Computing Energy Optimal Paths in Time-Varying Flows

Dhanushka Kularatne[1], Subhrajit Bhattacharya[2] and M. Ani Hsieh[1]

***Abstract*— Autonomous marine vehicles (AMVs) are typically deployed for long periods of time in the ocean to monitor different physical, chemical, and biological processes. Given their limited energy budgets, it makes sense to consider motion plans that leverage the dynamics of the surrounding flow field so as to minimize energy usage for these vehicles. In this paper, we present two graph search based methods to compute energy optimal paths for AMVs in two-dimensional (2-D) time-varying flows. The novelty of the proposed algorithms lies in a unique discrete graph representation of the 3-D configuration space spanned by the spatio-temporal coordinates. This enables a more efficient traversal through the search space, as opposed to a full search of the spatio-temporal configuration space. Furthermore, the proposed strategy results in solutions that are closer to the global optimal when compared to greedy searches through the spatial coordinates alone. We demonstrate the proposed algorithms by computing optimal energy paths around the Channel Islands in the Santa Barbara bay using time-varying flow field forecasts generated by the Regional Ocean Model System. We verify the accuracy of the computed paths by comparing them with paths computed via an optimal control formulation.**

I. INTRODUCTION

Scientific activities in aquatic environments that were traditionally performed manually, are increasingly being automated using autonomous marine vehicles (AMVs). These vehicles could either be surface vehicles or underwater vehicles. Examples include characterizing the dynamics of plankton assemblages [3], measurement of temperature profiles [5], and monitoring of harmful algae blooms [15]. In these and similar environmental monitoring applications in the ocean, AMVs are often deployed over long periods while operating with limited energy budgets. As such, researchers have to design motion strategies that are energy efficient to make maximum use of the capabilities of these autonomous platforms.

While the high inertia environment of the ocean couples the environmental dynamics to the marine vehicle dynamics, it presents a unique opportunity for vehicles to exploit the surrounding flows for more efficient navigation. As such, there is a substantial amount of recent work on determining optimal paths in flow fields. Existing work on planning optimal paths for marine vehicles mostly focus on time invariant flows. While, some work has been reported on planning *energy optimal* paths in time-varying flows, these work mostly focus on planning *time optimal* paths.

[1]Dhanushka Kularatne and M. Ani Hsieh are with the Scalable Autonomous Systems Lab, Drexel University, Philadelphia, PA 19104, USA. {dnk32,mhsieh1}@drexel.edu

[2]Subhrajit Bhattacharya is with the Department of Mechanical Engineering and Mechanics, Lehigh University, Bethlehem, PA 18015, USA. sub216@lehigh.edu

Examples include the work by Garau *et al.*[6] where they used a graph search method to plan time optimal paths in time invariant flows. Graph search methods have also been proposed for computing energy optimal paths in flow fields in [9, 14, 11], however these works only address planning in time-invariant flows. Alternatives to graph search methods include [10, 17] for determining energy optimal paths in time-varying flows. However, these methods run the risk of producing a path that is only a local minimum [4] and require optimizations at several levels. Eichorn [4] presented a graph search based method to compute time optimal paths in time-varying flows. Lolla *et al.*[12, 13] presented a level set expansion method to find time optimal paths in time-varying flows and Subramani *et al.*[16] extended this method to compute energy optimal paths from amongst these time optimal paths. Unfortunately, all these methods cannot be extended to compute optimal energy paths in time-varying flows.

To the best of our knowledge, no graph search based method has been proposed for planning optimal energy paths in time-varying flow fields. Though most exist graph search methods for time-invariant flows can theoretically be extended to time-varying flows, the extension is not trivial in practice. Different from time-invariant flows where the energy cost is solely a function of the spatial coordinates, both the traversal time and the energy cost of travel from one vertex to an adjacent vertex depend on the time of departure from the first vertex making the cost and traversal times functions of both time and space in time varying flows. Thus, to compute optimal energy paths in time-varying flow fields, the search space must explicitly consider the time dimension. However, if the time dimension is not handled with care, the search will become very inefficient and return inaccurate results. Furthermore, the design of the search algorithm must also consider additional complexities such as sampling rates, and the effects of flow and actuation constraints on the accessible configuration space.

In this paper, we present two graph search based methods that could be used to obtain energy optimal paths in time-varying flow fields. The rationale for developing these graph search methods are 1) they are simpler to implement than existing energy optimal path planning methods for time-varying flows, 2) it is easier to incorporate the effects of actuation constraints and the effects of static and dynamic obstacles to graph search based methods and 3) they can be easily extended to include topological constraints such as homotopy classes [2, 8] which have applications in environmental exploration and monitoring. Both of the proposed methods result in trajectories that are significantly closer to

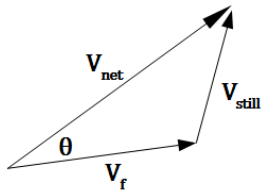


Fig. 1. The net velocity of the vehicle is the vector sum of the flow velocity and the vehicle's still-water velocity.

the true optimal than trajectories obtained from searches in spatial coordinates alone. Additionally, the proposed methods are significantly more efficient than direct searches using a straight forward discretized representation of the spatio-temporal configuration space. Our experimental analysis show that depending on the flow field, one method may outperform the other with respect to computation efficiency or how close the computed trajectory is to the true optimal. While only energy optimal paths are addressed in this paper, the presented methods are general enough to accommodate arbitrary cost functions.

The remainder of the paper is organized as follows. The problem, assumptions and the cost functions are presented in section II and the graph search methods used are described in section III. In section V we present simulation results and we conclude with a discussion of our findings in section VI.

II. PROBLEM FORMULATION

A. Assumptions

In this paper, we consider an aquatic environment $\mathbb{W} \subseteq \mathbb{R}^2$, subject to a time-varying flow field $\mathbf{V_f}(\mathbf{x}, t)$, where $\mathbf{x} = [x,\ y]^T \in \mathbb{W}$ denotes the coordinates of a point in $\mathbb{W}$ expressed in an inertial frame, and $t \in [t_s,\ t_f] \subset \mathbb{R}^+$ denotes time. In the remainder of the paper it is assumed that this flow description is known beforehand or a reliable forecast is available. We assume a holonomic kinematic model for the autonomous marine vehicle (AMV). This is a reasonable assumption when the dimensions of the AMV are small when compared with the dimensions of the flow structures [6]. Thus, the net velocity of a vehicle with respect to the inertial frame is given by $\mathbf{V_{net}}(\mathbf{x}, t) = \mathbf{V_f}(\mathbf{x}, t) + \mathbf{V_{still}}(\mathbf{x}, t)$, where $\mathbf{V_{still}}$ is the velocity of the AMV with respect to the flow. To achieve a given velocity $\mathbf{V_{net}}$, the AMV speed with respect to the flow needs to be

$$V_{still} = \sqrt{(V_{net} - V_f \cos\theta)^2 + (V_f \sin\theta)^2} \tag{1}$$

where $V_{net} = \|\mathbf{V_{net}}\|$, $V_f = \|\mathbf{V_f}\|$, $V_{still} = \|\mathbf{V_{still}}\|$ and θ is the angle between $\mathbf{V_f}$ and $\mathbf{V_{net}}$ as shown in Fig. 1. We further assume that that the actuation capability of the vehicle is limited and the vehicle speed is slower than the surrounding flow *i.e.,* $V_{still}(\mathbf{x}, t) < V_{max} < V_{fm}$ where $V_{fm} = \max_{\mathbf{x}\in\mathbb{W},\ t\in[t_s t_f]} V_f(\mathbf{x}, t)$ is the maximum flow speed encountered in the domain.

In this work we use graph search methods to find energy optimal paths for AMVs in two-dimensional (2D) time-varying flows. We consider the total energy consumed by the vehicle as $E_{total} = E_{hotel} + E_{drag}$ where E_{hotel} is the energy consumed to operate the vehicle's computing and sensor systems independent of propulsion [7], and E_{drag} is the energy expended to overcome drag forces exerted by the fluid. Assuming constant power usage by the computing and sensor systems gives $E_{hotel} = \int_{t_0}^{t} K_h dt$ and assuming a linear drag model gives $E_{drag} = \int_{t_0}^{t} K_d \|\mathbf{V}_{stll}\|^2 dt$. Thus, for infinitesimal path segments, $[dx, dy]^T$, traversed in time dt, we can write,

$$dE = \left(K_h + K_d V_{still}^2\right) dt \tag{2}$$

where V_{still} is given by (1). The implicit assumption made is that V_f remains constant within dx, dy and dt. Note that, K_h and K_d can also be thought of as weighting parameters between minimum time paths and minimum energy paths. If a minimum time path is required, we could set $K_d = 0$ and proceed, and vice versa. If exact energy minimization is required, actual values for K_h and K_d should be used.

B. Problem Statement

Given the above energy cost function, the objective is to find a path that minimizes the total cost. As such, we propose to find a trajectory $\mathbf{\Gamma} : [t_s,\ t_g] \mapsto \mathbb{W}$, that satisfies the following optimization problem:

$$\begin{aligned} &\min_{\Gamma} \int_{\Gamma} dE \\ \text{subject to} \quad &\Gamma(t_s) = \mathbf{x}_s, \\ &\Gamma(t_g) = \mathbf{x}_g, \\ &V_{still} \leq V_{max}. \end{aligned} \tag{3}$$

Here, $\mathbf{x}_s$ and $\mathbf{x}_g$ are the desired start and goal positions.

In this work, we propose a graph based search strategy for computing the optimal path $\mathbf{\Gamma}^\star$ that satisfies the problem stated in (3). Recall that an admissible heuristic function is a function (of the search coordinates) that gives a lower bound for the actual cost (in the graph) to the goal, and is fundamental to implementing an efficient search algorithm such as A*. To find an admissible heuristic function for determining the shortest path from a location $\mathbf{x}_t = [x_t,\ y_t]^T$ to the goal node $\mathbf{x}_g = [x_g,\ y_g]^T$, we consider an idealized flow condition between $\mathbf{x}_t$ and $\mathbf{x}_g$ where the flow is always directed towards $\mathbf{x}_g$ and the flow speed is at the maximum possible value V_{fm} as the vehicle travels between $\mathbf{x}_t$ and $\mathbf{x}_g$. Thus from (1), $V_{net} = V_{still} + V_{fm}$ with $\theta = 0$ since $\mathbf{V_f}$ is always assumed to be along $\mathbf{V_{net}}$ and the travel time given by $\Delta t = \Delta x / V_{net}$ where $\Delta x = \|\mathbf{x}_t - \mathbf{x}_g\|$. Using (2), the cost of travel from $\mathbf{x}_t$ to $\mathbf{x}_g$ is thus given by

$$dE = \left(K_h + K_d V_{still}^2\right) \frac{\Delta x}{V_{still} + V_{fm}}.$$

It can be shown that this cost is minimized when,

$$V_{still} = V_{still_m} = -V_{fm} + \sqrt{V_{fm}^2 + K_h / K_d}.$$

Thus, the heuristic for energy expenditure to travel from $\mathbf{x}_t$ to $\mathbf{x}_g$ is given by

$$h = \left(K_h + K_d v_{sel}^2\right) \Delta t \tag{4}$$

where $v_{sel} = \min(V_{max}, V_{still_m})$ and $\Delta t = \frac{\Delta x}{V_{fm}+v_{sel}}$. In the derivation of (4), we select the minimum between V_{max} and V_{still_m} as the vehicle speed to account for the vehicle actuation constraint. If we ignore the actuation constraint given by $V_{still} < V_{max}$, the heuristic will still be admissible but will result in a further underestimation of the travel cost and lead to more nodes being expanded during the search.

III. METHODOLOGY

Existing graph search methods for energy optimal path planning in flow fields consider time-invariant flows. In these methods, the graph is constructed by uniformly discretizing $\mathbb{W}$. The cost of any edge is considered to be the minimum or optimal energy cost to travel between the two corresponding vertices. Finding the optimal energy path consists of piecing together these optimal path segments from start to goal such that the total cost is minimized and can be done using graph search methods like Dijkstra or A*. This approach is possible since the traversal time from a node $\mathbf{x}_i$ to an adjacent node $\mathbf{x}_j$, as well as the corresponding energy cost of travel, is independent of the time of departure from node $\mathbf{x}_i$. However, in the case of time-varying flows, both the traversal time and the energy cost of travel from $\mathbf{x}_i$ to $\mathbf{x}_j$, are dependent on the time of departure from $\mathbf{x}_i$. Thus, in time-varying flows, it is possible to travel to an intermediate node in a suboptimal manner in order to encounter a favorable current towards the destination later on that results in minimizing the overall cost of the path. Hence, greedy strategies that may be suitable for time-invariant flows are no longer valid. As such, the search space needs to explicitly take into account the time dimension when computing optimal paths in time-varying flow fields.

In this work, we present two methods, the Multi Timestep Search (MTS) method and the Single Timestep Search (STS) method, that can be used to plan paths in time-varying flow fields. These methods allow searching across different actuation speeds in order to compute the global optimum trajectory. In the MTS method, we discretize the 3-D space (spanned by the X-Y-Time coordinates) to construct a uniformly discretized grid such that each point on the grid is represented by a node in the graph. Edges between a vertex and its neighbors exist in this graph if the neighboring vertices are reachable while respecting the actuation constraints. Whereas in the STS method, only the temporal coordinate has a fixed, uniform discretization, and the spatial coordinates of the vertices in the graph are determined on-the-fly based on prevalent flow conditions as we explore the space. In both methods, the graph expansion is guided by the A* algorithm. At each iteration, the vertex to expand is obtained from a priority queue (Q) that is sorted by the f values of the vertices, where $f = cost + heuristic$ and the $cost$ and $heuristic$ values are computed using (2) and (4) respectively..

A. The Multi-Timestep Search (MTS)

In this method (see Algorithm 1), the 3-D space is discretized uniformly with increments $\Delta x, \Delta y, \Delta t$ in each of the x, y, t directions respectively. The discretized space is represented by a vertex set V_{3D} where each vertex $q_i \in V_{3D}$ is identified by the pair $(\mathbf{x}_i, t_i)$. Each q_i also has a set of tentative neighbors $\mathcal{N}(q_i)$, where each $q_j \in \mathcal{N}(q_i)$ satisfies $\max(|x_j - x_i|/\Delta x,\ |y_j - y_i|/\Delta y) \leq N_{sHops}$ and $0 < (t_j - t_i)/\Delta t \leq N_{tHops}$ (see Fig. 2(a)). N_{sHops} defines the number of spatial hops considered as neighbors, and its value is typically set to a small integer in MTS in order to keep the average degree of the vertices in the graph low. If $N_{sHops} = 1$, only the immediate spatial neighbors are considered and if $N_{sHops} = 2$ then 2-hop spatial neighbors are also considered. For a given q_i, with associated flow speed V_{f_i}, the required still-water speed and the associated traversal cost to go to any $q_j \in \mathcal{N}(q_i)$ with a given spatial coordinate $\overline{\mathbf{x}}$ depend primarily on $\delta t_{ij} = t_j - t_i$. The cost to reach $\overline{\mathbf{x}}$ will decrease as δt_{ij} increases (see Fig. 2(a)).

The main idea behind the MTS method is to search through all of these path segments to find the least cost path from start to goal. Naturally, low cost paths can be obtained by setting N_{tHops} to a large value, because it allows transitions between two adjacent spatial locations, $\mathbf{x_i}$ and $\mathbf{x_j}$, albeit over a long period of time ($\delta t_{ij} = N_{tHops} \times \Delta t$). However, such a transition breaks the assumption that the flow velocity $\mathbf{V_{f_i}}$ remains constant between two vertices, q_i and q_j, in practice since flow conditions are more likely to change over the large the transition time δt_{ij}. Thus, N_{tHops} should be selected such that the underlying flow remains relatively constant during $N_{tHops} \times \Delta t$. In this paper, N_{tHops} is set such that $N_{tHops} \times \Delta t < T_s$, where T_s is the sampling period of the available data.

Furthermore, only a subset of the tentative neighbors $\mathcal{N}(q_i)$ is accessible due to actuation constraints, and only those accessible vertices are considered in the construction of the search graph $\mathcal{G}_{3D}$ (line 23 in Algorithm 1). Specifically, $q_j \in \mathcal{N}(q_i)$ is considered to be a neighbor of q_i only if the required still-water speed, $V_{still_{req}} = \sqrt{(V_{net} - V_{f_i}\cos\theta)^2 + (V_{f_i}\sin\theta)^2} < V_{max}$ (where, $V_{net} = \|\mathbf{x}_j - \mathbf{x}_i\|/(\delta t_{ij})$ and $V_{f_i} = \|\mathbf{V_f}(\mathbf{x}_i, t_i)\|$). The A* algorithm is used to guide the construction and expansion of $\mathcal{G}_{3D}$, and to obtain the minimum cost path.

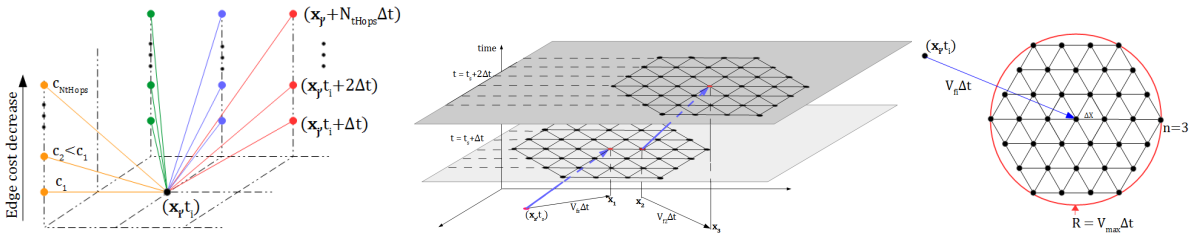
The main highlights of this method are:

- given a node $q_i = (\mathbf{x}_i, t_i)$, for every location $\mathbf{x}_j$ who is a neighbor of q_i in the spatial domain, multiple vertices with $t_j = [t_i + \Delta t, t_i + 2\Delta t, ..., t_i + N_{tHops}\Delta t]$ are considered as neighbors in the 3-D graph (see Fig. 2(a)).
- actuation constraints (maximum possible still-water speed) and prevalent flow conditions are explicitly considered to determine which neighboring vertices are reachable and which edges are to be constructed during the graph search.

Since multiple time steps are considered at each spatial neighbor, this method is called the "Multi Timestep Search (MTS)" method.

B. The Single-Timestep Search (STS)

In the MTS method, all the vertices that can be accessed in a given time step might not be considered due to the N_{sHops}



(a) (b) (c)

Fig. 2. (a) The construction of the MTS graph for $N_{hops} = 1$. Only a portion of the neighbor set is shown. For each spatial neighbor $\mathbf{x}_j$, multiple neighbors are considered along the time axis. (b) The construction of the STS graph. In contrast to the MTS method, only one timestep is considered at each node, and all possible points that can be reached are considered as neighbors. All neighbors have the same time coordinate. (c) The lattice structure with $n = 3$ used to construct the set of accessible neighbors $\mathcal{N}(q_i)$ from q_i.

Algorithm 1: Multi Timestep Search (MTS) to compute optimal paths in time-varying flows

Input : Vertex set V_{3D}, Start vertex $q_s = (\mathbf{x}_s, t_s)$, Goal $\mathbf{x}_g$
Output: Optimal cost path $\mathbf{\Gamma}$

foreach $q_i \in V_{3D}$ **do**
  $q_i.f = \infty, \quad q_i.cost = \infty, \quad q_i.parent = \varnothing$
  $q_i.heuristic = \text{getHeuristic}(q_i, \mathbf{x}_g)$
  $q_i.expanded = false$
end
$Q = \varnothing, \quad \mathcal{G}_{3D} = \varnothing$
$q_s.f = 0, \quad q_s.cost = 0$
$Q.insert(q_s), \quad \mathcal{G}_{3D}.addNode(q_s)$
while $(Q! = \varnothing)$ **do**
  $q_i = Q.extractMin()$
  $q_i.expanded = true$
  $[V_{f_i}, \ \theta] = \text{getFlow}(q_i)$ `// get velocity at` q_i
  if $(\mathbf{x_i} == \mathbf{x}_g)$ **then** `// goal is reached`
    while $q_i! = \varnothing$ **do** `// retrieve path`
      $\mathbf{\Gamma}.push(q_i)$
      $q_i = q_i.parent$
    end
    break;
  end
  foreach $q_j \in \mathcal{N}(q_i)$ **do** `// for each neighbor`
    $V_{net} = \|\mathbf{x}_j - \mathbf{x}_i\|/\delta t_{ij}$ `// required net speed`
    $V_{still_{req}} = \sqrt{(V_{net} - V_{f_i}\cos\theta)^2 + (V_{f_i}\sin\theta)^2}$
    `//` V_{still} `required to reach` q_j
    if $\left(V_{still_{req}} < V_{max}\right)$ **then**
      continue;
    end
    $cost = \left(K_h + K_d V_{still_{req}}^2\right)\delta t_{ij}$ `// cost of edge` $\overline{q_i q_j}$
    if $q_j \in \mathcal{G}_{3D}$ **then**
      if $(!q_j.expanded \ \& \ q_j.cost > q_i.cost + cost)$ **then**
        $q_j.cost = q_i.cost + cost$
        $q_j.parent = q_i$
        $q_j.f = q_j.cost + q_j.heuristic$
        $Q.update(q_j)$
      end
    else
      $q_j.cost = q_i.cost + cost$
      $q_j.parent = q_i$
      $q_j.f = q_j.cost + q_j.heuristic$
      $Q.insert(q_j), \ \mathcal{G}_{3D}.addNode(q_j)$
    end
  end
end
return $\mathbf{\Gamma}$

Algorithm 2: Single Timestep Search (STS) to compute optimal paths in time-varying flows

Input : Start vertex $q_s = (\mathbf{x}_s, t_s)$, Goal $\mathbf{x}_g$, Δt, n
Output: Optimal cost path $\mathbf{\Gamma}$

$Q = \varnothing, \quad \mathcal{G}_{3D} = \varnothing$
$\Delta x = V_{max}\Delta t/n$
$[\mathbf{X}_{lattice}, \ \mathbf{C}_{lattice}] = getLattice(\Delta x, n)$ `// precompute Lattice structure`
$q_s.f = 0, \quad q_s.cost = 0 \quad q_s.parent = \varnothing$
$Q.insert(q_s), \quad \mathcal{G}_{3D}.addNode(q_s)$
while $(Q! = \varnothing)$ **do**
  $q_i = Q.extractMin()$
  $q_i.expanded = true$
  $\mathbf{V_{f_i}} = \text{getFlow}(q_i)$ `// get flow velocity at` q_i
  if $(\|\mathbf{x_i} - \mathbf{x}_g\| < \Delta x/2)$ **then** `// goal is reached`
    while $q_i! = \varnothing$ **do** `// retrieve path`
      $\mathbf{\Gamma}.push(q_i)$
      $q_i = q_i.parent$
    end
    break;
  end
  for $j = 1$ *to* m **do** `// for each vertex in lattice`
    $\mathbf{x_j} = \mathbf{x_i} + \mathbf{V_{f_i}}\Delta t + \mathbf{X}_{lattice}(j)$ `// spatial coordinates of` q_j
    $t_j = t_i + \Delta t$
    $q_j = (\mathbf{x_j}, \ t_j)$
    $cost = \mathbf{C}_{lattice}(j)$ `// cost of edge` $\overline{q_i q_j}$
    if $q_j \in \mathcal{G}_{3D}$ **then**
      if $(!q_j.expanded \ \& \ q_j.cost > q_i.cost + cost)$ **then**
        $q_j.cost = q_i.cost + cost$
        $q_j.parent = q_i$
        $q_j.f = q_j.cost + q_j.heuristic$
        $Q.update(q_j)$
      end
    else
      $q_j.cost = q_i.cost + cost$
      $q_j.parent = q_i$
      $q_j.heuristic = \text{getHeuristic}(q_j, \mathbf{x}_g)$
      $q_j.f = q_j.cost + q_j.heuristic$
      $Q.insert(q_j), \ \mathcal{G}_{3D}.addNode(q_j)$
    end
  end
end
return $\mathbf{\Gamma}$

hops restriction. Thus, in some cases, all possible paths to the goal may not be considered since vertices that are accessible due to prevalent flow conditions are not included in the search since they are farther away than N_{sHops} hops. Thus the results can be suboptimal. To overcome this, the Single Timestep Search (STS) considers all possible vertices that could be accessed in a single time step from a given vertex as its neighbors. However, the spatial locations of these accessible vertices depend on the flow conditions prevalent at the time at that location. As such, in the STS method, the spatial locations of the vertices are determined on-the-fly based on prevalent flow conditions, as we explore the space. Since only one time step is considered from each neighbor, we refer to this method as the "Single Timestep Search (STS)" method (see Algorithm 2).

Similar to the MTS, each vertex q_i is identified by the pair $(\mathbf{x_i},\ t_i)$. The spatial location of the vertex that could be reached from q_i in a single time step Δt with zero vehicle speed is $\overline{\mathbf{x}} = \mathbf{x_i} + \mathbf{V_{f_i}}\Delta t$. Thus the accessible spatial region from q_i in a single time step Δt, is a circle of radius $V_{max}\Delta t$ centered at $\overline{\mathbf{x}}$. In the STS method, this accessible region is represented by a hexagonal lattice centered at $\overline{\mathbf{x}}$ with $2n+1$ vertices on the major axis (see Fig. 2(c)). Thus there will be $m = 3n^2 + 3n + 1$ number of vertices in the neighbor set $\mathcal{N}(q_i)$ of q_i, and the inter-vertex spacing would be $\Delta x = V_{max}\Delta t/n$. The relative position of each of the vertices in the lattice with respect to $\overline{\mathbf{x}}$, and the cost required to reach them from $\overline{\mathbf{x}}$ are the same for any $\overline{\mathbf{x}}$. Thus these values are precomputed in arrays $\mathbf{X}_{lattic}$ and $\mathbf{C}_{lattice}$ respectively (line 3 in Algorithm 2), to save computational resources. The actual spatial coordinate of each neighbor is calculated (line 18) using these precomputed values and the flow velocity at q_i. All the vertices in $\mathcal{N}(q_i)$ have the same time coordinate $t_j = t_i + \Delta t$ (line 19). Furthermore, no additional accessibility check is required (as in the MTS method), since all vertices are guaranteed to be accessible. A* algorithm is used to search for the optimal path in this graph as its built. In the termination condition in line 10, it is assumed that the goal location is reached when the distance to the goal location is less than $\Delta x/2$. This same condition is used when searching the graph in line 22 to find if a vertex already exists in the graph.

IV. Time Complexity Analysis

In our analysis, we assume that the priority queue Q is implemented as a min-priority queue using a heap data structure in both algorithms. Thus, for a queue of length n_q, each of the $extractMin()$, $update(q_j)$, & $insert(q_j)$ operations has a worst case running time of $\mathbb{O}(\log n_q)$. We also assume that all other value assignment operations can be performed in constant time. The $q_j \in \mathcal{G}_{3D}$ operation (lines 20 and 17 in Algorithms 1 and 2 respectively) can also be performed in constant time by using a hash table. Thus, we only consider the $extractMin()$, $update(q_j)$, & $insert(q_j)$ operations contained inside the $while$ loop, in our analysis since all other operations can be done in constant time.

Lets assume that N nodes are expanded by each algorithm to find a path $\mathbf{\Gamma}$, and m neighbors are considered at each node, *i.e.,* $m = \|\mathcal{N}(q_i)\|$. In both algorithms, for each node expanded, the $extractMin()$ operation is performed exactly once, and one of the $update(q_j)$ or $insert(q_j)$ operations is performed at the most m times. Furthermore, in the worst case, there are Nm number of nodes in the priority queue Q. As such, the worst case running time of both algorithms is $\mathbb{O}(Nm \log Nm)$ (since altogether Nm number of $insert(q_j)/update(q_j)$ operations are carried out on a queue of length Nm in the worst case).

In the STS method, N and m depend only on the discretization $\Delta x, \Delta t$, while in the MTS method, N and m would depend on the discretization $\Delta x, \Delta y, \Delta t$, as well as on N_{sHops} and N_{tHops}. Therefore, the N and m values that are required by the two methods to compute a path with a given accuracy (in comparison to the true optimal path), will be different for the two methods. Thus, in order to compare the running times of the two methods, we need to compare the Nm product required by each method. The method that computes the path with a smaller Nm value will be faster. This is analyzed through simulations in Section V.

TABLE I

Performance values of the paths shown in Fig. 3 and Fig. 4

	Path1 $K_h = 0.0005,\ K_d = 1$		Path2 $K_h = 0.05,\ K_d = 1$	
	cost (Nm)	mE (m)	cost (Nm)	mE (m)
STS	3526	497	12603	556
MTS	3873	477	12819	515
Optimal Path	3787	-	12533	-

V. Simulation Results

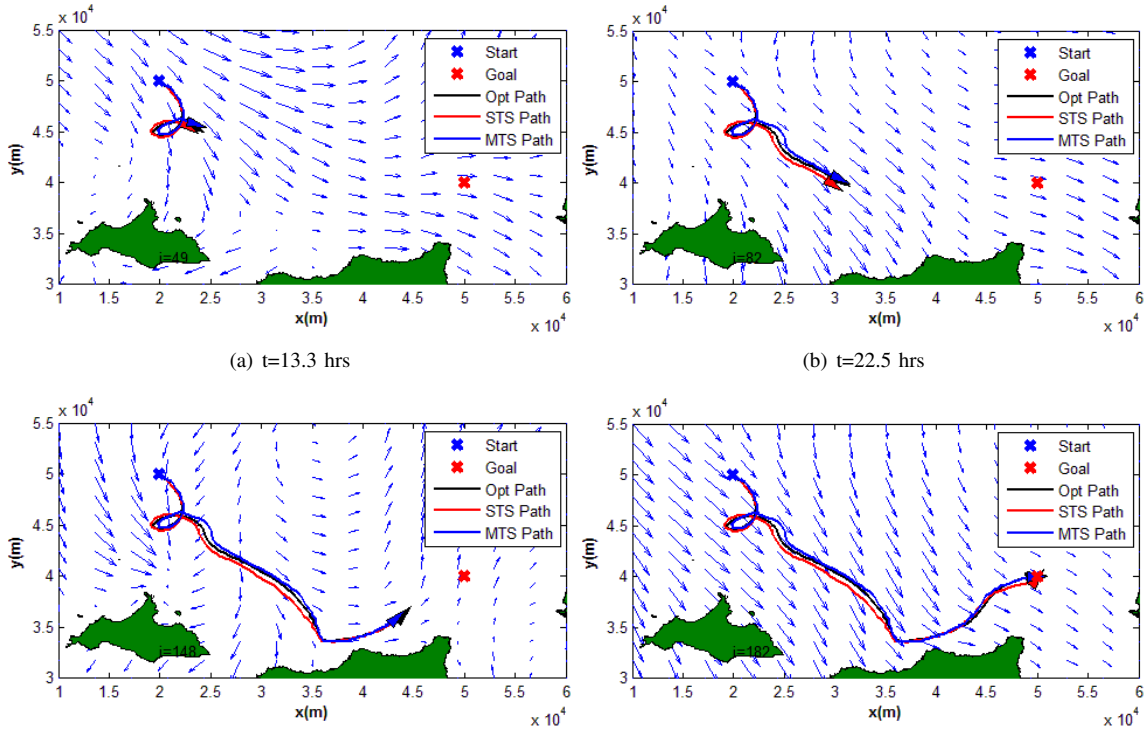
In this section, the performance of the methods presented in the section III are evaluated in simulations. Flow data generated by the Regional Ocean Model System (ROMS) for the Santa Barbara Bay area off the coast of southern California were used in these simulations. The Southern California Coastal Ocean Observing System (SCCOOS) generates these hourly ocean current forecasts everyday and each forecast is for 72 hours [1]. The the data generated on July 7 and July 8 2016 were used for the simulations. The maximum flow speed was $V_{fm} = 0.73m/s$ and as such V_{max} was selected to be $0.5m/s$. All the simulations were run on a Core I-7 3.4GHz PC with 16GB of RAM.

The accuracy of the paths computed by the two methods is evaluated by comparing them against a path obtained by solving the corresponding optimal control problem. Let $\mathbf{\Gamma}^* : [t_s,\ t_g] \mapsto \mathbb{W}$ be the baseline path obtained from the optimal control or optimal trajectory generation formulation, and let $\mathbf{\Gamma} : [t_s,\ t_g] \mapsto \mathbb{W}$ be the path computed by any one of the proposed methods. The mean error (mE) between $\mathbf{\Gamma}^*$ and $\mathbf{\Gamma}$, defined by

$$mE = \int_{t_s}^{t_g} \frac{\|\mathbf{\Gamma}^*(t) - \mathbf{\Gamma}(t)\|}{t_g - t_s} dt \tag{5}$$

is used to evaluate the relative accuracy of a given path.

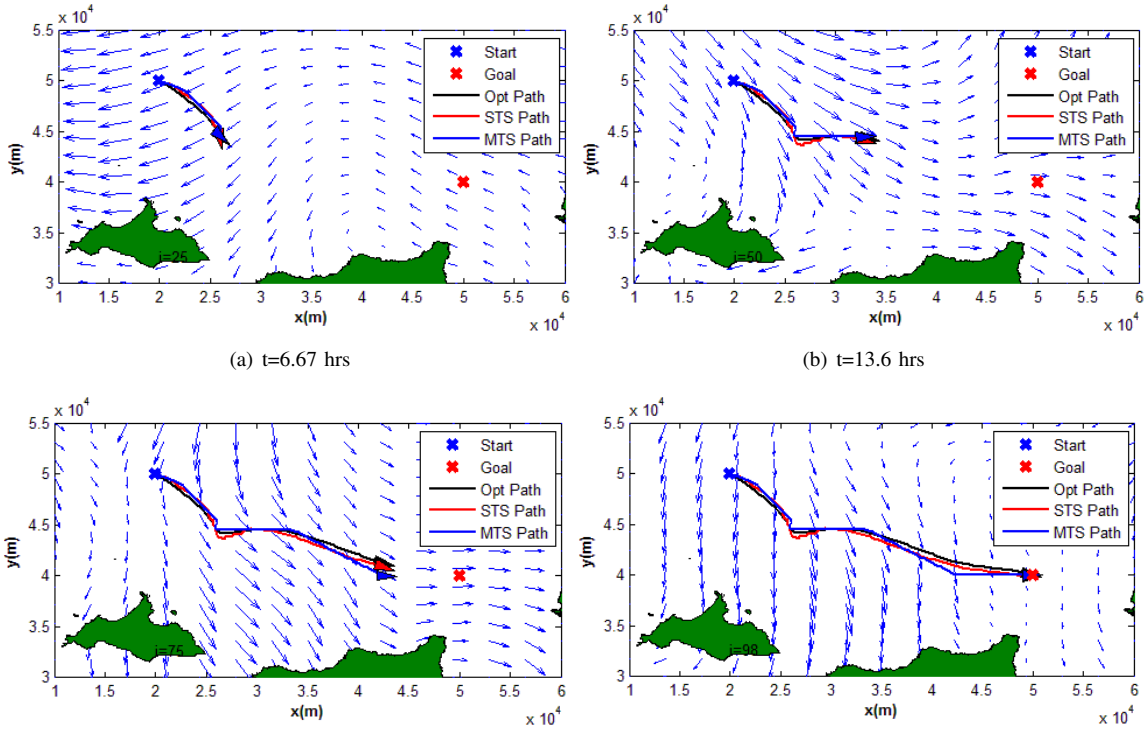
Fig. 3 shows the time evolution of the paths computed by the MTS and STS methods against that of the optimal path



(a) t=13.3 hrs (b) t=22.5 hrs

(c) t=40.8 hrs (d) t=50.3 hrs

Fig. 3. Comparison of optimal paths computed by the two methods against the optimal path computed from optimal control theory for $K_h = 0.0005$ and $K_d = 1$. Paths try to follow the flow at all times



(a) t=6.67 hrs (b) t=13.6 hrs

(c) t=20.6 hrs (d) t=26.9 hrs

Fig. 4. Comparison of optimal paths computed by the two methods against the optimal path computed from optimal control theory for $K_h = 0.05$ and $K_d = 1$. Paths take a more direct route towards the goal.

$\mathbf{\Gamma}^*(t)$, for $K_h = 0.0005$ and $K_d = 1$. In this case, more prominence is given to minimizing the energy expended to overcome drag (since $K_h << K_d$). As such, the computed paths try to follow the direction of the flow at all times to reduce the relative speed between the flow and the vehicle. This results in loop structures (in space) as can be seen in the

TABLE II

COMPARISON OF RUNNING TIMES OF THE TWO METHODS TO COMPUTE PATHS WITH SIMILAR ACCURACY VALUES BETWEEN THE SAME END POINTS. $K_d = 1$ FOR ALL PATHS.

Path	Start (km)	Goal (km)	K_h	STS Path			MTS Path		
				Accuracy mE (m)	Nodes $\times$ neighbors Nm (/ 10^6)	Running Time $\mathbb{O}(Nm \log Nm)$ (/ 10^9)	Accuracy mE (m)	Nodes $\times$ neighbors Nm (/ 10^6)	Running Time $\mathbb{O}(Nm \log Nm)$ (/ 10^9)
1	$[20, 50]^T$	$[50, 40]^T$	0.0005	497	1859	57.2	477	457	13.2
2	$[20, 50]^T$	$[50, 40]^T$	0.05	556	59.2	1.53	515	72.0	1.89
3	$[20, 50]^T$	$[25, 45]^T$	0.0005	335	185	5.08	322	657	19.2
4	$[20, 50]^T$	$[25, 45]^T$	0.05	181	14.8	0.353	188	354	10.0

figure. Fig. 4 shows the time evolution of the paths for the same start and end locations, but with $K_h = 0.05$ and $K_d = 1$. In this case, more prominence is given to minimizing the the hotel load ($\int K_h dt$). Thus the paths try to minimize time spent to complete the path, and as a result, the computed paths tend to take a more direct route to the destination while also trying to align with the flow as much as possible. From the performance parameters for the two cases given in Table I, it can be seen that the mean error (mE) for the paths computed by both methods is low.

As described in section IV, both N (number of nodes expanded) and m (the number of neighbors), that is required to produce a path with a given accuracy, would be different for the two algorithms. Furthermore, the time complexity of both algorithms is $\mathbb{O}(Nm \log Nm)$, *i.e.,* grows with Nm. Thus, in order to compare the performance of the two algorithms, the Nm product required by the two methods to produce paths with similar accuracy values were compared (see Table II. The method with the lower Nm value will compute the path faster).

From the results, it can be seen that the MTS method seems to take longer than the STS method to compute a path with similar accuracy. However, exceptions to this rule (e.g. Path 1) were also found, indicating that the actual relative computation speed of the two methods also depends on the flow field prevalent in the region at the time of interest. Furthermore, it can be seen that both methods take considerably less time to compute paths between the same end points when K_h is larger. This is is expected because the number of nodes that are expanded (N) by each method to produce a solution depends on the effectiveness of the heuristic function. It can be seen from the derivation 4 that the heuristic $h \to 0$ as $K_h/K_d \to 0$. Thus when K_h is small (as in Paths 1 and 3), the effectiveness of the heuristic is reduced and more nodes are expanded.

In graph search based methods used to compute optimal paths in time invariant flows [9, 6, 11], in which only spatial coordinates are considered, the accuracy of the computed paths increase as the resolution of the discretization is increased. However, it was observed that this is not generally true for time-varying flows. Table III shows the variation of the accuracy values for a path computed using MTS, as the resolution used to discretize the 3D space is varied. Going from Path1 to Path2 and from Path3 to Path4, where Δt is held constant, the accuracy is increased (mE is reduced) as Δx is reduced. However, for Paths 4-6, the accuracy

TABLE III

THE VARIATION OF THE PATH ACCURACY WITH DISCRETIZATION RESOLUTION FOR THE MTS METHOD. FOR THE CONSIDERED PATH $\mathbf{x_s} = [20, 50]km$, $\mathbf{x_s} = [50, 40]km$, $K_h = 0.0005$ AND $K_d = 1$.

	Path1	Path2	Path3	Path4	Path5	Path6
$\Delta t(s)$	1000	1000	500	500	500	500
$\Delta x(m)$	200	150	250	200	150	100
N_{tHops}	3	3	6	6	6	6
N_{sHops}	3	3	3	3	3	3
mE	887	610	639	477	522	832

TABLE IV

THE VARIATION OF THE PATH ACCURACY WITH DISCRETIZATION RESOLUTION FOR THE STS METHOD. FOR THE CONSIDERED PATH $\mathbf{x_s} = [20, 50]km$, $\mathbf{x_s} = [50, 40]km$, $K_h = 0.0005$ AND $K_d = 1$.

	Path1	Path2	Path3	Path4	Path5	Path6
$\Delta t(s)$	1000	1000	1000	2000	2000	500
n	3	4	5	3	6	2
$\Delta x(m)$	167	125	100	333	167	125
$mE(m)$	896	1533	497	1010	901	1540

decreases (mE increases) as Δx is reduced (as the discretization is made finer). Table IV shows the corresponding measures for the STS method. When comparing Path 1 with Path2, it can be seen that the accuracy decreases as Δx is reduced, whereas comparing Path1 with Path3, the accuracy increases as Δx is reduced. Thus, the relationship between the accuracy and the discretization resolution for both methods is not entirely clear at this point. Interestingly, the STS method seems to produce paths with similar accuracy values when the same Δx is used (compare Path1 with Path 5, and Path2 with Path6). However, such a conclusion is preliminary and needs to be further investigated.

VI. CONCLUSIONS

In this paper two methods for generating optimal energy paths in time varying-flows were presented. The MTS method considers the possibility of traveling to each spatial neighbor in multiple time steps, where as the STS method considers traveling to all possible spatial locations in a single time step. The time complexity of both algorithms was found to be $\mathbb{O}(Nm \log Nm)$. The accuracy of the two methods was verified by comparing the computed paths with those obtained through optimal control theory.

While we only considered the generation of optimal energy paths, the presented methods are general enough to optimize any cost function in a time-varying flow. Both methods explicitly consider the time dimension in the graph construction, and thus computes trajectories that are signif-

icantly closer to the true optimal than trajectories obtained from search in spatial coordinates alone. In addition, it is not necessary to search through the whole time dimension due to the way that the graphs are constructed, and as a result the paths are computed significantly more efficiently than than searches in a full-blown discrete representation of the spatio-temporal configuration space.

It was found that the accuracy of the computed paths were dependent on the discretization used for the configuration space. However, in stark contrast to searches in spatial corrdinates alone (such as the ones used in time-invariant flow scenarios), increasing the discretization resolution sometimes resulted in reduced accuracy. To obtain the best accuracy, the discretization has to match the spatio-temporal scales of the underlying flow field. Finding this "sweet spot" remains a challenge and is a direction for future work.

REFERENCES

[1] Regional ocean model system (roms) model output. URL `http://www.sccoos.org/data/roms-3km/`.

[2] Subhrajit Bhattacharya, Maxim Likhachev, and Vijay Kumar. Topological constraints in search-based robot path planning. *Autonomous Robots*, 33(3):273–290, October 2012. ISSN 0929-5593. DOI: 10.1007/s10514-012-9304-1.

[3] D. Caron, B. Stauffer, S. Moorthi, A. Singh, M. Batalin, E. Graham, M. Hansen, W. Kaiser, J. Das, A. de Menezes Pereira, B. Zhang A. Dhariwal, C. Oberg, and G. Sukhatme. Macro- to fine-scale spatial and temporal distributions and dynamics of phytoplankton and their environmental driving forces in a small subalpine lake in southern california, usa. *Journal of Limnology and Oceanography*, 53(5):2333–2349, 2008.

[4] Mike Eichhorn. Optimal routing strategies for autonomous underwater vehicles in time-varying environment. *Robotics and Autonomous Systems*, 67:33 – 43, 2015.

[5] E Fiorelli, N E Leonard, P Bhatta, D A Paley, R Bachmayer, and D M Fratantoni. Multi-AUV Control and Adaptive Sampling in Monterey Bay. *Oceanic Engineering, IEEE Journal of*, 31(4): 935–948, October 2006. ISSN 0364-9059. doi: 10.1109/JOE.2006.880429.

[6] 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, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 194–198, April 2005. doi: 10.1109/ROBOT.2005.1570118.

[7] V.T. Huynh, M. Dunbabin, and R.N. Smith. Predictive motion planning for auvs subject to strong time-varying currents and forecasting uncertainties. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1144–1151, 2015. doi: 10.1109/ICRA.2015.7139335.

[8] Soonkyum Kim, Subhrajit Bhattacharya, and Vijay Kumar. Path planning for a tethered mobile robot. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, May 31 - June 7 2014.

[9] Teong-Beng Koay and M. Chitre. Energy-efficient path planning for fully propelled auvs in congested coastal waters. In *OCEANS - Bergen, 2013 MTS/IEEE*, pages 1–9, 2013. doi: 10.1109/OCEANS-Bergen.2013.6608168.

[10] 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*, pages 4265–4270, April 2007. doi: 10.1109/ROBOT.2007.364135.

[11] Dhanushka Kularatne, Subhrajit Bhattacharya, and M. Ani Hsieh. Time and energy optimal path planning in general flows. In *Proceedings of Robotics: Science and Systems*, AnnArbor, Michigan, June 2016. doi: 10.15607/RSS.2016.XII.047.

[12] T. Lolla, M. P. Ueckermann, P. Haley, and P. F. J. Lermusiaux. Path planning in time dependent flow fields using level set methods. In *in the Proc. IEEE International Conference on Robotics and Automation*, Minneapolis, MN USA, May 2012.

[13] Tapovan Lolla, Pierre F. J. Lermusiaux, Mattheus P. Ueckermann, and Patrick J. Haley. Time-optimal path planning in dynamic flows using level set equations: theory and schemes. *Ocean Dynamics*, 64(10):1373–1397, 2014. ISSN 1616-7228.

[14] Dushyant Rao and Stefan B Williams. Large-scale path planning for underwater gliders in ocean currents. In *Australasian Conference on Robotics and Automation (ACRA)*, Sydney),, 2009.

[15] Ryan N Smith, Jnaneshwar Das, Hordur Heidarsson, Arvind Pereira, Ivona Cetinić, Lindsay Darjany, Marie-Ève Garneau, Meredith D Howard, Carl Oberg, Matthew Ragan, Astrid Schnetzer, Erica Seubert, Ellen C Smith, Beth A Stauffer, Gerardo Toro-Farmer, David A Caron, Burton H Jones, and Gaurav S Sukhatme. {USC} {CINAPS} Builds Bridges: Observing and Monitoring the {S}outhern {C}alifornia {B}ight. *IEEE Robotics and Automation Magazine, Special Issue on Marine Robotics Systems*, 17(1):20–30, March 2010.

[16] Deepak N. Subramani and Pierre F.J. Lermusiaux. Energy-optimal path planning by stochastic dynamically orthogonal level-set optimization. *Ocean Modelling*, 100:57 – 77, 2016.

[17] Jonas Witt and Matthew Dunbabin. Go with the flow: Optimal auv path planning in coastal environments. In *Australasian Conference on Robotics and Automation (ACRA)*, 2008.