Towards Flight Readiness In Unmanned Aerial Systems

Undergraduate Honors Thesis

Presented in Partial Fulfillment of the Requirements for
Graduation with Honors Research Distinction in the
Department of Mechanical Engineering at
The Ohio State University

by

Bryce Ford

April 2023

Advisor: Dr. Mrinal Kumar

Abstract

Flight readiness in autonomous Unmanned Aerial Systems (UAS) is key in advancing their use in aiding humans in unstructured, uncertain, and hazardous environments. The objective of this thesis is to develop a framework for path planning in Unmanned Aerial Systems (UAS) monitoring a prescribed burn. A prescribed burn is a complex, dynamic environment that contains numerous dangers to an autonomous UAS. The primary focus is on the radiative and convective heat flux from a wild-land burn, specifically the danger that it poses to UAS operations near ground level. This danger is best described as a path-dependent resource constraint that is enforced during global planning. The Resource Constrained Shortest Path Problem (RCSPP) is solved using a novel backtracking algorithm derived from Hybrid-A* graph search. Performance gains are shown over unmodified Hybrid-A*, and a Lagrange Relaxation based method. A UAS platform is developed to meet the performance requirements necessary to operate in prescribed burn environments. Software In the Loop (SIL) driven development methods and Hardware In the Loop (HIL) testing methods are used to verify performance requirements. Field tests in a prescribed burn demonstrate the validation of algorithm design goals and performance requirements. The impact of this research is a step towards autonomous fire monitoring to accelerate the setup and execution of prescribed wildland burns.

Acknowledgments

I would first I would like to express my gratitude to my advisor Dr. Mrinal Kumar for his mentorship over the past two and a half years. Without his constant support and encouragement, I would not have been able to accomplish this work. I would like to thank the National Science Foundation award number 2132798 under the NRI 3.0: Innovations in Integration of Robotics Program for supporting my work. I would like to thank Dr. Roger Williams of OSU SENR for providing the opportunity to get up close to and fly in controlled burns.

I would like to thank my friend and mentor Dr. Rachit Aggarwal for putting in the time to teach me everything I needed to know to work with drones. I would like to thank my friend and colleague Andres Lu for providing his expertise in drone development, and his skills as an ace pilot. This project would never has gotten off the ground without him. Additionally, I would like to thank friends and colleagues Gabbie Mbabazi and Kevin Kim for their support in drone development and flight testing.

Lastly, I would like to thank my family for their faith in me and their constant encouragement during this endeavor.

Contents

I Introduction **6**

I.A Path Planning Problem . . . 7

I.B Kinematic Constraints . . . 8

I.C Resource Constraint . . . 9

I.D Additional Constraints . . . 10

II Methodology **11**

II.A Path Planning Methods . . . 11

II.B Resource Constrained Path Planning . . . 15

II.C Lagrange Relaxation Aggregate Cost . . . 15

II.D Backtracking Graph Searches . . . 16

II.E Stopping Criterion . . . 20

II.E.1 Criterion 1: Auxiliary Optimization . . . 20

II.E.2 Criterion 2: Maximum Edge Load Stopping . . . 21

II.E.3 Criterion 3: Random Stopping . . . 21

II.F Path Planning Performance Bench-marking . . . 21

II.F.1 Performance of Resource Constrained Hybrid A* . . . 22

II.F.2 Performance of LARAC . . . 25

II.F.3 Backtracking Hybrid A*: Random Stopping . . . 25

II.F.4 Improved Backtracking Stopping Criteria . . . 27

III Application **31**

III.A Experimental Platform . . . 31

III.A.1 Unmanned Aerial System . . . 31

III.A.2 Introduction to ROS . . . 33

III.A.3 ROS Application Design . . . 34

III.A.4 Simulation Driven Development . . . 36

III.B Flight Testing . . . 37

IV Conclusion **39**

IV.A Future Work . . . 39

IV.B Research Products . . . 39

List of Figures

1 Left: Point-wise Constraint Enforced in terms of Avoiding High (Critical) Heat Flux Contour. Right: Integral Path Loading Constraint Enforced in terms of Maintaining Platform Temperature Under Set Threshold (Temperature = Path Integral of Heat Flux) 6

2 Smoothness of A-Star Versus $\mathscr{H}A^{\star}$. 11

3 The $\mathscr{H}A^{\star}$ algorithm: an illustration of the companion grid, $\mathscr{D}$, and the generation of candidate nodes using Dubins motion primitives 12

4 This sequence details the process of backtracking. Nodes (circles) that are grey are in the visited set, nodes that are green are in the frontier, nodes that are white are unexplored. The shade of each Dubins primitive represents the current load at that point in the search. Darker shades represent higher loads. . 18

5 Path Planning Scenarios Used in Testing 22

6 Trajectory Generated by $\mathscr{H}A^{\star}$ in Each Test Scenario 23

7 Optimal trajectory generated by Lagrange Relaxation Aggregate Cost in each Scenario. 24

8 Optimal Trajectory Generated by $\mathscr{B}\mathscr{H}A^{\star}$ Using Random Stopping 26

9 Optimal Trajectory Generated by $\mathscr{B}\mathscr{H}A^{\star}$ Using Auxiliary Optimization Stopping 28

10 Optimal Trajectory Generated by $\mathscr{B}\mathscr{H}A^{\star}$ Using Maximum Edge Load Stopping 29

11 Eagle UAS Built on a Modified DJI Flamewheel F450 [1] 32

12 Estimator error causes wild variations in Eagle's estimated position during a flight test. 33

13 Base BlueBird UAS Platform 33

14 ROS Application State Diagram 35

15 ROS Application Software In the Loop (SIL) 36

16 ROS Application Hardware In the Loop (HIL) 37

17 Onboard, online demonstration of planning and tracking using BlueBird platform in live flight test. . . 38

18 Infrared image of the fire taken by BlueBird at Zelesky State Forest burn (03/21/2023) 38

List of Algorithms

1 AUX - Construct Auxiliary Grid 13

2 HA^{*} - Hybrid A* Search 14

3 LARAC - Lagrange Relaxation Aggregate Cost 16

4 Backtracking 17

5 Backtracking Hybrid A^{*} ($\mathscr{B}\mathscr{H}A^{\star}$) Search with Integral (Path-Load) Constraints 19

List of Tables

1 Table Comparing $\mathscr{BHA}^{\star}$ Performance in Different Scenarios with Different Stopping Criteria. Best Cost and Least Compute Time are Shown in **Bold**. 30

I. Introduction

Autonomous Unmanned Aerial Systems (UAS) have been proposed as a safe, economical solution to monitoring wildland burns and wildfires. The integration of autonomous UAS into prescribed wildland burns can be used to better understand how topographic, atmospheric, and forest fuel factors can affect the intensity and rate of spread of wildfires in temperate forests. A prescribed burn environment is an uncertain dynamic environment that poses many dangers to an Autonomous UAS. Static obstacles such as trees or other terrain are well understood, however accounting for the danger that the fire poses to a UAS is a developing problem.

Heat flux from the fire in a prescribed burn poses a significant risk to a UAS and must be considered a constraint during path planning [2]. This poses the problem of how to manage the danger that the heat flux from the fire represents. The fire can be represented as a set of heat flux contours as seen in Fig. (1). In this case, a critical heat flux contour

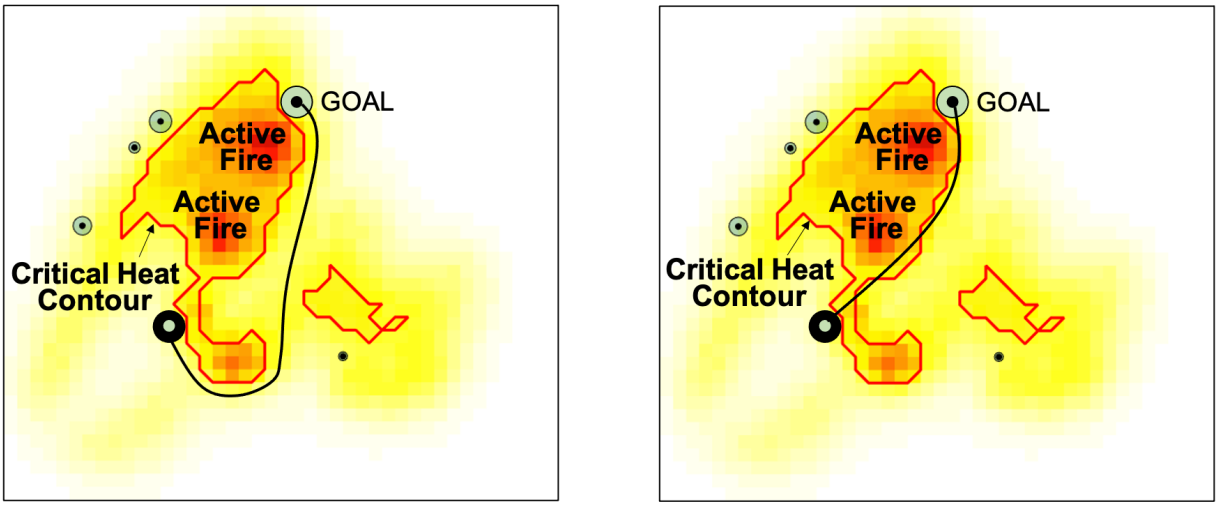


Fig. 1 Left: Point-wise Constraint Enforced in terms of Avoiding High (Critical) Heat Flux Contour. Right: Integral Path Loading Constraint Enforced in terms of Maintaining Platform Temperature Under Set Threshold (Temperature = Path Integral of Heat Flux)

represents a heat flux of 7 kW/m^2. The critical contour can be treated as the boundary of a static obstacle where any point within is considered unreachable during path planning. This point-wise enforcement fails to capture the danger that the fire will actually present to the UAS in that it is not the heat flux at a given point that will cause the failure of the platform, but the cumulative increase in temperature while tracking the path. This means that the fire in a wildland burn can be posed as a path-dependent resource constraint where the constraint is a set threshold for the platform temperature.

Path loading constraints have been studied under the framework of resource-constrained shortest path problems in the existing literature, e.g. see Ref. [3]. The shortest path problem with integral constraints is also known as the Resource Constrained Shortest Path Problem (RCSPP). RCSPPs are commonly found in operations research and network routing [4, 5]. The presence of the resource constraint makes the problem generally unsuitable for graph search methods like Dijkstra's algorithm or the $*$ algorithm. Although it can be solved using Mixed Integer Linear Programming (MILP),

the computational load is generally prohibitive as the size of the problem scales. Applying MILP to RCSPPs with a discretization fine enough to plan paths in complex environments results in compute times on the order of hours [6]. This performance is unacceptable when online path planning is required.

Graph search algorithms are a popular choice for path-planning problems without resource constraints because they can effectively deal with complex obstacles that create non-convex search spaces. As mentioned above, popular grid-based algorithms include Djikstra's (uniform) search [7], $*$ (directional) search [8], $\theta^\star$ (grid-detached) search [9] and their numerous variants. A common shortcoming of the techniques mentioned so far is that they do not conform to the vehicle's dynamic constraints, consequently the resulting "optimal" trajectory may not be feasible. Recently, the hybrid-$A^\star$ ($\mathscr{H}A^\star$) approach has gained popularity [10, 11] for kinematic agents with simple motion primitives, such as the Dubins model.

This work presents a novel *backtracking hybrid-$A^\star$ graph search* to approximate solutions to the RCSPP. As previously mentioned, traditional graph search is not suited to planning with path-loading constraints because such constraints accumulate over the trajectory of the agent. When a candidate node is determined to be inadmissible on account of the loading constraint, conventional graph search effectively treats it as the violation of a point-wise constraint (Fig. (1): left) as opposed to an integral constraint (Fig.(1): Right). The key idea behind the new approach is that the inadmissibility of a candidate node is on account of *the path taken to arrive at its location*. Therefore, instead of classifying the candidate node as "inadmissible", a better alternative is to retreat the search, receding away from the candidate node, thereby allowing the agent to shed the integral load along the path leading up to the constraint violation. Upon the conclusion of the backtracking step, $\mathscr{H}A^\star$ search resumes until another load violation is encountered (or the goal is reached). Of course, the stopping criteria for backtracking are important in this procedure. Several different stopping criteria are explored, and their effect on the performance of Backtracking $\mathscr{H}A^\star$ is evaluated. A "best" stopping criterion is selected and optimality gains over traditional $\mathscr{H}A^\star$ are clearly demonstrated. Numerical comparisons are also shown with the Lagrange Relaxation based Aggregate Cost (LARAC) algorithm.

A. Path Planning Problem

The shortest path planning problem in robotics is defined by constructing a path between two states that minimize distance traveled. The UAS's speed v is assumed to be constant so the minimum distance path is also the minimum time path. This can be stated as follows:

$$J^*(t, \mathbf{s}, u) \;=\; \min_u t_f \tag{1}$$

where $\mathbf{s}$ represents the UAS state vector.

B. Kinematic Constraints

The UAS employs the Dubins kinematic model for agent dynamics [12]

$$\dot{x} = v\cos\psi \tag{2a}$$

$$\dot{y} = v\sin\psi \tag{2b}$$

$$\dot{\psi} = u \tag{2c}$$

Eq. (2) defines the motion in a constant altitude (x, y) plane with a heading (yaw) angle ϕ. A control input u commands the heading rate. An upper bound is imposed on the maximum heading rate of the UAS to account for the minimum turning radius. This is also known as the curvature constraint, or, the maximum steering angle constraint, given as follows

$$|u| \leq U \tag{3}$$

The Dubins kinematic model is commonly used for path planning for ground vehicles with turn-rate constraints [12]. The rationale for including a turning radius constraint for a quad-rotor UAS it is unable to make right-angle turns while maintaining a constant speed. It offers a rapid means to path planning due to the existence of analytical optimal (shortest path) solutions between a given pair of admissible (unobstructed) waypoints [12]. Two notable extensions of the Dubins model have been proposed in the literature:

1) Reeds-Shepp paths [13], which include reversing motion (i.e. $\dot{\psi} = \pm u$). This model is applicable to ground vehicle motion planning, especially slow-moving vehicles in tight spaces such as a parking lot.
2) G^2 and G^3 paths (Refs. [14, 15]), which are smoother variants of the Dubins path. The Dubins path's curvature profile is discontinuous (a G^1 path), which translates to an uncomfortable ride at best and an unreasonably high control actuation at worst. Path feasibility is improved by adding steering rate or steering acceleration constraints, which correspond to G^2 or G^3 trajectories, respectively. The G^3 option results in a continuously differentiable steering profile.

This work uses the original Dubins motion primitives without reversing motion. Consider an agent with minimum turn radius R traveling at constant speed v, such that $R = v/U$. A constant time discretization of Δt is used as the time step for motion planning, resulting in motion primitives of constant and equal arc lengths. The Dubins model admits only three unique motion primitives, which are enumerated below:

1) Heading straight (S): the agent continues without a change in heading, resulting in "forward motion" covering a distance of $v\Delta t$,
2) Turn left at minimum turn radius (L): the agent turns left at maximum steering rate $\dot{\psi} = U$, resulting in a

circular arc of length $v\Delta t$, and,

3) Turn right at minimum turn radius (R): the agent turns right at maximum steering rate $\dot{\psi} = -U$, resulting in a circular arc of length $v\Delta t$.

These motion primitives can be utilized in graph-based trajectory planning methods [16] and sampling-based trajectory planning methods [17].

C. Resource Constraint

The resource constraint is described as the increase in temperature that the UAS platform experiences while tracking a trajectory between an initial state and a terminal state. It can be formulated as the path integral over the heat flux contours as follows:

$$\underbrace{\int_0^{t_f} \underbrace{\mathcal{F}_{\mathcal{L}}(\tau, \mathbf{s}(\tau), u(\tau))}_{\text{rate of damage}}\, d\tau}_{\text{path load}} \leq \underbrace{\mathcal{L}^{\star}}_{\text{loading limit}} \tag{4}$$

The rate of damage in this case is normalized heat flux from the fire. This can be represented as follows:

$$\mathcal{F}_{\mathcal{L}}(t, \mathbf{s}(t), \mathbf{u}(t)) = \frac{A}{mc_p}\phi(\mathbf{s}(t)) \tag{5}$$

The variables above have the usual meaning (m = vehicle mass, A = incident area, c_p = heat capacity, $\phi(\cdot)$ = heat flux). Heat flux can be allowed to be negative (cooling effect), allowing the planner to perform load-shedding by interspacing flight through hot regions with flight over cooler areas, thus keeping the total rise in temperature under loading limits:

$$\dot{\phi}(x(t), y(t)) = \begin{cases} \dot{\phi}_{\text{rad}}(x(t), y(t)) = \dot{q}'', & \text{if } \dot{\phi}_{\text{rad}}(x(t), y(t)) > 0 \\ \dot{\phi}_{\text{cool}}(x(t), y(t)) = -h(T - T_{\text{min}}), & \text{otherwise} \end{cases} \tag{6}$$

This work only considers non-negative loading constraints such that loading relief is not possible. In other words, $\dot{\phi}_{\text{cool}}(x(t), y(t)) = 0$. While convection and radiation are the primary modes of heat transfer from the wildfire, radiation becomes the dominant form of heat transfer for surfaces exceeding $400°C$ [18]. As the flame temperatures of a wildfire can vary between $800°C$ and $1000°C$, the fire's radiative heat that extends up into the atmosphere is the primary concern as it affects the safety the UAV. Each burning location is treated as a radiative surface that emanates energy per unit area at a rate according to the Stefan-Boltzmann law:

$$E = \sigma_s \epsilon T_f^4 \tag{7}$$

where T_f is the absolute temperature of the flame in K, $\epsilon \in [0, 1]$ is the emissivity and $\sigma_s = 5.67 \times 10^{-8} W/m^2K^4$ is the Stefan-Boltzmann constant. This rate of heat energy loss is referred to as the *heat flux* radiates in all directions from the flame surface A_1. The radiant heat flux that is incident upon a small element of a secondary surface A_2 due to the flame's flux is given as:

$$\dot{q}'' = E\phi \tag{8}$$

where ϕ, known as the configuration factor - a dimensionless quantity that describes the geometric relation (e.g. distance and orientation) between surfaces A_1 and A_2. The heat flux from the multiple radiating surfaces are added to obtain the net heat flux.

D. Additional Constraints

Static obstacles such as trees or terrain can be modeled as polygons as follows:

$$\bigvee_{j=1}^{M_i} a_{i,j}x + b_{i,j}y > c_{i,j}, \quad i = 1, \dots N \tag{9}$$

where, $(a_{i,j}, b_{i,j}, c_{i,j})$ are parameters of the j^{th} vector of the i^{th} polygonal comprising of M_i vertices. The trajectory that the UAS tracks is not permitted to intersect with the polygonal obstacle. The boundary conditions that must be satisfied to generate a path are

$$\begin{aligned} \text{Terminal Conditions: } \{x(0), y(0), \psi(0)\} &= \{x_0, y_0, \psi_0\} \\ \{x(t_f), y(t_f), \psi(t_f)\} &= \{x_f, y_f, \psi_f\} \end{aligned} \tag{10a}$$

$$\begin{aligned} \text{State Bounds: } x_{\min} &\leq x(t) \leq x_{\max} \\ y_{\min} &\leq y(t) \leq y_{\max} \end{aligned} \tag{10b}$$

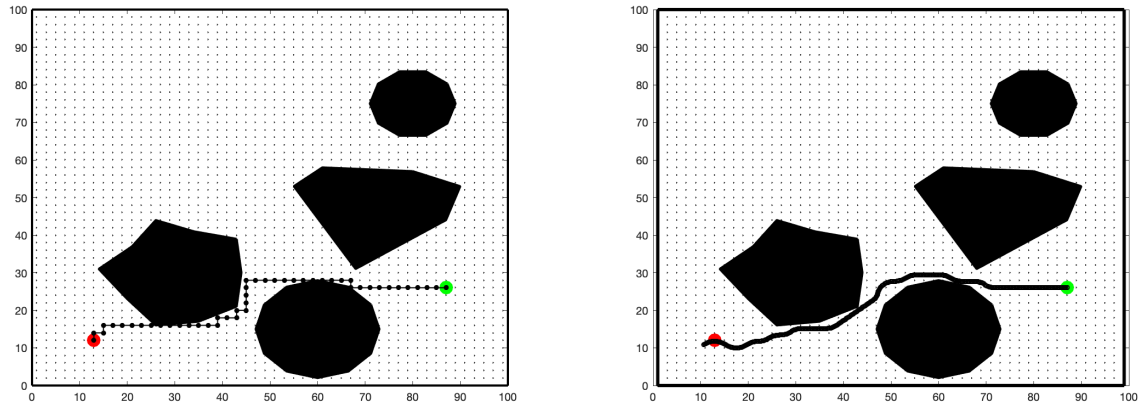
This ensures that the planned path actually starts and terminates at the desired states. Additionally, it constrains the planning method to a prescribed region, and thus the path to the prescribed region.

II. Methodology

This chapter lays out the methods used to solve path planning problems posed with the constraints described in Section I. To account for the UAS kinematic constraints 'hybrid' graph searches described in [16] are utilized. To account for the path-dependent resource constraint a novel graph search algorithm is presented that utilizes backtracking is presented. Additionally, LARAC is applied to the problem and used as a performance benchmark.

A. Path Planning Methods

The hybrid $A^\star$ algorithm was developed as a means to "detach" the graph search from the grid to achieve smoother trajectories. This is especially relevant to robotic agents with non-holonomic constraints, e.g. fixed-wing unmanned air vehicles, that are unable to change their heading in a non-smooth manner. Fig.(2) illustrates the difference between optimal paths generated by the A^* search and the hybrid-A^* ($\mathscr{H}A^\star$) search. Fig.(2(a)) shows the case in which four actions are considered in the A^* search (Up, Down, Left, Right). The $*$ solution is bound to the graph, resulting in a non-smooth, "node-hopping" trajectory.



(a) Optimal Trajectory Generated by A^* Graph Search

(b) Optimal Trajectory Generated by HA^* Graph Search

Fig. 2 Smoothness of A-Star Versus $\mathscr{H}A^\star$.

On the other hand, the $\mathscr{H}A^\star$ algorithm builds its graph using the motion primitives described in Sec.(2). Here, the search space contains three variables, namely, two components of position (x, y) and one component of heading (ψ) for path-planning in two-dimensional space, e.g., ground robots or constant altitude flight. It does use a companion grid, $\mathscr{D}$, in the three-dimensional search space, that functions in the background to determine visitation of various regions of the three-dimensional search space. The view shown in Fig.(2(b)) only illustrates a two-dimensional cross-section of the companion grid, $\mathscr{D}$. Clearly, $\mathscr{H}A^\star$ is still a discrete graph search, in the sense that the trajectory is constructed by patching together finite-sized motion primitives extracted from a motion model. The time-step is fixed, such that each

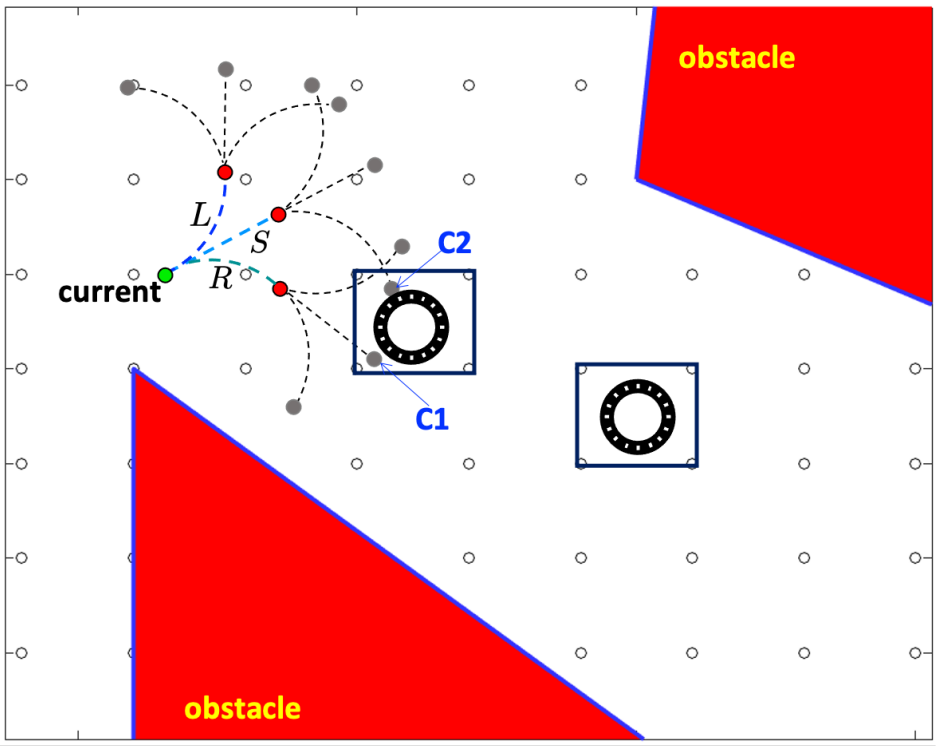


Fig. 3 The $\mathscr{H}A^\star$ algorithm: an illustration of the companion grid, $\mathscr{D}$, and the generation of candidate nodes using Dubins motion primitives

candidate action (S, L, R) has the same path length: see Fig.(3) for an illustration. A zoomed-in view is shown, with the companion grid, $\mathscr{D}$, depicted using empty circles. Note that only the $x - y$ cross-section of the grid $\mathscr{D}$ is visible. Discretization of the third variable, i.e. heading angle, is illustrated using round dials the represent a discrete set of heading angles. A good rule of thumb for the design of the companion grid is to use the vehicle's kinematic capabilities, e.g. $\delta x \approx v\Delta t$, $\delta y \approx v\Delta t$ and $\delta\psi \approx v\Delta t/R$, such that

$$\mathscr{D} \;=\; \{x_i\}_{i=1}^{N_x} \otimes \{y_i\}_{i=1}^{N_y} \otimes \{\psi_i\}_{i=1}^{N_\psi} \tag{11}$$

where, $N_x \;=\; \left\lceil\, [(x_{\max} - x_{\min})/\delta x] \,\right\rceil$, and N_y and N_ψ are similarly defined. Candidate paths (S, L, R) emerging from the *current* location of the graph search (green circle) are shown, resulting in three new *candidate* nodes (red circles). Clearly, the candidate nodes do not coincide with the nodes on the companion grid, $\mathscr{D}$. The purpose of the companion grid is to restrain the growth of candidate nodes. Unhindered kinematic exploration through the use of motion primitives will invariably cause an explosive growth in the number of candidate nodes, e.g. the filled gray circles in Fig.(3) show the second level of candidate nodes generated starting from the current node (green). The home cell of each candidate node in the companion grid is identified using the position *and* heading information of the node. If this node is marked as "visited" (or "closed") during the graph search, its home cell is simultaneously marked as visited or closed. All

future candidate nodes generated in this cell are automatically discarded, thereby restricting the computational burden of $\mathscr{H}A^\star$. Some important points to note about this process:

- Cells in $\mathscr{D}$ are three dimensional, characterized by x, y and ψ information. Fig.(3) shows nodes in $\mathscr{D}$ (empty black circles) in the $x-y$ cross section. The ψ-dimension is illustrated using a dial for highlighted two $x-y$ cells.
- Consider the two gray candidate nodes marked C1 and C2. While they both fall in the same $x-y$ cell, they have very different heading coordinates and are in different cells in $\mathscr{D}$. Therefore, the closeness of candidate nodes must account for all three dimensions.
- As previously mentioned, when a candidate node is marked visited (moved to the closed set), so is its home cell and no further candidate nodes in this cell are considered.

The steps required to set up the companion grid prior to performing a hybrid graph search are detailed in Algorithm (1). The rest of the conventional $\mathscr{H}A^\star$ method is outlined in in Algorithm (2). The following nomenclature is employed in this algorithm:

- O: Starting pose for the graph search (initial conditions), such that $O.x = x_0$, $O.y = y_0$ and $O.\psi = \psi_0$.
- G: Goal (destination) pose of path-planning, such that $G.x = x_f$, $G.y = y_f$ and $G.\psi = \psi_f$.
- s: Starting node in the search.
- t: Goal node in the search.
- c: Current node being considered by the search.
- n: Potential new node to be added to the frontier, or a generic node in the search domain.
- DOM: domain of solution
- OBS: set of obstacles
- Frontier (F): the set of "open nodes" (i.e. admissible candidate nodes)
- Closed (V): the set of "closed nodes" (i.e. nodes marked visited)

Algorithm 1 AUX - Construct Auxiliary Grid

Require: $O, G, v, R, \Delta t$, DOM

1: $\delta x \leftarrow v * \Delta t$, $\delta y \leftarrow v * \Delta t$, $\delta\psi \leftarrow v * \Delta t / R$
2: $Gx \leftarrow \mathrm{DOM}_{x_{\min}}$: $-\delta x$: $O.x$: δx: $\mathrm{DOM}_{x_{\max}}$
3: $Gy \leftarrow \mathrm{DOM}_{y_{\min}}$: $-\delta y$: $O.y$: δy: $\mathrm{DOM}_{y_{\max}}$
4: $G\psi \leftarrow \mathrm{DOM}_{\psi_{\min}}$: $-\delta\psi$: $O.\psi$: $\delta\psi$: $\mathrm{DOM}_{\psi_{\max}}$
5: $\mathscr{D} \leftarrow Gx \otimes Gy \otimes G\psi$
6: $s \leftarrow D(O), t \leftarrow D(G)$
7: return $\mathscr{D}, s, t$

Each node has a parent ($n[parent]$) in the Closed Set (e.g. green node is the parent for each of the three red nodes in Fig.(3).). Additionally each node has a current pose ($n[pose]$), current cost ($n[cost]$), and current load, ($n[load]$). The current cost and current load represent the distance traveled from the start pose and the value of the path integral at the considered node respectively. The subroutine GETACTIONS employs Dubins' motion primitives to generate new

Algorithm 2 HA^* - Hybrid A* Search

Require: $O, G, v, R, \Delta t$, DOM, OBS

$\mathscr{D}, s, t \leftarrow$ AUX($O, G, v, R, \Delta t$, DOM)
Closed Set $V = \emptyset$
Frontier Set $F = \emptyset$
s[parent] $\leftarrow$ null, s[cost] $\leftarrow 0$
t[pose] $\leftarrow G$
add s to F
while $F \neq \emptyset$ **do**
    $c \leftarrow$ node in F with $\min_p f(s, c)$
    add c to V
    if $c == t$ **then**
        return c
    else
        actions $\leftarrow$ GETACTIONS($c, \mathscr{D}$)
        for aci $\in$ actions **do**
            n[parent] $\leftarrow c$
            n[pose] $\leftarrow$ aci[pose]
            n[cost] $\leftarrow c$[cost] + aci[cost]
            $f \leftarrow n$[cost] + $h(n, t)$
            if CHECK(DOM, OBS) and $c \notin V$ **then**
                if $n \notin F$ or $f \leq f(s, n) \in F$ **then**
                    if $n \in F$ **then**
                        remove n from F
                    end if
                    $f(s, n) \leftarrow f$
                    $F \leftarrow n$
                end if
            end if
        end for
    end if
end while

candidate nodes starting from the *Current* node. It computes the cost in distance traveled as well as the load incurred (By integrating Eq. (8)) along the Dubins Primitive. The result of the function f is used to order the frontier, where $f(n) = n[cost] + h(n,t)$, i.e. the current cost of node end plus a heuristic estimating the remaining cost to go from n to t. In this work the heuristic $h(n,t) = \|(x_n, y_n) - (x_t, y_t)\|$.

B. Resource Constrained Path Planning

To account for the resource constraint only a simple modification has to be made to the $\mathscr{H}A^\star$ algorithm. The resource consumption for the potential path is determined by summing the resource consumption along motion primitives in the current path. The constraint is applied in $\mathscr{H}A^\star$ by checking if the current node has a load in violation of the resource constraint, i.e. $\sum_{k\in\text{path}} \Delta\mathcal{F}_\mathcal{L}(k)\Delta t > \mathcal{L}^*$. This is not dissimilar to checking if a node is colliding with a static obstacle.

This modification poses a significant problem as the problem is now NP-hard [19]. Simply, if a problem is NP-hard it means that every possible solution has to be enumerated to guarantee that the result is the cost-optimal solution. This is an issue because the number of solutions to any given resource-constrained shortest path problem scales exponentially with the size of the graph describing it. As a result, finding optimal paths quickly becomes infeasible even with moderately sized problems. $\mathscr{H}A^\star$ can still be applied to the resource-constrained path planning problem, but it no longer has the guarantee of optimality, or even that it will find a solution.

C. Lagrange Relaxation Aggregate Cost

The Lagrange relaxation-based aggregate cost (LARAC) algorithm described in Refs.[20, 21] is a well-developed solution to the resource-constrained path planning problem. LARAC is adapted to the kinematic constraints by employing a Dubins Primitive based Hybrid-Dijksta's algorithm instead of a grid search-based Dijkstra's Algorithm. Hybrid-Dijkstra's is denoted by the function $\mathscr{H}D(O, G, v, R, \Delta t, \text{DOM}, \text{OBS}, J)$ where J describes the cost function for Dijkstra's to minimize. Practically, this is the same algorithm as (2) but when the heuristic is set to zero, i.e., $h(n,t) = 0$ In this J can represent the minimum time path, the minimum resource consumption path, or something else entirely.

LARAC starts by finding the minimum cost solution without considering the loading constraint. The minimum cost solution is then evaluated to determine if it violates the path loading limit. If it does, then the minimum load solution is found, as described in Eq.(12).

$$J^*(t, u^*, \mathbf{s}) \;=\; \min_u \int \mathcal{F}_\mathcal{L}(\tau, \mathbf{s}(\tau), u(\tau))d\tau \tag{12}$$

If the minimum load solution is feasible and the minimum cost solution is infeasible, LARAC relaxes the resource

Algorithm 3 LARAC - Lagrange Relaxation Aggregate Cost

Require: $O, G, v, R, \Delta t, \mathcal{L}^\star$ DOM, OBS
$p_c \leftarrow \mathscr{H}D(O, G, v, R, \Delta t$, DOM, OBS, J = Eq. 1)
if $p_c[\texttt{load}] \leq \mathcal{L}^\star$ **then**
    return p_c
end if
$p_l \leftarrow \mathscr{H}D(O, G, v, R, \Delta t$, DOM, OBS, J = Eq. 12)
if $p_c[\texttt{load}] \geq \mathcal{L}^\star$ **then**
    return no solution
end if
while true **do**
    $\lambda = \frac{p_c[cost]-p_l[cost]}{p_l[load]-p_c[cost]}$
    $r \leftarrow \mathscr{H}D(O, G, v, R, \Delta t$, DOM, OBS, J = Eq. 13)
    if $r\ [\lambda cost] == p_c\ [\lambda cost]$ **then**
        return p_d
    else
        if $r[\texttt{load}] \leq \mathcal{L}^\star$ **then**
            $p_l \leftarrow r$
        else
            $p_c \leftarrow r$
        end if
    end if
end while

constrained problem to a minimum cost problem where the cost is defined by

$$J_\lambda(t, \mathbf{s}, u) \;=\; J(t, \mathbf{s}, u) + \lambda \int_0^t \mathcal{F}_\mathcal{L}(\tau, \mathbf{s}(\tau), u(\tau)) d\tau \tag{13}$$

A solution is then found using an iterative application of the hybrid Dijkstra's algorithm to minimize the relaxed cost J_λ, unconstrained from the path loading constraint. The greater the value of λ is, the more the search deviates from the minimum cost solution. During each iteration, the Lagrange parameter λ is modified in a manner similar to a binary search with the goal being to find the minimum λ that produces a path that obeys the loading constraint, and thus deviates the least from the minimum cost solution. A major benefit of LARAC is that if a solution is feasible it will find it, and it will be able to discern if no solution is feasible.

D. Backtracking Graph Searches

This work develops a novel *backtracking* procedure for $\mathscr{H}A^\star$ based on the insights outlined in the previous section. It offers a way to adapt a directional graph search method (including other methods like $*$, $\theta^\star$, or Dijkstra's) to adapt to problems including path dependent loading constraints. The approach is based on the following principles:

- Node-inadmissibility due to the violation of the integral constraint (Eq.(4)) is path dependant. The resource constraint is violated at a certain node due to the path taken to get to it.
- There is more than one way to get to a node on the graph. $\mathscr{H}A^\star$ will find a shortest possible path to a node, but

that path is not necessarily the only minimum cost path, or even the only possible path. Changing the search behavior at some point before the then inadmissible node is reached can result in a new path that might not violate the integral constraint.

- This change in search behavior can be achieved by backtracking away from the inadmissible node until some stopping criterion is met. The search can then resume from that point, picking a different node from the frontier, thus modifying the search behavior. This allows for the re-exploration of a section of the visited set that lead to a violation of the integral constraint.

Backtracking is performed by stepping backwards in the visited set, backing away from the current node in violation of the integral constraint. The backtrack proceeds until the stopping criterion has been met. The stopping criterion takes the following general form of a binary function:

$$\mathcal{S}(n) \rightarrow \{0, 1\} \tag{14}$$

which takes a node n in the path from $s \rightarrow c$ as input. It evaluates to true if the stopping criterion is satisfied, and false if it is not. Then starting with the node n^*, such that $\mathcal{S}(n^*) \rightarrow 1$, all of its children are identified in the Frontier Set and the Closed Set. The nodes identified are removed from their respective sets and "released" into the unexplored area of the domain. This process is detailed in Algorithm (4).

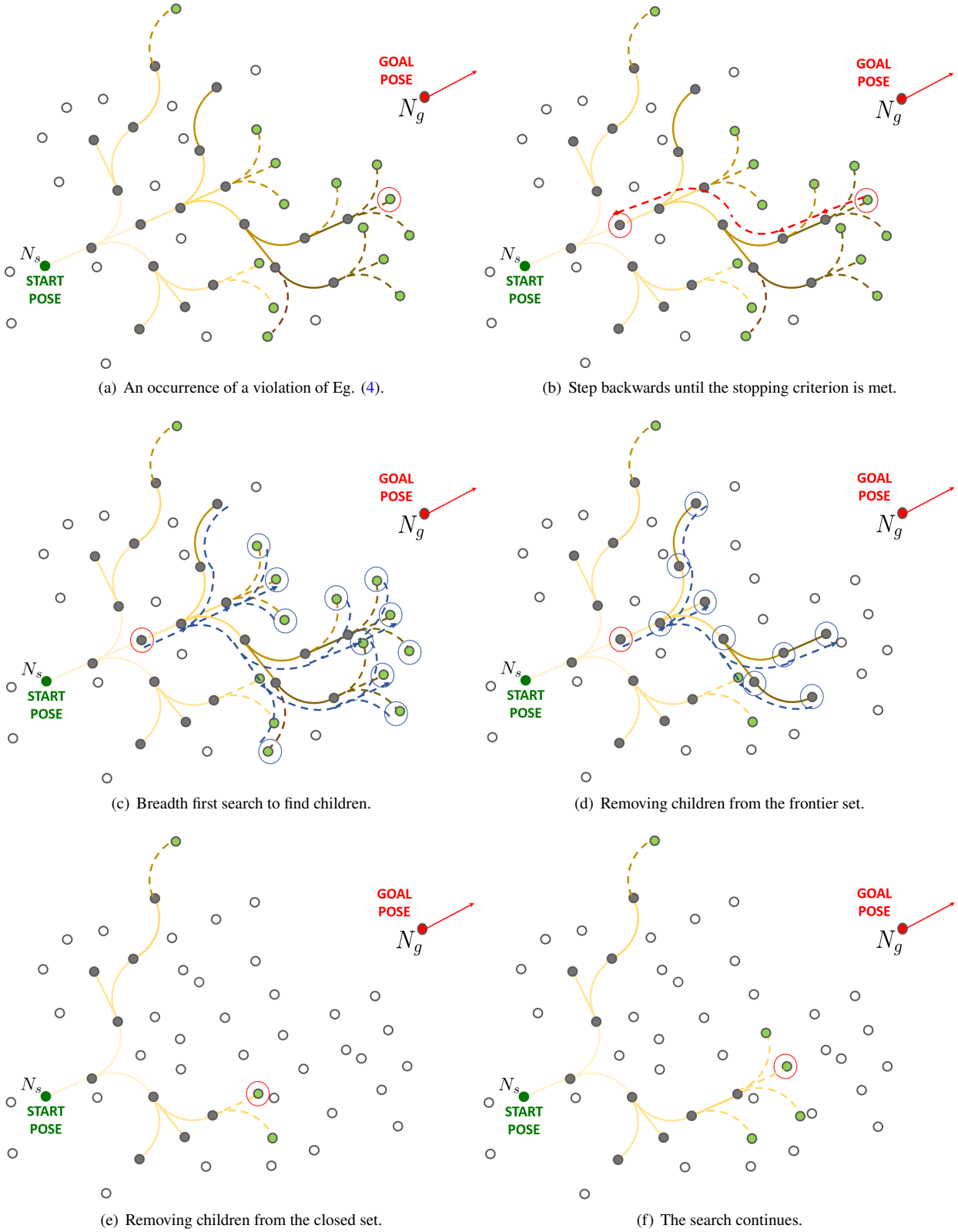
Algorithm 4 Backtracking

Require: V, F, c, s

1: Find the first node n^* in the path $s \rightarrow c$ that satisfies the stopping criterion as seen in Fig.(4(b)).
2: Starting with n^* use a breadth first search to find all nodes n' in V and F where the path $n^* \rightarrow n'$ exists as seen in Fig.(4(c)).
3: Remove all of the children identified from the breadth first search from F as seen in Fig.(4(d)).
4: Remove all of the children identified from the breadth first search from V including n^* as seen in Fig.(4(e)).
5: Return V, F

After the backtracking process is complete and the nodes have been removed from the Frontier Set and Closed Set set the search ($\mathscr{H}A^\star$) will resume. Because the search has not generated new child for the node stopped at, ($\mathscr{H}A^\star$) will be able to re-explore the are of the Closed Set that was removed by backtracking. This re-exploration allows for the search to efficiently handle the integral constraint. The full $\mathcal{B}\mathscr{H}A^\star$ algorithm in given in Algorithm (5).

While this may appear reasonable, it is an incorrect way of dealing with integral constraints. If a candidate node violates obstacle avoidance conditions, the node is certainly inadmissible. However, if it exceeds the path-loading limit, it is not through its own fault. The violation of the integral constraint at node n is on account of the particular path (sequence of parents) that led to the candidate, i.e. $\sum_{k \in \text{path}} \Delta\mathcal{F}_{\mathcal{L}}(k)\Delta t > \mathcal{L}^*$. It is entirely possible to find an alternate path, possibly at a higher cost that permits the node n to be admissible.



(a) An occurrence of a violation of Eg. (4).

(b) Step backwards until the stopping criterion is met.

(c) Breadth first search to find children.

(d) Removing children from the frontier set.

(e) Removing children from the closed set.

(f) The search continues.

Fig. 4 This sequence details the process of backtracking. Nodes (circles) that are grey are in the visited set, nodes that are green are in the frontier, nodes that are white are unexplored. The shade of each Dubins primitive represents the current load at that point in the search. Darker shades represent higher loads.

Algorithm 5 Backtracking Hybrid A^* ($\mathscr{BH}A^\star$) Search with Integral (Path-Load) Constraints

Require: $O, G, v, R, \Delta t, \mathcal{L}^\star$ DOM, OBS

$\delta x \leftarrow v * \Delta t$, $\delta y \leftarrow v * \Delta t$, $\delta\psi \leftarrow v * \Delta t / R$
$Gx \leftarrow \text{DOM}_{x_{\min}}\colon -\delta x\colon O.x : \delta x\colon \text{DOM}_{x_{\max}}$
$Gy \leftarrow \text{DOM}_{y_{\min}}\colon -\delta y\colon O.y : \delta y\colon \text{DOM}_{y_{\max}}$
$G\psi \leftarrow \text{DOM}_{\psi_{\min}}\colon -\delta\psi\colon O.\psi : \delta\psi\colon \text{DOM}_{\psi_{\max}}$
$\mathscr{D} \leftarrow Gx \otimes Gy \otimes G\psi$
Closed Set $V = \emptyset$
Frontier Set $F = \emptyset$
s[parent] $\leftarrow$ null, s[pose] $\leftarrow 0$, s[cost] $\leftarrow 0$, s[load] $\leftarrow 0$
t[pose] $\leftarrow G$
add s to F
while $F \neq \emptyset$ **do**
    $c \leftarrow$ node in F with $\min_p f(s, c)$
    if c[load] $\leq \mathcal{L}^\star$ {Is Eq.(4) satisfied?} **then**
        add c to V
        if $c == t$ **then**
            return c
        else
            actions $\leftarrow$ GETACTIONS($c, v, R, \Delta t$)
            for aci $\in$ actions **do**
                n[parent] $\leftarrow c$
                n[pose] $\leftarrow$ aci[pose]
                n[cost] $\leftarrow c$[cost] + aci[cost]
                n[load] $\leftarrow c$[load] + aci[load]
                $f \leftarrow n$[cost] $+h(n, t)$
                if CHECK(DOM, OBS) and $c \notin V$ **then**
                    if $n \notin F$ or $f \leq f(n) \in F$ **then**
                        if $n \in F$ **then**
                            remove n from F
                        end if
                        $f(s, n) \leftarrow f$
                        $F \leftarrow n$
                    end if
                end if
            end for
        end if
    else
        $(V, F) \leftarrow$ Backtracking(V, F, c, s) {Use Alg.(4)}
    end if
end while

E. Stopping Criterion

The stopping criteria is a crucial element in the success of backtracking as an effective strategy for producing "good" solutions to resource constrained path planning problems. The stopping criteria allows for control over how much of the Visited Set is released into the unexplored domain when backtracking. Because children for the node n^* are not re-generated, backtracking has the unavoidable effect of removing potential solutions. Too many backtracks, or backtracks that stop deep in the visited set close to the start node s can cause the search to "collapse" as backtracking eliminates all nodes in the frontier. A good stopping criterion strikes the balance of opening up enough of the Visited Set for re-exploration, while avoiding "collapsing" the search and thus not producing a solution. This criterion can be picked arbitrarily, however it is most effective when it is chosen to take aspects of the loading function into consideration. In this work four potential stopping criterion are considered.

1. Criterion 1: Auxiliary Optimization

With Auxiliary Optimization Stopping, backtracking is stopped with the aid of the solution of an auxiliary optimization problem that seeks to minimize the loading function, starting from the specified starting pose. This step is completed offline before the $\mathscr{H}A^\star$ with backtracking commences. This optimization (minimization) procedure is defined as:

$$\mathcal{J}^*(t, u^*, \mathbf{s}) \;=\; \min_{u} \int \mathcal{F}_{\mathcal{L}}(\tau, \mathbf{s}(\tau), u(\tau)) d\tau \tag{15}$$

The constraints for the above minimization problem include the kinematics given in Eq.(2), initial conditions given in Eq.(10a) and domain bounds in Eq.(10b). There are no terminal conditions because the objective is to determine the minimum constraint value within each cell in $\mathscr{D}$. In other words, the above problem (Eq.(12)) is to be solved in a 'hybrid-Djikstra' framework, in the sense that:

1) Dubins motion primitives are employed for candidate node generation, and,
2) a uniform search is conducted in the (x, y, ψ) domain, i.e. no heuristic cost is employed

The outcome of the hybrid-Djikstra search is a tabularization of the minimum possible path-load value inside each cell of the companion grid, $\mathscr{D}$. This information can now be used to build a stopping criterion for $\mathscr{BH}A^\star$. Consider a relaxation parameter $\xi > 1$ (strictly greater than). Then, backtracking stops at the first node M^* for which

$$\int_0^t \mathcal{F}_{\mathcal{L}}(\tau, \mathbf{s}(\tau), u(\tau)) d\tau \;\leq\; \xi \mathcal{J}^*(\cdot, \cdot, \mathscr{C}(n^*)) \tag{16}$$

where, $\mathscr{C}(n^*)$ is the cell in $\mathscr{D}$ containing n^*. In words, the backtracking process stops at a node n^* (depicted as the node highlighted in red in Fig.(4(b))) where the actual path-load encountered while minimizing the cost given in Eq.(1) is less

than or equal to ξ times the minimum possible path-load possible at that node. This means that the re-exploration starts from a point where the search is closer to the minimum load solution that that point, thus relaxing the load on the search.

2. Criterion 2: Maximum Edge Load Stopping

Solving the offline minimization problem described above presents a significant computational burden as it involves visiting every node in the search space as well as a unique solution for every starting pose. The "maximum edge load" criterion described in this section allows the search to shed load without having to solve an expensive offline optimization problem. This can be stated as follows:

$$\mathbf{s}^* = \max_{\tau \epsilon [0,t]} \mathcal{F}_{\mathcal{L}}(\tau, \mathbf{s}(\tau), u(\tau)) \tag{17}$$

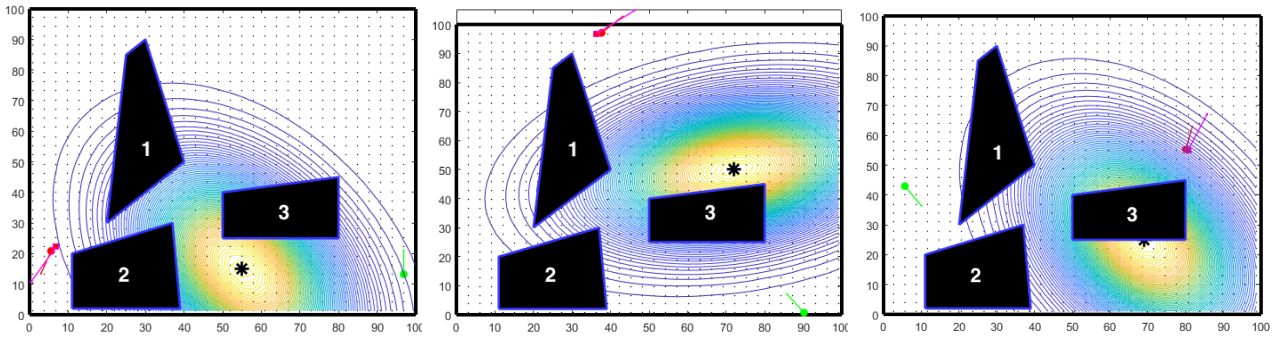
where $\mathbf{s}^*$ is the pose along the path to the invalid node where the value of the loading function is maximised. In practice this means that the node n^* is identified where the load accumulated along the Dubins primitive that lead to it is maximised. When n^* is found the search backtracks from the node in violation c to the node n^*. This means that when stopping when Eq.(17) the re-exploration starts where the rate of load accumulation was the highest along the search path. The advantage of stopping in this manner is that the search is pushed away from what was a local maximum along current path, and when repeatedly applied, pushing the search away from local maximums in the loading functions itself.

3. Criterion 3: Random Stopping

To serve as a baseline for other stopping criteria to be judged against, a random stopping criteria is used. This stopping criteria terminates the backtracking process at the current node with probability ϵ (an arbitrarily picked threshold). In other words, backtracking stops when $u < \epsilon$, where u is drawn from $\mathcal{U}[0, 1]$ (uniform distribution on $[0, 1]$). This models backtracking as a random walk process that is independent of search related variables and only depends on the stopping threshold (ϵ). This criterion functions as a baseline because it is the "doing nothing" approach to backtracking, and can be compared to "doing something", i.e. using Eq.(16), or Eq.(17)

F. Path Planning Performance Bench-marking

To serve as a test bench to compare $\mathscr{BH}A^\star$ to other algorithms, a set of three scenarios is constructed. Each scenario has the same search space and static obstacles (shaded in black and labeled 1-3). These scenarios are designed to illustrate difficulties posed by the resource constraint. Comparisons of the $\mathscr{BH}A^\star$ algorithm will be made against traditional $\mathscr{H}A^\star$ without backtracking (Sec.(II.F.1)) and the Lagrange relaxed aggregate cost algorithm (Sec.(II.C)). An "easy scenario" occurs when for a given pair of starting and destination locations, the integral constraint is never violated by a traditional $\mathscr{H}A^\star$ algorithm. In such cases, backtracking in never needed, and indeed, $\mathscr{H}A^\star$ and $\mathscr{BH}A^\star$ provide the same outcomes. Most difficult scenarios involve a "choke point" that the search must navigate to achieve an optimal



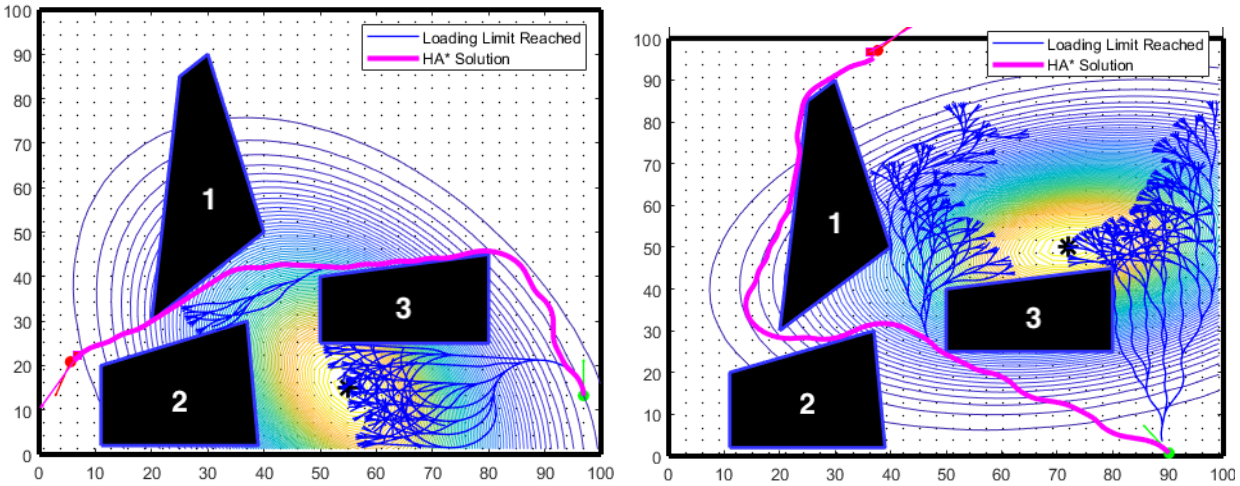
(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Fig. 5 Path Planning Scenarios Used in Testing

solution with respect to cost. In the following scenarios, the stopping criteria discussed above are used to generate a solution. To replicate the heat contours of a fire in a manner that is simple to modify, a Gaussian distribution is used. The contours of the Gaussian distribution represent the heat-flux contours of the fire, and because it is a Gaussian distribution, its shape can be easily modified by changing the underlying mean vector and covariance matrix. In each scenario, the loading limit is set to 6 units. The length of each Dubins primitive is 3 meters, so each generated path has a cost that is a multiple of 3. Every solution generated by algorithms used in this work is highlighted in magenta. Each scenario presented in Fig.(5) is constructed to have a path from the start pose to the goal pose that is inadmissible due to the loading constraint. For example, these are shown using blue trajectories in Fig.(6) generated by the $\mathscr{H}A^\star$ algorithm that meet with early termination at choke points where the paths exceed total allowable path load. In each of these scenarios, there are some regions in the search space that have numerous choke points, but within these regions there also exist potential *keyholes* that contain paths which stay under the total path loading constraint while also having a relatively low cost. It is expected that the backtracking routine will be able to retract from choke points in these regions and find an admissible path through the keyhole, eventually reaching the goal pose. Finally, in each scenario, there is a set of paths that avoid the loading function entirely and have a high cost and low load, usually skirting around the boundaries of the search space: for example, see Fig.(7) showing the "minimum load solution" in blue color.

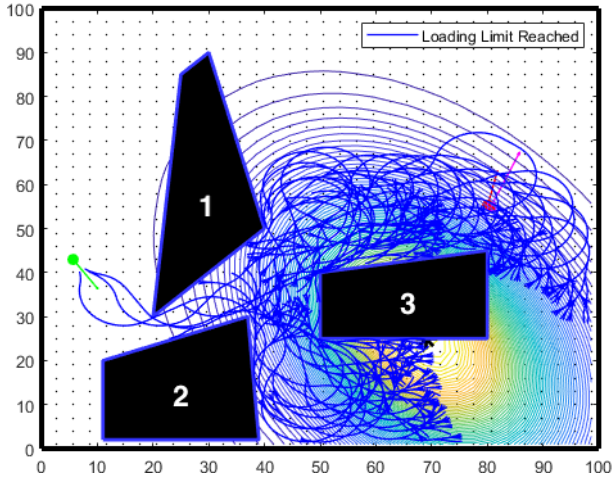
*1. Performance of Resource Constrained Hybrid A**

In this section, $\mathscr{H}A^\star$ is applied to the scenarios depicted in Fig.(5) to demonstrate the need for backtracking. Illustrative results are shown in Fig.(6). Here, the blue lines represent prematurely terminated trajectories because $\mathscr{H}A^\star$ violated the path loading limit. In scenario 1 (Fig.(6(a))), $\mathscr{H}A^\star$ is able to find a solution that successfully navigates the choke point in the common region between the three static obstacles (magenta line). The solution generated achieves a cost of 120 meters and an algorithm compute time of 0.49 seconds. In scenario 2 (Fig.(6(b))), $\mathscr{H}A^\star$ produces a solution with a cost of 150 meters with a run-time of 1.80 seconds. It is unable to find a path through the choke point



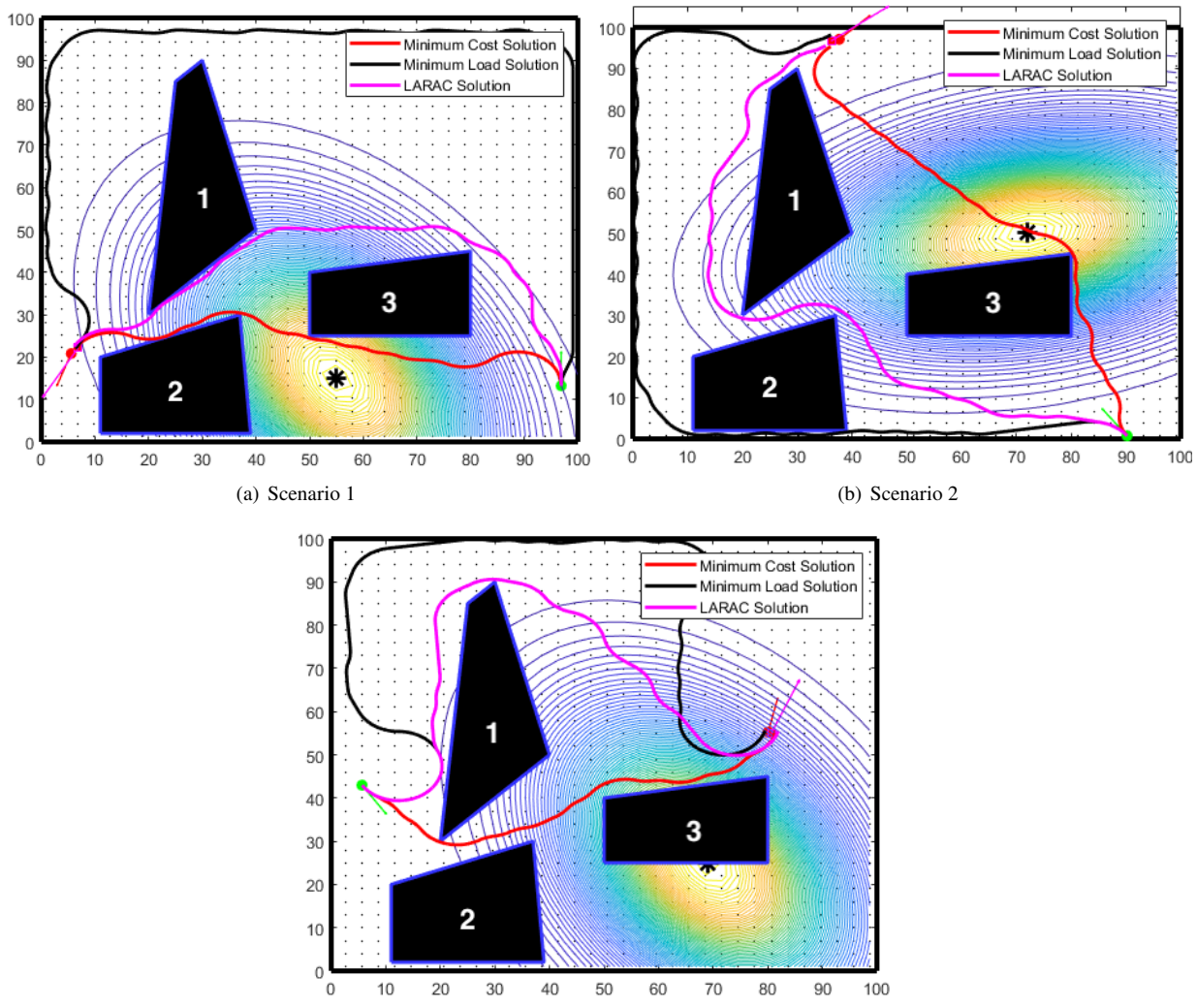
(a) Scenario 1

(b) Scenario 2



(c) Scenario 3

Fig. 6 Trajectory Generated by $\mathscr{H}A^{\star}$ in Each Test Scenario



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

Fig. 7 Optimal trajectory generated by Lagrange Relaxation Aggregate Cost in each Scenario.

between obstacles 1 and 3, and is forced to take the high cost, low load route around the obstacle # 1. In scenario 3 (Fig.(6(c))), $\mathscr{H}A^\star$ is unable to find a solution, and consumes 4.69 seconds of compute time as it out of nodes in the search frontier. Given the chosen fineness of search space discretization, an $\mathscr{H}A^\star$ solution is not possible in scenario 3 as all of the nodes that can lead to the goal are visited as $\mathscr{H}A^\star$ explores past the choke point, blocking a possible high cost solution going over the obstacles. Although $\mathscr{H}A^\star$ is able to find a good solution in scenario 1, it finds a poor solution in scenario 2, and is unable to find a solution in scenario 3. These results serve as a reference for LARAC and $\mathscr{BH}A^\star$ to be compared against.

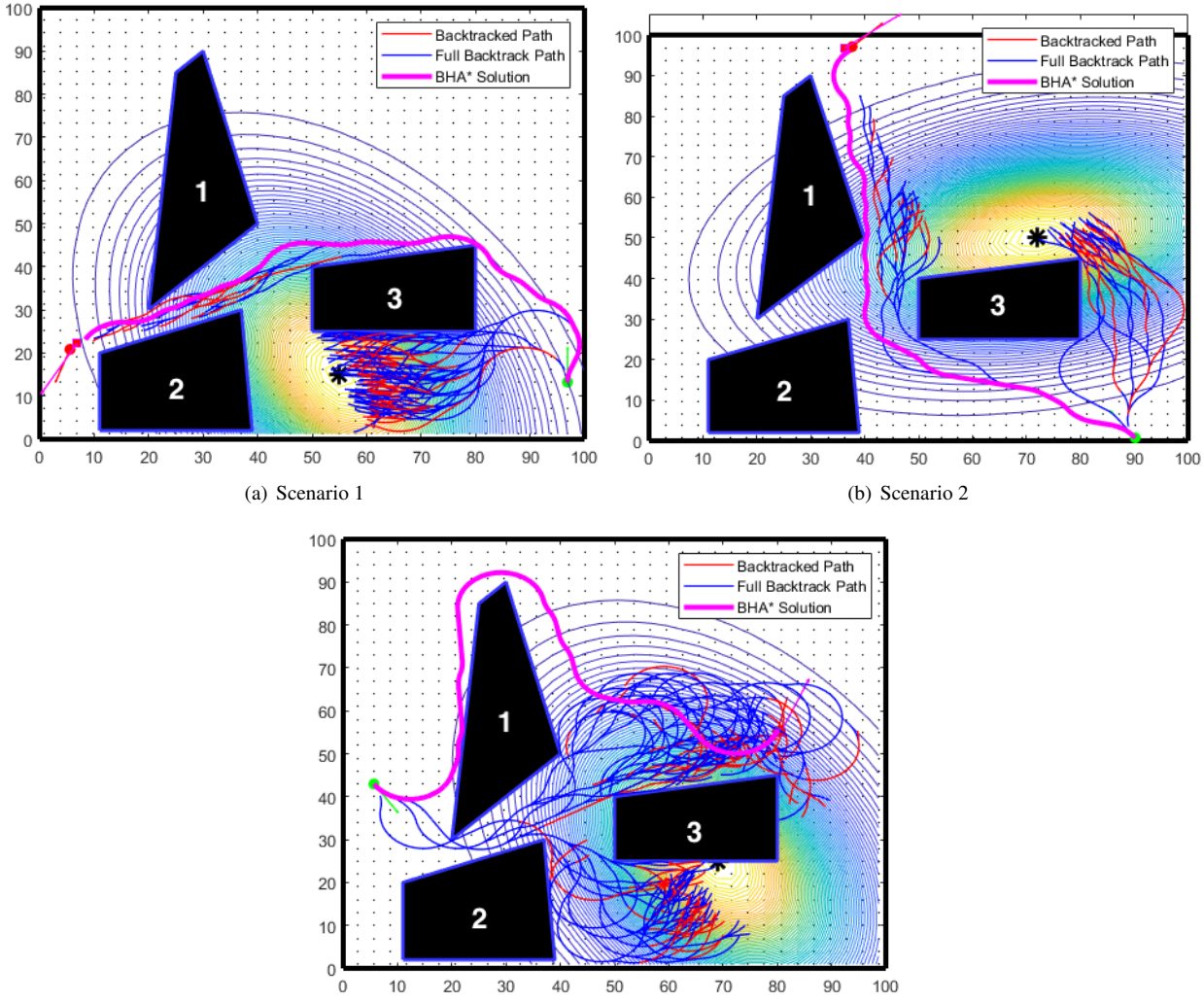
2. Performance of LARAC

In this section, LARAC is applied to the scenarios depicted in Fig.(5). Illustrative results are shown in Fig.(7). The black path represents the minimum load solution and the red path represents the minimum cost solution. In scenarios 1-3 LARAC generates solutions with costs of 126 meters, 162 meters, and 144 meters respectively. Compute times in these cases are 24.76 seconds, 26.63 seconds, and 22.19 seconds respectively. These results are shown in Fig.(7), where the red path represents the minimum cost solution, the blue path represents the minimum load solution and the magenta path represents the final LARAC solution. In all three scenarios, LARAC generates a solution with a high cost while consuming an order of magnitude longer compute times than $\mathscr{H}A^\star$. The compute time difference is attributed to the fact that LARAC relies on the repeated application of the hybrid Dijkstra's algorithm, which is especially cumbersome because it must repeatedly perform path integration to evaluate the augmented cost (Eq.(13)). The increase in cost is likely due to the fact that LARAC is minimizing the relaxed cost function, and while it is guaranteed to generate an admissible solution, it is not guaranteed to be optimal with respect to cost. Additionally, LARAC relies on a continuous search space to produce good solutions. When it is operating on a noncontinuous search space such as the ones in the scenarios where there are several distinct sets of possible solution paths, LARAC can get stuck optimizing within one of those sets and produce a sub-optimal solution.

3. Backtracking Hybrid A: Random Stopping*

In this section, results of the proposed backtracking $\mathscr{H}A^\star$ algorithm are introduced using the randomized stopping criterion (Criterion #4 in Sec.(II.E)). Random stopping is the simplest of the four stopping criteria presented in this paper and helps set a baseline for performance of the backtracking approach. As before, the path from the starting node to a choke point is shown in blue. However, part of each blue path is covered in red: this part illustrates the backwards path taken by the backtracking algorithm as it recedes from the choke point, until the stopping criterion is satisfied. The purpose of this visual is to be able to discern during the search where $\mathscr{BH}A^\star$ reaches the loading constraint limit, how far from the choke point the stopping criterion decides to backtrack, and finally, how repeated backtracking is able to guide the search. This allows us to compare various stopping criteria not just with respect to the final path produced, but also by the decisions they make during the search to navigate around choke points.

This section employs the random stopping criteria to set a baseline for backtracking performance. As explained in Sec.(II.E), backtracking stops at the current node with probability ϵ, and we set $\epsilon = 0.3$ for all the results shown in Figs.(8). To control the stochastic variability that comes from the random nature of this stopping criterion, 51 instances of the graph search were performed for each scenario. Results shown in this section represent the median cost solution for each scenario (Figs.(8)). An odd number of runs was chosen to ensure that "averaging of trajectories" is not required to compute the median. The median solution for scenario 1 using $\mathscr{BH}A^\star$ with random stopping achieves a cost of 120 meters, while consuming 0.54 seconds of compute time. This matches the performance of $\mathscr{H}A^\star$ while outperforming



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

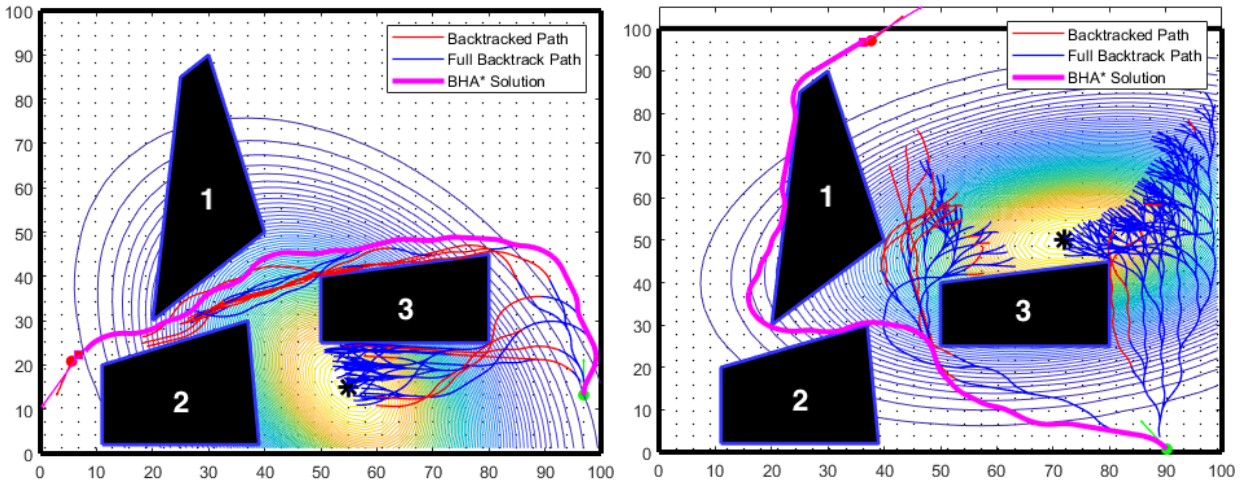
Fig. 8 Optimal Trajectory Generated by $\mathscr{BH}A^{\star}$ Using Random Stopping

LARAC by a considerable margin in terms of both compute time and path cost. In scenario 2, the random stopping $\mathscr{BH}A^\star$ median solution achieves a cost of 129 meters, while consuming 1.26 seconds. This outperforms $\mathscr{H}A^\star$ and LARAC in terms of both solution cost and compute time. Finally in scenario 3, random stopping $\mathscr{BH}A^\star$ median solution has a cost of 147 meters, while taking 5.58 seconds of compute time. This solution cost is on par with the solution produced by LARAC while still taking a fraction of the compute time. Recall that $\mathscr{H}A^\star$ is unable to find a solution for scenario 3. These results are visualized in Fig. (8). In all three scenarios, the random stopping $\mathscr{BH}A^\star$ is either on par with $\mathscr{H}A^\star$, or outperforms both LARAC and $\mathscr{H}A^\star$. The renewed exploration that backtracking allows for means that it is able to produce solutions in scenarios where $\mathscr{H}A^\star$ is simply unable to, and match or outperform $\mathscr{H}A^\star$ in terms of both cost and compute time. LARAC is able to keep up in terms of cost, however, the computational overhead induced by backtracking is far outweighed by the iterative application of Dijkstra's Algorithm used in LARAC.

4. Improved Backtracking Stopping Criteria

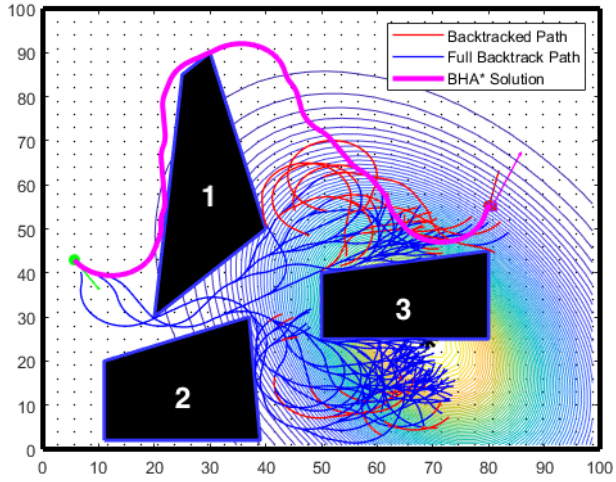
Random stopping is a simple technique that does not involve any deliberate logic to guide the search. It nevertheless performs well in scenarios outlined in Sec.(II.F) in terms of the median cost solution. Indeed, there is variability around the median cost and the performance gets much worse when considering the worst case solution offered by backtracking with random stopping. In the worst case scenario for random stopping, both LARAC and $\mathscr{H}A^\star$ outperform $\mathscr{BH}A^\star$ in terms of path cost. In this section, we discuss the use of other stopping criteria described in Sec.(II.E). Unlike random stopping, these criteria leverage some aspect of the loading function to ascertain when to stop the backtracking process. Random stopping serves as a baseline for comparison among various backtracking strategies.

Auxiliary optimization stopping (criterion 1 in Sec.(II.E)) uses the minimum load solution to every node in the search domain to determine the stopping condition. The relaxation parameter ξ is used to determine how close the load at the current node must be to the minimum possible load at this node to stop at the backtracking process. $\mathscr{BH}A^\star$ using auxiliary optimization stopping is applied to the scenarios 1-3, with relaxation parameter value set at 1.4. Note that the disadvantage of this method is that there is no obvious choice of numerical value for the relaxation parameter and a given relaxation parameter is not expected to perform equally across different scenarios. In this study, a trial and error approach was used to study performance of backtracking across a range of scenarios, leading to the reported selection of $\xi = 1.4$, which appears to perform well. Recall that $\xi = 1.4$ implies that backtracking will stop at the current node if the actual loading accrued along the trajectory is no more than 40% of the minimum possible load at this node. In scenario 1-3, $\mathscr{BH}A^\star$ using auxiliary optimization stopping achieves costs of 126 meters, 159 meters, and 147 meters respectively, and it is able to achieve these results with compute times of 5.35 seconds, 6.41 seconds, and 7.76 seconds respectively. These results are visualized in Fig. (9). These relatively long compute times are due to the solution of the offline auxiliary load optimization problem which requires Dijkstra's algorithm to traverse the entire search space. The poor performance relative to median cost solution of random stopping (in terms of path cost) is the result of keeping a



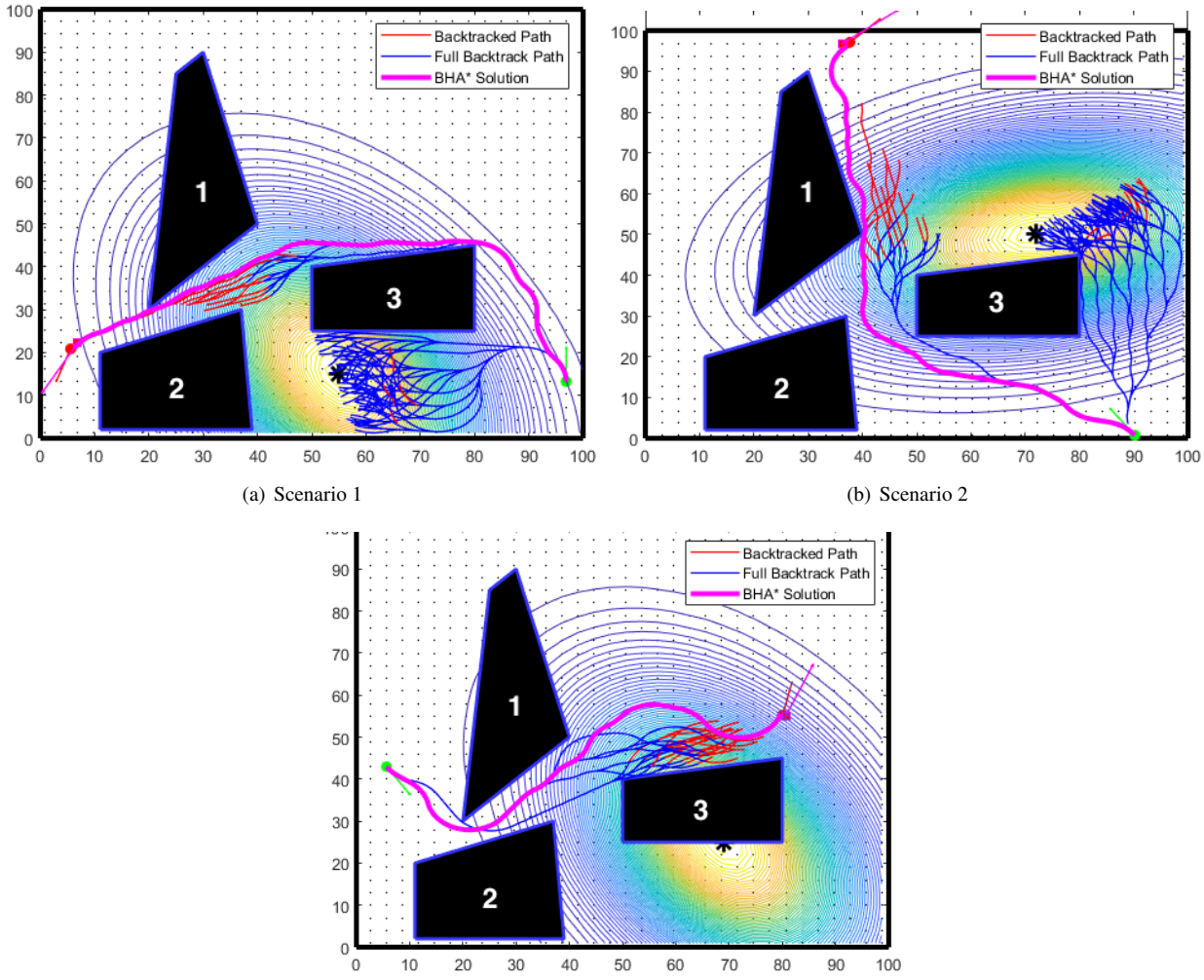
(a) Scenario 1

(b) Scenario 2



(c) Scenario 3

Fig. 9 Optimal Trajectory Generated by $\mathscr{BH}A^{\star}$ Using Auxiliary Optimization Stopping



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

Fig. 10 Optimal Trajectory Generated by $\mathscr{BH}A^\star$ Using Maximum Edge Load Stopping

uniform relaxation factor across scenarios.

Finally, maximum edge load stopping (criterion 3 in Sec.(II.E)) is a simplification over the load rate stopping approach. The goal is to start the re-exploration where the search is accumulating load at the fastest rate along the current path. Maximum edge load stopping is also not dependant on any tuning parameters, and does not require offline computation to function. In scenarios 1-3 $\mathscr{BH}A^\star$ using maximum edge load stopping was able to achieve a cost of 120 meters, 129 meters, and 99 meters, it was able to achieve these results in compute times of 0.65 seconds, 1.39 seconds, and 0.56 seconds. The results are visualized in Fig. (10).

Upon comparing results across scenarios it can be seen that auxiliary optimization stopping suffers primarily from having to run the load minimizing Dijkstra's algorithm which drives up the algorithm run-time consistently for each scenario. Random stopping is able to consistently find low cost solutions that compete with the solutions generated by

Scenario	Stopping Criterion	Terminal Cost (m)	Terminal Load	Compute Time (s)
1	$\mathscr{H}A^\star$	**120**	5.92	**0.49**
	LARAC	126	4.45	24.76
	Auxiliary Optimization Stopping	126	5.34	5.35
	Random Stopping (Median Cost)	**120**	5.92	0.54
	Maximum Edge Load Stopping	**120**	5.68	0.65
2	$\mathscr{H}A^\star$	159	3.16	1.80
	LARAC	162	2.13	26.63
	Auxiliary Optimization Stopping	159	2.73	6.41
	Random Stopping (Median Cost)	**129**	5.06	**1.26**
	Maximum Edge Load Stopping	**129**	4.83	1.39
3	$\mathscr{H}A^\star$	159	5.12	4.69
	LARAC	144	3.59	22.19
	Auxiliary Optimization Stopping	147	4.92	7.76
	Random Stopping (Median Cost)	147	4.36	5.58
	Maximum Edge Load Stopping	**99**	5.70	**0.56**

Table 1 Table Comparing $\mathscr{BH}A^\star$ Performance in Different Scenarios with Different Stopping Criteria. Best Cost and Least Compute Time are Shown in Bold.

maximum edge load stopping. However, the variance inherent to random stopping means that those low cost solutions are not guaranteed. Both auxiliary optimization stopping and random stopping rely on tuning parameters, for which it is difficult to prescribe general guidelines for selecting numerical values that produce good results across different scenarios. Finding the best parameter for each scenario adds a layer of complexity to the search that maximum edge load stopping is able to avoid. The maximum edge load stopping approach provides the most consistently good solutions while using less compute time.

III. Application

This chapter presents the design and development of a UAS platform and its integration in a wild-land burn environment using planning techniques detailed in Section II. The primary focus of this endeavor is to demonstrate online, onboard planning and tracking in a wild-land burn. The first problem that must be solved to achieve this integration is the development of a reliable and well-documented UAS platform. Then a system to manage accepting guidance goals, running the planner algorithm, and tracking the resulting trajectory must be developed. All system components must be rigorously tested as all operations within the wild-land burn environment will be Beyond Visual Line of Sight (BVLoS) for the UAS. The situational awareness required to fully account for temperature-based resource constraints is left to future work.

A. Experimental Platform

Many commercially available UAS exist such as DJI Mavic 3 [22] are capable of flying in prescribed wild-land burn environments, however, their source code is closed source and inflexible in their operation. One could not modify a DJI product with custom planning and tracking capabilities. To allow for the implementation of a custom onboard guidance solution an open-source flight stack must be chosen. PX4 Autopilot is an open-source flight stack that is compatible with many off-the-shelf drone components [23].

1. Unmanned Aerial System

The starting point for the UAS was an in-house system known as Eagle [1]. Eagle was originally designed as a remote sensing platform and carried a gimbal-mounted Flir Duo camera [24] capable of simultaneous infrared and RGB imaging. This data could be streamed to a ground receiver via a 5.8 Ghz analog video transmitter. The Eagle was built on a DJI F450 quadrotor airframe, paired with four DJI 2312E motors, DJI 430 Lite ESCs, and DJI 9.4"x5" propellers. It was equipped with a HolyBro Pixhawk 4 flight control board running PX4 Autopilot. Additionally, the airframe is equipped with GPS receiver, Radio Control receiver (2.4 Ghz), telemetry receiver (915 Mhz), and a RUSHFPV Tank Solo (5.8 Ghz) video transmitter for and sensor package default to the HolyBro Pixhawk 4 flight board. This package includes an IMU, Magnetometer, and Barometer to facilitate state estimation. The assembled Eagle platform can be seen in Fig. (11). The Eagle can be commanded via telemetry using a laptop running QGroundControl [25]. QGroundControl is an open-source ground station application that can communicate with Pixhawk flight control boards using the MAVLink [26], an open-source micro air vehicle marshaling protocol. Using QGroundControl, the pilot can command the UAS to change flight modes, take off, land in place, return to home, and plan and manage basic waypoint tracking missions. The Eagle could also be controlled manually using a FrSky Taranis X9D remote controller. This allows for manual control of the UAS's position, attitude, and camera gimbal position if needed. The eagle could be mounted with a companion computer, i.e. a computer that does not handle the PX4 flight stack and is able to perform

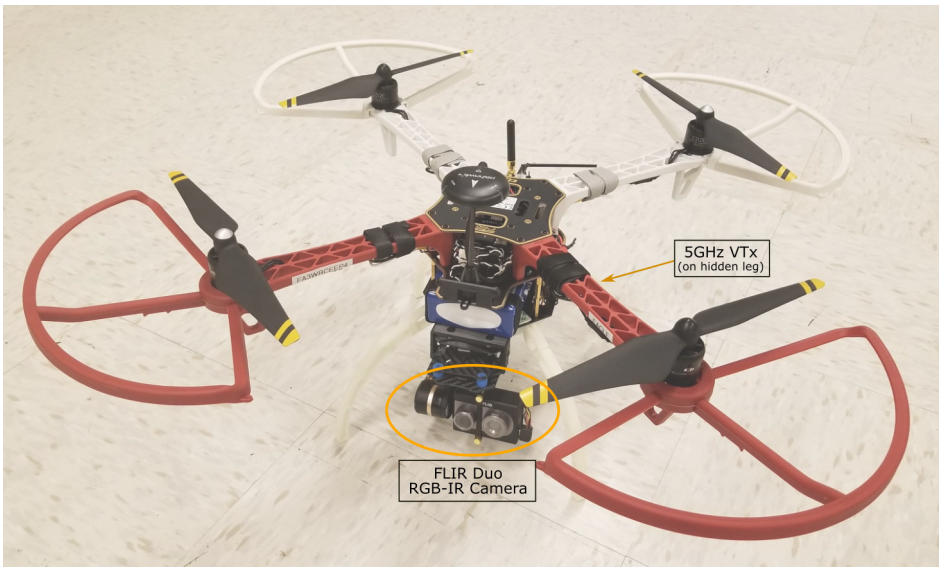


Fig. 11 Eagle UAS Built on a Modified DJI Flamewheel F450 [1]

guidance, navigation, and control tasks supporting the flight controller.

The Eagle proved to be an unreliable platform in flight testing and in prescribed burns. Significant attitude vibrations were present when attempting to maintain a position hold, and accelerometer and magnetometer issues caused persistent estimator errors as can be seen in Fig. (12). This caused several catastrophic crashes during flight testing, all of which resulted in complete rebuilds of Eagle. This proved cumbersome due to the design of the DJI F450 frame. The sensors, subjected to multiple crashes and seemed to be the consistent point of failure, were not readily replaceable. These failures prompted a complete redesign of the UAS platform with the goal of improving reliability.

This was dubbed the BlueBird project. The primary design goal was to meet or surpass Eagle's payload and flight time, maintain compatibility with the camera gimbal system, and to be able to have a variety of companion computers be able to be mounted. A BlueBird UAS would be built using a standard template recipe for easy reproduction. The frame chosen was the modern HolyBro X500 V2. The X500 V2 provides quick access to the Pixhawk board which makes servicing the platform simple rekative to Eagle. Aikon 32-bit 35A ESCs paired with TMotor MN3110 motors and TMotor MS1101 11"x4.2" were used. A HolyBro Pixhawk 6C was chosen as the flight control board, as it is a more recent update of the Pixhawk 4 board which includes a modern sensor package. The same radio control and telemetry system were used as in Eagle. The result was a platform that had a flight time of 23 minutes, and a maximum range of 1 kilometer. Additionally, there is a complete parts list and build tutorial so that the BlueBird template can be used to easily construct new BlueBird platforms.

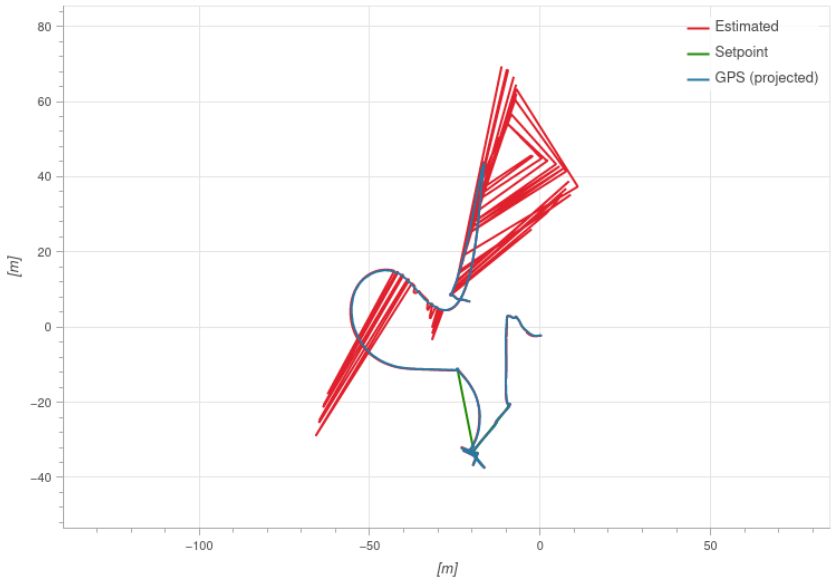


Fig. 12 Estimator error causes wild variations in Eagle's estimated position during a flight test.

(a) Isometric View (b) Top View

Fig. 13 Base BlueBird UAS Platform

2. Introduction to ROS

The Robot Operating System (ROS) is an open-source robotics middleware. It serves as a framework to compartmentalize processes, manage information flow, low-level device control, and package management. A ROS application is comprised of a ROS master, nodes, topics, and services [27]. The ROS master is the central process organizing communication between nodes. ROS nodes can encapsulate processes or algorithms and can communicate

with each other by publishing standardized messages to topics and subscribing to topics. They can also advertise and request services.

The primary difference between messages and services are that messages are one-way communication, while services are two way communication. Subscribing to a topic means that a ROS node will receive a stream of data in the form of messages. Requesting a service entails that the requesting node will send some data to the service node, which will then execute some code, then send a response back to the requesting node.

ROS has an ecosystem where open-source packages can be installed and used to provide functionality that uses standardized ROS messages and services. A ROS package that is maintained as part of the PX4 project is MAVROS [28]. MAVROS is composed of a ROS node that handles the hardware abstraction and acts as a translation layer between a MAVLink stream and ROS topics and messages. This allows a ROS network to communicate with a PX4 flight stack using the MAVLink protocol over a serial port. A MAVROS node running on a companion computer mounted on the UAS platform can enable offboard control of PX4, i.e., where a portion of the feedback control loop is handled on a companion computer.

3. ROS Application Design

The goal of the ROS application is to facilitate guidance and control operation on board the UAS platform in real-time. The system must be able to operate without a WiFi connection to the ground station as it must be able to operate at the 1-kilometer range that the BlueBird is capable of where high bandwidth network communication is not simple. The ROS application does not have any situational awareness so obstacle avoidance is impossible. This means that the ROS application must have a failsafe where a pilot can retake manual control in case of a potential collision.

To meet these requirements the ROS application is organized around a state transition diagram as seen in Fig. (14). There are four possible states:

- Idle: The ROS application starts in idle. When the ROS application is in Idle, it is only waiting for a guidance command from QGroundControl.
- Planning: When the ROS application receives a guidance command, it calls the planner service given its current position and guidance goals. If the service returns a planned trajectory, the ROS application transitions to the Tracking state, if the planner fails, the ROS application transitions back to the Idle state.
- Tracking: When the ROS application receives a planned trajectory it begins tracking. The ROS application triggers PX4 to transition to Offboard control and begins publishing setpoints. For the duration of the tracking, if the pilot uses the radio control to change PX4's flight mode, the ROS application abandons the track and transitions to Radio Control (RC) mode. If the track completes without interruption, the ROS application transitions back to Idle.
- RC: At any point, if the pilot changes PX4's state at any point, the current operation is abandoned and the ROS

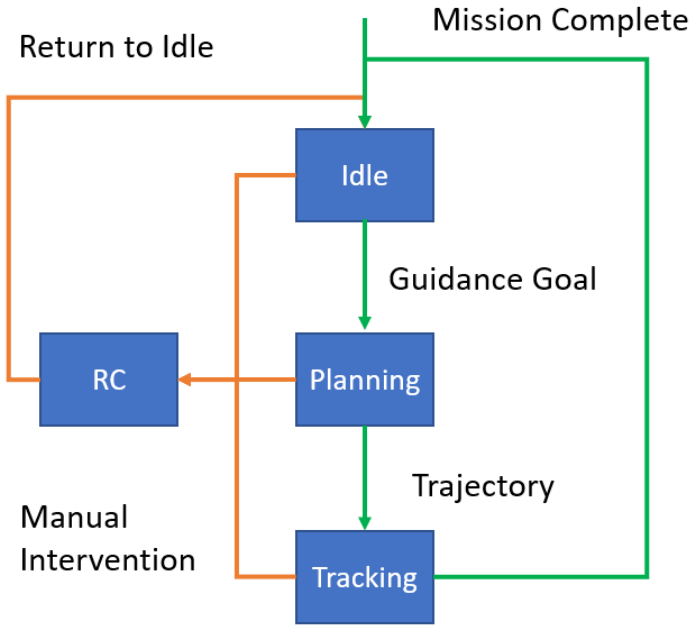


Fig. 14 ROS Application State Diagram

application remains dormant until PX4 is transitioned back to Hold. At this point, the ROS application will transition back to idle.

The ROS application receives guidance goals by monitoring PX4 for waypoint missions. When PX4 has received a mission from QGroundControl, this mission can be retrieved using MAVROS. Guidance goals are encoded as the waypoints in a waypoint mission. When PX4 is transitioned into mission mode from QGroundControl, the network halts the mission before it begins, and the ROS application transitions into Planning mode. The advantage of this method is that guidance goals can be delivered to the ROS application using purely the telemetry radio, which operates at 915 Mhz and has the longest range.

The path planner is an implementation of $\mathscr{H}A^{\star}$ as seen in Algorithm (2). It takes a starting pose $\{x_0, y_0, \psi_0\}$ defined by the current pose of the platform and goal pose $\{x_f, y_f, \psi_f\}$ defined by the waypoint pose. The trajectory altitude is set to the altitude of the waypoint. The turning radius and the platform speed can be set when the ROS application is launched. The trajectory generated consists of a series of timestamped setpoints that can be published to PX4. This is a very simple trajectory tracker that does not use an acceptance radius to confirm setpoints and only includes a pose setpoint, not a twist setpoint.

The integration is achieved using a Raspberry Pi 4 [29] running Ubuntu 20.04 [30]. The ROS package RobotUpstart [31] by Clearpath Robotics. RobotUpstart enables the creation of background jobs on Linux to launch ROS networks on boot, meaning that there is no need for outside interaction with the ROS application on the RPi on boot.

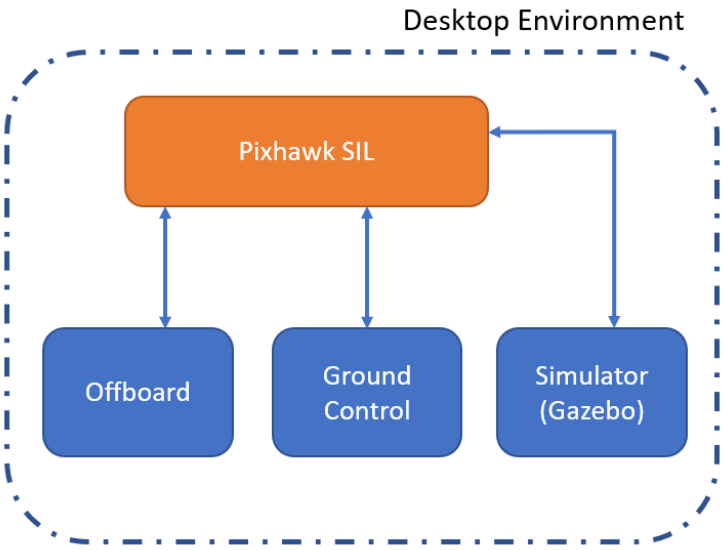


Fig. 15 ROS Application Software In the Loop (SIL)

4. Simulation Driven Development

Simulation is an incredibly important piece of autonomous system development. It allows for the verification of the functionality of a ROS application without having to set up and risk real hardware. This enables rapid prototyping of features that would take weeks to develop and test without simulation. Software In the Loop (SIL) testing/simulation is defined by the emulation of the PX4 flight stack in a desktop environment. The emulated PX4 interacts with a Gazebo [32] simulation of the quadrotor dynamics. A UDP port serving as a MAVLink connection is exposed, allowing a MAVROS node and QGroundControl to be run alongside the simulation. This configuration can be seen in Fig. (15). The Offboard block represents the ROS application and other Offboard processes that may be active. The state diagram for the ROS application was developed in a SIL simulation. The conditions for transitions between states were identified and tested in simulation before being validated on BlueBird.

Hardware In the Loop simulation is very similar to SIL in setup, but the purpose is very different. In HIL PX4 is running on the Pixhawk 6C Flight Control Unit, and the ROS application is running on the RPi. As seen in Fig. (16), the RPi (Offboard) is connected to the FCU via a UART port, and PX4 is connected to QGroundControl and Gazebo running on a desktop environment via a USB Port. This setup allows for the performance of the planning algorithm to be verified on the RPi as previously it had only been tested in a desktop environment. It also allows for the bandwidth between the Pixhawk 6C and the RPi to be tested. This can prove to be an issue if the Offboard computer is part of the feedback control loop. It was determined in HIL testing that using a UART connection between the RPi and the Pixhawk 6C was limited to a baud rate of 57600. This resulted in communication delays between the RPi and the Pixhawk 6C that peaked at 200 ms, which is far too slow to perform accurate trajectory tracking. This was remedied by using a USB

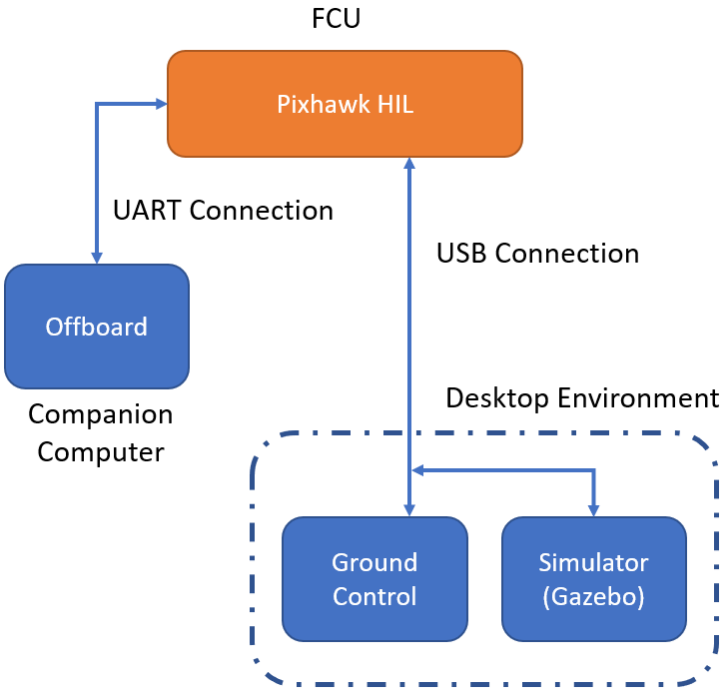


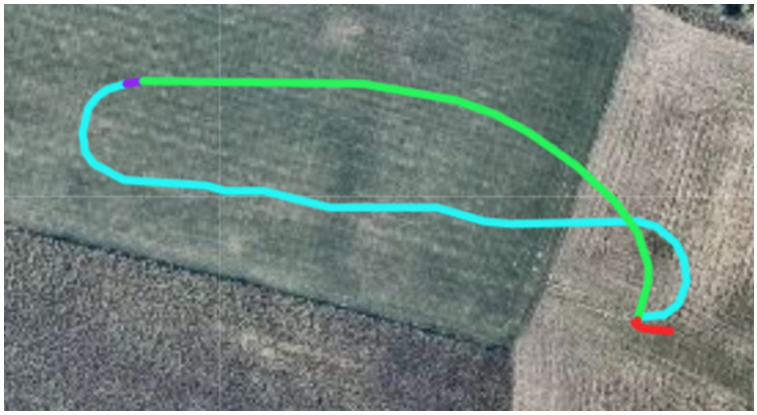
Fig. 16 ROS Application Hardware In the Loop (HIL)

connection between the RPi and the Pixhawk 6C, which has a baud rate of 115200 and did not produce any significant communication lag.

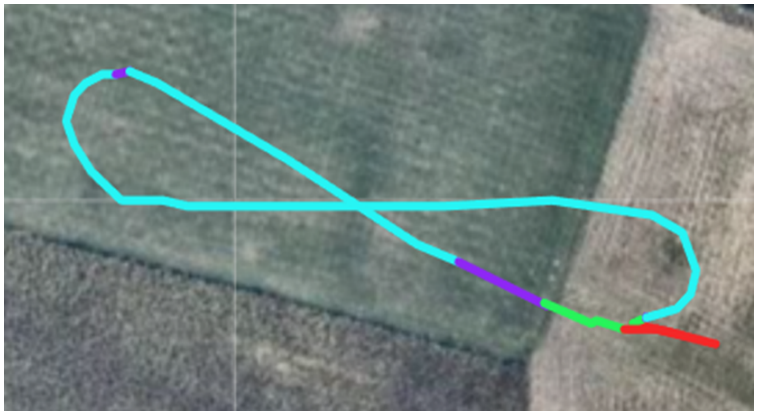
B. Flight Testing

The BlueBird UAS platform is used to test the ROS application functionality. Fig. (17) provides a visualization of the BlueBird platform tracking paths planned via $\mathscr{H}A^{\star}$ onboard using the ROS application. In Fig. (17) the blue lines represent what the UAS platform is under Offboard control. Guidance goals were sent to the ROS network via telemetry radio from QGroundControl running on a laptop. All tests were performed as if the UAS was BVLoS to verify that the system is capable of performing planning and tracking operations without pilot observation.

A prescribed burn was performed in Zelesky State Forest in South East Ohio in March 2023. During the year leading up to the burn test flights were performed using a DJI Mavic 3 to practice flying in the forest environment. There was no open field to launch from, and when the UAS is no longer directly above the launch point it is BVLoS. During the burn, about 100 square acres of forest were ignited. The BlueBird platform was flown during the burn. The Flir Duo camera was used to scout for high-interest regions, and then using the ROS application, guidance goals were set, and a trajectory was planned onboard and tracked. The Flir Duo was used to film the evolution of fire in infrared. Fig. (18) is a frame of an infrared video taken by BlueBird during the burn.



(a) Test 1



(b) Test 2

Fig. 17 Onboard, online demonstration of planning and tracking using BlueBird platform in live flight test.

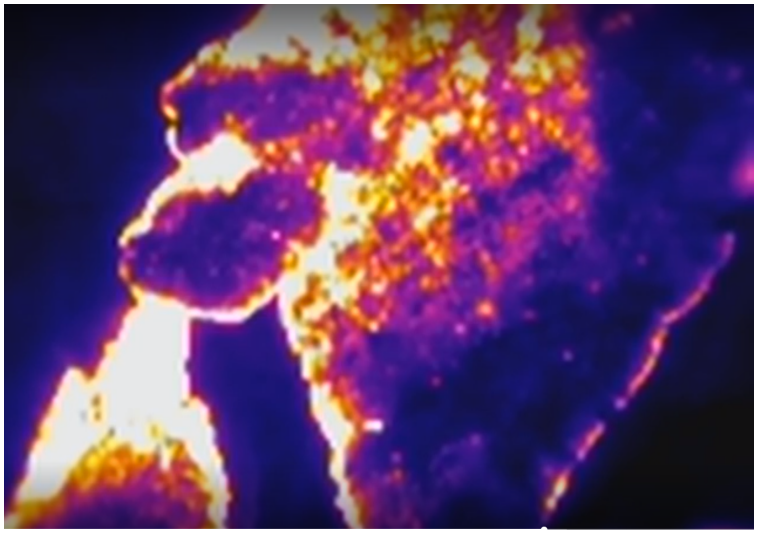


Fig. 18 Infrared image of the fire taken by BlueBird at Zelesky State Forest burn (03/21/2023)

IV. Conclusion

In this work, a new approximation method utilizing backtracking applied to $\mathscr{H}A^\star$ is presented as an approximation method for the Resource Constrained Shortest Path Problem. Various implementations of a backtracking-stopping criterion are explored. Its performance is verified against a Lagrange Relaxation based method and $\mathscr{H}A^\star$ without backtracking. It is shown to outperform both in the given test scenarios. An Unmanned Aerial System platform is designed and tested, improving on previous UAS iterations developed in the lab. Additionally, a ROS application is developed to allow for path planning to be performed online, onboard the UAS platform. It is rigorously tested in simulation using SIL and HIL techniques and in live flight tests. The platform is used in a prescribed burn to demonstrate online, onboard planning and tracking, while also recording infrared video of the fire.

A. Future Work

There are several challenges that will need to be addressed in the future. It is important to study the theoretical performance of backtracking as a method to solve RCSPPs. This includes theoretical time complexity and performance guarantees. Backtracking $\mathscr{H}A^\star$ is able to outperform LARAC in this scenario, but it is valuable to determine if this is the case in different RCSPP formulations.

Another major aspect that needs to be addressed is the UAS's situational awareness. The RCSPP being defined using the heat flux contours of the fire relies on online onboard construction of those heat contours. Infrared images alongside platform state information can be used to build a global map of the fire. This capability is useful for post-mission analysis as well as it can be used to estimate the rate of spread of the fire are a given time and location.

Finally, the tracking used in the ROS application is a purely timed position, setpoint-based method. It does not utilize any state feedback so tracking performance is poor. This causes the UAS to be unable to meet the target speed assumed in path planning, as well as introducing roll and pitch oscillations due to setpoint changes. This means that path tracking is very power intensive. This issue can be remedied by implementing a trajectory-tracking controller in the ROS application.

B. Research Products

- Journal Papers
    1) Ford, B., Aggarwal, R., Kumar, M., Manyam, S.G., Casbeer, D., and Grymin, D., "Backtracking Hybrid A* for Resource Constrained Path Planning", *in preparation*
    2) Cortez, A., Ford, B., Nayak, I., Narayanan, S. and Kumar, M., "Hybrid $A^\star$ Path Search with Resource Constraints and Dynamic Obstacles", *Frontiers in Aerospace Engineering*, Special Issue on Enabling Technologies for Advanced Air Mobility, 2023, https://doi.org/10.3389/fpace.2022.1076271
- Conference Papers

1) Cortez, A., Ford, B., Nayak, I., Narayanan, S. and Kumar, M., "Path Planning for a Dubins Agent with Resource Constraints and Dynamic Obstacles", Guidance, Navigation and Control Conference at Scitech Forum, AIAA Paper 2023-1054, National Harbor MD, Jan 23-27, 2023, https://doi.org/10.2514/6.2023-1054

2) Ford, B., Aggarwal, R., Kumar, M., Manyam, S.G., Casbeer, D., and Grymin, D., "Backtracking Hybrid A^* for Resource Constrained Path Planning", Guidance, Navigation and Control Conference at Scitech Forum, AIAA Paper 2022-1592, San Diego CA, Jan 3-7, 2022, https://doi.org/10.2514/6.2022-1592

- Conference Presentations

1) Ford, B., "Path Planning for a Dubins Agent with Resource Constraints and Dynamic Obstacles", 2023 AIAA SciTech Forum, National Harbor MD, Session: *Modeling and Simulation for Autonomous Guidance, Navigation, and Control III*, Video Presentation: https://doi.org/10.2514/6.2023-1054.vid

2) Ford, B., "Backtracking Hybrid A^* for Resource Constrained Path Planning", 2022 AIAA SciTech Forum, San Diego CA, Session: *Guidance, Navigation and Control in Intelligent Systems*, Video Presentation: https://doi.org/10.2514/6.2022-1592.vid

References

[1] Aggarwal, R., "Chance-Constrained Path Planning in Unstructured Environments," Ph.D. thesis, The Ohio State University, 2021.

[2] Aggarwal, R., Soderlund, A., Kumar, M., and Grymin, D., "Risk Aware SUAS Path Planning in an Unstructured Wildfire Environment," *2020 American Control Conference (ACC)*, 2020, pp. 1767–1772. https://doi.org/10.23919/ACC45564.2020.9147696.

[3] Thomas, B. W., Calogiuri, T., and Hewitt, M., "An exact bidirectional A* approach for solving resource-constrained shortest path problems," *Networks*, Vol. 73, 2019, p. 187–205.

[4] Ahujia, R., Magnanti, T. L., and Orlin, J. B., "Network flows: Theory, algorithms and applications," *New Jersey: Rentice-Hall*, 1993.

[5] Bertsekas, D. P., "Nonlinear Programming," 1999.

[6] Scott, D., Manyam, S. G., Casbeer, D. W., Kumar, M., Rothenberger, M. J., and Weintraub, I. E., "Power Management for Noise Aware Path Planning of Hybrid UAVs," *2022 American Control Conference (ACC)*, 2022, pp. 4280–4285. https://doi.org/10.23919/ACC53348.2022.9867385.

[7] Dijkstra, E. W., "A note on two problems in connexion with graphs," *Numerische Mathematik*, Vol. 1, 1959, p. 269–271.

[8] Hart, P., Nilsson, N., and Raphael, B., "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, Vol. SCC-4, No. 2, 1968, p. 100–107.

[9] Daniel, K., Nash, A., Koenig, S., and Felner, A., "Theta*: Any-Angle Path Planning on Grids," *Journal of Artificial Intelligence Research*, Vol. 39, 2010, pp. 533–579.

[10] Dolgov, D., Thrun, S., a, M. M., and Diebel, J., "Path Planning for Autonomous Driving in Unknown Environments," *Experi. Robotics: The 11th Intern. Sympo., STAR 54*, Changshu, Suzhou, China, June 26-30, 2018, p. 55–64.

[11] Petereit, J., Emter, T., Frey, C. W., Kopfstedt, T., and Beutel, A., "Application of Hybrid A* to an Autonomous Mobile Robot for Path Planning in Unstructured Outdoor Environments," *Robotik 2012: 7th German Conference on Robotics*, Munich, Germany, 2012.

[12] Shkel, A. M., and Lumel, V., "Classification of the Dubins set," *Robotics and Autonomous Systems*, Vol. 34, 2001, pp. 179–202.

[13] Reeds, J., and Shepp, L., "Optimal paths for a car that goes both forwards and backwards," *Pacific Journal of Mathematics*, Vol. 145, 1990, p. 367–393.

[14] Banzhaf, H., Berinpanathan, N., Nienhuser, D., and Zollner, J. M., "From G2 to G3 Continuity: Continuous Curvature Rate Steering Functions for Sampling-Based Nonholonomic Motion Planning," *IEEE Intelligent Vehicles Symposium (IV)*, Changshu, Suzhou, China, June 26-30, 2018.

[15] Oliveira, R., Lima, P. F., Cirillo, M., Martensson, J., and Wahlberg, B., "Trajectory Generation using Sharpness Continuous Dubins-like Paths with Applications in Control of Heavy-Duty Vehicles," *European Control Conference (ECC)*, Limassol, Cyprus, June 12-15, 2018.

[16] Dolgov, D., Thrun, S., Montemerlo, M., and Diebel, J., "Practical Search Techniques in Path Planning for Autonomous Driving," *Ann Arbor*, Vol. 1001, 2008, p. 48105.

[17] Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., and Teller, S., "Anytime Motion Planning using the RRT*," *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1478–1483. https://doi.org/10.1109/ICRA.2011.5980479.

[18] Drysdale, D., *An introduction to fire dynamics*, John Wiley & Sons, 2011.

[19] Mehlhorn, K., and Ziegelmann, M., "Resource Constrained Shortest Paths," *Algorithms - ESA 2000*, edited by M. S. Paterson, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 326–337.

[20] Jüttner, A., Szviatovski, B., Mécs, I., and Rajkó, Z., "Lagrange relaxation based method for the QoS routing problem," *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, Vol. 2, IEEE, 2001, pp. 859–868.

[21] Xiao, Y., Thulasiraman, K., Xue, G., Jüttner, A., and Arumugam, S., "The constrained shortest path problem: algorithmic approaches and an algebraic study with generalization," *AKCE International Journal of Graphs and Combinatorics*, Vol. 2, No. 2, 2005, pp. 63–86.

[22] "DJI Professional Drones," https://www.dji.com/products/professional/.

[23] "PX4 Open Source Autopilot," https://px4.io/.

[24] "Teledyn Flir Duo Camera," https://www.flir.com/support/products/duo-r/#Overview.

[25] "QGroundControl - Ground Control Station for MAVLink Protocol," http://qgroundcontrol.com/.

[26] "MAVLink - MAV Communication Protocol," https://mavlink.io/.

[27] Stanford Artificial Intelligence Laboratory et al., "Robotic Operating System," URL https://www.ros.org.

[28] "MAVROS - ROS package for MAVLink," https://github.com/mavlink/mavros.

[29] "Raspberry Pi 4 Model B," https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[30] "Ubuntu 20.04 LTS - Focal Fossa," https://releases.ubuntu.com/focal/.

[31] "Robot Upstart," http://wiki.ros.org/robot_upstart.

[32] "Gazebo - Open Source Robotics Simulation," https://gazebosim.org/home.