Autonomous Data Collection with Limited Time for Underwater Vehicles

James McMahon and Erion Plaku

Abstract—This paper studies the problem of autonomous data collection where an underwater vehicle is required to reach several target regions within a specified time limit. The proposed approach takes into account the vehicle dynamics, the time-varying ocean currents, and the obstacles in the region in order to effectively plan a collision-free and dynamically-feasible trajectory whose time duration does not exceed the time limit. When the time limit makes it impossible to reach every target, the approach seeks to reduce the penalty accrued by the target regions that are not visited. The approach combines sampling-based motion planning with constraint-based solvers. In fact, a constraint-based solver searches a navigation roadmap to compute bounded tours which minimize the accrued penalty. Sampling-based motion planning is then used to expand a motion tree along these tours. Unsuccessful tour expansions are penalized to avoid becoming stuck and enable the approach to effectively plan collision-free and dynamically-feasible trajectories that reduce the accrued penalty.

Index Terms—Motion and Path Planning; Marine Robotics

I. INTRODUCTION

AUTONOMOUS underwater vehicles (AUVs) are an integral part of oceanographic data collection. Much like the problem of data gathering from wireless sensor networks [1]–[3], AUVs are often tasked with reaching specific areas of interest to collect data for exfiltration or to take measurements. A pertinent example arises in Mine Countermeasures where an AUV is tasked with surveying areas of interest for mines. To work effectively, an AUV needs to reason about the importance of the data, the locations where the data will be collected, and the restrictions imposed by the interaction between the AUV and the environment.

Another significant challenge relates to the limited energy resources available to the AUV. As a result, the planner must ensure that the AUV accomplishes the mission within a given time so that sufficient energy resources remain for its return.

At a high level, the autonomous data-collection problem is often framed as the Prize Collecting Traveling Salesman Problem (PC-TSP) [4]. In a PC-TSP setting, each city that is visited yields a prize, or, equivalently, a penalty is accrued for each city that is skipped. The objective is to minimize the accrued penalty while not exceeding a specified upper bound on the distance traveled. This formulation lends itself to autonomous data collection where targets may provide data with varying levels of importance. A first approach is to connect the targets via a graph and then use a PC-TSP solver to find a tour that minimizes the distance traveled and the accrued penalty. Different methods such as those based on Lagrangian heuristics [5], self-organizing maps [6], or tabu search [7] have been developed. The work in [8] uses PC-TSP to develop path-planning algorithms and communication protocols to enable an AUV gather data from an underwater sensor network. The work in [9] develops a mixed integer linear programming approach to plan paths in a 2D discretized setting that optimizes an objective function. The work in [10] considers the online version of PC-TSP where targets are revealed over time. TSP and PC-TSP formulations have been used to develop approaches that determine inspection locations [11], inspect underwater 3D structures [12], survey seabeds [13], or route vehicles [14].

While there is extensive research on PC-TSP, related work has generally focused on a setting that does not take into account the vehicle dynamics or external forces acting on it. As a result, the planned tours are not necessarily dynamically feasible or consider the actual cost of following the tour.

To account for these challenges, the proposed approach combines PC-TSP with sampling-based motion planning [15]. The approach takes into account the limited energy resources available to the AUV and plans a trajectory whose time duration does not exceed the specified upper bound, focusing on the navigation of the vehicle between data collection sites and the associated constraints rather than the act of collecting the data itself. The approach seeks to reduce the overall distance traveled while minimizing the accrued penalty. The vehicle dynamics, the effects of the time-varying ocean currents, and the obstacles in the region are also taken into account during planning so that the planned motions are dynamically-feasible and collision-free.

Sampling-based motion planning is used to incrementally expand a motion tree whose branches correspond to collision-free and dynamically-feasible trajectories. A constraint-based solver guides the tree expansion by computing short PC-TSP tours that minimize the accrued penalty. The PC-TSP solver operates over a graph representation obtained by constructing a navigation roadmap that connects the target regions with collision-free paths. The motion tree is expanded by seeking to follow the PC-TSP tours while avoiding the obstacles and accounting for the vehicle dynamics and the drift caused by the time-varying ocean currents. Unsuccessful tour expansions are penalized to avoid becoming stuck and enable the approach.
to discover alternative tours. This coupling of sampling-based motion planning with PC-TSP solvers enables the approach to efficiently compute collision-free and dynamically-feasible trajectories that enable the AUV to reach multiple targets while reducing the distance traveled and the accrued penalty.

The proposed approach significantly extends prior work which combines sampling-based motion planning with a TSP solver but does not take into account the limited resources [16]–[18]. The PC-TSP setting makes the problem considerably harder and requires significant algorithmic improvements. Specifically, the planner has to determine which branches in the motion tree it should expand to reduce the accrued penalty while respecting the upper bound on the execution time. A multi-objective criterion is developed which leverages the time duration of the trajectories in the motion tree, accrued penalty, previous selections, and the time duration of the trajectories in the motion tree.

The approach is validated both in simulation and field experiments conducted in the Boston Harbor with the Reliant AUV, which is a Bluefin-21 vehicle available at the U.S. Naval Research Laboratory (NRL) Code 7130. In practice, the approach provides a planning and re-planning framework for use on an AUV for the reacquisition phase of Mine Countermeasures.

II. Problem Formulation

The approach takes as input a map of the 3D environment, providing information about forbidden regions, which the AUV should always avoid, and target regions, which the AUV should reach. The approach also requires the ocean bathymetry, a model of the time-varying ocean currents, and an AUV simulator that takes into account its dynamics and the ocean currents. The mission specifies the penalty for each target region and an upper bound on the time to execute the mission. Give all these, the approach computes a collision-free and dynamically-feasible trajectory that enables the AUV to reach as many targets as possible, without violating the upper bound on the execution time, while also reducing the accrued penalty and the distance traveled. Details follow.

A. Components

1) Environment: Let \( \mathcal{W} \) denote the bounding box of the 3D environment where the AUV conducts its mission. Forbidden regions, which could correspond to obstacles or unsafe areas, are specified as a set \( \mathcal{O} = \{ \mathcal{O}_1, \ldots, \mathcal{O}_m \} \), where \( \mathcal{O}_i \subseteq \mathcal{W} \) denotes an area that the AUV should always avoid. Target regions, which the AUV should visit, are also specified as a set \( \mathcal{R} = \{ \mathcal{R}_1, \ldots, \mathcal{R}_n \} \), where \( \mathcal{R}_i \subseteq \mathcal{W} \).

2) Bathymetry: A bathymetry map provides the depth at each location \((x, y)\) inside \( \mathcal{W} \). For the experiments, the bathymetry map is obtained from the National Geophysical Data Center (NGDC) [19] as a table which stores the depth at each cell in a grid imposed over the \( xy \)-dimensions of \( \mathcal{W} \).

3) Ocean Currents: A model \( \text{CURRENTS}(x, y, d, t) \) indicates the drift forces caused by the time-varying ocean currents at each position \((x, y)\), depth \(d\), and time \(t\). For the experiments, the model is obtained from the Chesapeake Bay Operational Forecast System [20] as several tables labeled by the time duration where each table stores the drift forces at each cell in a grid imposed over \( \mathcal{W} \).

4) AUV Simulator: MOOS-IvP [21] has been shown to accurately simulate the AUV dynamics while also taking into account the ocean currents. The simulated vehicle used in this paper corresponds to the Reliant AUV, which was also used in the field experiment. MOOS-IvP uses a second-order dynamical model, which includes velocity, thrust, rudder, elevator, turn radius, and buoyancy. In the simulation, each state \( s \in \mathcal{S} \), where \( \mathcal{S} \) denotes the state space, comprises the position, depth, orientation, linear and rotational velocity. Each control input \( u \in \mathcal{U} \), where \( \mathcal{U} \) denotes the control space, specifies the desired heading, depth, and speed. Given the current state \( s \in \mathcal{S} \), control input \( u \in \mathcal{U} \), and currents \( f \in \mathbb{R}^3 \), MOOS-IvP computes the new state \( s_{\text{new}} \in \mathcal{S} \) by simulating the AUV dynamics for one time step \( dt \), i.e.,

\[
    s_{\text{new}} \leftarrow \text{SIMULATE}(s, u, f, dt).
\] (1)

Internally, MOOS-IvP uses PID controllers to steer the AUV toward the desired heading, depth, and speed specified by \( u \). The same controllers are also used on the real Reliant AUV.

a) Collision Checking: Collision checking determines whether or not the AUV collides with any of the forbidden regions or the ocean floor when placed according to the position and orientation specified by the new state. The PQP [22] package is used to efficiently implement the collision-checking function \( \text{COLLISION} : \mathcal{S} \rightarrow \{ \text{true}, \text{false} \} \).

b) Motion Trajectory: A trajectory \( \zeta : \{ 0, 1, \ldots, \ell \} \rightarrow \mathcal{S} \) corresponds to a sequence of states which is obtained by applying input controls \( \langle u_0, u_1, \ldots, u_{\ell-1} \rangle \) in succession starting at some state \( s \in \mathcal{S} \) and time \( t \). Specifically,

\[
    \zeta(0) \leftarrow s \quad \text{and} \quad \zeta(i + 1) \leftarrow \text{SIMULATE}(\zeta(i), u_i, f_i, dt),
\] (2)

where \( f_i \leftarrow \text{CURRENTS}(\text{POS}(\zeta(i)), \text{DEPTH}(\zeta(i)), t + i dt) \) corresponds to the drift forces caused by the time-varying ocean currents. Note that \( \text{POS}(\zeta(i)) \) and \( \text{DEPTH}(\zeta(i)) \) denote the position and depth components of the state \( \zeta(i) \).

B. Multi-Target Motion Planning with Limited Time

The AUV is required to reach as many target regions as possible without exceeding an upper bound, \( \mathcal{H} \), on the execution time. The problem setting considered in this paper does not force the vehicle to return to the initial position. The objective is to reduce the penalty accrued by the targets that are not reached, where \( \text{PENALTY}(\mathcal{R}_i) \) denotes the penalty associated with \( \mathcal{R}_i \). Let \( \text{TARGET}(s) \) indicate the target, if any, that the AUV has reached at state \( s \in \mathcal{S} \), i.e.,

\[
    \text{TARGET}(s) = \begin{cases} 
    \{ \mathcal{R}_1 \}, & \text{if } \text{POSD}(s) \in \mathcal{R}_1 \\
    \emptyset, & \text{if } \text{POSD}(s) \notin \bigcup_{\mathcal{R}_i \in \mathcal{R}_1} \mathcal{R}_i,
    \end{cases}
\] (3)

where \( \text{POSD}(s) \) denotes the position and depth components of the state \( s \). Let \( \text{TARGET}(\zeta) \) denote the targets, if any, reached by the motion trajectory \( \zeta : \{ 0, 1, \ldots, \ell \} \rightarrow \mathcal{S} \), i.e.,

\[
    \text{TARGET}(\zeta) = \text{TARGET}(\zeta(0)) \cup \ldots \cup \text{TARGET}(\zeta(\ell)).
\] (4)
The penalty associated with \( \zeta \) accrues the penalties of the target regions that it did not reach, i.e.,

\[
\text{Penalty}(\zeta) = \sum_{R_i \in \mathcal{G}} \text{Penalty}(R_i).
\]

The problem can now be stated as follows.

**Definition 1:** Given

- a description of the environment in terms of its bounding box \( \mathcal{W} \), forbidden regions \( \mathcal{O} = \{O_1, \ldots, O_m\} \), target regions \( \mathcal{R} = \{R_1, \ldots, R_n\} \), bathymetry map, and time-varying drift model currents \((x, y, d, t)\),
- an AUV model including the state space \( \mathcal{S} \), control space \( \mathcal{U} \), time step \( dt \), and simulate \((s, u, f, dt)\),
- an initial state \( s_{\text{init}} \in \mathcal{S} \),
- an upper bound \( \mathcal{H} \) on the execution time of the mission,
- a function \( \text{Penalty} : \mathcal{R} \rightarrow \mathbb{R}^{\geq 0} \) to determine the penalty for each region that is not reached by the AUV.

The approach seeks to reduce overexploration and promote the generation of trajectories that minimize the distance traveled and the accrued penalty. The group-expansion procedure seeks to expand the motion tree from the selected group along the roadmap path defined by its tour. The approach invokes at each iteration the group selection and the group expansion until it finds a solution or reaches a runtime limit. The roadmap construction, the motion-tree partition, the computation of the bounded tours, and the overall search are described in more detail in Sections III-A, III-B, III-C, III-D, respectively.

### III. Method

The approach has several key components: (i) construction of a navigation roadmap, (ii) computation of PC-TSP tours, and (iii) expansion of a motion tree. The roadmap captures the connectivity of the operational area and provides navigation routes that connect the target regions. PC-TSP tours, computed over the roadmap, guide the expansion of a motion tree, which takes into account the AUV dynamics, the ocean currents, and the forbidden regions. Fig. 1 shows a schematic representation of the approach.

Drawing from PRM [23], the navigation roadmap is constructed by randomly sampling waypoints and connecting neighboring waypoints with collision-free paths. The roadmap does not take into account the AUV dynamics or the ocean currents since doing so requires exact steering between the endpoints of each roadmap edge. Exact steering leads to two-boundary value problems, which cannot be solved analytically due to the nonlinearity of the dynamics and the ocean currents. Numerical solutions leave large gaps and impose significant computational costs which render the roadmap construction impractical [24], [25].

To account for the AUV dynamics and the ocean currents, sampling-based motion planning is used to expand a tree of collision-free and dynamically-feasible motions. The motion-tree expansion does not require exact steering but only the ability to simulate the AUV dynamics and the effects of the ocean currents. The roadmap and the target regions are used to partition the motion tree into equivalent groups. Each group is associated with a list of target regions that have already been reached by the motion tree. Using the PC-TSP solver to search the roadmap, a bounded tour is computed for each group in order to determine which additional targets should be reached within the remaining time.

The core loop of the approach consists of selecting a group and expanding the motion tree from the selected group. The group selection relies on a multi-objective criterion which leverages the tour length, accrued penalty, and the number of previous selections. By giving preference to groups associated with short tours, small penalties, and few previous selections, the approach seeks to reduce overexploration and promote the generation of trajectories that minimize the distance traveled and the accrued penalty. The group-expansion procedure seeks to expand the motion tree from the selected group along the roadmap path defined by its tour. The approach invokes at each iteration the group selection and the group expansion until it finds a solution or reaches a runtime limit. The roadmap construction, the motion-tree partition, the computation of the bounded tours, and the overall search are described in more detail in Sections III-A, III-B, III-C, III-D, respectively.

### A. Roadmap Construction over the 3D Environment

The roadmap is constructed over the 3D operational area, as defined by the bounding box \( \mathcal{W} \) and the bathymetry map. The roadmap is maintained as a graph \( \mathcal{G} = (\mathcal{V}_\text{RM}, \mathcal{E}_\text{RM}, \mathcal{D}_\text{RM}) \), where \( \mathcal{V}_\text{RM}, \mathcal{E}_\text{RM}, \mathcal{D}_\text{RM} \) denote the vertices, edges, and edge distances, respectively. An illustration is shown in Fig. 2. During the roadmap construction, a waypoint \( p_i \) is added to \( \mathcal{V}_\text{RM} \) for each \( R_i \in \mathcal{R} \). Random sampling is used to further populate the roadmap. In particular, a collision-free waypoint is generated by first sampling a position \((x, y)\) inside \( \mathcal{W} \). The maximum depth \( d_{\text{max}} \) at position \((x, y)\) is retrieved from the bathymetry map and a depth \( d \) generated uniformly at random from \([0, d_{\text{max}}]\). The waypoint \( p = (x, y, d) \) is added to \( \mathcal{V}_\text{RM} \) when it is at least a certain distance, \( d_{\text{clear}} \), away from the forbidden regions \( \mathcal{O} \) and the ocean floor. The clearance distance, which is set to twice the length of the AUV in the experiments, ensures that the AUV will maintain a separation from the obstacles when reaching the waypoints. Waypoints are repeatedly generated until \( \mathcal{V}_\text{RM} \) reaches a user-defined number of vertices.

To capture the connectivity of the free space, after generating the waypoints, attempts are made to connect neighboring waypoints with collision-free paths. For each waypoint \( p \in \mathcal{V}_\text{RM} \) its \( k \)-nearest neighbors from \( \mathcal{V}_\text{RM} \), denoted by \( \text{NEIGHS}(p) \), are computed using efficient nearest-neighbors
algorithms [26]. For each \( p_{\text{neigh}} \in \text{NEIGHS}(p) \), the edge \((p,p_{\text{neigh}})\) is added to \( E_{\text{RM}} \) when the straight-line segment from \( p \) to \( p_{\text{neigh}} \) is at least a certain distance, \( d_{\text{clear}} \), away from the forbidden regions and the ocean floor. The edge cost, denoted by \( D_{\text{RM}}(p,p_{\text{neigh}}) \), is defined as \( ||p - p_{\text{neigh}}|| \).

Since the objective is to reach as many targets as possible, it is important to ensure the connectivity of the waypoints \( \{p_1, \ldots, p_n\} \) associated with the target regions. For this reason, the process of generating and connecting waypoints is repeated until \( p_1, \ldots, p_n \) belong to the same graph component in \( RM \). The probabilistic completeness of PRM [23] ensures that, when there are collision-free paths connecting the waypoints, \( \{p_1, \ldots, p_n\} \) will eventually be connected.

B. Motion-Tree Partition

Starting from the initial state \( s_{\text{init}} \), the motion tree is incrementally expanded by generating collision-free and dynamically-feasible trajectories as branches. An illustration is shown in Fig. 2. The motion-tree expansion takes into account the AUV dynamics and the ocean currents. The motion tree is maintained as a directed acyclic graph, denoted by \( \mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}}) \). Each vertex \( v \in V_{\mathcal{T}} \) is labeled with a collision-free state, denoted by \( v.s \). Each edge \((v,v') \in E_{\mathcal{T}}\) is labeled with a pair \((u,f)\) which represents the motion from \( v.s \) to \( v'.s \), i.e., \( v'.s = \text{SIMULATE}(v.s,u,f,dt) \). Such motion is obtained by simulating the AUV dynamics starting at \( v.s \) and applying the control input \( u \) for one time step where \( f \) denotes the drift caused by the time-varying ocean currents.

Let \( \zeta_{\mathcal{T}}(v) \) denote the motion trajectory obtained as the sequence of states connecting the root of \( \mathcal{T} \) to the vertex \( v \in V_{\mathcal{T}} \). Each \( v \in V_{\mathcal{T}} \) keeps track of the time duration of \( \zeta_{\mathcal{T}}(v) \), denoted by \( v.t \). When \( v \) is added to \( V_{\mathcal{T}} \), \( v.t \) is updated by adding \( dt \) to the time duration associated with its parent. This information is used to determine the remaining time duration, i.e., \( H - v.t \), for any trajectory expanded from \( v \).

The motion tree is partitioned into equivalent groups based on the targets reached by the trajectories associated with the vertices and their nearest roadmap waypoints, where \( v \).targets = \text{TARGETS}(\zeta_{\mathcal{T}}(v)) \) and \( v.waypt = \arg\min_{p \in V_{\text{RM}}} ||p - \text{POSD}(v.s)||_2 \).

Vertices \( v, v' \in V_{\mathcal{T}} \) are said to belong to the same group if \( v.waypt = v'.waypt \) and \( v \).targets = \( v' \).targets. In this way, the group defined by a waypoint \( p \in V_{\text{RM}} \) and targets \( \mathcal{X} \subseteq \mathcal{R} \) includes all the tree vertices that map to \( p \) and whose trajectories have reached all and only the targets in \( \mathcal{X} \), i.e.,

\[
G((x,p)) = \{ v : v \in V_{\mathcal{T}} \land v \text{.targets} = \mathcal{X} \land v \text{.waypt} = p \}. \tag{7}
\]

An illustration is shown in Fig. 2. As a result, the motion tree \( \mathcal{T} \) can be partitioned into several non-empty groups, i.e.,

\[
\mathcal{G} = \{ G((x,p)) : \mathcal{X} \subseteq \mathcal{R} \land p \in V_{\text{RM}} \land |G((x,p))| > 0 \}. \tag{8}
\]

The partition is updated in practically constant time each time a vertex \( v \) is added to \( \mathcal{T} \) by representing \( \mathcal{G} \) as a hashmap indexed by \((\mathcal{X}, p)\). If the hashmap does not have \((\mathcal{X}, p)\), where \( \mathcal{X} = v \).targets and \( p = v \).waypt, then \( G((\mathcal{X}, p)) \) is created and added to \( \mathcal{G} \). The time \( v.t \), which represents the execution time to reach \( v \) from the root of \( \mathcal{T} \), is recorded as the creation time of \( G((\mathcal{X}, p)) \), and denoted by \( G((\mathcal{X}, p))\). As described next, the remaining time, \( H - G((\mathcal{X}, p))\), is used by the PC-TSP solver to compute a tour to reach the remaining targets \( \mathcal{R} \setminus \mathcal{X} \). If the hashmap contains \((\mathcal{X}, p)\), then \( G((\mathcal{X}, p)) \) is retrieved from \( \mathcal{G} \). After creating or retrieving \( G((\mathcal{X}, p)) \), \( v \) is added to \( G((\mathcal{X}, p)) \).

C. Bounded PC-TSP Tours

The partition \( \mathcal{G} \) and the navigation roadmap \( \mathcal{R} \) are used to guide the motion-tree expansion. A PC-TSP solver is invoked for each group \( G((\mathcal{X}, p)) \) to compute a tour that starts at \( p \) and reaches as many of the remaining regions, namely \( \mathcal{R} \setminus \mathcal{X} \), as possible without violating the time constraint, reducing the tour length, and minimizing the accrued penalty. As mentioned in Section II, the problem setting does not require the AUV to return to the initial position. As such, the PC-TSP solver computes open tours\(^1\). Specifically, let \( \pi \) denote a path over \( \mathcal{R} \), where \( \pi(i) \) denotes the \( i \)-th vertex in \( \pi \). Let \( \text{DISTANCE}(\pi) \) denote the length of \( \pi \), i.e.,

\[
\text{DISTANCE}(\pi) = \sum_{i=1}^{||\pi||-1} ||\pi(i) - \pi(i+1)||_2. \tag{9}
\]

Let \( \text{TARGETS}(\pi) \) denote the targets reached by \( \pi \), i.e.,

\[
\text{TARGETS}(\pi) = \{ \mathcal{R}_j : \mathcal{R}_j \in \mathcal{R} \land \exists i : \pi(i) = \mathcal{R}_j \}. \tag{10}
\]

The penalty accrued by \( \pi \) with respect to \( \mathcal{R} \setminus \mathcal{X} \) is given by

\[
\text{PENALTY}(\pi, \mathcal{R} \setminus \mathcal{X}) = \sum_{\mathcal{R}_i \in \{ \mathcal{R} \setminus \mathcal{X} \} \setminus \text{TARGETS}(\pi)} \text{PENALTY}(\mathcal{R}_i). \tag{11}
\]

PCTSPSolver(\( \mathcal{R} \), \( \mathcal{X} \), \( p \), \( \mathcal{L} \)) seeks to compute a path \( \pi \) with \( \pi(1) = p \) and \( \text{DISTANCE}(\pi) \leq \mathcal{L} \) which reduces \( \text{PENALTY}(\pi, \mathcal{R} \setminus \mathcal{X}) \) and \( \text{DISTANCE}(\pi) \). The upper bound \( \mathcal{L} \) reflects the maximum distance that the AUV could travel.

\(^1\)Open tours are computed by adding a virtual node and connecting it to the start with zero cost and to all the other nodes with a high cost (higher than the highest possible tour cost).
without exceeding the remaining execution time. For this reason, \( L \) is set to \( \nu(H - G(x,p),t) \), where \( \nu \) is the maximum AUV speed and \( G(x,p) \) is the time when \( G(x,p) \) was first created (Section III-B). The tour computed by the PC-TSP solver is stored as a field, denoted by \( G(x,p) \), in the data structure representing \( G(x,p) \). This paper uses the constraint-based PC-TSP solver provided by Google ORTOOLS [27].

D. Overall Search

After constructing the roadmap \( RM \) and rooting the motion tree \( T = (V_T, E_T) \) at the initial state \( s_{init} \), the approach proceeds iteratively by selecting a group \( G(x,p) \) from \( G \) and expanding \( T \) from \( G(x,p) \) along \( G(x,p) \). These procedures are invoked repeatedly until a solution is found or a runtime limit is reached.

1) Group Selection based on PC-TSP Tours: The group selection seeks to promote generation of trajectories that reduce the accrued penalty and the distance traveled. Priority is given to \( G(x,p) \) when its tour \( G(x,p) \) is short and has a small penalty. To avoid overexploitation, the selection also takes into account the number of times \( G(x,p) \) has been selected for expansion previously, denoted by \( G(x,p) \). These factors are combined into a weight defined as

\[
  w(G(x,p)) = \frac{2^\text{PENALTY}(G(x,p),\pi)|R(x)| G(x,p) \text{nsel}}{\text{DISTANCE}(G(x,p),\pi)}
\]

(12)

where \( 0 < \alpha < 1 \) (set to 0.95 in the experiments). The group with the maximum weight is selected for expansion, i.e.,

\[
  \arg\max_{G(x,p) \in G} w(G(x,p))
\]

(13)

2) Group Expansion along PC-TSP Tour: The group expansion seeks to expand \( T \) from vertices in \( G(x,p) \) along \( G(x,p) \). Specifically, a point \( q \) is first sampled from \( G(x,p) \), i.e., \( q \leftarrow G(x,p) \) where \( i \) is selected uniformly at random from 1 to \( |G(x,p)| \). The objective then becomes to expand \( T \) from the vertex \( v \in G(x,p) \) with \( v.t < H \) that is closest to \( q \). This is achieved by setting the desired heading and depth to point to \( q \) and using the PID controllers available in MOOS-IvP to steer the AUV from \( v \) to \( q \).

If the new state \( s_{new} \) computed after each simulation step, is in collision, then the group expansion terminates, and the planner goes back to the group selection. The group expansion also terminates if \( v.t + dt > H \) since \( s_{new} \) would violate the upper bound on the mission duration. Otherwise, a new vertex \( v_{new} \) and a new edge \( (v, v_{new}) \) are added to \( T \). The vertex \( v_{new} \) is associated with \( s_{new} \) and the time \( v_{new}.t \) is computed as \( v.t + dt \). If \( s_{new} \) has reached a target, it is added to the list of targets associated with \( v_{new} \), i.e.,

\[
  v_{new}.targets = v.targets \cup \text{TARGET}(s_{new})
\]

(14)

The nearest waypoint in the roadmap is also updated and stored as \( v_{new}.waypt \).

The motion-tree partition is also updated accordingly. Specifically, if \( G(x,p) \) \( \notin G \) where \( X = v_{new}.targets \) and \( p = v_{new}.waypt \), then \( G(x,p) \) is created and added to \( G \). When \( G(x,p) \) is first created, \( \text{PCTSPOOLVER}(RM, X, p, L) \) is also invoked to compute the tour associated with \( G(x,p) \). The upper bound \( L \) is set to \( \nu(H - v_{new}.t) \) to reflect the maximum distance that the AUV could travel for the remaining time (recall that \( \nu \) denotes the maximum speed). After creating or retrieving \( G(x,p) \), \( v_{new} \) is added to \( G(x,p) \).

The expansion continues from \( v_{new} \) as described above until \( || \text{POS}(v_{new},s) - q || \leq d_{near} \) or a maximum number of steps is reached. A tolerance \( d_{near} > 0 \) is used (set to 0.1m in the experiments) since constraints imposed by AUV dynamics and ocean currents may make it difficult to reach \( q \) exactly. The maximum number of steps is set to \( \beta || \text{POS}(v, s) || / \text{SPEED}(v,s) \), which reflects the expected number of steps based on the distance and speed, where \( v \) is the vertex from which the branch started. The parameter \( \beta \geq 1 \) (set to 1.1 in the experiments) is used to allow some more steps in case the AUV deviates from the expected path.

When the group expansion terminates, a new iteration starts. The group selection and expansion continues until a solution is found or a runtime limit is reached. By using roadmap tours to guide the motion-tree expansion, as shown by the experiments, the approach is able to effectively plan a collision-free and dynamically-feasible trajectory that reduces the accrued penalty and the distance traveled.

IV. EXPERIMENTS AND RESULTS

The approach is tested both in simulation and on a field experiment with the Reliant AUV. Efficiency and scalability is shown by varying the number of targets and the bound \( H \) on the mission duration. Comparisons with a greedy variant which does not use PC-TSP tours show the importance of combining motion planning with PC-TSP solvers.

A. Simulation Experiments

The first simulation scene, shown in Fig. 3, has an operational area of 1.28km \( \times \) 1.28km with a maximum depth of 60m. Obstacles are added to the scene to show the ability of the approach to compute collision-free trajectories. The first barrier parallel to the \( x \)-axis starts at the maximum depth and has a height of 40m, leaving only a gap of 20m at the top for the AUV to pass. The second barrier starts at depth 0 and goes down 40m, leaving a gap of 20m at the bottom. These barriers force the AUV to go up and down. Barriers parallel to the \( y \)-axis have small openings placed at different depths through which the AUV can pass.

The second scene, shown in Fig. 3, corresponds to the Chesapeake Bay. This is a larger scene with an operational area of 26.65km \( \times \) 33.31km and a maximum depth of 31m, characterized by complex bathymetry and time-varying ocean currents. Data for the bathymetry and ocean currents are obtained from the National Geophysical Data Center [19] and the Chesapeake Bay Operational Forecast System [20].

To test scalability, the number of targets is varied from 10 to 35 in increments of 5. For a given scene and a number \( n \) of targets, 60 problem instances are generated, denoted by \( T_{\text{scene},n} \). A problem instance is obtained by placing the \( n \) target regions at random positions and depths while ensuring a minimum separation distance. Specifically, each region corresponds to a box whose dimensions are sampled
uniformly at random between some minimum and maximum values. A position and depth is sampled for $R_i$ until it is not in collision with obstacles and it is a certain distance away from the previously placed regions $R_1, \ldots, R_{i-1}$.

Experiments were run on an Intel Core i7 (CPU: 3 GHz, RAM: 16GB) using MAC OSX Yosemite and CLANG 602.0.53. Runtime measures everything, including the roadmap construction. Results report mean values and standard deviations which are calculated after dropping the best and the worst five runs out of the sixty runs for each problem instance to avoid the influence of the outliers.

### B. Results

Fig. 4 shows the results on the mean runtime of the approach and the time duration of the solution trajectory when varying the number of targets. These results are obtained when $\mathcal{H} = \infty$, which corresponds to the case where the mission is to reach every target region. As shown in Fig. 4, the approach effectively solves even the problem instances with 35 targets. The trends indicate linear scalability.

Fig. 5 shows the results when keeping the number of targets to 35 but varying $\mathcal{H}$. Region penalties are assigned random values in $[0, 1]$. As shown in Fig. 5, the accrued penalty decreases as the runtime increases since the motion tree reaches more and more targets. The decrease in the accrued penalty continues for sometime and then stops. This is due to $\mathcal{H}$ which, in cases where it is small, makes it impossible to reach all the targets. In fact, as $\mathcal{H}$ is increased, the accrued penalty decreases as more and more targets are reached. When $\mathcal{H}$ is set to a large value, the accrued penalty becomes 0, which indicates that all the targets were reached. Fig. 5 also shows that the time duration of the solution trajectory increases with the runtime since more and more targets are reached. As in the case of the accrued penalty, the increase continues for sometime but then stops due to $\mathcal{H}$. Overall, the results indicate that the approach effectively reaches the targets while reducing the accrued penalty.

To show the importance of combining sampling-based motion planning with PC-TSP solvers, comparisons are carried out with a greedy variant of the approach. In the greedy version, the tour for each $G_{\mathcal{X}, \mathcal{P}}$ is computed by ordering the remaining regions $\mathcal{R} \setminus \mathcal{X}$ in decreasing order according to their penalties. Fig. 6 shows that the greedy version reaches considerably fewer regions for the same runtime as the original approach. This is due to the larger distances of the greedy tour since it does not attempt to reduce the distance traveled. For the accrued penalty, as a function of runtime, the greedy version initially results in larger reductions since it goes to the region with the highest penalty first. As the runtime increases, the original version is able to reduce the penalty more since it is able to plan trajectories that follow entire PC-TSP tours. When $\mathcal{H} = \infty$, the results show that the greedy version plans significantly longer trajectories than the original version.

### C. Field Experiment

A field experiment using the Reliant AUV (Fig. 7) was conducted in the Boston Harbor to simulate a Mine Coun-
termeasures mission. The AUV was given the mission of reacquiring data on potential mine-like targets for classification and identification. The target locations and penalties were generated at random onboard the AUV at the time of deployment. For the purposes of this experiment, to acquire data at a target, the AUV must pass within 50m of the target (this follows the paradigm of wireless sensor network data harvesting). The mission duration, $H$, was set to 42 minutes.

The planner developed in this paper was installed on an embedded processor (Intel Core i7 with 4GB of RAM) running onboard the AUV. The planner was implemented as a MOOS application designed to run along side the IVP helm. The planner monitors the MOOS database for the current state of the AUV and a list of targets to be reacquired. Once it receives the list of targets, the planner generates a collision-free and dynamically-feasible trajectory which starts at the current state and reaches as many targets as possible, within the remaining time limit, while reducing the accrued penalty.

The application we implemented also has the capability to re-plan by monitoring the overall execution of the trajectory. While the AUV executes the planned trajectory, if the position of the AUV deviates from the trajectory with 20% track error, it re-runs the planner to generate a new trajectory.

Fig. 5 shows the executed trajectory, which took 36 minutes. The AUV reached three targets with high penalties, which reduced the accrued penalty to only 7.7%. No other target could be reached within the remaining 6 minutes. The field experiment showed the potential of the planner to increase the capability of the AUV to conduct MCM missions. Due to operational constraints and costs, only one experiment was run to validate the approach at-sea.
Fig. 7. Field experiment with the Reliant AUV. Each circle represents a target, where the radius (50m) expresses the maximum distance from which the AUV can acquire data at the target. The penalty at each target is shown as a percentage of the sum of all the penalties. The execution time bound \( T \) was set to 42 minutes. The trajectory executed by the AUV is shown in blue. The targets that were reached are marked with-X.

V. DISCUSSION

The approach developed in this paper was shown to take into account the limited energy resources of the AUV and effectively plan a collision-free and dynamically-feasible trajectory that enabled the AUV to reach many of the targets while reducing the accrued penalty and the distance traveled. The effectiveness of the approach derived from a combination of sampling-based motion planning and PC-TSP solvers. Bounded tours computed by the PC-TSP solver guided the sampling-based expansion of a motion tree. One direction for future research is to improve the interplay between sampling-based motion planning and the PC-TSP solver. Another direction is to also incorporate high-level goal reasoners in order to adapt the mission based on new information discovered during the exploration. We also plan to investigate extending the approach to multiple AUVs.

ACKNOWLEDGMENT

The authors thank Dr. Benjamin Dzikowicz and Dr. Artur Wolek for assistance with the field experiments.

REFERENCES


