuous Curvature Path Planning for Semi-Autonomous Vehicle Maneuvers Using RRT. In: 2015 European Control Conference (ECC), pp. 2360–2365. (2015). https://doi.org/10.1109/ECC.2015.7330891
11. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006). http://planning.cs.uiuc.edu/
12. Li, Y., Littlefield, Z., Bekris, K.E.: Asymptotically Optimal Sampling-Based Kinodynamic Planning. Int. J. Robot. Res. **35**(5), 528–564 (2016). https://doi.org/10.1177/0278364915614386
13. Masehian, E., Kakahaji, H.: NRR: a Nonholonomic Random Replanner for Navigation of Car-Like Robots in Unknown Environments. Robotica **32**(7), 1101–1123 (2014). https://doi.org/10.1017/S0263574713001276
14. Moll, M., Sucan, I.A., Kavraki, L.E.: Benchmarking Motion Planning Algorithms: An Extensible Infrastructure for Analysis and Visualization. IEEE Robot. Autom. Mag. **22**(3), 96–102 (2015)
15. Moon, C., Ching, W.: Kinodynamic Planner Dual-Tree RRT (DT-RRT) for Two-Wheeled Mobile Robots Using the Rapidly Exploring Random Tree. IEEE Trans. Ind. Electron. **62**, 1080–1090 (2015). https://doi.org/10.1109/TIE.2014.2345351
16. Nasir, J., Islam, F., Malik, U., Ayaz, Y., Hasan, O., Khan, M., Muhammad, M.: Rrt*-smart: A Rapid Convergence Implementation Of RRT*. Int. J. Adv. Robot. Syst. **10**, (2013). https://doi.org/10.5772/56718
17. Noreen, I., Khan, A., Habib, Z.: Optimal Path Planning Using RRT* Based Approaches: A Survey and Future Directions. Int. J. Adv. Comput. Sci. Appl. **7**(11), (2016). https://doi.org/10.14569/IJACSA.2016.071114
18. Qureshi, A.H., Ayaz, Y.: Intelligent Bidirectional Rapidly-Exploring Random Trees for Optimal Motion Planning in Complex Cluttered Environments. Robot. Auton. Syst. **68**, 1–11 (2015). https://doi.org/10.1016/j.robot.2015.02.007. https://www.sciencedirect.com/science/article/pii/S0921889015000317
19. Tahir, Z., Qureshi, A.H., Ayaz, Y., Nawaz, R.: Potentially Guided Bidirectionalized RRT* for Fast Optimal Path Planning in Cluttered Environments. Robot. Auton. Syst. **108**, 13–27 (2018). https://doi.org/10.1016/j.robot.2018.06.013
20. Venables, W.N., Ripley, B.D.: Modern Applied Statistics with S, 4th edn. Springer, New York (2002). https://www.stats.ox.ac.uk/pub/MASS4/. ISBN 0-387-95457-0
21. Yang, K., Gan, S.K., Huh, J., Joo, S.: Optimal Spline-Based RRT Path Planning Using Probabilistic Map. In: 2014 14th International Conference on Control, Automation and Systems (ICCAS 2014), pp. 643–646. (2014b)
22. Yang, K., Sukkarieh, S.: An Analytical Continuous-Curvature Path-Smoothing Algorithm. IEEE Trans. Robot. **26**(3), 561–568 (2010)
23. Yang, K., Moon, S., Yoo, S., Kang, J., Doh, N.L., Kim, H.B., Joo, S.: Spline-Based RRT Path Planner for Non-Holonomic Robots. J. Intell. Robot. Syst. **73**(1–4), 763–782 (2014)

James Swedeen is a doctoral candidate in the department of Electrical and Computer Engineering at Utah State University. He has worked as a graduate research assistant at Utah State University since Fall 2018. His research interests lie in autonomous robotic path planning under kinematic constraints. His research has been focused on applying kinematic constraints within a Rapid-exploring Random Tree paradigm. He completed his BS in Computer Engineering at Utah State University in 2021.

Greg N. Droge is an Assistant Professor in the department of Electrical and Computer Engineering at Utah State University. His research is focused autonomous decision-making capabilities to enable unmanned teams to react to actionable intelligence to accomplish a higher-level objective. He completed his PhD at Georgia Institute of Technology in Electrical and Computer Engineering in 2014. His thesis focused on distributed optimization and multi-vehicle coordinated path planning. After graduating, he worked in industry developing, testing, and demonstrating a collaborative cross domain mission planning capability aimed at improving coordinated operations of teams of unmanned systems. Dr. Droge also received an MS from Georgia Tech in 2012. Prior to Georgia Tech, he studied at Brigham Young University where he graduated Magna Cum Laude with a BS in Electrical Engineering in 2009.

Randall Christensen received his PhD in Mechanical and Aerospace Engineering in 2013 from Utah State University. His early career focused on inertial navigation and Kalman filtering, high precision pointing control, and target tracking algorithms for missile terminal homing. Following his time in industry, Dr. Christensen returned to Utah State University as an assistant professor, performing research in UAV path planning, terrain-relative navigation, and Linear Covariance analysis for lunar landers and satellite rendezvous/proximity operations. He is currently employed at Blue Origin LLC, working on a variety of space missions including commercial space stations and lunar landers.

CHAPTER 5

FILLET-BASED BATCH INFORMED TREES (FB-BIT*): RAPID CONVERGENCE PATH PLANNING FOR CURVATURE CONSTRAINED VEHICLES

Published in the *IEEE International Conference on Robotic Computing (IRC)* [74]

Fillet-Based Batch Informed Trees (FB-BIT*): Rapid Convergence Path Planning for Curvature Constrained Vehicles

James Swedeen
Utah State University, USA
Email: james.swedeen@usu.edu

Greg Droge
Utah State University, USA
Emial: greg.droge@usu.edu

2023 Seventh IEEE International Conference on Robotic Computing (IRC) | 979-8-3503-9574-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/IRC59093.2023.00019

***Abstract*—In recent years the problem of planning paths through complex obstacles while respecting kinematic constraints has seen increased attention. Many have found that respecting curvature constraints is particularly difficult without substantially slowing convergence speed. Batch Informed Trees (BIT*) is a path-planning algorithm that has been shown to converge rapidly in large environments without considering kinematic constraints. This work proposes an extension to BIT* that employs fillets as motion primitives, enabling the incorporation of curvature constraints into the planning process. Path-length heuristics for fillet-based planning are introduced to accelerate convergence. Comparisons to pre-existing approaches are made with an Unmanned Aerial Vehicle (UAV) simulation modeled off of Manhattan, New York.**

I. Introduction

The ability to plan paths through complex obstacles is a fundamental requirement of many mobile robot applications. The literature has seen explosive growth in sampling-based motion planning algorithms that provide asymptotic optimality guarantees for this problem [1]–[4]. These algorithms operate on the principles of dynamic programming, breaking the problem into many smaller problems that can each be solved individually and then combined to make the overall solution. In recent years the problem of planning paths that respect kinematic constraints, such as max curvature, has received increased attention [5]–[8]. This is a problem of interest because many mobile robots are unable to follow paths that turn on a point. For example, an Unmanned Aerial Vehicle (UAV) must maintain forward velocity while executing a turn. For path planning applications with curvature constraints, this work develops Fillet-Based Batch Informed Trees (FB-BIT*).

Many sampling-based motion planning algorithms are based on Optimal Rapidly-exploring Random trees (RRT*) [9]. RRT* iteratively samples the continuous configuration space, building a rooted search tree from the initial location to every reachable part of the obstacle-free space. As RRT* searches, local optimizations are performed on the search tree that shorten the length of the paths through the tree. As the number of iterations RRT* performs goes to infinity, the solution path to the target location converges to optimality [9]. However, this convergence can be slow [1], [2], [10].

Fast Marching Trees (FMT*) is another sampling-based path-planning algorithm that makes use of the principles of dynamic programming to avoid unnecessary calculations [11]. FMT* builds a rooted search tree, similar to RRT*. However, instead of iteratively sampling the state space and building the search tree simultaneously, FMT* samples a fixed number of times at startup. FMT* then builds a search tree using the sampled points. With the knowledge of where each sample is from the start, FMT* avoids many of the calculations that RRT* performs while iteratively optimizing its tree. This makes FMT* faster than RRT* at generating a solution. However, it loses RRT*'s ability to refine the solution indefinitely.

Batch Informed Trees (BIT*) merges the iterative nature of RRT* with the efficient graph searching used in FMT* [12]. BIT* iteratively generates batches of samples of the state space, then uses a procedure similar to FMT* to incorporate the new batch of samples into the pre-existing search tree. The performance of BIT* varies with the size of each batch of samples. When the batch size is only one, BIT* is nearly equivalent to RRT*. When the batch size is very large, BIT* is nearly equivalent to FMT*, except for considering subsequent batches. This allows the user of the algorithm to tune BIT* to have the desired performance characteristics.

Curvature constraints in path planning problems can be trivially satisfied in near-open environments by smoothing the straight-line path generated by one of the planners described above [13]. However, in tight corridors, this smoothing approach often results in obstacle collisions or requires turns that violate the curvature constraints. Adding curvature constraints to sampling-based algorithms can be difficult. First, when applying the principles of dynamic programming, difficulty arises when attempting to connect sub-problem solutions while respecting curvature constraints. Furthermore, the dynamic programming assumption that local optimizations produce globally optimal results is not always maintained [8].

Several approaches attempt to plan paths with RRT*-based algorithms while respecting kinematic constraints, e.g., [5], [7], [14]. However, many have found curvature constraints to be difficult to respect in RRT*-based algorithms without significantly slowing the convergence rate of the algorithms they are using [15], [16]. Fillet-Based Rapidly-exploring Random Tree (FB-RRT*) is an extension of RRT* specifically designed for curvature-constrained path planning [8]. FB-RRT* works in the same way RRT* does except it uses a

979-8-3503-9574-7/23/$31.00 ©2023 IEEE
DOI 10.1109/IRC59093.2023.00019

general fillet formulation to connect nodes in the search tree instead of point-to-point connections. FB-RRT* uses fillets to naturally incorporate curvature constraints while combining sub-problem solutions and adds additional constraints to local optimizations to guarantee global convergence.

This work develops Fillet-Based Batch Informed Trees (FB-BIT*), a path-planning algorithm for rapid convergence in large state spaces while respecting curvature constraints. FB-BIT* draws on ideas from BIT* to leverage the advantages of RRT* and FMT* while using fillets to locally connect points in the search tree. This means the resulting path will respect any curvature constraint that can be formulated as a fillet, for example, max curvature, max curvature rate, etc. Section II provides the notation used in this work and a brief description of BIT*. Section III discusses the fillet motion primitive. Section IV presents the FB-BIT* algorithm, which is applied to fixed-wing UAV path planning in Section V.

II. Notation and Background

Sections II-A and II-B define the notation and functions used throughout this work. Section II-C describes BIT*.

A. Nomenclature

RRT* and BIT*-based algorithms iteratively construct a tree to find a path through the obstacle-free state space. The tree is an acyclic-directed graph denoted as $T \triangleq (V, E)$, where V is the set of nodes or vertices within the tree and $E \subset V \times V$ denotes the set of edges between vertices. The root vertex, x_r , has no parent while all other vertices have exactly one parent. Each vertex can have multiple children. Each vertex within V corresponds to a state in the d-dimensional state space denoted as $X \subset \mathbb{R}^d$. Nodes are added to the tree to find a path to the target set, $X_t \subset X$. The path must avoid the space blocked by obstacles, $X_{obs} \subset X$, staying within the obstacle-free space, $X_{free} \triangleq X \setminus X_{obs}$. $\mathbb{R}$ is the set of all real numbers, $\mathbb{Z}$ is the set of all integers, and $\mathbb{R}_+$ is the set of all real positive numbers. The notation $X \xleftarrow{+} \{x\}$ and $X \xleftarrow{-} \{x\}$ is used to compactly represent the set updating operations $X \leftarrow X \cup \{x\}$ and $X \leftarrow X \setminus \{x\}$ respectively.

BIT* makes use of several cost functions that calculate various quantities of importance. $g_T(x)$ is the cost-to-come from x_r to x through T. $\hat{g}(x)$ is a lower bounded heuristic of $g_T(x)$, i.e., $\forall x \in V, \hat{g}(x) \leq g_T(x) < \infty$. $\hat{h}(x)$ is a lower bounded heuristic of the true optimal cost-to-go from x to the goal state. $c(x, y)$ and $\hat{c}(x, y)$ are the true cost of an edge and a lower bounded heuristic of the cost of an edge connecting the states $x, y \in X$, respectively. Edges that are blocked by obstacles have a cost of infinity, thus $\forall x, y \in X, \hat{c}(x, y) \leq c(x, y) \leq \infty$. Note that $\hat{c}$ typically will not consider obstacles, which makes it a fast operation.

When using straight-line paths in this work, $\hat{c}$, $\hat{g}$, and $\hat{h}$ are defined as follows. $\hat{c}$ is defined using Euclidean distance as $\hat{c}(x, y) := \|x - y\|$. $\hat{g}$ and $\hat{h}$ are defined using $\hat{c}$ as $\hat{g}(x) := \hat{c}(x_r, x)$ and $\hat{h}(x) := \hat{c}(x, x_t)$, where $x_t \in X_t$.

B. Common Procedures

Several BIT* procedures are now defined. They can be found in [8], [12] with notation updated to match the sequel.

Definition 1 ($X_{rand} \leftarrow Sample(m)$): Generates a set of $m \in \mathbb{Z}_+$ random samples of X_{free}.

Definition 2 ($X_{near} \leftarrow Near_\rho(x, X_{search})$): Finds all vertices in X_{search} that are within a given heuristic edge cost radius, $\rho \in \mathbb{R}_+$, of the point $x \in X$, i.e., $Near_\rho(x, X_{search}) := \{v \in X_{search} | \hat{c}(x, v) \leq \rho\}$.

Definition 3 ($x_p \leftarrow Par(x_c)$): Returns the parent node of $x_c \in V$ in the tree T, or $\varnothing$ if x_c is the root node.

Definition 4 ($X_{children} \leftarrow Children(x_p)$): Returns every node in T that has x_p as its parent.

Definition 5 ($X_{sol} \leftarrow Solution(x)$): Finds the path through T that leads from the root node to $x \in V$.

Definition 6 ($cost \leftarrow BestValue(\mathcal{Q}_i)$): Finds the element of the generic queue $\mathcal{Q}_i$ with the lowest queue cost and returns the queue cost of that element.

Definition 7 ($x \leftarrow PopBest(\mathcal{Q}_i)$): Finds the element of the generic queue $\mathcal{Q}_i$ with the lowest queue cost, removes it from the queue, and returns that element.

C. Batch Informed Trees

BIT* functions by iteratively generating batches of samples from the state space and incorporating those new samples into the pre-existing search tree. To achieve this BIT* defines two sorted queues, the vertex and edge queues. At the beginning of each batch, the vertex queue is populated with all the nodes currently in the tree. Vertices are then iteratively removed from the vertex queue and all potential edges that start at that vertex are added to the edge queue. Once all potential edges between samples of the state space have been found, the search tree is updated to include them if doing so will shorten the length of the resulting path. When both queues become empty a new batch of samples is generated and the process begins again. For a thorough description of BIT* see [12], [17].

III. Fillet Planning Approach

An efficient way to formulate curvature constraints is through fillets. Fillets connect two line segments (defined with three points) with a curve transitioning smoothly between them, as shown in Figure 1. This section introduces some general fillet concepts and requirements for creating a path using fillets. The arc-fillet is defined for demonstration purposes in Section III-B. Section III-C uses fillets as motion primitives while defining primitive procedures that are used in FB-BIT*.

A. Fillet Considerations

Given three input points $x_1, x_2, x_3 \in X$, a fillet connects x_1 to x_3 with the combination of two straight-line segments and a curve. The line segments constitute portions of the lines $\overline{x_1 x_2}$ and $\overline{x_2 x_3}$. The curve intersects the line $\overline{x_1 x_2}$ at point $x_{s,2}$ and line $\overline{x_2 x_3}$ at $x_{e,2}$. The resulting fillet moves in a straight-line from x_1 to $x_{s,2}$, along the curve from $x_{s,2}$ to $x_{e,2}$, and then along the straight-line from $x_{e,2}$ to x_3, as depicted in Figure 1. This work assumes symmetric fillets, resulting in

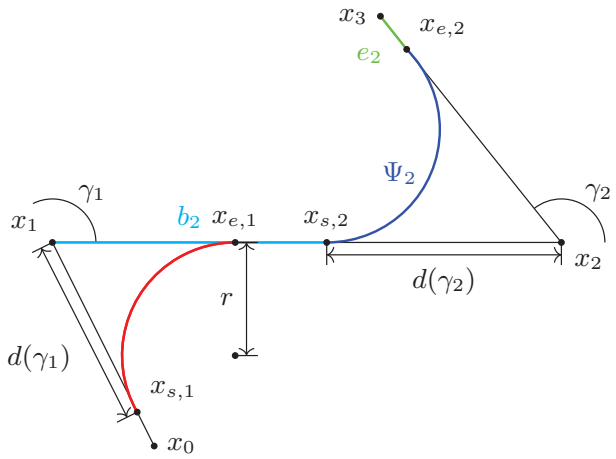


Fig. 1: The fillet generated to connect x_1 and x_3 is shown in blue with the fillet that comes before it shown in red. Note that $d(\gamma_1)+d(\gamma_2) \leq \|x_1 - x_2\|$, providing a continuous path.

an equivalent distance between the center point of the fillet, x_2, and the two ends of the fillet curve, $x_{s,2}$ and $x_{e,2}$. Let that distance be denoted as $d(\gamma_2)$ where γ_2 is the change in orientation between $\overrightarrow{x_1x_2}$ and $\overrightarrow{x_2x_3}$.

1) Continuity Conditions: The key to using the fillets for path planning purposes is to ensure that the path resulting from joining multiple fillets is continuous. There are two conditions to ensure feasibility: one to ensure that the fillet curve ends before the final point in the fillet and another to ensure that all fillet curves end before the next one begins [8]. Assuming x_i is the middle node, these conditions can be expressed as

$$\begin{aligned} d(\gamma_i) &\leq \|x_i - x_{i+1}\| \\ d(\gamma_{i-1}) + d(\gamma_i) &\leq \|x_i - x_{i-1}\| \end{aligned}. \tag{1}$$

2) Initial Orientation Constraint: It is common for path planning problems with curvature constraints to have an initial orientation constraint. This constraint can be satisfied with the use of a pseudo-root node. Given the initial position, $x_r \in X$, and initial orientation, $\psi_r \in [-\pi, \pi)$, the pseudo-root node is defined as $x_{sr} \triangleq x_r + d_{init}\begin{bmatrix}\cos(\psi_r) & \sin(\psi_r)\end{bmatrix}^T$ where $d_{init} \in \mathbb{R}_+$ is how far the pseudo-root node, x_{sr}, is from the root node, x_r. While initializing the search tree, x_{sr} is added to the tree as a child of x_r. The search tree is then grown by extending edges from x_{sr}. Note that extending edges from x_r is impossible because it takes three nodes to make a fillet and, by definition, x_r has no parent.

3) Edge Costs: Another important consideration is the relationship between the cost of a node and that of its parent. In sampling-based path planners, the cost of an edge connecting two nodes is typically strictly positive. This intuitively makes sense because the path the edge represents goes from one node to the other. This leads to the relationship

$$g_T(x_3) = g_T(x_2) + c(x_2, x_3)$$

where $x_2, x_3 \in V$ such that $x_2 = Par(x_3)$.

When planning with fillets the cost of an edge becomes more complicated. As shown in Figure 1, the cost of node x_3 can be stated as $g_T(x_3) = g_T(x_2) + \Psi_2 + e_2 - d(\gamma_2)$ where $e_2 = \|x_{e,2} - x_3\|$ and Ψ_2 is the length of the curved section shown in blue. This means the cost of the edge that connects x_3 to x_2 is

$$c(x_1, x_2, x_3) = \Psi_2 + e_2 - d(\gamma_2). \tag{2}$$

Note that: 1) $c(x_1, x_2, x_3)$ is not guaranteed to be positive, 2) There is a dependence upon x_1 in addition to x_2 and x_3.

4) The Non-Monotonicity of Node Costs: In straight-line sampling-based path planners, the cost of a node is always greater than its parent. However, when planning with fillets, the cost of an edge is no longer bounded below by zero. As a result, the costs of the nodes that make up a path from the root node to the target set are not monotonically increasing. This has important repercussions for FB-BIT*.

B. Arc-Fillet

In this work an arc-fillet is used for demonstration purposes, however, FB-BIT* can be used with any symmetric-fillet type. The arc-fillet formulates the curve portion of the fillet using a circle of radius $r \in \mathbb{R}_+$ that is tangential to the two line segments at $x_{s,1}$ and $x_{e,1}$, see the red curve in Figure 1. The distance between the curve intersection points, $x_{s,1}$ and $x_{e,1}$, and the intermediary point, x_1, is expressed in Lemma 1:

Lemma 1 (see [8]): The arc-fillet distance, $d(\gamma)$, for a circular curve of radius r is

$$d(\gamma) := \frac{r(1 - \cos(\gamma))}{\sin(\gamma)}. \tag{3}$$

C. Fillet-Based Cost Functions

The costs c, $\hat{c}$, and $\hat{h}$ are redefined for fillets to consider the parent node, as is done in (2). Note that g_T and $\hat{g}$ do not need to be redefined.

$c(x_1, x_2, x_3)$ and $\hat{c}(x_1, x_2, x_3)$ are the true cost of an edge and a lower bounded heuristic for the cost of an edge that connects $x_2 \in X$ to $x_3 \in X$ assuming the fillet that ends at x_3 starts at $x_1 \in X$, respectively. $c(x_1, x_2, x_3) = \infty$ if any part of the fillet is blocked by an obstacle or the conditions in (1) are violated. In this work $\hat{c}$ is defined as the true path length of the edge that connects x_3 to x_2 when the first condition in (1) is satisfied and infinity otherwise. Drawing from (2), (3), and that $\Psi_2 = r\gamma_2$ for arc-fillets, $\hat{c}$ is defined as

$$\hat{c}(x_1, x_2, x_3) := \begin{cases} r\gamma_2 + e_2 - d(\gamma_2) & d(\gamma_2) \leq d_{23} \\ \infty & \text{Otherwise} \end{cases} \tag{4}$$

where $d_{23} := \|x_2 - x_3\|$.

$\hat{h}(x_2, x_1)$ is a lower bounded heuristic of the true optimal cost-to-go of $x_2 \in X$ assuming x_2 is connected to $x_1 \in X$, i.e., $x_1 = Par(x_2)$ if $x_2 \in V$. Unlike the non-fillet case using the edge cost heuristic to the end of the best solution found, i.e., $\hat{c}(x_1, x_2, x_t)$, is insufficient because $\hat{c}(x_1, x_2, x_t)$ is not a lower bound on the true cost-to-go. This can be shown through an example. Consider the case where $x_2 = \begin{bmatrix}1 & 1\end{bmatrix}^T$, $x_1 =$

Algorithm 1: *FB-BIT**

Input: $x_r, \psi_r, d_{init}, X_{goal}, X_t$
Output: X_{sol}

1 $x_{sr} \leftarrow x_r + d_{init}\begin{bmatrix}\cos(\psi_r) & \sin(\psi_r)\end{bmatrix}^T$;
2 $V \leftarrow \{x_r, x_{sr}\}$; $E \leftarrow \{(x_r, x_{sr})\}$; $T \triangleq (V, E)$;
3 $\mathcal{Q}_V \leftarrow \{x_n\}$; $\mathcal{Q}_E \leftarrow \varnothing$; $\mathcal{Q} \triangleq (\mathcal{Q}_V, \mathcal{Q}_E)$;
4 $X_{ncon} \leftarrow X_{goal}$; $X_{new} \leftarrow X_{ncon}$;
5 $V_{exp} \leftarrow \varnothing$; $V_{rewire} \leftarrow \varnothing$; $V_{sol} \leftarrow V \cap X_t$;
6 **if** $V_{sol} = \varnothing$ **then** $c_{sol} \leftarrow \infty$ **else** $c_{sol} \leftarrow \min_{v \in V_{sol}} g_T(v)$;
7 $X_{flags} \triangleq (X_{new}, V_{exp}, V_{rewire}, V_{sol}, c_{sol})$;
8 **repeat**
9 **if** $\mathcal{Q}_V = \varnothing \wedge \mathcal{Q}_E = \varnothing$ **then** `// Batch end`
 `// Prune sub-optimal nodes`
10 $\{X_{reuse}, T, X_{ncon}, X_{flags}\} \leftarrow$ $Prune(T, X_{ncon}, X_{flags})$;
11 $X_{new} \leftarrow Sample(m)$;
 `// Add new samples to queues`
12 $X_{ncon} \xleftarrow{+} X_{new} \cup X_{reuse}$;
13 $\mathcal{Q}_V \leftarrow V \cap \{x_r\}$;
14 **else if** $BestValue(\mathcal{Q}_V) \leq BestValue(\mathcal{Q}_E)$ **then**
 `// Process best vertex available`
15 $\{\mathcal{Q}, X_{flags}\} \leftarrow ExpVertex(T, \mathcal{Q}, X_{ncon}, X_{flags})$;
16 **else** `// Process best edge available`
17 $\{T, \mathcal{Q}, X_{ncon}, X_{flags}\} \leftarrow$ $ExpEdge(T, \mathcal{Q}, X_{ncon}, X_{flags}, X_t)$;
18 **until** *STOP*;
19 **return** $Solution\big(\arg\min_{v_t \in V_{sol}} g_T(v_t)\big)$;

Algorithm 2: *Prune*

Input: $T \triangleq (V, E), X_{ncon}, X_{flags} \triangleq$ $(X_{new}, V_{exp}, V_{rewire}, V_{sol}, c_{sol})$
Output: $X_{reuse}, T, X_{ncon}, X_{flags}$

1 $X_{reuse} \leftarrow \varnothing$;
2 $X_{ncon} \xleftarrow{-} \left\{x \in X_{ncon} \middle| \hat{g}(x) + \hat{h}(x, x_r) \geq c_{sol}\right\}$;
3 **forall** $v \in V$ **do**
4 **if** $1 = RecursiveCheck(v, T, c_{sol})$ **then**
 `// Remove v from the search`
5 $V \xleftarrow{-} \{v\}$; $E \xleftarrow{-} \{(Par(v), v)\}$;
6 $X_t \xleftarrow{-} \{v\}$; $X_{exp} \xleftarrow{-} \{v\}$; $X_{rewire} \xleftarrow{-} \{v\}$;
 `// If v can possibility help`
7 **if** $\hat{g}(v) + \hat{h}(v, x_r) \leq c_{sol}$ **then** $X_{reuse} \xleftarrow{+} \{v\}$;
8 **return** $\{X_{reuse}, T, X_{ncon}, X_{flags}\}$;

$\begin{bmatrix}0 & 0\end{bmatrix}^T$, and $x_t = \begin{bmatrix}0 & 2\end{bmatrix}$. Using the definition of the edge cost heuristic in (4) with $r = 1/4$, $\hat{c}(x_1, x_2, x_t) = 1.3069$. Now consider the case where a new point, $x_3 = \begin{bmatrix}0 & 1\end{bmatrix}^T$, has x_2 as a parent and x_t as a child. This means there is a path from x_2 to x_t through x_3. Then the true cost-to-go of x_2 is $c(x_1, x_2, x_3) + c(x_2, x_3, x_t) = 1.2746$. The true cost-to-go of x_2 in this example is less than $\hat{c}(x_1, x_2, x_t)$, which shows that $\hat{c}$ is not a lower bounded cost-to-go heuristic.

In this work $\hat{h}$ is defined as

$$\hat{h}(x_2, x_1) := \|x_{e,1} - x_t\| - \|x_{e,1} - x_2\|$$

where $x_t \in X_t$ is the end of the best solution found and $x_{e,1}$ is the end point of the curve portion of the fillet centered at x_1, see Figure 1. This can be thought of as the shortest length cost-to-go from $x_{e,1}$ to x_t minus the extra path length from $x_{e,1}$ to x_2. $x_{e,1}$ is used because it is the closest point to x_2 on the line $\overline{x_1 x_2}$ that is guaranteed to be part of any path that includes the edge (x_1, x_2).

IV. Fillet-Based Batch Informed Trees

This section describes the Fillet-Based Batch Informed Trees (FB-BIT*) algorithm. FB-BIT*, like BIT*, functions by iteratively generating batches of samples from the state space and incorporating those new samples into the pre-existing search tree. Where FB-BIT* differs from BIT* is the conditions under which an edge can be added to the tree. When planning with fillets only edges that respect the conditions in (1) can be used in the tree, as described in Section III-A1.

FB-BIT* is given in Algorithm 1. The inputs are the root node, x_r, the initial orientation, $\psi_r \in [-\pi, \pi)$, the pseudo-root node distance, $d_{init} \in \mathbb{R}_+$, a sampling of the target set, $X_{goal} \subset X_t$, and the target set, $X_t \subset X_{free}$. Lines 1 and 2 initialize the search tree, T, with x_r and x_{sr} as is described in Section III-A2.

FB-BIT* maintains two queues: the vertex queue, $\mathcal{Q}_V$, and the edge queue, $\mathcal{Q}_E$. $\mathcal{Q}_V$ is used to keep track of vertices that are under consideration for making potential edges and $\mathcal{Q}_E$ is used to keep track of edges that are under consideration for addition to T. $\mathcal{Q}_V$ is organized in terms of the current cost-to-come plus the heuristic cost-to-go of the vertices given their current parent in the tree, i.e., $g_T(v) + \hat{h}(v, Par(v)), v \in \mathcal{Q}_V$. $\mathcal{Q}_E$ is organized in terms of the sum of the current cost-to-come of the edge's source vertex, the heuristic cost of the edge, and the heuristic cost-to-go through the edge, i.e., $g_T(v) + \hat{c}(Par(v), v, x) + \hat{h}(x, v), (v, x) \in \mathcal{Q}_E$. These queues are initialized on line 3.

Lines 4 through 7 initialize sets that keep track of the state of each vertex. $X_{ncon} \subset X_{free}$ is the set of all samples of the state space that are *not connected* to the search tree. $V_{sol} = V \cap X_t$ is the set of all vertices in the tree that are also in the target set. $X_{new} \subset X_{ncon}$ is the set of all samples that are *new* this batch. $V_{exp} \subset V$ and $V_{rewire} \subset V$ are the sets of vertices that have been considered for *expansion* and *rewiring*, respectively. c_{sol} is the cost of the current best solution or infinity if no solution has been found.

The loop on lines 8 through 18 performs the path planning and ends when a user-defined stopping condition is met. The conditionals on lines 9 and 14 determine if the batch has ended and whether to process a vertex from $\mathcal{Q}_V$ or an edge from $\mathcal{Q}_E$, respectively. Each case is discussed below.

A. Generating New Batches

When $\mathcal{Q}_V$ and $\mathcal{Q}_E$ are both empty it signifies the end of the batch, see line 9 of Algorithm 1. When this happens, all vertices that cannot contribute to the optimal solution given their current parent are removed from the search tree by calling the $Prune$ procedure on line 10.

The $Prune$ procedure is given in Algorithm 2. Line 2 of Algorithm 2 removes all unconnected samples with heuristic cost-to-come plus heuristic cost-to-go value greater than the current best solution cost. This removes all samples that fall outside the "informed set" that Informed RRT* (I-RRT*) defines [1] and, as such, provably cannot contribute to the optimal solution no matter how they are connected to T. Note that when deciding which unconnected samples to remove, the root node, x_r, is used as the parent node in $\hat{h}$. This is because choosing x_r to represent the parent of any sample of X minimizes the heuristic cost-to-go of that sample, i.e.,

$$x_r = \underset{x_p \in X}{\arg\min}\, \hat{h}(x, x_p),\ \forall x \in X.$$

Thus, $\hat{g}(x) + \hat{h}(x, x_r)$ is a lower bound on the length of a solution path that is constrained to make use of $x \in X$.

The loop on lines 3 though 7 removes any vertices in the search tree that cannot contribute to the optimal solution given their current connection to T. The condition on line 4 checks if v and all descendants of v in T have the potential to contribute to the optimal solution given their current connection to T, i.e., $g_T(v) + \hat{h}(v, Par(v)) \leq c_{sol}$. The descendants of v are checked because node costs are non-monotonic, see Section III-A4, which means v might contribute to the solution while having a cost greater than c_{sol}. The only way to make sure that v is not contributing to the current best solution is to perform this check on v and all descendants of v, as $RecursiveCheck$ does. If v and all of the descendants of v cannot contribute to the optimal solution given their current parent, v is removed from the tree on lines 5 and 6.

Line 7 checks if v has the potential to contribute to the optimal solution given the ideal cost-to-come and cost-to-go. This is effectively the same condition that is used on line 2. If there is a chance of v contributing to the optimal solution given a different connection to T, v is added to X_{reuse} to be reused as an unconnected sample.

After $Prune$ is finished, line 11 of Algorithm 1 generates m new samples from X_{free}. Line 12 adds the new samples and reused vertices to X_{ncon}. Line 13 adds all but the root node to $\mathcal{Q}_V$. This insures all vertices in the T will be considered when looking for ways to connect the new samples to T. x_r is left out of $\mathcal{Q}_V$ because, in the fillet-based framework, an edge cannot be made from a node with no parent and, by definition, x_r has no parent.

B. Expanding Vertices

Lines 14 and 15 of Algorithm 1 find potential edges to add to $\mathcal{Q}_E$ from the vertices in $\mathcal{Q}_V$. The condition on line 14 evaluates to true until it is impossible for any vertex in $\mathcal{Q}_V$ to produce an edge of lower heuristic cost then the best edge in $\mathcal{Q}_E$. This can be seen by noting that $\forall v \in \mathcal{Q}_V, \forall x \in X, g_T(v) + \hat{h}(v, Par(v)) \leq g_T(v) + \hat{c}(Par(v), v, x) + \hat{h}(x, v)$ as $\hat{h}(v, Par(v))$ is an underestimate of the true cost-to-go of vertex v. Thus, the vertex queue cost, $g_T(v) + \hat{h}(v, Par(v))$, is a lower bound on the edge queue cost, $g_T(v) + \hat{c}(Par(v), v, x) + \hat{h}(x, v)$ of any edge that can be made from that vertex.

Algorithm 3: $ExpVertex$

Input: $T \triangleq (V, E), \mathcal{Q} \triangleq (\mathcal{Q}_V, \mathcal{Q}_E), X_{ncon}, X_{flags} \triangleq (X_{new}, V_{exp}, V_{rewire}, V_{sol}, c_{sol})$

Output: $\mathcal{Q}, X_{flags}$

1 $v_b \leftarrow PopBest(\mathcal{Q}_V)$; $v_p \leftarrow Par(v_b)$;
// Edges to unconnected samples
2 **if** $v_b \notin V_{exp}$ **then**
3 $\quad X_{near} \leftarrow Near_\rho(v_b, X_{ncon})$; $V_{exp} \xleftarrow{+} \{v_b\}$;
4 **else** // v_b has been expanded before
5 $\quad X_{near} \leftarrow Near_\rho(v_b, X_{new} \cap X_{ncon})$;
6 $\mathcal{Q}_E \xleftarrow{+} \{(v_b, x), x \in X_{near} | \hat{g}(v_b) + \hat{c}(v_p, v_b, x) + \hat{h}(x, v_b) < c_{sol}\}$;
7 **if** $v_b \notin V_{rewire} \wedge c_{sol} < \infty$ **then**
$\quad$ // Edges to connected nodes
8 $\quad V_{near} \leftarrow Near_\rho(v_b, V)$; $V_{rewire} \xleftarrow{+} \{v_b\}$;
9 $\quad \mathcal{Q}_E \xleftarrow{+} \{(v_b, w), w \in V_{near} | (v_b, w) \notin E, \hat{g}(v_b) + \hat{c}(v_p, v_b, w) + \hat{h}(w, v_b) < c_{sol}, \hat{g}(v_b) + \hat{c}(v_p, v_b, w) < g_T(w)\}$;
10 **return** $\{\mathcal{Q}, V_{flags}\}$;

$ExpVertex$ removes the lowest cost vertex in $\mathcal{Q}_V$ and adds edges to $\mathcal{Q}_E$ if they might be part of the optimal solution. The $ExpVertex$ procedure is given in Algorithm 3. Line 1 of Algorithm 3 pops the lowest cost vertex in $\mathcal{Q}_V$. Lines 2 through 6 add edges to $\mathcal{Q}_E$ that go from v_b to unconnected samples of the state space. The condition on line 2 checks if this is the first time v_b has been considered for expansion. In that case, all unconnected samples are considered for connection to v_b, see line 3. If v_b has been considered for expansion before, only the samples that are new this batch are considered for connection, see line 5. This prevents redundant calculations as the samples that are not new this batch have already been considered for connection to v_b. Line 6 adds all edges in the set $\{(v_b, x), x \in X_{near}\}$ that have potential to improve the current solution to $\mathcal{Q}_E$.

Lines 7 through 9 handle the case where v_b might be a better parent for its neighbors then their current parent, in RRT* terminology, rewiring. Note that if a solution has not been found, the condition on line 7 will always be evaluated as false. This reduces the time it takes to find an initial solution by skipping any potential tree rewirings. Once the first solution is found, all rewirings that were skipped previously are considered. The condition on line 7 also evaluates to false if v_b has been considered for rewiring before. This prevents redundant calculations as the potential to perform

Algorithm 4: $ExpEdge$

Input: $T \triangleq (V, E), \mathcal{Q} \triangleq (\mathcal{Q}_V, \mathcal{Q}_E), X_{ncon}, X_{flags} \triangleq (X_{new}, V_{exp}, V_{rewire}, V_{sol}, c_{sol}), X_t$

Output: $T, \mathcal{Q}, X_{ncon}, X_{flags}$

1 $(v_b, x_b) \leftarrow PopBest(\mathcal{Q}_E)$; $v_p \leftarrow Par(v_b)$;
2 **if** $g_T(v_b) + \hat{c}(v_p, v_b, x_b) + \hat{h}(x_b, v_b) \geq c_{sol}$ **then**
3 $\quad \mathcal{Q}_E \leftarrow \varnothing$; $\mathcal{Q}_V \leftarrow \varnothing$; **goto** 24;
4 **if** $x_b \in X_{ncon}$ **then** // x_b is unconnected
5 $\quad$ **if** $g_T(v_b) + c(v_p, v_b, x_b) + \hat{h}(x_b, v_b) < c_{sol}$ **then**
$\quad\quad$ // Add x_b to the tree
6 $\quad\quad X_{ncon} \xleftarrow{-} \{x_b\}$; $V \xleftarrow{+} \{x_b\}$; $E \xleftarrow{+} \{(v_b, x_b)\}$;
7 $\quad\quad \mathcal{Q}_V \xleftarrow{+} \{x_b\}$;
8 $\quad\quad$ **if** $x_b \in X_t$ **then**
9 $\quad\quad\quad V_{sol} \xleftarrow{+} \{x_b\}$; $c_{sol} \leftarrow \min_{v \in V_{sol}} g_T(v)$;
10 **else** // x_b is connected
11 $\quad \hat{c}_{x_b} \leftarrow g_T(v_b) + \hat{c}(v_p, v_b, x_b)$;
12 $\quad$ **if** $\hat{c}_{x_b} < g_T(x_b)$ **then**
13 $\quad\quad$ **forall** $v_c \in Children(x_b)$ **do**
14 $\quad\quad\quad$ **if** $\hat{c}_{x_b} + \hat{c}(v_b, x_b, v_c) > g_T(v_c)$ **then goto** 24;
15 $\quad\quad\quad$ **forall** $v_{gc} \in Children(v_c)$ **do**
16 $\quad\quad\quad\quad$ **if** $\infty = \hat{c}(x_b, v_c, v_{gc})$ **then goto** 24;
17 $\quad\quad c_{x_b} \leftarrow g_T(v_b) + c(v_p, v_b, x_b)$;
18 $\quad\quad$ **if** $c_{x_b} + \hat{h}(x_b, v_b) < c_{sol}$ **then**
19 $\quad\quad\quad$ **if** $c_{x_b} < g_T(x_b)$ **then**
20 $\quad\quad\quad\quad$ **forall** $v_c \in Children(x_b)$ **do**
21 $\quad\quad\quad\quad\quad$ **if** $c_{x_b} + c(v_b, x_b, v_c) > g_T(v_c)$ **then goto** 24;
22 $\quad\quad\quad\quad E \xleftarrow{-} \{(Par(x_b), x_b)\}$; $E \xleftarrow{+} \{(v_b, x_b)\}$;
23 $\quad\quad\quad\quad c_{sol} \leftarrow \min_{v \in V_{sol}} g_T(v)$;
24 **return** $\{T, \mathcal{Q}, X_{ncon}, X_{flags}\}$;

rewirings around v_b has already been considered. Line 8 finds all vertices in T that are near v_b. Line 9 adds all edges from v_b to $w \in V_{near}$ that are not already part of the tree, have the potential to improve the current solution, and have the potential to improve the cost of w. Note that $ExpVertex$ uses only heuristic cost evaluations which keeps the procedure computationally lightweight and fast.

C. Evaluating Possible Edges

In the case that the lowest heuristic cost edge possible has been generated from $\mathcal{Q}_V$, the condition on line 14 of Algorithm 1 evaluates to false and $ExpEdge$ is called. $ExpEdge$ removes the most promising edge from $\mathcal{Q}_E$ and considers it for addition to the search tree.

The $ExpEdge$ procedure is given in Algorithm 4. Line 1 removes the lowest cost edge from the queue $\mathcal{Q}_E$. Line 2 checks whether the edge under consideration has a chance of improving the current solution. Note that this condition is true only if the most promising edge in $\mathcal{Q}_E$, and by extension all the edges in $\mathcal{Q}_E$, cannot contribute to the optimal solution. For this reason $\mathcal{Q}_E$ and $\mathcal{Q}_V$ are cleared on line 3.

Line 4 checks if x_b is already part of the tree. If x_b is not part of T, line 5 checks if connecting x_b to the tree through v_b can improve the current solution. If it can, x_b is added to T with v_b as its parent. Lines 8 and 9 check if x_b is in the target set and adds x_b to V_{sol} if so.

If x_b is already part of T at line 4, extra checks are performed for FB-BIT* before adding the edge under consideration to the tree. As is described in Lemma 5 of [8], for the conditions in (1) to hold and the cost of the nodes in the tree to not increase, three conditions must be met before rewiring can be performed:

1) The cost of x_b is improved.
2) The costs of the children of x_b do not increase.
3) The edges connecting to the grandchildren of x_b must not violate (1).

Lines 12 though 16 evaluate each of these conditions using heuristic costs before they are verified with the true costs. The conditions are evaluated with heuristics first because evaluating the true costs is time-consuming and evaluating the heuristic costs can rule out edges where the above three conditions are not met. Line 12 checks if connecting x_b through v_b can improve the cost of x_b. Line 14 checks that the cost of the children of x_b might not increase. Line 16 checks that the fillets connecting to the grandchildren of x_b do not violate (1). If all conditions pass when evaluated with heuristics, lines 18 though 21 evaluate each condition without the use of heuristics. Lines 18 and 19 check that this operation can improve the cost of the current solution and will improve the cost of x_b, respectively. Line 21 checks that the operation will not increase the costs of the children of x_b. If all checks pass, x_b is rewired to have v_b as its parent and the current solution cost is updated on lines 22 and 23.

Unlike $ExpVertex$, the $ExpEdge$ procedure evaluates edges using the true cost of the edge instead of heuristics and performs obstacle checking before adding the edge to the search tree. This makes $ExpEdge$ a much slower operation than $ExpVertex$, which is why $ExpEdge$ is only run with the best edge available.

V. Simulation

Simulation results are now presented to demonstrate the effectiveness of the FB-BIT* algorithm. Comparisons are made between FB-BIT* and FB-RRT* planning with arc-fillets. Results from BIT* and RRT* planning with straight-line motion primitives are also presented for the sake of completeness. Note that a comparison between motion primitive types is not meant to show that one motion primitive is better than another. Planning with straight-line paths is going to have better convergence characteristics due to their simplicity and lack of kinematic constraints. The straight-line primitive is included to show a best-case scenario.

A. Simulation Details

To test the capabilities of FB-BIT*, it is used to plan paths for a simulated UAV through an urban environment. Figure 2a shows the UAV simulation used, with buildings drawn in red. The buildings are modeled off the real buildings in Manhattan, New York. The location and height of the buildings

(a) The UAV simulation where Manhattan's buildings are shown in red.

(b) The resulting paths from running BIT* and FB-BIT* in the Manhattan world, shown in red and green respectively.

Fig. 2: A UAV simulation of Manhattan shown on the left and the corresponding occupancy grid shown on the right.

are gathered from the NYC OpenData project [18]. The initial position of the UAV is located in Central Park. Maintaining an altitude of 10 meters, the UAV plans a path to the goal location on Governors Island while avoiding the buildings.

While planning a path, obstacles are represented with an occupancy grid with each pixel corresponding to ten square centimeters. Every building that is taller than $10m$ is considered an obstacle. Figure 2b shows the resulting occupancy grid with black representing obstacles. The occupancy grid covers a $10.7km \times 5.65km$ area of Manhattan. The initial location of the UAV is $x_r = \begin{bmatrix} 0 & 0 \end{bmatrix} km$, with the initial orientation in the direction of the target set. The target set, X_t, is a circle of radius $1mm$ centered at $\begin{bmatrix} -9 & -3.8 \end{bmatrix} km$. Paths are checked for obstacles four times per meter of path length. The neighborhood search radius is $\rho = 500m$. When using BIT* and FB-BIT*, the batch size, m, is 1500 samples and the target set samples, X_{goal}, is a singleton set of the center of X_t. When planning with fillets, $d_{init} = 150m$. The minimum turning radius imposed on the arc-fillet generation, r, is $150m$.

RRT* and FB-RRT* have three additional parameters, η, α, and b_t, which are described in [8] but only given values here for brevity. The steering constant, η, is $500m$. The max number of neighbors to search, α, is 100 neighbors. The check target period, b_t, is 1 out of every 50 samples.

Results are gathered using the Open Motion Planning Library (OMPL) [19]. As the sampling is random, each convergence plot is averaged from over 100 individual simulations. Convergence plots are made by fitting a 15$^{\text{th}}$-order polynomial to the data using a least-squares fitting algorithm as described in [20]. The simulation code can be found in our open-source repository https://gitlab.com/utahstate/robotics/fillet-rrt-star.

B. Results

Before convergence results are analyzed below, an example is given where the path generated by BIT* is infeasible due to the curvature constraints of the UAV. Note the two example paths planned with BIT* and FB-BIT* in Figure 2b. BIT*, planning with straight-line paths and shown in red, finds a shorter path than FB-BIT*. However, the UAV is unable to follow the path that BIT* generates because it turns sharply around corners with buildings on either side (see the section of the plan highlighted in yellow). FB-BIT*, planning with arc-fillets and shown in green, finds a different path through the buildings that can be followed by the UAV because it respects max curvature constraints.

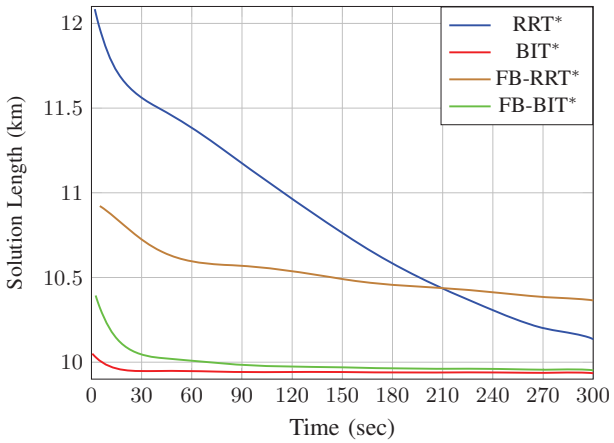


Fig. 3: Averaged convergence plots from 100 simulations in the Manhattan world over 5 minutes. RRT* and BIT* planning with straight-line motion primitives are shown in blue and red respectively. FB-RRT* and FB-BIT* planning with arc-fillets are shown in brown and green respectively.

The convergence results from benchmarking RRT*, BIT*, FB-RRT*, and FB-BIT* in the Manhattan environment are shown in Figure 3. The two planners planning with straight-lines are RRT* and BIT*, shown in blue and red, respectively. Clearly BIT* outperforms RRT* in terms of converging to a near-optimal solution rapidly. BIT* initially finds a solution much shorter than RRT* and proceeds to converge to near optimality by the time 30 seconds have passed. RRT* on the other hand starts with a long initial solution. Despite converging for five minutes, the solution that RRT* produces

is still substantially longer than that of BIT*.

The two planners planning with arc-fillets are FB-RRT* and FB-BIT*, shown in brown and green, respectively. Once again, the BIT*-based planner significantly outperforms the RRT*-based planner. FB-BIT* finds a moderately short initial solution and converges to almost the same length solution as BIT*. FB-RRT* starts with a longer initial solution than FB-BIT* and, over the five-minute period, converges slowly. Note that as time goes to infinity FB-BIT* and FB-RRT* will find the same length solution because they respect the same kinematic constraints. However, our simulation suggests that it will take much longer for FB-RRT* to find a near-optimal solution than FB-BIT*.

Comparing all four planners, some interesting patterns emerge. The most notable is that both of the BIT*-based planners far outperform the RRT*-based planners. This is particularly impressive because FB-BIT* respects max curvature and path continuity constraints that RRT* does not respect. This rapid convergence comes from BIT* and FB-BIT*'s use of cost-to-come and cost-to-go heuristics to focus their search efforts in directions that are most likely to improve the solution cost. Thanks to these heuristics, FB-BIT* can converge nearly as fast as BIT* while respecting curvature constraints.

When comparing RRT* to FB-RRT* it might seem odd that FB-RRT* finds a shorter solution than RRT* for the first 210 seconds. This is because there are many sub-optimal paths through the obstacle topology that FB-RRT* is unable to reach because of the curvature constraints it respects. RRT* on the other hand does not respect any kinematic constraints and, as such, can find local minima through the obstacles that result in long initial solutions. When RRT* finds a long initial solution it is easy for it to find shorter paths, which is why RRT*'s convergence plot decreases rapidly.

VI. Conclusion

In this work, a BIT*-based path-planning algorithm that uses general fillets as motion primitives is developed. The properties of fillets are analyzed in the context of sampling-based path-planning algorithms. The operations of BIT* are modified to function in the fillet-based path planning framework. A detailed, step-by-step explanation of FB-BIT* is given. Simulation results are presented where paths are planned for a UAV navigating the buildings of Manhattan. FB-BIT* is shown to generate paths that respect curvature constraints while converging nearly as quickly as BIT*, and much faster than RRT* and FB-RRT*.

Acknowledgments

This work was supported in part by Alion Science and Technology through the Air Force Research Laboratory under Grant 203463.

References

[1] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2997–3004.

[2] J. Nasir, F. Islam, U. Malik, Y. Ayaz, O. Hasan, M. Khan, and M. Muhammad, "Rrt*-smart: A rapid convergence implementation of rrt*," *International Journal of Advanced Robotic Systems*, vol. 10, 2013.

[3] I. Noreen, A. Khan, and Z. Habib, "Optimal path planning using rrt* based approaches: A survey and future directions," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 11, 2016. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2016.071114

[4] K. Yang, S. Moon, S. Yoo, J. Kang, N. L. Doh, H. B. Kim, and S. Joo, "Spline-based rrt path planner for non-holonomic robots," *Journal of Intelligent & Robotic Systems*, vol. 73, no. 1-4, pp. 763–782, 2014a.

[5] C. Moon and W. Ching, "Kinodynamic planner dual-tree rrt (dt-rrt) for two-wheeled mobile robots using the rapidly exploring random tree," *IEEE Transactions on Industrial Electronics*, vol. 62, pp. 1080–1090, 2015.

[6] S. Sedighi, D.-V. Nguyen, and K.-D. Kuhnert, "Guided hybrid a-star path planning algorithm for valet parking applications," in *2019 5th International Conference on Control, Automation and Robotics (ICCAR)*, 2019, pp. 570–575.

[7] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 528–564, 2016. [Online]. Available: https://doi.org/10.1177/0278364915614386

[8] J. Swedeen, G. Droge, and R. Christensen, "Fillet-based RRT*: A rapid convergence implementation of RRT* for curvature constrained vehicles," *Journal of Intelligent & Robotic Systems*, vol. 108, no. 4, p. 68, Jul. 2023.

[9] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011. [Online]. Available: https://doi.org/10.1177/0278364911406761

[10] I.-B. Jeong, S.-J. Lee, and J.-H. Kim, "Quick-rrt*: Triangular inequality-based implementation of rrt* with improved initial solution and convergence rate," *Expert Systems with Applications*, vol. 123, pp. 82–90, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417419300326

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

[12] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Batch informed trees (bit*): Informed asymptotically optimal anytime search," *The International Journal of Robotics Research*, vol. 39, no. 5, pp. 543–567, 2020. [Online]. Available: https://doi.org/10.1177/0278364919890396

[13] R. Beard and T. McLain, *Small Unmanned Aircraft: Theory and Practice*. Princeton University Press, 2012. [Online]. Available: https://books.google.com/books?id=YqQtjhPUaNEC

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

[15] P. Cui, W. Yan, and X. Guo, "Path planning for robot with pose constraints using dubins-rrt," in *2018 3rd International Conference on Advanced Robotics and Mechatronics (ICARM)*, 2018, pp. 560–565.

[16] X. Lan and S. Di Cairano, "Continuous curvature path planning for semi-autonomous vehicle maneuvers using rrt," in *2015 European Control Conference (ECC)*, 2015, pp. 2360–2365.

[17] J. Swedeen and G. Droge, "Batch informed trees (bit*)," 2023. [Online]. Available: https://arxiv.org/abs/2302.11670

[18] "Building footprints," May 2016. [Online]. Available: https://data.cityofnewyork.us/Housing-Development/Building-Footprints/nqwf-w8eh

[19] M. Moll, I. A. Sucan, and L. E. Kavraki, "Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization," *IEEE Robotics Automation Magazine*, vol. 22, no. 3, pp. 96–102, 2015.

[20] W. N. Venables and B. D. Ripley, *Modern Applied Statistics with S*, 4th ed. New York: Springer, 2002, iSBN 0-387-95457-0. [Online]. Available: https://www.stats.ox.ac.uk/pub/MASS4/

CHAPTER 6

FB-BIT* MISSION PLANNING OF AIRCRAFT UNDER THREAT OF DETECTION WITH REDUCED OBSERVABILITY

Submitted to *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*

FB-BIT* Mission Planning of Aircraft Under Threat of Detection with Reduced Observability

Journal Title
XX(X):1–32

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

James Swedeen[1], **Greg Droge**[1]

Abstract

The planning of missions for fixed-wing aircraft under threat of detection by adversarial radar systems is complicated by several factors. The aircraft often has to fly through global positioning system (GPS)-denied regions where exact localization is impossible. Resulting position estimation errors must be considered while planning, or else risk the aircraft getting closer to adversarial radar systems than is intended. Additionally, fixed-wing aircraft are restricted to following curvature-constrained paths due to the physical capabilities of the vehicle. In this work, the accuracy of a radar detection metric is improved by evaluating the state uncertainties of a representative closed-loop aircraft model. The calculation of the uncertainties is accelerated using linear covariance (LinCov) analysis. The cutting-edge path planning algorithm Fillet-Based Batch Informed Trees (FB-BIT*) is modified so that it can plan paths that respect the radar detection constraints. The proposed FB-BIT* algorithm is compared against two existing mission planners. Comparison shows that FB-BIT* far outperforms existing planners in terms of reliably finding a solution to highly constrained problems and the resulting optimality of the solutions produced.

Keywords

Radar Detection, Navigation, Mission Planning, Closed-Loop Linear Covariance (LinCov) Analysis, Sampling-Based Algorithms, Batch Informed Trees (BIT*), Fillet-Based Batch Informed Trees (FB-BIT*), Kinematic Constraints

1 Introduction

Unmanned aerial vehicles (UAVs) are increasingly being used in real-world domains, especially military and search and rescue operations[1]. UAVs are advantageous in these domains because their use minimizes fuel consumption, improves human safety, and can improve clandestine capabilities. UAVs are often used in applications where adversarial ground-based radar systems are scanning for unrecognized aircraft. These applications include reconnaissance[2], radar countermeasure deployment[3,4], and combat operations[5,6]. During these missions, any unrecognized aircraft that are detected by adversarial radar systems are downed, resulting in mission failure. Mission planners used in these domains must consider this risk.

Probability of detection (POD) is a measure of the likelihood that an UAV is detected by adversarial radar systems. When planning missions in adversarial environments, the POD must be kept to a minimum to avoid failure. Existing mission planners that minimize the POD typically rely on simple heuristics[4,7–9], with no guarantee of finding a solution, or direct optimization techniques[5,10], which scale poorly to large problems. Many of these works assume that the parameters that influence the POD are fixed and known in advance[2–5,7,8,10–14]. Unfortunately, the radar-specific parameters and position of the UAV are rarely known exactly. Furthermore, these estimation errors cause significant deviations in the POD[15,16]. These deviations must be controlled to ensure mission reliability. However, few works are available that take these deviations into account while planning.

An exception is the work developed in [9], where the POD variability framework derived in [15,16] is utilized to constrain an open-loop UAV model while planning. In the sequel, the accuracy of the framework is improved by defining a representative closed-loop fixed-wing UAV system simulation, with an inertial navigation-based extended Kalman filter (EKF) providing state estimates. The closed-loop nature of the simulation preserves the coupling between system dynamics, exogenous disturbances, guidance, navigation, and control, which were

[1]Electrical and Computer Engineering Department, Utah State University, Logan, UT, USA

Corresponding author:
James Swedeen, 270 W 750 S Apt. 4, Logan, UT, USA
Email: james.swedeen@usu.edu

not considered in previous works. These coupling terms can have a significant effect on the produced covariance estimates, especially in global positioning system (GPS)-denied regions. Therefore, this work contributes to the mission planning under threat of detection literature with a way to evaluate the POD with considerations for the effects of loop control when navigation estimates are poor, which have not been considered while planning in previous works. Additionally, the calculation of aircraft state covariances is accelerated using a derived closed-loop linear covariance (LinCov) framework.

The recently proposed FB-BIT* algorithm [17] is modified to accommodate constraints that depend on the path taken to reach a state. FB-BIT* is a sampling-based path planning algorithm designed for rapid convergence in large search spaces while respecting kinematic constraints. FB-BIT* is guaranteed to find the solution to a problem if one exists, and is very efficient in iteratively improving that solution. However, the original FB-BIT* algorithm can only respect constraints that are evaluated on a location-by-location basis, e.g., hitting walls in the absence of estimation errors, either a location is in a wall or it is not. The improved POD variability framework produces a constraint that depends on the location of evaluation and the path taken to get there. Hence, this work contributes to the field of sampling-based path planners with a version of FB-BIT* that can respect such constraints. In addition, an error in the original presentation of FB-BIT* is corrected with justification. The modified algorithm is benchmarked against and compared with preexisting algorithms in the literature.

The remainder of the paper proceeds as follows. Section 2 discusses the relevant existing literature. Section 3 describes the POD constraints of the problem. Section 4 defines the UAV model simulated in this work. Section 5 defines the methods used to propagate the uncertainties associated with the UAV. Section 6 details how FB-BIT* is modified for the probabilistic path planning problem. Section 7 compares FB-BIT* against two existing mission planners using an averaged simulation benchmark. Section 8 provides some concluding remarks. Appendix A provides a full list of acronyms and nomenclature.

2 Literature Review

This section provides an overview of the relevant literature. First, the field of generalized mission planning while respecting probabilistic constraints is discussed. Then, some of the ways in which the works have considered the POD are listed, followed by a discussion of the mission planning under threat of detection literature. Finally, the evolution of path planners that produced FB-BIT* is described. This overview of the literature allows for a clear statement of the contributions of this work, which is given at the end of the section.

Probabilistic Mission Planning

The mission-planning literature is full of works that plan paths while respecting probabilistic constraints. For example, [18] utilizes an Rapidly-exploring Random Tree (RRT)-based algorithm where a closed-loop UAV model is used to estimate the probability of collision with uncertain obstacles in GPS-denied areas. A similar problem is studied in [19], where environmental obstacles move in random ways over time. They develop a two-stage path generation scheme. In the first stage, Optimal RRT (RRT*) is used to generate a reference trajectory assuming that there is no obstacle movement. In the second stage, a model predictive controller solves a mixed integer convex program that considers the uncertainty in the position of the obstacles to generate the control signal.

Planning for unmanned ground vehicles to ensure path traversability is considered in [20]. A RRT*-based algorithm is developed in which the problem constraints ensure that the probability of getting stuck at a location is kept below a threshold. Additionally, B-splines are used to smooth the resulting path as a post-processing step.

Decision making for a general partially observable Markov decision process is considered in [21]. They develop an Monte-Carlo tree search to model the environment and plan optimal actions. Upper confidence bounds applied to trees is used to guide the search effort in the directions that are most likely to produce optimal results.

POD Mission Planning

This work is concerned with the POD of a fixed-wing UAV, which is a function of the physical characteristics of the aircraft, the position and orientation of the aircraft relative to the radar, and the parameters of the radar system [22]. When using the POD in mission planning algorithms, estimating it using high-fidelity simulation models takes a prohibitively long time. Many planning techniques utilize functional estimates of the POD because they can be calculated quickly. Common POD models include those based on the integrated inverse range [3,4,7,10], the peak or aggregate radar cross section (RCS) [11], and models that consider both [5,8,13,23].

The mission planning literature frequently makes the assumption when calculating the POD that the aircraft and radar positions/parameters are deterministic and known [2–5,7,8,10–14]. However, in practice, these values always have inherent uncertainty. Radar parameters are estimated from any intelligence gathering that is possible, and radar and aircraft positions are estimated through sensor measurements, which are always noisy and biased [24]. Furthermore, many missions where the threat of detection must be minimized will have to enter GPS-denied areas where the aircraft position estimates are poor. Without GPS, accurately tracking a planned trajectory becomes

impossible, resulting in large deviations from the planned route, possibly bringing the UAV closer to the radar stations than intended. It is shown in [15,16] that these deviations in aircraft position and radar parameters cause significant variation in the POD. Therefore, a mission planner must consider these uncertainties to ensure that the resulting plan keeps the POD below an acceptable threshold.

Most of the path planning under threat of detection literature can be categorized into heuristic-based planners with no optimality considerations and direct optimization-based planners, which are typically computationally expensive. Within the category of heuristic-based planners, most have a shared structure. A set of waypoints is generated for the UAV to follow, the POD is calculated along the resulting trajectory, and the waypoints are iteratively adjusted until POD constraints are satisfied. For example, [9] defines a polygon of waypoints around each radar, solves for the shortest path with Dijkstra's algorithm, and repeatedly moves the waypoints away from the radars when the POD constraints are violated. In [11], a greedy waypoint selection process is used where each waypoint is chosen to maximize its distance from the nearest radar. Some works avoid the need to iteratively refine their waypoints, instead using a POD model that is simple enough to allow them to define waypoints that are known to satisfy constraints. In [4], Delaunay triangulations are used to generate a waypoint graph that is known to satisfy constraints, and then Dijkstra's algorithm is used to find the shortest path through the graph. A Voronoi tessellation that has been adjusted to keep POD to a minimum is proposed in [7] to generate the waypoint graph. Other works avoid the need to define waypoints at all by using other control techniques; for example, potential field descent is used in [8].

Direct optimization-based planners typically formulate the problem as a mixed integer program and then solve the program with a third-party optimization library, e.g., [3,5]. Some works use metaheuristics such as particle swarm optimization to solve the program, e.g., [14]. Others use POD and vehicle models that are simple enough to allow an analytically optimal solution to be found, e.g., [10].

Sampling-Based Mission Planners

While most previous methods of solving the mission planner under threat of detection problem fall into the two categories listed above, this work uses a sampling-based algorithm. A sampling-based planning algorithm that has been studied extensively, and is used for comparison herein, is RRT[25]. In RRT, a rooted search tree grows from a starting location to every reachable configuration in the search space without violating the constraints of the problem. RRT has been used in a wide range of applications because of its speed and probabilistic completeness. However, the solution RRT produces is typically far from optimal, and the probability that RRT produces the optimal solution is zero[24]. RRT* was developed from RRT to provide asymptotic optimally guarantees[24]. It is similar to RRT, however, as RRT* builds its search tree, local optimizations are performed on the tree that cause the paths in the search tree to converge to optimality as the number of iterations performed approaches infinity. However, this convergence can be slow[26–28].

Fast Marching Tree (FMT*) is another sampling-based planning algorithm that uses the principles of dynamic programming to avoid unnecessary calculations[29]. FMT* builds a rooted search tree, similar to RRT*. However, instead of iteratively sampling the search space as the algorithm runs, FMT* samples a fixed number of times at startup. With knowledge of where samples are from the start, FMT* avoids many of the repetitive calculations that RRT* performs. However, it loses the ability that RRT* has to refine the solution indefinitely.

BIT* merges the iterative nature of RRT* with the efficient graph searching techniques of FMT*[30]. BIT* iteratively generates batches of samples from the search space, then uses a procedure similar to FMT* to incorporate the new batch of samples into the preexisting search tree. The characteristics of BIT* vary with the size of each batch of samples. When the batch size is only one, BIT* is nearly equivalent to RRT*. When the batch size is very large, BIT* is nearly equivalent to FMT*, except for considering subsequent batches. This allows the user of the algorithm to tune BIT* to have the desired performance characteristics.

In recent years, the problem of planning paths for nonholonomic vehicles has received increased attention[31–37]. One nonholonomic constraint of particular importance to this work is that of maximum curvature. Many vehicles have a maximum curvature restriction, e.g., cars, fixed-wing UAVs, and boats. In the case of a fixed-wing UAV, this constraint exists because the vehicle must maintain forward velocity while executing turns.

Recently, fillet-primitives have been incorporated into RRT* and BIT* to produce Fillet-Based RRT* (FB-RRT*) and FB-BIT*, respectively[17,34]. FB-RRT* and FB-BIT* work in the same way RRT* and BIT* do, except they use a fillet-based motion primitive to connect nodes in their search trees instead of straight-line connections. They naturally respect curvature constraints while combining subproblem solutions by adding additional constraints to local optimizations that guarantee global convergence. Like BIT*, FB-BIT* utilizes several heuristics and operation reordering techniques that minimize the number of times that edge generation and obstacle checking must be performed. These advances are critical to the proposed work because obstacle checking is an especially time-consuming process.

Statement of Contributions

Table 1. Features present in the mission planning literature.

	Paper	Constraints		POD Model		Position Estimation			Solution Properties		Solution Method			
		Probabilistic	Kinematic	Range	RCS	Truth Model	Kalman Filter	LinCov	Complete	Optimal	Exact	Heuristic	Metaheuristic	Sampling-based
POD Constraints	**Proposed**	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	✓
	[3]	-	✓	✓	-	-	-	-	✓	✓	✓	-	-	-
	[4]	-	-	✓	-	-	-	-	-	-	-	✓	-	-
	[5]	-	✓	✓	✓	-	-	-	-	✓	✓	-	-	-
	[7]	-	-	✓	-	-	-	-	✓	-	-	✓	-	-
	[8]	-	✓	✓	✓	-	-	-	-	-	-	✓	-	-
	[9]	✓	✓	✓	✓	-	✓	✓	-	-	-	✓	-	-
	[10]	-	-	✓	-	-	-	-	✓	✓	✓	-	-	-
	[11]	-	-	-	✓	-	-	-	-	-	-	-	-	-
General Planning	[2]	-	-	-	-	-	-	-	✓	✓	-	✓	-	-
	[12]	✓	-	-	-	-	-	-	-	-	-	✓	-	-
	[14]	✓	-	-	-	-	-	-	✓	✓	-	-	✓	-
	[17,32–37]	-	✓	-	-	-	-	-	✓	✓	-	-	-	✓
	[18]	✓	-	-	-	✓	✓	✓	✓	-	-	-	-	✓
	[19–21]	✓	✓	-	-	-	-	-	✓	✓	-	-	-	✓
	[24,26–30]	-	-	-	-	-	-	-	✓	✓	-	-	-	✓
	[25]	-	-	-	-	-	-	-	✓	-	-	-	-	✓
	[31,38]	-	✓	-	-	-	-	-	✓	-	-	-	-	✓

Table 1 compares the existing literature with the proposed work. The columns *Probabilistic* and *Kinematic Constraints* denote whether the work includes probabilistic or kinematic constraints in the planning process. Likewise, the *POD Model* columns show which works use POD models based on *Range*, *RCS*, or both. The *Position Estimation* columns describe how, if at all, the guidance and navigation is implemented. A paper having a check mark in the *Truth Model* column includes a representative vehicle model with a closed-loop controller in addition to a navigation model. The *Kalman Filter* and *LinCov* columns denote if the paper uses a Kalman filter for navigation estimates or LinCov for rapid covariance generation. *Complete* refers to an algorithm that is guaranteed to find a solution if one exists*. The *Optimal* column shows which papers produce optimal results or results that converge to optimality. The *Solution Method* columns denote the type of algorithm each paper uses.

As shown, few other mission planners consider a detailed closed-loop simulation like this work, and no existing POD aware planners incorporate a full closed-loop model of the UAV being planned for. Furthermore, [9] is the only existing work, to the knowledge of the authors, that considers the naturally probabilistic nature of the POD constraints. The one that does is neither complete nor optimal and does not consider a closed-loop simulation model. Most works in the path planning under threat of detection literature use heuristic planning methods with no guarantees of finding a solution when it exists. The few that employ exact solution strategies are typically slow to run and make use of a very simplified version of the POD constraints. However, this work uses a sampling-based algorithm that consistently plans paths despite the complicated constraints used.

Within the field of general path planning, sampling-based algorithms are very common, especially when probabilistic constraints are being considered. However, when probabilistic constraints are involved, most works use simplistic algorithms that are complete but sacrifice optimality. This work adds to the fields of probabilistic path planning and path planning under threat of detection with the use of FB-BIT*, a cutting-edge and advanced path planning algorithm with several advantageous properties. These properties include consistently producing a solution

*With a slight lapse in notation, this work is referring to many technical definitions of complete and optimal as if they were not distinct in Table 1.

when one exists, converging to an optimal solution given enough time, and highly efficient computational complexity.

3 Probability of Detection

This section describes the metric used to estimate the POD throughout this work and the subsequent mission constraint. The metric is designed to model single-pulse, nonfluctuating radar detection probabilities. The metric is a function of the RCS, the radar range, and the radar-specific parameters. The formulation of this metric was originally presented in [16] and is included here for completeness. It is not fundamental to the contributions of this work. Indeed, a mission planning problem with the requirement of avoiding obstacles in GPS-denied environments, as is studied in [18], could be improved with the same closed-loop UAV model proposed in this work and the use of FB-BIT* to solve the problem requires the same modifications proposed herein. This work uses a POD mission constraint because it is an excellent example of a situation where the proposed contributions are needed.

3.1 Radar Cross Section

When a radar system is scanning for an unknown object, it transmits electromagnetic waves, waits for the signals to bounce off an object, and listens for the reflected waves. The RCS is the intensity of the received "backscattered" signal that has the same polarization as the receiving antenna[23]. In other words, RCS is a measure of the visibility of the aircraft to a radar. Given that calculating the true RCS of an aircraft at a particular position is very computationally intensive, estimates of the RCS have been developed in the literature, e.g., [15]. This work uses the ellipsoidal RCS model[23]. This model captures the important characteristics of the RCS of a fixed-wing UAV and is continuously differentiable, which is needed in Section 3.3.

Let $p_r^n \in \mathbb{R}^3$ be the position of a radar in the north east down (NED) frame. The position vector of the radar relative to the aircraft in the aircraft body frame is given as

$$p_r^b \triangleq \begin{bmatrix} p_{r,x}^b & p_{r,y}^b & p_{r,z}^b \end{bmatrix}^T \triangleq R_\Theta^T(\Theta_b^n)(p_r^n - p^n), \tag{1}$$

where $p^n \in \mathbb{R}^3$ is the position of the aircraft in the NED frame and $\Theta_b^n \in \mathbb{R}^3$ is the rotation from the aircraft body frame to the NED frame as an Euler angle sequence. $R_\Theta : \mathbb{R}^3 \mapsto \mathbb{SO}$ is a mapping from a roll, pitch, yaw Euler angle sequence to a rotation matrix that represents the same rotation; see Appendix D for details.

The RCS azimuth, $\xi \in \mathbb{R}$, and elevation, $\zeta \in \mathbb{R}$, angles are measured from the x-axis of the aircraft body frame to the projection of p_r^b onto the xy-plane of the aircraft body frame and from the xy-plane of the aircraft body frame to p_r^b with the convention of a positive angle toward the bottom of the aircraft, respectively. These angles are calculated as

$$\xi \triangleq \arctan\left(\frac{p_{r,y}^b}{p_{r,x}^b}\right), \tag{2}$$

$$\zeta \triangleq \arctan\left(\frac{p_{r,z}^b}{\sqrt{\left(p_{r,x}^b\right)^2 + \left(p_{r,y}^b\right)^2}}\right). \tag{3}$$

The ellipse RCS model is a function of the azimuth and elevation angles and is given in (4), where $\mathfrak{a}, \mathfrak{b}, \mathfrak{c} \in \mathbb{R}_+$ are the lengths of each ellipsoid axis and $\mathrm{s}(\cdot), \mathrm{c}(\cdot)$ are shorthand for sin and cos.

$$\varrho \triangleq \frac{\pi(\mathfrak{a}\mathfrak{b}\mathfrak{c})^2}{\left((\mathfrak{a}\,\mathrm{s}(\xi)\,\mathrm{c}(\zeta))^2 + (\mathfrak{b}\,\mathrm{s}(\xi)\,\mathrm{s}(\zeta))^2 + (\mathfrak{c}\,\mathrm{c}(\xi))^2\right)^2} \tag{4}$$

3.2 Probability of Detection Metric

In this work, the POD, $P_D \in \mathbb{R}_{[0,1]}$, is calculated using the approximation (5)[23], where $p_d : \mathbb{R}^6 \times \mathbb{R}^3 \times \mathbb{R} \mapsto \mathbb{R}_{[0,1]}$ is the POD function, $P_{fa} \in \mathbb{R}_{[0,1]}$ is the probability of false alarm which is considered a constant for each radar system, and $\mathrm{erfc} : \mathbb{R} \mapsto \mathbb{R}_{[0,2]}$ is the complementary error function.

$$P_D \approxeq p_d(x_a, p_r^n, c_r) \triangleq \frac{1}{2}\,\mathrm{erfc}\left(\sqrt{-\ln P_{fa}} - \sqrt{S + 0.5}\right) \tag{5}$$

$S \in \mathbb{R}$ is the signal-to-noise ratio of the radar measurement. It is a function of radar-specific parameters, the RCS, and the aircraft to radar range, see (6), where $\kappa \in \mathbb{R}_+$ is Boltzmann's constant, $\|\cdot\| : \mathbb{R}^3 \mapsto \mathbb{R}_+$ is the two-norm, and $c_r \in \mathbb{R}_+$ is the consolidated radar constant. The consolidated radar constant is a function of the parameters of a specific radar, such as power, aperture area, loss factor, etc. The value of each parameter individually is not important in this work, so they are grouped into a single constant.

$$S \triangleq \frac{c_r \varrho}{\kappa \|p^n - p_r^n\|^4} \tag{6}$$

3.3 Probability of Detection Variance

The POD is a function of the aircraft pose, radar position, and the consolidated radar constant. None of these quantities are known perfectly. When traveling through a GPS-denied area, the variance of the aircraft pose grows quite large, and the variances of the other two quantities are usually smaller but not negligible. This leads to variability in P_D. Let the pose vector of the aircraft be defined as

$$x_a \triangleq \begin{bmatrix} (p^n)^T & (\Theta_b^n)^T \end{bmatrix}^T. \tag{7}$$

The perturbation of P_D is approximated to the first-order in (8), where the notation is used throughout this work that the partial derivative of a function with respect to a parameter is denoted as the capital of the function name with the subprescript of the variable to which the partial derivative is with respect to, i.e., ${}_{x_a}P_D \triangleq \frac{\partial p_d(x_a, p_r^n, c_r)}{\partial x_a}$.

$$\delta P_D \approxeq {}_{x_a}P_D \delta x_a + {}_{p_r^n}P_D \delta p_r^n + {}_{c_r}P_D \delta c_r \tag{8}$$

The partial derivatives used in (8) are derived in Appendix C. δP_D is used to calculate the variance of the POD, $\sigma_{pd}^2 \in \mathbb{R}_+$, in (9), where $R_{x_a} \in \mathcal{S}^6 \succeq 0, R_{p_r^n} \in \mathcal{S}^3 \succeq 0, R_{c_r} \in \mathbb{R}_+$ are the covariance matrices of x_a, p_r^n, and c_r, respectively. $R_{p_r^n}$ and R_{c_r} are taken as constants and R_{x_a} is calculated using techniques described in Section 5.

$$\begin{aligned}\sigma_{pd}^2 &\triangleq E\left[(\delta P_D)^2\right] \\ &\approxeq {}_{x_a}P_D R_{x_a} {}_{x_a}P_D^T + {}_{p_r^n}P_D R_{p_r^n} {}_{p_r^n}P_D^T + {}_{c_r}P_D^2 R_{c_r}\end{aligned} \tag{9}$$

3.4 Mission Requirement

The POD must always remain below a given threshold, $P_D^{max} \in \mathbb{R}_{(0,1)}$. To ensure that not only the average POD respects this requirement but also most of the realizations of the mission, the constraint is applied to the average POD plus three times its standard deviation (SD), i.e.,

$$P_D + 3\sigma_{pd} \leq P_D^{max}. \tag{10}$$

This ensures that at every instant in time along a solution path 99.7% of the realizations satisfy this bound. As is common in the literature, e.g., [3,4,7–11], this constraint does not reflect the overall probability of the UAV being detected throughout the entire mission. This constraint restricts the POD at every instant in time along the solution path. This is done because integrating the POD over time would require knowledge of the period at which the radar station transmits the detection signals, which is unknown. However, by reducing the POD at every point along the solution path, the constraint indirectly reduces the overall mission POD. Furthermore, the mission planner discussed in Section 6 minimizes the path length of the solution path, which also reduces the overall mission POD.

4 Aircraft Model

This section describes the closed-loop UAV simulation used to produce state uncertainties for the POD constraint evaluation. This includes definitions of truth, navigation, and reference state vectors and generation descriptions for each; see Sections 4.1 through 4.3. The mappings between the state definitions are defined in Section 4.4. A controller that keeps the truth model following the reference trajectory using the navigation state estimates is given in Section 4.5. Inertial measurements for propagating navigation states and error state covariances are defined in Sections 4.6 and 4.7. Finally, the EKF measurement update equations and the measurements simulated herein are given in Section 4.8.

4.1 Truth Model

The truth model used in this work is representative of fixed-wing UAVs. It mimics the characteristics of a UAV while flying in trim, that is, the forces and moments of the body are in equilibrium with the external forces produced through air pressure. Under these conditions, the dynamics of a fixed-wing UAV can be approximately decoupled into longitudinal motion, airspeed, pitch angle, and altitude, and lateral motion, roll and heading angles[39]. Let the truth state vector, $x \in \mathbb{R}^7$, be defined as

$$x \triangleq \begin{bmatrix}(p^n)^T & (\Theta_b^n)^T & V_a\end{bmatrix}^T, \tag{11}$$

where $p^n \in \mathbb{R}^3$ is the position of the aircraft in the NED frame. $\Theta_b^n \in \mathbb{R}^3$ is the Euler angles of the aircraft with respect to the NED frame, i.e.,

$$\Theta_b^n \triangleq \begin{bmatrix}\phi & \theta & \psi\end{bmatrix}^T, \tag{12}$$

where $\phi, \theta, \psi \in \mathbb{R}$ are the roll, pitch, and yaw of the aircraft, respectively. $V_a \in \mathbb{R}_+$ is the velocity at which the aircraft is traveling relative to the surrounding air.

The truth state dynamics of the UAV are defined in (13). They are modeled from the "Dubin's Airplane" model given in [39] and are similar to other path planning dynamic models used in the literature, e.g., [8]. $\beta, \alpha, A_a \in \mathbb{R}$ are components of the control vector, $u \in \mathbb{R}^3$; see Section 4.5 for a detailed description. $g \in \mathbb{R}_+$ is the magnitude of the acceleration felt by the aircraft as a result of gravity. $w_n, w_e, w_d \in \mathbb{R}$ are the North, East, and Down components of the white process noise, $w \triangleq \begin{bmatrix}w_n & w_e & w_d\end{bmatrix}^T$, which accounts for random perturbations in position that are caused by gusts of wind.

$$\dot{x} = f(x,u,w) \triangleq \begin{bmatrix} V_a\cos(\theta)\cos(\psi) + w_n \\ V_a\cos(\theta)\sin(\psi) + w_e \\ -V_a\sin(\theta) + w_d \\ \beta \\ \alpha \\ \frac{g}{V_a}\tan(\phi) \\ A_a \end{bmatrix} \tag{13}$$

Note that this model assumes the coordinated turn condition is satisfied when defining $\dot{\psi}$[39]. w is zero mean and has power spectral density (PSD), $Q_w \triangleq \begin{bmatrix}q_n & q_e & q_d\end{bmatrix}^T I_3$ where $q_n, q_e, q_d \in \mathbb{R}_+$ are the PSDs of w_n, w_e, and w_d, i.e.,

$$E\left[w(t_0)w^T(t_1)\right] = Q_w\delta(t_0 - t_1),\ \forall t_0, t_1 \in \mathbb{R}, \tag{14}$$

where $\delta : \mathbb{R} \mapsto \mathbb{R}$ is the Dirac delta function.

4.2 Navigation Model

The navigation state estimates the truth state using the standard inertial navigation method of integrating the inertial measurement unit (IMU) signal through time. The navigation state vector, $\hat{x} \in \mathbb{R}^{10}$, is defined in (15), where $\hat{p}^n \in \mathbb{R}^3$ and $\hat{v}^n \in \mathbb{R}^3$ are estimates of the position and velocity of the aircraft in the NED frame and the estimated attitude of the aircraft is given in the NED frame as the quaternion $\hat{q}_b^n \in \mathbb{H}$.

$$\hat{x} \triangleq \begin{bmatrix} (\hat{p}^n)^T & (\hat{q}_b^n)^T & (\hat{v}^n)^T \end{bmatrix}^T, \tag{15}$$

The navigation state dynamics are defined in (16), where $\tilde{\omega}^b \in \mathbb{R}^3$ and $\tilde{a}^b \in \mathbb{R}^3$ are components of the inertial measurement vector, $\tilde{y} \in \mathbb{R}^6$; see Section 4.6. $\otimes : \mathbb{H} \times \mathbb{H} \mapsto \mathbb{H}$ is the Hamiltonian quaternion product and $R_q : \mathbb{H} \mapsto \mathbb{SO}(3)$ is a function that maps a quaternion to a rotation matrix that represents the same rotation; see Appendix D for details. $g^n \in \mathbb{R}^3 \triangleq \begin{bmatrix} 0 & 0 & g \end{bmatrix}^T$ is the acceleration vector due to gravity in the NED frame.

$$\dot{\hat{x}} = \hat{f}(\hat{x}, \tilde{y}) \triangleq \begin{bmatrix} \hat{v}^n \\ \frac{1}{2}\hat{q}_b^n \otimes \begin{bmatrix} 0 \\ \tilde{\omega}^b \end{bmatrix} \\ R_q(\hat{q}_b^n)\tilde{a}^b + g^n \end{bmatrix} \tag{16}$$

4.3 Reference Trajectory

The reference trajectory defines the desired or nominal state of the UAV over time. The reference trajectory is generated by connecting a set of waypoints with a smoothed path that respects the maximum curvature constraints of the UAV. The waypoints are defined as north and east positions, with orientations defined between them. The waypoints are typically generated with a path planning algorithm as described in Section 6. The reference state vector $\bar{x} \in \mathbb{R}^9$ must include all the information needed to construct the truth and navigation states; see Section 5.2 for a detailed explanation of why. To this end, let

$$\bar{x} \triangleq \begin{bmatrix} (\bar{p}^n)^T & \left(\bar{\Theta}_b^n\right)^T & (\bar{v}^n)^T \end{bmatrix}^T, \tag{17}$$

where $\bar{p}^n \in \mathbb{R}^3$ is the reference position in the NED frame. $\bar{\Theta}_b^n \in \mathbb{R}^3$ is the roll, pitch, and yaw of the reference point. $\bar{v}^n \in \mathbb{R}^3$ is the reference UAV velocity in the NED frame.

The generation of a smooth path that follows a given set of waypoints is accomplished via fillets. A fillet connects two line segments (defined with three points) with a curve that smoothly transitions between them. Fillets are an efficient way to formulate curvature constraints because the curve portion of the fillet can be designed to respect any constraints required. Take Figure 1 as an example, where a smoothed path is made from the waypoints A, B, C, and D. The portions of the path where the path is straight are colored blue, and the curved portions are colored magenta. In this work, the curved part of the path is defined as an arc with a radius $r \in \mathbb{R}_+$.

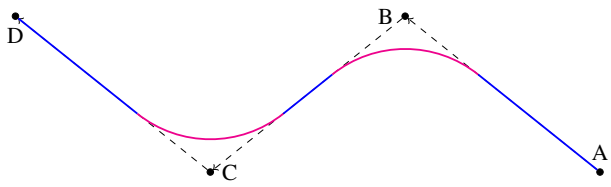


Figure 1. Fillet smoothed path connecting waypoints A, B, C, and D shown with blue straight-line sections and magenta curves.

The generation of the reference state vector along the straight-line segments is defined in (18), where $s \in \mathbb{R}_+$ is the length along the segment. $d_{nom} \in \mathbb{R}_+$ is the nominal aircraft down and $V_{nom} \in \mathbb{R}_+$ is the nominal aircraft ground speed. Both are constants that are determined on a scenario-by-scenario basis. $p_{s0,n}^n, p_{s0,e}^n \in \mathbb{R}$ is the north and east component of the starting point of the path segment, and $\psi_{s0} \in \mathbb{R}$ is the initial heading of the segment.

$$\bar{x}_s(s) = \begin{bmatrix} \begin{bmatrix} p_{s0,n}^n + s\cos(\psi_{s0}) \\ p_{s0,e}^n + s\sin(\psi_{s0}) \\ d_{nom} \end{bmatrix} \\ \begin{bmatrix} 0 \\ 0 \\ \psi_{s0} \end{bmatrix} \\ \begin{bmatrix} V_{nom}\cos(\psi_{s0}) \\ V_{nom}\sin(\psi_{s0}) \\ 0 \end{bmatrix} \end{bmatrix} \tag{18}$$

The generation of the reference state vector along the curved segments is defined as

$$\bar{x}_c(s) = \begin{bmatrix} \begin{bmatrix} p_{s0,n}^n - \iota r\sin(\psi_{s0}) + \iota\cos\left(\frac{s}{r}\right) \\ p_{s0,e}^n + \iota r\cos(\psi_{s0}) + \iota\sin\left(\frac{s}{r}\right) \\ d_{nom} \end{bmatrix} \\ \begin{bmatrix} \bar{\phi} \\ 0 \\ \psi_{s0} + \frac{\iota}{r}s \end{bmatrix} \\ \begin{bmatrix} V_{nom}\cos\left(\psi_{s0} + \frac{\iota}{r}s\right) \\ V_{nom}\sin\left(\psi_{s0} + \frac{\iota}{r}s\right) \\ 0 \end{bmatrix} \end{bmatrix}, \tag{19}$$

where

$$\bar{\phi} \triangleq \arctan\left(\frac{\iota V_{nom}^2}{rg}\right), \tag{20}$$

and $\iota \in \{-1, 1\}$ is used to distinguish between the directions of turning, i.e.,

$$\iota \triangleq \begin{cases} 1, & \text{counter-clockwise (left) turn} \\ -1, & \text{clockwise (right) turn.} \end{cases} \tag{21}$$

4.4 State Mappings

In the following sections, it will become necessary to map state vectors of one type to state vectors of other types. To that end, let the mapping from the truth state vector to the navigation state vector, $m : \mathbb{R}^7 \mapsto \mathbb{R}^{10}$, be defined as

$$m(x) \triangleq \begin{bmatrix} (p^n)^T & q_\Theta^T(\Theta_b^n) & V_a \left({}^1R_\Theta(\Theta_b^n)\right)^T \end{bmatrix}^T, \quad (22)$$

where $q_\Theta : \mathbb{R}^3 \mapsto \mathbb{H}$ is a function that maps a roll-pitch-yaw Euler angle sequence to a quaternion that represents the same rotation; see Appendix D for details. The notation ${}^1R_\Theta(\Theta_b^n)$ is used as shorthand for extracting just the first column of $R_\Theta(\Theta_b^n)$, i.e.,

$${}^1R_\Theta(\Theta_b^n) \triangleq R_\Theta(\Theta_b^n)\begin{bmatrix}1 & 0 & 0\end{bmatrix}^T. \quad (23)$$

Occasionally, mapping from the navigation state to the truth state is also needed. To this end, let this mapping be denoted as $m^{-1} : \mathbb{R}^{10} \mapsto \mathbb{R}^7$ and be defined in (24), where $\Theta_q : \mathbb{H} \mapsto \mathbb{R}^3$ is a mapping from quaternions to a column vector containing the Euler angles that correspond to the provided quaternion; see Appendix D for details.

$$m^{-1}(\hat{x}) \triangleq \begin{bmatrix} (\hat{p}^n)^T & \Theta_q^T(\hat{q}_b^n) & \|\hat{v}^n\| \end{bmatrix}^T \quad (24)$$

Now let the mapping from the reference state vector to the truth state vector, $\bar{m} : \mathbb{R}^9 \mapsto \mathbb{R}^7$, be defined as

$$\bar{m}(\bar{x}) \triangleq \begin{bmatrix} (\bar{p}^n)^T & \left(\bar{\Theta}_b^n\right)^T & \|\bar{v}^n\| \end{bmatrix}^T. \quad (25)$$

4.5 Controller

The controller is responsible for making the aircraft follow the reference trajectory. The controller in this work is broken down into feedforward and feedback terms. The feedforward term defines the control that takes the aircraft from one state along the reference trajectory to the next, assuming that there is no noise that corrupts the position of the aircraft. This term is needed so that the LinCov integration produces valid statistics; see Section 5. The feedback term counters any perturbations from the reference trajectory that occur due to process noise or estimation error. It uses a vector-field-based path following approach, which is particularly effective at rejecting the disturbances that UAVs encounter[40]. The control vector, $u \in \mathbb{R}^3$, is defined as

$$u \triangleq \begin{bmatrix}\beta & \alpha & A_a\end{bmatrix}^T, \quad (26)$$

where β is the commanded roll-rate, α is the commanded pitch-rate, and A_a is the commanded air-speed-rate. $a : \mathbb{R}^{10} \times \mathbb{R}^9 \times \mathbb{R}^9 \mapsto \mathbb{R}^3$ is the function that calculates the control and is defined as

$$u = a(\hat{x}, \bar{x}, \bar{x}_n) \triangleq a_{ff}(\bar{x}, \bar{x}_n) + a_{fb}(\hat{x}, \bar{x}), \quad (27)$$

where $\hat{x}$ is the current navigation state, $\bar{x}$ is the current reference state, and $\bar{x}_n$ is the next reference state in the reference trajectory. $a_{ff} : \mathbb{R}^9 \times \mathbb{R}^9 \mapsto \mathbb{R}^3$ is the feedforward part of the control and $a_{fb} : \mathbb{R}^{10} \times \mathbb{R}^9 \mapsto \mathbb{R}^3$ is the feedback part of the control.

The feedforward part of the control is derived using linear system theory. Consider linearizing (13) around the current reference state and using Euler integration to define a one-step update approximation as

$$\bar{m}(\bar{x}_n) \approxeq \bar{m}(\bar{x}) + \Delta t\left({}_x\bar{F}x + {}_u\bar{F}u\right), \quad (28)$$

where $\Delta t \in \mathbb{R}_+$ is the time difference between $\bar{x}$ and $\bar{x}_n$ and the process noise is assumed to be zero. The bar over this partial derivative denotes that the derivative has been evaluated along the reference trajectory, i.e., ${}_x\bar{F} \triangleq \left.\frac{\partial f(x,u,w)}{\partial x}\right|_{x=\bar{m}(\bar{x}),u=a_{ff}(\bar{x},\bar{x}_n),w=0_{3\times1}}$. The partial derivatives used in (28) are given in Appendix C for completeness. Solving (28) for u results in (29), where ${}_u\bar{F}^\dagger$ is the pseudoinverse of the matrix ${}_u\bar{F}$. $\ominus : \mathbb{R}^7 \times \mathbb{R}^7 \mapsto \mathbb{R}^7$ is element-wise subtraction of all states. The attitude subtraction is calculated so that the result always lies in the range $(-\pi, \pi]$; see Appendix D for details.

$$a_{ff}(\bar{x}, \bar{x}_n) \triangleq \frac{{}_u\bar{F}^\dagger\left(\bar{m}(\bar{x}_n) \ominus \left(I_7 + \Delta t {}_x\bar{F}\right)\bar{m}(\bar{x})\right)}{\Delta t} \quad (29)$$

The feedback control is defined using a method based on successive loop closure[39]. Before giving the control function itself, the reference centric position error, $\delta e^r \in \mathbb{R}^3$, and the desired course angle, $\psi_d \in \mathbb{R}$, are described. The reference centric position error is the difference between the current navigation state and the desired reference state in the reference trajectory coordinate frame.

$$\delta e^r \triangleq R_\Theta\left(\bar{\Theta}_b^n\right)(\bar{p}^n - \hat{p}^n) \quad (30)$$

The desired course angle is designed to drive the lateral position error to zero. When the lateral error is large, ψ_d points in the direction of the current reference point. As the lateral error decreases to zero, the difference between the reference and desired headings also diminishes to zero. This can be seen in (31), where $\psi_\infty \in \left(0, \frac{\pi}{2}\right]$ is the maximum difference between the desired and reference course angles, which is achieved when the lateral error is infinite. $k_{ct} \in \mathbb{R}_+$ is a constant gain that influences the rate at which ψ_d transitions from $\bar{\psi} \pm \psi_\infty$ to $\bar{\psi}$.

$$\psi_d \triangleq \bar{\psi} + \psi_\infty \frac{2}{\pi} \arctan\left(k_{ct}\begin{bmatrix}0 & 1 & 0\end{bmatrix}\delta e^r\right) \quad (31)$$

The feedback control is defined in (32) through (35), where $k_\psi^p, k_d^p, k_{at}^p, k_\psi^d, k_d^d, k_{at}^d \in \mathbb{R}_+$ are the proportional

and derivative gains of heading, down and along track, respectively.

$$\delta\dot{e}^r \triangleq -R_\Theta\left(\bar{\Theta}_b^n\right)\hat{v}^n \tag{32}$$

$$\hat{V}_a \triangleq \begin{bmatrix}1 & 0 & 0\end{bmatrix} R_q^T(\hat{q}_b^n)\hat{v}^n \tag{33}$$

$$\begin{bmatrix}\hat{\phi} & \hat{\theta} & \hat{\psi}\end{bmatrix}^T \triangleq \Theta_q(\hat{q}_b^n) \tag{34}$$

$$a_{fb}(\hat{x},\bar{x}) \triangleq \begin{bmatrix} k_\psi^p\left(\psi_d - \hat{\psi}\right) - k_\psi^d \frac{g}{\hat{V}_a}\tan\left(\hat{\phi}\right) \\ -\begin{bmatrix}0 & 0 & 1\end{bmatrix}\left(k_d^p\delta e^r + k_d^d\delta\dot{e}^e\right) \\ \begin{bmatrix}1 & 0 & 0\end{bmatrix}\left(k_{at}^p\delta e^r + k_{at}^d\delta\dot{e}^r\right)\end{bmatrix} \tag{35}$$

4.6 Inertial Measurements

Inertial measurements from a simulated IMU are used to propagate the navigation state in a technique called inertial navigation. The IMU measurement vector,

$$\tilde{y} \triangleq \begin{bmatrix}\left(\tilde{\omega}^b\right)^T & \left(\tilde{a}^b\right)^T\end{bmatrix}^T, \tag{36}$$

is composed of the measured angular rate, $\tilde{\omega}^b \in \mathbb{R}^3$, and specific force, $\tilde{a}^b \in \mathbb{R}^3$, of the UAV and given in the body frame. The IMU measurement is defined in (37), where $\omega^b \in \mathbb{R}^3$ is the true angular rate of the UAV in the body frame, $a^b \in \mathbb{R}^3$ is the true specific force felt by the UAV in the body frame, and $\eta \in \mathbb{R}^6$ corrupting additive noise.

$$\tilde{y} = i(x,u,\eta) \triangleq \begin{bmatrix}\left(\omega^b\right)^T & \left(a^b\right)^T\end{bmatrix}^T + \eta \tag{37}$$

η is zero mean and has PSD defined by

$$E\left[\eta(t_0)\eta^T(t_1)\right] = Q_\eta\delta(t_0 - t_1),\ \forall t_0, t_1 \in \mathbb{R}, \tag{38}$$

where

$$Q_\eta \triangleq \begin{bmatrix} q_g I_3 & 0_{3\times3} \\ 0_{3\times3} & q_a I_3\end{bmatrix}, \tag{39}$$

and $q_g, q_a \in \mathbb{R}_+$ are the PSDs of each element of the gyroscope and accelerometer measurements. Throughout this work, the notation is used that $0_{n\times m}$ is an n by m matrix of zeros and I_n is an n by n identity matrix, where it is understood that the identity matrix is always square.

In (40) the true angular rate of the UAV is expressed in terms of the truth state and control, where $T_\Theta : \mathbb{R}^3 \mapsto \mathbb{R}^{3\times3}$ is a mapping from Euler angles to the corresponding matrix that can be used to rotate Euler angles from one frame to another; see Appendix D for details.

$$\omega^b \triangleq T_\Theta^{-1}(\Theta_b^n)\begin{bmatrix}\beta & \alpha & \frac{g}{V_a}\tan(\phi)\end{bmatrix}^T \tag{40}$$

The true specific force in the body frame is expressed as

$$a^b \triangleq \begin{bmatrix}A_a \\ 0 \\ 0\end{bmatrix} + \left(\omega^b \times \begin{bmatrix}V_a \\ 0 \\ 0\end{bmatrix}\right) - R_\Theta^T(\Theta_b^n)g^n. \tag{41}$$

4.7 Error State Covariance

This section provides the equations needed to propagate the error state covariance in an EKF. The error state covariance, $P \in \mathcal{S}^9 \succeq 0$, is used in an EKF when updating the navigation state with noninertial measurements; see Section 4.8. The error state vector, $\delta x_e \in \mathbb{R}^9$, is defined as the difference between the truth and navigation states, i.e.,

$$\delta x_e \triangleq m(x)\,\hat{\ominus}\,\hat{x}, \tag{42}$$

where $\hat{\ominus} : \mathbb{R}^{10} \times \mathbb{R}^{10} \mapsto \mathbb{R}^9$ is element-wise subtraction of all states but attitude; see Appendix D for details. The error state dynamics are derived by taking the derivative with respect to time of (42) and linearizing as

$$\delta\dot{x}_e = {}_{\hat{x}}\hat{F}\delta x_e + {}_xM_w F w - {}_{\tilde{y}}\hat{F}_\eta C\eta. \tag{43}$$

The partial derivatives used in (43) are given in Appendix C for completeness. From (43) and the fact that $P \triangleq E\left[\delta x_e \delta x_e^T\right]$, the dynamics of P can be derived as the continuous-time Riccati equation (44).

$$\begin{aligned}\dot{P} = {}&{}_{\hat{x}}\hat{F}P + P{}_{\hat{x}}\hat{F}^T + {}_xM_w F Q_{ww} F^T {}_xM^T \\ &+ {}_{\tilde{y}}\hat{F}_\eta C Q_{\eta\eta} C^T {}_{\tilde{y}}\hat{F}^T\end{aligned} \tag{44}$$

4.8 Noninertial Measurements

As the EKF propagates forward in time, discrete-time noninertial measurements become available to it at periodic intervals. These measurements are used to update the navigation states and error state covariance.

4.8.1 Measurement Functions The EKF in this work processes several types of noninertial measurements. Each measurement model takes the form

$$\tilde{z}_j \triangleq h_j(x) + \nu_j, \tag{45}$$

where $\tilde{z}_j \in \mathbb{R}^{n_j}$ is a measurement of type j, $n_j \in \mathbb{R}_+$ is the number of states in measurement vectors of type j, $h_j : \mathbb{R}^7 \mapsto \mathbb{R}^{n_j}$ is a function that maps the truth state to the corresponding measured value, and $\nu_j \in \mathbb{R}^{n_j}$ is corrupting white noise on the measurement. All ν_js are zero-mean and have covariances, $R_j \in \mathcal{S}^{n_j} \succeq 0$, defined by

$$E\left[\nu_j(t_0)\nu_j^T(t_1)\right] = R_j\delta(t_0 - t_1),\ \forall t_0, t_1 \in \mathbb{R}\,. \tag{46}$$

For each measurement type, the estimated measurement model takes the form (47), where $\hat{\tilde{z}}_j \in \mathbb{R}^{n_j}$ is the best estimate of what the measurement would be if the aircraft was at state $\hat{x}$ and $\hat{h}_j : \mathbb{R}^{10} \mapsto \mathbb{R}^{n_j}$ is the measurement estimate function for measurement type j.

$$\hat{\tilde{z}}_j \triangleq \hat{h}_j(\hat{x}) \tag{47}$$

The first set of noninertial measurements used in the work is provided by a global navigation satellite system or GPS package. The GPS package provides measurements of the position and heading of the aircraft. When GPS measurements are available, each is received $\Delta t_{gps} \in \mathbb{R}_+$ seconds after the last. Because this work is concerned with planning in adversarial environments, GPS will be jammed in some regions, meaning that when the aircraft is in those regions no GPS measurements can be received. The true measurement functions for GPS measurements of position and coarse angle are given by

$$h_{gps,p}(x) \triangleq p^n \quad \text{and} \quad h_{gps,\psi}(x) \triangleq \psi. \tag{48}$$

Likewise, the measurement estimate functions are given by

$$\hat{h}_{gps,p}(\hat{x}) \triangleq \hat{p}^n \quad \text{and} \tag{49}$$

$$\hat{h}_{gps,\psi}(\hat{x}) \triangleq \begin{bmatrix}0 & 0 & 1\end{bmatrix}\Theta_q(\hat{q}^n_b). \tag{50}$$

While in GPS-denied regions, a terrain camera is used to produce information about the position of the UAV relative to known features in the environment. Specifically, the measurement the terrain camera produces is a line of sight (LOS) vector between the UAV and a preloaded map feature projected onto the focal plane of the camera[41,42]. The truth model for the LOS measurement is given by

$$h_{los}(x) \triangleq \begin{bmatrix}\frac{p^b_{f,x}}{p^b_{f,z}} & \frac{p^b_{f,y}}{p^b_{f,z}}\end{bmatrix}^T, \tag{51}$$

where $p^b_{f,x}, p^b_{f,y}, p^b_{f,z} \in \mathbb{R}$ are components of the LOS vector, $p^b_f \in \mathbb{R}^3$. The LOS vector is defined as the position vector of the feature in the aircraft body frame, i.e.,

$$p^b_f \triangleq \begin{bmatrix}p^b_{f,x} & p^b_{f,y} & p^b_{f,z}\end{bmatrix}^T \triangleq R^T_\Theta(\Theta^n_b)\left(p^n_f - p^n\right), \tag{52}$$

where $p^n_f \in \mathbb{R}^3$ is the position of the feature in the NED frame. The measurement estimate function is given by

$$\hat{h}_{los}(\hat{x}) \triangleq \begin{bmatrix}\frac{\hat{p}^b_{f,x}}{\hat{p}^b_{f,z}} & \frac{\hat{p}^b_{f,y}}{\hat{p}^b_{f,z}}\end{bmatrix}^T, \tag{53}$$

where

$$\hat{p}^b_f \triangleq \begin{bmatrix}\hat{p}^b_{f,x} & \hat{p}^b_{f,y} & \hat{p}^b_{f,z}\end{bmatrix}^T \triangleq R^T_q(\hat{q}^n_b)\left(p^n_f - \hat{p}^n\right). \tag{54}$$

In addition to feature LOS information, a laser range finder is used to measure the distance between the UAV and each feature within range. The truth model and estimate function of the feature range measurement are given as

$$h_{fr}(x) \triangleq \left\|p^n_f - p^n\right\| \quad \text{and} \quad \hat{h}_{fr}(\hat{x}) \triangleq \left\|p^n_f - \hat{p}^n\right\|. \tag{55}$$

The final set of three noninertial measurements are always available at their respective periods $\Delta t_j \in \mathbb{R}_+$ where j denotes the measurement type. While they are always available, they are significantly noisier than the measurements described above. They also do not provide a way for the EKF to fix its position the same way that the previous noninertial measurements do. There is a measurement of heading provided by a digital compass, a measurement of altitude provided by an absolute pressure sensor, and a measurement of airspeed provided by a differential pressure sensor[39]. The true measurement functions are defined as

$$h_\psi(x) \triangleq \psi, \tag{56}$$

$$h_{ap}(x) \triangleq -\rho g\begin{bmatrix}0 & 0 & 1\end{bmatrix}p^n, \text{ and} \tag{57}$$

$$h_{dp}(x) \triangleq \frac{\rho}{2}V_a, \tag{58}$$

where $\rho \in \mathbb{R}_+$ is the density of air. The measurement estimate functions are given as

$$\hat{h}_\psi(\hat{x}) \triangleq \begin{bmatrix}1 & 0 & 0\end{bmatrix}\Theta_q(\hat{q}^n_b), \tag{59}$$

$$\hat{h}_{ap}(\hat{x}) \triangleq -\rho g\begin{bmatrix}0 & 0 & 1\end{bmatrix}\hat{p}^n, \text{ and} \tag{60}$$

$$\hat{h}_{dp}(\hat{x}) \triangleq \frac{\rho}{2}\begin{bmatrix}1 & 0 & 0\end{bmatrix}R^T_q(\hat{q}^n_b)\hat{v}^n. \tag{61}$$

4.8.2 Update Equations When discrete-time noninertial measurements become available, the EKF updates the navigation state with

$$\hat{x}^+ = \hat{x}^- + \hat{K}_j\left(\tilde{z}_j - \hat{\tilde{z}}_j\right), \tag{62}$$

where $\hat{x}^-$ and $\hat{x}^+$ are known as a priori and a posteriori estimates and are the estimated value of x immediately before and after the measurement update, respectively. $\hat{K}_j \in \mathbb{R}^{10\times n_j}$ is the Kalman gain of measurement type j. In this work, the Kalman gain is calculated as

$$\hat{K}_j = P^-\,{}_{\hat{x}}\hat{H}^T_j\left({}_{\hat{x}}\hat{H}_j P^-\,{}_{\hat{x}}\hat{H}^T_j + R_j\right)^{-1}, \tag{63}$$

where ${}_{\hat{x}}\hat{H}_j$ is the Jacobian of the measurement estimate function $\hat{h}_j$. The Jacobians of every measurement function are given in Appendix C for completeness. The error state covariance is also updated with the noninertial measurements. This work uses the Joseph form of the update given in (64)[43].

$$\begin{aligned}P^+ = &\left(I_9 - \hat{K}_{j\,\hat{x}}\hat{H}_j\right)P^-\left(I_9 - \hat{K}_{j\,\hat{x}}\hat{H}_j\right)^T \\ &+ \hat{K}_j R_j \hat{K}^T_j\end{aligned} \tag{64}$$

5 Pose Variance Generation

As shown in (9), the covariance of the aircraft pose, R_{x_a}, is needed to calculate the variance of the POD. R_{x_a} is a time-varying function of the aircraft model and, in general,

can be computationally intensive to calculate. This section describes two common approaches taken to calculate R_{x_a} and compares the two.

Both approaches calculate R_{x_a} by first calculating a related quantity, the truth state covariance. The truth state covariance, $R_{true} \in \mathcal{S}^7$, is defined as

$$R_{true} \triangleq E\Big[(x - \bar{m}(\bar{x}))(x - \bar{m}(\bar{x}))^T\Big], \tag{65}$$

where the assumption has been made that the average truth state is equal to the reference state mapped into the truth state space, i.e., $\bar{m}(\bar{x}) \equiv E[x]$. From the truth state covariance R_{x_a} can be found via the similarity transformation

$$R_{x_a} = \begin{bmatrix} I_6 & 0 \end{bmatrix} R_{true} \begin{bmatrix} I_6 & 0 \end{bmatrix}^T. \tag{66}$$

5.1 Monte Carlo Analysis

The most direct way of calculating the truth state covariances is a Monte Carlo simulation. In a Monte Carlo simulation, the equations described in Section 4 are used to propagate the system along a reference trajectory of interest N times. The truth state covariance can then be approximated with[44]

$$R_{true} \approxeq \frac{1}{1-N} \sum_{i=1}^{N} (x_i - \bar{m}(\bar{x}))(x_i - \bar{m}(\bar{x}))^T, \tag{67}$$

where x_i is the truth state generated during the ith Monte Carlo run. The accuracy of the approximation increases as N increases. Typical values of N range from 100 to 1000.

Using Monte Carlo analysis to generate the truth state covariance is typically a very time-consuming process. This is because the full system has to be propagated forward N times before the truth state covariance is produced from post-processing. In this work, the truth state covariance is used in planners that require the evaluation of thousands of trajectories. A faster method must be found to make planning with FB-BIT* feasible.

5.2 Linear Covariance Analysis

LinCov analysis is a way to rapidly generate truth and navigation state covariances[9,18,41,42,44]. It has been studied before and is described here for completeness. LinCov works by defining an augmented state vector that contains the difference between the truth and navigation states and the reference trajectory. The covariance of this augmented state is then propagated along the reference trajectory. The truth covariance is extracted from the augmented covariance using a simple transformation matrix.

Let the truth state dispersions, $\delta x \in \mathbb{R}^7$, and navigation state dispersions, $\delta\hat{x} \in \mathbb{R}^9$, be defined as

$$\delta x \triangleq x \ominus \bar{m}(\bar{x}) \quad \text{and} \quad \delta\hat{x} \triangleq \hat{x} \,\hat{\ominus}\, m(\bar{m}(\bar{x})), \tag{68}$$

where $\ominus : \mathbb{R}^7 \times \mathbb{R}^7 \mapsto \mathbb{R}^7$ is element-wise subtraction of all states but attitude. The attitude subtraction is calculated such that the result always lies in the range $(-\pi, \pi]$; see Appendix D for details. The state propagation equations (13) and (16) are linearized along the reference trajectory along with (27) and (37) to produce (69) and (70).

$$\delta\dot{x} = {}_x\bar{F}\delta x + {}_u\bar{F}_{\hat{x}}\bar{A}\delta\hat{x} + {}_w\bar{F}w \tag{69}$$

$$\delta\dot{\hat{x}} = {}_{\tilde{y}}\bar{\hat{F}}_x\bar{I}\delta x + \Big({}_{\hat{x}}\bar{\hat{F}} + {}_{\tilde{y}}\bar{\hat{F}}_u\bar{I}_{\hat{x}}\bar{A}\Big)\delta\hat{x} + {}_{\tilde{y}}\bar{\hat{F}}_\eta\bar{I}\eta \tag{70}$$

The measurement update equation (62) is also linearized along the reference trajectory along with (45) and (47) to produce

$$\delta\hat{x}^+ = \Big(I_9 - \bar{K}_{j\hat{x}}\bar{\hat{H}}_j\Big)\delta\hat{x}^- + \bar{K}_{jx}\bar{H}_j\delta x + \bar{K}_j\nu_j, \tag{71}$$

where

$$\bar{K}_j = P^-{}_{\hat{x}}\bar{\hat{H}}_j^T\Big({}_{\hat{x}}\bar{\hat{H}}_jP^-{}_{\hat{x}}\bar{\hat{H}}_j^T + R_j\Big)^{-1}. \tag{72}$$

Note that the P^- in (72) comes from integrating (44) and (64) along the reference trajectory.

Let the augmented state vector, $X \in \mathbb{R}^{16}$, be defined as

$$X \triangleq \begin{bmatrix} (\delta x)^T & (\delta\hat{x})^T \end{bmatrix}. \tag{73}$$

The dynamics and measurement update equations of X are derived from (69), (70), and (71) as

$$\dot{X} = \mathcal{F}X + \mathcal{G}\eta + \mathcal{W}w \qquad \text{and} \tag{74}$$

$$X^+ = \mathcal{A}_jX^- + \mathcal{D}\nu_j, \tag{75}$$

where

$$\mathcal{F} \triangleq \begin{bmatrix} {}_x\bar{F} & {}_u\bar{F}_{\hat{x}}\bar{A} \\ {}_{\tilde{y}}\bar{\hat{F}}_x\bar{I} & {}_{\hat{x}}\bar{\hat{F}} + {}_{\tilde{y}}\bar{\hat{F}}_u\bar{I}_{\hat{x}}\bar{A} \end{bmatrix}, \quad \mathcal{G} \triangleq \begin{bmatrix} 0_{7\times6} \\ {}_{\tilde{y}}\bar{\hat{F}}_\eta\bar{I} \end{bmatrix}, \tag{76}$$

$$\mathcal{A}_j \triangleq \begin{bmatrix} I_7 & 0_{7\times9} \\ \bar{K}_{jx}\bar{H}_j & I_9 - \bar{K}_{j\hat{x}}\bar{\hat{H}}_j \end{bmatrix}, \quad \mathcal{W} \triangleq \begin{bmatrix} {}_w\bar{F} \\ 0_{9\times3} \end{bmatrix}, \tag{77}$$

$$\text{and} \qquad \mathcal{D}_j \triangleq \begin{bmatrix} 0_{7\times n_j} \\ \bar{K}_j \end{bmatrix}. \tag{78}$$

Note that $E[x] \equiv E[\bar{m}(\bar{x})]$ and $E[\hat{x}] \equiv E[m(\bar{m}(\bar{x}))]$ which in conjunction with (68) implies that the augmented state vector is zero-mean. The augmented state covariance, $C_A \in \mathcal{S}^{16} \succeq 0$, is defined as

$$C_A \triangleq E\big[XX^T\big]. \tag{79}$$

Using (74), (75), and (79), the following propagation and measurement update equations can be derived.

$$\dot{C}_A = \mathcal{F}C_A + C_A\mathcal{F}^T + \mathcal{G}Q_\eta\mathcal{G}^T + \mathcal{W}Q_w\mathcal{W}^T \tag{80}$$

$$C_A^+ = \mathcal{A}_jC_A^-\mathcal{A}_j^T + \mathcal{D}_jR_j\mathcal{D}_j^T \tag{81}$$

Table 2. The time it took LinCov and Monte Carlo analysis to generate the linearization validation statistics.

	LinCov	Monte Carlo
Time (sec)	0.16	208.865

Once the augmented state covariance has been propagated along a reference trajectory with (80) and (81), the truth state covariance can be extracted from C_A via

$$R_{true} = \begin{bmatrix} I_7 & 0_{7\times 9} \end{bmatrix} C_A \begin{bmatrix} I_7 & 0_{7\times 9} \end{bmatrix}^T. \tag{82}$$

Note that while only one set of dynamics is propagated once, the resulting augmented covariance still accounts for all navigation errors, control errors, and external disturbances such as noise sources. Thus, LinCov analysis produces an accurate approximation of the same covariances as a full Monte Carlo simulation; see Appendix B for accuracy validation results.

5.3 Timing Comparison

A timing comparison is made in this section showing how much LinCov analysis reduces the time it takes to generate pose uncertainty statistics compared to Monte Carlo simulations. The comparison is performed in the linearization validation scenario described in Appendix B and shown in Figure 7. This scenario has a path length of $8421\ m$ and takes an hour for the UAV to traverse. The Monte Carlo analysis is averaged over 1000 individual simulations to ensure accuracy, while the LinCov analysis only takes one integration along the reference trajectory to produce the same statistics.

Table 2 shows how long it took to produce the statistics plotted in Appendix B. As can be seen, Monte Carlo simulations took more than 1300 times the time that the LinCov analysis took. This shows the value of developing LinCov analysis when it is necessary to produce many sets of statistics, e.g., when performing an error budget analysis[9]. This work requires LinCov because a single planning run requires the generation and POD constraint evaluation of thousands of edges. Without LinCov analysis, the planning methods described herein would take weeks or months to produce results.

6 Mission Planning

Sections 4 and 5 describe a method for calculating the aircraft pose covariance. This covariance is necessary for evaluating the POD constraint described in Section 3. It remains to define the waypoints that the reference trajectory follows. As described in Section 4.3, the generation of the reference trajectory requires the north and east positions of each waypoint. All other variables in the reference trajectory generation are specified by a mission requirement or are functions of other parameters. To avoid confusion, the space in which the proposed planner plans in is denoted $\mathcal{X} \subseteq \mathbb{R}^2$, which contains the north and east components of the waypoint positions.

This section defines a mission planner called FB-BIT* that generates reference waypoints while meeting mission requirements and minimizing the resulting path length. These mission requirements include the POD constraint given in (10) and the kinematic constraints imposed by the dynamics of the aircraft. The most important kinematic constraint is the maximum possible path curvature. This constraint exists because it is impossible for a fixed-wing UAV to change the direction it is traveling in instantaneously. FB-BIT* was originally proposed in [17], however, that version of the algorithm is incompatible with the POD constraints, as will be described in the sequel. As such, this work proposes a significant modification to the FB-BIT* algorithm that makes it applicable to the problem considered herein. Additionally, there is an error in the way [17] defines the conditions under which a node can be removed from consideration. This error is corrected with justification for the correction in the sequel.

This section proceeds as follows. Section 6.1 covers the notation needed to describe FB-BIT*. Section 6.2 discusses the considerations that must be made when using fillets. FB-BIT* is described in Section 6.3. Much of this section is stated similarly in [17] and is given here for clarity when describing the proposed modifications to FB-BIT*.

6.1 Planning Notation

RRT* and BIT*-based algorithms construct a rooted search tree that finds constraint-satisfying paths through the state space. The tree is an acyclic-directed graph denoted $T \triangleq (V, E)$, where $V \subset \mathcal{X}$ is the set of nodes or vertices within the tree and $E \subseteq V \times V$ denotes the set of edges between the vertices. The root vertex, $x_r \in V$, has no parent while all other vertices have exactly one parent. Vertices can have any number of children. Vertices are added to the tree to find a path to the target set $\mathcal{X}_t \subset \mathcal{X}$. The paths in T must avoid situations where the POD exceeds the given threshold, that is, the constraint (10) must be satisfied for all paths in T. The notation $X \xleftarrow{+} \{x\}$ and $X \xleftarrow{-} \{x\}$ is used to compactly represent the set update operations $X \leftarrow X \cup \{x\}$ and $X \leftarrow X \setminus \{x\}$, respectively. Several tree building and updating procedures are now defined.

Definition 1. $x_p \leftarrow Par(x_c)$. *Returns the parent node of $x_c \in V$ in the tree T, or $\emptyset$ if x_c is the root node.*

Definition 2. $X_{children} \leftarrow Children(x_p)$. *Returns every node in T that has $x_p \in V$ as its parent.*

Definition 3. $X_{sol} \leftarrow Solution(x)$. *Finds the path through T that leads from the root node to $x \in V$.*

Definition 4. $\mathcal{X}_{rand} \leftarrow Sample(n)$. *Generates a set of $n \in \mathbb{Z}_+$ random samples of $\mathcal{X}$. If a solution has been found, these random samples are biased towards the smart and informed beacon set with a sampling bias of $b_b \in \mathbb{Z}_+$. The smart and informed beacon set roughly follows the currently found best solution; see [34] for details. The smart and informed sampling heuristic is used to accelerate the convergence of the solution. In addition, a sample rejection mechanism is used to improve the asymptotic complexity of FB-BIT*. Specifically, after drawing a sample, the POD constraint in (10) is tested with an aircraft state covariance of zero, that is, $R_{x_a} = 0_{6\times6}$. If the constraint fails in this situation, the sample can never be included in the tree without violating (10). As such, the sample is discarded and redrawn.*

Definition 5. $\mathcal{X}_{near} \leftarrow Near_l(x, V)$. *Finds all vertices in the vertex set V that are within a given heuristic edge cost radius, $l \in \mathbb{R}_+$, of the point $x \in \mathcal{X}$, i.e.,*

$$Near_l(x, V) \triangleq \{v \in V \setminus \{x_r\} \mid \hat{c}(Par(x), x, v) \leq l\}, \tag{83}$$

where $\hat{c} : \mathcal{X} \times \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is an edge cost heuristic function for the fillet that connects x to v assuming $Par(x)$ is the parent of x; see Section 6.2.3. Note that the root vertex, x_r, is removed from this search because a fillet cannot be formed directly off the root vertex, as described in Section 6.2.2.

Definition 6. $cost \leftarrow BestVal(\mathcal{Q}_i)$. *Finds the element of the generic queue $\mathcal{Q}_i$ with the lowest queue cost and returns the queue cost of that element.*

Definition 7. $x \leftarrow PopBest(\mathcal{Q}_i)$. *Finds the element of the generic queue $\mathcal{Q}_i$ with the lowest queue cost, removes it from the queue, and returns that element.*

6.2 Fillet Considerations

Given three points $x_1, x_2, x_3 \in \mathcal{X}$, a fillet connects x_1 to x_3 with a combination of two straight-line segments and a curve; see Figure 2. The line segments are portions of the lines $\overline{x_1x_2}$ and $\overline{x_2x_3}$. The curve intersects the line $\overline{x_1x_2}$ at the point $x_{s,2}$ and the line $\overline{x_2x_3}$ at $x_{e,2}$. The resulting fillet moves along the straight line from x_1 to $x_{s,2}$, along the curve from $x_{s,2}$ to $x_{e,2}$, and then along the straight line from $x_{e,2}$ to x_3.

FB-BIT* requires symmetric fillets, which means that there is an equivalent distance between the center point of the fillet, x_2, and the two ends of the fillet curve, $x_{s,2}$ and $x_{e,2}$, i.e., $\|x_2 - x_{s,2}\| \equiv \|x_2 - x_{e,2}\|$. Let that distance be denoted as $d(\gamma_2)$, where $\gamma_2 \in \mathbb{R}_+$ is the change in orientation between $\overrightarrow{x_1x_2}$ and $\overrightarrow{x_2x_3}$; see Figure 2. Arc-fillets are used in this work, where the curve portion of the fillet is a circle of radius $r \in \mathbb{R}_+$ that is tangential to the two line segments at $x_{s,2}$ and $x_{e,2}$. Lemma 1 defines the arc-fillet distance function, d.

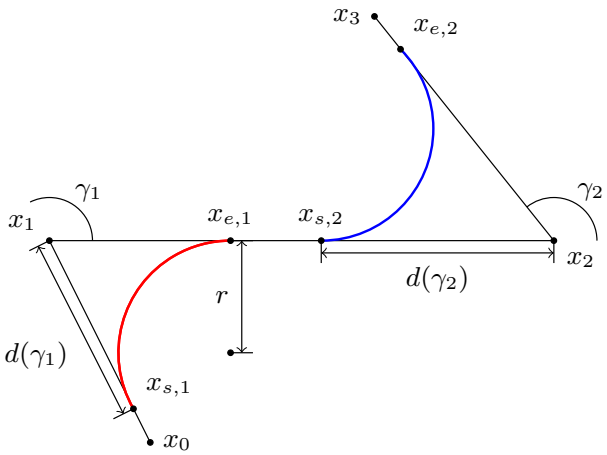


Figure 2. The fillet generated to connect x_1 and x_3 is shown in blue with the fillet that comes before it shown in red.

Lemma 1. See [34]. *The arc-fillet distance, $d : [0, \pi) \mapsto \mathbb{R}_+$, for a circular curve of radius $r \in \mathbb{R}_+$ is*

$$d(\gamma) \triangleq \frac{r(1 - \cos(\gamma))}{\sin(\gamma)}. \tag{84}$$

6.2.1 Path Continuity Conditions When using fillets for path planning, it is necessary to ensure that the path resulting from joining multiple fillets together is continuous. There are two conditions to ensure continuity: one to ensure that the fillet curve ends before the final point in the fillet, and another to ensure that all fillet curves end before the next begins[34]. Let $x_i \in \mathcal{X}$ be the middle point of a fillet connecting $x_{i-1} \in \mathcal{X}$ to $x_{i+1} \in \mathcal{X}$, these conditions are expressed as

$$\begin{aligned} d(\gamma_i) &\leq \|x_i - x_{i+1}\| \\ d(\gamma_{i-1}) + d(\gamma_i) &\leq \|x_i - x_{i-1}\| \end{aligned}. \tag{85}$$

The first condition ensures that the end of the curved portion of the fillet ends before x_{i+1} is reached. Consider Figure 2, for the blue curve, this condition is expressed as $d(\gamma_2) \leq \|x_2 - x_3\|$ or put another way $\|x_2 - x_{e,2}\| \leq \|x_2 - x_3\|$. If this condition is violated, the curved portion of the fillet would stick out past x_3. The second condition ensures that the red and blue curves never overlap. This can be seen by noting that the condition is expressed as $d(\gamma_1) + d(\gamma_2) \leq \|x_1 - x_2\|$ or, in other words, as $\|x_1 - x_{e,1}\| + \|x_2 - x_{s,2}\| \leq \|x_1 - x_2\|$. These conditions are integrated into how BIT* operates to produce FB-BIT*.

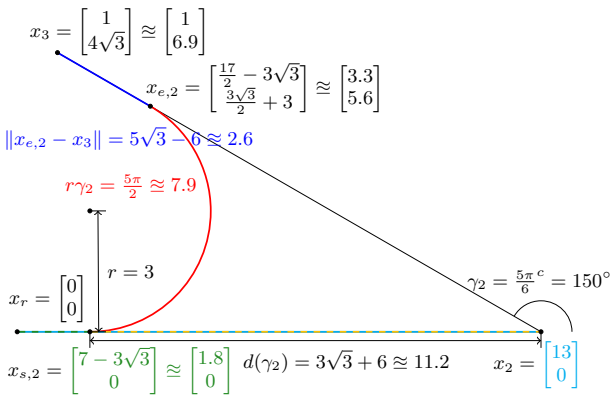


Figure 3. An example where the cost of the edge that connects x_2 to x_3 is negative. Note that $g_T(x_2) = 13$ and $g_T(x_3) = g_T(x_2) - d(\gamma_2) + r\gamma_2 + \|x_{e,2} - x_3\| \approx 12.3$ which means that $c(x_r, x_2, x_3) \approx -0.7$.

6.2.2 Initial Orientation Constraint One of the ways FB-BIT* is modified to plan for curvature-constrained vehicles is by adding an initial orientation constraint to the root node. This constraint ensures that the generated path will move from the starting position in the direction of the initial orientation. This constraint can be enforced with the use of a pseudoroot node. Given the initial position, $x_r \in \mathcal{X}$, and an initial orientation, $\psi_r \in [-\pi, \pi)$, the pseudoroot node is defined in (86), where $d_{init} \in \mathbb{R}_+$ is how far the pseudoroot node, $x_{sr} \in \mathcal{X}$, is from the root node, x_r.

$$x_{sr} \triangleq x_r + d_{init}\begin{bmatrix}\cos(\psi_r) & \sin(\psi_r)\end{bmatrix}^T \tag{86}$$

While initializing the search tree, x_{sr} is added to the tree as a child of x_r. The search tree is then grown by extending edges from x_{sr}. Note that extending edges from x_r is impossible because it takes three points to make a fillet and, by definition, x_r has no parent. However, x_{sr} has a defined parent, and any path that starts by going from x_r to x_{sr} will satisfy the initial orientation constraint by construction.

6.2.3 Cost Functions FB-BIT* makes use of several value functions that are used to guide the search in directions that are likely to produce an optimal solution. $g_T(x) : V \mapsto \mathbb{R}_+$ is the cost-to-come from the root node to $x \in V$ through T, which is defined as the length of the path that goes from the root node to x along the edges of the tree.

The relationship between the cost of a node and that of its parent is important to sampling-based planners. In sampling-based path planners, the cost of an edge is typically strictly positive. This intuitively makes sense as the path the edge represents goes from one node to another, and the distance between two points is always greater than or equal to zero. Assuming straight-line path connections leads to the recurrence relationship

$$\begin{aligned} g_T(x_3) &= g_T(x_2) + \|x_2 - x_3\| \\ &= g_T(x_2) + c(x_2, x_3), \end{aligned} \tag{87}$$

where $x_2, x_3 \in V$ such that x_2 is the parent of x_3 and $c : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_+$ is the path length or cost of the edge that connects x_2 to x_3. When planning with fillets, the cost of an edge becomes more complicated. As shown in Figure 3, the cost of node x_3 can be stated as

$$g_T(x_3) = g_T(x_2) - d(\gamma_2) + r\gamma_2 + \|x_{e,2} - x_3\| \tag{88}$$

$$= g_T(x_2) + c(x_1, x_2, x_3), \tag{89}$$

where $r\gamma_2$ is the length of the curve shown in red, because the curve is an arc of radius r, and $c(x_1, x_2, x_3)$ is the cost difference between x_2 and x_3, assuming x_1 is the parent of x_2. The intuition behind $c(x_1, x_2, x_3) = -d(\gamma_2) + r\gamma_2 + \|x_{e,2} - x_3\|$ is that it subtracts the distance between x_2 and the start of the red curve, then adds the path length from $x_{s,2}$ to x_3. In FB-BIT*, the cost function of an edge must also consider whether the edge satisfies the constraints of the problem. Namely, the POD condition in (10) and the continuity constraints in (85). When these constraints are violated, the cost of the edge is infinity to ensure the edge will not be used. Putting this together, the cost of the edge that connects $x_3 \in \mathcal{X}$ to $x_2 \in V$, assuming $x_1 \in V$ is the parent of x_2 is defined as

$$c(x_1, x_2, x_3) \triangleq \begin{cases} c_{potential} & \text{If (85), (10) satisfied} \\ \infty & \text{Otherwise} \end{cases}, \tag{90}$$

$$c_{potential} \triangleq -d(\gamma_2) + r\gamma_2 + \|x_{e,2} - x_3\|. \tag{91}$$

Note that $c : V \times V \times \mathcal{X} \mapsto \mathbb{R}$ is not guaranteed to be positive, see Figure 3, and c is dependent upon x_1 in addition to x_2 and x_3. This change in the properties of c leads to changes in the structure of FB-BIT*, as will be seen in the sequel.

$\hat{c}(x_1, x_2, x_3)$ is a lower bounded heuristic for the cost of an edge that connects $x_2 \in \mathcal{X}$ to $x_3 \in \mathcal{X}$ assuming that the fillet that ends at x_3 starts at $x_1 \in \mathcal{X}$, i.e., $\forall x_1, x_2, x_3 \in \mathcal{X}$,

$$\hat{c}(x_1, x_2, x_3) \leq c(x_1, x_2, x_3) \leq \infty. \tag{92}$$

$\hat{c} : \mathcal{X} \times \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is defined as the path length of the edge that connects x_3 to x_2 when the first condition in the continuity conditions (85) is satisfied and infinity otherwise. Drawing from (85) and (91), $\hat{c}$ is defined as

$$\hat{c}(x_1, x_2, x_3) \triangleq \begin{cases} c_{potential} & \text{If } d(\gamma_2) \leq \|x_2 - x_3\| \\ \infty & \text{Otherwise} \end{cases}. \tag{93}$$

Note that the POD conditions are not part of this calculation, making it a fast calculation compared to c.

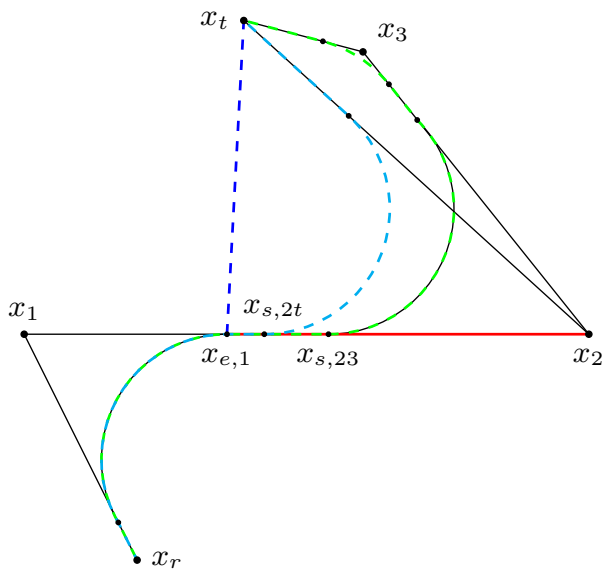


Figure 4. A demonstration of the cost-to-go heuristic for vertex x_2. The cost-to-go estimate of x_2 is given as $\hat{v}(x_2, x_1) = \|x_{e,1} - x_t\| - \|x_{e,1} - x_2\|$. $x_{s,23}$ is the start of the curve centered at x_2 if x_3 is the next vertex and $x_{s,2t}$ is the start of the curve if x_t is next.

$\hat{g} : \mathcal{X} \mapsto \mathbb{R}_+$ is defined as

$$\hat{g}(x) \triangleq \|x_r - x\|, \tag{94}$$

and is a lower-bounded heuristic of $g_T(x)$, i.e., $\forall x \in V$,

$$\hat{g}(x) \leq g_T(x) < \infty. \tag{95}$$

$\hat{v}(x_2, x_1)$ is a lower bounded heuristic of the true optimal cost-to-go from $x_2 \in \mathcal{X}$ to the target set assuming that x_2 is connected to $x_1 \in V$, that is, $x_1 = Par(x_2)$ if $x_2 \in V$. $\hat{v} : \mathcal{X} \times V \mapsto \mathbb{R}$ is defined as (96), where $x_t \in \mathcal{X}_t$ is the end of the lowest-cost solution found so far and $x_{e,1}$ is the end point of the curve portion of the fillet centered at x_1; see Figure 4. The reason behind measuring the distance to the target set from $x_{e,1}$ instead of x_2 is that $x_{e,1}$ is the closest point to x_2 that is guaranteed to be on any solution path where x_1 is the parent of x_2. To see why, consider the green and cyan dashed lines in Figure 4 that represent the solution paths generated from the waypoints $\{x_r, x_1, x_2, x_3, x_t\}$ and $\{x_r, x_1, x_2, x_t\}$, respectively. As you can see, the starting position of the curve centered at x_2 depends on the waypoint that follows and can reside anywhere on the line between $x_{e,1}$ and x_2.

$$\hat{v}(x_2, x_1) \triangleq \|x_{e,1} - x_t\| - \|x_{e,1} - x_2\| \tag{96}$$

6.3 Fillet-Based Batch Informed Trees

FB-BIT* and BIT* function by repeatedly generating batches of samples from the state space and incorporating those new samples into the preexisting search tree. To achieve this, they define two queues, the vertex and edge queues, which are sorted in terms of the quality of the potential solutions that can be made from those vertices and edges. At the beginning of each batch, the vertex queue is populated with all the nodes currently in the tree. Vertices are then iteratively removed from the vertex queue, and all potential edges that start at that vertex are added to the edge queue. Once all potential edges between samples of the state space have been found, the search tree is updated to include them if doing so will shorten the length of the resulting path. When both queues become empty, a new batch of samples is generated, and the process begins again.

Where FB-BIT* differs from BIT* are the conditions under which an edge can be added to the tree. In BIT* any two near vertices can be connected with an edge, assuming that the edge is obstacle-free. When planning with fillets, only edges that respect the conditions in (85) can be used in the tree, as described in Section 6.2.1. The added logic needed to ensure these conditions are respected appears throughout FB-BIT*, as will be seen in this section.

FB-BIT* is given in Algorithm 1. The inputs are the root node, $x_r \in \mathcal{X}$, the initial orientation, $\psi_r \in [-\pi, \pi)$, the

Algorithm 1: *FB-BIT**

In: $x_r, \psi_r, d_{init}, \mathcal{X}_{goal}, \mathcal{X}_t$
Out: $\mathcal{X}_{sol}$

$x_{sr} \leftarrow x_r + d_{init}\begin{bmatrix}\cos(\psi_r) & \sin(\psi_r)\end{bmatrix}^T$;
$V \leftarrow \{x_r, x_{sr}\}$; $E \leftarrow \{(x_r, x_{sr})\}$; $T \triangleq (V, E)$;
$\mathcal{Q}_V \leftarrow \{x_{sr}\}$; $\mathcal{Q}_E \leftarrow \emptyset$; $\mathcal{Q} \triangleq (\mathcal{Q}_V, \mathcal{Q}_E)$;
$\mathcal{X}_{ncon} \leftarrow \mathcal{X}_{goal}$; $\mathcal{X}_{new} \leftarrow \mathcal{X}_{goal}$;
$V_{exp} \leftarrow \emptyset$; $V_{rewire} \leftarrow \emptyset$;
$c_{sol} \leftarrow \min_{v_t \in V \cap \mathcal{X}_t} g_T(v_t)$;
$X_{flags} \triangleq (\mathcal{X}_{ncon}, \mathcal{X}_{new}, V_{exp}, V_{rewire}, c_{sol})$;
repeat
  if $\mathcal{Q}_V = \emptyset \wedge \mathcal{Q}_E = \emptyset$ **then** `// Batch end`
    `// Prune sub-optimal nodes`
    $\{T, X_{flags}\} \leftarrow Prune(T, X_{flags})$;
    `// Generate new batch of samples`
    $\mathcal{X}_{new} \leftarrow Sample(n)$;
    `// Add new samples to queues`
    $\mathcal{X}_{ncon} \xleftarrow{+} \mathcal{X}_{new}$;
    $\mathcal{Q}_V \leftarrow V \setminus \{x_r\}$;
  else if $BestVal(\mathcal{Q}_V) \leq BestVal(\mathcal{Q}_E)$ **then**
    `// Process best vertex available`
    $\{\mathcal{Q}, X_{flags}\} \leftarrow ExpVert(T, \mathcal{Q}, X_{flags})$;
  else `// Process best edge available`
    $\{T, \mathcal{Q}, X_{flags}\} \leftarrow ExpEdge(T, \mathcal{Q}, X_{flags})$;
until *STOP*;
return $Solution\big(\arg\min_{v_t \in V \cap \mathcal{X}_t} g_T(v_t)\big)$;

pseudoroot node distance, $d_{init} \in \mathbb{R}_+$, a sampling of the target set, $\mathcal{X}_{goal} \subseteq \mathcal{X}_t$, and the target set, $\mathcal{X}_t \subset \mathcal{X}$. Lines 1 and 2 initialize the search tree, T, with x_r and x_{sr} as described in Section 6.2.2.

FB-BIT* maintains two queues: the vertex queue, $\mathcal{Q}_V$, and the edge queue, $\mathcal{Q}_E$. $\mathcal{Q}_V$ is used to keep track of the vertices in T that are under consideration for making potential edges, and $\mathcal{Q}_E$ is used to keep track of the potential edges that are under consideration for addition to T. RRT*-based algorithms omit the use of vertex and edge queues at the cost of not being able to control the execution order of time-consuming operations like nearest-neighbor searching and obstacle checking. BIT*-based algorithms perform these operations in order of the greatest chance to improve the current best solution found. When the cost of the current solution decreases, the operations that cannot contribute to a better solution are removed from consideration, reducing the overall computational complexity of the algorithm.

The vertices in $\mathcal{Q}_V$ are processed in order of minimum current cost-to-come plus the heuristic cost-to-go of the vertices given their current parent in the tree, i.e., the queue cost of $v \in \mathcal{Q}_V$ is

$$g_T(v) + \hat{v}(v, Par(v)). \tag{97}$$

This is a lower-bounded estimate of the cost of any solution that is restricted to using the path from the root node to v; in other words, it is a lower bound on the cost of any solution that will result from adding children to v. The potential edges in $\mathcal{Q}_E$ are processed in order of minimal sum of the current cost-to-come of the edge's source vertex, the heuristic cost of the edge, and the heuristic cost-to-go through the edge, i.e., the queue cost of $(v, x) \in \mathcal{Q}_E$ is

$$g_T(v) + \hat{c}(Par(v), v, x) + \hat{v}(x, v). \tag{98}$$

This is a lower-bounded heuristic for the cost of any solution that is restricted to the use of edge (v, x). This means that edge expansion occurs only if there is a potential for that edge to contribute to the optimal solution. These queues are initialized on line 3 with no potential edges and only the pseudoroot node to consider for expansion.

Lines 4 and 5 initialize sets that keep track of the state of each vertex. $\mathcal{X}_{ncon} \subset \mathcal{X}$ is the set of all samples of the state space that are *not connected* to the search tree yet. $X_{new} \subset \mathcal{X}$ is the set of all samples that are *new* in this batch. $V_{exp} \subseteq V$ and $V_{rewire} \subseteq V$ are the sets of vertices that have been considered for *expansion* and *rewiring*, respectively. The inclusion of X_{new}, V_{exp}, and V_{rewire} in the algorithm prevents redundant calculations during vertex expansion, see Section 6.3.2, and significantly reduces the algorithmic complexity of FB-BIT*, see [30] for details. Line 6 initializes $c_{sol} \in \mathbb{R}_+$ with the cost of the current best solution or infinity if no solution has been found. Line 7 defines the tuple X_{flags} that is used purely to condense the notation in the sequel.

Algorithm 2: $Prune$

In: $T \triangleq (V, E)$,
$X_{flags} \triangleq (\mathcal{X}_{ncon}, \mathcal{X}_{new}, V_{exp}, V_{rewire}, c_{sol})$
Out: T, X_{flags}

1 **forall** $v \in V$ **do**
2 **if** $g_T(v) + \hat{v}(v, Par(v)) > c_{sol}$ **then**
  `// Remove v from the tree`
3 $V \xleftarrow{-} \{v\}$; $E \xleftarrow{-} \{(Par(v), v)\}$;
4 $V_{exp} \xleftarrow{-} \{v\}$; $V_{rewire} \xleftarrow{-} \{v\}$;
  `// Add v to the unconnected set`
5 $\mathcal{X}_{ncon} \xleftarrow{+} \{v\}$;
6 $\mathcal{X}_{ncon} \xleftarrow{-} \{x \in \mathcal{X}_{ncon} | \hat{g}(x) + \hat{v}(x, x_r) \geq c_{sol}\}$;
7 **return** $\{T, X_{flags}\}$;

The loop on lines 8 through 18 performs the planning and ends when a user-defined stopping condition is met[†]. The conditionals on lines 9 and 14 determine if the batch has ended and whether to process a vertex from $\mathcal{Q}_V$ or an edge from $\mathcal{Q}_E$, respectively. Each case is discussed in the following.

6.3.1 Generating New Batches When $\mathcal{Q}_V$ and $\mathcal{Q}_E$ are both empty, it signifies the end of the batch and the need to generate a new set of samples, see line 9 of Algorithm 1. Before sampling the configuration space, the $Prune$ procedure is called on line 10. $Prune$ performs two operations that improve the algorithmic performance of FB-BIT*. First, it removes samples and vertices from consideration that are not able to contribute to the optimal solution. Second, it prunes vertices in the search tree that, given the current state of the tree, cannot contribute to a better solution than the current best found. These pruned vertices are kept under consideration by leaving them in the unconnected set in the hope that future versions of the tree will yield better connections.

The $Prune$ procedure is given in Algorithm 2. The loop on lines 1 through 5 removes any vertices in the search tree that cannot contribute to a better solution given their current connection to T. The condition on line 2 checks if v lacks the potential to contribute to a better solution given its current connection to T. Lacking this potential means that the cost of the vertex plus a lower-bounded estimate of the cost-to-go is greater than the cost of the current solution,

[†]Common stopping conditions include achieving a desirable solution cost or expending the extent of the planning time given.

i.e., $v \in V$ lacks potential if

$$g_T(v) + \hat{v}(v, Par(v)) > c_{sol}. \tag{99}$$

This condition ensures that there is no way to connect v to the target while producing a better solution than the current best solution without v having a different parent in the tree. The work that proposes FB-BIT* has an error in the statement of this condition[17]. In [17], condition (99) is evaluated on v and all of the descendants of v. v is only pruned if all descendants of v have no potential to contribute to the best solution. However, checking all descendants of v is unnecessary because the condition (99) being true for v implies that the condition will be true for all descendants of v. Take, for example, Figure 4. No matter where x_3 is placed, its cost plus cost-to-go will always be greater than that of x_2, that is, $g_T(x_3) + \hat{v}(x_3, x_2) \geq g_T(x_2) + \hat{v}(x_2, x_1)$ no matter where x_3 is. Hence, if condition (99) is true for x_2, it must be true for all descendants of x_2.

If v cannot contribute to an improved solution given its current parent, v is removed from T on lines 3 and 4. v is then put in the not connected set on line 5 in case a different connection to T can benefit the search. Line 6 removes all unconnected samples from consideration that cannot contribute to the optimal solution. Note that a sample $x \in \mathcal{X}$ cannot contribute to optimal solutions if its heuristic cost-to-come plus heuristic cost-to-go value is greater than the current best solution cost, i.e.,

$$\hat{g}(x) + \hat{v}(x, x_r) \geq c_{sol}. \tag{100}$$

This intuitively makes sense because $\hat{g}(x) + \hat{v}(x, x_r)$ is a lower-bounded estimate of the cost of any solution that uses x. The root node, x_r, is used as the "parent" of x in this calculation because this choice minimizes the heuristic cost-to-go of any sample of $\mathcal{X}$, i.e.,

$$x_r \equiv \arg\min_{x_p \in \mathcal{X}} \hat{v}(x, x_p), \qquad \forall x \in \mathcal{X}. \tag{101}$$

Thus, $\hat{v}(x, x_r)$ is a lower-bounded heuristic for the cost-to-go of x given any connection to the tree.

After $Prune$ has finished, line 11 of Algorithm 1 generates $n \in \mathbb{Z}_+$ new samples of $\mathcal{X}$ and marks them as new this batch. Line 12 adds these new samples to the unconnected set. Line 13 adds all but the root node to $\mathcal{Q}_V$. This ensures that all vertices in T will be considered when looking for ways to connect the new samples to T. x_r is left out of $\mathcal{Q}_V$ because, in the fillet-based framework, an edge cannot be made from a node without a parent and, by definition, x_r has no parent.

6.3.2 Expanding Vertices Lines 14 and 15 of Algorithm 1 find potential edges to add to $\mathcal{Q}_E$ from the vertices in $\mathcal{Q}_V$. The condition on line 14 is true until it is impossible for any vertex in $\mathcal{Q}_V$ to produce an edge of lower heuristic cost than any edge in $\mathcal{Q}_E$. This can be seen by noting that the vertex queue cost (97) lower bounds the edge queue cost (98) of any edge that is expanded from the vertex, i.e., $\forall v \in \mathcal{Q}_V$ and $\forall x \in \mathcal{X}$ let $v_p \triangleq Par(v)$ then

$$g_T(v) + \hat{v}(v, v_p) \leq g_T(v) + \hat{c}(v_p, v, x) + \hat{v}(x, v) \tag{102}$$

as $\hat{v}(v, Par(v))$ is an underestimate of the true cost-to-go of vertex v given its connection to the tree.

The $ExpVert$ procedure removes the lowest-cost vertex in $\mathcal{Q}_V$ and adds edges to $\mathcal{Q}_E$ if they could be part of the optimal solution. $ExpVert$ is given in Algorithm 3. Line 1 of Algorithm 3 removes the lowest-cost vertex in $\mathcal{Q}_V$ from $\mathcal{Q}_V$. Lines 3 through 8 add edges to $\mathcal{Q}_E$ that go from v_b to unconnected samples of the state space. The condition on line 3 checks if this is the first time v_b has been considered for expansion. In that case, all unconnected samples are considered for connection to v_b; see line 5. If v_b has been considered for expansion before, only the samples that are new in this batch are considered for connection; see line 7. This avoids redundant calculations, as the samples that are not new in this batch have already been considered for connection to v_b during a previous batch. Line 8 adds all edges connecting to near and unconnected samples that have the potential to improve the current solution to $\mathcal{Q}_E$.

Algorithm 3: $ExpVert$

In: $T \triangleq (V, E), \mathcal{Q} \triangleq (\mathcal{Q}_V, \mathcal{Q}_E)$, $X_{flags} \triangleq (\mathcal{X}_{ncon}, \mathcal{X}_{new}, V_{exp}, V_{rewire}, c_{sol})$

Out: $\mathcal{Q}, X_{flags}$

1 $v_b \leftarrow PopBest(\mathcal{Q}_V)$;
2 $v_p \leftarrow Par(v_b)$;
`// Edges to unconnected samples`
3 **if** $v_b \notin V_{exp}$ **then**
4 $\quad V_{exp} \xleftarrow{+} \{v_b\}$;
5 $\quad \mathcal{X}_{near} \leftarrow Near_l(v_b, \mathcal{X}_{ncon})$;
6 **else** `//` v_b `has been expanded before`
7 $\quad \mathcal{X}_{near} \leftarrow Near_l(v_b, \mathcal{X}_{new} \cap \mathcal{X}_{ncon})$;
8 $\mathcal{Q}_E \xleftarrow{+} \{(v_b, x), x \in \mathcal{X}_{near} | \hat{g}(v_b) + \hat{c}(v_p, v_b, x) + \hat{v}(x, v_b) < c_{sol}\}$;
`// If` v_b `has not been rewired`
9 **if** $v_b \notin V_{rewire} \wedge c_{sol} < \infty$ **then**
10 $\quad V_{rewire} \xleftarrow{+} \{v_b\}$;
`// Edges to connected nodes`
11 $\quad V_{near} \leftarrow Near_l(v_b, V)$;
12 $\quad \mathcal{Q}_E \xleftarrow{+} \{(v_b, w), w \in V_{near} | (v_b, w) \notin E, \hat{g}(v_b) + \hat{c}(v_p, v_b, w) + \hat{v}(w, v_b) < c_{sol}, \hat{g}(v_b) + \hat{c}(v_p, v_b, w) < g_T(w)\}$;
13 **return** $\{\mathcal{Q}, V_{flags}\}$;

Lines 9 through 12 handle the case where v_b could be a better parent for its neighbors than their current parent, that is, in RRT* terminology, rewiring. Note that if no solution has been found, the condition on line 9 is always false. This reduces the time it takes to find an initial solution by skipping potential tree rewirings. Once the first solution is found, all rewirings that were skipped previously are considered. The reason finding an initial solution quickly is important is that once a solution is found, c_{sol} will no longer be infinite, and operations that cannot produce a better solution can be skipped. The condition on line 9 is also false if v_b has been considered for rewiring before. This avoids redundant calculations, as the potential to perform rewirings around v_b has already been considered. Line 11 finds all vertices in T that are near v_b. Line 12 adds all edges from v_b to $w \in V_{near}$ that are not already part of the tree, have the potential to improve the current solution, and have the potential to improve the cost of w. Note that $ExpVert$ uses only heuristic cost evaluations, which makes the procedure computationally lightweight and fast because LinCov evaluations are not necessary.

6.3.3 Evaluating Possible Edges In the case where the lowest possible heuristic cost edge has been generated from $\mathcal{Q}_V$, the condition on line 14 of Algorithm 1 is false and $ExpEdge$ is called. $ExpEdge$ removes the most promising edge from $\mathcal{Q}_E$ and considers it for addition to the search tree.

The $ExpEdge$ procedure is given in Algorithm 4. Line 1 removes the lowest queue cost edge from the queue $\mathcal{Q}_E$. Line 3 checks whether the edge under consideration does not have the potential to improve the current solution. Note that this condition is true only if the most promising edge in $\mathcal{Q}_E$, and by extension all edges that can be made between the samples that have been drawn so far, cannot contribute to the optimal solution. For this reason, $\mathcal{Q}_E$ and $\mathcal{Q}_V$ are cleared on line 4 to avoid redundant calculations and trigger the next batch of samples.

Line 6 checks if x_b is already part of the tree. If x_b is not part of T, line 7 checks if connecting x_b to the tree through v_b can improve the current solution. If it can, x_b is added to T with v_b as its parent. Line 10 checks if x_b is in the target set and updates the current solution cost if it is.

If x_b is already part of T at line 6, additional checks are performed before adding the edge under consideration to the tree. The conditions are evaluated with heuristics first because evaluating the true costs requires a time-consuming integration, and evaluating the heuristic costs can rule out edges where the conditions are not met. Line 12 checks if connecting x_b through v_b can improve the cost of x_b. Line 14 checks that this operation can improve the cost of the current solution and does improve the cost of x_b, respectively. This includes LinCov integration so that the POD constraints can be checked while calculating c.

Algorithm 4: $ExpEdge$

In: $T \triangleq (V, E), \mathcal{Q} \triangleq (\mathcal{Q}_V, \mathcal{Q}_E), \mathcal{X}_t,$
$X_{flags} \triangleq (\mathcal{X}_{ncon}, \mathcal{X}_{new}, V_{exp}, V_{rewire}, c_{sol})$
Out: $T, \mathcal{Q}, X_{flags}$

1 $(v_b, x_b) \leftarrow PopBest(\mathcal{Q}_E)$;
2 $v_p \leftarrow Par(v_b)$;
3 **if** $g_T(v_b) + \hat{c}(v_p, v_b, x_b) + \hat{v}(x_b, v_b) \geq c_{sol}$ **then**
  `// Edge cannot improve solution`
4 $\mathcal{Q}_E \leftarrow \emptyset$; $\mathcal{Q}_V \leftarrow \emptyset$;
5 **goto** 32;
6 **if** $x_b \in X_{ncon}$ **then** `//` x_b `is unconnected`
7 **if** $g_T(v_b) + c(v_p, v_b, x_b) + \hat{v}(x_b, v_b) < c_{sol}$ **then**
   `// Add` x_b `to the tree`
8   $\mathcal{X}_{ncon} \xleftarrow{-} \{x_b\}$; $V \xleftarrow{+} \{x_b\}$; $E \xleftarrow{+} \{(v_b, x_b)\}$;
9   $\mathcal{Q}_V \xleftarrow{+} \{x_b\}$;
10   **if** $x_b \in \mathcal{X}_t$ **then** $c_{sol} \leftarrow \min_{v_t \in V \cap \mathcal{X}_t} g_T(v_t)$;
11 **else** `//` x_b `is part of the tree`
12 **if** $g_T(v_b) + \hat{c}(v_p, v_b, x_b) \geq g_T(x_b)$ **then goto** 32;
13 $c_{x_b} \leftarrow g_T(v_b) + c(v_p, v_b, x_b)$;
14 **if** $c_{x_b} + \hat{v}(x_b, v_b) \geq c_{sol} \vee c_{x_b} \geq g_T(x_b)$ **then**
15   **goto** 32;
16 $x_{bn} \leftarrow x_b$; $V \xleftarrow{+} \{x_{bn}\}$; $E \xleftarrow{+} \{(v_b, x_{bn})\}$;
17 **if** $x_b \in V_{exp}$ **then** $V_{exp} \xleftarrow{+} \{x_{bn}\}$;
18 **if** $x_b \in V_{rewire}$ **then** $V_{rewire} \xleftarrow{+} \{x_{bn}\}$;
19 $V_{exp} \xleftarrow{+} \{x_b\}$; $V_{rewire} \xleftarrow{+} \{x_b\}$;
20 $V_{cn} \leftarrow \{(x_{bn}, v_c), \forall v_c \in Children(x_b)\}$;
21 **while** $V_{cn} \neq \emptyset$ **do**
22   $V_{gcn} \leftarrow \emptyset$;
23   **foreach** $(v_{pn}, v_c) \in V_{cn}$ **do**
24   **if** $c(Par(v_{pn}), v_{pn}, v_c) = \infty$ **then** continue;
25    $v_{cn} \leftarrow v_c$; $V \xleftarrow{+} \{v_{cn}\}$; $E \xleftarrow{+} \{(v_{pn}, v_{cn})\}$;
26   **if** $v_c \in V_{exp}$ **then** $V_{exp} \xleftarrow{+} \{v_{cn}\}$;
27   **if** $v_c \in V_{rewire}$ **then** $V_{rewire} \xleftarrow{+} \{v_{cn}\}$;
28    $V_{exp} \xleftarrow{+} \{v_c\}$; $V_{rewire} \xleftarrow{+} \{v_c\}$;
29    $V_{gcn} \xleftarrow{+} \{(v_{cn}, v_{gc}), \forall v_{gc} \in Children(v_c)\}$;
30   $V_{cn} \leftarrow V_{gcn}$;
31 $c_{sol} \leftarrow \min_{v_t \in V \cap \mathcal{X}_t} g_T(v_t)$;
32 **return** $\{T, \mathcal{Q}, X_{flags}\}$;

If all checks pass, typically x_b would be rewired to have v_b as its parent in [17]. However, when this occurs, the covariance values of x_b and all its descendants change because the LinCov integration now follows a different set of waypoints. This change has the potential to invalidate the descendants of x_b, including descendants that contribute to the current best solution found. To ensure that the cost

of the best solution found never increases, the rewiring process used herein differs from that of previous FB-BIT* works[17]. In this work, x_b is duplicated into a new copy, x_{bn}, which is then added to the tree with v_b as its parent on line 16. The copy of x_b takes on the expanded and rewired state of x_b, and x_b is set expanded and rewired on lines 17 through 19 to avoid redundant calculations. Lines 20 through 30 recursively copies and repropagates the descendants of x_b using a similar process as was used for x_b. The only difference is that the descendants of x_b do not require the cost checks that x_b does. The condition on line 24 ensures that the copied edges satisfy the continuity constraints in (85) and the POD constraints in (10). Line 31 updates the current solution cost in case it changed while updating the tree.

Unlike $ExpVert$, the $ExpEdge$ procedure evaluates the edges using the true cost of the edge instead of heuristics and performs LinCov integration for the POD condition checking before adding the edge to the tree. This makes $ExpEdge$ a much slower operation than $ExpVert$, which is why $ExpEdge$ is only run with the best edges available.

7 Simulation

In this section, the proposed FB-BIT* algorithm is compared to two preexisting solution methods in the literature. The first of these is the probability of detection visibility graph (PDVG) planner described in [9]. The PDVG planner represents the environment as a set of polygons, each centered at the location of a radar station. A visibility graph is generated between the vertices of the polygons while considering the inside of each polygon opaque, i.e., connecting all vertices with lines and removing any line that goes through a radar polygon. A candidate solution is generated by finding the shortest path through the visibility graph, and the POD constraint is evaluated along the candidate solution. The radar polygons that are nearest to the locations along the path where the POD constraint is violated are expanded to push the UAV further away from the radar, thus lowering the POD. The process begins again with the new radar polygons until a valid solution is generated or the visibility graph becomes unsolvable. The PDVG planner is used for comparison because, of all the works in the literature, the problem constraints considered in [9] are most similar to the constraints considered herein. However, [9] assumes an open-loop UAV model, while this work considers the effects of closed-loop control; see Table 1.

The second planner FB-BIT* is compared to is an RRT-based planner described in [18]. In [18], a closed-loop UAV model is described and used to approximate the probability of colliding with obstacles while passing through GPS-denied regions. RRT is then used to generate paths with a low probability of collision with the obstacles present in the environment. The planner in [18] is used for comparison to FB-BIT* because [18] is the only probabilistic path planner in the literature that considers closed-loop dynamics as this work does; see Table 1.

Table 3. Vehicle related parameter.

Parameter	Value	Parameter	Value
$\mathfrak{a}$	$0.18\ m$	$\mathfrak{b}$	$0.17\ m$
$\mathfrak{c}$	$0.20\ m$	d_{nom}	$-3.5\ km$
V_{nom}	$80\ \frac{m}{s}$	d_{init}	$20\ km$

Table 4. Radar parameter values.

Parameter	Value	Parameter	Value
$\sqrt{R_{p_r^n}}$	$\frac{100}{3}\ I_3\ m$	P_{fa}	1×10^{-9}
$\sqrt{R_{c_r}}$	$\frac{1}{3}\ \frac{Jm^2}{^\circ K}$	c_r 2 types	20 or 50 $\frac{Jm^2}{^\circ K}$

When comparing the PDVG and RRT-based planners to FB-BIT*, the closed-loop UAV model described in Section 4 and the POD constraints described in Section 3 are substituted into the planners. This is done for a fair comparison and to evaluate the preexisting probabilistic path planning literature on the problem described herein. Additionally, this work uses the Fillet-Based RRT (FB-RRT) planner described in [34] instead of the standard RRT because the dynamic model used requires paths with continuity in heading. As described in [34], FB-RRT maintains all the properties of RRT while using fillets to connect states in its search tree, as is done in FB-BIT*.

The rest of this section proceeds as follows. Section 7.1 discusses the example scenarios used for comparison. Section 7.2 provides the averaged planning results and discusses how each planner performs.

7.1 Scenarios

In this section, some planning scenarios are described. These scenarios are designed to mimic situations that might arise in real life and test the planners compared herein in various ways. The planning scenarios are shown in Figure 5, with example solutions from each planner. Note that some example solutions are omitted because the planner was unable to find a valid solution to the problem. Additionally, there are two types of radar station models considered. One where the consolidated radar constant, c_r, is 20 $\frac{Jm^2}{^\circ K}$ which is called "Radar 20" and a higher power radar station model where $c_r = 50\ \frac{Jm^2}{^\circ K}$ that is called "Radar 50".

The Snake scenario, shown in the upper left of Figure 5, is the smallest of the scenarios considered herein. It has only two Radar 20 radar models that monitor a $600\ km^2$ region. In this scenario, no features are included since only $80\ km^2$ of the area is GPS-denied. This scenario is intended to be as easy to solve as possible. It is included in this work

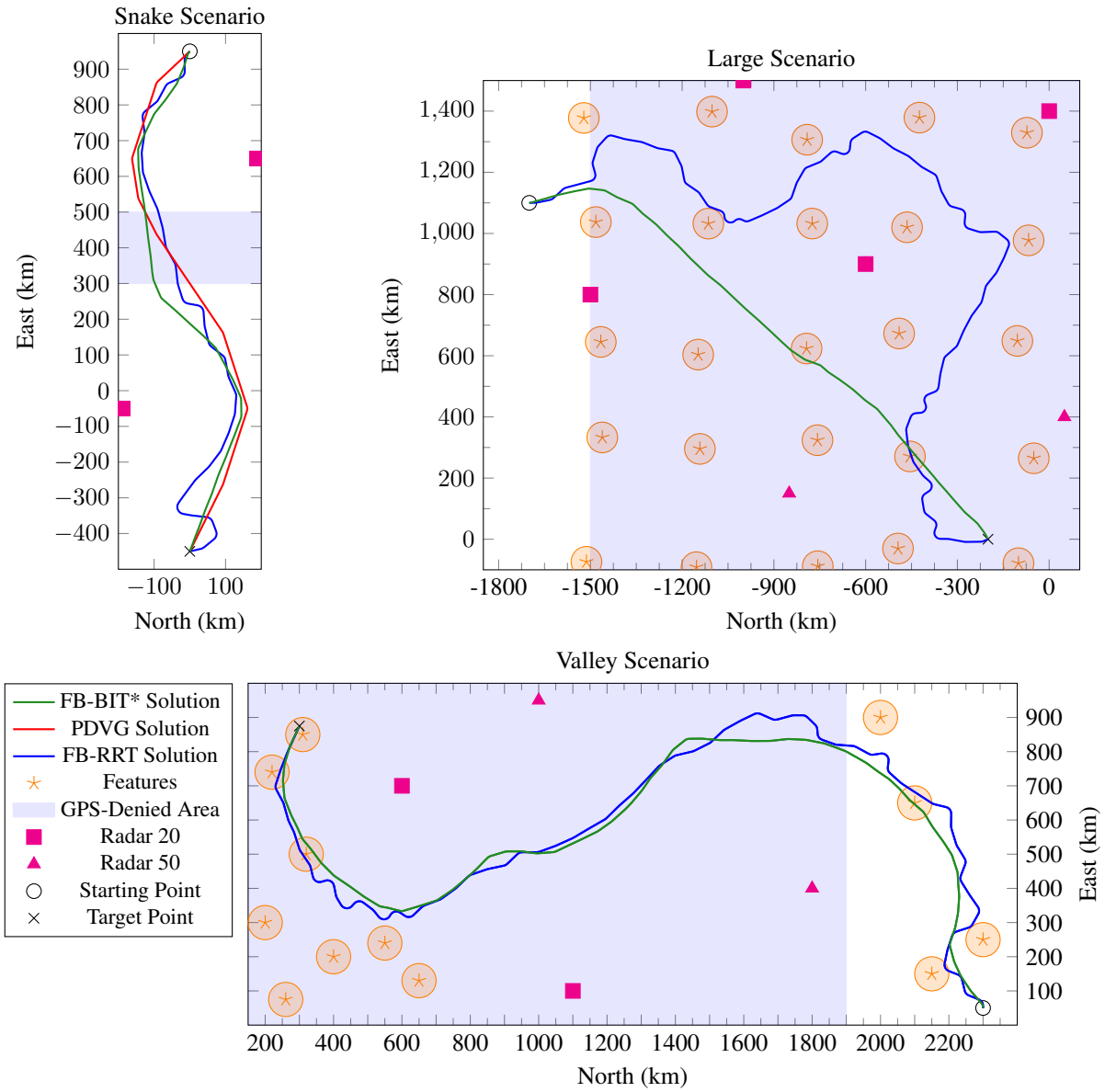


Figure 5. The planning scenarios with example FB-BIT*, PDVG, and FB-RRT solutions in green, red, and blue respectively.

because some of the planners compared herein, especially the PDVG planner, have a very difficult time solving most of the problems that the FB-BIT* planner can solve.

The Large scenario is shown in the upper right of Figure 5. It is the largest of the scenarios considered with an area of 3120 km^2. It also has the largest number of radars modeled and features present with 25 features, 4 Radar 20 models, and 2 Radar 50 models. The scenario includes two local minima. The example FB-BIT* path goes through the optimal local minima, whereas the FB-RRT example goes through the less optimal minima that loops around an extra Radar 20 model. When traversing the less optimal of these two minima, the POD constraints are easier to satisfy, while the true optimal path must pass through a narrow region of constraint validity between a Radar 20 and Radar 50 radar model. The presence of multiple local minima makes it difficult for planners that iteratively converge on an optimal solution, such as FB-BIT*, to find the global optimum because they can get stuck in local minima.

The Valley scenario is shown at the bottom of Figure 5. This scenario contains 12 features, 2 Radar 20 models, and 2 Radar 50 models in its 2250 km^2 area. It is meant to mimic the situation where a UAV must cross a valley filled with water where map features do not exist. While

Table 5. Sensor related parameter values.

Parameter	Value	Parameter	Value
$\sqrt{q_g}$	$0.015\ \frac{\frac{deg.}{s}}{\sqrt{Hz.}}$	$\sqrt{q_a}$	$250\ \frac{\mu g}{\sqrt{Hz.}}$
$\sqrt{R_{ap}}$	$0.01\ kPa$	Δt_{ap}	$0.05\ s$
$\sqrt{R_{dp}}$	$0.002\ kPa$	Δt_{dp}	$0.05\ s$
$\sqrt{R_{\psi}}$	$0.3\ deg.$	Δt_{ψ}	$0.125\ s$
$\sqrt{R_{los}}$	$65\ arcsec$	Δt_{los}	$10\ s$
$\sqrt{R_{fr}}$	$0.1\ m$	Δt_{fr}	$10\ s$
$\sqrt{R_{gps,\psi}}$	$2.5 \times 10^{-4}\ rad.$	Δt_{gps}	$0.2\ s$
$\sqrt{q_n}$	$1 \times 10^{-4}\ \frac{\frac{m}{s}}{\sqrt{Hz.}}$	$\sqrt{R_{gps,n}}$	$0.05\ m$
$\sqrt{q_e}$	$1 \times 10^{-4}\ \frac{\frac{m}{s}}{\sqrt{Hz.}}$	$\sqrt{R_{gps,e}}$	$0.05\ m$
$\sqrt{q_d}$	$1 \times 10^{-6}\ \frac{\frac{m}{s}}{\sqrt{Hz.}}$	$\sqrt{R_{gps,d}}$	$0.15\ m$

Table 6. Controller related parameter values.

Parameter	Value	Parameter	Value
k_{at}^p	$5 \times 10^{-4}\ \frac{1}{s^2}$	r	$20\ km$
k_{at}^d	$1 \times 10^{-3}\ \frac{1}{s}$	Δt	$1\ s$
k_{ct}	$1 \times 10^{-4}\ \frac{1}{m}$	ψ_{∞}	$0.05\ rad.$
k_d^p	$1 \times 10^{-6}\ \frac{\frac{rad.}{s}}{m}$	k_{ψ}^p	$0.5\ \frac{1}{s}$
k_d^d	$1 \times 10^{-4}\ \frac{rad.}{m}$	k_{ψ}^d	1.8

Table 7. General constant values.

Parameter	Value	Parameter	Value
κ	$1.38 \times 10^{-23}\ \frac{J}{^{\circ}K}$	g	$9.81\ \frac{m}{s^2}$
ρ	$1.26\ \frac{kg}{m^3}$	P_D^{max}	0.1

traversing the GPS-denied and feature-free region in the middle of the scenario, the state uncertainties of the UAV grow large, making it difficult to respect the POD constraints. The planner must traverse the long stretch of position uncertainty and find a feature on the right side of the valley to shrink the position uncertainty before reaching the target. As will be shown in the sequel, this is the scenario for which the planners considered herein have the hardest time finding a valid solution.

The various sensor and radar related constants are given in Tables 3 through 5. These values are from papers [9, 39,41], where similar problems are being evaluated. The controller related constants have been tuned by hand and are given in Table 6. Finally, the general constants of the problem are given in Table 7. Note that the maximum allowed POD is chosen to be $P_D^{max} \equiv 0.1$, which means that 99.7% of the realizations will maintain a POD of 0.1 or less.

FB-BIT* has three hyperparameters, i.e., parameters that are tuned to optimize algorithm performance. These parameters are the neighborhood radius, l, the number of new samples drawn per batch, n, and the smart and informed beacon bias, b_b. The hyperparameters have been tuned using a Bayesian optimization algorithm provided by the Ax library[45]. Each scenario was run with a maximal solve time of 15 minutes while tuning the hyperparameters. The objective was to minimize the sum of the objective value of the best solution that FB-BIT* produced across all three scenarios. The Ax was given over 300 trials to tune with. The result of the hyperparameter tuning is given in Table 8.

Table 8. The hyperparameter values from optimization.

Parameter	Value
l	$55.154\ km$
n	16893
b_b	2

The most notable result is that the batch size is set to a large value. This suggests that FB-BIT* produced its best results when the planner used only a few batches, keeping the number of rewires and subsequent tree duplications to a minimum. It also suggests that FMT* would be suitable for this problem, since FB-BIT* with a large batch size performed similarly to FMT*. The hyperparameter results also show that the beacon bias is optimally set to 2, which means that every other sample drawn from the configuration space is drawn from the smart and informed ellipses. This shows the advantage of using a greedy sampling heuristic, such as smart and informed sampling, when producing high-quality solutions quickly. See [34] for more details and a detailed comparison of different sampling strategies.

7.2 Results

This section describes the benchmarking results generated for the three planners considered in each environment. Each planner was given 30 minutes to find the best solution path it could. One hundred solving attempts were performed on each planner with each scenario. The results are averaged over those 100 runs. All results are gathered on an AMD Ryzen™ Threadripper™ 3970X processor. The planning code can be found in our open-source repository `https://github.com/james-swedeen/advanced_path_planning`.

Although each planner was given 100 attempts at each scenario, some did not find a valid solution consistently. Table 9 shows the percentage of trials that found a valid plan in 30 minutes. The PDVG planner was only able to find a solution for the Snake scenario. The other two scenarios proved too complicated for the simplistic planner to find a constraint-satisfying solution.

When comparing the two iterative algorithms, FB-BIT* and FB-RRT, it is seen that FB-BIT* found a valid solution 100% of the time in all scenarios. However, the

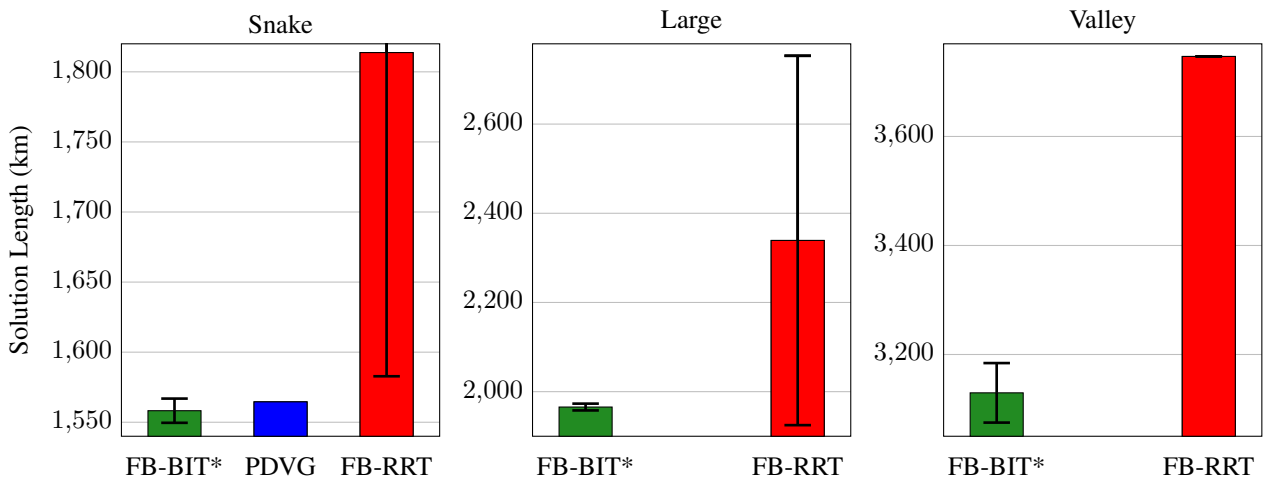


Figure 6. Planning results averaged over 100 runs in km after 30 minutes of run time. Bars show the average and error margins show the plus/minus 3-SD region of the best solution found.

Table 9. Percent of planning attempts that found a valid plan after 30 minutes of run time.

Planner	FB-BIT*	FB-RRT	PDVG
Snake	100%	17%	100%
Large	100%	3%	0%
Valley	100%	1%	0%

Table 10. Mean and SD solution length in km after 30 minutes of run time.

Planner	FB-BIT*		FB-RRT		PDVG
	Mean	SD	Mean	SD	Mean
Snake	1558	2.9	1814	77	1566
Large	1966	2.5	2339	138	∞
Valley	3130	18	3747	0	∞

FB-RRT planner found a solution 17% of the time for the Snake scenario, the easiest scenario, and only found a valid solution once for the Valley scenario. Initially, it might seem odd that two planners that operate on similar principles would have such drastically different success rates. However, these results are a testament to the advantage of the sample batching mechanism that FB-BIT* employs. Using it, FB-BIT* avoids a staggering amount of repeated work that planners based on FB-RRT and other RRT and RRT*-based planners must perform. This repeated work occurs most frequently in environments with many obstacles/constraints to navigate. In RRT-based planners, one sample is drawn anywhere in the environment and then projected closer to the preexisting search tree before a connection attempt is made. If the search tree is already up against an obstacle, then the projected point is likely to be within the obstacle, and thus any edge connecting the point to the tree will violate obstacle constraints. In applications like the one considered herein, the time it takes to generate the edge from the search tree to the point and to perform obstacle checking is significant. Every time a potential edge is generated, then discarded because of problem constraints, is wasted time and computing power.

BIT*-based planners avoid this wasted time by using batched sampling and careful maintenance of the vertex and edge queues. They guarantee that an edge is never made between two samples of the search space more than once, while simultaneously exhaustively checking every combination of samples that can contribute to an optimal solution. Additionally, because of the batching of samples, they never need to project samples to the current search tree. Using the example from before, where the search tree is up against an obstacle, this means that the probability of an edge being generated into an obstacle is proportional to the neighborhood radius, instead of the volume of the entire unexplored search space, as is the case with RRT-based algorithms. This dramatically reduces the number of failed edge generation attempts that are made, which is why FB-BIT* performs far more reliably.

The resulting cost, or solution path length, that each planner produces is shown in Table 10 and Figure 6. As can be seen, FB-RRT consistently produces the longest paths. Furthermore, SDs of the results are the largest for FB-RRT. When comparing PDVG to FB-BIT* in the Snake scenario, it can be seen that on average FB-BIT* produces a solution shorter than PDVG. The PDVG result is within the three-SD region of FB-BIT*, but as can be seen in Figure 6, PDVG is in the upper end of what FB-BIT* produces. Additionally, considering PDVG is unable to produce any results for the other two scenarios, it is much less reliable. FB-BIT*

has both the lowest averaged results and the smallest SDs among the planners considered. This shows that the FB-BIT* planner is the most reliable and reliably produces higher quality solutions than the other planners.

8 Conclusion

In this work, the cutting-edge path planning algorithm FB-BIT* is modified for use in the mission planning of a fixed-wing UAV through a region monitored by adversarial ground-based radar stations. Additionally, an error in the original presentation of FB-BIT* is corrected. A detailed POD model is described and uncertainties in the POD are modeled. Probabilistic constraints are derived and evaluated using a closed-loop simulation of a UAV with an inertial EKF providing navigation state estimates. The state covariance calculation is accelerated through LinCov analysis, which is validated through extensive Monte Carlo simulations. FB-BIT* is compared to two existing planners in the literature within several planning scenarios that include GPS-denied regions. It is shown that FB-BIT* outperforms the existing methods in both reliability and optimality. The directions of future work include the definition of an overall mission evaluation of the POD and applying the proposed FB-BIT* planner to other problem domains, e.g., planning UAVs paths around tall buildings where GPS is unreliable.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the Air Force Research Laboratory, Wright-Patterson Air Force Base, OH. [grant number 203463].

References

1. Peschel JM and Murphy RR. On the human–machine interaction of unmanned aerial system mission specialists. *IEEE Transactions on Human-Machine Systems* 2012; 43(1): 53–62.
2. Ceccarelli N, Enright JJ, Frazzoli E et al. Micro uav path planning for reconnaissance in wind. In *2007 American Control Conference*. IEEE, pp. 5310–5315.
3. Larson R, Pachter M and Mears M. Path planning by unmanned air vehicles for engaging an integrated radar network. In *AIAA guidance, navigation, and control conference and exhibit*. p. 6191.
4. Xiao-wei F, Zhong L and Xiao-guang G. Path planning for uav in radar network area. In *2010 Second WRI Global Congress on Intelligent Systems*, volume 3. IEEE, pp. 260–263.
5. Kabamba PT, Meerkov SM and Zeitz III FH. Optimal path planning for unmanned combat aerial vehicles to defeat radar tracking. *Journal of Guidance, Control, and Dynamics* 2006; 29(2): 279–288.
6. Erlandsson T and Niklasson L. Automatic evaluation of air mission routes with respect to combat survival. *Information Fusion* 2014; 20: 88–98.
7. Bortoff SA. Path planning for uavs. In *Proceedings of the 2000 american control conference. ACC (IEEE Cat. No. 00CH36334)*, volume 1. IEEE, pp. 364–368.
8. McFarland MB, Zachery RA and Taylor BK. Motion planning for reduced observability of autonomous aerial vehicles. In *Proceedings of the 1999 IEEE International Conference on Control Applications (Cat. No. 99CH36328)*, volume 1. IEEE, pp. 231–235.
9. Costley A, Swedeen J, Droge G et al. Path planning with uncertainty for aircraft under threat of detection from ground-based radar. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology* 2025; In Press.
10. Pachter M and Hebert J. Optimal aircraft trajectories for radar exposure minimization. In *Proceedings of the 2001 American Control Conference.(Cat. No. 01CH37148)*, volume 3. IEEE, pp. 2365–2369.
11. Moore FW. Radar cross-section reduction via route planning and intelligent control. *IEEE Transactions on Control Systems Technology* 2002; 10(5): 696–700.
12. Jun M and D'Andrea R. Path planning for unmanned aerial vehicles in uncertain and adversarial environments. *Cooperative control: models, applications and algorithms* 2003; : 95–110.
13. Sun X, Cai C, Yang J et al. Route evaluation for unmanned aerial vehicle based on type-2 fuzzy sets. *Engineering Applications of Artificial Intelligence* 2015; 39: 132–145.
14. Erlandsson T. Route planning for air missions in hostile environments. *The Journal of Defense Modeling and Simulation* 2015; 12(3): 289–303.
15. Costley A, Christensen R, Leishman RC et al. Sensitivity of single-pulse radar detection to aircraft pose uncertainties. *IEEE Transactions on Aerospace and Electronic Systems* 2022; : 1–10DOI:10.1109/TAES.2022.3213793.
16. Costley A, Christensen R, Leishman RC et al. Sensitivity of the probability of radar detection to radar state uncertainty. *IEEE Transactions on Aerospace and Electronic Systems* 2023; : 1–20DOI:10.1109/TAES.2023.3266177.
17. Swedeen J and Droge G. Fillet-based batch informed trees (fb-bit*): Rapid convergence path planning for curvature constrained vehicles. In *2023 Seventh IEEE International Conference on Robotic Computing (IRC)*. IEEE, pp. 71–78.
18. Christensen RS, Droge G and Leishman RC. Closed-loop linear covariance framework for path planning in static uncertain obstacle fields. *Journal of Guidance, Control, and Dynamics* 2022; 45(4): 669–683. DOI:10.2514/1.G006228. URL https://doi.org/10.2514/1.G006228.

19. Hakobyan A, Kim GC and Yang I. Risk-aware motion planning and control using cvar-constrained optimization. *IEEE Robotics and Automation Letters* 2019; 4(4): 3924–3931. DOI:10.1109/LRA.2019.2929980.
20. Jiang C, Hu Z, Mourelatos ZP et al. R2-rrt*: Reliability-based robust mission planning of off-road autonomous ground vehicle under uncertain terrain environment. *IEEE Transactions on Automation Science and Engineering* 2022; 19(2): 1030–1046. DOI:10.1109/TASE.2021.3050762.
21. Silver D and Veness J. Monte-carlo planning in large pomdps. *Advances in neural information processing systems* 2010; 23.
22. Elsherbeni BRMAZ. Simulations for radar system design, 2003.
23. Mahafza BR and Elsherbeni A. *MATLAB Simulations for Radar Systems Design*. CRC Press, 2003.
24. Karaman S and Frazzoli E. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research* 2011; 30(7): 846–894. DOI:10.1177/0278364911406761. URL https://doi.org/10.1177/0278364911406761.
25. LaValle SM. *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. Available at http://planning.cs.uiuc.edu/.
26. Gammell JD, Srinivasa SS and Barfoot TD. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 2997–3004. DOI:10.1109/IROS.2014.6942976.
27. Nasir J, Islam F, Malik U et al. Rrt*-smart: A rapid convergence implementation of rrt*. *International Journal of Advanced Robotic Systems* 2013; 10. DOI:10.5772/56718.
28. Jeong IB, Lee SJ and Kim JH. Quick-rrt*: Triangular inequality-based implementation of rrt* with improved initial solution and convergence rate. *Expert Systems with Applications* 2019; 123: 82–90. DOI: https://doi.org/10.1016/j.eswa.2019.01.032. URL https://www.sciencedirect.com/science/article/pii/S0957417419300326.
29. Janson L, Schmerling E, Clark A et al. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International journal of robotics research* 2015; 34(7): 883–921.
30. Gammell JD, Barfoot TD and Srinivasa SS. Batch informed trees (bit*): Informed asymptotically optimal anytime search. *The International Journal of Robotics Research* 2020; 39(5): 543–567. DOI:10.1177/0278364919890396. URL https://doi.org/10.1177/0278364919890396.
31. Moon C and Ching W. Kinodynamic planner dual-tree rrt (dt-rrt) for two-wheeled mobile robots using the rapidly exploring random tree. *IEEE Transactions on Industrial Electronics* 2015; 62: 1080–1090. DOI:10.1109/TIE.2014.2345351.
32. Sedighi S, Nguyen DV and Kuhnert KD. Guided hybrid a-star path planning algorithm for valet parking applications. In *2019 5th International Conference on Control, Automation and Robotics (ICCAR)*. pp. 570–575. DOI:10.1109/ICCAR.2019.8813752.
33. Li Y, Littlefield Z and Bekris KE. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research* 2016; 35(5): 528–564. DOI:10.1177/0278364915614386. URL https://doi.org/10.1177/0278364915614386.
34. Swedeen J, Droge G and Christensen R. Fillet-based RRT*: A rapid convergence implementation of RRT* for curvature constrained vehicles. *Journal of Intelligent & Robotic Systems* 2023; 108(4): 68. DOI:https://doi.org/10.1007/s10846-023-01846-x.
35. Webb DJ and van den Berg J. Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In *2013 IEEE International Conference on Robotics and Automation*. pp. 5054–5061. DOI:10.1109/ICRA.2013.6631299.
36. Cui P, Yan W and Guo X. Path planning for robot with pose constraints using dubins-rrt. In *2018 3rd International Conference on Advanced Robotics and Mechatronics (ICARM)*. pp. 560–565. DOI:10.1109/ICARM.2018.8610812.
37. Lan X and Di Cairano S. Continuous curvature path planning for semi-autonomous vehicle maneuvers using rrt. In *2015 European Control Conference (ECC)*. pp. 2360–2365. DOI:10.1109/ECC.2015.7330891.
38. Yang K, Moon S, Yoo S et al. Spline-based rrt path planner for non-holonomic robots. *Journal of Intelligent & Robotic Systems* 2014a; 73(1-4): 763–782.
39. Beard R and McLain T. *Small Unmanned Aircraft: Theory and Practice*. Princeton University Press, 2012. ISBN 9781400840601. URL https://books.google.com/books?id=YqQtjhPUaNEC.
40. Nelson DR, Barber DB, McLain TW et al. Vector field path following for miniature air vehicles. *IEEE Transactions on Robotics* 2007; 23(3): 519–529. DOI:10.1109/TRO.2007.898976.
41. Christensen R, Geller D and Hansen M. Linear covariance navigation analysis of range and image measurement processing for autonomous lunar lander missions. In *2020 IEEE/ION Position, Location and Navigation Symposium (PLANS)*. IEEE, pp. 1524–1535.
42. Geller DK. Linear covariance techniques for orbital rendezvous analysis and autonomous onboard mission planning. *Journal of Guidance, Control, and Dynamics* 2006; 29(6): 1404–1414.
43. Maybeck PS. *Stochastic models, estimation, and control*. Academic press, 1982.
44. Christensen RS and Geller D. Linear covariance techniques for closed-loop guidance navigation and control system design and analysis. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace*

Engineering 2014; 228(1): 44–65.

45. Baird SG, Liu M and Sparks TD. High-dimensional bayesian optimization of 23 hyperparameters over 100 iterations for an attention-based network to predict materials property: A case study on crabnet using ax platform and saasbo. *Computational Materials Science* 2022; 211: 111505. DOI:https://doi.org/10.1016/j.commatsci.2022.111505. URL https://www.sciencedirect.com/science/article/pii/S0927025622002610.
46. Zanetti R. Rotations, transformations, left quaternions, right quaternions? *The Journal of the Astronautical Sciences* 2019; 66(3): 361–381. DOI:10.1007/s40295-018-00151-2. URL https://doi.org/10.1007/s40295-018-00151-2.
47. Crassidis JL and Junkins JL. *Optimal Estimation of Dynamic Systems, Second Edition (Chapman & Hall/CRC Applied Mathematics & Nonlinear Science)*. 2nd ed. Chapman & Hall/CRC, 2011. ISBN 1439839859.

A Acronyms and Nomenclature

Acronyms

BIT*	Batch Informed Trees
EKF	extended Kalman filter
FB-BIT*	Fillet-Based Batch Informed Trees
FB-RRT	Fillet-Based RRT
FB-RRT*	Fillet-Based RRT*
FMT*	Fast Marching Tree
GPS	global positioning system
IMU	inertial measurement unit
LinCov	linear covariance
LOS	line of sight
NED	north east down
PDVG	probability of detection visibility graph
POD	probability of detection
PSD	power spectral density
RCS	radar cross section
RRT	Rapidly-exploring Random Tree
RRT*	Optimal RRT
SD	standard deviation
UAV	unmanned aerial vehicle

Nomenclature

Number Sets

$\mathbb{Z}_+$	Set of all positive integers
$\mathbb{H}$	Set of all quaternions
$\mathbb{R}_{[0,1]}$	Set of all real numbers between 0 and 1
$\mathbb{SO}$	Set of all special orthogonal matrices
$\mathbb{R}$	Set of all real numbers
$\mathbb{R}_+$	Set of all positive real numbers
$\mathcal{S}$	Set of all symmetric matrices

Operators and Helper Functions

$\arctan$	Inverse of tangent trig function
erfc	Complementary error function
$\hat{\ominus}$	Element wise subtraction of navigation vectors
$(\cdot)^*$	Quaternion conjugate
$[\cdot]_\times$	Mapping from a vector to cross product matrix
$\ln$	Natural logarithm
$\text{s}(\cdot), \text{c}(\cdot)$	Sin and cos trig functions
$\ominus$	Element wise subtraction of truth states
$\otimes$	Hamiltonian quaternion product
Θ_q	Mapping from quaternion to Euler angles
$\times$	Cross-product
$\overleftarrow{+}, \overleftarrow{-}$	Set addition and subtraction, respectively
$E[\cdot]$	Expected value
q_Θ	Mapping from Euler angles to a quaternion
R_q	Mapping from a quaternion to a rotation matrix
R_Θ	Mapping from Euler angles to a rotation matrix
T_Θ	Maps Euler angles to an angular rotation matrix
${}^nX$	The nth column of X
δ	Dirac delta function

Constants

$0_{n\times m}$	n by m matrix of zeros
κ	Boltzmann's constant

ρ	Air density
g^n	Gravitational acceleration vector
I_n	n by n identity matrix
g	Magnitude of acceleration due to gravity

Probability of Detection

$\mathfrak{a}, \mathfrak{b}, \mathfrak{c}$	Length of RCS ellipse axis of the aircraft
σ_{pd}	Standard deviation of probability of detection
ϱ	Radar cross section
ξ	The RCS azimuth angle
ζ	The RCS elevation angle
c_r	Consolidated radar constant
P_D	Probability of detection
p_d	Probability of detection function
P_D^{max}	Max POD allowed
p_r^n	Position of radar in NED frame
P_{fa}	Probability of false alarm
R_{c_r}	Variance of c_r
$R_{p_r^n}$	Covariance of p_r^n
R_{x_a}	Covariance of x_a
S	Signal-to-noise ratio of detection signal
x_a	Pose vector of aircraft

Aircraft Model Functions

a	Control function
a_{fb}	Feedback part of control function
a_{ff}	Feedforward part of control function
f	State dynamics

Aircraft Model Variables

$\bar{x}_n$	Next state in reference trajectory
$\delta\dot{e}^r$	Time derivative of δe^r
δe^r	Difference in navigation and reference points
Δt	Period between controller evaluations
ι	Distinguishes between left and right turns
Ω_x	State dependent component of angular velocity
ψ_∞	Max difference in desired and reference course
ψ_d	Desired course angle
ψ_{s0}	Initial heading of a reference path segment
Θ_b^n	Euler angles from aircraft body to NED frames
d_{nom}	Nominal aircraft down
k_ψ^p, k_ψ^d	Heading controller gains
k_d^p, k_d^d	Down controller gains
k_{at}^p, k_{at}^d	Along-track controller gains
k_{ct}	Cross-track controller path gain
p_{s0}^n	Initial position of a reference path segment
q_b^n	Quaternion from body to NED frames
r	Arc radius of the reference trajectory
s	Length along a reference path segment
V_{nom}	Nominal aircraft ground speed
β, α, A_a	Commanded roll, pitch, and airspeed rates
ϕ, θ, ψ	Roll, pitch, and yaw of aircraft
p^n	Position of aircraft in NED frame
u	Control vector
V_a	Airspeed of aircraft
x	Truth state vector

Sensor Functions

$\hat{h}_j$	Measurement estimate function of type j
h_j	True measurement function of type j
i	Inertial measurement function
m	Mapping from navigation to truth state vectors

Sensor Variables

Δt_j	The period between measurements of type j
δx_e	Error state vector
η	Additive noise on inertial measurements
$\hat{\tilde{z}}_j$	Estimated noninertial measurement of type j
$\hat{K}_j$	Kalman gain for measurement type j
ν_j	Noise on noninertial measurement of type j
ω^b	Angular rate of the aircraft in body frame

$\tilde{y}$	Inertial measurement vector
$\tilde{z}_j$	True noninertial measurement of type j
a^b	Specific force of the aircraft in the body frame
n_j	Number of elements in measurement of type j
P	Error state covariance
p_f^n	Position of a map feature in the NED frame
Q_η	PSD of η
q_g, q_a	PSD of the gyroscope and accelerometer noise
R_j	Covariance of ν_j
q_n, q_e, q_d	PSDs of w_n, w_e, and w_d
Q_w	PSD of w
w	Process noise vector
w_n, w_e, w_d	North, East, and Down process noise

Pose Variance Generation

$\delta\hat{x}$	Navigation state dispersion off reference
δx	Truth state dispersion off reference
R_{true}	Truth state covariance
X	Augmented state vector

Mission Planning Functions

$\hat{c}$	Heuristic estimate of c
$\hat{g}$	Heuristic estimate of g_T
$\hat{v}$	Heuristic estimate of the cost-to-go
$BestVal$	Finds the lowest cost element of a queue
c	Cost of an edge
$Children$	Finds the children of a vertex
d	Fillet distance function
$ExpEdge$	Processes the next edge under consideration
$ExpVert$	Expands the next vertex under consideration
g_T	Cost-to-come from the root vertex
$Near$	Finds all elements of a set close to a point
Par	Finds the parent of a vertex
$PopBest$	Removes the lowest cost element of a queue
$Prune$	Tree pruning procedure
$Sample$	Generates random samples of $\mathcal{X}$
$Solution$	Finds the path that ends a given node

Mission Planning Variables

γ	Change in orientation between three vertices
$\mathcal{Q}_E$	Edges under consideration queue
$\mathcal{Q}_V$	Vertices under consideration queue
$\mathcal{X}$	Planning state space
$\mathcal{X}_t$	Target set
$\mathcal{X}_{ncon}$	Set of samples that are not connected to the tree
ψ_r	Initial path direction
b_b	Smart and informed beacon bias
c_{sol}	Cost of the best solution found so far
d_{init}	Root to pseudoroot distance
E	Planning tree edge set
l	Near neighbor search radius
n	Number of new samples drawn each batch
T	Planning search tree
V	Planning tree vertex set
V_{exp}	Expanded vertices
V_{rewire}	Vertices that have been considered for rewiring
x_r	Planning tree root vertex
X_{flags}	Information about planning progress
X_{new}	Set of samples that are new this batch
x_{sr}	Pseudoroot vertex

Variable Modifies

$\bar{m}$	Mapping from reference to truth state vectors
$\bar{x}$	Nominal or average of x
δx	Perturbation on x
$\hat{x}$	Estimated or navigational value of x
$\hat{x}^-, \hat{x}^+$	A priori and a posteriori estimates of x
$\tilde{x}$	Measured value of x
$\dot{x}$	Time derivative of x

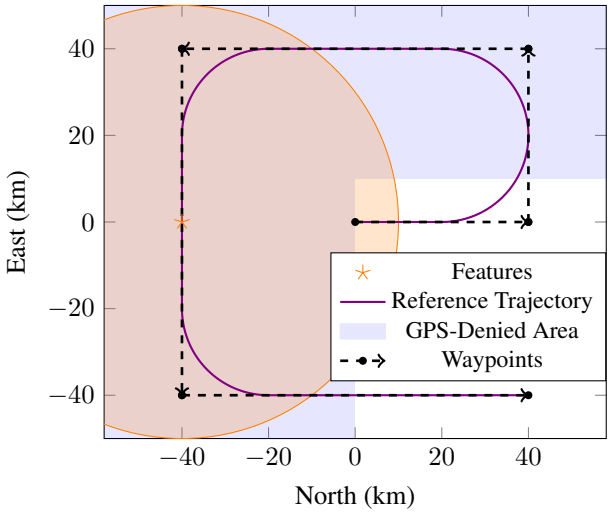


Figure 7. The Linearization Validation scenario.

B Linearization Validation

In this appendix, the linearization assumptions made in Sections 3.3 and 5.2 are validated through example. The scenario used for this task is shown in Figure 7. It is designed to be a scale model of the situations that arise while planning, containing a GPS-denied area as well as a feature along the reference trajectory. Monte Carlo results have been averaged over 1000 individual simulations.

Figure 8 shows the truth and estimated position dispersions as they are calculated by LinCov and via Monte Carlo analysis. Figure 9 shows the POD and the 3-SD bounds along the scenario path and the difference between the LinCov and Monte Carlo calculated average of the POD. As can be seen, there is a close similarity between the LinCov and Monte Carlo results. This shows that the system linearization is valid and LinCov produces accurate statistics.

C Linearization

This appendix provides the partial derivatives of the various functions that are needed to propagate the UAV pose variances. For the partial derivatives of the RCS model (4) and the POD metric (5) needed to evaluate (9), see [16].

Let the component of the truth angular velocity that is dependent on the aircraft states be denoted as

$$\Omega_x(x) \triangleq \begin{bmatrix} 0 & 0 & \frac{g}{V_a}\tan(\phi) \end{bmatrix}^T. \tag{103}$$

Then the partial derivative of the truth dynamics (13) is given by (104), where the partial derivative of the first column of the body to NED frame rotation matrix with respect to the Euler angle is given by (105) and the partial derivative of Ω_x with respect to Θ_b^n and V_a is given in (106) and (107), respectively. The partial derivatives of f with respect to the control and process noise vectors are given in (108) and (109), respectively.

$$ {}_xF = \begin{bmatrix} 0_{3\times3} & V_a \frac{\partial\, {}^1R_\Theta(\Theta_b^n)}{\partial \Theta_b^n} & {}^1R_\Theta(\Theta_b^n) \\ 0_{3\times3} & \frac{\partial \Omega_x(x)}{\partial \Theta_b^n} & \frac{\partial \Omega_x(x)}{\partial V_a} \\ 0_{1\times3} & 0_{1\times3} & 0 \end{bmatrix} \tag{104}$$

$$ \frac{\partial {}^1R_\Theta(\Theta_b^n)}{\partial \Theta_b^n} = \begin{bmatrix} 0 & -\mathrm{s}(\theta)\,\mathrm{c}(\psi) & -\mathrm{c}(\theta)\,\mathrm{s}(\psi) \\ 0 & -\mathrm{s}(\theta)\,\mathrm{s}(\psi) & \mathrm{c}(\theta)\,\mathrm{c}(\psi) \\ 0 & -\mathrm{c}(\theta) & 0 \end{bmatrix} \tag{105}$$

$$ \frac{\partial \Omega_x(x)}{\partial \Theta_b^n} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{g}{V_a}\sec^2(\phi) & 0 & 0 \end{bmatrix} \tag{106}$$

$$ \frac{\partial \Omega_x(x)}{\partial V_a} = \begin{bmatrix} 0 \\ 0 \\ -\frac{g}{V_a^2}\tan(\phi) \end{bmatrix} \tag{107}$$

$$ {}_uF \triangleq \frac{\partial f(x,u,w)}{\partial u} = \begin{bmatrix} & 0_{3\times3} & \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{108}$$

$$ {}_wF \triangleq \frac{\partial f(x,u,w)}{\partial w} = \begin{bmatrix} I_3 \\ 0_{3\times3} \\ 0_{1\times3} \end{bmatrix} \tag{109}$$

The partial derivative of the navigation state dynamics (16) with respect to the navigation state and the inertial measurement is given by (110) and (111), respectively.

$$ {}_{\hat{x}}\hat{F} = \begin{bmatrix} 0_{3\times3} & 0_{3\times3} & I_3 \\ 0_{3\times3} & -\left[\tilde{\omega}^b\right]_\times & 0_{3\times3} \\ 0_{3\times3} & -R_q(\hat{q}_b^n)\left[\tilde{a}^b\right]_\times & 0_{3\times3} \end{bmatrix} \tag{110}$$

$$ {}_{\tilde{y}}\hat{F} \triangleq \frac{\partial \hat{f}(\hat{x},\tilde{y})}{\partial \tilde{y}} = \begin{bmatrix} 0_{3\times3} & 0_{3\times3} \\ I_3 & 0_{3\times3} \\ 0_{3\times3} & R_q(\hat{q}_b^n) \end{bmatrix} \tag{111}$$

The partial derivative of the inertial measurement function (37) with respect to the truth state is given by

$$ {}_xI \triangleq \frac{\partial c(x,u,\eta)}{\partial x} = \begin{bmatrix} 0_{3\times3} & \frac{\partial \omega^b}{\partial \Theta_b^n} & \frac{\partial \omega^b}{\partial V_a} \\ 0_{3\times3} & \frac{\partial a^b}{\partial \Theta_b^n} & \frac{\partial a^b}{\partial V_a} \end{bmatrix}, \tag{112}$$

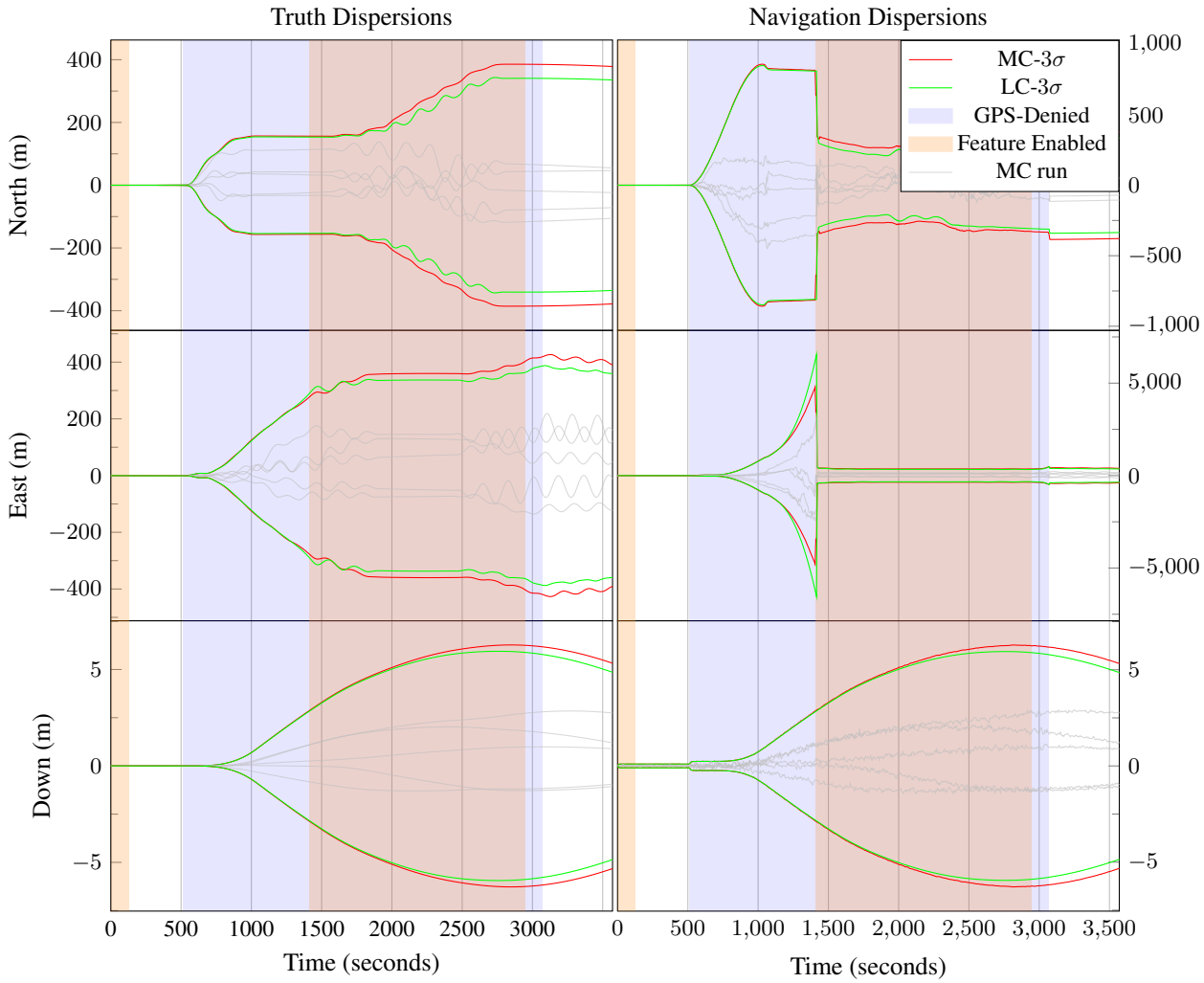


Figure 8. The truth and navigation position dispersions from the Linearization Validation scenario.

where

$$\frac{\partial \omega^b}{\partial \Theta_b^n} = {}_{\Theta_b^n} T_\Theta^{-1} \begin{bmatrix} \beta \\ \alpha \\ \frac{g\tan(\phi)}{V_a} \end{bmatrix} + T_\Theta^{-1}(\Theta_b^n)_{\Theta_b^n} \Omega_x(x), \quad (113)$$

$$\frac{\partial \omega^b}{\partial V_a} = T_\Theta^{-1}(\Theta_b^n) \frac{\partial \Omega_x(x)}{\partial V_a}, \quad (114)$$

$$\frac{\partial a^b}{\partial \Theta_b^n} = -\left[\left[\begin{bmatrix} V_a \\ 0 \\ 0 \end{bmatrix}\right]\right]_\times \frac{\partial \omega^b}{\partial \Theta_b^n} - g \frac{\partial^3 R_\Theta^T(\Theta_b^n)}{\partial \Theta_b^n}, \text{ and} \quad (115)$$

$$\frac{\partial a^b}{\partial V_a} = \left[\omega^b\right]_\times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} - \left[\left[\begin{bmatrix} V_a \\ 0 \\ 0 \end{bmatrix}\right]\right]_\times \frac{\partial \omega^b}{\partial V_a}. \quad (116)$$

The first term in $\frac{\partial \omega^b}{\partial \Theta_b^n}$ can be calculated by evaluating the matrix multiplication and taking the derivative of each column of $T_\Theta^{-1}(\Theta_b^n)$ separately as

$$\frac{\partial T_\Theta^{-1}(\Theta_b^n)}{\partial \Theta_b^n} \begin{bmatrix} \beta \\ \alpha \\ \frac{g}{V_a}\tan(\phi) \end{bmatrix} = \alpha \frac{\partial^2 T_\Theta^{-1}(\Theta_b^n)}{\partial \Theta_b^n} + \frac{g}{V_a}\tan(\phi) \frac{\partial^3 T_\Theta^{-1}(\Theta_b^n)}{\partial \Theta_b^n}, \quad (117)$$

where

$$\frac{\partial^2 T_\Theta^{-1}(\Theta_b^n)}{\partial \Theta_b^n} = \begin{bmatrix} 0 & 0 & 0 \\ -\sin(\phi) & 0 & 0 \\ -\cos(\phi) & 0 & 0 \end{bmatrix} \quad (118)$$

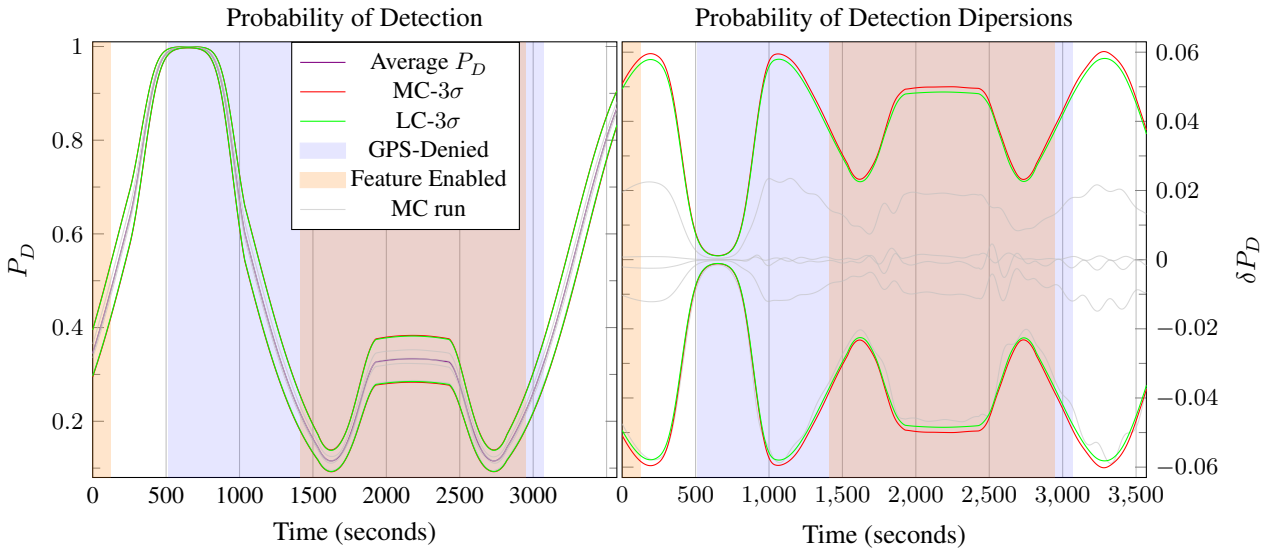


Figure 9. The probability of detection and probability of detection desperation off the reference trajectory over time from the Linearization Validation scenario.

and since ${}^3T_\Theta^{-1}(\Theta_b^n) \equiv {}^3R_\Theta^T(\Theta_b^n)$ we can say

$$\frac{\partial\, {}^3T_\Theta^{-1}(\Theta_b^n)}{\partial \Theta_b^n} = \frac{\partial\, {}^3R_\Theta^T(\Theta_b^n)}{\partial \Theta_b^n} = \begin{bmatrix} 0 & -\cos(\theta) & 0 \\ \cos(\phi)\cos(\theta) & -\sin(\phi)\sin(\theta) & 0 \\ -\sin(\phi)\cos(\theta) & -\cos(\phi)\sin(\theta) & 0 \end{bmatrix}. \tag{119}$$

The partial derivative of the inertial measurement function with respect to the control vector is given by

$${}_uI = \begin{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} \begin{bmatrix} V_a \\ 0 \\ 0 \end{bmatrix} \end{bmatrix}_\times \frac{\partial \omega^b}{\partial u} \end{bmatrix}^{\frac{\partial \omega^b}{\partial u}}, \tag{120}$$

where

$$\frac{\partial \omega^b}{\partial u} = T_\Theta^{-1}(\Theta_b^n) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{121}$$

The inertial measurement noise is additive, hence

$${}_\eta I \triangleq \frac{\partial c(x,u,\eta)}{\partial \eta} = I_6. \tag{122}$$

The partial derivative of the truth to navigation state mapping (22) with respect to the truth state is given by

$${}_xM = \begin{bmatrix} I_3 & 0_{3\times3} & 0_{3\times1} \\ 0_{3\times3} & I_3 & 0_{3\times1} \\ 0_{3\times3} & V_a \frac{\partial\, {}^1R_\Theta(\Theta_b^n)}{\partial \Theta_b^n} & {}^1R_\Theta(\Theta_b^n) \end{bmatrix}. \tag{123}$$

The partial derivative of the navigation to truth state mapping (24) with respect to the navigation state is

$${}_{\hat{x}}M^{-1} \triangleq \frac{\partial m^{-1}(\hat{x})}{\partial \hat{x}} = \begin{bmatrix} I_3 & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & I_3 & 0_{3\times3} \\ 0_{1\times3} & 0_{1\times3} & \frac{(\hat{v}^n)^T}{\|\hat{v}^n\|} \end{bmatrix}. \tag{124}$$

The partial derivative of the controller (27) with respect to the navigation state is given by

$${}_{\hat{x}}A = \begin{bmatrix} R_\Theta(\bar{\Theta}_b^n) \begin{bmatrix} 0 & 0 & k_{at}^p \\ \frac{-2k_\psi^p \psi_\infty k_{ct}}{\pi \cos^2(k_{ct}\, {}^2 \delta e^r)} & 0 & 0 \\ 0 & -k_d^p & 0 \end{bmatrix} \\ \begin{bmatrix} -k_\psi^d \frac{g \sec^2(\hat{\phi})}{\hat{V}_a} \\ 0 \\ -k_\psi^p \end{bmatrix} \quad 0_{3\times2} \\ \frac{k_\psi^d g \tan(\hat{\phi})\, {}^1R_q^T(\hat{q}_b^n)}{\hat{V}_a^2} \quad R_\Theta(\bar{\Theta}_b^n) \begin{bmatrix} 0 & k_{at}^d \\ 0 & 0 \\ -k_d^d & 0 \end{bmatrix} \end{bmatrix} \tag{125}$$

The measurement sensitivity matrices for each measurement will now be given. The measurement sensitivity matrices for the two GPS measurements are given by

$${}_{\hat{x}}\hat{H}_{gps,p} = \begin{bmatrix} I_3 & 0_{3\times3} & 0_{3\times3} \end{bmatrix}, \tag{126}$$

$${}_xH_{gps,p} = \begin{bmatrix} I_3 & 0_{3\times3} & 0 \end{bmatrix}, \tag{127}$$

$${}_{\hat{x}}\hat{H}_{gps,\psi} = \begin{bmatrix} 0_{1\times3} & 0 & 0 & 1 & 0_{1\times3} \end{bmatrix}, \text{ and} \tag{128}$$

$${}_xH_{gps,\psi} = \begin{bmatrix} 0_{1\times3} & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{129}$$

Sensitivity matrices of the LOS measurement are given in (130) through (135).

$$ {}_{\hat{x}}\hat{H}_{los} \triangleq \frac{\partial \hat{h}_{los}(\hat{x})}{\partial \hat{x}} = \frac{\partial \hat{h}_{los}}{\partial \hat{p}_f^b}\frac{\partial \hat{p}_f^b}{\partial \hat{x}} \tag{130} $$

$$ {}_{x}H_{los} \triangleq \frac{\partial h_{los}(x)}{\partial x} = \frac{\partial h_{los}}{\partial p_f^b}\frac{\partial p_f^b}{\partial x} \tag{131} $$

$$ \frac{\partial \hat{p}_f^b}{\partial \hat{x}} = \begin{bmatrix} -R_q^T(\hat{q}_b^n)\left[R_q^T(\hat{q}_b^n)(p_f^n - \hat{p}^n)\right]_\times & 0_{3\times 3}\end{bmatrix} \tag{132} $$

$$ \frac{\partial p_f^b}{\partial x} = \begin{bmatrix} -R_\Theta^T(\Theta_b^n)\left[R_\Theta^T(\Theta_b^n)(p_f^n - p^n)\right]_\times & 0_{3\times 1}\end{bmatrix} \tag{133} $$

$$ \frac{\partial \hat{h}_{los}}{\partial \hat{p}_f^b} = \begin{bmatrix} \frac{1}{\hat{p}_{f,z}^b} & 0 & -\frac{\hat{p}_{f,x}^b}{\left(\hat{p}_{f,z}^b\right)^2} \\ 0 & \frac{1}{\hat{p}_{f,z}^b} & -\frac{\hat{p}_{f,y}^b}{\left(\hat{p}_{f,z}^b\right)^2} \end{bmatrix} \tag{134} $$

$$ \frac{\partial h_{los}}{\partial p_f^b} = \begin{bmatrix} \frac{1}{p_{f,z}^b} & 0 & -\frac{p_{f,x}^b}{\left(p_{f,z}^b\right)^2} \\ 0 & \frac{1}{p_{f,z}^b} & -\frac{p_{f,y}^b}{\left(p_{f,z}^b\right)^2} \end{bmatrix} \tag{135} $$

The measurement sensitivity matrices for the feature range measurement are given as

$$ {}_{\hat{x}}\hat{H}_{fr} = \begin{bmatrix} \frac{\hat{p}^n - p_f^n}{\|p_f^n - \hat{p}^n\|} & 0_{1\times 3} & 0_{1\times 3}\end{bmatrix}, \text{ and} \tag{136} $$

$$ {}_{x}H_{fr} = \begin{bmatrix} \frac{p^n - p_f^n}{\|p_f^n - p^n\|} & 0_{1\times 3} & 0\end{bmatrix}. \tag{137} $$

Note that $h_{gps,\psi}(x) \equiv h_\psi(x)$ and $\hat{h}_{gps,\psi}(\hat{x}) \equiv \hat{h}_\psi(\hat{x})$ which means that

$$ {}_{\hat{x}}\hat{H}_{\psi} = {}_{\hat{x}}\hat{H}_{gps,\psi}, \quad \text{and} \quad {}_{x}H_{\psi} = {}_{x}H_{gps,\psi}. \tag{138} $$

The measurement sensitivity matrices for the differential and absolute pressure sensors are given by

$$ {}_{\hat{x}}\hat{H}_{dp} = \begin{bmatrix} 0_{1\times 3} & (\hat{v}^n)^T \frac{\partial\, {}^1R_\Theta(\Theta_b^n)}{\partial \Theta_b^n} & \frac{\rho}{2}{}^1R_q^T(\hat{q}_b^n)\end{bmatrix}, \tag{139} $$

$$ {}_{x}H_{dp} = \begin{bmatrix} 0_{1\times 3} & 0_{1\times 3} & \frac{\rho}{2}\end{bmatrix}, \tag{140} $$

$$ {}_{\hat{x}}\hat{H}_{ap} = \begin{bmatrix} 0 & 0 & -\rho g & 0_{1\times 3} & 0_{1\times 3}\end{bmatrix}, \text{ and} \tag{141} $$

$$ {}_{x}H_{ap} = \begin{bmatrix} 0 & 0 & -\rho g & 0_{1\times 3} & 0\end{bmatrix}. \tag{142} $$

D Attitude Related Functions

This appendix defines various functions that relate to the treatment of Euler angles and quaternions. The first set of functions relate to manipulating quaternions. Let the four elements of a generic quaternion, $q \in \mathbb{H}$, be denoted as

$$ q \triangleq \begin{bmatrix} q_w & q_x & q_y & q_z\end{bmatrix}^T. \tag{143} $$

The Hamiltonian quaternion product between two generic quaternions, ${}_1q, {}_2q \in \mathbb{H}$ is defined as[46]

$$ {}_1q \otimes {}_2q \triangleq \begin{bmatrix} {}_1q_w\, {}_2q_w - {}_1q_{xz}{}_2q_{xz}^T \\ {}_1q_w\, {}_2q_{xz} + {}_2q_w\, {}_1q_{xz} + {}_1q_{xz} \times {}_2q_{xz}\end{bmatrix}, \tag{144} $$

where q_{xz} is the last three elements of q. The difference function between two navigation state vectors is defined as

$$ \begin{bmatrix} {}_1\hat{p}^n \\ {}_1\hat{q}_b^n \\ {}_1\hat{v}^n \end{bmatrix} \hat{\ominus} \begin{bmatrix} {}_2\hat{p}^n \\ {}_2\hat{q}_b^n \\ {}_2\hat{v}^n \end{bmatrix} \triangleq \begin{bmatrix} {}_1\hat{p}^n - {}_2\hat{p}^n \\ 2\begin{bmatrix}0_{3\times 1} & I_3\end{bmatrix}\left(({}_2\hat{q}_b^n)^* \otimes {}_1\hat{q}_b^n\right) \\ {}_1\hat{v}^n - {}_2\hat{v}^n \end{bmatrix}, \tag{145} $$

where $(\cdot)^*$ is the quaternion conjugate, i.e.,

$$ \left(\begin{bmatrix} q_w & q_x & q_y & q_z\end{bmatrix}^T\right)^* \triangleq \begin{bmatrix} q_w & -q_x & -q_y & -q_z\end{bmatrix}^T. \tag{146} $$

This definition of $\hat{\ominus}$ takes the circular nature of attitude representations into account when defining the difference between two navigation state vectors. Similar considerations have to be made when defining the difference between two truth state vectors, which is why

$$ \begin{bmatrix} {}_1p^n \\ {}_1\phi \\ {}_1\theta \\ {}_1\psi \\ {}_1V_a \end{bmatrix} \ominus \begin{bmatrix} {}_2p^n \\ {}_2\phi \\ {}_2\theta \\ {}_2\psi \\ {}_2V_a \end{bmatrix} \triangleq \begin{bmatrix} {}_1p^n - {}_2p^n \\ \operatorname{atan2}(\mathrm{s}({}_1\phi - {}_2\phi), \mathrm{c}({}_1\phi - {}_2\phi)) \\ \operatorname{atan2}(\mathrm{s}({}_1\theta - {}_2\theta), \mathrm{c}({}_1\theta - {}_2\theta)) \\ \operatorname{atan2}(\mathrm{s}({}_1\psi - {}_2\psi), \mathrm{c}({}_1\psi - {}_2\psi)) \\ {}_1V_a - {}_2V_a \end{bmatrix}. \tag{147} $$

The conversion from a quaternion representation of attitude to an Euler angle representation and vice versa is needed. When going from Euler angles to quaternions

$$ q_\Theta(\phi, \theta, \psi) \triangleq \begin{bmatrix} \mathrm{c}\frac{\phi}{2}\mathrm{c}\frac{\theta}{2}\mathrm{c}\frac{\psi}{2} + \mathrm{s}\frac{\phi}{2}\mathrm{s}\frac{\theta}{2}\mathrm{s}\frac{\psi}{2} \\ \mathrm{s}\frac{\phi}{2}\mathrm{c}\frac{\theta}{2}\mathrm{c}\frac{\psi}{2} - \mathrm{c}\frac{\phi}{2}\mathrm{s}\frac{\theta}{2}\mathrm{s}\frac{\psi}{2} \\ \mathrm{c}\frac{\phi}{2}\mathrm{s}\frac{\theta}{2}\mathrm{c}\frac{\psi}{2} + \mathrm{s}\frac{\phi}{2}\mathrm{c}\frac{\theta}{2}\mathrm{s}\frac{\psi}{2} \\ \mathrm{c}\frac{\phi}{2}\mathrm{c}\frac{\theta}{2}\mathrm{s}\frac{\psi}{2} - \mathrm{s}\frac{\phi}{2}\mathrm{s}\frac{\theta}{2}\mathrm{c}\frac{\psi}{2} \end{bmatrix} \tag{148} $$

is used and when going from quaternions to Euler angles

$$ \Theta_q(q_w, q_x, q_y, q_z) \triangleq \tag{149} $$
$$ \begin{bmatrix} \operatorname{atan2}\big(2(q_w q_x + q_y q_z), q_w^2 + q_z^2 - q_x^2 - q_y^2\big) \\ \begin{cases} \frac{\pi}{2} & 2(q_w q_y - q_z q_x) \geq 1 \\ -\frac{\pi}{2} & 2(q_w q_y - q_z q_x) \leq -1 \end{cases} \\ \operatorname{asin}(2(q_w q_y - q_z q_x)) \quad \text{otherwise} \\ \operatorname{atan2}\big(2(q_w q_z + q_x q_y), q_w^2 + q_x^2 - q_y^2 - q_z^2\big) \end{bmatrix}. $$

When a rotation matrix needs to be made from Euler angles

$$ R_\Theta(\phi, \theta, \psi) \triangleq \tag{150} $$
$$ \begin{bmatrix} \mathrm{c}\theta\,\mathrm{c}\psi & \mathrm{s}\phi\,\mathrm{s}\theta\,\mathrm{c}\psi - \mathrm{c}\phi\,\mathrm{s}\psi & \mathrm{c}\phi\,\mathrm{s}\theta\,\mathrm{c}\psi + \mathrm{s}\phi\,\mathrm{s}\psi \\ \mathrm{c}\theta\,\mathrm{s}\psi & \mathrm{s}\phi\,\mathrm{s}\theta\,\mathrm{s}\psi + \mathrm{c}\phi\,\mathrm{c}\psi & \mathrm{c}\phi\,\mathrm{s}\theta\,\mathrm{s}\psi - \mathrm{s}\phi\,\mathrm{c}\psi \\ -\mathrm{s}\theta & \mathrm{s}\phi\,\mathrm{c}\theta & \mathrm{c}\phi\,\mathrm{c}\theta \end{bmatrix} $$

defines the 3-2-1 Euler angle rotation matrix[47]. Likewise, a 3-2-1 rotation matrix is made from quaternion values with

$$ R_q(q_w, q_x, q_y, q_z) \triangleq \tag{151} $$
$$ \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_w q_z) & 2(q_x q_z + q_w q_y) \\ 2(q_x q_y + q_w q_z) & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_w q_x) \\ 2(q_x q_z - q_w q_y) & 2(q_y q_z + q_w q_x) & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix}. $$

When rotating angular rates from one frame to another

$$T_{\Theta}(\phi,\theta,\psi) \triangleq \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \tag{152}$$

is used[47]. The inverse of this rotation is given by

$$T_{\Theta}^{-1}(\phi,\theta,\psi) \triangleq \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix}. \tag{153}$$

The skew-symmetric operator $[\cdot]_{\times}$ is defined as

$$\left[\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T\right]_{\times} \triangleq \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}. \tag{154}$$

Author Biographies

James Swedeen is a doctoral candidate in the department of Electrical and Computer Engineering at Utah State University. He has worked as a graduate research assistant at Utah State University since Fall 2018. His research interests lie in autonomous robotic path planning under kinematic constraints, state estimation, and population-best metaheuristic algorithms. His research has been focused on applying kinematic constraints within Rapid-exploring Random Tree and Batch Informed Trees paradigms. Additionally, applying linear covariance analysis to rapid path planning applications. He completed his BS in Computer Engineering at Utah State University in 2021.

Greg N. Droge is an Assistant Professor in the department of Electrical and Computer Engineering at Utah State University. His research is focused on dynamically constrained planning for both large groups and individual vehicles operating in uncertain environments. While the applications have been diverse (satellite constellations, battery electric bus coordination, GPS denied navigation, etc.), the underlying research includes developing receding horizon optimization techniques to manage and plan for resource use while considering dynamic constraints. Dr. Droge completed his PhD at Georgia Tech in Electrical and Computer Engineering in 2014 and an MS in 2012. Prior to Georgia Tech, he studied at Brigham Young University where he graduated Magna Cum Laude with a BS in Electrical Engineering in 2009

CHAPTER 7

METAHEURISTICS FOR THE ELECTRIC VEHICLE PATROL ROUTE PLANNING PROBLEM

Submitted to The Journal of *Applied Soft Computing*

Metaheuristics for the Electric Vehicle Patrol Route Planning Problem

James Swedeen*, Greg Droge
Department of Electrical and Computer Engineering,
Utah State University, Logan, UT 84322, USA,
E-mail addresses: james.swedeen@usu.edu (J. Swedeen), greg.droge@usu.edu (G. Droge)

***Abstract*—The patrol routing problem (PRP), a version of the vehicle routing problem (VRP), is vital to the health and well-being of the general public. This work considers a version of the PRP where battery electric vehicles (EVs) are being planned for, a consideration that has not been made in the PRP literature before. The EV model derived herein is more detailed than what exists in the VRP literature and is compared against a distance-based formulation of EV related constraints that is common in the VRP literature. Several population-based metaheuristic algorithms are proposed to solve the developed problem: hybrid population-based simulated annealing (HPSA), hybrid genetic path relinking algorithm (HGPRA), and hybrid firefly algorithm (HFA). A novel use of the Levenshtein edit-distances to define inner population distances and crossover strategies is proposed while developing the aforementioned algorithms. Each planner is evaluated with several selection procedures on problems queried from real-world street graphs. Comparisons show that the HGPRA is the most effective algorithm considered herein for solving the proposed PRP.**

***Index Terms*—Vehicle routing problem (VRP), Patrol routing problem (PRP), Electric vehicles (EVs), Metaheuristics, Evolutionary algorithms**

1 Introduction

Patrolling is the act of periodically traversing an area with the goal of protecting or monitoring the area. The patrol routing problem (PRP) is a specialization of the vehicle routing problem (VRP) where routes are planned for a fleet of agents so that they monitor an area. A typical strategy in law enforcement is to patrol "hotspots" or certain locations with increased incidence rates of vehicle crashes, criminal activity, or other public disturbances. Due to a limited number of patrolling agents, not all hotspots can be continuously monitored. Therefore, the PRP becomes a combinatoric optimization problem to maximize coverage of all hotspots.

Efficiently solving the PRP is essential as the purpose is to prevent harm and foster a public sense of security. Several factors complicate the design of a routing strategy for police patrolling. First, patrolling officers must monitor hotspots regularly and frequently. The increased presence of law enforcement deters would-be criminals before a crime is committed and encourages safe driving practices among the general public [1]. Second, covering all hotspots efficiently requires significant coordination among the patrolling officers. Third, while patrolling, at least one officer must respond to all emergency calls that occur, causing them to delay their patrol schedule while responding.

Additionally, the increasing use of electric vehicles (EVs) has added new complications to the PRP. When patrolling in EVs, considerations must be made to ensure that the EVs have enough battery power to complete the planned patrol route. To compound these considerations, one must also consider whether the patrolling officers will have enough battery power to respond to emergencies quickly. This work calls this problem the electric vehicle patrol routing problem (EVPRP).

Despite the PRP being of the utmost importance to public health, Dewinter, in one of the only reviews on the subject, notes that the literature on the problem is sparse [2]. Work focusing on search and rescue applications will typically not consider any environmental obstacles, instead using aerial vehicles to perform the monitoring [3]. Of the works that consider an urban setting where patrolling agents must follow the streets of a city, some use simple heuristic-based planners without any sort of optimality guarantee [4], [5]. Most use direct optimization techniques where the problem is formulated as an mixed integer program (MIP) and then solved using a third party optimization toolbox [4], [6]–[8]. While these planners can find optimal results, they also take prohibitively long to solve medium to larger sized problems. Many of these works will derive metaheuristic-based planners so they can solve larger problems [4], [6], [8], [9]. Metaheuristic planners are capable of finding solutions to much larger problems faster than MIP-based planners at the cost of having no guarantees that the resulting solution is optimal.

This work aims to contribute to the field of solving the PRP with several population-based metaheuristic algorithms. Additionally, this work expands the field to incorporate EV constraints that, to the knowledge of the authors, have not been studied in the PRP. The EV constraints modeled herein match the charging constraints more closely than any VRP literature the authors are aware of. In developing these metaheuristic algorithms, several novel ideas are proposed. First, the Levenshtein distance calculation, commonly used to find related words in spell checkers, is used to define the solution mixing procedure required by the path relinking (PR) heuristic [10], [11]. The same Levenshtein distance idea is used to define the attraction and movement between fireflies in a version of the

* Corresponding author.
This material is based in part upon work supported by the National Science Foundation through the ASPIRE Engineering Research Center under Grant No. EEC-1941524.

firefly algorithm (FA). The proposed FA formulation maintains more of the original form and intention of the algorithm than previous attempts at using FA for the VRP [12]–[14]. Finally, a hybrid population-based simulated annealing (HPSA) algorithm is proposed, and all proposed metaheuristic algorithms are tested with multiple selection strategies. Since the EVPRP literature is so sparse, this work contributes to the field by testing and comparing several solution strategies to determine those that are effective.

The rest of this work proceeds as follows. Section 2 gives an overview of the relevant literature. Section 3 describes a formulation of the EVPRP using both detailed battery-based constraints and distance-based constraints. Section 4 describes several procedures and methods that are used by the planners proposed in Section 5. Section 6 performs a simulation study on each planner and compares the strengths and weaknesses of each. Concluding remarks and directions for future work are given in Section 7. Appendix A provides a complete list of the acronyms and nomenclature used in this work.

2 Literature Review

This section discusses the literature relevant to this work. Due to the sparsity of literature on the PRP, works that cover the VRP are included in this section because they have challenges similar to the PRP and include more works that consider EV constraints. The VRP arises when a fleet of vehicles is tasked with achieving a set of geographically distributed goals with minimal cost. Common situations where the VRP arises are mail delivery and product distribution [14].

The VRP with time windows on when goods must be delivered in an urban environment is studied in [15]. In [16], the same problem is studied except that the streets have time-dependent travel times and vehicles have load capacity constraints. Another article studies the VRP with limitations on load and battery capacities, time-dependent travel times, and road tolls [17]. In [18], a version of the VRP is considered in which the objective is to water dirt roads to reduce dust. The humidity of the roads changes with time, and each truck has a maximum water capacity. In all four works, their problems are formulated as an mixed integer linear program (MILP) by breaking the planning time horizon into segments and formulating the constraints of the problem within each of these time windows. In [15], [16], a generalized procedure is provided to select time windows such that the start and end of each correspond to every decision boundary of the problem. Zhang and Riquelme split the time horizon into fixed-length intervals and incur a potential loss of precision as a result [17], [18]. To improve their solving times, [16]–[18] develop heuristic algorithms, versions of the adaptive large neighborhood search algorithm, which produce solutions faster than the MILP solvers but have weak optimality guarantees. In [15], solving times are accelerated by defining lower and upper bounds on the objective function and using these bounds in their solver.

Several metaheuristic algorithms have been proposed to solve VRPs. In [19], they develop a HPSA algorithm with a crossover operator that they call ISA-CO to solve the VRP in an urban setting. This algorithm is similar to a hybrid genetic algorithm (HGA), i.e., a genetic algorithm (GA) that has been augmented with occasional greedy local searches, except it uses the simulated annealing (SA) cooling heuristic when deciding to keep a less optimal solution after a local search. In [20], a few HGAs are developed to solve the distance-constrained vehicle routing problem (DVRP) in an urban setting where the objective is to minimize the distance traveled. In [21], the effectiveness of four common GA crossover operations, sequential constructive crossover (SCX), cycle crossover, partially mapped crossover (PMX), and alternate edge crossover, is studied in the context of the DVRP. They find that SCX works best for DVRPs. A novel k-means selection strategy for GAs is proposed in [22]. When performing selection, k-means clustering is performed on the population, and the resulting clusters are used to weight the probability of selecting a member of the population such that clusters with fewer members have a higher chance of being selected. This evens out the selection pressure and allows multiple parts of the search space to be explored efficiently at the same time.

In [13], [14], the FA is modified to solve the capacitated vehicle routing problem (CVRP) in an urban setting. The same problem with pickups, deliveries, and variable transport times is solved with a modified FA in [23]. All three use the Hamming distance to define inter-firefly similarity. Crossover operations are used to move less fit solutions towards more fit solutions in [13], [14]. Additionally, they hybridize the algorithm with the improved 2-opt and 2-h-opt procedures and include two mutation operators for diversification. In [14], they suggest a cooperative FA using concepts from the cooperative island model. In this algorithm, multiple hybrid FA instances are run independently. Periodically, some of the population of each instance is mixed with the other FA instances to prevent the algorithm from getting stuck in local minima. In [23], the firefly movement heuristic is implemented in a way that is similar to a restricted variable neighborhood descent (VND) operation, where several modifications are made to the plan and the one that produces the best cost becomes the moved firefly.

In [3], the problem of searching an open area with a fleet of aircraft is considered. They design a potential-field heuristic that pushes agents to unobserved regions of the search space and away from other agents in the fleet. While [3] has promising results, it does not plan over an urban street graph but a continuous, obstacle-free space.

The PRP in an urban setting, and what metrics are the most important for it, are studied in [5]. They develop a heuristic solver called Bayesian ant-based patrolling strategy, which operates on a discrete graph of hotspots. Each hotspot is associated with a “pheromone” level that increases when the hotspot is visited and decays exponentially when a patrolling agent is not present. Patrolling agents are drawn to close hotspots with low levels of pheromone. In [6], the same problem is considered, but the heuristic solution strategy models the problem in a different way. Each hotspot has an expected disturbance rate (EDR), an estimated model of how much crime or public disturbance will occur in that area. The

TABLE I: An overview of the features present in the literature. An asterisk (*) indicates a limited formulation.

	Paper	World			Battery			Hotspots		Solution Methods		
		Open	Urban	Asymmetric	Distance-Based	Battery Model	Regen. Braking	Nonuniform	EDR Modeled	MIP	Heuristic	Metaheuristic
	Proposed	-	✓	✓	✓	✓	✓	✓	✓	-	✓	✓
PRP	[3]	✓	-	-	-	-	-	-	✓	-	✓	-
PRP	[4]	-	✓	✓	-	-	-	-	-	✓	✓	✓
PRP	[5]	-	✓	✓	-	-	-	✓	-	-	✓	-
PRP	[6]	-	✓	✓	-	-	-	✓	✓	✓	-	✓
PRP	[7]	-	✓	✓	-	-	-	✓	-	✓	-	-
PRP	[8]	-	✓	-	-	-	-	✓	-	✓	-	✓
PRP	[9]	✓	-	-	-	-	-	-	✓	-	-	✓
VRP	[13]	-	✓	-	-	-	-	-	-	✓	-	✓
VRP	[14]	-	✓	-	✓	-	-	-	-	✓	-	✓
VRP	[15]	-	✓	✓	-	-	-	-	-	✓	-	-
VRP	[16], [18]	-	✓	✓	-	-	-	-	-	✓	-	✓
VRP	[17]	-	✓	✓	-	✓*	-	-	-	✓	-	✓
VRP	[19], [23]	-	✓	✓	-	-	-	-	-	✓	-	✓
VRP	[20], [21]	-	✓	✓	✓	-	-	-	-	-	-	✓

EDR decreases while an agent is monitoring it and increases otherwise. They first formulate an integer program, and then a metaheuristic cross entropy (CE) solver is derived to speed up the solve times. The CE method lends itself well to producing unpredictable paths, although it is unclear whether it would be amenable to complicated cost functions and large numbers of graph vertices.

The PRP in which the objective is to maximize the time spent monitoring the hotspots of an urban city that are only active during certain time windows is studied in [7]. They analyze the MIP formulation in [4], which is NP-hard. They show that the same problem can be formulated as a network flow problem and solved in polynomial time. They then extend the problem formulation to allow for multiple dispatch locations, which results in an NP-hard problem regardless of the formulation.

Two metaheuristic algorithms are developed for the urban PRP in [9] and [8]. A GA is developed to solve a PRP where a crime mapping algorithm predicts crime hotspots and sets the priority of patrolling each hotspot in [8]. In [9], an actor-critic machine learning method is used to find paths that monitor a grid of locations where the input of the actor-critic block comes from a graph attention network fed with the observations of each patrolling agent.

Table I compares the various works referenced herein with this work. The *World* columns denote if the paper is solving for routes along the streets of an *Urban* city, if those street graphs are *Asymmetric* in travel times or distances, or if the planning environment is *Open* without obstacles or roads that must be followed. The *Battery* columns denote how, if at all, the constraints related to battery EVs are formulated. *Distance-Based* constraints simply constrain the path length that can be traversed before recharging. When the *Battery Model* column is checked, the EV model includes a state of charge (SOC) that varies depending on the streets driven along. The *Regen. Braking* column shows which papers consider the case where the EV charges while going downhill via regenerative braking. Note that of the works on the PRP, this work is the only one to include EV considerations. Even in the broader field of solving the VRP, few consider EV constraints, and none, to the knowledge of the authors, consider regenerative braking as is done herein.

The *Hotspots* columns list aspects of how hotspots are handled in each paper. The *Nonuniform* column shows which works allow hotspots to have varying levels of importance, and the *EDR Modeled* column shows which papers directly model the EDR versus maximizing the duration when hotspots are being monitored. Only one other work directly models the EDR and allows for a nonuniform distribution of priority, as this work does. Finally, the *Solution Methods* columns show which works solve their VRPs using *MIP* solving libraries, simple *Heuristic* algorithms, and *Metaheuristic* algorithms.

3 Problem Formulation

In this section, the EVPRP is formulated. This work considers the case where law enforcement officers must patrol the streets of an urban city. The city has designated hotspots that must be regularly monitored.

Section 3.1 describes the streets of the city and the monitored hotspots. Section 3.2 describes the patrolling agents. Section 3.3 describes an abstraction of the street graph that is used by the planners herein. Sections 3.4 and 3.5 describe some additional nomenclature and the metrics used in the sequel.

3.1 Street Graph

The streets of a city are most readily represented as a directed graph. Let $\mathcal{G} \triangleq (\mathcal{V}, \mathcal{E})$ be a directed graph where $\mathcal{V} \subset \mathbb{R}^3$ is the vertex set and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set. The vertex set $\mathcal{V} \triangleq \{v_0\} \cup \{v_1, v_2, \ldots, v_{n_v}\}$ consists of $n_v + 1$ vertices where the streets intersect and $v_0 \in \mathbb{R}^3$ is the position of the dispatch station. The Cartesian position where two or more roads intersect is defined as $v_i \in \mathbb{R}^3 \; \forall i \in [0, n_v]$. Every edge in $\mathcal{E}$ represents one direction of a street, i.e., if a road is two-directional there will be one edge in $\mathcal{E}$ for each direction. Every edge connects two vertices in $\mathcal{V}$. However, every vertex in $\mathcal{V}$ can have any number of entering and exiting edges. Edges are denoted as $e_{ij} = (v_i, v_j) \in \mathcal{E}$ where e_{ij} is an edge that starts at $v_i \in \mathcal{V}$ and ends at $v_j \in \mathcal{V}$. Each edge has an associated travel time, $t_{ij} \in \mathbb{R}_{>0}$, physical length, $d_{ij} \in \mathbb{R}_{>0}$, and battery usage while traversing, $c_{ij} \in \mathbb{R}$.

The final component of the street graph is the hotspots. Each hotpot has an associated EDR. The EDR is a heuristic estimate of the probability that a dispatch call-in occurs from a given location. As described in [6], the EDR varies as a function of location, time of day, and time since the last patrol near a particular position. Let the set of $n_h \in \mathbb{Z}_{>0}$ hotspots considered be denoted $\mathcal{H} \subset \mathcal{V}$. Each hotspot is defined by a location in the graph ${}_h p_l \in \mathcal{V}$ where $l \in \mathcal{H}$ indexes the hotspot. This location is representative of the area in the immediate vicinity around the center point. The EDR within hotspot $l \in \mathcal{H}$ is denoted $\lambda_l(t) : \mathcal{T} \mapsto \left[\underline{\lambda}_l, \overline{\lambda}_l\right]$ where $\overline{\lambda}_l \in \mathbb{R}_{\geq 0}$ and

$\underline{\lambda}_l \in \mathbb{R}_{\geq 0}$ are the maximum and minimum EDR of hotspot l and $\mathcal{T} \triangleq [T_s, T_e]$ is the planning time window that starts at time $T_s \in \mathbb{R}$ and ends at time $T_e \in \mathbb{R}_{\geq T_s}$. While a patrolling agent monitors a particular hotspot, the EDR associated with that hotspot decreases over time at a rate of ${}_m\gamma_l \in \mathbb{R}_{<0}$. Likewise, while no patrolling agents are monitoring a hotspot, its associated EDR increases over time at a rate of ${}_n\gamma_l \in \mathbb{R}_{>0}$.

3.2 Patrolling Agents

This section defines the model used to simulate the patrolling agents. The set of $n_a \in \mathbb{Z}_{>0}$ agents for which trips must be planned is denoted $\mathcal{A} \triangleq [1, n_a]$. A trip consists of the streets and actions taken by an agent from the time the agent leaves dispatch to the next time the agent returns to the depot. The depot, v_0, is where the patrolling agents start and end their shifts. The position of agent $k \in \mathcal{A}$ at time $t \in \mathcal{T}$ is denoted as ${}_ap_k(t) : \mathcal{T} \mapsto \mathbb{R}^3$ as it traverses $\mathcal{G}$. Each agent has an associated battery SOC $q_k(t) : \mathcal{T} \mapsto [0, \overline{q}_k]$ where $\overline{q}_k \in \mathbb{R}_{>0}$ is the maximum SOC of agent k. The SOC of an agent will vary as the agent traverses the streets in $\mathcal{G}$, depending on the path the agent follows. Furthermore, while an agent is at the depot, v_0, it can charge its battery at a maximum rate of $r_k \in \mathbb{R}_{\geq 0}$. A linear charging model is used because it is an accurate estimate of the charging process up to a SOC of 80% [24]. To ensure accuracy, the maximum SOC of each agent, $\overline{q}_k$, is set to 80% of the physical capacity of the battery. This is a reasonable restriction to make because charging a battery past approximately 80% SOC significantly increases the rate of battery deterioration [24].

There are two sets of constraints associated with each agent. First, there are the working shift time constraints that force the agents to spend all time outside of their shift at the depot, i.e.,

$$\forall k \in \mathcal{A}, \forall t \in \mathcal{T} \setminus [T_k^s, T_k^e], {}_ap_k(t) \equiv v_0, \tag{1}$$

where $T_k^s, T_k^e \in \mathcal{T}$ are the start and end times of the shift assigned to agent k. Equation (1) is used to enforce the working shift schedule of the people driving the EVs that each agent represents.

The second set of constraints associated with each agent ensures proper battery usage throughout the plan. Specifically, the SOC of each agent cannot exceed its maximum possible SOC and cannot go below zero at any point. This work considers two formulations of this constraint. The first formulation integrates the SOCs of each agent directly and ensures that the SOC is always valid, i.e.,

$$\forall k \in \mathcal{A}, \forall t \in \mathcal{T}, 0 < q_k(t) \leq \overline{q}_k. \tag{2}$$

In the SOC formulation, the agents can charge at the depot for any length of time, including performing a partial charging cycle in which they leave the depot before reaching full SOC. This is because the planner is aware of how much SOC each agent has after completing a trip and can intelligently decide how much charging is advantageous.

The second formulation is a distance-based formulation that is common in the literature, e.g., [14], [20], [21]. In this formulation, each agent has an associated maximum travel distance that can be covered before recharging $\overline{d}_k \in \mathbb{R}_{>0}$.

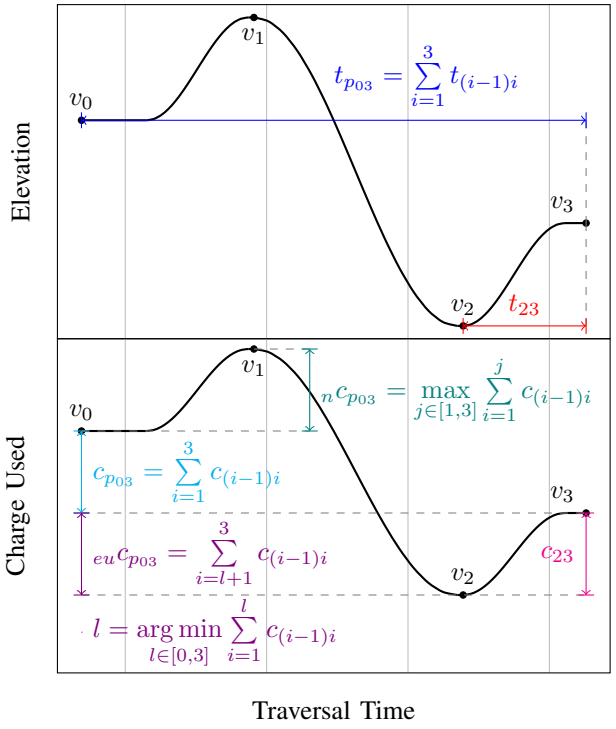


Fig. 1: The elevation over time of a path $p_{03} \in \mathcal{P}_{03}$ that goes from vertex $v_0 \in \mathcal{V}$ to $v_3 \in \mathcal{V}$ above and the charge used over time of the same path below. The traversal time and charge of edge (v_2, v_3) are shown in red and magenta, respectively. The traversal time and charge of path p_{03} are shown in blue and cyan, respectively. The minimum charge necessary to traverse path p_{03} is shown in teal. The end charge usage is shown in violet.

This distance is calculated from the properties of the agent to ensure that any trip of distance $\overline{d}_k$ or less will satisfy the SOC constraints if the agent starts with maximum SOC. When an agent returns to the depot, it is forced to recharge for at least long enough to bring the SOC of the agent from zero SOC to max SOC, i.e., $\overline{q}_k / r_k$ seconds. This is because the planner has no way to know what the SOC of the agent will be after a trip. To aid in formulation in the sequel, let $\forall k \in \mathcal{A}, d_k(t) : \mathcal{T} \mapsto \mathbb{R}_{\geq 0}$ be the distance that agent k has traveled at time t since the last time it has charged. The distance-based constraint can then be stated

$$\forall k \in \mathcal{A}, \forall t \in \mathcal{T}, d_k(t) \leq \overline{d}_k. \tag{3}$$

Distance-based constraints are included in this work because they are common in the literature and are used for comparison to the proposed SOC-based constraints.

3.3 Planning Graph

As is common in the literature, e.g., [13]–[16], [18]–[20], the solvers in this work will only consider the vertices in the street graph that correlate to the hotspots and the depot. Let this reduced-planning vertex set be denoted $\mathcal{V}^p \triangleq \mathcal{H} \cup \{v_0\}$. This reduced vertex set will be connected with paths or ordered sets of edges that an agent can follow through the graph.

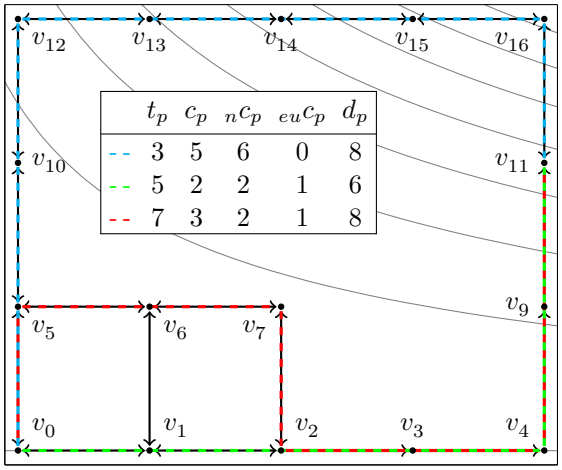


Fig. 2: Example paths for traversing between v_0 and v_{11} with topographic contour lines shown in gray. With a small breach in realism, it is assumed that the paths have the same charge usage going both from v_0 to v_{11} and from v_{11} to v_1.

The planning graph, $\mathcal{P}$, is the set of all possible paths through $\mathcal{G}$. The set of paths that start from the vertex v_i and go to the vertex v_j is denoted $\mathcal{P}_{ij} \subseteq \mathcal{P}$. The traversal time and distance of the path $p \in \mathcal{P}$ are given as $t_p \in \mathbb{R}_{>0}$ and $d_p \in \mathbb{R}_{>0}$, respectively. Likewise, the charge used while traversing the path p is $c_p \in \mathbb{R}$. The minimum charge necessary to traverse the path p is denoted as ${}_nc_p \in \mathbb{R}_{\geq 0}$. The difference between the charge used and the charge necessary is illustrated in Figure 1, where the two values are calculated for an example path. The charge used along the path p_{03} is calculated by summing the charge used along each edge that makes up the path. In the case of Figure 1 the charge used, $c_{p_{03}}$, is negative. However, a vehicle needs at least c_{01} charge (a positive amount) to reach the vertex v_1 at the top of the first incline. Finally, let the positive charge used after the minimum charge point in a path, or the end usage, be denoted ${}_{eu}c_p \in \mathbb{R}_{\geq 0}$. To see why this value is important, consider the case where a vehicle that traverses the path shown in Figure 1 begins the path with max SOC $\overline{q}_k$. Given that the charge used along this path is negative, $c_{p_{03}} < 0$, one might assume that the SOC at the end of the path would still be at max. However, this is not the case. The vehicle will have maximum SOC at the vertex v_2, but the charge will decrease as the vehicle traverses the edge e_{23}. The resulting charge at the end of the path is then given by $\overline{q}_k - {}_{eu}c_{p_{03}}$.

For some methods, it will be useful to define the optimal path between the vertices in $\mathcal{G}$ in terms of minimal time, length, charge used, and the charge needed. Let $p_{ij}^{t*} \in \mathcal{P}_{ij}$, $p_{ij}^{d*} \in \mathcal{P}_{ij}$, $p_{ij}^{c*} \in \mathcal{P}_{ij}$, and $p_{ij}^{cn*} \in \mathcal{P}_{ij}$ be the minimum time, distance, charge used, and charge needed paths that traverse from vertex $v_i \in \mathcal{V}$ to vertex $v_j \in \mathcal{V}$, respectively. The quantities p_{ij}^{t*} and p_{ij}^{d*} are calculated using the well known Dijkstra's algorithm [25]. However, the Bellman-Ford algorithm [26] is used to find p_{ij}^{c*} and p_{ij}^{cn*} because some edges can have negative traversal charge usages.

Many works on the VRP will only consider one path between any two vertices, that is, $\forall i, j \in \mathcal{V}^p, 1 \equiv |\mathcal{P}_{ij}|$. However, doing so overconstrains the problem. To see why, consider the street graph shown in Figure 2. It shows three possible paths between v_0 and v_{11} shown in red, green, and blue, along with the five parameters of the paths. Consider an agent currently at v_0 with SOC of $q_k(0) = 8$ and a maximum trip distance of $\overline{d}_k = 15$. The agent needs to get to v_{11} as quickly as possible while still being able to get back to v_0. When considering SOC constraints the agent can follow the blue path to arrive at v_{11} at time 3 and the green path back to v_0 because the ending SOC is $q_k(0) - 5 - 2 = 1 = q_k(8) > 0$. If one path existed between v_1 and v_{11}, then in the case where the one path is the green path, the agent would not be able to get to v_{11} as quickly. In the case where the one path was the blue path, the agent would not be able to reach v_{11} without violating SOC constraints. The same happens with distance-based constraints, where the agent must use the blue path to get to v_{11} as fast as possible, but can only do so if the green path exists.

Evaluating conditions like $\exists p_{ij} \in \mathcal{P}_{ij}$ such that some condition holds is computationally intractable because the set $\mathcal{P}_{ij}$ contains an infinite number of paths. To help with this, a new set of paths is defined $\widehat{\mathcal{P}} \subset \mathcal{P}$ such that all paths that can contribute to satisfying the constraints are present, but no additional paths are. An example of a path that is present in $\mathcal{P}$ but not in $\widehat{\mathcal{P}}$ is the red path shown in Figure 2. When comparing the red and green paths in this figure, it can be seen that the red path has greater than or equal traversal time, charge used, etc. Because of this, an agent traversing between v_0 and v_{11} would never use the red path. The only way doing so would be advantageous is if getting to v_{11} at time 7 is in some way better than time 5. However, the agent could always wait at v_0 until time 2 before following the green path, which would use less SOC and distance than taking the red path. The set of paths starting as v_i and ending at v_j that can be useful in a plan when distance-based constraints are used is defined as

$$\widehat{\mathcal{P}}_{ij} \triangleq \left\{ p_{ij} \in \mathcal{P}_{ij} \;\middle|\; \begin{array}{l} \nexists p'_{ij} \in \mathcal{P}_{ij} \setminus \{p_{ij}\} : \\ t_{p'_{ij}} \leq t_{p_{ij}} \wedge \\ d_{p'_{ij}} \leq d_{p_{ij}} \end{array} \right\}. \tag{4}$$

When full SOC-based constraints are considered, this set has the definition

$$\widehat{\mathcal{P}}_{ij} \triangleq \left\{ p_{ij} \in \mathcal{P}_{ij} \;\middle|\; \begin{array}{l} \nexists p'_{ij} \in \mathcal{P}_{ij} \setminus \{p_{ij}\} : \\ t_{p'_{ij}} \leq t_{p_{ij}} \wedge \\ c_{p'_{ij}} \leq c_{p_{ij}} \wedge \\ {}_nc_{p'_{ij}} \leq {}_nc_{p_{ij}} \wedge \\ {}_{eu}c_{p'_{ij}} \leq {}_{eu}c_{p_{ij}} \end{array} \right\}. \tag{5}$$

If there exist paths between v_i and v_j that satisfy the return time and SOC or distance constraints, at least one will be in this set, see Appendix D for derivation.

3.4 Notation

This section describes various notation used in the sequel. For example, the clamp operator, $\cdot|_{[\cdot,\cdot]}$, that bounds its input between a given range, is defined as

$$\forall x, a \in \mathbb{R}, \forall b \in \mathbb{R}_{\geq a},\ x|_{[a,b]} \triangleq \max(\min(x, b), a). \tag{6}$$

The notation $p \xleftarrow{R} \mathcal{P}$ denotes that p gets an element of $\mathcal{P}$ with uniform probability over all elements of the set. Furthermore, the notation $p \xleftarrow{R,\Pi} \mathcal{P}$ indicates that p gets an element of $\mathcal{P}$ with probably proportional to the corresponding element of $\Pi \subset \mathbb{R}_{[0,1]} : |\Pi| \equiv |\mathcal{P}|$. The notation $\omega_{[a,b]} \sim U(a,b)$ denotes a continuously-valued uniform random variable bounded between $a \in \mathbb{R}$ and $b \in \mathbb{R}_{>a}$.

Given an agent $k \in \mathcal{A}$ currently with SOC $q \in [0, \overline{q}_k]$, the SOC of the agent after traversing a path $p \in \mathcal{P}$ is calculated as

$$eSOC_k(q,p) \triangleq (q - c_p)|_{[0,\overline{q}_k - {}_{eu}c_p]}. \tag{7}$$

To see why the maximum possible SOC is lowered to $\overline{q}_k - {}_{eu}c_p$, consider the case from Figure 1 described before. If the agent reaches maximum SOC at v_2, some SOC is still used to reach v_3. Assuming the same agent is at the depot, the time it takes for the agent to charge to SOC $q_t \in [0, \overline{q}_k]$ is calculated as

$$chT_k(q, q_t) \triangleq \max((q_t - q)/r_k, 0). \tag{8}$$

Let the set of all possible plans for agent $k \in \mathcal{A}$ be denoted by $\mathcal{X}_k \subset \mathcal{P} \times \Delta T$, where $\Delta T \triangleq T_e - T_s \in \mathbb{R}_{\geq 0}$ is the duration of the planning time window. The form of these subplans is an ordered list of tuples, where each tuple defines the vertex to which the agent travels, the path it takes to get there, and the length of time the agent dwells at that vertex. The full solution space $\mathcal{X} \subseteq \mathcal{X}_1 \times \mathcal{X}_2 \cdots \times \mathcal{X}_k$ is defined as a list of n_a agent plans, i.e., $\forall k \in \mathcal{A}, x_k \in \mathcal{X}_k, x = \{x_1, x_2, \ldots, x_k\}$ where $x \in \mathcal{X}$. For example, consider (9) that defines the plan of an agent that is to travel to vertex 6 via path $p_{06} \in \mathcal{P}_{06}$ then to vertex 2 via path $p_{62} \in \mathcal{P}_{62}$ before returning to the depot via path $p_{20} \in \mathcal{P}_{20}$.

$$x \triangleq \left\{ \left(\emptyset, \tau_1^1\right), \left(p_{06}, \tau_2^1\right), \left(p_{62}, \tau_3^1\right), \left(p_{20}, \tau_4^1\right) \right\} \tag{9}$$

Note that the first visit tuple of each agent, $\left(\emptyset, \tau_1^1\right)$, has no previous vertex, thus no path from the previous vertex. The dwell times spent in the depot $\tau_1^1, \tau_4^1 \in \Delta T$ are used to charge the battery of the agent, whereas the dwell time spent at hotpots $\tau_2^1, \tau_3^1 \in \Delta T$ drives the EDR of the hotspots down. The process of adding a visit tuple to the plan of agent k at index $y \in \mathbb{Z}_{\geq 0}$ is denoted as $\overset{k,y}{+}$. Likewise, the process of removing the yth visit from the plan of agent k is denoted $\overset{k,y}{-}$. Additionally, $\overset{k,e}{+}$ means add the tuple to the end of the plan of agent k.

3.5 Metrics

This section defines the optimality metrics that the planners proposed in the sequel minimize. The primary objective of the proposed planners is to minimize the EDRs across the city being monitored. Metrics defined with EDRs allow for direct consideration of the nonuniform and time-of-day variable nature of crime occurrences [6]. As a secondary objective, this work considers the minimum response time of patrolling agents to possible emergencies throughout the planning period.

Often, patrolling officers will need to respond to emergency calls as they are received. The agent that responds to an emergency must have enough SOC to get to the emergency and return to a charger at the depot. With this in mind, the minimal response time to an emergency located at $v_e \in \mathcal{V}$ given solution $x \in \mathcal{X}$ and at time $t \in \mathcal{T}$ is defined in (10).

$$J_{rt}(x,t,v_e) \triangleq \min_{k \in [0,n_a]} t_{p^{t*}_{a p_k(t) e}} \tag{10a}$$

$$\text{s.t. } {}_nc_{p^{t*}_{a p_k(t) e}} \leq q_k(t) \tag{10b}$$

$${}_nc_{p^{cn*}_{e0}} \leq eSOC_k\left(q_k(t), p^{t*}_{a p_k(t) e}\right) \tag{10c}$$

The constraint in (10b) ensures that the selected agent has enough charge to get to the emergency, and (10c) ensures that the agent has enough charge to get back to the depot. When distance-based constraints are used, (10) reduces to (11).

$$J_{rt}(x,t,v_e) \triangleq \min_{k \in [0,n_a]} t_{p^{t*}_{a p_k(t) e}} \tag{11a}$$

$$\text{s.t. } d_k(t) + d_{p^{t*}_{a p_k(t) e}} + d_{p^{d*}_{e0}} \leq \overline{d}_k \tag{11b}$$

The constraint in (11b) ensures that the chosen agent can go to the event and return to the depot without violating the maximum trip distance constraints.

The two and infinity norms of the EDRs and response times across all potential emergency locations are considered. The 2-norm of these quantities gives an idea of the average value across the entire area being monitored. If the average is kept at a minimum, then it is known that the total summed EDR and response times will be minimized throughout the city. The infinity-norms provide the maximal or worst case values across the area being monitored. This is important to minimize because it incentivizes all locations to be considered instead of just the easy to access locations.

The overall objective function for a solution $x \in \mathcal{X}$ is given in (12), where $\alpha_{edr,2}, \alpha_{edr,\infty}, \alpha_{rt,2}, \alpha_{rt,\infty} \in \mathbb{R}_{\geq 0}$ are the weighting coefficients for the 2 and infinity norms of the EDR and response times, and $\mathcal{V}_e \subseteq \mathcal{V}$ is a representative subset of the vertices under consideration.

$$\begin{aligned} J(x) \triangleq\ & \alpha_{edr,2} \int_{T_s}^{T_e} \left\| \left[\lambda_1(t), \lambda_2(t), \ldots, \lambda_{n_h}(t)\right] \right\|_2 dt \\ & + \alpha_{rt,2} \int_{T_s}^{T_e} \left\| \left[J_{rt}(t, v_{e,1}), \ldots, J_{rt}\left(t, v_{e,|\mathcal{V}_e|}\right)\right] \right\|_2 dt \\ & + \alpha_{edr,\infty} \max_{t \in \mathcal{T}, l \in \mathcal{H}} \lambda_l(t) + \alpha_{rt,\infty} \max_{t \in \mathcal{T}, v_e \in \mathcal{V}_e} J_{rt}(t, v_e) \end{aligned} \tag{12}$$

4 Planning Sub-Procedures

This section defines several functions and procedures that are used by the population-based metaheuristic planners proposed in the sequel. Many metaheuristic algorithms operate on a set of solutions called a population. The members of the population are combined to produce new solutions or made to cooperate through a learned process. Many such algorithms have much in common with the first population-based metaheuristic, the GA, proposed by Holland in 1975 [27].

The GA is inspired by nature and mimics the natural biological process of evolution. A GA begins with a population of candidate solutions. The GAs this work is based on has three steps that are performed repeatedly on the population. Step

one is reproduction or crossover, where two solutions in the population mix their properties to produce an offspring, which hopefully has beneficial characteristics from both parents. Step two is mutation, where each solution in the population has a probability of randomly changing its characteristics. Mutations produce solutions with new characteristics to explore. The final step is selection, where a subset of the original population and the produced offspring are selected to become the next population. This step mimics the natural selection pressures described in the theory of evolution.

This section proceeds as follows. Section 4.1 defines a way of measuring dissimilarity between solutions in the population. Additionally, a way to pass characteristics from one solution to another is derived using this distance metric. The way in which the initial population is generated is described in Section 4.2. Sections 4.3 and 4.4 describe the selection and mutation methods used herein. Finally, a greedy local search procedure is defined in Section 4.5.

4.1 Distance Calculation

Some of the solvers used in this work require a measure of distance between solutions and/or a step-by-step sequence of operations that transition one solution into another. This work addresses both needs with a modified version of the unrestricted Levenshtein distance [10].

The Levenshtein distance is an example of an edit-distance and was originally developed for dictionary searches in spell checkers. Edit-distances define the distance between two words in terms of the minimum length sequence of edit-operations necessary to transform one word into the other word being compared. In the case of the Levenshtein distance, there are three such edit-operations: insertion, deletion, and substitution. Each inserts, deletes, or substitutes one letter in a word, e.g., $adc \rightarrow abc$ substitutes d with b

In this work, instead of comparing words, the distance calculation is carried out between solutions in $\mathcal{X}$. In doing so, the list of visit tuples that make up a solution are considered a "word" where each "letter" is a visit tuple. To show how this works, the distance increment associated with each edit-operation is described; $W_i, W_d, W_s \in \mathbb{R}_{>0}$ for the insertion, deletion, and substitution edit-operations, respectively. The sum of these distance increments will become the distance between solutions, so a small distance increment denotes a small change between visit tuples, and a large one denotes a large change. The distance increments are defined in terms of three weights. Each weight is associated with changing a different part of a visit tuple; $w_v, w_p, w_{dt} \in \mathbb{R}_{>0}$ for the vertex being visited, the previous path, and the dwell time, respectively. The distance increments of the insertion and deletion edit-operations are constants equal to modifying the vertex, path, and dwell time of a visit, that is, $W_i = W_d \triangleq w_v + w_p + w_{dt}$. The distance increment of substitution depends on the tuples being substituted. Let $\left(p_{ba}, \tau_c^d\right)$ be the tuple that is being substituted with $\left(p_{ze}, \tau_g^h\right)$ then

$$W_s \triangleq w_v \mathcal{I}_{v_a \neq v_e} + w_p \mathcal{I}_{p_{ba} \neq p_{ze}} + w_{dt} \frac{\left|\tau_c^d - \tau_g^h\right|}{\tau_{max}}, \tag{13}$$

where $\mathcal{I}_A \in \{0, 1\}$ is an indicator variable that equals 1 when condition A is satisfied and 0 otherwise. The maximum possible dwell time, $\tau_{max} \in \Delta T$, is chosen such that $0 \leq \frac{\left|\tau_c^d - \tau_g^h\right|}{\tau_{max}} \leq 1$ is always true.

The sequence of edit-operations that transforms one solution into another is called an edit-script. These edit-scripts are lists of edit-operations that can be applied to solutions in any order. The Levenshtein edit-script generation function is denoted $es : \mathcal{X} \times \mathcal{X} \mapsto \mathcal{ES}$, where $\mathcal{ES}$ is the set of all possible edit-scripts between solutions. Given two solutions, it produces the edit-script that connects those solutions. The sum of all distance increments in the edit-script between two solutions is the edit-distance between them. The Levenshtein edit-distance is denoted as $d : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_{\geq 0}$.

The application of an edit-script to a solution is denoted with the operator $\odot$, e.g., given $x_i, x_j \in \mathcal{X}$ the application of the edit-script between them is denoted $x_i \odot es(x_i, x_j) = x_j$ where the result of applying $es(x_i, x_j)$ to x_i is x_j by the definition of an edit-script. The application of a subset of the edit-operations in an edit-script is denoted by subscripting $\odot$ with a set of edit-script indices. For example, the application of the second and fifth edit-operations is denoted $x_i \odot_{\{2,5\}} es(x_i, x_j)$.

4.2 Initial Population

This section discusses how the initial population is generated in this work. While generating the initial population, the length of dwelling while at a hotspot is always chosen to be a set constant $\tau_{init} \in \Delta T$. The length of time charging at the depot is always chosen to be the shortest length necessary to reach maximal SOC. With this in mind, denote a set of paths that can be taken by an agent, $k \in \mathcal{A}$, at time, $t \in \mathcal{T}$, without violating the constraints of the problem as $\mathcal{VP}_k(t)$. When distance-based constraints are being used, the valid path set takes the form

$$\mathcal{VP}_k(t) \triangleq \left\{ \begin{matrix} p_{a p_k(t) j} \in \widehat{\mathcal{P}}_{a p_k(t) j}, \\ \forall j \in \mathcal{V}^p \end{matrix} \middle| \begin{matrix} \exists p_{j0} \in \widehat{\mathcal{P}}_{j0} : \\ d_k(t) + d_{p_{a p_k(t) j}} + d_{p_{j0}} \leq \overline{d}_k, \\ t + t_{p_{a p_k(t) j}} + \tau_{init} + t_{p_{j0}} \leq T_k^e \end{matrix} \right\}, \tag{14}$$

where condition $d_k(t) + d_{p_{a p_k(t) j}} + d_{p_{j0}} \leq \overline{d}_k$ ensures agent k can traverse path $p_{a p_k(t) j}$ and get back to the depot with violating the maximum trip distance constraint. The other condition, $t + t_{p_{a p_k(t) j}} + \tau_{init} + t_{p_{j0}} \leq T_k^e$, ensures that agent k has enough time before the end of its shift to traverse the path, dwell at vertex j, and get back to the depot. When SOC-based constraints are being used, this set takes the form

$$\mathcal{VP}_k(t) \triangleq \left\{ \begin{matrix} p_{a p_k(t) j} \in \widehat{\mathcal{P}}_{a p_k(t) j}, \\ \forall j \in \mathcal{V}^p \end{matrix} \middle| \begin{matrix} {}_n c_{p_{a p_k(t) j}} \leq q_k(t), \\ \exists p_{j0} \in \widehat{\mathcal{P}}_{j0} : \\ {}_n c_{p_{j0}} \leq eSOC_k\left(q_k(t), p_{a p_k(t) j}\right), \\ t + t_{p_{a p_k(t) j}} + \tau_{init} + t_{p_{j0}} \leq T_k^e \end{matrix} \right\}, \tag{15}$$

where the time-based condition is the same as before. The condition ${}_n c_{p_{a p_k(t) j}} \leq q_k(t)$ ensures agent k has enough SOC to traverse $p_{a p_k(t) j}$. The next condition, ${}_n c_{p_{j0}} \leq$

Algorithm 1: Initialization Planner Form

Input: τ_{init} Dwell times for initializing

```
// Initialize output solution
```
1 $x \leftarrow \emptyset$;
2 **foreach** $k \in \mathcal{A}$ **do**
```
    // Charge until max SOC
```
3 $x \leftarrow x \overset{k,e}{+} (\emptyset, chT_k(q_k(T_k^s), \overline{q}_k))$;
4 $t_k \leftarrow T_k^s + chT_k(q_k(T_k^s), \overline{q}_k)$;
```
// While there are valid actions
```
5 **while** $\exists k \in \mathcal{A}$ *s.t.* $\mathcal{VP}_k(t_k) \not\equiv \emptyset$ **do**
```
    // Select a valid path
```
6 $k, p_{ij} \leftarrow pickPath(\{\mathcal{VP}_k(t_k), \forall k \in \mathcal{A}\})$;
7 **if** $v_j \equiv v_0$ **then** `// Going to depot`
```
      // Charge until battery full
```
8 $x \leftarrow x \overset{k,e}{+} (p_{ij}, chT_k(q_k(t_k), \overline{q}_k))$;
9 $t_k \leftarrow t_k + chT_k(q_k(q_k(t_k)), \overline{q}_k)$;
10 **else**
```
      // Dwell at the hotspot
```
11 $x \leftarrow x \overset{k,e}{+} (p_{ij}, \tau_{init})$;
12 $t_k \leftarrow t_k + \tau_{init}$;
13 **return** x;

$eSOC_k\big(q_k(t), p_{a p_k(t) j}\big)$, ensures agent k has enough SOC to get back to the depot after traversing $p_{a p_k(t) j}$.

Three population initialization methods are used to ensure a diverse initial population. All three population initialization methods have the same form, with the only difference between them being how they choose which new paths to add to the plan. This form is given in Algorithm 1. Lines 1 through 4 initialize each agent subplan at the depot and charges them to maximal SOC. The loop on lines 5 through 12 repeatedly chooses valid paths to add to the current solution, x. The $pickPath$ function on line 6 is what differs between each planner used herein. It picks one of the valid paths to add to the current solution, and which agent subplan to add to. If the chosen path takes the agent to the depot, lines 8 and 9 add the corresponding tuple to the agent plan k that causes it to go to the depot and charge until achieving maximal SOC. Note that when distance-based constraints are used, these lines are replaced with the following.

$$x \leftarrow x \overset{k,e}{+} (p_{ij}, chT_k(0, \overline{q}_k)); \tag{16}$$

$$t_k \leftarrow t_k + chT_k(0, \overline{q}_k); \tag{17}$$

If the chosen path brings agent k to a hotspot, lines 11 and 12 add the corresponding tuple to the plan of agent k to make it dwell at vertex v_j for τ_{init} minutes. The while loop ends when there are no valid paths for any agent to follow. The constraints that define the valid path set are chosen such that when this occurs, the plan that has been made ends with each agent charging at the depot and following all problem constraints.

The initial population is denoted as $P_{init} \subset \mathcal{X}$ and the population size is denoted as $n_P \in \mathbb{Z}_{>0}$. The first element of P_{init} is generated with the greedy action planner (GAP). The GAP defines the $pathPath$ function to minimize the objective function of the solution after adding one more visit tuple to an agent plan. In other words, $pickPath$ takes the form

$$k, p_{ij} \leftarrow \underset{k \in \mathcal{A}, p_{ij} \in \mathcal{VP}_k(t_k)}{\arg\min} J\left(x \overset{k,e}{+} (p_{ij}, \tau_{init})\right). \tag{18}$$

This repeatedly picks the optimal one-step addition to the current plan. The GAP is similar to the sequential greedy algorithm (SGA) described in [28]. However, SGA considers the optimal reordering of visits within each agent plan, every iteration, whereas this planner only considers adding a visit to the end of an agent plan to reduce solve times. The GAP is used to ensure that a good starting solution is present in P_{init}.

The next element of P_{init} is generated with the potential-field accent planner (FAP). The FAP draws inspiration from [3]. They design a potential field that pushes agents to unobserved regions of a continuous, obstacle-free search space and away from other agents in the fleet. The potential field used in this work is identical except that the agents are pushed toward hotspots with high EDRs instead of unobserved regions. The hotspot attraction term is defined using a set of Gaussian radial basis functions (GRBFs) of width $\delta_h \in \mathbb{R}_{>0}$ and weighted by $w_h \in \mathbb{R}_{\geq 0}$. The agent repulsion term is defined using the inverse distance between the agents and is weighted by $w_a \in \mathbb{R}_{\geq 0}$. The value of the potential field for agent k at time t is defined in (19).

$$\begin{aligned} F_k(t) \triangleq \frac{w_h}{2} \sum_{l=1}^{n_h} \lambda_l(t) \exp\left(-\frac{\|{}_a p_k(t) - {}_h p_l\|_2^2}{\delta_h}\right) \\ - \frac{w_a}{2} \sum_{z=1, z\neq k}^{n_a} \|{}_a p_k(t) - {}_a p_z(t)\|_2^{-2} \end{aligned} \tag{19}$$

Each path is chosen in the direction of steepest ascent in $F_k(t)$ for each agent. The FAP is used to produce a high quality but distinct solution.

The final $n_P - 2$ initial solutions are generated with the random action planner (RAP). The RAP picks the path to take at random with a uniform probability of picking each path. This planner is used to produce diverse population samples.

4.3 Selection Procedures

This section describes the two generational selection procedures used in this work. They are used to select which elements of the current population are used as elements of the next population through the generations of the metaheuristic algorithms described in the sequel. When these methods are called, they are referred to via

$$P_{selected} \leftarrow selection(P, n), \tag{20}$$

where $P_{selected} \subseteq P$ is the selected subset of the given population $P \subset \mathcal{X}$ and n is the number of selected elements of P, that is, $n = |P_{selected}|$. The $selection$ procedure can refer to either of the alternatives described in Sections 4.3.1 and 4.3.2.

4.3.1 Fitness Proportional Selection

Given a starting population set, fitness proportional selection (FPS) picks members of the population to become part of the next generation with probability proportional to the fitness of the member being selected. Additionally, the element of the population with the lowest cost value is always selected for the next generation to ensure that progress toward the optimal solution is never lost. FPS is used as a baseline for comparison to the other selection method, as it is common in the literature, e.g., [11], [20].

4.3.2 K-Medoids Selection

Normally, population-based metaheuristics have difficulty with getting stuck in local minima [14]. To combat this tendency, this work proposes a selection method called k-medoids selection (KMS), which helps encourage population diversity and is based on the k-means selection strategy. The k-means selection strategy was first introduced for GAs in [22]. When population selection is needed, k-means clustering is performed on the population set. The resulting clusters are used to weight the probability of selecting a member of the population such that clusters with fewer members have a higher chance of being selected. This evens out the selection pressure across the search space and allows multiple regions of the search space to be explored efficiently at the same time.

The k-means selection process has two main steps: calculate a k-means clustering over the population and define the probability of being selected using the clustering results. In [22], the full k-means clustering algorithm is used over a continuous, 2-normed space. That algorithm iteratively picks new clusters until the process converges with minimal squared error between the cluster members and the means of the cluster members. Furthermore, in [22], the optimal number of clusters is calculated every generation by performing the clustering repeatedly with differing numbers of clusters and using the clustering with the lowest squared error between the members of the cluster and the centers.

To reduce the computational burden of this process and make it amenable to a discrete-valued problem space, five modifications are proposed to the clustering step. First, the unrestricted Levenshtein edit-distance, described in Section 4.1, is used to define the distance between solutions in the population. Second, since the solution space used herein is symbolic, the average of the population is not well defined. Instead, cluster centers are chosen from the elements of the population, otherwise known as the k-medoids problem [29]. Third, the population is partitioned into a fixed number of clusters each generation, $n_{cl} \in \mathbb{Z}_{>0}$, to avoid the time consuming process of finding the optimal number of clusters.

The next modification is that a maximum number of clustering iterations, $i_{cl}^{max} \in \mathbb{Z}_{>0}$, is performed. This means that after picking the location of the cluster centers, a loop runs for at most i_{cl} iterations that assigns each population element to the nearest center and then picks new cluster center solutions that have a minimal summed distance to all other elements of its cluster before repeating. If at any point the cluster centers stay the same as they were in the last iteration, the loop ends immediately. Again, this is to reduce the time it takes to calculate the population clusters. In the sequel, i_{cl}^{max} and many other parameters will be optimized. In the case of i_{cl}^{max}, this means that the optimizer will weigh the benefits of the clustering taking less time against any possible reduction in clustering optimality and set i_{cl}^{max} accordingly.

The final modification made is the use of the initialization heuristic k-means++ [30]. Typically, in k-means, the initial cluster centers are chosen randomly from the population. In k-means++, the initial location of the first cluster center is chosen at random from the population. Subsequent cluster centers are chosen stochastically from the remaining population with probability proportional to the squared distance of each point to the nearest cluster center. This initialization method improves the optimality of the initial clusters generated and reduces the number of subsequent iterations that must be performed.

Once the population clusters have been defined, $\forall c \in [1, \ldots, n_{cl}]$ $M_c \subset P$, the KMS used herein is similar to that of [22]. First, the population element with the lowest objective value in each cluster is automatically carried to the next generation, ensuring that progress made toward finding the optimal solution is not lost. Additional elements are chosen for the next generation stochastically with probability equal to the "membership probability index" of the element. The membership probability index is expressed for $x_i \in P$ assuming that x_i is a part of cluster $c \in [1, \ldots, n_{cl}]$ as

$$MP_c(x_i) \triangleq \frac{|M_c|}{|M_c| - 1} \cdot \frac{1}{|P|} \cdot \frac{\sum_{x_j \in M_c} J(x_j) - J(x_i)}{\sum_{x_j \in M_c} J(x_j)}. \quad (21)$$

This selection probability function has a few important characteristics;

- The sum of the membership probability indices of a given cluster is equal to $\frac{|M_c|}{|P|}$. Consequently, clusters with more members have a higher probability of being selected.
- Within each cluster, solutions with lower objective values receive larger membership probability indices. The result is that fitter solutions within a cluster have a better chance of selection.

This selection probability causes the selection pressure between population members to be stronger when they are in the same cluster than otherwise. The net result of KMS is that members of the selected population are more geographically distant from each other than is the case for FPS. This encourages population diversity and allows different members of the population to explore distinct local basins of the search space.

4.4 Mutation Operators

This section describes several mutation operators that are used to randomly perturb the solution. This simultaneously encourages population diversity and causes planners to explore different local minima in the solution space. The set of all mutation operators used in this work is denoted Θ, and $n_\Theta \triangleq |\Theta|$ is the number of such operators. Each mutation operator has a custom tuned probability of being selected π_j, where j refers to the mutation being considered. The set of all such probabilities is denoted as Π.

Mutation definitions 1 through 9 describe the mutation operators used in this work. During the explanation of each

mutation operator, the example starting solution given in (9) is used as the solution to which the mutation operator is applied. To understand each mutation, it might be helpful to consider the street graph shown in Figure 2.

Mutation 1 (Add Hotspot Visit m_{ahv}): A visit to a random hotspot, $i \xleftarrow{R} \mathcal{H}$, is added to the plan of a random agent, $k \xleftarrow{R} \mathcal{A}$, at a random index, $y \xleftarrow{R} [1, |x_k|]$. For example,

$$m_{ahv}(x) = \\ \left\{\left(\emptyset, \tau_1^1\right), \left(p_{06}, \tau_2^1\right), \left(p_{6i}, \tau_5^1\right), \left(p_{i2}, \tau_3^1\right), \left(p_{20}, \tau_4^1\right)\right\} \tag{22}$$

is the result of adding the visit tuple $\left(p_{6i}, \tau_5^1\right)$ to the middle of the plan of agent 1. The path leading to the new visit, $p_{6i} \xleftarrow{R} \widehat{\mathcal{P}}_{6i}$, and dwell time, $\tau_5^1 \xleftarrow{R} [1, \tau_{max}]$, are chosen randomly. Furthermore, note that the path from the new visit to the next visit in the plan, $p_{i2} \xleftarrow{R} \widehat{\mathcal{P}}_{i2}$, is reassigned randomly so that it starts at hotspot i.

Mutation 2 (Add Depot Visit m_{adv}): This mutation is identical to the Add Hotspot Visit mutation with the exception that the visit location that is being added to the current solution is chosen as the depot, $i \leftarrow 0$.

Mutation 3 (Remove Hotspot Visit m_{rhv} [16]): Removes a visit from the plan of a random agent, $k \xleftarrow{R} \mathcal{A}$, at a random index, $y \xleftarrow{R} [1, |x_k|]$, where only indexes that correlate with hotspot visits are considered. For example,

$$m_{rhv}(x) = \left\{\left(\emptyset, \tau_1^1\right), \left(p_{02}, \tau_3^1\right), \left(p_{20}, \tau_4^1\right)\right\}, \tag{23}$$

where $p_{02} \xleftarrow{R} \widehat{\mathcal{P}}_{02}$ is chosen randomly from the set of paths that start at the depot and end at vertex v_2.

Mutation 4 (Remove Depot Visit m_{rdv}): The mutation is identical to the Remove Hotspot Visit mutation, only the visit tuple being removed is restricted to depot visits.

Mutation 5 (Remove Multiple Visits m_{rmv} [17]): Randomly selects two agent plans, $k_1, k_2 \xleftarrow{R} \mathcal{H}$, and indexes from their plans, $y_1, y_2 \xleftarrow{R} \mathbb{Z}_{>0}$, and removes those visits and all visits that lie between them in the overall solution. Note that any agent subplan start visit tuples are not removed so that the resulting plan has n_a separate agent plans. In addition, any paths that need to be changed to start and end at the correct location are chosen randomly.

Mutation 6 (Swap Path m_{sp}): Randomly selects an agent subplan, $k \xleftarrow{R} \mathcal{A}$, an index in the plan of that agent $y \xleftarrow{R} [1, |x_k|]$, and randomly reassigns the path taken in the chosen visit tuple. For example,

$$m_{sp}(x) = \left\{\left(\emptyset, \tau_1^1\right), \left(p_{06}, \tau_2^1\right), \left(p_{62}^n, \tau_3^1\right), \left(p_{20}, \tau_4^1\right)\right\} \tag{24}$$

reassigns the path taken in the third visit tuple to $p_{62}^n \xleftarrow{R} \widehat{\mathcal{P}}_{62} \setminus \{p_{62}\}$.

Mutation 7 (Visit Swap m_{vs} [13], [14], [16], [18]–[21]): Randomly selects two agent plans, $k_1, k_2 \xleftarrow{R} \mathcal{H}$, and indices from their plans, $y_1, y_2 \xleftarrow{R} \mathbb{Z}_{>0}$, and swaps those visits. For example,

$$m_{vs}(x) = \left\{\left(\emptyset, \tau_1^1\right), \left(p_{06}, \tau_2^1\right), \left(p_{60}, \tau_4^1\right), \left(p_{02}, \tau_3^1\right)\right\} \tag{25}$$

swaps the third and fourth visit tuples choosing new paths which start and end at the correct locations, $p_{60} \xleftarrow{R} \widehat{\mathcal{P}}_{60}$, $p_{02} \xleftarrow{R} \widehat{\mathcal{P}}_{02}$.

Mutation 8 (Multiple Visit Swap m_{mvs} [18]): Randomly selects four visit tuple indices from the solution $y_1, y_2, y_3, y_4 \xleftarrow{R} \left[1, \sum_{k\in\mathcal{A}} |x_k|\right]$ such that $y_1 < y_2 < y_3 < y_4$. The visit tuples indexed from the set $[y_1, y_2]$ switch places within the overall solution with the visit tuples indexed from the set $[y_3, y_4]$ while preserving the order of the visits within each set. When necessary, the paths that connect the swapped sets to the rest of the solution are chosen randomly.

Mutation 9 (Simple Inversion Mutation m_{sim} [31]): Randomly selects three visit tuple indexes from the overall solution, $y_1, y_2, y_3 \xleftarrow{R} \left[1, \sum_{k\in\mathcal{A}} |x_k|\right]$ such that $y_1 < y_2$. Visit tuples indexed from the set $[y_1, y_2]$ are removed from the solution and reinserted after the visit tuple indexed by y_3 in reverse order. For example,

$$m_{sim}(x) = \left\{\left(\emptyset, \tau_1^1\right), \left(p_{00}, \tau_4^1\right), \left(p_{02}, \tau_3^1\right), \left(p_{26}, \tau_2^1\right)\right\} \tag{26}$$

moves the second and third visit tuples behind the fourth visit tuple and reverses their order. Additionally, the paths connecting these visit tuples are reassigned randomly, $p_{00} \xleftarrow{R} \widehat{\mathcal{P}}_{00}$, $p_{02} \xleftarrow{R} \widehat{\mathcal{P}}_{02}$, $p_{26} \xleftarrow{R} \widehat{\mathcal{P}}_{26}$.

Mutation 10 (Dwell Time Tweak m_{dtt}): Randomly selects an agent subplan, $k \xleftarrow{R} \mathcal{A}$, an index in the plan of that agent $y \xleftarrow{R} [1, |x_k|]$, and a time delta $\delta\tau \xleftarrow{R} [-\tau_{max,dtt}, \tau_{max,dtt}]$ to add to the dwell time of the selected visit tuple. The magnitude of $\delta\tau$ is bounded by a constant $\tau_{max,dtt} \in [1, \Delta T]$. For example,

$$m_{dtt}(x) = \left\{\left(\emptyset, \tau_1^1\right), \left(p_{06}, \tau_n^1\right), \left(p_{62}, \tau_3^1\right), \left(p_{20}, \tau_4^1\right)\right\} \tag{27}$$

reassigns the dwell time of the second visit tuple to $\tau_n^1 \triangleq \left(\tau_2^1 + \delta\tau\right)\big|_{[0,\tau_{max}]}$. The reassigned dwell time is bounded between 0 and τ_{max} minutes.

4.5 Variable Neighborhood Descent

As Labadie [11] notes, making use of a local search procedure of some form is essential to the performance of metaheuristic algorithms solving VRPs. The use of a greedy local search in a metaheuristic algorithm is referred to as hybridizing the algorithm. The local search greedily converges an element of the population to the local basin of optimality that they are within much faster than the baseline metaheuristic algorithm could. However, this only reaches a local optimum. It is left to the encompassing hybrid metaheuristic algorithm to search for new and potentially better local basins of optimality.

This work uses the VND algorithm to hybridize the solvers described in the sequel. The implemented VND algorithm makes use of four neighborhoods around a given solution, each defined with respect to a plan mutation operator. The first mutation changes the dwell times of one visit in the existing plan by one minute. To demonstrate, consider the example original solution given in (9) then the first neighborhood set around this solution, $N_1(x)$, is given as

$$\left\{\begin{array}{l} \left\{\left(\emptyset, \tau_1^1 + 1\right), \left(p_{06}, \tau_2^1\right), \left(p_{62}, \tau_3^1\right), \left(p_{20}, \tau_4^1\right)\right\}, \\ \left\{\left(\emptyset, \tau_1^1 - 1\right), \left(p_{06}, \tau_2^1\right), \left(p_{62}, \tau_3^1\right), \left(p_{20}, \tau_4^1\right)\right\}, \\ \left\{\left(\emptyset, \tau_1^1\right), \left(p_{06}, \tau_2^1 + 1\right), \left(p_{62}, \tau_3^1\right), \left(p_{20}, \tau_4^1\right)\right\}, \\ \qquad\qquad \vdots \\ \left\{\left(\emptyset, \tau_1^1\right), \left(p_{06}, \tau_2^1\right), \left(p_{62}, \tau_3^1\right), \left(p_{20}, \tau_4^1 - 1\right)\right\} \end{array}\right\}. \tag{28}$$

The second neighborhood, $N_2(x)$, is defined by removing a visit from the solution. This produces the set

$$\left\{\begin{array}{l}\left\{\left(\emptyset,\tau_1^1\right),\left(p_{02}^{t*},\tau_3^1\right),\left(p_{20},\tau_4^1\right)\right\},\\ \left\{\left(\emptyset,\tau_1^1\right),\left(p_{06},\tau_2^1\right),\left(p_{60}^{t*},\tau_4^1\right)\right\}\end{array}\right\}, \tag{29}$$

where the first and last visit of each agent plan are never removed because the first and last visit have to be depot visits by the constraints of the problem. Additionally, after the first and second visits are removed, the solution is left in an invalid state, $x^{-1,2} = \left\{\left(\emptyset,\tau_1^1\right),\left(p_{62},\tau_3^1\right),\left(p_{20},\tau_4^1\right)\right\}$, because the new second path taken starts at v_6 instead of the depot. To correct this, the minimal time path is substituted between the vertices when necessary, as shown in (29).

The third neighborhood is defined by adding a visit to each valid vertex to the solution in all locations but the start and end of each agent plan. This results in

$$\left\{\begin{array}{l}\left\{\left(\emptyset,\tau_1^1\right),\left(p_{01}^{t*},\tau_5^1\right),\left(p_{16}^{t*},\tau_2^1\right),\left(p_{62},\tau_3^1\right),\left(p_{20},\tau_4^1\right)\right\},\\ \left\{\left(\emptyset,\tau_1^1\right),\left(p_{0:}^{t*},\tau_5^1\right),\left(p_{:6}^{t*},\tau_2^1\right),\left(p_{62},\tau_3^1\right),\left(p_{20},\tau_4^1\right)\right\},\\ \left\{\left(\emptyset,\tau_1^1\right),\left(p_{05}^{t*},\tau_5^1\right),\left(p_{56}^{t*},\tau_2^1\right),\left(p_{62},\tau_3^1\right),\left(p_{20},\tau_4^1\right)\right\},\\ \left\{\left(\emptyset,\tau_1^1\right),\left(p_{07}^{t*},\tau_5^1\right),\left(p_{76}^{t*},\tau_2^1\right),\left(p_{62},\tau_3^1\right),\left(p_{20},\tau_4^1\right)\right\},\\ \left\{\left(\emptyset,\tau_1^1\right),\left(p_{0:}^{t*},\tau_5^1\right),\left(p_{:6}^{t*},\tau_2^1\right),\left(p_{62},\tau_3^1\right),\left(p_{20},\tau_4^1\right)\right\},\\ \left\{\left(\emptyset,\tau_1^1\right),\left(p_{06},\tau_2^1\right),\left(p_{60}^{t*},\tau_6^1\right),\left(p_{02}^{t*},\tau_3^1\right),\left(p_{20},\tau_4^1\right)\right\},\\ \left\{\left(\emptyset,\tau_1^1\right),\left(p_{06},\tau_2^1\right),\left(p_{61}^{t*},\tau_6^1\right),\left(p_{12}^{t*},\tau_3^1\right),\left(p_{20},\tau_4^1\right)\right\},\\ \left\{\left(\emptyset,\tau_1^1\right),\left(p_{06},\tau_2^1\right),\left(p_{63}^{t*},\tau_6^1\right),\left(p_{32}^{t*},\tau_3^1\right),\left(p_{20},\tau_4^1\right)\right\},\\ \left\{\left(\emptyset,\tau_1^1\right),\left(p_{06},\tau_2^1\right),\left(p_{6:}^{t*},\tau_6^1\right),\left(p_{:2}^{t*},\tau_3^1\right),\left(p_{20},\tau_4^1\right)\right\},\\ \vdots\end{array}\right\}, \tag{30}$$

where the minimal time path is again used to correct any path consistency problems.

The final neighborhood is defined with a switch path mutation. This mutation switches the paths used to get between each visit location. An example of this set generated from x is given as

$$\left\{\begin{array}{l}\left\{\left(\emptyset,\tau_1^1\right),\left(p_{06}^{n},\tau_2^1\right),\left(p_{62},\tau_3^1\right),\left(p_{20},\tau_4^1\right)\right\},\\ \left\{\left(\emptyset,\tau_1^1\right),\left(p_{06},\tau_2^1\right),\left(p_{62}^{n},\tau_3^1\right),\left(p_{20},\tau_4^1\right)\right\},\\ \left\{\left(\emptyset,\tau_1^1\right),\left(p_{06},\tau_2^1\right),\left(p_{20}^{n},\tau_4^1\right)\right\}\end{array}\right\}, \tag{31}$$
$$\forall p_{06}^n \in \widehat{\mathcal{P}}_{06}, p_{62}^n \in \widehat{\mathcal{P}}_{62}, p_{20}^n \in \widehat{\mathcal{P}}_{20}.$$

The VND procedure is given in Algorithm 2. VND is given an initial plan to converge on $x_{init} \in \mathcal{X}$. The neighborhood index, i, is initialized to correlate with the change dwell time neighborhood on line 2. This neighborhood is repeatedly used to find a one step improvement on the current best found plan on line 5. When no improvement can be made with the current neighborhood, the next neighborhood is switched to on line 10. However, when an improvement is found in the current neighborhood, the neighborhood under consideration rolls back to the first dwell time neighborhood until no improvements can be made with it. The algorithm returns the plan found when one of two conditions occurs. First, no improvements are possible with any neighborhood. Second, the length of real time that VND has been executing exceeds a user defined maximum run time, $t_{max}^{vnd} \in \mathbb{R}_{>0}$. This feature is included to prevent VND from taking an excessively long time during any one generation of the metaheuristic algorithms described in Section 5.

Algorithm 2: Variable Neighborhood Descent (VND)

Input: x_{init} An initial solution
`// Best solution found so far`
1 $x_{best} \leftarrow x_{init}$;
2 $i \leftarrow 1$;
3 $t_{start} \leftarrow curTime()$;
4 **while** $i \leq 4$ **and** $curTime() - t_{start} \leq t_{max}^{vnd}$ **do**
`// Find min cost neighbor solution`
5 $x_{new} \leftarrow \arg\min_{x \in N_i(x_{best})} J(x)$;
6 **if** $J(x_{new}) < J(x_{best})$ **then**
`// Update best and start from first neighborhood`
7 $x_{best} \leftarrow x_{new}$;
8 $i \leftarrow 1$;
9 **else**
`// Switch to next neighbor`
10 $i \leftarrow i + 1$;
11 **return** x_{best};

5 Planners

This section describes the various metaheuristic planners considered herein. Each has advantages that may make them effective at solving the EVPRP. A HPSA algorithm is described in Section 5.1. The HPSA algorithm is the only planner proposed herein that does not make use of the Levenshtein edit-distance idea. It is considered herein as a baseline for comparison with the use of the Levenshtein edit-distance calculation in planning. The hybrid genetic path relinking algorithm (HGPRA) is described in Section 5.2, where the Levenshtein edit-scripts are used to define the PR procedure as a crossover operator. The HGPRA algorithm makes use of the Levenshtein edit-scripts but not the Levenshtein distance value. Section 5.3 describes a FA formulation that maintains more of the properties of the original FA than previous attempts at using FA for the VRP. This final planner makes use of both the Levenshtein edit-scripts and distance values when controlling how fireflies interact. These planners each use the Levenshtein distance calculation to varying extents, hence providing information about the usefulness of the strategy.

5.1 Hybrid Population Simulated Annealing Algorithm

In this section a HPSA algorithm is proposed. HPSA is used in this work to evaluate if the Levenshtein distance idea is advantageous while solving the EVPRP, as the other proposed planners use the Levenshtein distance and HPSA does not. It repeatedly runs the SA algorithm and VND on each element of a population of solutions. The SA procedure is used to perturb solutions from one basin of attraction to another, and VND is used to converge the solutions within one local basin. Before describing HPSA, the SA procedure is discussed.

SA was originally introduced in 1983 as a combinatorial optimization algorithm [32]. The method was inspired by the physical behavior of atoms in a metal as the metal cools. The process of annealing a metal begins by heating the metal to near melting so that the constituent atoms move about

Algorithm 3: Simulated Annealing (SA)

Input: x_{init} An initial solution

1 $T \leftarrow T_{max}$;
2 $x_{cur} \leftarrow x_{init}$;
3 $x_{best} \leftarrow x_{init}$;
4 $t_{start} \leftarrow curTime()$;
5 **while** $T \geq T_{min}$ **and** $curTime() - t_{start} \leq t^{sa}_{max}$ **do**
6 **foreach** $i \in n_g$ **do**
  // Select mutation operator
7 $m \xleftarrow{R,\Pi} \Theta$;
  // Perform mutation
8 $x^{+}_{cur} \leftarrow m(x_{cur})$;
9 **if** *Constraints valid in* x_{cur} **then**
  // Within a factor of the best cost
10 **if** $J(x^{+}_{cur}) \leq J(x_{best})\alpha_a$ **then**
  // Accepted if it improves cost or due to cooling schedule
11 **if** $J(x^{+}_{cur}) < J(x_{cur})$ **or** $e^{\frac{J(x_{cur})-J(x^{+}_{cur})}{T}} < \omega_{[0,1]}$ **then**
  // Update current solution
12 $x_{cur} \leftarrow x^{+}_{cur}$;
13 **if** $J(x^{+}_{cur}) < J(x_{best})$ **then**
  // Update best solution
14 $x_{best} \leftarrow x^{+}_{cur}$;
  // Update temperature
15 $T \leftarrow T\alpha_T$;
16 **return** x_{best};

randomly. The metal is then slowly cooled, allowing the atoms of metal time to settle into the relatively low energy state of a crystal lattice.

Since its introduction, SA has been applied to various problems and has been augmented many times, e.g., [11]. The SA procedure used in this work simulates randomness during cooling with the mutation operators described in Section 4.4. Furthermore, a solution acceptance factor $\alpha_a \in \mathbb{R}_{\geq 1}$ is incorporated such that the randomly perturbed solutions produced are only excepted if they have a cost that is within α_a of the best cost found so far, as suggested in [11].

Algorithm 3 shows the SA algorithm. First, the current temperature, current solution, best solution found, and start time are initialized on lines 1 through 4, where $T_{max} \in \mathbb{R}_{>0}$ is a user set maximal temperature. The loop starting on line 5 performs mutations on x_{cur} while cooling the temperature. After the temperature has cooled passed $T_{min} \in \mathbb{R}_{[0,T_{max})}$ or the running time of the procedure is greater than the maximum time allowed for SA, $t^{sa}_{max} \in \mathbb{R}_{>0}$, the best solution found is returned on line 16. The maximum time for SA to run is an unusual addition to the algorithm that is included to ensure that SA runs quickly and the overall HPSA algorithm can get through many generations in a reasonable time. The temperature cooling schedule is controlled in two ways. First, the for loop on line 6 makes it so that the temperature is only updated once after every group number, $n_g \in \mathbb{Z}_{>0}$, number of mutations has been performed. Second, the temperature is reduced by a factor of $\alpha_T \in \mathbb{R}_{(0,1)}$ on line 15.

This SA algorithm makes use of the full set of mutation operators Θ, described in Section 4.4. Line 7 chooses a mutation operator to apply to x_{cur}, where each element of Θ has a probability of being selected proportional to the complementary element of Π. After applying the operator to the current solution and verifying that the problem constraints are satisfied on lines 8 and 9, the conditions for updating x_{cur} and x_{best} are evaluated.

Two conditions must be satisfied for x_{cur} to be updated. First, line 10 requires that the new x_{cur} be within a factor of $\alpha_a \in \mathbb{R}_{\geq 1}$ of the best cost found. This ensures that x_{cur} never mutates in a way where a significant amount of optimality is lost. The second condition, on line 11, can be satisfied if the cost of x_{cur} will be improved by accepting the current mutation or if the reduction in optimality satisfies

$$e^{\frac{J(x_{cur})-J\left(x^{+}_{cur}\right)}{T}} < \omega_{[0,1]}. \tag{32}$$

This condition has a high probability of being satisfied when the temperature, T, is large, resulting in x_{cur} taking larger steps in random directions, even when those steps result in a loss of optimality. This allows x_{cur} to search out different local minima. As the temperature decreases, this condition has a lower probability of being satisfied, which causes x_{cur} to only take steps in directions that improve its cost. If both of these conditions are met, x_{cur} is updated with its mutated version on line 12.

The only condition for updating the best solution found so far, which is in the population set, is that the cost of x^{+}_{cur} is an improvement on the cost of x_{best}. If this is the case, then x_{best} is updated with x^{+}_{cur} on line 14.

The HPSA algorithm used herein is similar to that of [19]. It repeatedly uses the SA heuristic on each element of the population to find new basins of optimality in the search space. After the SA procedure is complete, the VND method is used to find the optimal solution within the basin of attraction in which each element currently resides. At this point, a selection method chooses an elite subset of the population to survive into the next generation. The rest of the population is replaced with randomly generated solutions before the process is repeated.

Algorithm 4 shows the outline of the HPSA algorithm used in this work. When the algorithm begins, an initial population, $P \subset \mathcal{X}$, is provided, generated in the manner described in Section 4.2. The loop on lines 1 through 8 iterates on this initial population until a user defined stopping condition is met. Line 3 runs the SA algorithm on each element of the population to find new basins of attraction. The VND described in Section 4.5 is performed on the result of SA on line 4. This intensifies the solution with a greedy local search. Note that x_i is still held in the population set, P, so line 4 updates that element of the population.

Once SA and VND have been evaluated for each member of the current population, the next population is generated on lines 5 through 7. First, P is reduced to an $n_{elite} \in \mathbb{R}_{[0,n_P]}$

Algorithm 4: Hybrid Population SA (HPSA)

Input: P initial population
1 **repeat**
2 **forall** $x_i \in P$ **do**
   // Run SA
3 $x_i \leftarrow SA(x_i)$;
   // Perform local search
4 $x_i \leftarrow VND(x_i)$;
   // Pick the elite samples
5 $P \leftarrow selection(P, n_{elite})$;
   // Migrate in random plans
6 **while** $|P| < n_P$ **do**
7 $P \leftarrow P \cup \{RAP()\}$;
8 **until** *Stopping condition met*;
9 **return** $\arg\min_{x \in P} J(x)$;

number of elite samples using one of the selection methods described in Section 4.3. This ensures that any progress made on finding good solutions is not lost. Then random plans are added to P, generated with the RAP, until it has n_P elements. This migration method ensures population diversity.

5.2 Hybrid Genetic Path Relinking Algorithm

In this section, a planner is presented that differs from the HPSA in that it takes advantage of the Levenshtein edit-script calculation. A HGA is developed that uses the well known PR procedure as a crossover operator and the VND described in Section 4.5 as a greedy local search to accelerate convergence. PR is a greedy optimization heuristic that has shown promising results in solving the VRP [11]. PR explores the possible solutions that exist between two given solutions and finds the point along the line between the two solutions that produces a minimal objective value.

The edit-scripts or optimal alignments described in Section 4.1 are used to define the line between two solutions. Starting from one solution, the line that connects it to another solution is generated incrementally by applying one edit-operation after another. For example, if $x_i, x_j \in \mathcal{X}$ are two solutions, then $x_i \odot_{\{1\}} es(x_i, x_j)$ is a "point" along the line that connects those solutions. PR evaluates the objective of each of these points and produces the point with the lowest objective value along the line. To remove any bias produced by favoring certain combinations of edit-operations, the operations being applied are chosen in a random order. This has the added bonus of producing different "lines" between solutions each generation, preventing redundant calculations from generation to generation.

The PR process is shown in Algorithm 5. Lines 1 through 3 initialize the set that contains the solutions along the line connecting the inputs, $L \subset \mathcal{X}$, the set of remaining edit-script indices, $Z_{rem} \subset \mathbb{Z}_{>0}$, and the set of indexes that have been used, $Z_{use} \subset \mathbb{Z}_{>0}$. The loop on lines 4 through 10 fills L with candidate solutions. A random element of Z_{rem} is selected on line 5. After the selected index has been removed from Z_{rem} and included in Z_{use}, line 8 applies the edit-operations indexed by Z_{use} to x_i to produce a new candidate solution. As long as

Algorithm 5: Path Relinking (PR)

Input: x_i, x_j Solutions
   // Find the line connecting the solutions
1 $L \leftarrow \emptyset$;
2 $Z_{rem} \leftarrow [1, |es(x_i, x_j)|]$;
3 $Z_{use} \leftarrow \emptyset$;
4 **while** $|Z_{rem}| > 1$ **do** // Unique points remain
   // Select new edit-operation
5 $n \xleftarrow{R} Z_{rem}$;
6 $Z_{rem} \leftarrow Z_{rem} \setminus \{n\}$;
7 $Z_{use} \leftarrow Z_{use} \cup \{n\}$;
   // Produce the next point
8 $x_{new} \leftarrow x_i \odot_{Z_{use}} es(x_i, x_j)$;
9 **if** *Constraints valid in* x_{new} **then**
10 $L \leftarrow L \cup \{x_{new}\}$;
   // Find optimal solution along line
11 **return** $\arg\min_{x \in L} J(x)$;

Algorithm 6: Hybrid Genetic PR Algorithm (HGPRA)

Input: P initial population
1 **repeat**
   // Initialize offspring set
2 $P_{new} \leftarrow \emptyset$;
   // Loop through population
3 **forall** $i \in [1, 2, \ldots, n_P]$ **do**
   // Perform crossover
4 **forall** $j \in [i+1, i+2, \ldots, n_P]$ **do**
5 $P_{new} \leftarrow P_{new} \cup \{PR(x_i, x_j)\}$;
   // Perform local search
6 **if** $\pi_{vnd} > \omega_{[0,1]}$ **then**
7 $P_{new} \leftarrow P_{new} \cup \{VND(x_i)\}$;
   // Perform mutations
8 **forall** $m_j \in \Theta$ **do**
   // Pick mutations to perform
9 **if** $\pi_j > \omega_{[0,1]}$ **then**
10 $x_{mut} \leftarrow m_j(x_i)$;
11 **if** *Constraints valid in* x_{mut} **then**
12 $P_{new} \leftarrow P_{new} \cup \{x_{mut}\}$;
   // Pick the next generation
13 $P \leftarrow selection(P \cup P_{new}, n_P)$;
14 **until** *Stopping condition met*;
15 **return** $\arg\min_{x \in P} J(x)$;

the candidate solution satisfies the constraints of the problem, it is included in L on line 10. This loop ends when there is still one edit-operation in $es(x_i, x_j)$ which has not been used because the application of all edit-operations would produce x_j which is not a new solution for consideration. After L has been populated, line 11 returns the objective optimal element.

PR is used as a crossover operator in the proposed HGPRA. The HGPRA is given in Algorithm 6. The algorithm is given an initial population to start. The loop on lines 1 through 14 iterates through generations of this initial population until a

stopping condition is met; at which point the best solution found in the population is returned on line 15. The loop starting on line 3 loops over the current population to produce offspring that are stored in the set $P_{new} \subset \mathcal{X}$.

Lines 4 and 5 perform the PR operator on each unique pair of solutions in P, storing the result in P_{new}. The offspring it produces will have desirable characteristics from both parent solutions, melding the successful aspects of both into a better solution. Lines 6 and 7 perform VND on each element of the population with a given probability $\pi_{vnd} \in \mathbb{R}_{[0,1]}$. The presence of this asexual reproduction via the VND procedure hybridizes the search and intensifies its ability to find high quality solutions at the bottom of local basins in the objective function.

Lines 8 through 12 apply the mutation operators described in Section 4.4 to each element of the population with a given probability. If the probability is evaluated as true, then the mutation is performed on line 10. The resulting solution is included in P_{new} on line 12 as long as the constraints of the problem are satisfied. These mutated solutions are included to encourage population diversity and help the algorithm break out of local minima.

On line 13, the *selection* procedure described in Section 4.3 is used to select the next population. The *selection* procedure is given the previous population and all the solutions produced by the PR, VND, and mutation operators applied above. From that set, n_P elements are selected to make up the next population.

5.3 Hybrid Firefly Algorithm

In this section, an algorithm is developed that not only uses the Levenshtein edit-script calculation but also the resulting inner solution distance metric. Specifically, a variant of the FA is developed, which is hybridized with the VND procedure. Yang [12] proposed the FA for continuously-valued, multimodal optimization applications. Yang shows that the particle swarm optimization (PSO) algorithm has characteristics similar to those of the FA, but the FA is more adaptable to different problems. Unlike PSO, in FA each element of the population is attracted to all other elements with better fitness instead of just the global optimal and the historic best of the particle. The FA is a nature-inspired algorithm that emulates the flashing of fireflies or population members and the effect that those flashes have on surrounding fireflies when searching for a mate. FA idealizes this attraction to flashing with five assumptions:

1) Fireflies can be attracted to all other fireflies.
2) Firefly trajectories have a component of randomness.
3) Fitter fireflies are brighter than less fit ones.
4) The attraction of one firefly to another is determined by the visible light intensity of the latter as observed by the former. This brightness decreases as distance grows; thus, distant fireflies are not attracted.
5) Each firefly moves toward all other fireflies that are brighter (or fitter) than itself. That is, the brightest firefly will move randomly and all others will move toward it to some extent.

To show how these idealisms produce the FA, the original FA is described below [12]. Let $d : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_{\geq 0}$ denote the distance function between two fireflies. In [12] the search space is Euclidean, i.e., $\mathcal{X} = \mathbb{R}^n$ for some $n \in \mathbb{Z}_{>0}$. Accordingly, the distance between two fireflies $x_i, x_j \in \mathcal{X}$ is defined with the induced norm, i.e., $d(x_i, x_j) = \|x_i - x_j\|$. Attractiveness is relative to light intensity, and light intensity decreases with distance following the inverse square law and the light absorption of the air. As such, the effects of the inverse square law and light absorption are approximated via the coefficient

$$\beta(d) \triangleq \beta_0 e^{-\kappa d^2}, \tag{33}$$

where $\beta : \mathbb{R}_{\geq 0} \mapsto \mathcal{R}_{(0,1]}$ is the function that calculates the attractiveness coefficient at a given distance d and $\kappa \in \mathbb{R}_{\geq 0}$ is the light absorption coefficient for the "medium" the light is passing through. Finally, the movement of firefly x_i toward a brighter firefly x_j is determined by

$$x_i^+ \leftarrow x_i + \beta\big(d(x_i, x_j)\big)(x_j - x_i) + \alpha_\omega \omega_{[-1,1]}, \tag{34}$$

where x_i denotes the position of firefly i before movement and x_i^+ denotes its position after movement. Assumption 2, that all fireflies move randomly, is satisfied with the term $\alpha_\omega \omega_{[-1,1]}$, where $\alpha_\omega \in \mathbb{R}_{\geq 0}$ controls the magnitude of random perturbations.

Algorithm 7: Original Firefly Algorithm

Input: P initial population
1 **repeat**
2 **forall** $x_i \in P$ **do**
3 **forall** $x_j \in P$ **do**
4 **if** $J(x_i) > J(x_j)$ **then**
5 $x_i \leftarrow x_i + \beta(d(x_i, x_j))(x_j - x_i) + \alpha_\omega \omega_{[-1,1]}$;
6 **until** *Stopping condition met*;
7 **return** $\arg\min_{x \in P} J(x)$;

The FA can be seen in Algorithm 7. The algorithm is given an initial population. Line 4 determines whether x_i is brighter than x_j, and if it is line 5 moves x_i in the direction of x_j. Note that line 5 updates the value of x_i in the population $P \in \mathcal{X}$ directly.

5.3.1 Other Discrete Firefly Variants

Several variants of FA have been proposed for solving discrete-valued problem domains, e.g., in [13], [14], [23] FA is used to solve the CVRP. They define the distance between fireflies with the Hamming distance (HD). The HD measures the dissimilarity between two strings of equal length by the number of substitutions required to transform one string into the other. This method works for the CVRP because each solution has to visit each customer location once and only once, which means that each solution has the same length. However, it is unusable for the PRP because solutions have differing numbers of visits. This is why the Levenshtein distance is used in this work; see Section 4.1.

In [23], fireflies have no direction of movement; instead, a random number, bounded by the distance between fireflies, of visits are moved to random locations within the solution. The lowest cost solution produced from these visit moves is used as the new firefly. This way of using FA in a discrete domain

completely ignores Assumptions 1 and 5 because no notion of direction is considered during movement. This means that some of the advantages of FA are lost.

In [13], [14], they use the 2-opt and 2-h-opt procedures to hybridize the FA. Randomness in the movements of each firefly is produced using two variants of the Visit Swap mutation; see Mutation Definition 7. Additionally, these works redefine the condition on line 4 of Algorithm 7 with

$$\omega_{[0,HD(x_i,x_b)]} > \omega_{[0,HD(x_j,x_b)]}, \tag{35}$$

where x_b is the best firefly found so far. This violates Assumptions 3 and 4 because the attraction of one firefly to another is based on their distance from the best firefly and not their individual fitness. When (35) is satisfied, x_i is replaced with a solution produced by performing the PMX operator between x_i and x_b. This violates Assumption 4 because all fireflies move the same amount regardless of the distance between the fireflies. Furthermore, Assumption 1 is violated because fireflies only move toward the best firefly found instead of all brighter fireflies. This makes this variant of FA more similar to PSO than the original FA.

As [14] discusses, this algorithm tends to get stuck in local minima. They attribute this to hybridization; however, the authors of this work suspect that it is because they steer all solutions toward the global optimal instead of each pairwise optimal, as is done in the original FA. Additionally, the distance that fireflies move is the same for all inter-firefly distances, which causes all fireflies to get stuck in the same local minima. This work takes a different approach to developing a discrete-valued FA that maintains more of the characteristics of the original algorithm.

5.3.2 Proposed Hybrid Firefly Algorithm

Due to the symbolic solution space used herein, three major changes have to be made to the original FA for it to be used in this work: a new way to introduce randomization in the trajectories of the fireflies, a new distance function, and a new firefly movement procedure. Algorithm 8 shows the proposed modifications. In many ways, this algorithm is similar to the HGPRA proposed in Section 5.2 except that a version of the firefly operator is used as a crossover mechanism instead of the PR operator. The only differences between Algorithms 6 and 8 lie in lines 4 through 8 of Algorithm 8. The first difference is on line 4 where x_j is iterated over the whole population instead of only the unique pairs. This is done so that the firefly attraction condition on line 5 is evaluated for all combinations of x_i and x_j.

The attractiveness coefficient function β used on line 6 is modified to provide more flexibility than the original given in (33). This work uses the definition

$$\beta(d) \triangleq \beta_0 e^{-\kappa d^{\xi}}, \tag{36}$$

where instead of the distance being to the power of 2 it is raised to the power of $\xi \in \mathbb{R}_{>0}$. The distance calculation is performed using the Levenshtein distance described in Section 4.1. Additionally, the Levenshtein edit-script calculation is used when defining the movement procedure. Algorithm 9 describes this process. In general terms, it selects edit-operations from the edit-script that connects x_i to x_j until the number of selected edit-operations has greater summed weight than the attractiveness coefficient. Lines 1 though 3 initialize the newly produced solution, $x_{new} \in \mathcal{X}$, the set of indexes of the edit-operations that have not been used, $Z_{rem} \subset \mathbb{Z}_{>0}$, and the summed weight of all edit-operations that have been used, $w_{use} \in \mathbb{R}_{>0}$. The loop starting on line 4 loops until the summed weight of the applied edit-operations is greater than the attractiveness coefficient. Line 5 selects a random edit-operation to perform. After removing the selected index from Z_{rem}, the weight of the selected edit-operation is added to w_{use} on line 7. This additional weight will be W_i if the selected operation is an Insertion, W_d if it is a deletion, etc.

Algorithm 8: Hybrid Firefly Algorithm

Input: P initial population
1 **repeat**
  `// Initialize offspring set`
2 $P_{new} \leftarrow \emptyset$;
  `// Loop through population`
3 **forall** $x_i \in P$ **do**
    `// Perform crossover`
4 **forall** $x_j \in P$ **do**
5 **if** $J(x_i) > J(x_j)$ **then**
6 $x_{new} \leftarrow movement(x_i, x_j, \beta(d(x_i, x_j)))$;
7 **if** *Constraints valid in* x_{new} **then**
8 $P_{new} \leftarrow P_{new} \cup \{x_{new}\}$;
    `// Perform local search`
9 **if** $\pi_{vnd} > \omega_{[0,1]}$ **then**
10 $P_{new} \leftarrow P_{new} \cup \{VND(x_i)\}$;
    `// Perform mutations`
11 **forall** $m_j \in \Theta$ **do**
12 **if** $\pi_j > \omega_{[0,1]}$ **then**
13 $x_{mut} \leftarrow m_j(x_i)$;
14 **if** *Constraints valid in* x_{mut} **then**
15 $P_{new} \leftarrow P_{new} \cup \{x_{mut}\}$;
    `// Pick the next generation`
16 $P \leftarrow selection(P \cup P_{new}, n_P)$;
17 **until** *Stopping condition met*;
18 **return** $\arg\min_{x \in P} J(x)$;

Algorithm 9: movement

Input: x_i, x_j Solutions
Input: β Attractiveness coefficient
1 $x_{new} \leftarrow x_i$;
2 $Z_{rem} \leftarrow [1, |es(x_i, x_j)|]$;
3 $w_{use} \leftarrow 0$;
4 **while** $\beta > w_{use}$ **do** `// Distance not covered`
  `// Select new edit-operation`
5 $n \xleftarrow{R} Z_{rem}$;
6 $Z_{rem} \leftarrow Z_{rem} \setminus \{n\}$;
7 $w_{use} \leftarrow w_{use} + esw_n(x_i, x_j)$;
  `// Apply edit-operation`
8 $x_{new} \leftarrow x_{new} \odot_{\{n\}} es(x_i, x_j)$;
9 **return** x_{new};

Lines 8 apply the selected edit-operation to x_{new}, which is returned when the function is finished.

After the movement algorithm has been performed, line 8 of Algorithm 8 adds the newly created offspring to P_{new} as long as the constraints of the problem are satisfied in the offspring. Note that, unlike the original FA, this version of the movement does not overwrite the preexisting x_i but instead adds it to a set of offspring P_{new}.

6 Simulation

In this section, the various algorithms developed herein are tested on their ability to produce high quality solutions for the EVPRP. The street graphs used are produced by selecting the dispatch vertex location to be the actual police precinct location of a city. Then the Open Street Map database [33] is queried for the actual streets within a certain distance of the precinct. Open Street Maps provides up-to-date and accurate street directions, i.e., one-way or two-way, as well as speed limits and all other information necessary while generating testing scenarios. Actual street layouts are used to ensure that the conclusions drawn herein are realistic. See Appendix C.1 for a full description of how the street parameters are generated. Hotspot, $\mathcal{H}$, and representative emergency, $\mathcal{V}_e$, location sets are randomly chosen with a uniform distribution over the city graph. See Appendix C.2 for a description of the hotspot model parameters.

Each scenario described below simulates the area covered by one police precinct. In all of them, each agent has a 12 hour shift duration. There is always a fixed number of agents on duty at a given moment, but their shift start and end times are chosen such that no two agents end their shift at the same time. For example, if there are 4 agents on duty throughout the day, then every 3 hours one agent will end their shift and another will start their shift simultaneously. Each scenario has a $\Delta T \equiv 48$ hour planning window to allow the hotspot EDRs and patrolling agent routes to reach a steady state balance before the end of the window. A 2025 Chevrolet Blazer EV police patrol vehicle model is used for all agents. This EV has been designed for the patrolling duties of law enforcement officers. It has a maximum battery capacity of $\overline{q}_k \equiv 85$ kWh, giving it a maximum range of $\overline{d}_k \equiv 455$ km, and its maximum charge rate is $r_k \equiv 190$ kW [34].

The first scenario is set in the streets of Los Angeles, California, and is shown in Figure 3. This precinct covers a 7 km square of Los Angeles, containing 8617 streets and 3138 intersections. Four agents are continuously on duty throughout the day, which results in a total of $n_a \equiv 19$ agent shifts to plan. There are $n_h \equiv 25$ hotspots and $|\mathcal{V}_e| \equiv 10$ representative emergency locations. This scenario is a small but representative problem that can be solved quickly while parameter tuning.

The next scenario is a medium sized problem covering a 9 km square of Seattle, Washington, see Figure 4. It contains the largest number of streets at 15012 in total, with 5696 intersections. Ten agents are continuously on duty, resulting in $n_a \equiv 49$ agent shifts in total. These agents monitor $n_h \equiv 35$ hotspots and $|\mathcal{V}_e| \equiv 25$ emergency locations. This precinct has the densest grid of streets of all the scenarios considered in this work.

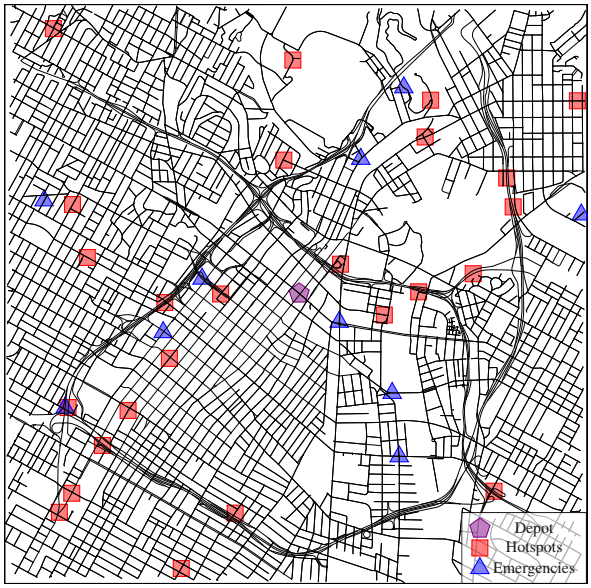


Fig. 3: The Los Angeles precinct street graph.

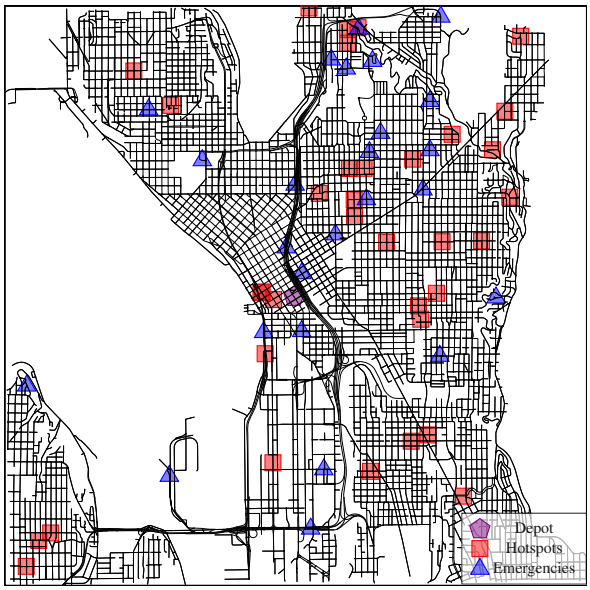


Fig. 4: The Seattle precinct street graph.

The next and largest scenario is modeled after New York City, New York, and is shown in Figure 5. This precinct encompasses a 30 km square of area containing 12867 streets and 5407 intersections. $n_h \equiv 50$ hotspots and $|\mathcal{V}_e| \equiv 30$ emergency locations are scattered throughout the region. With so many hotspots to cover, 16 agents are continuously needed to monitor the hotspots, which means that there are a total of $n_a \equiv 79$ agent shifts. This scenario tests the planners where there are geographically distinct locations to monitor. Note in Figure 5 that the graph contains two dense clusters of streets separated by long roads that cross a park. This scenario puts the battery longevity of the patrolling vehicles to the test.

The weights of the objective function are chosen as $\alpha_{edr,2} \equiv$

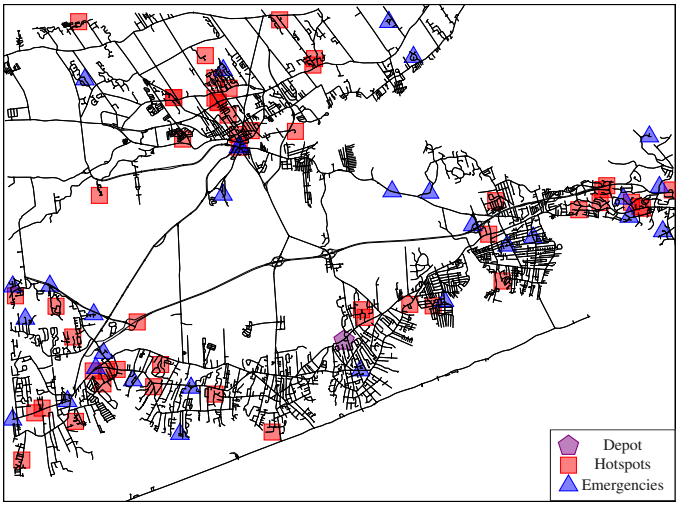


Fig. 5: The New York precinct street graph.

TABLE II: Comparison of initialization methods.

		Cost		Run Time (milisec)	
		Mean	SD	Mean	SD
Los Angeles with SOC Constraints	RAP	190.7	1.5	1.6	2.1
	GAP	148.4	0.0	1,031.7	14.6
	FAP	187.2	0.0	3.9	2.8
Seattle with SOC Constraints	RAP	254.2	2.1	3.8	1.8
	GAP	190.8	0.0	25,562.6	78.8
	FAP	325.3	0.0	7.4	2.9
New York with SOC Constraints	RAP	415.7	3.5	6.4	2.3
	GAP	321.6	0.0	97,394.1	317.3
	FAP	499.6	0.0	12.7	1.5

$1e^{-3}$, $\alpha_{edr,\infty} \equiv 10$, $\alpha_{rt,2} \equiv 1e^{-6}$, and $\alpha_{rt,\infty} \equiv 1e^{-2}$. They are chosen such that each metric makes a noticeable difference in the total objective function while making the two-norms of each more important than the infinity-norms. The weights for the calculation of the Levenshtein distance are set to $w_v \equiv 4096$, $w_p \equiv 2048$, and $w_{dt} \equiv 1024$. This means that changing the vertex being visited is weighted twice as heavily as changing the path being used to get there, and changing the path is weighted twice as heavily as changing the length of time spent dwelling at the visited vertex.

The hyperparameters for each planner, that is, the parameters that control how the algorithms operate, have been optimized via a Bayesian hyperparameter optimization algorithm provided by the Ax library [35]. The Los Angeles scenario is used with a maximal solve time of 1 minute while tuning the planner hyperparameter. The objective to minimize is the summed cost of the best solution each planner managed to produce over three runs. Each planner was given over 300 trials to tune. The results of the hyperparameter tuning are given in Appendix B.

Before discussing the results of the primary planners, it is worth studying the plans produced by the initialization methods. Table II provides statistics on the objective value of the plans produced by the initialization methods, as well as the time it took to generate those plans. All results in this work were collected on an AMD Ryzen™ Threadripper™ 3970X processor over 30 individual runs. The planning code can be found in our open-source repository https://github.com/james-swedeen/electric_vehicle_patrol_route_planning. As expected, the standard deviations (SDs) of the costs produced by the GAP and FAP are zero because they are deterministic planners. GAP consistently produces the lowest cost plans. However, GAP also takes up to 7669 times longer to produce a solution than GAP. FAP produces better plans than RAP, on average, in the Los Angeles scenario. However, FAP produces significantly worse plans than the average from the RAP in the other two scenarios. Considering that the planner hyperparameters are tuned to the Los Angeles scenario, this suggests that the hyperparameters of FAP must be custom tuned for each situation FAP is used in, or the optimality of the resulting plan suffers significantly.

Figure 6 shows the transient convergence of the best solution found by each of the primary planners. The solid lines are the costs of the best solution found at that time, averaged across all simulations. The dashed lines show the 3-SD regions of the averages, i.e., approximately 99.7% of the runs find a solution cost that lies within the regions.

The top two plots contain the transients of the planners within the Los Angeles scenario when SOC and distance-based constraints are used. As can be seen, the planners produce higher quality solutions when using the full SOC-based constraints. This is expected because the distance-based constraints overconstrain the problem, even when regenerative braking is not factored in. With the considerations of regenerative braking, which the distance-based constraints have no means of considering, the SOC-based constraints allow for plans that are much more efficient. This highlights the value of planning with a fully simulated battery and accompanying SOC-based constraints.

Looking at the results of the individual planners across all of the scenarios, a few trends can be seen. The HGPRA planners using either selection method perform the best in all scenarios. Especially the HGPRA using KMS, which performs significantly better than the other planners in the Los Angeles scenarios and approximately ties with the HGPRA-FPS planner in the New York scenario. In the final scenario, the Seattle scenario, the HGPRA-FPS planner performs the best. This shows that PR is an extremely useful heuristic for the EVPRP, and the way this work implements it, using Levenshtein edit-scripts, is highly effective.

Furthermore, the use of the Levenshtein edit-distance in the KMS procedure seems to be advantageous in most cases. All of the HPSAs that are using KMS outperform the instances using FPS, and three out of four of the instances of HGPRA using KMS outperform the FPS counterparts. The HPSA-KMS planners consistently perform near as well as the HGPRA planners, though their convergence seems to plateau faster than the HGPRA counterparts. This suggests that, given more time to plan, the HGPRA planners would find better solutions than the HPSA planners.

While the HPSA planners using KMS are middle of the pack in performance, those using FPS are among the worst performing planners across all scenarios. Conversely, while the

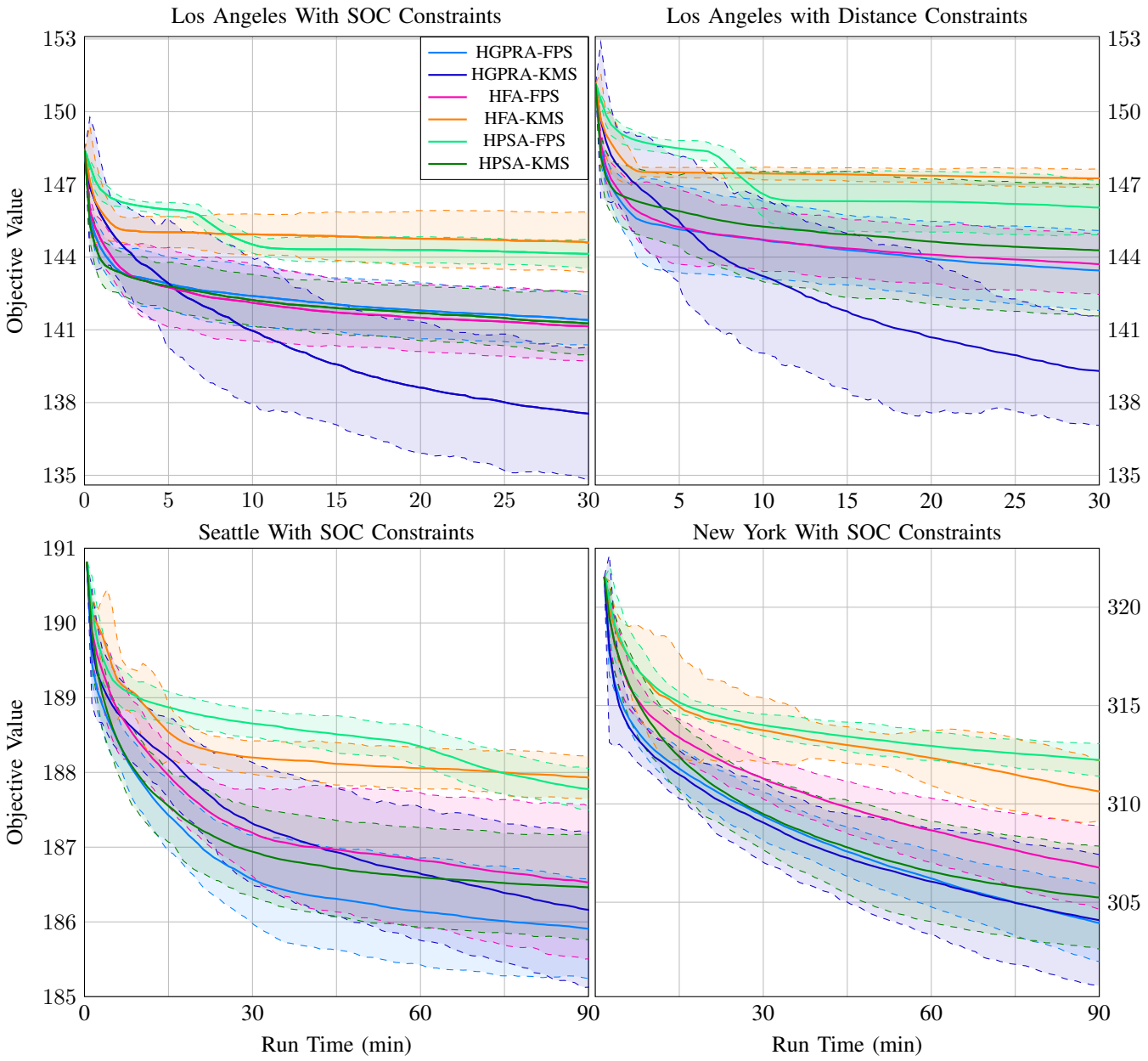


Fig. 6: Convergence plots averaged over 30 benchmarks. Solid lines are the average best solution cost found. Dashed lines are the 3-SD lines of the best solution found across the 30 benchmarks.

hybrid firefly algorithm (HFA) planners using FPS perform about as well as the HPSA-KMS planners, the HFA-KMS planners perform poorly in all scenarios. In fact, the lower cutoffs of the 3-SD regions of HPSA-FPS and HFA-KMS are still less optimized than the upper 3-SD regions of the other planners. Hence, they not only usually perform worse, but also perform worse almost always. The reasons for these trends are not entirely clear. It is possible that, despite the best efforts of the authors, the hyperparameter tuning for these two planners produced poor results. As is well known, hyperparameter tuning is not a simply solved problem, and it is very easy to get stuck in local minima while searching for optimal parameters. Additionally, these algorithms can be very sensitive to correctly chosen hyperparameters. This can be seen in the results of the HFA-FPS planners that perform almost identically to the HPSA-KMS planners in the Los Angeles scenarios. However, in the Seattle and New York scenarios, the HPSA-KMS planner performs significantly better. This is because the hyperparameters of the algorithms were tuned in the Los Angeles scenario, and HFA is more sensitive to correctly chosen hyperparameters for the scenario being run than HPSA is.

Figure 7 shows the averaged objective value subcomponents of the best solutions found by each planner. The most noticeable trend is that RAP and FAP have long response times. This makes sense because RAP plans the routes to hotspots regardless of emergency locations and FAP does not consider the response time in its potential field. When

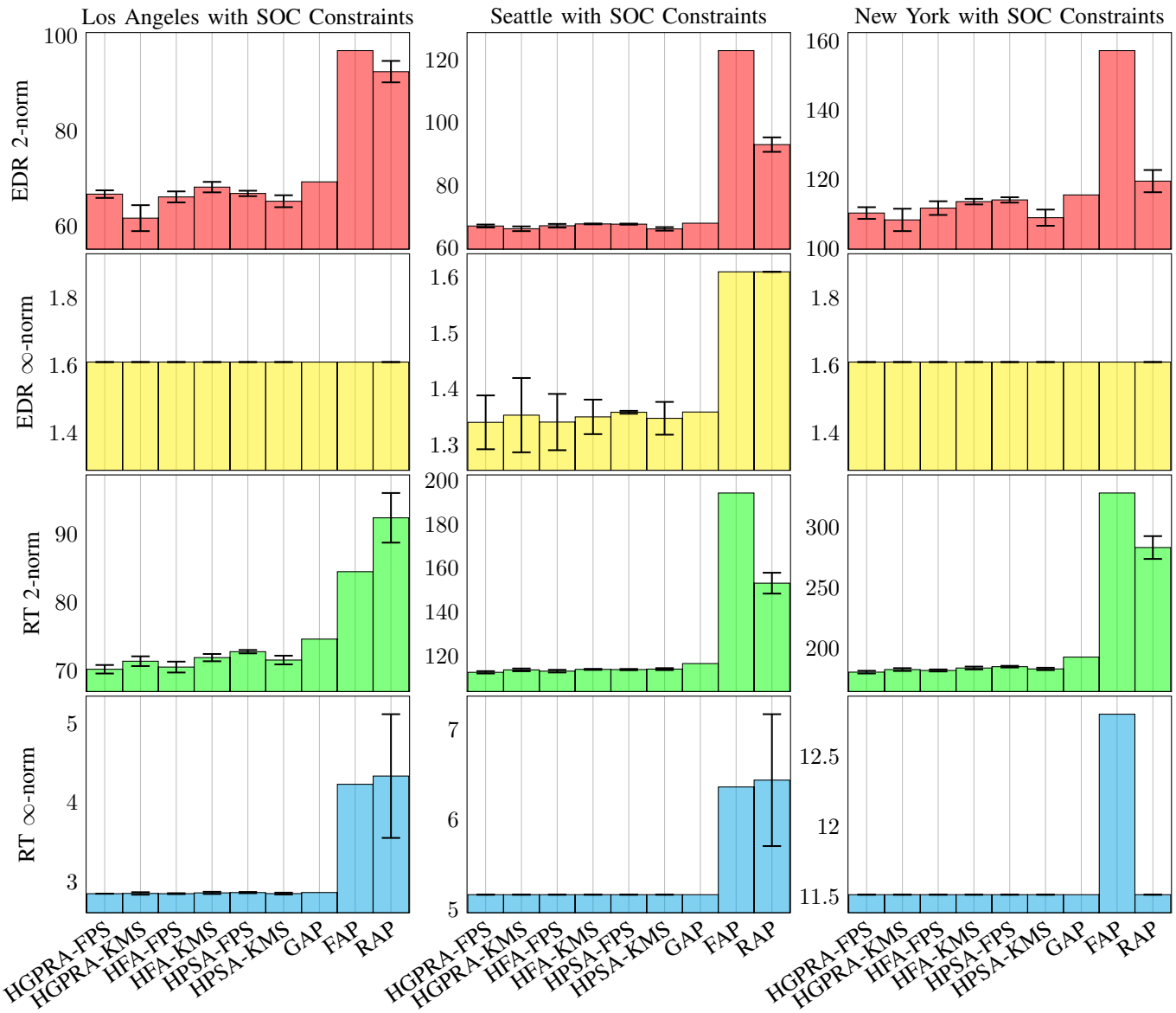


Fig. 7: Planning results broken into each sub-objective. Bars show the average, and error margins show the plus/minus 3-SD region of the best solution found.

comparing the high quality solution produced by GAP with the higher quality solutions produced by the six main planners, it is seen that the infinity-norms improve only marginally. Most of the performance improvement is achieved by reducing the 2-normed metrics. This suggests that the planners have a difficult time keeping the worst case EDR and response times controlled, instead focusing on maximizing the overall effectiveness across the precinct. This can be seen most readily in the EDR infinity-norm results of the Los Angeles and New York scenarios, where all planners produced the same results. This means that more patrolling agents are needed in order to cover those areas effectively.

7 Conclusion

In this work, the field of PRP is expanded to incorporate EV battery constraints. Furthermore, the derived EV battery constraints are more thorough than any that exist in the larger field of VRP because they include considerations of regenerative braking. A set of distance-based constraints is derived alongside the EV battery constraints and is shown to overconstrain the problem, resulting in suboptimal routing efficiency. The novel idea of using the Levenshtein edit-distance calculation to define the distances between solutions and step-by-step edit-operations to transform one solution into another is proposed. This edit-distance idea is used while defining three population-based metaheuristic algorithms: HPSA, HGPRA, and HFA. Additionally, the edit-distances are used while implementing a KMS procedure that encourages population diversity. Through comparisons are drawn between the proposed planners via averaged evaluations in scenarios that are generated from real world street graphs. It is found that the implementation of PR proposed herein, which uses the Levenshtein edit-scripts, is a

highly effective heuristic for solving the EVPRP.

Directions of future work include experimenting with different edit-distance formulations, e.g., the Damerau-Levenshtein edit-distance [10]. Additionally, the edit-operation weights, w_v, w_p, and w_{dt}, could be fine tuned to weight the operations that are more important to the objective value more heavily. A nonlinear charging model like the CC/CV model could be incorporated to increase realism [24]. As always, the hyperparameter tuning could be done more thoroughly, and the benchmark evaluations could be run for longer, given the time.

References

[1] P. J. Cook, "Research in criminal deterrence: Laying the groundwork for the second decade," *Crime and Justice*, vol. 2, pp. 211–268, 1980. [Online]. Available: https://doi.org/10.1086/449070

[2] M. Dewinter, C. Vandeviver, T. Vander Beken, and F. Witlox, "Analysing the police patrol routing problem: A review," *ISPRS International Journal of Geo-Information*, vol. 9, no. 3, 2020. [Online]. Available: https://www.mdpi.com/2220-9964/9/3/157

[3] J. R. Cooper, "Optimal multi-agent search and rescue using potential field theory," in *AIAA Scitech 2020 Forum*, 2020, p. 0879.

[4] B. B. Keskin, S. R. Li, D. Steil, and S. Spiller, "Analysis of an integrated maximum covering and patrol routing problem," *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 1, pp. 215–232, 2012.

[5] H. Chen, T. Cheng, and S. Wise, "Developing an online cooperative police patrol routing strategy," *Computers, Environment and Urban Systems*, vol. 62, pp. 19–29, 2017.

[6] X. Chen, "Fast patrol route planning in dynamic environments," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 42, no. 4, pp. 894–904, 2012.

[7] R. Dewil, P. Vansteenwegen, D. Cattrysse, and D. Van Oudheusden, "A minimum cost network flow model for the maximum covering and patrol routing problem," *European Journal of Operational Research*, vol. 247, no. 1, pp. 27–36, 2015.

[8] Y. Jiang, H. Li, B. Feng, Z. Wu, S. Zhao, and Z. Wang, "Street patrol routing optimization in smart city management based on genetic algorithm: a case in zhengzhou, china," *ISPRS International Journal of Geo-Information*, vol. 11, no. 3, p. 171, 2022.

[9] J. Chen, A. Baskaran, Z. Zhang, and P. Tokekar, "Multi-agent reinforcement learning for visibility-based persistent monitoring," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 2563–2570.

[10] L. Boytsov, "Indexing methods for approximate dictionary searching: Comparative analysis," *ACM J. Exp. Algorithmics*, vol. 16, may 2011. [Online]. Available: https://doi.org/10.1145/1963190.1963191

[11] N. Labadie, C. Prins, and C. Prodhon, *Metaheuristics for Vehicle Routing Problems*. Wiley, 2016. [Online]. Available: https://books.google.com/books?id=uv2OCwAAQBAJ

[12] X.-S. Yang, "Firefly algorithms for multimodal optimization," in *Stochastic Algorithms: Foundations and Applications*, O. Watanabe and T. Zeugmann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 169–178.

[13] A. M. Altabeeb, A. M. Mohsen, and A. Ghallab, "An improved hybrid firefly algorithm for capacitated vehicle routing problem," *Applied Soft Computing*, vol. 84, p. 105728, 2019.

[14] A. M. Altabeeb, A. M. Mohsen, L. Abualigah, and A. Ghallab, "Solving capacitated vehicle routing problem using cooperative firefly algorithm," *Applied Soft Computing*, vol. 108, p. 107403, 2021.

[15] R. Paradiso, R. Roberti, D. Laganá, and W. Dullaert, "An exact solution framework for multitrip vehicle-routing problems with time windows," *Operations Research*, vol. 68, no. 1, pp. 180–198, 2020.

[16] B. Pan, Z. Zhang, and A. Lim, "Multi-trip time-dependent vehicle routing problem with time windows," *European Journal of Operational Research*, vol. 291, no. 1, pp. 218–231, 2021.

[17] R. Zhang, J. Guo, and J. Wang, "A time-dependent electric vehicle routing problem with congestion tolls," *IEEE Transactions on Engineering Management*, vol. 69, no. 4, pp. 861–873, 2022.

[18] J.-P. Riquelme-Rodríguez, A. Langevin, and M. Gamache, "Adaptive large neighborhood search for the periodic capacitated arc routing problem with inventory constraints," *Networks*, vol. 64, no. 2, pp. 125–139, 2014.

[19] İ. İlhan, "An improved simulated annealing algorithm with crossover operator for capacitated vehicle routing problem," *Swarm and Evolutionary Computation*, vol. 64, p. 100911, 2021.

[20] Z. H. Ahmed, A. S. Hameed, M. L. Mutar *et al.*, "Hybrid genetic algorithms for the asymmetric distance-constrained vehicle routing problem," *Mathematical Problems in Engineering*, vol. 2022, 2022.

[21] K. Alabdulkareem and Z. H. Ahmed, "Comparison of four genetic crossover operators for solving distance-constrained vehicle routing problem," *IJCSNS International Journal of Computer Science and Network Security*, vol. 20, no. 7, pp. 114–123, 2020.

[22] A. Chehouri, R. Younes, J. Khoder, J. Perron, and A. Ilinca, "A selection process for genetic algorithm using clustering analysis," *Algorithms*, vol. 10, no. 4, 2017. [Online]. Available: https://www.mdpi.com/1999-4893/10/4/123

[23] E. Osaba, X.-S. Yang, F. Diaz, E. Onieva, A. D. Masegosa, and A. Perallos, "A discrete firefly algorithm to solve a rich vehicle routing problem modelling a newspaper distribution system with recycling policy," *Soft Computing*, vol. 21, pp. 5295–5308, 2017.

[24] H. Wu, G. K. H. Pang, and X. Li, "A realistic and non-linear charging process model for parking lot's decision on electric vehicles recharging schedule." Institute of Electrical and Electronics Engineers Inc., 6 2020, pp. 2–7.

[25] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec 1959. [Online]. Available: https://doi.org/10.1007/BF01386390

[26] R. Bellman, "On a routing problem," *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[27] J. H. Holland, "Adaptation in natural and artificial systems," 1975.

[28] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE transactions on robotics*, vol. 25, no. 4, pp. 912–926, 2009.

[29] E. Schubert and P. J. Rousseeuw, "Fast and eager k-medoids clustering: O(k) runtime improvement of the pam, clara, and clarans algorithms," *Information Systems*, vol. 101, p. 101804, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437921000557

[30] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," Stanford, Tech. Rep., 2006.

[31] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia tools and applications*, vol. 80, pp. 8091–8126, 2021.

[32] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[33] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org ," https://www.openstreetmap.org, 2017.

[34] *Police General Motors Blazer EV PPV & SSV*, General Motors, 2024. [Online]. Available: https://www.gmenvolve.com/content/dam/gmenvolve/na/us/english/index/police/2024-blazer-ev-ppv/02-pdfs/2025-BLAZER-EV-PPV-Municipal-Specification-Guide.pdf#page=2.30

[35] S. G. Baird, M. Liu, and T. D. Sparks, "High-dimensional bayesian optimization of 23 hyperparameters over 100 iterations for an attention-based network to predict materials property: A case study on crabnet using ax platform and saasbo," *Computational Materials Science*, vol. 211, p. 111505, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0927025622002610

[36] G. Boeing, "Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks," *Computers, Environment and Urban Systems*, vol. 65, pp. 126–139, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0198971516303970

[37] C. Fiori, K. Ahn, and H. A. Rakha, "Power-based electric vehicle energy consumption model: Model development and validation," *Applied Energy*, vol. 168, pp. 257–268, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S030626191630085X

[38] C. S. Koper, "Just enough police presence: Reducing crime and disorderly behavior by optimizing patrol time in crime hot spots," *Justice quarterly*, vol. 12, no. 4, pp. 649–672, 1995.

Appendix A
Acronyms and Nomenclature

Acronyms

CE	cross entropy
CVRP	capacitated vehicle routing problem
DVRP	distance-constrained vehicle routing problem

EDR	expected disturbance rate
EV	electric vehicle
EVPRP	electric vehicle patrol routing problem
FA	firefly algorithm
FAP	potential-field accent planner
FPS	fitness proportional selection
GA	genetic algorithm
GAP	greedy action planner
GRBF	Gaussian radial basis function
HD	Hamming distance
HFA	hybrid firefly algorithm
HGA	hybrid genetic algorithm
HGPRA	hybrid genetic path relinking algorithm
HPSA	hybrid population-based simulated annealing
KMS	k-medoids selection
MILP	mixed integer linear program
MIP	mixed integer program
PMX	partially mapped crossover
PR	path relinking
PRP	patrol routing problem
PSO	particle swarm optimization
RAP	random action planner
SA	simulated annealing
SCX	sequential constructive crossover
SD	standard deviation
SGA	sequential greedy algorithm
SOC	state of charge
VND	variable neighborhood descent
VRP	vehicle routing problem

NOMENCLATURE

Number Sets

$\mathbb{R}$	All real numbers
$\mathbb{Z}$	Set of all integers

Operators

$\wedge$	Logical and operator
$\cdot\,\vert_{[a,b]}$	Clamp operator between a and b
$\mathcal{I}_A$	Indicator variable for event A
$\omega_{[a,b]}$	Random uniform variable bounded between a and b
$\overset{k,e}{+}$, $\overset{k,e}{-}$	Add or remove last visit in the plan of agent k
$\overset{k,y}{+}$, $\overset{k,y}{-}$	Add or remove yth visit in the plan of agent k
$\overset{R,\Pi}{\longleftarrow}$	Randomly select an element of a set with probabilities proportional to the set Π
$\overset{R}{\longleftarrow}$	Randomly select an element of a set

General Problem Specification

ΔT	Length of planning time window
$\mathcal{T}$	Planning time window
$\mathcal{X}$	Set of all plans for all agents
$\mathcal{X}_k$	Set of all plans for agent k
τ_z^k	Dwell time of agent k at visit z
τ_{max}	Maximal dwell time
T_s, T_e	Start and end of planning time window

Graph Specification

$\mathcal{E}$	Directed set of edges/streets
$\mathcal{G}$	Directed street graph
$\mathcal{V}$	Set of street vertices
c_{ij}	SOC difference while traveling edge e_{ij}
d_{ij}	Street length along edge e_{ij}
e_{ij}	Edge starting at v_i and ending at v_j
n_v	Number of non-dispatch vertices
t_{ij}	Travel time along edge e_{ij}
v_0	Dispatch vertex

Agent Specification

$\mathcal{A}$	Set of all agents
$\overline{d}_k$	Max trip distance of agent k
$\overline{q}_k$	Max SOC of agent k
$chT_k(q, q_t)$	Time needed to charge agent k from q to q_t
$d_k(t)$	Distance since last charge of agent k at time t
$eSOC_k(q, p)$	The SOC of agent k after traversing path p and starting with SOC q
n_a	Number of agents
$q_k(t)$	The SOC of agent k at time t
r_k	Rate of charge of agent k
T_k^s, T_k^e	Start and end time of the shift assigned to agent k
${}_a p_k(t)$	The position of agent k at time t

Hotspot Specification

$\lambda_l(t)$	Expected crime rate within hotspot l at time t
$\mathcal{H}$	Set of all hotspots
$\underline{\lambda}_l, \overline{\lambda}_l$	Min and max EDR in hotspot l
n_h	Number of hotspots
${}_m\gamma_l, {}_n\gamma_l$	Change in λ_l per a second when hotspot l is and is not being monitored
${}_h p_l$	Position of hotspot l

Path Specification

$\mathcal{P}$	Set of all paths though $\mathcal{G}$
$\mathcal{P}_{ij}$	Set of all paths starting at v_i and ending at v_j
$\mathcal{V}^p$	The planning vertex set
$\widehat{\mathcal{P}}$	Subset of paths that contribute to constraints
$\widehat{\mathcal{P}}_{ij}$	Subset of paths between v_i and v_j that contribute to the constraints
c_p	Traversal charge of path p
d_p	Length of path p
p_{ij}^{c*}	Minimum charge usage path while traversing from v_i to v_j
p_{ij}^{cn*}	Minimum charge needed path while traversing from v_i to v_j
p_{ij}^{d*}	Minimum distance path traversing from v_i to v_j
p_{ij}^{t*}	Minimum time path traversing from v_i to v_j
t_p	Traversal time of path p
${}_n c_p$	Charge needed to be able to traverse path p
${}_{eu} c_p$	Charge used after the maximum charge point of the path p

Metric Definition

$\alpha_{edr,2}, \alpha_{edr,\infty}, \alpha_{rt,2}, \alpha_{rt,\infty}$	Objective function weighting coefficients
$\mathcal{V}_e$	Representative subset of $\mathcal{V}$ for response time calculation
$J(x)$	The overall objective function of solution x
$J_{rt}(x, t, v_e)$	Minimum response time to an emergency at location v_e at time t given solution x

Distance Calculation Definitions

$\mathcal{ES}$	Set of all edit-scripts
$\odot$	Edit-script application operator
$\odot_z$	Subset of an edit-script application operator
d	Unrestricted Levenshtein distance function
es	Unrestricted Levenshtein edit-script function
W_i, W_d, W_s	Distance increment of each edit-operation
w_v, w_p, w_{dt}	Weight of each plan edit type
Initial Population Definitions	
δ_h	GRBF width parameter for P_k
$\mathcal{VP}_k(t)$	Set of valid paths for agent k at time t
τ_{init}	How long to dwell at hotspots while initializing
$F_k(t)$	Potential field value for agent k at time t
n_P	Population size
P_{init}	Initial population set
w_h, w_a	Hotspot attraction and agent repulsion weighting
Selection Procedure Definitions	
i_{cl}^{max}	Max number of k-medoids iterations to run
M_c	Cluster c
MP	The membership probability index
n_{cl}	Number of clusters for k-medoids selection
P	The population set
$selection$	Population selection method
Mutation Definitions	
Π	The set of mutation probabilities
π_j	Probability of applying mutation j
$\tau_{max,dtt}$	Maximal magnitude of dwell time change during the Dwell Time Tweak mutation
Θ	Set of all mutation operators
n_Θ	Number of mutation operators
Local Search Definitions	
π_{vnd}	Probability of applying VND
N_i	Neighborhood i generator
t_{max}^{vnd}	Maximum length of time to run VND
VND	VND function
Simulated Annealing Definitions	
α_a	Cost accept factor
α_T	Cooling rate
n_g	Group number
n_{elite}	Number of elite samples
T_{max}	Max temperature
t_{max}^{sa}	Maximum length of time to run SA
T_{min}	Min temperature
Firefly Planner Definitions	
α_ω	Scaling term on movement randomization
β	Attractiveness coefficient function
κ	Light absorption coefficient
ξ	Distance power coefficient
Charge Model	
$\eta_b, \eta_m, \eta_{rb}$	Battery, motor, and regenerative braking energy efficiencies
$\mathfrak{C}_d$	Aerodynamic drag coefficient of vehicle
$\mathfrak{C}_r, \mathfrak{c}_1, \mathfrak{c}_2$	Roll resistance parameters
ρ_{air}	Air mass density
θ_{ij}	Road grade along edge e_{ij}
A_f	Frontal area of vehicle
g	Gravitational acceleration
m	Vehicles mass
P_{aux}	Instantaneous power usage by auxiliary systems
${}_aa(t)$	Instantaneous acceleration of a vehicle
${}_av(t)$	Instantaneous velocity of a vehicle
${}_bP_{ij}$	Instantaneous power usage at the battery
${}_ev_{ij}$	Traveling speed along edge e_{ij}
${}_mP_{ij}(t)$	Instantaneous power usage at the motor
${}_wP_{ij}(t)$	Instantaneous power usage at the wheels

Appendix B
Hyperparameter Tuning Results

The results of the hyperparameter tuning are given in Table III. HGPRA and HFA both perform the best with small population sizes. This is likely the result of the computational time that is taken up during the Levenshtein distance calculation, which is significant and must be performed between all elements of the population in those planners. Looking at the hyperparameters related to FAP, it can be seen that the agent avoidance term, w_a, is the dominant term compared to the hotspot attraction term, w_h. This means that avoiding going to the same hotspot as another agent is more important to overall plan quality than going to the closest hotspot.

Appendix C
Problem Generation

The graphs that are planned over are queried from the Open Street Map project [33] through the OSMnx interface [36]. OSMnx provides the position of each node in the graph, the travel speeds of every edge ${}_ev_{ij} \in \mathbb{R}_{>0}$, length of every edge d_{ij}, and the approximate grades of every edge $\theta_{ij} \in \mathbb{R}$. From this, the edge traversal time is calculated as $t_{ij} = d_{ij}/{}_ev_{ij}$.

C.1 Power Model

The power-based EV energy consumption model used is derived in [37]. It uses road grades, speeds, and lengths to estimate the battery usage over each edge. The model in [37] starts by defining the instantaneous power usage at the wheels ${}_wP_{ij}(t) : \mathcal{T} \mapsto \mathbb{R}$ of edge e_{ij} as

$$
\begin{aligned}
{}_wP_{ij}(t) \triangleq {}_av(t)\Bigg({}_aa(t)m + mg\cos(\theta_{ij})\frac{\mathfrak{C}_r({}_av(t)\mathfrak{c}_1 + \mathfrak{c}_2)}{1000} \\
+mg\sin(\theta_{ij}) + {}_av^2(t)\frac{\rho_{air}A_f\mathfrak{C}_d}{2}\Bigg),
\end{aligned}
\tag{37}
$$

where ${}_av(t) : \mathcal{T} \mapsto \mathbb{R}_{>0}$ and ${}_aa(t) : \mathcal{T} \mapsto \mathbb{R}$ are the instantaneous velocity and acceleration of a vehicle traversing the edge. The mass of the vehicle is denoted $m \in \mathbb{R}_{>0}$, $g \in \mathbb{R}_{>0}$ is the gravitational acceleration, $\rho_{air} \in \mathbb{R}_{>0}$ is the air mass density, $A_f \in \mathbb{R}_{>0}$ is the frontal area of the vehicle, and $\mathfrak{C}_d \in \mathbb{R}_{>0}$ is the coefficient of aerodynamic drag of the vehicle. The constants $\mathfrak{C}_r \in \mathbb{R}_{>0}$, $\mathfrak{c}_1 \in \mathbb{R}_{>0}$, and $\mathfrak{c}_2 \in \mathbb{R}_{>0}$ are the rolling resistance parameters of the road and vehicle tires.

The instantaneous power usage of the motor, ${}_mP_{ij}(t) : \mathcal{T} \mapsto \mathbb{R}$, is calculated from ${}_wP_{ij}(t)$ while considering motor efficiency, $\eta_m \in \mathbb{R}_{(0,1)}$. Furthermore, when going down a hill, the power usage at the wheels can be negative, which implies that regenerative braking is in effect. Regenerative

TABLE III: Results of hyperparameter tuning over the planner parameters. τ_{max}, τ_{init}, and $\tau_{max,dtt}$ are given in minutes and t^{vnd}_{max} and t^{sa}_{max} are given in seconds.

Algorithm	HGPRA		HFA		HPSA		RAP	GAP	FAP
Selection	FPS	KMS	FPS	KMS	FPS	KMS			
n_P	5	5	5	5	19	38	-	-	-
π_{ahv}	0.62	0.90	0.80	0.64	0.49	0.61	-	-	-
π_{adv}	0.38	0.30	0.82	0.50	0	0.31	-	-	-
π_{rhv}	0.50	0.29	0.67	0.65	0.15	0.35	-	-	-
π_{rdv}	0.59	0.29	0.09	0.41	0.69	0.05	-	-	-
π_{rmv}	0.35	0.22	0.63	0.71	0.11	0	-	-	-
π_{sp}	0.69	0.58	0.79	0.91	0.48	0.46	-	-	-
π_{vs}	0.63	0.73	0.47	0.40	0.68	0.50	-	-	-
π_{mvs}	0.42	0.75	0.52	0.16	0.28	0.57	-	-	-
π_{sim}	0.60	0.19	0.27	0.56	0.49	0.46	-	-	-
π_{dtt}	0.32	0.48	0.66	0.52	1	0.48	-	-	-
$\tau_{max,dtt}$	48	231	87	123	5	103	-	-	-
τ_{max}	263	296	387	339	-	371	-	-	-
π_{vnd}	0.63	0.01	0.83	0.71	-	-	-	-	-
t^{vnd}_{max}	0.01	17.61	0.01	16.49	44.29	0.01	-	-	-
κ	-	-	48.38	36.01	-	-	-	-	-
ξ	-	-	2.8	11.03	-	-	-	-	-
β_0	-	-	11.61	43.05	-	-	-	-	-
t^{sa}_{max}	-	-	-	-	5.48	56.98	-	-	-
α_a	-	-	-	-	2.75	3.54	-	-	-
α_T	-	-	-	-	0.09	0.66	-	-	-
n_g	-	-	-	-	99	76	-	-	-
n_{elite}	-	-	-	-	19	24	-	-	-
T_{max}	-	-	-	-	1414.7	812.36	-	-	-
T_{min}	-	-	-	-	86.79	51.16	-	-	-
i^{max}_{cl}	-	27	-	28	-	51	-	-	-
n_{cl}	-	2	-	2	-	20	-	-	-
τ_{init}	11	11	11	11	9	11	14	11	16
δ_h	124558	144957	78764	86316	140399	85369	-	-	39355
w_a	791.2	1159	5908	6768	1142	7983	-	-	8211
w_h	0	284	285	41	0	273	-	-	191

braking has an additional efficiency coefficient $\eta_{rb} \in \mathbb{R}_{(0,1)}$. Regenerative braking efficiency is a function of the magnitude of acceleration. A detailed description is omitted for brevity; see [37] for details. Putting this together, the instantaneous power usage at the motor is calculated as

$$ {}_mP_{ij}(t) \triangleq \begin{cases} \frac{{}_wP_{ij}(t)}{\eta_m} & \text{if } {}_wP_{ij}(t) \geq 0 \\ {}_wP_{ij}(t)\eta_m\eta_{rb} & \text{if } {}_wP_{ij}(t) < 0 \end{cases}. \tag{38} $$

The power consumption of the auxiliary systems, such as air conditioning and lights, is denoted $P_{aux} \in \mathbb{R}_{>0}$. Finally, the instantaneous power consumption at the battery is calculated considering the efficiency of the battery, $\eta_b \in \mathbb{R}_{(0,1)}$, as

$$ {}_bP_{ij}(t) \triangleq \begin{cases} ({}_mP_{ij}(t) + P_{aux})/\eta_b & \text{if } {}_mP_{ij}(t) + P_{aux} \geq 0 \\ ({}_mP_{ij}(t) + P_{aux})\eta_b & \text{if } {}_mP_{ij}(t) + P_{aux} < 0 \end{cases}. \tag{39} $$

This value is converted to the energy consumed while traversing an edge with

$$ c_{ij} \triangleq \int_0^{t_{ij}} {}_bP_{ij}(t)dt. \tag{40} $$

In this work, the velocity of the agents along an edge is kept constant at the traveling speed of the edge, i.e., $\forall t, {}_av(t) \equiv {}_ev_{ij}, {}_aa(t) \equiv 0$. Using this (37) and (40) become

$$ {}_wP_{ij} \triangleq {}_ev_{ij}\Bigg(mg\cos(\theta_{ij})\frac{\mathfrak{C}_r({}_ev_{ij}\mathfrak{c}_1 + \mathfrak{c}_2)}{1000} + mg\sin(\theta_{ij}) + {}_ev_{ij}^2\frac{\rho_{air}A_f\mathfrak{C}_d}{2}\Bigg), \tag{41} $$

$$ c_{ij} = {}_bP_{ij}t_{ij}. \tag{42} $$

This energy consumption model is used to simulate the 2025 Chevrolet Blazer EV police patrol vehicle. The various physical parameters of this vehicle are given in Table IV. These parameters are taken from the specifications sheet of the vehicle when possible, and estimated to the best of our ability when information is lacking.

TABLE IV: Estimated physical parameters.

Parameter	Value	Parameter	Value	Parameter	Value
η_b	0.95	η_m	0.8	$\mathfrak{C}_d$	0.3
$\mathfrak{C}_r$	1.75	$\mathfrak{c}_1$	0.0328	$\mathfrak{c}_2$	4.575
ρ_{air}	1.226 $\frac{\text{kg}}{\text{m}^3}$	A_f	3.252 m^2	g	9.81 $\frac{\text{m}}{\text{s}^2}$
m	2787 kg	P_{aux}	3 kW		

C.2 Hotspot Model

When an agent arrives at a hotspot, it can immediately drive to the next hotspot or dwell in the general area of the hotspot for a time. This dwell time is used to solidify the

presence of the police in the location, increasing the residual deterrence to crime when the agent leaves the hotspot. Justice research in [38] shows that an 11 to 15 minute dwell time at a hotspot increases the probability of no crimes occurring in the next half hour by 0.12 when compared to just a drive by being performed. The EDR model parameters are chosen to reflect this data. The study performed in [38] shows that the probability that a crime occurs in the half hour after a drive by is 0.161 and after an 11 to 15 minute stop the probability is 0.041. In this work, these values are the maximal and minimal EDR, i.e., $\forall l \in \mathcal{H}, \underline{\lambda}_l = 0.041, \overline{\lambda}_l = 0.161$. Following the results in [38], a 15 minute dwell time brings the EDR from maximal to minimal so

$$ {}_m\gamma_l \triangleq \frac{\overline{\lambda}_l - \underline{\lambda}_l}{15min.} \approxeq 1.33e^{-4}. \tag{43} $$

It is unclear how long it should take for the EDR to reach its maximum value from the available research. It is chosen to have the EDR reach its maximal value 3 hours after being patrolled, which leads to

$$ {}_n\gamma_l \triangleq \frac{\underline{\lambda}_l - \overline{\lambda}_l}{3hr.} \approxeq -1.11e^{-5}. \tag{44} $$

Appendix D
$\widehat{\mathcal{P}}$ Derivation

To see why the definition of $\widehat{\mathcal{P}}$ ensures that all constraint critical paths in $\mathcal{P}$ are an element of $\widehat{\mathcal{P}}$, consider an agent $k \in \mathcal{A}$ at time $t \in \mathcal{T}$ that is currently at vertex $v_i \in \mathcal{V}$. The next path that agent k takes, $p_{ij} \in \mathcal{P}_{ij}$, must satisfy some constraints. One such condition on the next path taken is that there must be a way to get back to the depot without running out of time in the planning horizon, i.e., $\exists p_{j0} \in \mathcal{P}_{j0}$ such that

$$ t + t_{p_{ij}} + t_{p_{j0}} \leq T_e. \tag{45} $$

This is why the minimal traversal time path is always included in $\widehat{\mathcal{P}}_{ij}$ with the condition $t_{p'_{ij}} \leq t_{p_{ij}}$.

The other constraints on p_{ij} depend on whether SOC or distance-based constraints are being used. When SOC-based constraints are used, agent k must have enough charge to follow p_{ij} and get back to the depot, i.e., $\exists p_{j0} \in \mathcal{P}_{j0}$ such that

$$ {}_nc_{p_{j0}} \leq eSOC_k(q_k(t), p_{ij}). \tag{46} $$

Expanding on (46) produces

$$ \begin{aligned} {}_nc_{p_{j0}} &\leq eSOC_k(q_k(t), p_{ij}) && (47)\\ &= \big(q_k(t) - c_{p_{ij}}\big)\big|_{[0, \overline{q}_k - {}_{eu}c_{p_{ij}}]} && (48)\\ &= \min\big(q_k(t) - c_{p_{ij}}, \overline{q}_k - {}_{eu}c_{p_{ij}}\big) && (49)\\ &= \min\big(\overline{q}_k - (\overline{q}_k - q_k(t)) - c_{p_{ij}}, \overline{q}_k - {}_{eu}c_{p_{ij}}\big) && (50)\\ &= \overline{q}_k - \max\big((\overline{q}_k - q_k(t)) + c_{p_{ij}}, {}_{eu}c_{p_{ij}}\big), && (51) \end{aligned} $$

where the third to last step is performed assuming $\overline{q}_k \geq c_{p_{ij}}$. Minimizing the left hand side the with inequality means minimizing ${}_nc_{p_{j0}}$ which is why the minimal charge needed path is always included in $\widehat{\mathcal{P}}_{ij}$ with the condition ${}_nc_{p'_{ij}} \leq {}_nc_{p_{ij}}$. Maximizing the right hand side means minimizing $c_{p_{ij}}$ and/or ${}_{eu}c_{p_{ij}}$ which is where the last two conditions on $\widehat{\mathcal{P}}_{ij}$ come from.

When distance-based constraints are used, the complementary constraint to (46) is that agent k must have enough trip distance remaining to follow p_{ij} and return to the depot, i.e., $\exists p_{j0} \in \mathcal{P}_{j0}$ such that

$$ d_{p_{ij}} + d_{p_{j0}} \leq \overline{d}_k - d_k(t). \tag{52} $$

Minimizing the right hand side of this inequality means minimizing the length of the paths taken, which is why the constraint $d_{p'_{ij}} \leq d_{p_{ij}}$ in part of the definition of $\widehat{\mathcal{P}}$.

CHAPTER 8

CONCLUSION

The field of vehicle mission planning is growing in importance as the use of autonomous systems becomes widespread. As the vehicle technologies used advance, the problem of vehicle mission planning problem grows in complexity. It is not surprising then that the literature in the field contains several gaps. This work makes significant progress towards filling these gaps by contributing several important new functionalities to the literature. Specifically, the contributions of this dissertation are the following:

1. This dissertation develops rapid path planning algorithms to generate paths and motion plans that can be followed by kinematically constrained vehicles. The algorithms produce paths of approximately equivalent quality to those of the unconstrained version of the algorithms, given the same amount of time.

    (a) Incorporates common kinematic vehicle constraints, specifically maximum curvature and maximum curvature rate, into the formulation of general fillet-based motion primitives.

    (b) Derives the necessary modifications to Optimal Rapidly-exploring Random Tree (RRT*) and Batch Informed Trees (BIT*), in order to use fillet-based motion primitives, producing Fillet-Based Rapidly-exploring Random Tree (FB-RRT*) and Fillet-Based Batch Informed Trees (FB-BIT*), respectively.

    (c) Develops the Smart and Informed sampling heuristic that accelerates the rate of convergence of sampling-based motion planners to the optimal solution.

2. This dissertation develops a version of FB-BIT* that is capable of respecting constraints that depend on the location of evaluation and the path taken to get there, which is a new contribution to the field of sampling-based path planners.

3. This dissertation uses the augmented version of FB-BIT* from Contribution 2 to produce minimal path-length unmanned aerial vehicle (UAV) flyable paths through adversarial environments with unreliable global positioning system (GPS) support while preventing ground-based radars from detecting the UAV.

   (a) Mathematically defines a closed-loop Monte Carlo simulation of a fixed-wing UAV with a Kalman filter estimator.

   (b) Incorporates the UAV truth state uncertainty into the calculation of the probability of detection (POD) constraint to increase the reliability of the final planner.

   (c) Implements a linear covariance (LinCov) framework to speed up the POD constraint evaluation.

4. This dissertation develops three algorithms to solve the electric vehicle patrol routing problem (EVPRP) such that the expected disturbance rate (EDR) of the region being monitored is minimized.

   (a) Devises a representative problem model of a city road network with nonuniform EDRs. This problem model consists of actual street locations and speed limits and mimics the way EDR is commonly modeled in the criminal justice literature.

   (b) Devises a set of state of charge (SOC)-based constraints to enforce feasible charging and driving schedules for electric vehicles (EVs) with regenerative braking, which no previous work considers. Additionally, compares the devised SOC-based constraints to the distance-based constraints that are common in the literature to show that the SOC-based constraints allow for additional optimality. These constraints are modeled on a 2025 Chevrolet Blazer EV police patrol vehicle to ensure real-world applicability.

   (c) Uses the Levenshtein edit-distance in a novel way to enable the development of two mission planners for solving the EVPRP: hybrid genetic path relinking algorithm (HGPRA) and hybrid firefly algorithm (HFA). Additionally, developed a hybrid population-based simulated annealing (HPSA) mission planner

for comparison and validation that the way the Levenshtein distance is applied is advantageous when solving the EVPRP. These planners minimize the EDR of all hotspots while respecting battery constraints and considering emergency response times.

These contributions provide significant progress towards filling high-impact gaps in the vehicle mission planning literature. Curvature constrained vehicles are ever prevalent in society today. With the growing push to build autonomous systems, the need for an efficient motion planner for these curvature constrained vehicles has never been greater. The fillet-based path planners this work develops produce solutions that respect kinematic constraints far more efficiently than preexisting algorithms. With the increasing number of autonomous vehicles, it has become clear that GPS cannot be counted on in many important applications. Developing planners that take this unreliability into account while producing safe paths to follow is essential. The augmented FB-BIT* algorithm proposed in this work far exceeds previous work in its ability to consistently produce valid paths and the optimality of the resulting paths. Furthermore, the growing use of EVs has added more complications to autonomous vehicle planning. The SOC constraint framework proposed herein is a big step towards utilizing fleets of EVs efficiently, especially in the field of patrol routing problems (PRPs) where EV constraints have not been considered before. Additionally, the use of advanced edit-distances while defining solution differences and crossover operators within the field of PRPs and the larger field of vehicle routing problems (VRPs) seems to be novel and a highly effective strategy for solving these problems.

REFERENCES

[1] J. R. Cooper, "Optimal multi-agent search and rescue using potential field theory," in *AIAA Scitech 2020 Forum*, 2020, p. 0879.

[2] S. A. Cox, "Mission planning techniques for cooperative leo spacecraft constellations," 2023.

[3] J. Whitaker, G. Droge, M. Hansen, D. Mortensen, and J. Gunther, "A network flow approach to battery electric bus scheduling," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 9, pp. 9098–9109, 2023.

[4] C. S. Koper, "Just enough police presence: Reducing crime and disorderly behavior by optimizing patrol time in crime hot spots," *Justice quarterly*, vol. 12, no. 4, pp. 649–672, 1995.

[5] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011. [Online]. Available: https://doi.org/10.1177/0278364911406761

[6] R. S. Christensen, G. Droge, and R. C. Leishman, "Closed-loop linear covariance framework for path planning in static uncertain obstacle fields," *Journal of Guidance, Control, and Dynamics*, vol. 45, no. 4, pp. 669–683, 2022. [Online]. Available: https://doi.org/10.2514/1.G006228

[7] A. Hakobyan, G. C. Kim, and I. Yang, "Risk-aware motion planning and control using cvar-constrained optimization," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3924–3931, 2019.

[8] D. Silver and J. Veness, "Monte-carlo planning in large pomdps," *Advances in neural information processing systems*, vol. 23, 2010.

[9] A. Costley, R. Christensen, R. C. Leishman, and G. Droge, "Sensitivity of single-pulse radar detection to aircraft pose uncertainties," *IEEE Transactions on Aerospace and Electronic Systems*, pp. 1–10, 2022.

[10] M. Jun and R. D'Andrea, "Path planning for unmanned aerial vehicles in uncertain and adversarial environments," *Cooperative control: models, applications and algorithms*, pp. 95–110, 2003.

[11] A. Costley, J. Swedeen, G. Droge, R. Christensen, and R. C. Leishman, "Path planning with uncertainty for aircraft under threat of detection from ground-based radar," *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 2025, in press.

[12] A. Costley, R. Christensen, R. C. Leishman, and G. Droge, "Sensitivity of the probability of radar detection to radar state uncertainty," *IEEE Transactions on Aerospace and Electronic Systems*, pp. 1–20, 2023.

[13] B. R. Mahafza and A. Elsherbeni, *MATLAB Simulations for Radar Systems Design*. CRC Press, Dec. 2003.

[14] F. Xiao-wei, L. Zhong, and G. Xiao-guang, "Path planning for uav in radar network area," in *2010 Second WRI Global Congress on Intelligent Systems*, vol. 3. IEEE, 2010, pp. 260–263.

[15] F. W. Moore, "Radar cross-section reduction via route planning and intelligent control," *IEEE Transactions on Control Systems Technology*, vol. 10, no. 5, pp. 696–700, 2002.

[16] M. B. McFarland, R. A. Zachery, and B. K. Taylor, "Motion planning for reduced observability of autonomous aerial vehicles," in *Proceedings of the 1999 IEEE International Conference on Control Applications (Cat. No. 99CH36328)*, vol. 1. IEEE, 1999, pp. 231–235.

[17] R. Larson, M. Pachter, and M. Mears, "Path planning by unmanned air vehicles for engaging an integrated radar network," in *AIAA guidance, navigation, and control conference and exhibit*, 2005, p. 6191.

[18] T. Erlandsson, "Route planning for air missions in hostile environments," *The Journal of Defense Modeling and Simulation*, vol. 12, no. 3, pp. 289–303, 2015.

[19] B. R. M. A. Z. Elsherbeni, "Simulations for radar system design," 2003.

[20] P. T. Kabamba, S. M. Meerkov, and F. H. Zeitz III, "Optimal path planning for unmanned combat aerial vehicles to defeat radar tracking," *Journal of Guidance, Control, and Dynamics*, vol. 29, no. 2, pp. 279–288, 2006.

[21] M. Pachter and J. Hebert, "Optimal aircraft trajectories for radar exposure minimization," in *Proceedings of the 2001 American Control Conference.(Cat. No. 01CH37148)*, vol. 3. IEEE, 2001, pp. 2365–2369.

[22] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[23] ——, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998.

[24] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 995–1001 vol.2.

[25] R. Beard and T. McLain, *Small Unmanned Aircraft: Theory and Practice*. Princeton University Press, 2012. [Online]. Available: https://books.google.com/books?id=YqQtjhPUaNEC

[26] S. A. Bortoff, "Path planning for uavs," in *Proceedings of the 2000 american control conference. ACC (IEEE Cat. No. 00CH36334)*, vol. 1, no. 6. IEEE, 2000, pp. 364–368.

[27] N. Ceccarelli, J. J. Enright, E. Frazzoli, S. J. Rasmussen, and C. J. Schumacher, "Micro uav path planning for reconnaissance in wind," in *2007 American Control Conference*. IEEE, 2007, pp. 5310–5315.

[28] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2997–3004.

[29] J. Nasir, F. Islam, U. Malik, Y. Ayaz, O. Hasan, M. Khan, and M. Muhammad, "Rrt*-smart: A rapid convergence implementation of rrt*," *International Journal of Advanced Robotic Systems*, vol. 10, 2013.

[30] I. Noreen, A. Khan, and Z. Habib, "Optimal path planning using rrt* based approaches: A survey and future directions," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 11, 2016. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2016.071114

[31] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 2640–2645.

[32] M. Kobilarov, "Cross-entropy motion planning," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 855–871, 2012. [Online]. Available: https://doi.org/10.1177/0278364912444543

[33] I.-B. Jeong, S.-J. Lee, and J.-H. Kim, "Quick-rrt*: Triangular inequality-based implementation of rrt* with improved initial solution and convergence rate," *Expert Systems with Applications*, vol. 123, pp. 82–90, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417419300326

[34] Z. Tahir, A. H. Qureshi, Y. Ayaz, and R. Nawaz, "Potentially guided bidirectionalized rrt* for fast optimal path planning in cluttered environments," *Robotics and Autonomous Systems*, vol. 108, pp. 13–27, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889017309387

[35] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[36] M. Likhachev, G. J. Gordon, and S. Thrun, "Ara* : Anytime a* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems*, S. Thrun, L. Saul, and B. Scholkopf, Eds., vol. 16. MIT Press, 2003. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2003/file/ee8fe9093fbbb687bef15a38facc44d2-Paper.pdf

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

[38] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 3067–3074.

[39] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Batch informed trees (bit*): Informed asymptotically optimal anytime search," *The International Journal of Robotics Research*, vol. 39, no. 5, pp. 543–567, 2020. [Online]. Available: https://doi.org/10.1177/0278364919890396

[40] K. Yang and S. Sukkarieh, "An analytical continuous-curvature path-smoothing algorithm," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 561–568, 2010.

[41] C. Jiang, Z. Hu, Z. P. Mourelatos, D. Gorsich, P. Jayakumar, Y. Fu, and M. Majcher, "R2-rrt*: Reliability-based robust mission planning of off-road autonomous ground vehicle under uncertain terrain environment," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 2, pp. 1030–1046, 2022.

[42] K. Yang, S. Moon, S. Yoo, J. Kang, N. L. Doh, H. B. Kim, and S. Joo, "Spline-based rrt path planner for non-holonomic robots," *Journal of Intelligent & Robotic Systems*, vol. 73, no. 1-4, pp. 763–782, 2014a.

[43] C. Moon and W. Ching, "Kinodynamic planner dual-tree rrt (dt-rrt) for two-wheeled mobile robots using the rapidly exploring random tree," *IEEE Transactions on Industrial Electronics*, vol. 62, pp. 1080–1090, 2015.

[44] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 528–564, 2016. [Online]. Available: https://doi.org/10.1177/0278364915614386

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

[46] P. Cui, W. Yan, and X. Guo, "Path planning for robot with pose constraints using dubins-rrt," in *2018 3rd International Conference on Advanced Robotics and Mechatronics (ICARM)*, 2018, pp. 560–565.

[47] X. Lan and S. Di Cairano, "Continuous curvature path planning for semi-autonomous vehicle maneuvers using rrt," in *2015 European Control Conference (ECC)*, 2015, pp. 2360–2365.

[48] K. Yang, S. K. Gan, J. Huh, and S. Joo, "Optimal spline-based rrt path planning using probabilistic map," in *2014 14th International Conference on Control, Automation and Systems (ICCAS 2014)*, 2014b, pp. 643–646.

[49] Y. Jiang, H. Li, B. Feng, Z. Wu, S. Zhao, and Z. Wang, "Street patrol routing optimization in smart city management based on genetic algorithm: a case in zhengzhou, china," *ISPRS International Journal of Geo-Information*, vol. 11, no. 3, p. 171, 2022.

[50] P. J. Cook, "Research in criminal deterrence: Laying the groundwork for the second decade," *Crime and Justice*, vol. 2, pp. 211–268, 1980. [Online]. Available: https://doi.org/10.1086/449070

[51] R. Dewil, P. Vansteenwegen, D. Cattrysse, and D. Van Oudheusden, "A minimum cost network flow model for the maximum covering and patrol routing problem," *European Journal of Operational Research*, vol. 247, no. 1, pp. 27–36, 2015.

[52] J. Chen, A. Baskaran, Z. Zhang, and P. Tokekar, "Multi-agent reinforcement learning for visibility-based persistent monitoring," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 2563–2570.

[53] X. Chen, "Fast patrol route planning in dynamic environments," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 42, no. 4, pp. 894–904, 2012.

[54] H. Chen, T. Cheng, and S. Wise, "Developing an online cooperative police patrol routing strategy," *Computers, Environment and Urban Systems*, vol. 62, pp. 19–29, 2017.

[55] B. Pan, Z. Zhang, and A. Lim, "Multi-trip time-dependent vehicle routing problem with time windows," *European Journal of Operational Research*, vol. 291, no. 1, pp. 218–231, 2021.

[56] R. Zhang, J. Guo, and J. Wang, "A time-dependent electric vehicle routing problem with congestion tolls," *IEEE Transactions on Engineering Management*, vol. 69, no. 4, pp. 861–873, 2022.

[57] Z. H. Ahmed, A. S. Hameed, M. L. Mutar *et al.*, "Hybrid genetic algorithms for the asymmetric distance-constrained vehicle routing problem," *Mathematical Problems in Engineering*, vol. 2022, 2022.

[58] J.-P. Riquelme-Rodríguez, A. Langevin, and M. Gamache, "Adaptive large neighborhood search for the periodic capacitated arc routing problem with inventory constraints," *Networks*, vol. 64, no. 2, pp. 125–139, 2014.

[59] İ. İlhan, "An improved simulated annealing algorithm with crossover operator for capacitated vehicle routing problem," *Swarm and Evolutionary Computation*, vol. 64, p. 100911, 2021.

[60] K. Alabdulkareem and Z. H. Ahmed, "Comparison of four genetic crossover operators for solving distance-constrained vehicle routing problem," *IJCSNS International Journal of Computer Science and Network Security*, vol. 20, no. 7, pp. 114–123, 2020.

[61] A. M. Altabeeb, A. M. Mohsen, and A. Ghallab, "An improved hybrid firefly algorithm for capacitated vehicle routing problem," *Applied Soft Computing*, vol. 84, p. 105728, 2019.

[62] A. M. Altabeeb, A. M. Mohsen, L. Abualigah, and A. Ghallab, "Solving capacitated vehicle routing problem using cooperative firefly algorithm," *Applied Soft Computing*, vol. 108, p. 107403, 2021.

[63] R. Paradiso, R. Roberti, D. Laganá, and W. Dullaert, "An exact solution framework for multitrip vehicle-routing problems with time windows," *Operations Research*, vol. 68, no. 1, pp. 180–198, 2020.

[64] B. B. Keskin, S. R. Li, D. Steil, and S. Spiller, "Analysis of an integrated maximum covering and patrol routing problem," *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 1, pp. 215–232, 2012.

[65] C. Fiori, K. Ahn, and H. A. Rakha, "Power-based electric vehicle energy consumption model: Model development and validation," *Applied Energy*, vol. 168, pp. 257–268, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S030626191630085X

[66] L. Boytsov, "Indexing methods for approximate dictionary searching: Comparative analysis," *ACM J. Exp. Algorithmics*, vol. 16, may 2011. [Online]. Available: https://doi.org/10.1145/1963190.1963191

[67] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*, ser. Advanced Textbooks in Control and Signal Processing. Springer London, 2008. [Online]. Available: https://books.google.com/books?id=VsTOQOnQjCAC

[68] A. H. Qureshi and Y. Ayaz, "Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments," *Robotics and Autonomous Systems*, vol. 68, pp. 1–11, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889015000317

[69] S. Sedighi, D.-V. Nguyen, and K.-D. Kuhnert, "Guided hybrid a-star path planning algorithm for valet parking applications," in *2019 5th International Conference on Control, Automation and Robotics (ICCAR)*, 2019, pp. 570–575.

[70] X. Sun, C. Cai, J. Yang, and X. Shen, "Route evaluation for unmanned aerial vehicle based on type-2 fuzzy sets," *Engineering Applications of Artificial Intelligence*, vol. 39, pp. 132–145, 2015.

[71] X.-S. Yang, "Firefly algorithms for multimodal optimization," in *Stochastic Algorithms: Foundations and Applications*, O. Watanabe and T. Zeugmann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 169–178.

[72] E. Osaba, X.-S. Yang, F. Diaz, E. Onieva, A. D. Masegosa, and A. Perallos, "A discrete firefly algorithm to solve a rich vehicle routing problem modelling a newspaper distribution system with recycling policy," *Soft Computing*, vol. 21, pp. 5295–5308, 2017.

[73] J. Swedeen, G. Droge, and R. Christensen, "Fillet-based rrt*: A rapid convergence implementation of rrt* for curvature constrained vehicles," *Journal of Intelligent & Robotic Systems*, vol. 108, no. 4, p. 68, 2023.

[74] J. Swedeen and G. Droge, "Fillet-based batch informed trees (fb-bit*): Rapid convergence path planning for curvature constrained vehicles," in *2023 Seventh IEEE International Conference on Robotic Computing (IRC)*, 2023, pp. 71–78.

[75] R. S. Christensen and D. Geller, "Linear covariance techniques for closed-loop guidance navigation and control system design and analysis," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 228, no. 1, pp. 44–65, 2014.

[76] D. K. Geller, "Linear covariance techniques for orbital rendezvous analysis and autonomous onboard mission planning," *Journal of Guidance, Control, and Dynamics*, vol. 29, no. 6, pp. 1404–1414, 2006.

[77] J. Crassidis and J. Junkins, *Optimal Estimation of Dynamic Systems, Second Edition*, ser. Chapman & Hall/CRC Applied Mathematics & Nonlinear Science. Taylor & Francis, 2011. [Online]. Available: https://books.google.com/books?id=ITKKkBFxgNsC

APPENDICES

APPENDIX A

AN INTRODUCTION TO BATCH INFORMED TREES (BIT*)

Batch Informed Trees (BIT*)

James Swedeen
Department of Electrical and Computer Engineering
Utah State University
Logan, Utah 84322
Email: james.swedeen@usu.edu

Greg Droge
Department of Electrical and Computer Engineering
Utah State University
Logan, Utah 84322
Email: greg.droge@usu.edu

***Abstract*—Path planning through complex obstacle spaces is a fundamental requirement of many mobile robot applications. Recently, a rapid convergence path planning algorithm, Batch Informed Trees (BIT*), was introduced. This work serves as a concise introduction and explanation of BIT*. This work includes a description of BIT* and how BIT* operates, a graphical demonstration of BIT*, and simulation results where BIT* is compared to Optimal Rapidly-exploring Random Tree (RRT*).**

I. Introduction

Planning paths between starting and target locations while respecting mission requirements is a fundamental requirement of many robotic applications and is an NP-complete problem in general [1]. A path planning algorithm that has been extensively studied is Rapidly-exploring Random Tree (RRT) [1]. RRT is a sampling-based algorithm in which a rooted search tree grows from the starting location to every reachable configuration in the search space without violating the constraints of the problem. RRT has been used in a wide range of applications because of its speed and probabilistic completeness. However, the solution RRT produces is typically far from optimal, and the probability that RRT produces the optimal solution is zero [2].

Sampling-based mission planning algorithms that provide asymptotic optimality guarantees have become increasingly common in the literature [3]–[6]. These algorithms operate on the principles of dynamic programming, breaking the problem into many smaller problems that can each be solved individually and then combined to make the overall solution. Many sampling-based planning algorithms are based on Optimal Rapidly-exploring Random Tree (RRT*) [2]. RRT* is similar to RRT, building a rooted search tree from the initial location by randomly sampling the search space and connecting the sample to the tree. Unlike RRT, as RRT* searches, local optimizations are performed on the search tree. As the number of iterations RRT* performs goes to infinity, the solution path to the target location converges to optimality [2]. However, this convergence can be slow [3], [4], [7].

Fast Marching Tree (FMT*) is another sampling-based planning algorithm that uses the principles of dynamic programming to avoid unnecessary calculations [8]. FMT* builds a rooted search tree, similar to RRT*. However, instead of iteratively sampling the search space as the algorithm goes, FMT* samples a fixed number of times at startup. A search tree is formed from these sampled points. With knowledge of where samples are from the start, FMT* avoids many of the repetitive calculations that RRT* performs while still producing the same solution. This makes FMT* faster than RRT* at generating a solution. However, it loses the ability RRT* has to refine the solution indefinitely.

Batch Informed Trees (BIT*) merges the iterative nature of RRT* with the efficient graph searching techniques of FMT* [9]. BIT* iteratively generates batches of samples from the search space, then uses a procedure similar to FMT* to incorporate the new batch of samples into the preexisting search tree. The characteristics of BIT* vary with the size of each batch of samples. When the batch size is only one, BIT* is nearly equivalent to RRT*. When the batch size is very large, BIT* is nearly equivalent to FMT*, except for considering subsequent batches. This allows the user of the algorithm to tune BIT* to have the desired performance characteristics.

The rest of this work proceeds as follows. Section II describes general RRT* and BIT* notation used throughout this work. Section III gives a detailed description and step-by-step explanation of the BIT* algorithm. Section IV provides a demonstration of how BIT* operates, graphically, on three batches of samples. Section V provides averaged convergence results using the Open Motion Planning Library (OMPL) and makes comparisons to RRT*.

II. Notation and Background

Section II-A describes the notation used throughout this work as it relates to BIT*. Section II-B defines some general helper procedures to be used in the description of BIT*.

A. Nomenclature

RRT* and BIT*-based algorithms iteratively construct a rooted outbranching tree to find a path through the state space to a desired target location. The d-dimensional state space that RRT* and BIT* plan over is denoted as $X \subseteq \mathbb{R}^d$. The planned path must avoid the space blocked by obstacles, $X_{obs} \subset X$, staying within the obstacle-free space, $X_{free} \triangleq X \setminus X_{obs}$. The target location or target set, $X_t \subset X_{free}$, is a user-defined subset of the obstacle-free space. The planning tree is an acyclic-directed graph denoted as $T \triangleq (V, E)$, where V is the set of nodes or vertices within the tree and $E \subset V \times V$ denotes the set of edges between the vertices. The root vertex has no parent, while all other vertices have exactly one parent.

Each vertex can have multiple children. Each vertex within V corresponds to a state in the obstacle-free state space, X_{free}. The tree is initialized with only the root node, i.e. $V = \{x_r\}$, $E = \varnothing$. Nodes are iteratively added to the tree to find a path to the target set, X_t.

$\mathbb{R}$ is the set of all real numbers, and $\mathbb{R}_+$ is the set of all real positive numbers. The notation $X \xleftarrow{+} \{x\}$ and $X \xleftarrow{-} \{x\}$ is used to compactly represent the set updating operations $X \leftarrow X \cup \{x\}$ and $X \leftarrow X \setminus \{x\}$ respectively.

B. Common Procedures

In this section, several primitive sub-procedures are defined for use while describing BIT*. We now define several generic procedures that can be found in [9], [10] with the notation updated to match the sequel.

Definition 1 ($cost \leftarrow c(x, y)$): Calculates the length of the path in X that connects $x \in X$ to $y \in X$. Note that c returns infinity if the path is blocked by an obstacle.

Definition 2 ($cost \leftarrow \hat{c}(x, y)$): Calculates a lower-bounded heuristic for the cost of the path that goes from $x \in X$ to $y \in X$, that is, $0 \leq \hat{c}(x, y) \leq c(x, y)$. For BIT* to function properly, $\hat{c}$ must be a valid distance metric [11].

$\hat{c}$ typically will not consider obstacles or have to generate the edge explicitly, making it a fast operation. In this work, $\hat{c}$ is defined using the Euclidean distance as

$$\hat{c}(x, y) \triangleq \|x - y\|.$$

Definition 3 ($cost \leftarrow g_T(x)$): Calculates the cost-to-come from the root node to $x \in X$ through the tree, T. If x is not in the tree, then $g_T(x) = \infty$.

Definition 4 ($cost \leftarrow \hat{g}(x)$): Calculates a lower-bounded heuristic for the cost-to-come of $x \in X$, that is, $0 \leq \hat{g}(x) \leq g_T(x)$.

In this work, $\hat{g}$ is defined using the edge cost heuristic as

$$\hat{g}(x) \triangleq \hat{c}(x_r, x),$$

where $x_r \in X$ is the initial state of the path planning problem.

Definition 5 ($cost \leftarrow \hat{h}(x)$): Calculates a lower bounded heuristic for the cost-to-go from $x \in X$ to the target set, X_t.

In this work, $\hat{h}$ is again defined using the edge cost heuristic as

$$\hat{h}(x) \triangleq \hat{c}(x, x_t),$$

where $x_t \triangleq \arg\min_{x \in V \cap X_t} g_T(x)$ is the end of the best solution found by the planner so far.

Definition 6 ($X_{rand} \leftarrow Sample(m)$): Generates a set of $m \in \mathbb{R}_+$ random samples of the obstacle-free state set, X_{free}[1]. The sampling is an independent, identically, and uniformly distributed (i.i.u.d.) sample of X_{free}[2].

Definition 7 ($X_{near} \leftarrow Near_\rho(x, X_{search})$): Given a state of interest, $x \in X$, and some subset of the state space, $X_{search} \subseteq X$, $X_{near} \subseteq X_{search}$ is the set of states that are within a given edge cost radius[3], $\rho \in \mathbb{R}_+$, of the point $x \in X$, i.e.

$$Near_\rho(x, X_{search}) \triangleq \{v \in X_{search} | c(x, v) \leq \rho\}.$$

Definition 8 ($X_{sol} \leftarrow Solution(x)$): Finds the path through $T \triangleq (V, E)$, $X_{sol} \subset V$, that leads from the root node to $x \in V$. If $x \notin V$, then the empty set $\varnothing$ is returned.

Definition 9 ($x_p \leftarrow Par(x_c)$): Returns the parent node of $x_c \in V$ in the tree $T \triangleq (V, E)$, or $\varnothing$ if x_c is the root node.

Definition 10 ($X_{children} \leftarrow Children(x_p)$): Returns every node from the set V in $T \triangleq (V, E)$ that has $x_p \in V$ as its parent.

Definition 11 ($cost \leftarrow BestValue(\mathcal{Q})$): Finds the element in the generic input queue $\mathcal{Q}$ with the lowest queue cost and returns the queue cost of that element.

Definition 12 ($x \leftarrow PopBest(\mathcal{Q})$): Finds the element in the generic input queue $\mathcal{Q}$ with the lowest queue cost, removes it from the queue, and returns that element.

III. Batch Informed Trees

BIT* functions by iteratively generating batches of samples from the state space and methodically incorporating those new samples into the preexisting search tree. To achieve this, BIT* defines two priority queues, the vertex queue, $\mathcal{Q}_V$, and the edge queue, $\mathcal{Q}_E$. At the beginning of each batch, $\mathcal{Q}_V$ is populated with all nodes that are currently in the search tree, i.e., $\mathcal{Q}_V$ is initialized with V. $\mathcal{Q}_V$ is sorted according to a queue cost consisting of the current cost to traverse the tree to the node plus a heuristic on the cost to get from the node to the target set. Vertices are then iteratively removed from $\mathcal{Q}_V$, and all potential edges that start from that vertex and end at any other sample of the state space are added to the edge queue, $\mathcal{Q}_E$. The queue cost used to order the elements of $\mathcal{Q}_E$ is the cost of traversing the tree to the end of the edge plus a cost heuristic for the cost to get to the target set. Once all potential edges between samples of the state space have been found, the search tree is iteratively updated to include them if doing so will shorten the length of the resulting path. Once both of the queues are empty, a new batch is sampled, and the process is repeated.

Algorithm 1 shows the BIT* algorithm. The inputs are the root node, $x_r \in X$, a set of samples from the target set, $X_{goal} \subseteq X_t$, and the target set, $X_t \subset X_{free}$. Note that while X_t defines the target set, which is possibly continuous and infinite in cardinality, of the path planning problem, X_{goal} is a finite sampling of X_t to aid BIT* in finding a path to X_t. Line 1 initializes the search tree, T, with the root node, x_r, in its vertex set, V, and no edges in its edge set, E. Line 2 initializes the vertex queue, $\mathcal{Q}_V$, and the edge queue, $\mathcal{Q}_E$. $\mathcal{Q}_V$ is used to keep track of vertices that are under consideration for

[1]Because it is computationally expensive to uniformly sample an arbitrary set, all of X is sampled instead and the sample is discarded and re-sampled if it happens to be in the obstacle set.

[2]It is important that the sampling of X_{free} is i.i.u.d. to guarantee asymptotic optimality [2].

[3]In this work ρ is held constant, but many sampling-based algorithms vary ρ as the algorithm runs [2].

Algorithm 1: BIT^*

Input: x_r, X_{goal}, X_t
Output: X_{sol}
1 $V \leftarrow \{x_r\}$; $E \leftarrow \varnothing$; $T \triangleq (V, E)$;
2 $\mathcal{Q}_V \leftarrow V$; $\mathcal{Q}_E \leftarrow \varnothing$; $\mathcal{Q} \triangleq (\mathcal{Q}_V, \mathcal{Q}_E)$;
3 $X_{ncon} \leftarrow X_{goal}$;
4 $X_{new} \leftarrow X_{ncon}$;
5 $V_{exp} \leftarrow \varnothing$; $V_{rewire} \leftarrow \varnothing$;
6 $c_{sol} \leftarrow \min_{v \in V \cap X_t} g_T(v)$;
7 $X_{flags} \triangleq (X_{ncon}, X_{new}, V_{exp}, V_{rewire}, c_{sol})$;
8 **repeat**
9 **if** $\mathcal{Q}_V = \varnothing \wedge \mathcal{Q}_E = \varnothing$ **then** `// End of batch`
`// Prune sub-optimal nodes`
10 $\{T, X_{flags}\} \leftarrow Prune(T, X_{flags})$;
`// Generate new batch of samples`
11 $X_{new} \leftarrow Sample(m)$;
`// Add new samples to queues`
12 $X_{ncon} \xleftarrow{+} X_{new}$;
13 $\mathcal{Q}_V \leftarrow V$;
14 **else if** $BestValue(\mathcal{Q}_V) \leq BestValue(\mathcal{Q}_E)$ **then**
`// Process best vertex available`
15 $\{\mathcal{Q}, X_{flags}\} \leftarrow ExpVertex(T, \mathcal{Q}, X_{flags})$;
16 **else** `// Process best edge available`
17 $\{T, \mathcal{Q}, X_{flags}\} \leftarrow ExpEdge(T, \mathcal{Q}, X_{flags}, X_t)$;
18 **until** *STOP*;
19 **return** $Solution\big(\arg\min_{v_t \in V \cap X_t} g_T(v_t)\big)$;

making potential edges and is organized in terms of the current cost-to-come plus the heuristic cost-to-go of the vertices, i.e.,

$$v \in \mathcal{Q}_V, \qquad g_T(v) + \hat{h}(v).$$

$\mathcal{Q}_E$ is used to track the edges that are being considered in addition to T. $\mathcal{Q}_E$ is organized in terms of the sum of the current cost-to-come of the source vertex of the edge, the heuristic cost of the edge, and the heuristic cost-to-go of the target vertex of the edge, i.e.,

$$(v, x) \in \mathcal{Q}_E, \qquad g_T(v) + \hat{c}(v, x) + \hat{h}(x).$$

As is shown later in the document, the vertex and edge queue costs are always well defined because only the vertices that are in the search tree are added to $\mathcal{Q}_V$ and the source vertex of the edges in $\mathcal{Q}_E$ comes from $\mathcal{Q}_V$ by construction.

Lines 3 through 7 initialize a few sets that are needed to keep track of the state of each vertex. $X_{ncon} \subset X$ is the set of all samples that are not connected to the search tree. Note that X_{ncon} is initialized with the provided samples of the target set, X_{goal}. $X_{new} \subset X$ is the set of samples that are from the most recent batch of samples. $V_{exp} \subseteq V$ is the set of vertices that have been considered for expansion[4]. $V_{rewire} \subset V$ is the set of vertices that have been considered for rewiring[4]. $c_{sol} \in \mathbb{R}_+$ is the cost of the current best solution. X_{flags} is a tuple of sets and values that keep track of the state of the algorithm, and is included in this work to condense the notation used in Algorithm 1.

[4]What is meant by "considered for expansion/rewiring" will become clear in Sections III-B and III-C.

Algorithm 2: $Prune$

Input: $T \triangleq (V, E)$,
$X_{flags} \triangleq (X_{ncon}, X_{new}, V_{exp}, V_{rewire}, c_{sol})$
Output: T, X_{flags}
1 **forall** $v \in V$ **do**
2 **if** $g_T(v) + \hat{h}(v) > c_{sol}$ **then**
`// Remove v from the search`
3 $V \xleftarrow{-} \{v\}$; $E \xleftarrow{-} \{(Par(v), v)\}$;
4 $X_{exp} \xleftarrow{-} \{v\}$; $X_{rewire} \xleftarrow{-} \{v\}$;
`// Add v to the unconnected set`
5 $X_{ncon} \xleftarrow{+} \{v\}$;
6 $X_{ncon} \xleftarrow{-} \left\{x \in X_{ncon} \mid \hat{g}(x) + \hat{h}(x) \geq c_{sol}\right\}$;
7 **return** $\{T, X_{flags}\}$;

The loop on lines 8 through 18 performs the rest of the planning process and ends when a user-defined stopping condition is met[5]. The conditionals on lines 9 and 14 determine if the batch has ended and whether to process a vertex from $\mathcal{Q}_V$ or an edge from $\mathcal{Q}_E$, respectively. Each case is discussed in the following.

A. Generating New Batches

When $\mathcal{Q}_V$ and $\mathcal{Q}_E$ are both empty, it signifies the end of the batch, see line 9 of Algorithm 1. When that happens, the $Prune$ procedure is called on line 10. The $Prune$ procedure removes all vertices from the tree that cannot contribute to the optimal solution given their current connection to the tree, i.e., their cost-to-come plus heuristic cost-to-go value is greater than the current solution cost. Additionally, all vertices that cannot contribute to the optimal solution, regardless of their connection to the tree, that is, whose heuristic cost-to-come plus heuristic cost-to-go value is greater than the current solution cost, are removed from consideration.

The $Prune$ procedure is given in Algorithm 2. The loop starting on line 1 of Algorithm 2 removes all vertices in the search tree that cannot contribute to the optimal solution given their current connection to the tree. Line 2 checks if the vertex being evaluated has the potential to contribute to the optimal solution given its current connection to the search tree. If this check fails, the vertex is removed from the search tree and added to the not connected set on line 5. It is added to the not-connected set because the sample might be able to contribute to the optimal solution with a different connection to the tree. Line 6 removes from consideration all unconnected samples with heuristic cost-to-come plus heuristic cost-to-go value greater than the current best solution. Note that this can be thought of as removing all nodes that fall outside of the "informed set" that Informed RRT* (I-RRT*) defines [3] and, as such, provably cannot contribute to the optimal solution.

After $Prune$ is finished, line 11 of Algorithm 1 generates m *new* samples of the obstacle-free state space, X_{free}. Line 12 adds the new samples, X_{new}, to X_{ncon}. Line 13 adds all

[5]Common stopping conditions include achieving a desirable solution cost or expending the extent of the planning time given.

Algorithm 3: $ExpVertex$

Input: $T \triangleq (V, E), \mathcal{Q} \triangleq (\mathcal{Q}_V, \mathcal{Q}_E)$,
$X_{flags} \triangleq (X_{ncon}, X_{new}, V_{exp}, V_{rewire}, c_{sol})$
Output: $\mathcal{Q}, X_{flags}$

1 $v_b \leftarrow PopBest(\mathcal{Q}_V)$;
// Make edges to unconnected samples
2 **if** $v_b \notin V_{exp}$ **then**
3 $V_{exp} \xleftarrow{+} \{v_b\}$;
4 $X_{near} \leftarrow Near_\rho(v_b, X_{ncon})$;
5 **else** // v_b has been expanded before
6 $X_{near} \leftarrow Near_\rho(v_b, X_{new} \cap X_{ncon})$;
7 $\mathcal{Q}_E \xleftarrow{+} \left\{(v_b, x), x \in X_{near} \middle| \hat{g}(v_b) + \hat{c}(v_b, x) + \hat{h}(x) < c_{sol}\right\}$;
// If v_b has not been rewired before
8 **if** $v_b \notin V_{rewire} \wedge c_{sol} < \infty$ **then**
// Make edges to connected nodes
9 $V_{rewire} \xleftarrow{+} \{v_b\}$;
10 $V_{near} \leftarrow Near_\rho(v_b, V)$;
11 $\mathcal{Q}_E \xleftarrow{+} \{(v_b, w), w \in V_{near} \,|\, (v_b, w) \notin E, \hat{g}(v_b) + \hat{c}(v_b, w) < g_T(w), \hat{g}(v_b) + \hat{c}(v_b, w) + \hat{h}(w) < c_{sol}\}$;
12 **return** $\{\mathcal{Q}, V_{flags}\}$;

Algorithm 4: $ExpEdge$

Input: $T \triangleq (V, E), \mathcal{Q} \triangleq (\mathcal{Q}_V, \mathcal{Q}_E), X_t$,
$X_{flags} \triangleq (X_{ncon}, X_{new}, V_{exp}, V_{rewire}, c_{sol})$
Output: $T, \mathcal{Q}, X_{flags}$

1 $(v_b, x_b) \leftarrow PopBest(\mathcal{Q}_E)$;
2 **if** $g_T(v_b) + \hat{c}(v_b, x_b) + \hat{h}(x_b) \geq c_{sol}$ **then**
// The best edge in the edge queue cannot improve the solution
3 $\mathcal{Q}_E \leftarrow \varnothing$; $\mathcal{Q}_V \leftarrow \varnothing$;
4 **return** $\{T, \mathcal{Q}, X_{flags}\}$;
5 **if** $x_b \in X_{ncon}$ **then** // x_b is unconnected
6 **if** $g_T(v_b) + c(v_b, x_b) + \hat{h}(x_b) < c_{sol}$ **then**
// Add x_b to the tree
7 $X_{ncon} \xleftarrow{-} \{x_b\}$; $V \xleftarrow{+} \{x_b\}$; $E \xleftarrow{+} \{(v_b, x_b)\}$;
8 $\mathcal{Q}_V \xleftarrow{+} \{x_b\}$;
9 **if** $x_b \in X_t$ **then**
10 $c_{sol} \leftarrow \min_{v_{sol} \in V \cap X_t} g_T(v_{sol})$;
11 **else** // x_b is connected
12 **if** $g_T(v_b) + \hat{c}(v_b, x_b) < g_T(x_b)$ **then**
13 **if** $g_T(v_b) + c(v_b, x_b) + \hat{h}(x_b) < c_{sol}$ **then**
14 **if** $g_T(v_b) + c(v_b, x_b) < g_T(x_b)$ **then**
15 $E \xleftarrow{-} \{(Par(x_b), x_b)\}$; $E \xleftarrow{+} \{(v_b, x_b)\}$;
16 $c_{sol} \leftarrow \min_{v_{sol} \in V \cap X_t} g_T(v_{sol})$;
17 **return** $\{T, \mathcal{Q}, X_{flags}\}$;

vertices in the search tree to the vertex queue. This ensures that all vertices in the search tree will be considered when looking for ways to connect the new samples to the search tree.

B. Expanding Vertices

Lines 14 and 15 of Algorithm 1 find potential edges to add to $\mathcal{Q}_E$ from the vertices in $\mathcal{Q}_V$. The condition on line 14 is true until it is impossible for the best vertex in $\mathcal{Q}_V$ to produce an edge of lower heuristic cost than the best edge in $\mathcal{Q}_E$. This can be seen by noting that $\forall v \in \mathcal{Q}_V$ and $\forall x \in X$,

$$g_T(v) + \hat{h}(v) \leq g_T(v) + \hat{c}(v, x) + \hat{h}(x),$$

as $\hat{h}(v)$ is an underestimate of the true cost-to-go of vertex v. Thus, the vertex queue cost, $g_T(v) + \hat{h}(v)$, is a lower bound on the edge queue cost, $g_T(v) + \hat{c}(v, x) + \hat{h}(x)$ of any edge that can be made from that vertex.

$ExpVertex$ removes the lowest-cost vertex in $\mathcal{Q}_V$ and adds edges to $\mathcal{Q}_E$ for every neighbor that might be part of the optimal solution. The $ExpVertex$ procedure is given in Algorithm 3. Line 1 of Algorithm 3 removes the lowest-cost vertex in $\mathcal{Q}_V$, v_b, from $\mathcal{Q}_V$. Lines 2 through 7 add edges to $\mathcal{Q}_E$ that start at v_b and go to unconnected samples that are within a radius, ρ, of v_b. The condition on line 2 checks if this is the first time v_b has been considered for expansion. In that case, all unconnected samples are considered for connection to v_b, see line 4. If it is not the first time v_b has been considered for expansion, only the samples that are new this batch are considered for connection; see line 6. This avoids redundant calculations, as the samples that are not new this batch have already been considered for connection to v_b. Once X_{near} has been found, line 7 adds edges to $\mathcal{Q}_E$ from the set $\{(v_b, x), x \in X_{near}\}$ that have the potential to improve the current solution, i.e., the heuristic cost estimate of using that edge in the solution path is less than the current best solution cost.

Lines 8 through 11 handle the case where v_b might be a better parent for its neighbors than their current parent, that is, rewiring. Note that if BIT* has not found a solution, the condition on line 8 will always be false. This is done to reduce the time it takes to find an initial solution to the problem by skipping any potential tree rewirings. Once the first solution is found, all rewirings that were skipped previously are considered. The condition on line 8 is also false if this is not the first time that v_b has been considered for rewiring. This avoids redundant calculations, as the edges starting at the neighbors of v_b and ending at v_b have already been considered during previous batches by construction. Line 10 finds all vertices in the search tree that are near v_b. Line 11 adds all edges from v_b to elements of V_{near} that are not already part of the tree, have the potential to improve the cost of the neighbor, and have the potential to improve the current solution.

Note that everything done in $ExpVertex$ is done completely with heuristic values and without obstacle checking. This keeps the procedure computationally lightweight and fast.

C. Evaluating Possible Edges

In the case that the best heuristic cost edge possible has been generated from $\mathcal{Q}_V$, the condition on line 14 of Algorithm 1 evaluates to false, and $ExpEdge$ is called. $ExpEdge$, defined in Algorithm 4, removes the most promising edge from $\mathcal{Q}_E$ and considers it for addition to the search tree. $ExpEdge$ adds

the edge to the tree if it is determined that the edge has the potential to contribute to the optimal solution.

Line 1 of Algorithm 4 removes the lowest queue cost edge, (v_b, x_b), from $\mathcal{Q}_E$. Note that $v_b \in V$ by construction, but x_b may not be part of the tree. Line 2 checks if there is a chance that the edge (v_b, x_b) will improve the current solution. Note that if (v_b, x_b) cannot heuristically contribute to a better solution than the existing solution, then no edge in $\mathcal{Q}_E$ can contribute to the optimal solution. For this reason, $\mathcal{Q}_E$ and $\mathcal{Q}_V$ are cleared on line 3, triggering a new batch of samples to be drawn.

Before adding (v_b, x_b) to the tree, checks are performed to ensure that the operation is beneficial. These checks vary slightly depending on whether the edge is connecting two vertices that are already part of the tree or not. In the case that (v_b, x_b) is connecting to an unconnected sample, only checks to ensure (v_b, x_b) can contribute to the optimal solution are needed. Otherwise, additional checks are also needed to ensure that the cost of the target vertex, x_b, is reduced.

Line 5 checks if x_b is already in the tree. If x_b is not part of the search tree, line 6 checks if connecting x_b to the tree through v_b can improve the current solution. If it can, x_b is added to the search tree with v_b as its parent. Lines 9 and 10 check if x_b is in the target set and updates c_{sol} if so.

If x_b is already part of the search tree at line 5, additional checks are performed before adding the edge under consideration to the tree. Line 12 checks if connecting x_b through v_b can improve the cost of x_b. Line 13 checks that the edge under consideration can improve the current solution. Line 14 checks that connecting x_b through v_b will improve the cost of x_b. If all checks pass, x_b is rewired to have v_b as its parent, and the current solution cost is updated if needed.

IV. Demonstration

Figures 1 through 4 graphically show how BIT* operates over three batches of samples. The upper right corner of each figure shows the vertices and edges currently in $\mathcal{Q}_V$ and $\mathcal{Q}_E$ at that step of the algorithm. Each figure shows the state of the search tree and queues after the operations described in its caption are complete. Red is used to emphasize the vertex or edge that is being operated on. In this example, each batch consists of five samples, i.e., $m = 5$.

Figure 1 shows batch 0. This is the part of the planning process where the only node in $\mathcal{Q}_V$ is the root node and no samples have been made from the state space. This part of the algorithm checks for the trivial case in which it is possible to directly connect the root node to the target set.

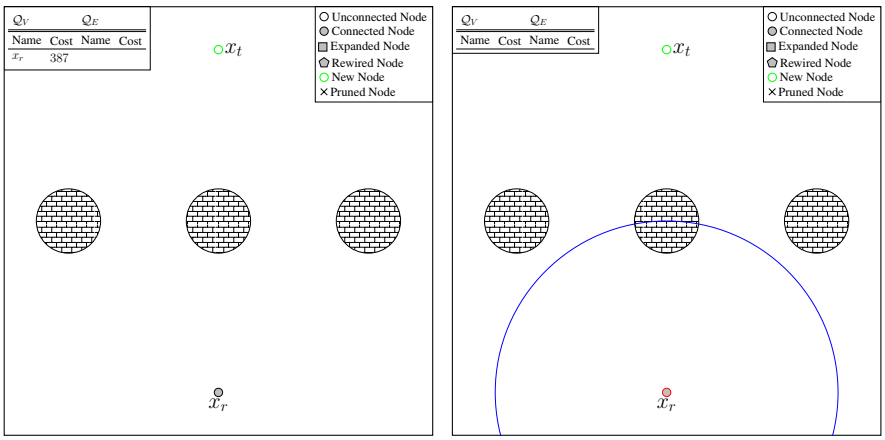
Figure 2 shows the process of generating the first batch of samples and starting to build a search tree. Note that when batch 1 completes, a path to the target set has not yet been found. This is a common occurrence in BIT* where the search tree is unable to grow much until the sample density in X_{free} reaches a critical mass after a few batches.

Figure 3 shows how the second batch of samples is incorporated into the search tree. By the end of the batch, a path to the target set has been found and the "informed ellipse" is defined. This enables pruning to be performed before batch 3 begins.

Figure 4 shows how a new batch of samples is generated within the informed ellipse and used to refine the current solution. Note that as the solution length reduces in the third batch, the size of the informed ellipse also shrinks. This leads to more nodes being pruned and the search process becoming more focused on the area that can improve the current solution.

V. Simulation

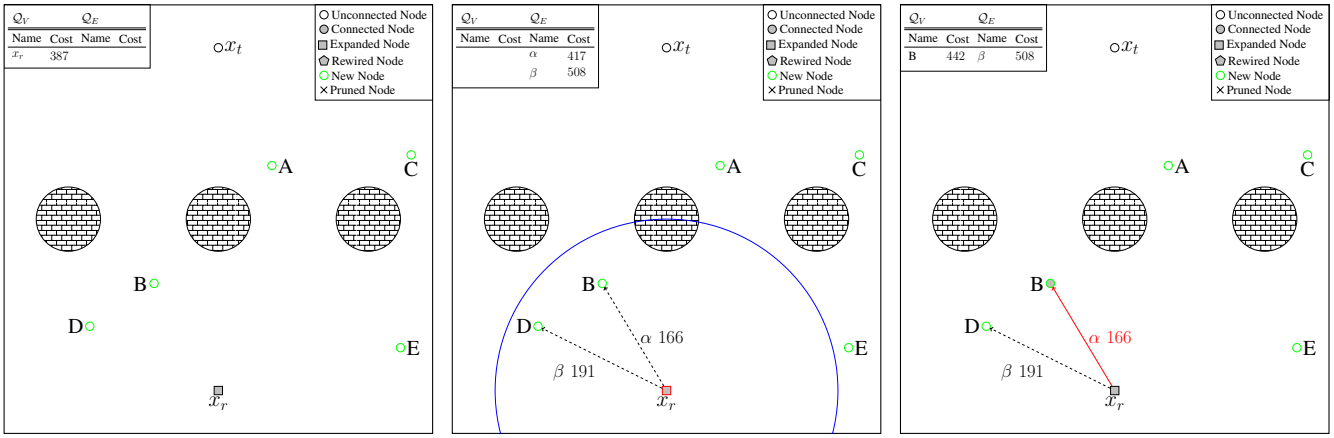
Simulation results are now presented to demonstrate the effectiveness of the BIT* algorithm. Comparisons are made



(a) Lines 1 through 7 of Algorithm 1 initialize the sets to have only the root node in $\mathcal{Q}_V$.

(b) x_r is the vertex with the lowest cost in $\mathcal{Q}_V$, so it is removed from the queue and considered for expansion. x_r does not have any neighbors, so no edges are added to $\mathcal{Q}_E$.

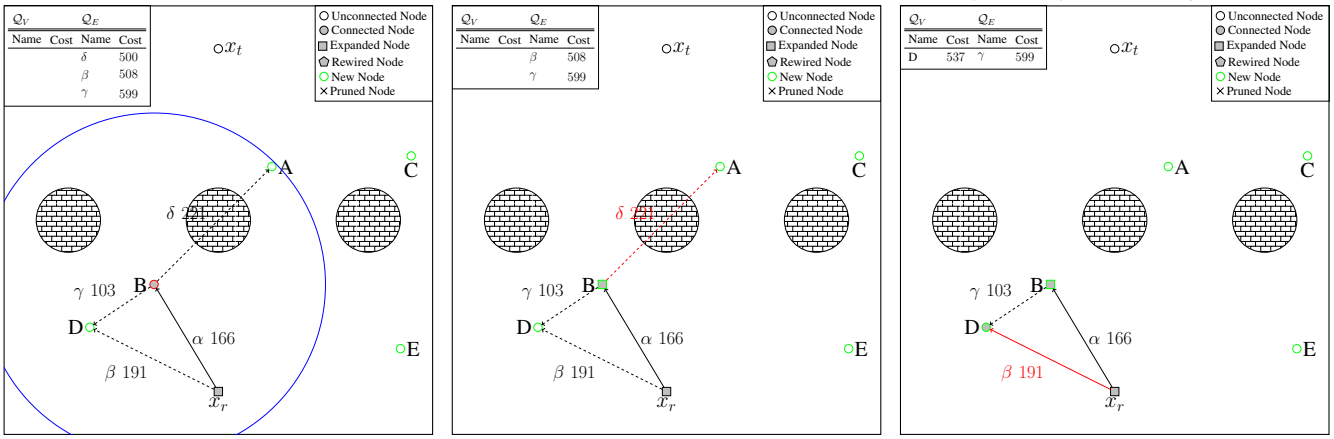
Fig. 1: Batch 0 of BIT*.



(a) $\mathcal{Q}_E$ and $\mathcal{Q}_V$ are empty, so a new batch of samples is made. Pruning is performed, but it has no effect until a solution is found. Samples A through E are generated. $\mathcal{Q}_V$ is filled with all connected nodes.

(b) The lowest queue cost vertex, x_r, is popped from $\mathcal{Q}_V$ and used for expansion in $ExpVertex$. Edges α and β pass the heuristic cost tests and are added to $\mathcal{Q}_E$.

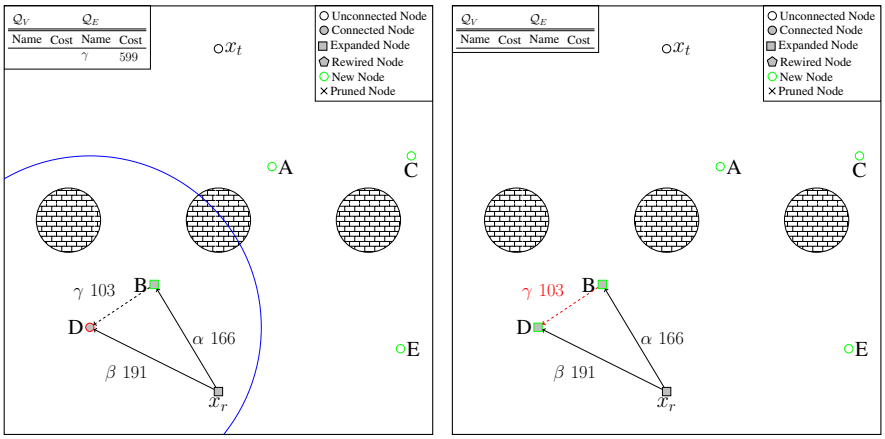
(c) The vertex queue is empty so the $ExpEdge$ procedure starts. α is the lowest cost edge in $\mathcal{Q}_E$, so it is removed from $\mathcal{Q}_E$ and, after passing all validity and cost tests, added to the tree. B is now connected to the tree and, as such, added to $\mathcal{Q}_V$.



(d) The queue cost of node B is less than the queue cost of edge β, so the $ExpVertex$ procedure is called. B is removed from $\mathcal{Q}_V$, and the edges γ and δ are added to $\mathcal{Q}_E$. Note that an edge is not made from B to x_r because that edge fails the heuristic cost tests.

(e) $\mathcal{Q}_V$ is empty so we move to $ExpEdge$. Edge δ is removed from $\mathcal{Q}_E$. δ passes the heuristic cost tests but fails the true cost tests because of the obstacle it passes through. δ is not added to the tree.

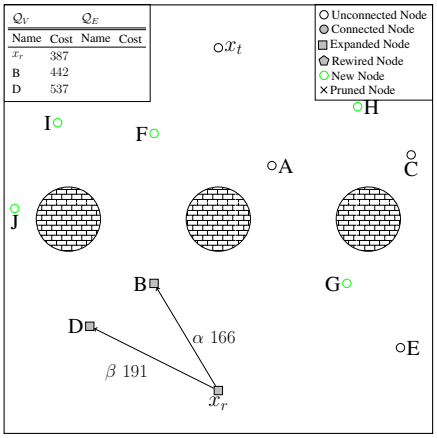
(f) Edge β is removed from $\mathcal{Q}_E$ and, after passing all tests, added to the tree. Node D is now connected to the tree and added to $\mathcal{Q}_V$.



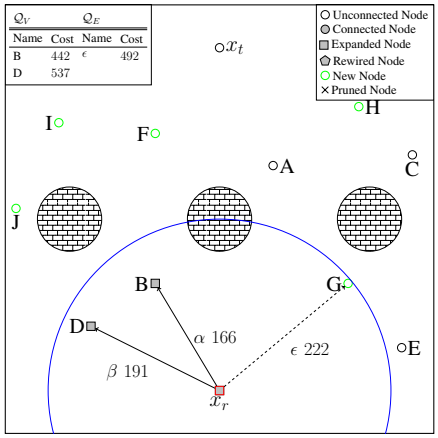
(g) The cost of node D is less then γ so $ExpVertex$ begins. Node D is removed from $\mathcal{Q}_V$ and used for expansion. However, no solution has been found, so connected neighbors are not considered and D does not have unconnected neighbors.

(h) With $\mathcal{Q}_V$ empty we move back $ExpEdge$. Edge γ is removed from $\mathcal{Q}_E$ but fails the heuristic cost tests, so it is not added to the tree.

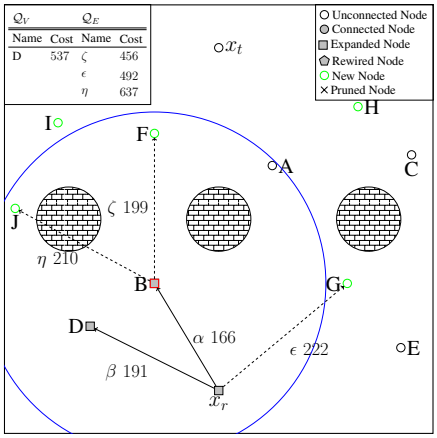
Fig. 2: Batch 1 of BIT*.



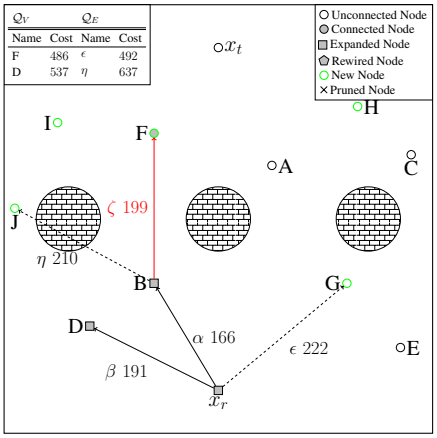
(a) $\mathcal{Q}_E$ and $\mathcal{Q}_V$ are empty, so a new batch of samples is made. Samples F through J are generated. $\mathcal{Q}_V$ is filled with all connected nodes.



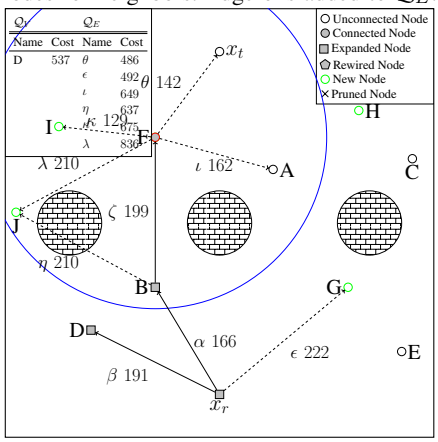
(b) In $ExpVertex$, x_r is removed from $\mathcal{Q}_V$. Because x_r is still part of the expanded set from last batch, we only search over the new nodes for neighbors. Edge ϵ is added to $\mathcal{Q}_E$.



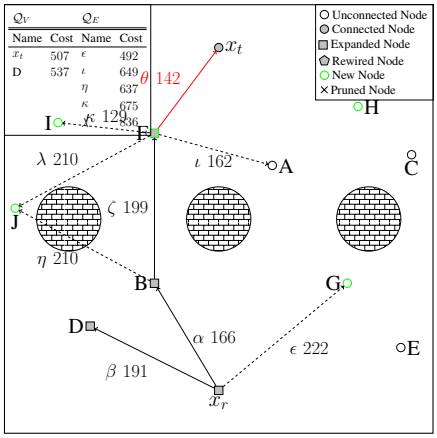
(c) Node B is removed from $\mathcal{Q}_V$. Node B has already been expanded, so only edges to new nodes are considered. Edges η and ζ are added to $\mathcal{Q}_E$.



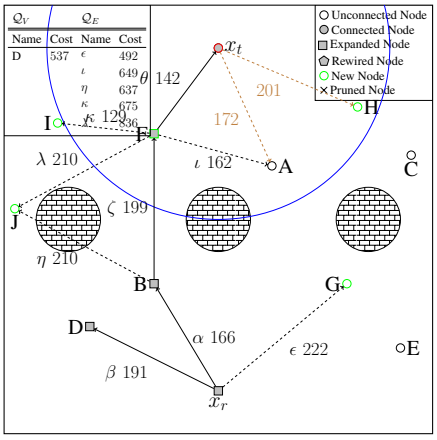
(d) Edge ζ has a lower heuristic cost then node D so $ExpEdge$ starts. Edge ζ is removed from $\mathcal{Q}_E$ and, after passing all tests, added to the tree. Node F is now connected to the tree and added to $\mathcal{Q}_V$.



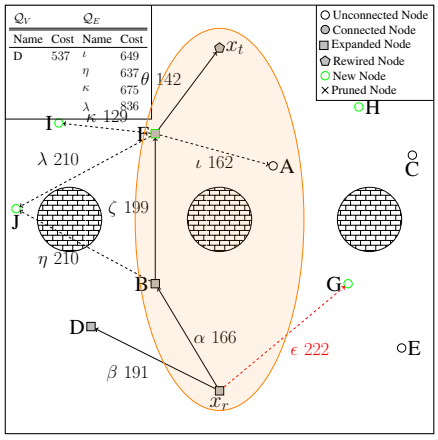
(e) In $ExpVertex$, node F is removed from $\mathcal{Q}_V$. Node F has not been expanded, so edges to all unconnected nodes are considered. Edges λ, κ, θ, and ι are added to $\mathcal{Q}_E$.



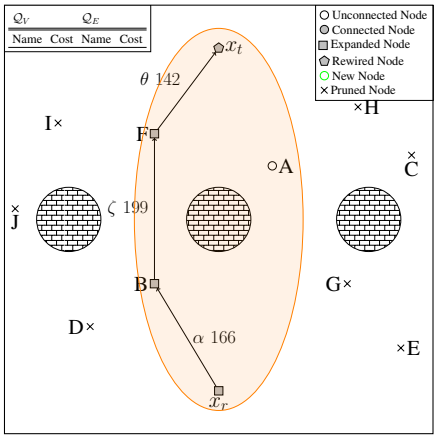
(f) Edge θ has a lower heuristic cost then node D so $ExpEdge$ is called. Edge θ is removed from $\mathcal{Q}_E$ and, after passing all tests, added to the tree. Node x_t is now part of the tree and added to $\mathcal{Q}_V$.



(g) Node x_t has a lower heuristic cost then edge ϵ so $ExpVertex$ is used. Node x_t is removed from $\mathcal{Q}_V$. Node x_t has not been expanded, so edges to all near nodes are considered. Edges from x_t to nodes A and H fail to pass the heuristic cost test and, as such can not improve the current solution, and are not added to $\mathcal{Q}_E$.

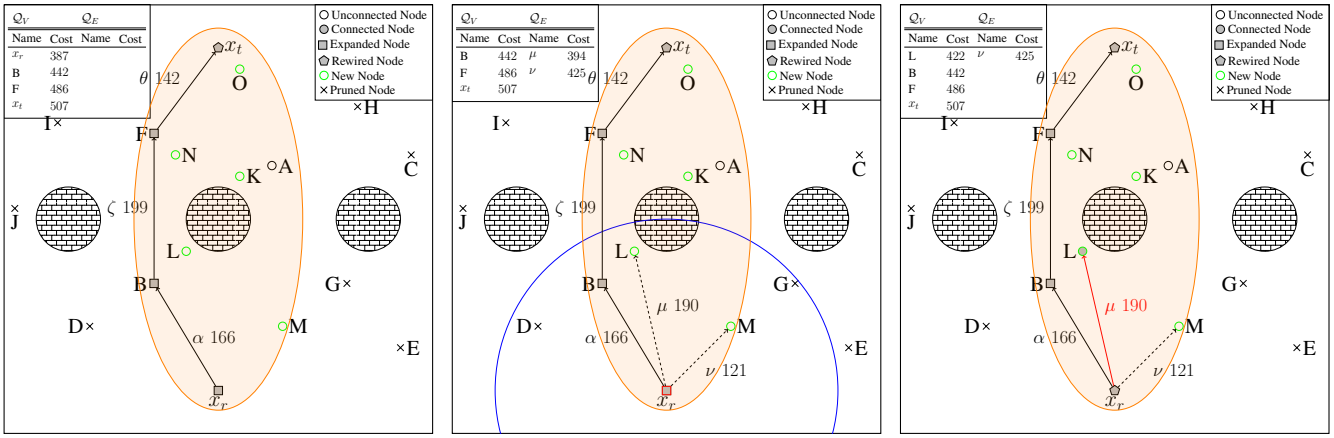


(h) In $ExpEdge$, edge ϵ is removed from $\mathcal{Q}_E$ and considered for addition to the tree. Edge ϵ fails the primary heuristic cost check, meaning that ϵ has no chance of improving the solution. The rest of the nodes and edges in $\mathcal{Q}_V$ and $\mathcal{Q}_E$ also can not help the solution, so both queues are cleared. Note the orange ellipse that shows the informed set of states, and how G falls outside of the ellipse. This is a visual way of seeing why node G was not added to the tree.



(i) Before generating batch 3, all nodes in the tree or unconnected are checked to make sure they fall within the informed ellipse. All that fall outside of the ellipse are pruned and not considered moving forward.

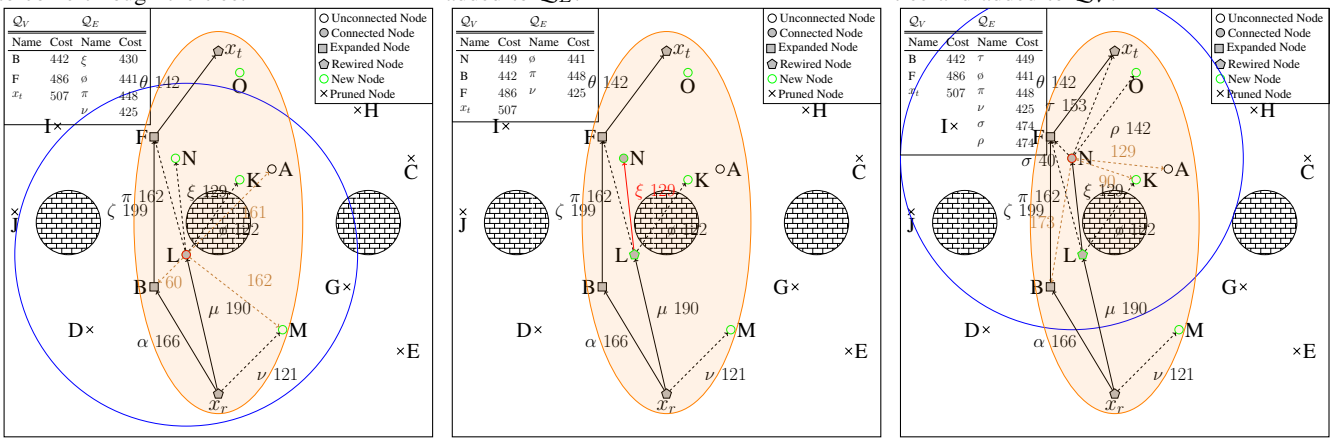
Fig. 3: Batch 2 of BIT*.



(a) Nodes K through O are sampled from the informed ellipse. All connected nodes are added to $\mathcal{Q}_V$. Note that node F is before x_t because ties are broken based on true cost-to-come through the tree.

(b) In $ExpVertex$, node x_r is removed from $\mathcal{Q}_V$. Node x_r has been expanded but not rewired, so only edges to new and connected nodes are considered. Edges μ and ν are added to $\mathcal{Q}_E$.

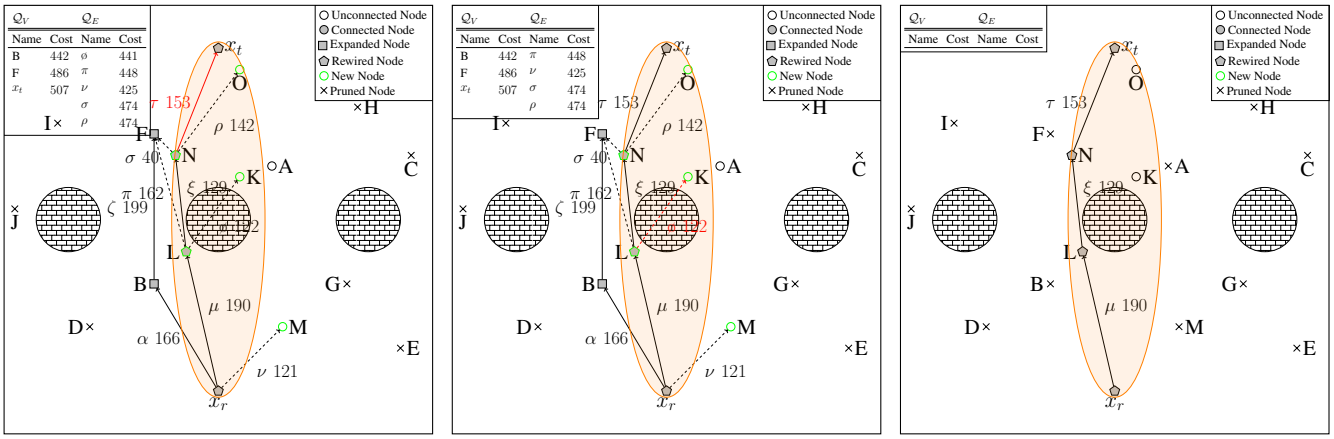
(c) Edge μ has a lower heuristic cost than node B, so we move to $ExpEdge$. Edge μ is removed from $\mathcal{Q}_E$ and, after passing all tests added to the tree. Node L is now part of the tree and added to $\mathcal{Q}_V$.



(d) In $ExpNode$, node L is removed from $\mathcal{Q}_V$. This is the first time Node L is used for expansion, so all non-pruned nodes are under consideration. Edges ξ, $\emptyset$, and π are added to $\mathcal{Q}_E$. Edges from L to M, B, and A fail the heuristic cost test and are not added to $\mathcal{Q}_E$.

(e) Edge ξ has a lower heuristic cost than node B, so we move to $ExpEdge$. Edge ξ is removed from $\mathcal{Q}_E$ and, after passing all tests, added to the tree. Node N is now a part of the tree and added to $\mathcal{Q}_V$.

(f) In $ExpNode$, node N is removed from $\mathcal{Q}_V$. Edges ρ, σ, and τ are added to $\mathcal{Q}_E$. Edges from node N to nodes A, K, and B fail their heuristic cost tests. Note that σ comes before ρ because ties are broken based on true cost-to-come.



(g) In $ExpEdge$, edge τ is removed from $\mathcal{Q}_E$ and, after passing all tests, added to the tree. In order for node x_t to only have one parent, edge θ is removed from the tree. Note that this changes the current solution cost and the informed ellipse.

(h) Edge $\emptyset$ is removed from $\mathcal{Q}_E$ and considered for addition to the tree. Edge $\emptyset$ fails the primary heuristic cost tests and is not added to the tree. $\mathcal{Q}_V$ and $\mathcal{Q}_E$ cleared and batch 2 ends.

(i) Before generating batch 4, all nodes either in the tree or unconnected are checked to make sure they fall within the informed ellipse. All that fall outside of the ellipse are pruned and not considered moving forward.

Fig. 4: Batch 3 of BIT*.

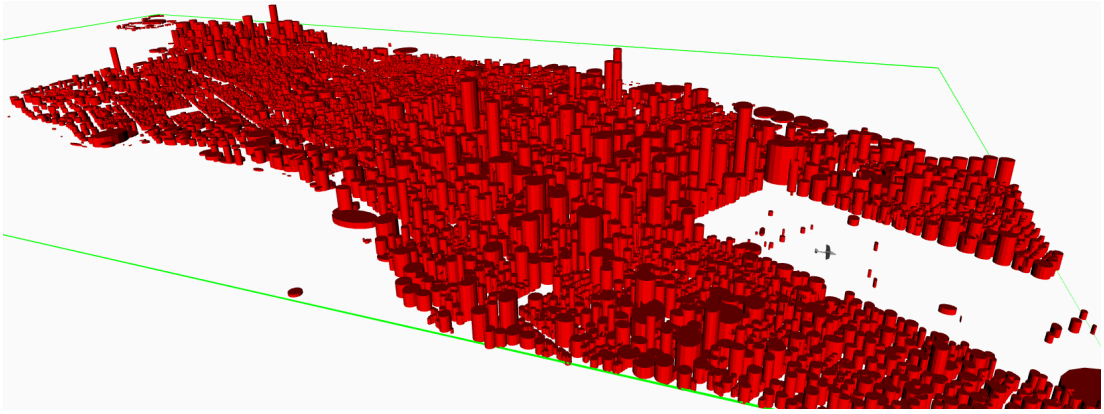


Fig. 5: The UAV simulation with Manhattan's buildings shown in red.

Fig. 6: The resulting path from running BIT* in the Manhattan world shown in red.

between BIT* and RRT* planning with straight-lines.

A. Simulation Details

To test the capabilities of BIT*, it is used to plan the routes of a simulated UAV through an urban environment. The UAV simulation is shown in Figure 5. The buildings in the UAV simulation are modeled after the real buildings in Manhattan, New York. The placement and height of the buildings are obtained from the New York Open Data project [12]. The initial position of the UAV is located in Central Park. Maintaining an altitude of 10 meters, the UAV plans a path through the buildings to the goal location on Governors Island.

During the planning of a path, obstacles are represented with an occupancy grid with each pixel corresponding to ten square centimeters. Every building that is taller than 10 m is considered an obstacle in the occupancy grid. Figure 6 shows the resulting occupancy grid with black representing obstacles. The occupancy grid covers a $10.7\ km \times 5.65\ km$ area of Manhattan. The initial location of the UAV in the coordinate frame used for this problem is $x_r = \begin{bmatrix} 0 & 0 \end{bmatrix}\ km$, with the UAV orientation defined in the direction of the target set. The target set, X_t, is a circle of radius $0.001\ m$ centered at $\begin{bmatrix} -9 & -3.8 \end{bmatrix}\ km$. The samples of the target set that are provided to BIT*, X_{goal}, is the singleton set of the center of X_t. The neighborhood search radius, ρ, is $500\ m$. Paths are generated and checked for obstacles four times per meter of path length. When using BIT*, the batch size, m, is set to 1500 samples.

When using RRT*, there are three additional parameters, η, α, and b_t, which are described in [10], but only values are

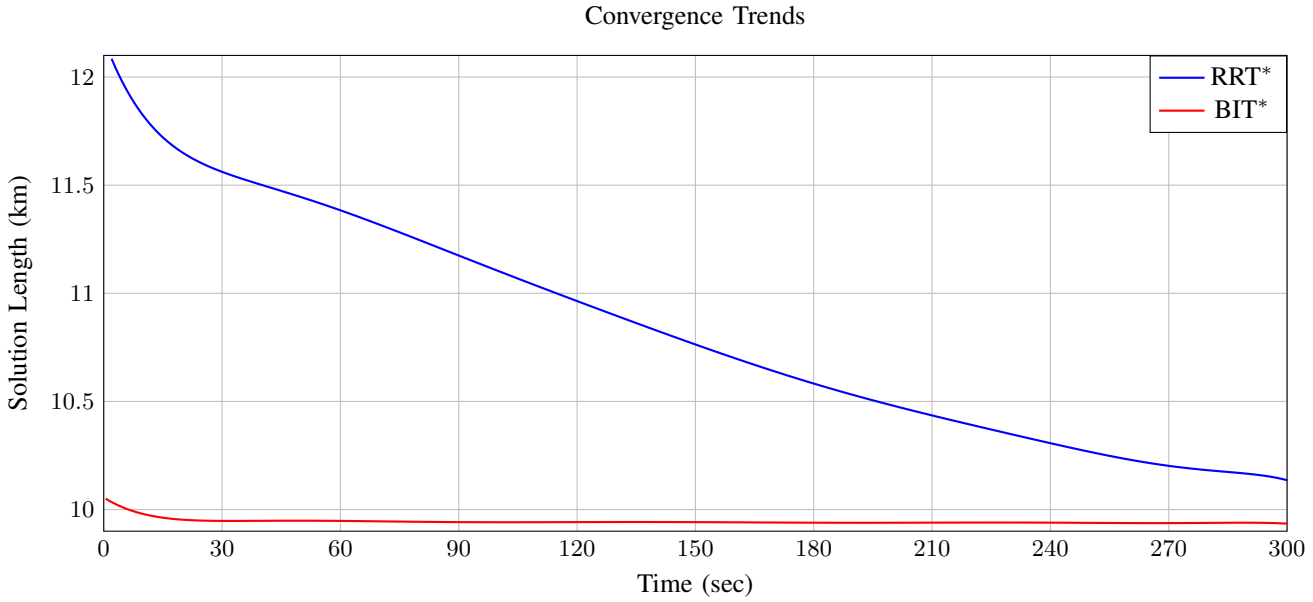


Fig. 7: The convergence plots of the Manhattan world over 5 minutes. RRT* and BIT* planning with straight-line motion primitives are shown in blue and red, respectively.

given here for brevity. The steering constant, η, is 500 m. The maximum number of neighbors to search, α, is 100 neighbors. The check target period, b_t, is once out of every 50 samples.

The results are gathered using the Open Motion Planning Library (OMPL) [13]. The simulation code can be found in our open-source repository https://gitlab.com/utahstate/robotics/fillet-rrt-star. As the sampling is random, each simulation consists of over 100 individual simulations with the average results being presented. The results were gathered on an AMD Ryzen™ Threadripper™ 3970X processor. The convergence plots were made by fitting a 15$^{\text{th}}$-order polynomial using a least-squares fitting algorithm as described in [14].

A least-squares approach is used, as the sampling times for path length are not uniform across all simulations, and not all simulations find the initial path at the same time. Note that while the path length for any one run will be monotonically decreasing with time, the least-squares fitted plot does not always have the same monotonic property. The reason is that a particular run may not find a solution until well after other runs, and the initial solution it finds may be much larger than the current solution of the other runs, effectively causing the average to increase at the time the run first produces path length data.

B. Results

The convergence results from benchmarking RRT* and BIT* in the Manhattan environment are shown in Figure 7. RRT* and BIT* are shown in blue and red respectively. Clearly BIT* outperforms RRT* in terms of rapidly converging to a near-optimal value. BIT* initially finds a solution much shorter than RRT* and proceeds to converge to near optimality by the time 30 seconds have passed. This comes from BIT*'s use of cost-to-come and cost-to-go heuristics to focus their search efforts in directions that are most likely to improve the solution cost. RRT*, on the other hand, starts with a long initial solution. Despite converging for five minutes, the solution that RRT* produces is still substantially longer than that of BIT*.

References

[1] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[2] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011. [Online]. Available: https://doi.org/10.1177/0278364911406761

[3] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2997–3004.

[4] J. Nasir, F. Islam, U. Malik, Y. Ayaz, O. Hasan, M. Khan, and M. Muhammad, "Rrt*-smart: A rapid convergence implementation of rrt*," *International Journal of Advanced Robotic Systems*, vol. 10, 2013.

[5] I. Noreen, A. Khan, and Z. Habib, "Optimal path planning using rrt* based approaches: A survey and future directions," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 11, 2016. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2016.071114

[6] K. Yang, S. Moon, S. Yoo, J. Kang, N. L. Doh, H. B. Kim, and S. Joo, "Spline-based rrt path planner for non-holonomic robots," *Journal of Intelligent & Robotic Systems*, vol. 73, no. 1-4, pp. 763–782, 2014a.

[7] I.-B. Jeong, S.-J. Lee, and J.-H. Kim, "Quick-rrt*: Triangular inequality-based implementation of rrt* with improved initial solution and convergence rate," *Expert Systems with Applications*, vol. 123, pp. 82–90, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417419300326

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

[9] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Batch informed trees (bit*): Informed asymptotically optimal anytime search," *The International Journal of Robotics Research*, vol. 39, no. 5, pp. 543–567, 2020. [Online]. Available: https://doi.org/10.1177/0278364919890396

[10] J. Swedeen, G. Droge, and R. Christensen, "Fillet-based rrt*: A rapid convergence implementation of rrt* for curvature constrained vehicles," 2023. [Online]. Available: https://arxiv.org/abs/2302.11648

[11] T. K. Moon and W. C. Stirling, *Mathematical methods and algorithms for signal processing*. Prentice hall Upper Saddle River, NJ, 2000, vol. 1.

[12] "Building footprints," May 2016. [Online]. Available: https://data.cityofnewyork.us/Housing-Development/Building-Footprints/nqwf-w8eh

[13] M. Moll, I. A. Sucan, and L. E. Kavraki, "Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization," *IEEE Robotics Automation Magazine*, vol. 22, no. 3, pp. 96–102, 2015.

[14] W. N. Venables and B. D. Ripley, *Modern Applied Statistics with S*, 4th ed. New York: Springer, 2002, iSBN 0-387-95457-0. [Online]. Available: https://www.stats.ox.ac.uk/pub/MASS4/

APPENDIX B

Linear Covariance Analysis Derivation

In this appendix, some general dynamic and measurement update equations for the Kalman filter and linear covariance (LinCov) analysis are derived. The notation follows that of [6, 75, 76].

B.1 Definitions

The general expression for the dynamics of the truth states of a vehicle can be expressed with the non-linear differential equation

$$\dot{x}(t) \triangleq f(t, x(t), u(t), w(t)), \tag{B.1}$$

where $t \in \mathbb{R}$ is the current time, $x(t) \in \mathbb{R}^n$ is the truth state vector, $u(t) \in \mathbb{R}^{n_u}$ is the control vector, and $w(t) \in \mathbb{R}^{n_w}$ is the process noise on the truth state dynamics. w is assumed to have the following properties $\forall t, t_1 \in \mathbb{R}$

$$E[w(t)] \triangleq 0 \qquad \text{and} \qquad E\big[w(t)w^T(t_1)\big] \triangleq S_w(t)\delta(t - t_1), \tag{B.2a}$$

where $S_w(t) \in \mathcal{S}_+^{n_w}$ is the power spectral density of $w(t)$ and $\delta : \mathbb{R} \mapsto \{0, 1\}$ is the Kronecker delta function.

Observations of the truth state are made through discrete non-inertial, continuous non-inertial, and continuous inertial measurements. The continuous inertial measurements, $\tilde{y}(t) \in \mathbb{R}^{n_{\tilde{y}}}$, (e.g. accelerometer and gyroscope measurements) are given as a function of time, the truth state, commands, and corrupting continuous noise, $\eta(t) \in \mathbb{R}^{n_\eta}$,

$$\tilde{y}(t) \triangleq c(t, x(t), u(t), \eta(t)), \tag{B.3}$$

where η satisfies the following properties $\forall t, t_1 \in \mathbb{R}$

$$E[\eta(t)] \triangleq 0 \qquad \text{and} \qquad E\big[\eta(t)\eta^T(t_1)\big] \triangleq S_\eta(t)\delta(t-t_1), \tag{B.4a}$$

where $S_\eta(t) \in \mathcal{S}_+^{n_\eta}$ is the power spectral density matrix of $\eta(t)$. Discrete non-inertial measurements become available at discrete points in time, $t_k, k \in \mathbb{Z}$. Let the set of all discrete non-inertial measurements be denoted by $\mathbb{DM}$ and of cardinality $n_{\mathbb{DM}} \triangleq |\mathbb{DM}|$. The discrete non-inertial measurements are expressed as

$$\tilde{z}_j(t_k) \triangleq h_j(t_k, x(t_k)) + \nu_j(t_k), \tag{B.5}$$

where $j \in \mathbb{DM}$ signifies the different types of discrete measurements used, $\nu_j(t_k) \in \mathbb{R}^{n_{\nu_j}}$ is discrete measurement noise, and $\tilde{z}_j(t_k) \in \mathbb{R}^{n_{\nu_j}}$. All ν_js are assumed to satisfy the following properties $\forall k, k_1 \in \mathbb{Z}$

$$E\big[\nu_j(t_k)\big] \triangleq 0 \qquad \text{and} \qquad E\big[\nu_j(t_k)\nu_j^T(t_{k_1})\big] \triangleq R_{\nu_j}(t_k)\delta_{k,k_1}, \tag{B.6a}$$

where $R_{\nu_j}(t_k) \in \mathcal{S}_+^{n_{\nu_j}}$ is the covariance of $\nu_j(t_k)$ and $\delta : [\mathbb{Z}]^2 \mapsto \{0,1\}$ is the Dirac delta function. Continuous non-inertial measurements are continuously available. Let the set of all continuous non-inertial measurements be denoted by $\mathbb{CM}$ and of cardinality $n_{\mathbb{CM}} \triangleq |\mathbb{CM}|$. The continuous non-inertial measurements are expressed as

$$\tilde{z}_i(t_k) \triangleq h_i(t_k, x(t_k)) + \nu_i(t_k), \tag{B.7}$$

where $i \in \mathbb{CM}$ signifies the different types of continuous measurements used, $\nu_i(t_k) \in \mathbb{R}^{n_{\nu_i}}$ is continuous measurement noise, and $\tilde{z}_i(t_k) \in \mathbb{R}^{n_{\nu_i}}$. All ν_is are assumed to satisfy the following properties $\forall t, t_1 \in \mathbb{R}$

$$E[\nu_i(t)] \triangleq 0 \qquad \text{and} \qquad E\big[\nu_i(t)\nu_i^T(t_1)\big] \triangleq S_{\nu_i}(t)\delta(t-t_1), \tag{B.8a}$$

where $S_{\nu_i}(t) \in \mathcal{S}_+^{n_{\nu_i}}$ is the power spectral density matrix of $\nu_i(t)$.

The navigation state dynamics are a function of time, the navigation state, $\hat{x}(t) \in \mathbb{R}^{\hat{n}}$, the inertial measurements, and the continuous non-inertial measurements.

$$\dot{\hat{x}}(t) \triangleq \hat{f}(t, \hat{x}(t), \tilde{y}(t)) + \sum_{i \in \mathbb{CM}} \hat{K}_i(t)\rho_i(t) \tag{B.9}$$

A few other quantities must be defined when using non-inertial measurements to update $\hat{x}$. First, the estimate of what the measurement would be without noise is given by

$$\hat{\tilde{z}}_i(t) \triangleq \hat{h}_i(t, \hat{x}(t)), i \in \mathbb{DM} \cup \mathbb{CM}, \tag{B.10}$$

where $\hat{\tilde{z}}_i(t) \in \mathbb{R}^{n_{\nu_i}}$. The residual, $\rho_i(t) \in \mathbb{R}^{n_{\nu_i}}$, is the difference between the true measurement and the predicted measurement, i.e.,

$$\rho_i(t) \triangleq \tilde{z}_i(t) - \hat{\tilde{z}}_i(t), i \in \mathbb{DM} \cup \mathbb{CM}. \tag{B.11}$$

Now, the measurement update equation can be stated as

$$\hat{x}\big(t_k^+\big) \triangleq \hat{x}\big(t_k^-\big) + \hat{K}_j\big(t_k^-\big)\rho_j\big(t_k^-\big), i \in \mathbb{DM}, \tag{B.12}$$

where $\hat{x}\big(t_k^-\big)$ is the "a priori" estimate of x at time t_k, $\hat{x}\big(t_k^+\big)$ is the "a posteriori" estimate of x at time t_k, and $\hat{K}_j\big(t_k^-\big)$ is the Kalman gain for measurement j, at time t_k. The Kalman gain is given as

$$\begin{aligned}
\hat{K}_j\big(t_k^-\big) &= \hat{P}\big(t_k^-\big)_{\hat{x}}\hat{H}_j^T\big(t_k^-\big)\Big(_{\hat{x}}\hat{H}_j\big(t_k^-\big)\hat{P}\big(t_k^-\big)_{\hat{x}}\hat{H}_j^T\big(t_k^-\big) + R_{\nu_j}(t_k)\Big)^{-1}, j \in \mathbb{DM} \\
\hat{K}_i(t) &= \hat{P}(t)_{\hat{x}}\hat{H}_i^T(t)\Big(_{\hat{x}}\hat{H}_i(t)\hat{P}(t)_{\hat{x}}\hat{H}_i^T\big(t_k^-\big) + S_{\nu_i}(t)\Big)^{-1}, i \in \mathbb{CM}
\end{aligned} \tag{B.13}$$

or equivalently [77]

$$\begin{aligned}\hat{K}_j(t_k^-) &= \hat{P}(t_k^+)_{\hat{x}}\hat{H}_j^T(t_k^-)R_{\nu_j}^{-1}(t_k), j \in \mathbb{DM} \\ \hat{K}_i(t) &= \hat{P}(t)_{\hat{x}}\hat{H}_i^T(t)S_{\nu_i}^{-1}(t), i \in \mathbb{CM}\end{aligned} \tag{B.14}$$

where

$${}_{\hat{x}}\hat{H}_j(t) \triangleq \left.\frac{\partial \hat{h}_j(t,\hat{x})}{\partial \hat{x}}\right|_{\hat{x}=\hat{x}(t)}, j \in \mathbb{DM} \cup \mathbb{CM}. \tag{B.15}$$

The error free navigation state, $x_n(t) \in \mathbb{R}^{\hat{n}}$, can be found from the truth state to navigation state mapping, $m : \mathbb{R}^n \mapsto \mathbb{R}^{\hat{n}}$, i.e.,

$$x_n(t) \triangleq m(x(t)). \tag{B.16}$$

Finally, the control vector $u(t) \in \mathbb{R}^{n_u}$ is a function of time, the navigation state, and the inertial measurements, i.e.,

$$u(t) \triangleq g(t, \hat{x}(t)). \tag{B.17}$$

Note that all noise is assumed to be uncorrelated.

B.2 Error State Covariance Propagation

Before giving the equations related to the estimated error state covariance matrix, $\hat{P}(t) \in \mathcal{S}_+^{n_e \times n_e}$, we must define the error state, $e(t) \in \mathbb{R}^{n_e}$, as

$$e(t) \triangleq m(x(t)) - \hat{x}(t). \tag{B.18}$$

Then the covariance of the error state can be stated as

$$\hat{P}(t) \triangleq E\big[e(t)e^T(t)\big]. \tag{B.19}$$

The error state covariance discrete measurement update equation is given by

$$\begin{aligned}\hat{P}(t_k^+) = &\Big(I_{n_e} - \hat{K}_j(t_k^-)_{\hat{x}}\hat{H}_j^T(t_k^-)\Big)\hat{P}(t_k^-)\Big(I_{n_e} - \hat{K}_j(t_k^-)_{\hat{x}}\hat{H}_j^T(t_k^-)\Big)^T \\ &+ \hat{K}_j(t_k^-)R_{\nu_j}(t_k)\hat{K}_j^T(t_k^-), j \in \mathbb{DM}.\end{aligned} \tag{B.20}$$

The error state covariance dynamics will now be derived. First the truth state dynamics (B.1), navigation state dynamics (B.9), inertial measurement equation (B.3), and control function (B.17) are linearized as

$$\dot{x}(t) \approxeq {}_xF(t)x(t) + {}_uF(t)u(t) + {}_wF(t)w(t), \tag{B.21a}$$

$$\dot{\hat{x}}(t) \approxeq {}_{\hat{x}}\hat{F}(t)\hat{x}(t) + {}_{\tilde{y}}\hat{F}(t)\tilde{y}(t) + \sum_{i\in\mathbb{CM}} \hat{K}_i(t)\Big({}_xH_i(t)x(t) - {}_{\hat{x}}\hat{H}_i(t)\hat{x}(t) + \nu_i(t)\Big), \tag{B.21b}$$

$$\tilde{y}(t) \approxeq {}_xC(t)x(t) + {}_uC(t)u(t) + {}_\eta C(t)\eta(t), \text{ and} \tag{B.21c}$$

$$u(t) \approxeq {}_{\hat{x}}G(t)\hat{x}(t), \tag{B.21d}$$

where

$${}_xF(t) \triangleq \left.\frac{\partial f(t,x,u,w)}{\partial x}\right|_{x=x(t),u=u(t),w=w(t)}, \quad {}_{\hat{x}}G(t) \triangleq \left.\frac{\partial g(t,\hat{x},\tilde{y})}{\partial \hat{x}}\right|_{\hat{x}=\hat{x}(t),\tilde{y}=\tilde{y}(t)}, \tag{B.22}$$

$${}_uF(t) \triangleq \left.\frac{\partial f(t,x,u,w)}{\partial u}\right|_{x=x(t),u=u(t),w=w(t)}, \quad {}_xH_j(t) \triangleq \left.\frac{\partial h_j(t,x)}{\partial x}\right|_{x=x(t)}, \tag{B.23}$$

$${}_wF(t) \triangleq \left.\frac{\partial f(t,x,u,w)}{\partial w}\right|_{x=x(t),u=u(t),w=w(t)}, \quad {}_\eta C(t) \triangleq \left.\frac{\partial c(t,x,u,\eta)}{\partial \eta}\right|_{x=x(t),u=u(t)}, \tag{B.24}$$

$${}_{\hat{x}}\hat{F}(t) \triangleq \left.\frac{\partial \hat{f}(t,\hat{x},\tilde{y})}{\partial \hat{x}}\right|_{\hat{x}=\hat{x}(t),\tilde{y}=\tilde{y}(t)}, \quad {}_xC(t) \triangleq \left.\frac{\partial c(t,x,u,\eta)}{\partial x}\right|_{x=x(t),u=u(t)}, \tag{B.25}$$

$${}_{\tilde{y}}\hat{F}(t) \triangleq \left.\frac{\partial \hat{f}(t,\hat{x},\tilde{y})}{\partial \tilde{y}}\right|_{\hat{x}=\hat{x}(t),\tilde{y}=\tilde{y}(t)}, \text{ and} \quad {}_uC(t) \triangleq \left.\frac{\partial c(t,x,u,\eta)}{\partial u}\right|_{x=x(t),u=u(t)} \tag{B.26}$$

are the partial derivatives of each function. Using (B.18) and taking the time derivative

$$\dot{e}(t) = {}_{x}M(t)\dot{x}(t) - \dot{\hat{x}}(t) \tag{B.27}$$

$$= {}_{x}M(t)_{x}F(t)x(t) - {}_{\hat{x}}\hat{F}(t)\hat{x}(t) - {}_{\tilde{y}}\hat{F}(t)_{x}C(t)x(t) \tag{B.28}$$

$$+ {}_{x}M(t)_{u}F(t)u(t) - {}_{\tilde{y}}\hat{F}(t)_{u}C(t)u(t) \tag{B.29}$$

$$- \sum_{i\in\mathbb{CM}} \hat{K}_i(t)\Big({}_{x}H_i(t)x(t) - {}_{\hat{x}}\hat{H}_i(t)\hat{x}(t)\Big) \tag{B.30}$$

$$+ {}_{x}M(t)_{w}F(t)w(t) - {}_{\tilde{y}}\hat{F}(t)_{\eta}C(t)\eta(t) - \sum_{i\in\mathbb{CM}} \hat{K}_i\nu_i(t) \tag{B.31}$$

$$= \Big({}_{x}M(t)_{x}F(t) - {}_{\tilde{y}}\hat{F}(t)_{x}C(t)\Big)x(t) - {}_{\hat{x}}\hat{F}(t)\hat{x}(t) \tag{B.32}$$

$$+ {}_{x}M(t)_{u}F(t)u(t) - {}_{\tilde{y}}\hat{F}(t)_{u}C(t)u(t) \tag{B.33}$$

$$- \sum_{i\in\mathbb{CM}} \hat{K}_i(t)\Big({}_{x}H_i(t)x(t) - {}_{\hat{x}}\hat{H}_i(t)\hat{x}(t)\Big) \tag{B.34}$$

$$+ {}_{x}M(t)_{w}F(t)w(t) - {}_{\tilde{y}}\hat{F}(t)_{\eta}C(t)\eta(t) - \sum_{i\in\mathbb{CM}} \hat{K}_i\nu_i(t), \tag{B.35}$$

where

$${}_{x}M(t) \triangleq \left.\frac{\partial m(x)}{\partial x}\right|_{x=x(t)}. \tag{B.36}$$

Assuming ${}_{x}M(t)_{x}F(t) - {}_{\tilde{y}}\hat{F}(t)_{x}C(t) \equiv {}_{\hat{x}}\hat{F}(t)$, ${}_{x}M(t)_{u}F(t) \equiv {}_{\tilde{y}}\hat{F}(t)_{u}C(t)$, and ${}_{x}H_i(t) \equiv {}_{\hat{x}}\hat{H}_i(t)$ it can be stated that

$$\dot{e}(t) = \left({}_{\hat{x}}\hat{F}(t) - \sum_{i\in\mathbb{CM}} \hat{K}_i(t){}_{\hat{x}}\hat{H}_i(t)\right)e(t) + {}_{x}M(t)_{w}F(t)w(t) - {}_{\tilde{y}}\hat{F}(t)_{\eta}C(t)\eta(t) - \sum_{i\in\mathbb{CM}} \hat{K}_i\nu_i(t) \tag{B.37}$$

$$= E(t)e(t) + z(t), \tag{B.38}$$

where

$$E(t) \triangleq {}_{\hat{x}}\hat{F}(t) - \sum_{i \in \mathbb{CMM}} \hat{K}_i(t)_{\hat{x}}\hat{H}_i(t) \text{ and} \tag{B.39}$$

$$z(t) \triangleq {}_{x}M(t)_{w}F(t)w(t) - {}_{\tilde{y}}\hat{F}(t)_{\eta}C(t)\eta(t) - \sum_{i \in \mathbb{CMM}} \hat{K}_i\nu_i(t). \tag{B.40}$$

Using the definition of the state transition matrix, this dynamic equation can be solved as

$$e(t) = \Phi(t, t_0)e(t_0) + \int_{t_0}^{t} \Phi(t, \tau)z(\tau)d\tau. \tag{B.41}$$

Using the definition of $\hat{P}$ and assuming $z(t)$ is uncorrelated with $e(t_0)$ it can be stated that

$$\hat{P}(t) = E\big[e(t)e^T(t)\big] \tag{B.42}$$

$$= \Phi(t, t_0)E\big[e(t_0)e^T(t_0)\big]\Phi^T(t, t_0) + \int_{t_0}^{t} \Phi(t, \tau)E\big[z(\tau)z^T(\tau)\big]\Phi^T(t, \tau)d\tau \tag{B.43}$$

$$= \Phi(t, t_0)\hat{P}(t_0)\Phi^T(t, t_0) \tag{B.44}$$

$$+ \int_{t_0}^{t} \Phi(t, \tau)({}_{x}M(t)_{w}F(t)S_w(t)_{w}F^T(t)_{x}M^T(t) \tag{B.45}$$

$$+ {}_{\tilde{y}}\hat{F}(t)_{\eta}C(t)S_{\eta}(t)_{\eta}C^T(t)_{\tilde{y}}\hat{F}^T(t))\Phi^T(t, \tau) \tag{B.46}$$

$$+ \sum_{i \in \mathbb{CMM}} \hat{K}_i(t)S_{\nu_i}(t)\hat{K}_i^T(t)d\tau. \tag{B.47}$$

Finally, taking the derivative with respect to time yields the error state covariance propagation equation.

$$\begin{aligned}\dot{\hat{P}}(t) = &\left({}_{\hat{x}}\hat{F}(t) - \sum_{i \in \mathbb{CMM}} \hat{K}_i(t)_{\hat{x}}\hat{H}_i(t)\right)\hat{P}(t) + \hat{P}(t)\left({}_{\hat{x}}\hat{F}(t) - \sum_{i \in \mathbb{CMM}} \hat{K}_i(t)_{\hat{x}}\hat{H}_i(t)\right)^T \\ &+ {}_{x}M(t)_{w}F(t)S_w(t)_{w}F^T(t)_{x}M^T(t) + {}_{\tilde{y}}\hat{F}(t)_{\eta}C(t)S_{\eta}(t)_{\eta}C^T(t)_{\tilde{y}}\hat{F}^T(t) \\ &+ \sum_{i \in \mathbb{CMM}} \hat{K}_i(t)S_{\nu_i}(t)\hat{K}_i^T(t)\end{aligned} \tag{B.48}$$

Equation (B.48) can be condensed, making use of (B.14), resulting in (B.49).

$$\begin{aligned}\dot{\hat{P}}(t) = {}_{\hat{x}}\hat{F}(t)\hat{P}(t) + \hat{P}(t){}_{\hat{x}}\hat{F}^T(t) - \hat{P}(t)\left(\sum_{i\in\mathbb{CM}} {}_{\hat{x}}\hat{H}_i^T(t)S_{\nu_i}^{-1}(t){}_{\hat{x}}\hat{H}_i(t)\right)\hat{P}(t) \\ + {}_{x}M(t){}_{w}F(t)S_w(t){}_{w}F^T(t){}_{x}M^T(t) + {}_{\tilde{y}}\hat{F}(t){}_{\eta}C(t)S_\eta(t){}_{\eta}C^T(t){}_{\tilde{y}}\hat{F}^T(t)\end{aligned} \tag{B.49}$$

B.3 Linear Modeling

To derive LinCov, the equations in the preceding sections are linearized about the mean reference trajectory defined by $\bar{x}(t) \in \mathbb{R}^{\bar{n}}$. The states are then defined in terms of the true state dispersions from the reference $\delta x(t) \in \mathbb{R}^{\delta n}$ and the navigation state dispersions from the reference $\delta\hat{x}(t) \in \mathbb{R}^{n_e}$. The dispersions are defined as

$$\delta x(t) \triangleq x(t) - \bar{m}(\bar{x}(t)) \text{ and} \tag{B.50}$$

$$\delta\hat{x}(t) \triangleq \hat{x}(t) - m(\bar{m}(\bar{x}(t))), \tag{B.51}$$

where $\bar{m} : \mathbb{R}^{\bar{n}} \mapsto \mathbb{R}^n$ maps the reference states to the truth states.

The state propagation equations (B.1), (B.9), (B.3), and (B.17) are linearized about $\bar{x}$ to produce

$$\delta\dot{x}(t) = {}_{x}\bar{F}(t)\delta x(t) + {}_{u}\bar{F}(t){}_{\hat{x}}\bar{G}(t)\delta\hat{x}(t) + {}_{w}\bar{F}(t)w(t) \tag{B.52}$$

and

$$\begin{aligned}\delta\dot{\hat{x}}(t) = &\left({}_{\hat{x}}\bar{\hat{F}}(t) + {}_{\tilde{y}}\bar{\hat{F}}(t){}_{u}\bar{C}(t){}_{\hat{x}}\bar{G}(t) - \sum_{i\in\mathbb{CM}} \bar{K}_i(t){}_{\hat{x}}\bar{\hat{H}}_i(t)\right)\delta\hat{x}(t) \\ &+ \left({}_{\tilde{y}}\bar{\hat{F}}(t){}_{x}\bar{C}(t) + \sum_{i\in\mathbb{CM}} \bar{K}_i(t){}_{x}\bar{H}_i(t)\right)\delta x(t) \\ &+ {}_{\tilde{y}}\bar{\hat{F}}(t){}_{\eta}\bar{C}(t)\eta(t) + \sum_{i\in\mathbb{CM}} \bar{K}_i(t)\nu_i(t),\end{aligned} \tag{B.53}$$

where

$$ {}_x\bar{F}(t) \triangleq \left.\frac{\partial f(t,x,u,w)}{\partial x}\right|_{x=\bar{m}(\bar{x}(t)),u=\bar{u}(t),w=0}, \quad {}_x\bar{H}_j(t) \triangleq \left.\frac{\partial h_j(t,x)}{\partial x}\right|_{x=\bar{m}(\bar{x}(t))}, \tag{B.54} $$

$$ {}_u\bar{F}(t) \triangleq \left.\frac{\partial f(t,x,u,w)}{\partial u}\right|_{x=\bar{m}(\bar{x}(t)),u=\bar{u}(t),w=0}, \quad {}_{\hat{x}}\bar{\hat{H}}_j(t) \triangleq \left.\frac{\partial \hat{h}_j(t,\hat{x})}{\partial \hat{x}}\right|_{\hat{x}=m(\bar{m}(\bar{x}(t)))}, \tag{B.55} $$

$$ {}_w\bar{F}(t) \triangleq \left.\frac{\partial f(t,x,u,w)}{\partial w}\right|_{x=\bar{m}(\bar{x}(t)),u=\bar{u}(t),w=0}, \quad {}_{\hat{x}}\bar{G}(t) \triangleq \left.\frac{\partial g(t,\hat{x})}{\partial \hat{x}}\right|_{\hat{x}=m(\bar{m}(\bar{x}(t)))}, \tag{B.56} $$

$$ {}_x\bar{C}(t) \triangleq \left.\frac{\partial c(t,x,u,\eta)}{\partial x}\right|_{x=\bar{m}(\bar{x}(t)),u=\bar{u}(t)}, \quad {}_u\bar{C}(t) \triangleq \left.\frac{\partial c(t,x,u,\eta)}{\partial u}\right|_{x=\bar{m}(\bar{x}(t)),u=\bar{u}(t)}, \tag{B.57} $$

$$ {}_{\hat{x}}\bar{\hat{F}}(t) \triangleq \left.\frac{\partial \hat{f}(t,\hat{x},\tilde{y})}{\partial \hat{x}}\right|_{\hat{x}=m(\bar{m}(\bar{x}(t))),\tilde{y}=\bar{\tilde{y}}(t)}, \quad {}_{\tilde{y}}\bar{\hat{F}}(t) \triangleq \left.\frac{\partial \hat{f}(t,\hat{x},\tilde{y})}{\partial \tilde{y}}\right|_{\hat{x}=m(\bar{m}(\bar{x}(t))),\tilde{y}=\bar{\tilde{y}}(t)}, \tag{B.58} $$

$$ {}_\eta\bar{C}(t) \triangleq \left.\frac{\partial c(t,x,u,\eta)}{\partial \eta}\right|_{x=\bar{m}(\bar{x}(t)),u=\bar{u}(t)}, \quad \bar{u}(t) \triangleq g(t, m(\bar{m}(\bar{x}(t)))), \text{ and} \tag{B.59} $$

$$ \bar{\tilde{y}}(t) \triangleq c(t, \bar{m}(\bar{x}(t)), \bar{u}(t)), \tag{B.60} $$

are the dynamics functions linearized about the nominal trajectory. The Kalman gain is linearized about the nominal trajectory as

$$ \begin{aligned} \bar{K}_j(t_k^-) &= \hat{P}(t_k^-)\,{}_{\hat{x}}\bar{\hat{H}}_j^T(t_k^-)\Big({}_{\hat{x}}\bar{\hat{H}}_j(t_k^-)\hat{P}(t_k^-)\,{}_{\hat{x}}\bar{\hat{H}}_j^T(t_k^-) + R_{\nu_j}(t_k)\Big)^{-1}, j \in \mathbb{DM} \\ \bar{K}_i(t) &= \hat{P}(t)\,{}_{\hat{x}}\bar{\hat{H}}_i^T(t)\Big({}_{\hat{x}}\bar{\hat{H}}_i(t)\hat{P}(t)\,{}_{\hat{x}}\bar{\hat{H}}_i^T(t) + S_{\nu_i}(t)\Big)^{-1}, i \in \mathbb{CM} \end{aligned} \tag{B.61} $$

where $\hat{P}(t)$ comes from integrating (B.49). The navigation measurement update equations (B.12), (B.5), and (B.10) are linearized to produce

$$ \delta\hat{x}(t_k^+) = \Big(I_{n_e} - \bar{K}_j(t_k^-)\,{}_{\hat{x}}\bar{\hat{H}}_j(t_k^-)\Big)\delta\hat{x}(t_k^-) + \bar{K}_j(t_k^-)\,{}_x\bar{H}_j(t_k^-)\delta x(t_k^-) + \bar{K}_j(t_k^-)\nu_j(t_k^-). \tag{B.62} $$

The true and navigation dispersion vectors are now appended to form the augmented state vector, $X(t) \in \mathbb{R}^{\delta n+n_e}$.

$$X(t) \triangleq \begin{bmatrix} \delta x(t) \\ \delta \hat{x}(t) \end{bmatrix} \tag{B.63}$$

The dynamic and measurement update equations for X can be derived from (B.52), (B.53), and (B.62) as

$$\dot{X}(t) = \left(\mathcal{F}(t) + \sum_{i \in \mathbb{CM}} \mathcal{K}_i \mathcal{H}_i \right) X(t) + \mathcal{G}(t)\eta(t) + \mathcal{W}(t)w(t) + \sum_{i \in \mathbb{CM}} \mathcal{K}_i(t)\nu_i(t) \tag{B.64}$$

$$X\left(t_k^+\right) = \mathcal{A}_j\left(t_k^-\right) X\left(t_k^-\right) + \mathcal{D}_j\left(t_k^-\right)\nu_j\left(t_k^-\right), j \in \mathbb{DM} \tag{B.65}$$

where

$$\mathcal{F}(t) \triangleq \begin{bmatrix} {}_x\bar{F}(t) & {}_u\bar{F}(t)_{\hat{x}}\bar{G}(t) \\ {}_{\tilde{y}}\bar{\hat{F}}(t)_x\bar{C}(t) & {}_{\hat{x}}\bar{\hat{F}}(t) + {}_{\tilde{y}}\bar{\hat{F}}(t)_u\bar{C}(t)_{\hat{x}}\bar{G}(t) \end{bmatrix}, \qquad \mathcal{G}(t) \triangleq \begin{bmatrix} 0_{\delta n \times n_\eta} \\ {}_{\tilde{y}}\bar{\hat{F}}(t)_\eta\bar{C}(t) \end{bmatrix}, \tag{B.66}$$

$$\mathcal{A}_j(t) \triangleq \begin{bmatrix} I_{\delta n} & 0_{\delta n \times n_e} \\ \bar{K}_j\left(t_k^-\right)_x\bar{H}_j\left(t_k^-\right) & I_{n_e} - \bar{K}_j\left(t_k^-\right)_{\hat{x}}\bar{\hat{H}}_j\left(t_k^-\right) \end{bmatrix}, \quad \mathcal{W}(t) \triangleq \begin{bmatrix} {}_w\bar{F}(t) \\ 0_{n_e \times n_w} \end{bmatrix}, \tag{B.67}$$

$$\mathcal{K}_i(t) \triangleq \begin{bmatrix} 0_{\delta n \times n_{\nu_i}} \\ \bar{K}_i(t) \end{bmatrix}, \qquad \mathcal{D}_j(t) \triangleq \begin{bmatrix} 0_{\delta n \times n_{\nu_j}} \\ \bar{K}_j\left(t_k^-\right) \end{bmatrix}, \text{ and} \tag{B.68}$$

$$\mathcal{H}_i(t) \triangleq \begin{bmatrix} {}_x\bar{H}_i(t) & -{}_{\hat{x}}\bar{\hat{H}}_i(t) \end{bmatrix}. \tag{B.69}$$

The augmented state covariance, $C_A(t) \in \mathcal{S}_+^{\delta n+n_e}$, is defined as

$$C_A(t) \triangleq E\left[X(t)X^T(t)\right] \tag{B.70}$$

and is propagated between measurements with

$$\begin{aligned}\dot{C}_A(t) = &\left(\mathcal{F}(t) + \sum_{i\in\mathbb{CM}} \mathcal{K}_i(t)\mathcal{H}_i(t)\right)C_A(t) + C_A(t)\left(\mathcal{F}(t) + \sum_{i\in\mathbb{CM}} \mathcal{K}_i(t)\mathcal{H}_i(t)\right)^T \\ &+ \mathcal{G}(t)S_\eta(t)\mathcal{G}^T(t) + \mathcal{W}(t)S_w(t)\mathcal{W}^T(t) + \sum_{i\in\mathbb{CM}} \mathcal{K}_i(t)S_{\nu_i}(t)\mathcal{K}_i^T(t)\end{aligned} \tag{B.71}$$

and updated with

$$C_A\big(t_k^+\big) = \mathcal{A}_j\big(t_k^-\big)C_A\big(t_k^-\big)\mathcal{A}_j^T\big(t_k^-\big) + \mathcal{D}_j\big(t_k^-\big)R_{\nu_j}(t_k)\mathcal{D}_j^T\big(t_k^-\big) \quad . \tag{B.72}$$

B.4 Performance Evaluation

When examining the performance of the augmented system, the covariance of the truth state dispersions, $D_{true}(t) \in \mathcal{S}_+^{\delta n}$, the covariance of the navigation state dispersions, $\bar{\hat{P}}(t) \in \mathcal{S}_+^{n_e} \triangleq E\big[\delta\hat{x}(t)\delta\hat{x}^T(t)\big]$, and the covariance of the true navigation state errors, $P_{true}(t) \in \mathcal{S}_+^{n_e}$, can be computed as

$$D_{true}(t) \triangleq E\big[\delta x(t)\delta x^T(t)\big] = \begin{bmatrix} I_{\delta n} & 0_{\delta n\times n_e}\end{bmatrix} C_A(t) \begin{bmatrix} I_{\delta n} \\ 0_{n_e\times\delta n}\end{bmatrix} \tag{B.73}$$

$$\bar{\hat{P}}(t) \triangleq E\big[\delta\hat{x}(t)\delta\hat{x}^T(t)\big] = \begin{bmatrix} 0_{n_e\times\delta n} & I_{n_e}\end{bmatrix} C_A(t) \begin{bmatrix} 0_{\delta n\times n_e} \\ I_{n_e}\end{bmatrix} \tag{B.74}$$

$$P_{true}(t) \triangleq E\Big[(\delta\hat{x}(t) - {}_{\bar{x}}M(t)\delta x(t))(\delta\hat{x}(t) - {}_{\bar{x}}M(t)\delta x(t))^T\Big] \tag{B.75}$$

$$= \begin{bmatrix} -{}_{\bar{x}}M(t) & I_{n_e}\end{bmatrix} C_A(t) \begin{bmatrix} -{}_{\bar{x}}M^T(t) \\ I_{n_e}\end{bmatrix} \tag{B.76}$$

where

$${}_{\bar{x}}M(t) \triangleq \left.\frac{\partial m(x)}{\partial x}\right|_{x=\bar{m}(\bar{x}(t))}. \tag{B.77}$$

CURRICULUM VITAE

James Swedeen

Published Journal Articles

- Metaheuristics for the Electric Vehicle Patrol Route Planning Problem, James Swedeen and Greg Droge, *Applied Soft Computing*, *Under Review*.
- FB-BIT* Planning of Aircraft Under Threat of Detection and Reduced Observability, James Swedeen and Greg Droge, *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, *Under Review*.
- Path planning with uncertainty for aircraft under threat of detection from ground-based radar, Austin Costley, James Swedeen, Greg Droge, Randall Christensen, and Robert C. Leishman, *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, *Accepted*.
- Fillet-based RRT*: A rapid convergence implementation of RRT* for curvature constrained vehicles, James Swedeen, Greg Droge, and Randall Christensen, *Journal of Intelligent & Robotic Systems*, vol. 108, no. 4, p. 68, 2023.

Published Conference Papers

- Mission Planning and Execution Architecture for Robotic Systems Using BPMN, Justin Whitaker, James Swedeen, and Greg Droge, *Intermountain Engineering, Technology and Computing (IETC)*, pp. 34-39, 2024.
- Fillet-based Batch Informed Trees (FB-BIT*): Rapid convergence path planning for curvature constrained vehicles, James Swedeen and Greg Droge, *Seventh IEEE International Conference on Robotic Computing (IRC)*, pp. 71–78, 2023.

- Rapid Discrete Planning for Satellite Constellation Imaging Missions, Konnor Andrews, James Swedeen, and Greg Droge, *Astrodynamics Specialist Conference*, 2022.