Any-Time Path-Planning: Time-Varying Wind Field + Moving Obstacles

Michael Otte[1,2], William Silva[1], and Eric Frew[1]

***Abstract*— We consider the problem of real-time path-planning in a spatiotemporally varying wind-field with moving obstacles. We are provided with changing wind and obstacle predictions along a $(D+1)$-dimensional space-time lattice. We present an Any-Time algorithm that quickly finds an $\alpha\beta$-suboptimal solution (a path that is not longer than $\alpha\beta$ times the optimal time-length), and then improves α and β while planning time remains or until new wind/obstacle predictions trigger a restart. The factor α comes from an α-overestimate of the A*-like cost heuristic. β is proportional to motion modeling error. Any-Time performance is achieved by: (1) improving the connectivity model of the environment from a discrete graph to a continuous cost-field (decreasing β); (2) using the established method of incrementally deflating α. Our method was deployed as the global planner on a fixed-wing unmanned aircraft system that uses Doppler radar and atmospheric models for online real-time wind sensing and prediction. We compare its performance vs. other state-of-the-art methods in simulated environments.**

I. INTRODUCTION

Unmanned Aircraft Systems (UAS) and Autonomous Underwater Vehicles (AUVs) often have top speeds that are less than those of the currents through which they travel. In these cases, accounting for wind is as important as avoiding obstacles to ensure feasible/optimal motion, and necessary for avoiding collisions. Considering wind is challenging; wind speeds vary across time and space, and forecasts become increasingly inaccurate into the future and often change. We present a novel real-time method for long-term path-planning that considers spatiotemporally varying dynamic wind and dynamic moving obstacles.

We desire a time-optimal solution, and assume the UAS moves at a constant airspeed. Our method is *Any-Time*: it quickly finds a feasible[1] path of bounded suboptimality, and then improves this solution as long as planning time remains. Each solution is no worse than $\alpha\beta$ times the optimal solution length. α comes from an α-overestimate of the A*-like cost heuristic and decreases vs. planning time [1]. β is a function of motion modeling error and decreases as our hypergraph connection model evolves from being discrete to continuous. Simple graph-based (A*-like [2]) solutions are found quickly, while more optimal continuous (Field-D*-like [3]) solutions can be used if enough planning time exists to calculate them. α decreases separately vs. each value of β.

This work was supported by the Control Science Center of Excellence at the Air Force Research Laboratory, the National Science Foundation grant IIP-1161029, the Center for Unmanned Aircraft Systems, and the Air Force Office of Scientific Research under grant FA9550-12-1-0412.

[1]Aerospace Engineering Sciences, University of Colorado at Boulder, Campus Box 429, Boulder, CO 80309 ottemw@gmail.com

[2]This work was done while the author was "in residence" at AFRL.

[1]A feasible path is a path that causes the UAS to move from start to goal without intersecting obstacles.

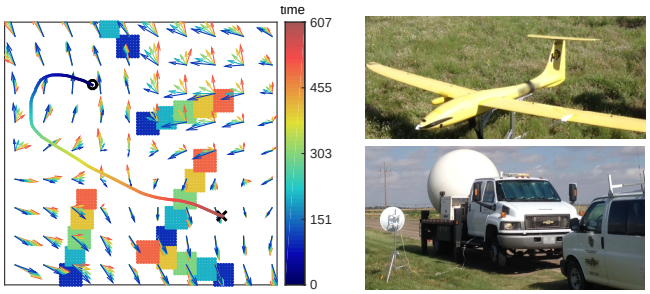

Fig. 1: Left: Our method generates a space-time path that respect spatiotemporally varying wind (vectors) and moving obstacles (blocks); color represents time. Right: Our method was deployed as the global planner on a UAS (top) that uses Doppler radar (bottom) for wind data and atmospheric models for wind predictions.

We assume a space-time lattice of wind values is provided, and may include a sequence of wind predictions into the future. Obstacles are handled using a space-time occupancy grid. In practice, we find that updated wind predictions tend to modify *all* values in the wind lattice — eliminating the usual benefits of incremental repair-based replanning (e.g., D*-Lite [4]). Consequently, we use a brute-force replanning approach that restarts Any-Time planning whenever wind/obstacles values/predictions change. Our method is designed for long-term global path-planning *within* a hierarchical planning framework, and assumes a local planner exists to optimize solutions with respect to vehicle kinodynamics.

Our method was deployed in Lubbock, Texas, in June of 2015, as part of a fixed-wing UAS using Doppler radar and atmospheric models for online planning (AMOP) [5] for real-time wind sensing and prediction, respectively. We supplement this real-world experience with simulations comparing performance vs. other state-of-the-art methods in a number of challenging wind scenarios.

This paper is organized as follows: Section II surveys related work, Sections III and IV describe notation and our algorithm, respectively. Experiments, Discussion, and Conclusions appear in Sections V, VI, and VII, respectively.

II. RELATED WORK

The problem of a vehicle navigating from start to goal through a nonuniform wind field was first posed by Zermelo in 1931 [6]. Variations include cases of constant wind [7], spatially varying wind [8], [9], [10], spatiotemporally varying wind [11], [12], [13], and wind with obstacles [14], [15], [16], [17]. Autonomous robotic approaches include: case based reasoning [18], genetic algorithms [19], [20], sequen-

tial quadratic programming [21], Ant-colony optimization [22], and dynamic programming [23].

Our work borrows tools from graph methods such as A* [2], [8] and D*-Lite [4], as well as grid-based continuous-field methods such as Field-D* [3], [24] and their corresponding path extraction techniques [25].

It is closely related to level-set methods that consider wind [26], [27], [28], [29], [30], [17]. Differences include: our use of a cost-heuristic and heap to focus planning effort (vs. [27], [28], [29], [30], [17]), our use of nodes along a grid of wind samples (vs. [26], [29], [17]), our consideration of obstacles (vs. [27]), and our consideration of time-varying wind (vs. [27], [26], [30]) or unpredictable wind/obstacles (vs. [17]).

Our approach is an Any-Time algorithm, and therefore useful in real-time systems that have an ordering on solution quality, but where suboptimal solutions are preferable to none at all [31]. A major difference vs. previous Any-Time path-planning methods ([32], [1], [33]) is our use of a hypergraph that evolves from modeling (1+1)-dimensional (graph-like) connectivity to continuous $(D+1)$-dimensional (field-like) connectivity as time allows. A similarity is the α-inflation of the A*-like cost heuristic [1], [33], [34]. Our approach is the first Any-Time method to consider wind.

Work by [35] uses a similar cost function (soonest-time-at-goal) and a graph structure similar to our early-phase hypergraph (detailed in Section III). Differences include: the Any-Time properties of our algorithm, and our late-phase hypergraph that models a continuous cost-field. [36] considers spatiotemporally varying obstacles and wind using a combination of potential fields and optimization swarms. In contrast, our method is Any-Time and we plan/replan using a hypergraph embedded in a cost-field.

III. PRELIMINARIES

The state space $\mathcal{X}_{\text{full}}$ of the UAS is a Cartesian product of time T, location X, orientation θ, and derivatives of X and θ with respect to time. $\mathcal{X}_{\text{full}} = T \times X \times \dot{X} \times \ddot{X} \times \theta \times \dot{\theta} \times \ddot{\theta}$ In our hardware deployment we use $X \subset \mathbb{R}^3$ and $\theta \subset \text{SO}(3)$. In simulation we use $X \subset \mathbb{R}^2$ and $\theta \subset \text{SO}(2)$.

We assume our method is the global planning half of a hierarchical framework that also includes a local planner for trajectory optimization [5]. Our method considers $\mathcal{X}_{\text{global}} = \mathcal{X}_{\text{full}} \cap (T \times X)$, a low-dimensional representation of the world that assumes rotational effects and derivatives can be ignored for long-term planning. The local planner considers the full dimensionality restricted to an area near the robot, $\mathcal{X}_{\text{local}} \subset \mathcal{X}_{\text{full}}$ s.t. $0 < \mathscr{L}(\mathcal{X}_{\text{local}}) \ll \mathscr{L}(\mathcal{X}_{\text{full}})$, where $\mathscr{L}(\cdot)$ is the Lebesgue measure of '$\cdot$'.

$\mathcal{W}$ is the space of possible wind values. $\mathcal{W} \subset \mathbb{R}^3$ in our deployment and $\mathcal{W} \subset \mathbb{R}^2$ in simulation. We are provided with wind values/predictions at points on a regular rectangular lattice W embedded in space-time. Without loss of generality (because we can re-scale time and distance units), our presentation uses the simplifying convention that lattice points exist at unit intervals: $w : (\mathbb{N} \times \mathbb{N}^D) \cap \mathcal{X}_{\text{global}} \to \mathcal{W}$ where $\mathbb{N}$ is the set of the natural numbers $\{0, 1, \ldots\}$, and D is the dimensionality of X, i.e., $\mathcal{X}_{\text{global}} \subset (T \times X) \subset (\mathbb{R} \times \mathbb{R}^D)$.

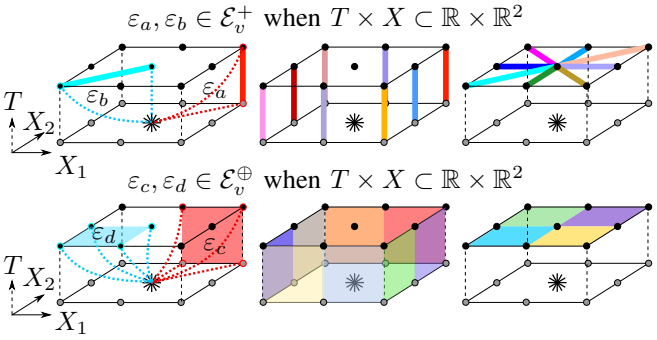

Fig. 2: Left: Hyperedges (color) $\varepsilon_a, \varepsilon_b \in \mathcal{E}_v^+$ (Top) and $\varepsilon_c, \varepsilon_d \in \mathcal{E}_v^\oplus$ (Bottom) when $D = 2$. Center/Right: Subfacets $f_{Nv} \in F_{Nv}$ (each subfacet is a different color). There is one hyperedge per each f_{Nv}.

We assume a user defined interpolation function $\hat{w}$ exists to estimate wind values at non-lattice points, $\hat{w} : \mathcal{X}_{\text{global}} \to \mathcal{W}$.

We track obstacles using a space-time occupancy grid O, where each grid is a closed hypercube *between* integer grid values (i.e., wind values are located at the corner points of obstacle grids). The function $o(x)$ returns the collision status of a point x, where $o : \mathcal{X}_{\text{global}} \to \{\textbf{true}, \textbf{false}\}$ and where **true** and **false** denote in-collision and collision-free.

We use a hypergraph $G = (V, \mathcal{E})$ to represent connectivity, where V is a set of nodes and $\mathcal{E}$ is a set of directed hyperedges. Nodes $v \in V$ are defined at the same locations as wind readings/predictions from W. For brevity, we allow the abuse of notation that nodes can be used notionally in place of their associated positions.

$$V = \{v \mid v \in (\mathbb{N} \times \mathbb{N}^D) \cap \mathcal{X}_{\text{global}}\}$$

In general, a hyperedge $(A, B) = \varepsilon$ defines connectivity from a *set* of nodes $A = \{v_a, \ldots, v_{a+i}\}$ to another *set* of nodes $B = \{v_b, \ldots, v_{b+j}\}$. That said, all of our hyperedges originate at a single node, $|A| = 1$ for all $(A, B) = \varepsilon \in \mathcal{E}$, and travel forward vs. time. Hyperedge $(\{v\}, B) = \varepsilon \in \mathcal{E}$ represents movement from v to *any* point on the continuous space-time patch bounded by the nodes in B (see Figure 2).

We use two mutually exclusive sets of hyperedges, denoted "graph-like" $\mathcal{E}^+$ and "field-like" $\mathcal{E}^\oplus$. Figures 2-3 show examples of these for $D = 2$ and $D = 3$. Graph-like hyperedges have $|B| = 2$ (the reason for 2 is discussed shortly) while field-like hyperedges have $|B| = 2^D$. When $D = 2$, projecting the sub-hypergraph over $\mathcal{E}^+$ from $X \times T$ to X gives a result that is *visually* similar to an 8-connected graph. Applying the same projection to sub-hypergraph over $\mathcal{E}^\oplus$ results in a connectivity structure *resembling* that of two-dimensional Field-D*.

$u, v \in V$ are neighbors if their lattice indices differ by at most 1 per space-time dimension. The neighbor set of v is:

$$N_v = \{u \mid \|u - v\|_\infty = 1\} \cap V$$

where $\|\cdot\|_\infty$ is the L-∞ norm. A node is not a neighbor of itself. Figures 4 and 5 depict N_v when $D = 2$ and $D = 3$.

v_T and v_X denote the projections of v into T and X, respectively. N_v^- and N_v^+ are the subsets of N_v located

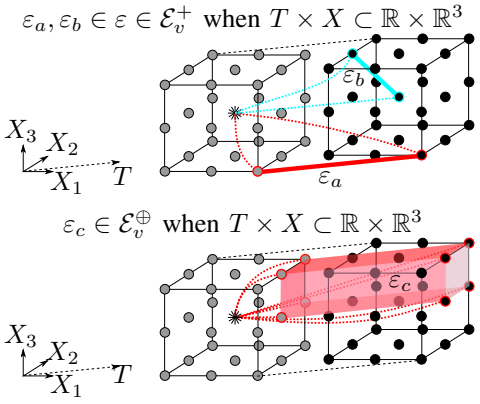

Fig. 3: Selected examples of hyperedges (color) $\varepsilon_a, \varepsilon_b \in \mathcal{E}_v^+$ (Top) and $\varepsilon_c \in \mathcal{E}_v^\oplus$ (Bottom) when $D = 3$.

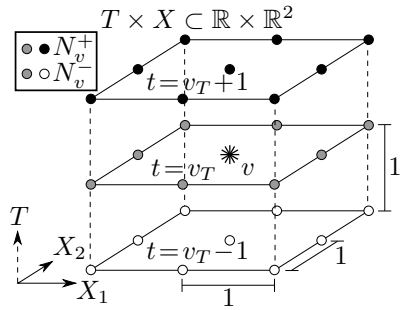

Fig. 4: Node v (star) in $\mathbb{R} \times \mathbb{R}^2$. Neighbors $u \in N_v$ are circles, color corresponds to time-slice. All neighbors are located on the surface of a unit hypercube $\mathcal{H}_v$ centered at v. The in- and out-neighbor sets (N_v^- and N_v^+) contain gray/white and gray/black neighbors, respectively.

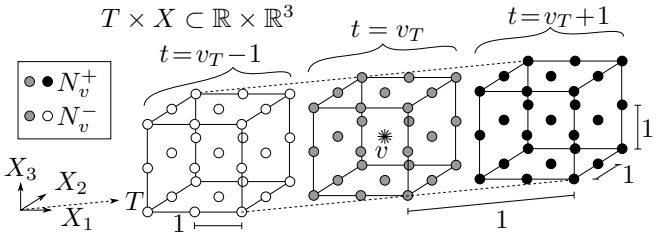

Fig. 5: Node v (star) in $\mathbb{R} \times \mathbb{R}^3$. All quantities are analogous to those in Figure 4, except that time-slices rise left to right.

before/after v (including the same time as v):

$$N_v^- = N_v \cap \{u \mid u_T \leq v_T\}$$
$$N_v^+ = N_v \cap \{u \mid u_T \geq v_T\}.$$

The convex hull over N_v is the surface of a unit radius space-time hypercube $\mathcal{H}_v$. Let $\mathcal{H}_v^+$ denote the closed half-surface of $\mathcal{H}_v$ located "after" v with respect to time.

$\mathcal{E}_v^+$ is the set of graph-like hyperedges $(\{v\}, B)$ originating at v. All $(\{v\}, B) \in \mathcal{E}_v^+$ have head-sets containing *two* nodes ($|B| = 2$) to accommodate movement through time.

$$\mathcal{E}_v^+ = \{(\{v\}, \{u, u'\}) \mid u_X = u'_X \wedge v_T = u_T = u'_T - 1\}$$
$$\cup \{(\{v\}, \{v', u'\}) \mid v_X = v'_X \wedge u'_T = v'_T = v_T + 1\}.$$

There are two distinct types, illustrated by ε_a (red) and ε_b (blue), respectively, in Figure 2-Top-Left for $D = 2$ (and in Figure 2-Top for $D = 3$). They correspond to movement constrained by the space or time bounds of $\mathcal{H}_v^+$, respectively.

A "sub-facet" f_{Nv} is the set of 2^D nodes that are mutually neighbors *and* co-planar along an axis-aligned D-dimensional hyperplane. The latter happens for neighboring nodes that are no greater than D away from each other with respect to the L_1 norm $\|\cdot\|_1$. $f_{Nv} = \{u_1, \ldots, u_{2^D}\}$ s.t. $\forall i, j = [1, 2^D], i \neq j, u_i \in N_{u_j} \wedge \|u_i - u_j\|_1 \leq D$. See Figure 2-Center/Right for examples of f_{Nv}.

$F_{Nv} = \bigcup_{N_v} \{f_{Nv}\}$ is the set of all D-dimensional sub-facets f_{Nv} with respect to v. The nodes in any $f_{Nv} \in F_{Nv}$ bound a patch of space-time that is exactly $1/(2^D)$-th of a D-dimensional facet of $\mathcal{H}_v$.

$\mathcal{E}_v^\oplus$ is the set of all field-like hyperedges that go from v forward through time to some f_{Nv}.

$$\mathcal{E}_v^\oplus = \{(\{v\}, \{u_1, \ldots, u_c\}) \mid \{u_1, \ldots, u_c\} \in F_{Nv} \wedge u_1, \ldots, u_c \in N_v^+\}$$

Again, there are two distinct cases depending on if movement is constrained by the space or time bounds of $\mathcal{H}_v$.

The set of hyperedges is therefore: $\mathcal{E} = \bigcup_{v \in V} \mathcal{E}_v^+ \cup \mathcal{E}_v^\oplus$. Any-Time planning starts by considering only $\bigcup_{v \in V} \mathcal{E}_v^+$ (graph-like planning), and then incorporating $\bigcup_{v \in V} \mathcal{E}_v^\oplus$ (field-like planning) when planning time permits.

We assume a subset of space-time is defined as the goal $\mathcal{X}_{\text{goal}}$. Without loss of generality, we assume the goal contains nodes and we define $V_{\text{goal}} = V \cap \mathcal{X}_{\text{goal}} \neq \emptyset$.

Our algorithm creates and maintains estimates of $d(v)$ the "soonest-time-at-goal" that can be achieved from each $v \in V$. Formally, $d : V \to \mathbb{R}$. Soonest-time-at-goal is similar to the more traditional "cost-to-goal" that is used by many planning methods (assuming the search tree is rooted at the goal) and is updated in a similar way. The main difference is that soonest-time-at-goal measures the estimated arrival time *at the goal* instead of the cost required to reach the goal.

Search progresses from V_{goal}, i.e., in the reverse direction of robotic movement. As in other field-based methods, we update $d(v)$ using (user provided) interpolation functions $\hat{d}$ and $\Delta\hat{t}$, where $\hat{d}(x)$ estimates $d(x)$ assuming $x \in \mathcal{H}_v^+$ and using interpolation over $\hat{d}(u)$ and $w(u)$ for $u \in N_v^+ \cup \{v\}$,

$$\hat{d} : \mathcal{X}_{\text{global}} \times \mathcal{W}^k \to \mathbb{R}$$

where $k = |N_v^+ \cup \{v\}|$, i.e., there is one wind value per node in $N_v^+ \cup \{v\}$. Similarly, $\Delta\hat{t}(v, x)$ estimates the time required to reach x from v given $w(u)$ for $u \in N_v^+ \cup \{v\}$.

$$\Delta\hat{t} : \mathcal{X}_{\text{global}} \times \mathcal{X}_{\text{global}} \times \mathcal{W}^k \to \mathbb{R}$$

Obstacles imply infinite time, $o(v) \vee o(x) \Rightarrow \Delta\hat{t}(v, x) = \infty$.

We use an approximation method similar to Field-D* in that we only consider local movement possibilities that travel through some point $x \in \mathcal{H}_v^+$.

$$d(v) = \min_{x \in \mathcal{H}_v^+} \left(\Delta\hat{t}(v, x) + \hat{d}(x)\right) \quad (1)$$

The point $x^* \in \mathcal{H}_v^+$ that yields the best local connection is called the "target point" of v.

$$x^* = \arg\min_{x \in \mathcal{H}_v^+} \left(\Delta\hat{t}(v, x) + \hat{d}(x)\right) \quad (2)$$

The particular implementations of $\hat{d}$ and $\Delta\hat{t}$ that we use for experiments are discussed in Section IV.

Assuming the UAS is located at x_{robot} and given G, our method extracts a path $P : [0, 1] \to \mathcal{X}_{\text{global}}$ such that $P(0) = x_{\text{robot}}$ and $P(1) = x_{goal} \in \mathcal{X}_{\text{goal}}$. Alternatively, it reports failure if no such path can be found. As in other field-based methods the path is returned as a sequence of points, $P = \{x_1, \ldots x_\ell\}$ where $x_1 = x_{\text{robot}}$ and $x_\ell = x_{goal}$ and $\ell = |P|$, extracted using an iterated function:

$$x_{i+1} = \underset{x \in \mathcal{H}_x^+}{\arg\min} \left(\Delta\hat{t}(x_i, x) + \hat{d}(x) \right)$$

where, with some abuse of notation, $\mathcal{H}_x^+ = \bigcup_{v \in N_x^+} \mathcal{H}_v^+$ and $N_x^+ = \{v \mid \|v - x\|_\infty \leq 1 \ \wedge \ v_T > x_T\}$.

We assume the local trajectory optimizer takes $\hat{P} \subset P \subset \mathcal{X}_{\text{global}}$ as input and then produces a trajectory $\zeta \subset \mathcal{X}_{\text{local}}$. If $\hat{P}$ is a strict subset of P located at the beginning of P (e.g., for reasons of tractability, and in our experiments), then $\hat{P}$ can be defined based on distance and/or time, e.g., $\hat{P} = \{x_1, \ldots x_k\}$ where either $x_k = \arg\max_{x \in P} x_T - x_{1T} \leq c_{\text{time}}$ or $x_k = \arg\max_k \sum_{i=2}^{k} \|x_i - x_{i-1}\|_X \leq c_{\text{dist}}$ where $\| \cdot \|_X$ is a distance metric based only on location and c_{time} and c_{dist} are user defined parameters. Projecting ζ from $\mathcal{X}_{\text{local}}$ to $\mathcal{X}_{\text{global}}$ will most likely not reproduce $\hat{P}$ given that the trajectory optimizer's purpose is to improve the path vs. the constraints of the full system. However, we assume the endpoints remain unchanged.

IV. ALGORITHM

Our presentation assumes separate modules exist for wind prediction, global planning, local planning, and auto-pilot control, and that these communicate over a network. Algorithm 1 contains (as an example) the system-wide start-up procedure used in our deployment.

A. Global Planning (Our Method)

The global planner is an Any-Time algorithm that quickly finds and returns a solution that reaches the goal at most $\alpha\beta$ times slower than the optimal solution, and then proceeds to recalculate solutions for progressively better $\alpha\beta$ over successive Any-Time planning epochs. α is an Any-Time inflation factor that is reduced as a function of epoch number κ, and β depends on environmental modeling error.

Node v tracks $closed(v)$, the last epoch v was inserted into the (A*-like) closed list. Similar to D*-Lite we estimate $d(v)$ using an incrementally refined running look-ahead value $lmc(v)$ that tracks the minimum soonest-time-at-goal achieved via any of v's hyperedges (i.e. reflecting paths via neighbors that have been examined vs. the current $\alpha\beta$ value). $d(v) \leftarrow lmc(v)$ whenever v is inserted into the closed list.

A heap H is used to determine the order that nodes are added to the search graph and uses the standard `Top`, `Pop`, `Insert`, and `Update` functions. $h(v)$ is an admissible heuristic estimate of the shortest time duration in which the UAS can reach v from its current location. Like ARA* [1] we use the Any-Time idea of sorting nodes in H based on

$$\texttt{Key}(v) = \alpha h(v) + \min(d(v), lmc(v))$$

Path improvement happens at two time scales. First, we perform a complete pass over a sequence of predefined α values $(\alpha_1, \alpha_2, \ldots, 1)$ using *only* the subgraph of G involving $\mathcal{E}_v^+$ for all $v \in V$, where $\alpha_i > \alpha_j \geq 1$ for $i < j$. This is relatively fast due to a small branching factor and simple interpolations, but has large β as a result (e.g., in the simple case of no wind $\beta = \sqrt{5}/(1 + \sqrt{2}) \approx 1.08$). Next, we perform another pass over the same α values using the full G involving $\mathcal{E}_v^+ \cup \mathcal{E}_v^{\oplus}$. The second pass is more difficult due to the higher branching factor and higher-dimensional interpolation functions necessary to yield smaller β (e.g., $\beta \approx 1.003$ in no wind [25]). Planning effort from previous epochs is leveraged as in ARA*. We use the flag τ to track whether or not we are planning though the sparse or full hypergraph.

`Decrease`(κ) decreases κ and then updates τ and α values based on κ. `ExtractPath`(x_{robot}) returns the path P between the UAS's current location x_{robot} and the goal.

The global planning loop appears in Algorithm 2. There is an outer loop (lines 1-7) for replanning in the event of new wind ($\Delta W \neq 0$) or obstacles ($\Delta O \neq 0$), and an inner loop (lines 3-7) for Any-Time planning epochs of decreasing α and β. The graph is reinitialized each time the wind or obstacles changes (line 2). During each planning epoch the soonest-time-at-goal field is updated as much as possible given a particular combination of α and τ (line 4); the iteration counter κ is increased and α and τ recalculated accordingly (lines 5-6); and the heap H is re-balanced to reflect the new values of α and τ (line 7).

Graph initialization appears in Algorithm 3. κ is set to 1 and α and τ accordingly using (line 2). Non-goal nodes are initialized to have infinite soonest-time-at-goal values, empty parent pointers, and to be in the open list (lines 4-7). Goal nodes are initialized to have soonest-time-at-goal as their own time-of-day and neighbors updated (line 9-11).

The `ImproveField` subroutine appears in algorithm 4. While the heap contains nodes able to reduce $d(x_{\text{robot}})$, we pop the top node v (line 2), recalculate its look-ahead value (line 4), mark it in the close list for iteration κ (line 5), and update its neighbors in case they can benefit by using a hyperedge containing v as their parent (line 6). In practice, increasing `Key`(x_{robot}) by $c/(s_{\text{robot}} + \max_{u \in V} \|w(u)\|_2)$ causes all nodes within radius c of x_{robot} to (also) have their soonest-time-at-goal values updated by `ImproveField`, which increases robustness vs. drift, etc.

`UpdateNeighbors` appears in Algorithm 5. It loops over all nodes u that could conceivably use v as a parent ($u \in N_v^-$), checking if their look-ahead $lmc(u)$ values can be improved by using v as their parent (lines 6-12). Nodes u in the goal set are ignored (line 2). The hypergraph structure depends on τ (line 3). All relevant hyperedges ϵ from u that involve v are checked (line 4). Nodes that have been prematurely added to the closed list (due to the α inflation factor) are added to V_{incon} the inconsistent list, line 12.

`RecalculateLMC`, Algorithm 6, is similar to the inner loop of `UpdateNeighbors` except that it operates on a single node v and does not require heap interaction. After

Algorithm 1: System Start

```
1 start wind sensor (computer 1);
2 start wind prediction engine (computer 2);
3 start global planner (computer 3, thread 1);
4 start global path server (computer 3, thread 2);
5 start local planner (computer 4);
6 start controller/auto-pilot on (on robot);
```

Algorithm 2: Global Planning Loop

```
1 while x_robot ∉ X_goal do
2     InitGraph();
3     while ΔW = 0 and ΔO = 0 do
4         ImproveField(κ, α, τ);
5         κ ← κ + 1;
6         (α, τ) ← Decrease(κ, α, τ);
7         RebalanceHeap(H, α, τ);
```

Algorithm 3: InitGraph()

```
1  κ ← 1;
2  (α, τ) ← Decrease(κ, α, τ);
3  for all v ∈ V \ V_goal do
4      lmc(v) ← ∞;
5      d(v) ← ∞;
6      parent(v) ← ∅;
7      closed(v) ← 0;
8  for all v ∈ V_goal do
9      lmc(v) ← v_T;
10     d(v) ← v_T;
11     UpdateNeighbors(v);
```

Algorithm 4: ImproveField(κ, α, τ)

```
1 while Top(H) ≠ ∅ and (Key(Top(H)) <
  Key(x_robot) or d(x_robot) = ∞) do
2     v ← Pop(H);
3     RecalculateLMC(v);
4     d(v) ← lmc(v);
5     closed(v) = κ;
6     UpdateNeighbors(v);
```

Algorithm 5: UpdateNeighbors(v)

```
1  for all u ∈ N_v^- do
2      if u ∈ V_goal then continue ;
3      if τ then E_u ← E_v^+ else E_u ← E_u^+ ∪ E_u^⊕ ;
4      for all ε ∈ E_u s.t. v ∈ ε do
5          d' ← EstViaHyperEdge(u, ε);
6          if d' < lmc(u) then
7              lmc(u) ← d';
8              parent(u) ← ε;
9              if closed(u) < κ then
10                 Update(H, u);
11             else
12                 V_incon ← V_incon ∪ {u};
```

Algorithm 6: RecalculateLMC(v)

```
1 if τ then E_v ← E_v^+ else E_v ← E_v^+ ∪ E_v^⊕ ;
2 for all ε ∈ E_v do
3     d' ← EstViaHyperEdge(v, ε);
4     if d' < lmc(v) then
5         lmc(v) ← d';
6         parent(v) ← ε;
```

Algorithm 7: RebalanceHeap(H, α)

```
1  P ← ExtractPath(x_robot) ;
2  for all v ∈ P \ V_goal do
3      for all u ∈ N_v ∪ {v} do
4          d(u) ← ∞ ;
5          RecalculateLMC(u) ;
6          V_incon ← V_incon ∪ {u} ;
7  V_incon ← V_incon ∪ H ;
8  H ← ∅ ;
9  for all v ∈ V_incon do
10     V_incon ← V_incon \ {v} ;
11     RecalculateKey(v, α);
12     Insert(H, v);
```

Algorithm 8: Global Path Server

```
1 if PathRequested() then
2     P̂ ⊂ P ← ExtractPath(x_robot);
3     SendPath(P̂);
4     x_robot ← P̂(end);
```

Algorithm 9: Local Trajectory Loop

```
1 P̂ ← requestPath(x_robot);
2 ζ ← CreateTrajectory(P̂);
3 while x'_robot ∉ X_goal do
4     x'_robot = P̂(end);
5     SendToController(ζ);
6     P̂ ← requestPath(x'_robot);
7     ζ ← CreateTrajectory(P̂);
8     while ‖x_robot − x'_robot‖ ≥ c_r do
9         Sleep(c_sleep);
```

determining which hypergraph structure is being used (line 1) it checks all edges from v to see if they yield better look-ahead values, updating $lmc(v)$ and the parent hyperedge of v appropriately (lines 2-6).

`RebalanceHeap` in Algorithm 7 first forces all nodes in the current best-path to be inconsistent (lines 1-6), and then adds these plus any nodes remaining in the heap H to the inconsistent list V_{incon}, line 7 (emptying the heap, line 8). Next, all nodes in V_{incon} are inserted back into H using new key values based on (the new) α, lines 9-12.

B. Interaction With Local Planner

We assume the global path server, Algorithm 8, is a callback function responsible for handling requests from the local planner. When a request is received, it extracts the appropriate $\hat{P}$ from x_{robot} and returns it to the local planner.

The local planning loop used in our deployment is depicted in Algorithm 9 and included here as an example. It requests a global path $\hat{P}$ (lines 1, 6), and then creates a trajectory ζ in the full dimensional space of the system based on $\hat{P}$ (lines 2, 7) using a user provided objective function to define the method `CreateTrajectory`($\hat{P}$). ζ is sent to the autopilot for control (line 5), and then the local planner calculates the next trajectory starting from the end of the current one (lines 4-7). It calculates at most one trajectory ahead of the controller (lines 8-9), tuned via the user defined constants c_{r} and c_{sleep}. Alternative criteria can also be used, such as replanning if the wind field changes or obstacles move.

The `CreateTrajectory`($\hat{P}$) used in our deployment uses a three-dimensional kinematic point-mass model to solve a nonlinear program online and generate energy-optimal trajectories through $\mathcal{X}_{\text{local}}$. $\hat{P}$ is discretized and optimized using an objective function while constrained by the robot's kinematics. Mission-specific objective functions typically include a term that minimizes the energy expenditure of the robot's on-board power source as well as any secondary objectives, such as the minimization of time on trajectory. The local planner has the ability to automatically exchange objective functions and associated constraints to accommodate the mission's state during execution.

C. Calculating Soonest-Time-At-Goal $d(v)$

Mathematically, the estimates of soonest-time-at-goal values $d(v)$ are calculated using Equations 1 and Equations 2. Algorithmically this happens indirectly by calling $lmc(v) \leftarrow$ `EstViaHyperEdge`(v, ε) on Algorithm lines 5.5 and 6.3, and then setting $d(v) \leftarrow lmc(v)$ on line 4.4. Note $\hat{d}(v) = \min(d(v), lmc(v))$ given $d(v)$ and $lmc(v)$.

A particular call to `EstViaHyperEdge`(v, ε) deals with the possibility that x^* is located in the subset of $\mathcal{H}_v^+$ defined by B, where $\varepsilon = (\{v\}, B)$. Thus, we solve Equations 1 and 2 piecemeal for each $\varepsilon \in \mathcal{E}_v$ using Equations 3 and 4:

$$d(v) = \min_{x \in \varepsilon \cap \mathcal{H}_v^+} \left(\Delta\hat{t}(v, x) + \hat{d}(x) \right) \tag{3}$$

$$x^* = \arg\min_{x \in \varepsilon \cap \mathcal{H}_v^+} \left(\Delta\hat{t}(v, x) + \hat{d}(x) \right). \tag{4}$$

The geometric set described by $\varepsilon \cap \mathcal{H}_v^+$ depends on the type of hyperedge of ε:

- If $\varepsilon \in \mathcal{E}_v^+$ then $\varepsilon \cap \mathcal{H}_v^+ \equiv [u_1, u_2]$ is the line-segment in space-time between u_1 and u_2 where $\varepsilon = (\{v\}, \{u_1, u_2\})$.

- If $\varepsilon \in \mathcal{E}_v^{\oplus}$ then $\varepsilon \cap \mathcal{H}_v^+ \equiv f_{Nv}$ is the patch of space-time located between 2^D neighbors of v (taking the form of a D-dimensional hypercube) where $\varepsilon = (\{v\}, f_{Nv})$.

D. Implementation Details

Equations 3 and 4 can be implemented in a variety of ways; we now motivate the one used in our deployment and simulations. We assume we are allowed to pick the *spatial* trajectory $\xi \subset X$ that the UAS flies but that we are at the mercy of the wind with respect to how much time this requires along T. Given a particular ξ from v to x, the time required to move from some point v to some other point x is $\Delta t = x_T - v_T$, where $v_X, x_X \in X$ and $v_T \in T$ are fixed and $x_T \in T$ depends on ξ and $\hat{w}$.

$$\Delta\hat{t} = \int_v^x \frac{\partial t}{\partial \xi} \partial \xi.$$

The soonest-time-at-goal at an arbitrary point x on the space-time patch or line segment described by $\varepsilon \cap \mathcal{H}_v^+$ is:

$$\hat{d}(x) = f(G, x)$$

where G represents graph structure, wind, and soonest-time-at-goal values. If we can find a differentiable expression for $\Delta\hat{t} + f(G, x)$, then we can check a small finite number (depending on D) of candidate x^* and then select the best one. The first possible location for x^* is where the derivative with respect to X is 0 (if that location is within $\varepsilon \cap \mathcal{H}_v^+$). Another set of possible x^* exist at the locations on B_i the lower-dimensional boundaries of $\varepsilon \cap \mathcal{H}_v^+$ (also found by solving for zero derivative along B_i), and the final set exists at the corners of $\varepsilon \cap \mathcal{H}_v^+$. Formally,

$$x^* = \underset{x \in \left(\{x_a^*\} \cup \{x_b^*\} \cup \{x_c^*\}\right) \cap \left(\varepsilon \cap \mathcal{H}_v^+\right)}{\arg\min} \left[\Delta\hat{t} + f(G, x)\right] \quad (5)$$

where $\{x_a^*\}$ and $\{x_b^*\}$ and $\{x_c^*\}$ are given by

$$0 = \frac{d}{dx}\left[\Delta\hat{t} + f(G, x_a^*)\right] \quad (6)$$

$$0 = \frac{d}{dx}\left[\Delta\hat{t} + f(G, x_b^*)\right]\Big|_{B_i} \text{ for all } B_i \in \varepsilon \cap \mathcal{H}_v^+ \quad (7)$$

$$\{x_c^*\} = \varepsilon \cap \mathcal{H}_v^+ \cap \mathbb{N}^{D+1} \quad (8)$$

respectively, and describe the different sets of candidate x^*.

That said, we are unable to find a differentiable closed form solution to Equations 5-7. It is possible to perform these calculation numerically, but this is too slow for real-time applications. Instead, we approximate $\Delta\hat{t} + f(G, x)$ locally using a polynomial function. As resolution increases, our approximation approaches the correct solution.

We make a the following simplifying assumptions: $\xi \equiv [v_X, x_X] \subset X$ is the straight line segment from v to x. The UAS is fast enough vs. time (or time granularity small enough) such that wind *within* a single grid wind can be assumed locally constant vs. T and linearly varying vs. X (and obviously defined by W vs. $T \times X$ between different grids). *These assumptions are required only for the way we approximate Equations 3-7 in our implementation, and are not required by our method, in general.*

We use the following 3-part estimation procedure:
(1) Estimate $\Delta\hat{t} + f(G, x')$ at points x' along a local fixed grid $\mathcal{G}$ (at double the resolution of the wind field), where:

- D-dimensional linear interpolation is used to fill in any missing values of $f(G, x')$ for each $x' \in \mathcal{G}$.
- D-dimensional linear interpolation is used to approximate $\Delta\hat{t}$ for each $x' \in \mathcal{G}$ as follows:

$$\Delta\hat{t} = \int_v^{x'} \frac{\partial t}{\partial \xi} \partial \xi \approx \int_v^{x'} \sum_{u \in \varepsilon \setminus \{v\}} \rho(x, u) \frac{\partial t}{\partial \xi}(v, x', u) \partial \xi$$

  where $\rho(x, u)$ is the linear interpolation weight vs. u experienced at $x \in \xi$ and $\frac{\partial t}{\partial \xi}(v, x', u)$ is the $\frac{\partial t}{\partial \xi}$ that would be experienced by a UAS flying the heading from v to x' while experiencing the wind $w(u)$.

(2) Locally fit either a polynomial surface (for cases involving Equation 6) or a polynomial (for cases involving Equation 7) based on the resulting values from (1).
(3) Solve Equations 6 or 7 using the polynomial surface or polynomial from (2), respectively.

This has the advantage of producing closed form approximations to Equations 5-7, respectively, and is very fast in practice. This approximation requires the implicit assumption the errors resulting from using a best-fit polynomial surface (i.e., instead of numerical root-finding and integration methods) and D-dimensional linear interpolations (used for calculating the $\Delta\hat{t} + f(G, x')$ and $\Delta\hat{t}$ values for $x' \in \mathcal{G}$ from which the surface is created) are tolerable. We find this to be reasonable as long as wind values do not change greatly between neighboring lattice points. Note that all interpolations approach the true values, in the limit, as resolution increases to infinity.

In the event that UAS speed is insufficient to overcome the effects of oncoming wind (e.g., a head-wind) along ξ, then we define $t = \infty$ for that hyperedge.

V. EXPERIMENTS

A. Real-World Deployment

Our method was deployed near Lubbock Texas as the global planner on a multi-institution deployment involving Doppler radar, atmospheric models for online real-time wind sensing and prediction, and a fixed wing UAS. This real-world experience demonstrates that our method works in practice as part of a larger system, and in the field with a state space $\mathcal{X}_{\text{global}} \subset X \times T = \mathbb{R}^3 \times \mathbb{R}$.

B. Comparison To Other Methods On Synthetic Wind Data

We now compare our method to A*, ARA*, and Field-D* in challenging scenarios using synthetic data in $\mathcal{X}_{\text{global}} \subset X \times T = \mathbb{R}^2 \times \mathbb{R}$. The UAS has an airspeed of 20 m/s, space-time lattice grids are 150 meters long and wide by 100 seconds in duration. The lattice is 100 by 110 by 5 grids (x, y, time). Wind and obstacle values before/after G are set to the values located at the first/last time-slice of G, respectively. A* and ARA* use $\bigcup_{v \in V} \mathcal{E}_v^+$, while Field-D* uses the full $\bigcup_{v \in V} \mathcal{E}_v^+ \cup \mathcal{E}_v^{\oplus}$. The calculation of time vs. motion along each hyperedge is identical for all methods. ARA* and our method use α values $(10.5, 10.0, 9.5, \ldots, 1.0)$.

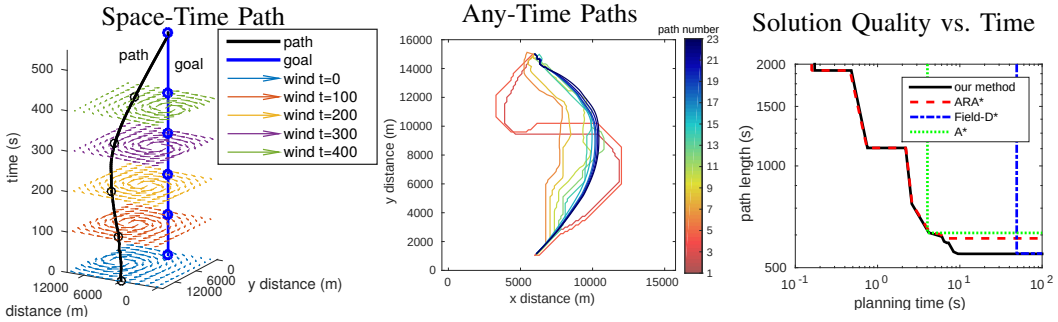

Fig. 6: Experiment 1, a vortex moves across the robot's path. Left: the space-time path of the robot. Center: the sequence of Any-Time paths found by our method. Right: comparison of our method vs. other state-of-the art methods.

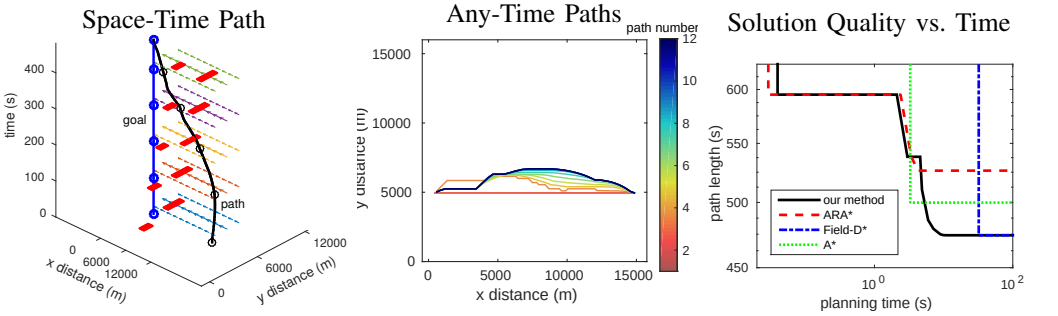

Fig. 7: Experiment 2, the UAS can use the air jet to achieve it's goal more quickly (jet located in the positive y direction from the robot), but it must avoid the obstacles. Left: space-time path. Center: Any-Time paths found by our method. Right: comparison vs. other methods.

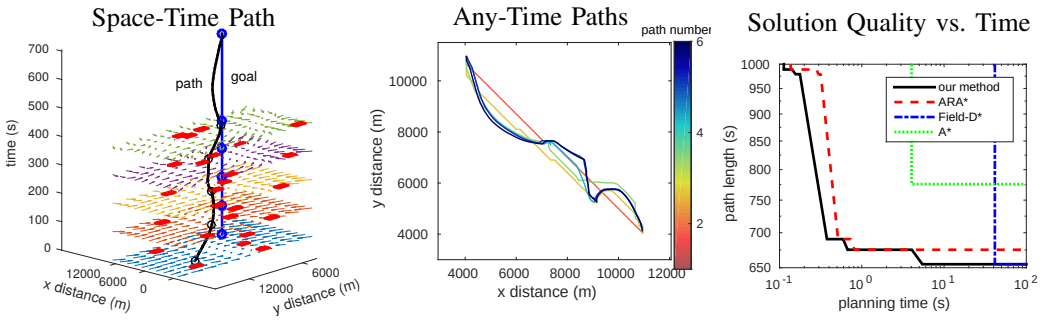

Fig. 8: Experiment 3, fractally generated wind data with 5 moving obstacles. Left: space-time path. Center: Any-Time paths found by our method. Right: comparison vs. other methods.

The first experiment requires the UAS to fly around a moving vortex. The UAS starts at position $[6000, 15000]$ at time 0 and the goal is located at $[6000, 1000]$. The center of the vortex starts at $[8000, 0]$ and moves at velocity $[-20, 0]$ (across the robot's path). The vortex rotates clockwise, rotation speed is 30 m/s at its center, decreasing linearly to 0 m/s at distance 15000 m. There are no obstacles in this experiment. Figure 6 shows the space-time path of the robot, the sequence of Any-Time paths found by our method, and a comparison of solution lengths vs. time for our method and the comparison methods.

In the second experiment the UAS moves from $[15500, 5000]$ to $[500, 5000]$. There is 6000 m wide jet of air centered at $y = 7500$ m that flows in the robot's direction of travel. Wind speed increase linearly from 0 at the jet boundary to 28 m/s at its center. An moving obstacle 1000.0 m by 2000.0 m starts at $[1000.0, 7500.0]$ and travels along the center of the jet, in the opposite direction of the robot, at 20 m/s. A second obstacle, 1000.0 m by 1000.0 m, starts at $[3000.0, 3000.0]$ and moves 10 m/s in the positive y direction (i.e., across the robot's path); it is timed to hinder the final movement of the robot. Results appear in Figure 7.

The third experiment uses fractally generated wind data with x and y components of wind varying between -30 and 30 m/s. The UAS moves from $[4000.0, 11000.0]$ to $[11000.0, 4000.0]$ while avoiding 5 moving obstacles of various size and speed. Results appear in Figure 8.

VI. DISCUSSION

The Any-Time properties of our method are evident in Figures 6-8-Center/Right. Initial paths are found very quickly, while better paths are found vs. planning time.

Modifying hypergraph connectivity vs. planning epoch (i.e., switching from graph-like planning to field-like planning) works well for Any-Time planning. Our method was able to find paths comparable to Field-D* but in a fraction of the time. The initial phase of graph-like planning yields results similar to ARA*—as expected, given that the two algorithms construct the same search graph over $\bigcup_{v \in V} \mathcal{E}_v^+$)—but eventually finds better solutions than ARA* after $\bigcup_{v \in V} \mathcal{E}_v^+ \cup \mathcal{E}_v^{\oplus}$ is considered, i.e., due to lower β.

The simultaneous consideration of wind and obstacles is well illustrated by the Jet experiment (Figure 7-Center). Any-Time solutions shift upward to take advantage of the fast-moving center of the jet, until they move to avoid the oncoming obstacles.

VII. SUMMARY AND CONCLUSIONS

We present a global path-planning algorithm that enables an autonomous UAS to fly through a spatiotemporally varying wind-field with moving obstacles. Our method is an Any-Time algorithm that returns a sequence of $\alpha\beta$-optimal solutions, where α is a user defined inflation parameter that decreases vs. time and β is a modeling error factor that decreases as hypergraph connectivity shifts from graph-like to field-like.

Our method has been deployed in the field as the global planning component of UAS involving Doppler radar, atmospheric prediction models, and a fixed wing UAS. Experiments show that it performs well vs. other current state of the art methods. It is the first algorithm that modifies hypergraph structure to help archive Any-Time performance. It is also the first Any-Time method that simultaneously considers spatiotemporally varying wind and obstacles.

REFERENCES

[1] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems*, 2003.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.

[3] D. Ferguson and A. Stentz, "Field d*: An interpolation-based path planner and replanner," in *Robotics Research*. Springer, 2007, pp. 239–253.

[4] S. Koenig and M. Likhachev, "Improved fast replanning for robot navigation in unknown terrain," in *Robotics and Automation, IEEE International Conference on*, vol. 1, 2002, pp. 968–975.

[5] E. W. F. Will Silva and W. Shaw-Cortez, "Implementing path planning and guidance layers for dynamic soaring and persistence missions," in *International Conference on Unmanned Aircraft Systems*, Denver, CO, June 2015.

[6] E. Zermelo, "Über das navigationsproblem bei ruhender oder veränderlicher windverteilung," *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 11, no. 2, pp. 114–124, 1931.

[7] T. G. McGee, S. Spry, and J. K. Hedrick, "Optimal path planning in a constant wind with a bounded turning rate," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*. American Institute of Aeronautics and Astronautics, 2005.

[8] B. Garau, A. Alvarez, and G. Oliver, "Path planning of autonomous underwater vehicles in current fields with complex spatial variability: an A* approach," in *Robotics and Automation. IEEE International Conference on*, April 2005, pp. 194–198.

[9] E. Bakolas and P. Tsiotras, "Time-optimal synthesis for the Zermelo-Markov-Dubins problem: The constant wind case," in *American Control Conference (ACC), 2010*, June 2010, pp. 6163–6168.

[10] N. Lawrance and S. Sukkarieh, "Path planning for autonomous soaring flight in dynamic wind fields," in *Robotics and Automation, IEEE International Conference on*, May 2011, pp. 2499–2505.

[11] W. H. Al-Sabban, L. F. Gonzalez, R. N. Smith, and G. F. Wyeth, "Wind-energy based path planning for electric unmanned aerial vehicles using markov decision processes," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[12] R. P. Anderson, E. Bakolas, D. Milutinović, and P. Tsiotras, "Optimal feedback guidance of a small aerial vehicle in a stochastic wind," *Journal of Guidance, Control, and Dynamics*, vol. 36, no. 4, pp. 975–985, 2013.

[13] A. Chakrabarty and J. Langelaan, "UAV fight path planning in time varying complex wind-fields," in *American Control Conference*, June 2013, pp. 2568–2574.

[14] M. Soulignac and P. Taillibert, "Fast trajectory planning for multiple site surveillance through moving obstacles and wind," in *Proceedings of the Workshop of the UK Planning and Scheduling Special Interest Group*, 2006, pp. 25–33.

[15] D. Kruger, R. Stolkin, A. Blum, and J. Briganti, "Optimal AUV path planning for extended missions in complex, fast-flowing estuarine environments," in *Robotics and Automation, 2007 IEEE International Conference on*, 2007, pp. 4265–4270.

[16] M. Seleck, P. Váňa, M. Rollo, and T. Meiser, "Wind corrections in flight path planning," *International Journal of Advanced Robotic Systems*, vol. 10, no. 248, 2013.

[17] T. Lolla, P. J. Haley Jr., and P. F. J. Lermusiaux, "Path planning in multi-scale ocean flows: Coordination and dynamic obstacles," *Ocean Modelling*, vol. 94, pp. 46–66, 2015.

[18] C. Vasudevan and K. Ganesan, "Case-based path planning for autonomous underwater vehicles," *Autonomous Robots*, vol. 3, no. 2-3, pp. 79–89, 1996.

[19] K. Sugihara and J. Yuh, "GA-based motion planning for underwater robotic vehicles," in *International Symposium on Unmanned Untethered Submersible Technology*, 1997, pp. 406–415.

[20] A. Alvarez, A. Caiti, and R. Onken, "Evolutionary path planning for autonomous underwater vehicles in a variable ocean," *Oceanic Engineering, IEEE Journal of*, vol. 29, no. 2, pp. 418–429, April 2004.

[21] Y. Petillot, I. Ruiz, D. Lane, Y. Wang, E. Trucco, and N. Pican, "Underwater vehicle path planning using a multi-beam forward looking sonar," in *OCEANS Conference Proceedings*, vol. 2, Sep 1998, pp. 1194–1199 vol.2.

[22] A. L. Jennings, R. Ordonez, and N. Ceccarelli, "An ant colony optimization using training data applied to UAV way point path planning in wind," in *Swarm Intelligence Symposium, 2008. SIS 2008. IEEE*, 2008, pp. 1–8.

[23] A. Jennings, R. Ordonez, and N. Ceccarelli, "Dynamic programming applied to UAV way point path planning in wind," in *Computer-Aided Control Systems. IEEE International Conference on*, Sept 2008, pp. 215–220.

[24] M. Soulignac, P. Taillibert, and M. Rueher, "Adapting the wavefront expansion in presence of strong currents," in *Robotics and Automation. IEEE International Conference on*, 2008, pp. 1352–1358.

[25] M. Otte and G. Grudic, "Extracting paths from fields built with linear interpolation," in *Intelligent Robots and Systems, IEEE/RSJ International Conference on*, Oct 2009, pp. 4406–4413.

[26] C. Petres, Y. Pailhas, P. Patron, Y. Petillot, J. Evans, and D. Lane, "Path planning for autonomous underwater vehicles," *Robotics, IEEE Transactions on*, vol. 23, no. 2, pp. 331–341, 2007.

[27] Z. Liang, X. Ma, and X. Dai, "A novel path planning algorithm based on twofold interpolations," in *Mechatronics and Automation. IEEE International Conference on*, 2008, pp. 718–723.

[28] J. Elston and E. W. Frew, "Unmanned aircraft guidance for penetration of pretornadic storms," *Journal of guidance, control, and dynamics*, vol. 33, no. 1, pp. 99–107, 2010.

[29] T. Lolla, P. F. J. Lermusiaux, M. P. Ueckermann, and P. J. Haley Jr., "Time-optimal path planning in dynamic flows using level set equations: Theory and schemes," *Oceanic Dynamics*, vol. 64, no. 10, pp. 1373–1397, 2014.

[30] T. Lee, H. Kim, H. Chung, Y. Bang, and H. Myung, "Energy efficient path planning for a marine surface vehicle considering heading angle," *Ocean Engineering*, vol. 107, pp. 118–131, 2015.

[31] S. Zilberstein and S. J. Russell, "Approximate reasoning using anytime algorithms," in *Imprecise and Approximate Computation*, S. Natarajan, Ed. Kluwer Academic Publishers, 1995. [Online]. Available: http://rbr.cs.umass.edu/shlomo/papers/ZRchapter95.html

[32] R. Zhou and E. A. Hansen, "Multiple sequence alignment using anytime a*," in *Eighteenth National Conference on Artificial Intelligence*, 2002, pp. 975–976.

[33] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic A*: An anytime, replanning algorithm," in *International Conference on Automated Planning and Scheduling*, 2005.

[34] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1984.

[35] D. R. Thompson, S. Chien, Y. Chao, P. Li, B. Cahill, J. Levin, O. Schofield, A. Balasuriya, S. Petillo, M. Arrott, *et al.*, "Spatiotemporal path planning in strong, dynamic, uncertain currents," in *Robotics and Automation, IEEE International Conference on*, 2010, pp. 4778–4783.

[36] J. Witt and M. Dunbabin, "Go with the flow: Optimal AUV path planning in coastal environments," in *Australian Conference on Robotics and Automation*, no. 2, 2008.