Long-horizon Robotic Search and Classification using Sampling-based Motion Planning

Geoffrey A. Hollinger
Robotics Program
School of Mechanical, Industrial & Manufacturing Engineering
Oregon State University, Corvallis, OR 97331, USA
Email: geoff.hollinger@oregonstate.edu

***Abstract*—This paper presents the Rapidly-exploring Adaptive Search and Classification (ReASC) algorithm, a sampling-based algorithm for planning the trajectories of mobile robots performing real-time target search and classification tasks in the field. The proposed algorithm incrementally builds up a tree of solutions and evaluates the utility of each solution for identifying targets in an environment. An optimistic approximation for the classification utility is used, which reduces the computational cost of evaluating trajectories and makes real-time adaptive planning feasible. The proposed algorithm is tested on an autonomous aquatic vehicle and are shown to outperform myopic methods by up to 36% in a lake monitoring scenario.**

I. Introduction

In a number of high-impact applications, including ocean monitoring [31], aerial surveillance [5], and emergency response [25], it is necessary to identify and classify a number of initially unknown targets. Autonomous vehicles are well suited for performing these search and classification tasks due to their ability to go where it may be dangerous or impossible for humans to go (e.g., underwater, disaster sites, and remote locations) and to process large quantities of sensor data. However, autonomous vehicles often have limited deployment time, particularly in aquatic and aerial scenarios, and they must return to refuel after a fixed mission time.

To effectively search for and classify targets with autonomous vehicles during their limited deployment time, it is beneficial to plan the trajectories of the vehicles to maximize the chance of correctly classifying the largest possible number of targets (see Figure 1). In addition, the vehicle can improve its pre-planned trajectory by adapting as new information is received (e.g., targets may be identified more quickly than expected or be more difficult to identify than expected).

The adaptive planning problem described above is computationally difficult to solve (typically NP-hard or even P-SPACE hard [28]). Many prior solutions have been limited to pre-planned trajectories [3] or require computation exponential in the size of the problem [30]. While a number of sampling-based motion planning algorithms have been successfully implemented on fielded systems [2, 12, 22], computational limitations have prohibited complex search and classification objectives from being optimized in real time in the field.

In this paper, we propose a sampling-based algorithm, the Rapidly-exploring Adaptive Search and Classification (ReASC) algorithm, that takes advantage of an optimistic approximation of the classification accuracy of the pre-planned path to make real-time field applications feasible. This approach allows the algorithm to incrementally build a tree of trajectories for long-horizon planning and improved classification. We implement the ReASC algorithm both in simulation and in experiments using an autonomous aquatic surface vehicle performing an aquatic monitoring task on a lake. The main novelty of this paper is the development of the ReASC algorithm that allows for long-horizon adaptive search and classification on real-world platforms operating in the field.

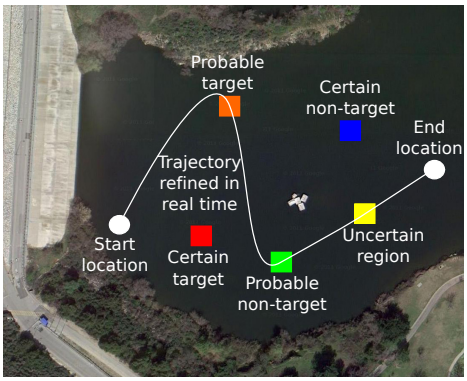


Fig. 1. Visualization of the adaptive search and classification task to identify uncertain targets in the environment (e.g., in surveillance and scientific data collection applications). Solving this problem requires gathering information about an area with high uncertainty in measurements. The proposed sampling-based algorithm allows for real-time long-horizon planning in the field.

The remainder of this paper is organized as follows. We first discuss related work in adaptive planning and classification to show the need for a scalable algorithm (Section II). Next, we provide a formulation of the classification and search problem (Section III). We then propose the ReASC incremental sampling-based algorithm (Section IV), present simulations showing its improved performance over greedy methods (Section V), and discuss experimental results on an autonomous surface vehicle performing an aquatic monitoring task in a lake. Finally, we provide concluding remarks and discuss avenues for future work (Section VII).

II. Related Work

The proposed algorithm draws on work in sampling-based motion planning (e.g., the RRT* algorithm [17] and

information-rich RRT [23]) and stochastic optimization [11] to design algorithms capable of adaptive behavior in classification scenarios. Unlike prior work in sampling-based motion planning, which typically deals with minimizing a cost function given local constraints on traversability, the proposed algorithm allows for optimizing a search and classification reward function given a constraint on budget.

General adaptive decision making problems have been recently studied in the context of stochastic optimization (e.g., optimization of objective functions that require an expectation over measurements to compute). Example problems that have been extended to allow for stochastic variants include covering [10], packing [7], and knapsack [8]. These prior works have developed algorithms with guarantees on performance in these domains. However, prior work has not demonstrated extensions that take into account the trajectory constraints inherent in robot motion planning.

The recent property of adaptive submodularity [11] has provided additional insight into stochastic optimization problems by combining submodular optimization [18] with stochastic objective functions. Algorithms that take advantage of this property typically require computation exponential in the size of the problem [30]. The proposed algorithm allows for the application of scalable sampling-based techniques to the optimization of stochastic objective functions.

Motion planning to optimize stochastic objectives is closely related to the problem of adaptive sampling [32] and Bayesian decision making [24]. In addition, the proposed work is connected to research in active mapping problems [9]. Prior work in these areas has focused on modeling the stochastic objective and then employing greedy strategy for optimization. Our work demonstrates that greedy algorithms often perform poorly in the domains of interest.

Stochastic optimization problems can also be formulated using the POMDP architecture [19]. While the POMDP provides a baseline for optimal performance in these domains, such general solvers are notoriously difficult to scale when facing large problem domains and long-horizon planning. Sampling-based planners have previously been applied to related domains in persistent monitoring [21] and motion planning under uncertainty [4]. Alternative planners that operate in the high-dimensional belief space have also been proposed [1, 16]. However, these prior algorithms are not directly applicable to general search and classification objectives.

In our own prior work, we have examined the problems of autonomous inspection [14] and bathymetric mapping [15] using sampling-based planners. More recently, we developed a general Rapidly-exploring Information Gathering (RIG) algorithm [13] for maximizing information objectives with mobile robots. The algorithm proposed in the current paper is based on similar principles as RIG and RRT*, but it improves on the existing algorithms' capabilities by allowing for real-time optimization of search and classification objective functions. Thus, the proposed ReASC algorithm fills an important gap that is not currently covered by existing algorithms: real-time long-horizon optimization for search and classification on fielded systems.

III. Problem Formulation

The goal is to locate and classify some number of targets in the environment (e.g., biological phenomena in an environmental monitoring scenario or targets of interest in a surveillance scenario). For each discrete location x in the environment $\mathcal{X}$, there is some probability $P(x)$ of a target being in that location. An autonomous vehicle is equipped with a sensor capable of determining the true classification of all targets in a region surrounding its location. We denote a positive realization for a random variable as x and a negative realization as $\neg x$.

We assume there is some known false negative rate $r_{neg} = P(\neg Y \mid x)$ and false positive rate $r_{pos} = P(Y \mid \neg x)$. The vehicle is constrained by a budget denoted as B (e.g., time, fuel, or energy). We also assume that the total number of targets is unknown, and we maintain a probability distribution over each discrete location in the environment. We assume that the probabilities of a target existing in different locations are independent; however, extensions to dependencies between targets is an interesting possible extension.

We denote the space of possible trajectories for an autonomous vehicle as Ψ, and the cost (relative to the budget) of executing each trajectory $\mathcal{P} \in \Psi$ as $c(\mathcal{P})$. For the experiments in this paper, we parametrize a trajectory by discrete waypoints; however, extensions of the proposed algorithm to continuous space are straightforward (see Section IV-D). The vehicle receives measurements about the target's location, which we denote as $Y \in \mathcal{Y}$, where $\mathcal{Y}$ is the space off all possible measurements (i.e., true or false for all possible locations in the environment).

At a time t and at each location x, we maintain an estimated probability $P(x)$ that a target exists in that location and conversely a probability $P(\neg x)$ that no target exists in that location. Given a reward for each successful classification R_s, a penalty for each unsuccessful classification R_f, we will ultimately choose to classify each location x as a target if:

$$R_s P(x \mid Y) \geq R_f P(\neg x \mid Y). \tag{1}$$

Let $\mathcal{X}_+ \subseteq \mathcal{X}$ be the subset of the space that we choose to classify as a location containing a target. Based on this decision choice and a space of possible measurements $\mathcal{Y}$, we can formulate the expected reward function as:

$$R(\mathcal{P}) = \mathbb{E}_{Y \subseteq \mathcal{Y}} \left[\sum_{x \in \mathcal{X}_+} R_s P(x \mid Y) - R_f P(\neg x \mid Y) \right]. \tag{2}$$

For a trajectory $\mathcal{P}$, we can calculate the posterior probability $P(x \mid Y)$ for a given set of measurements Y using a standard Bayesian update if we assume the prior $P(x)$, the false negative rate r_{neg}, and false positive rate r_{pos} are known. Note that calculating $R(\mathcal{P})$ exactly would require iterating over the space of all possible measurements, which would scale exponentially in the budget (i.e., the tree of all possible

measurements would expand with branching factor two). In Section IV-C we propose an optimistic approximation of expected reward to make this calculation feasible.

Given a way of approximating the expected reward $R(\mathcal{P})$, the goal is to optimize this reward given the budget and trajectory constraints:

$$\mathcal{P}^* = \underset{\mathcal{P}\in\Psi}{\operatorname{argmax}} \; R(\mathcal{P}) \text{ s.t. } c(\mathcal{P}) \leq B, \tag{3}$$

where Ψ is the space of possible trajectories, B is a budget (e.g., time, fuel, or energy), and $R(\mathcal{P})$ is the expected reward after executing trajectory $\mathcal{P}$. This problem falls within the broader class of informative path planning problems, which are all at least NP-hard [30]. Thus, we focus our effort on scalable approximations that take advantage of sampling to make the required computation feasible.

IV. Adaptive Search and Classification Algorithm

We now describe the proposed ReASC sampling-based algorithm for planning the trajectories of mobile robots performing adaptive classification tasks. The key idea is to sample the space of possible locations and build up a tree of possible trajectories that maximize the expected reward. The algorithm is inspired by the RIG [13] and RRT* [17] algorithms, which are designed to optimize objectives that are deterministic functions of the robot's trajectory. To apply such techniques to search and classification objectives, it is necessary to (1) provide adaptive in-the-loop behavior as the belief distribution is updated, and (2) approximate the expected reward of a trajectory in an efficient manner. We now describe how we fulfill these requirements.

A. Algorithm Outer Loop

The outer loop of the ReASC algorithm is described in Algorithm 1. The outer loop begins with the full budget and iterates until the robot has expended its entire budget. A step size Δ is specified by the user to determine how often the vehicle should run the underlying planner. The user also specifies the workspace $\mathcal{X}$, the traversable space $\mathcal{X}_{free}$, the initial belief $P(x)$, and the starting configuration $\mathbf{x}_{start}$ of the vehicle. Finally, an internal parameter r for the sampling-based planner is needed to determine how many connections will be made in the trajectory tree (see Section IV-B).

At each iteration, the vehicle runs a sampling-based planner (Alg 1, Line 7) to generate a reward maximizing trajectory. The trajectory is encoded in a tree of trajectories as described below. After the reward maximizing trajectory is extracted (Alg 1, Line 8), the vehicle executes the first step in this trajectory (Alg 1, Line 10). Once the vehicle has reached its new location, it receives a new measurement (Alg 1, Line 11) and updates the target belief distribution based on this new information (Alg 1, line 12) using a Bayesian update. Finally, the trajectory tree is updated (Alg 1, Line 13) such that all solutions stemming from the previous configuration are removed (see Section IV-B), and the expected reward at each node is updated to match the new belief distribution (see Section IV-C). The next iteration then repeats this process.

Algorithm 1 Rapidly-exploring Adaptive Search and Classification (ReASC)

1: **Input:** Step size Δ, Budget B, Workspace $\mathcal{X}$, Free space $\mathcal{X}_{free}$, Initial belief $P(x)$, Start configuration $\mathbf{x}_{start}$, Near radius r
2: % Initialize budget, starting point, and tree
3: $R_{init} \leftarrow 0$, $C_{init} \leftarrow 0$, $n \leftarrow \langle \mathbf{x}_{start}, C_{init}, R_{init} \rangle$
4: $B_{cur} \leftarrow B$, $\mathbf{x}_{cur} \leftarrow \mathbf{x}_{start}$, $\mathcal{T} \leftarrow (n, \emptyset)$
5: **while** $B_{cur} > \Delta$ **do**
6: % Run planner to determine next step
7: $\mathcal{T} = PlanStep(\Delta, B_{cur}, \mathcal{X}, \mathcal{X}_{free}, P(x), \mathbf{x}_{cur}, r)$
8: $\mathcal{P} \leftarrow ExtractMaxRewardTrajectory(\mathcal{T})$
9: % Execute step, receive measurement, update belief
10: $\mathbf{x}_{cur} \leftarrow ExecuteTrajectoryStep(\mathcal{P}, \mathbf{x}_{cur})$
11: $Y \leftarrow ReceiveMeasurement(\mathbf{x}_{cur})$
12: $P(x) \leftarrow UpdateBelief(Y, P(x))$
13: $\mathcal{T} \leftarrow UpdateTree(\mathcal{T}, P(x), \mathbf{x}_{cur})$
14: $B_{cur} = B_{cur} - \Delta$
15: **end while**
16: % Classify targets and receive reward
17: $R(\mathcal{P}) \leftarrow FindExpectedReward(P(x))$

After the budget has been expended, the vehicle makes the final decision on which locations to classify as target and non-target using Equation 1. The final expected reward is then calculated by simply dropping the expectation over measurements in Equation 2 and using the updated belief. If the algorithm has successfully optimized the objective, the resulting expected reward will be higher than those produced by competing methods.

B. Sampling-based Re-planning

The core of the ReASC algorithm uses a sampling-based motion planner to determine the next action for the robot to take (Alg 1, Line 7). This core *PlanStep* function is described in Algorithm 2. The algorithm begins by initializing the node and edge list using the tree from the previous iteration (Alg 2, Line 4). Samples are then taken of the workspace (Alg 2, Line 7), and the nearest node in the tree is determined (Alg 2, Line 8). We note that samples can be taken using a uniform distribution or using some informed distribution that biases towards areas of higher expected reward. The development of a more informed sampling distribution is an interesting avenue for future work.

Once the nearest point to the new sample is identified, a feasible point in the configuration is identified by running the *Steer* function [17] that extends the existing point in the tree towards the new point. All points within some user-specified radius r are then retrieved (Alg 2, Line 11) and also extended towards the feasible point (Alg 2, Line 14). Nearest neighbor checking can be performed using a KD-tree, which allows for efficient computation in large trees. Each connection is checked for collision (Alg 2, Line 15), and then

Algorithm 2 Function: *PlanStep*

1: **Input:** Step size Δ, Budget B, Workspace $\mathcal{X}$, Free space $\mathcal{X}_{free}$, Belief $P(x)$, Start configuration $\mathbf{x}_{start}$, Near radius r, Initial tree $\mathcal{T}_i(V_{init}, E_{init})$
2: **Output:** Final tree $\mathcal{T}_f$
3: % Initialize node list, edge list, and closed list
4: $V \leftarrow V_{init}, V_{closed} \leftarrow \emptyset$, $E \leftarrow E_{init}$
5: **while** not terminated **do**
6: % Sample vehicle conf. space and find nearest node
7: $\mathbf{x}_{samp} \leftarrow Sample(\mathcal{X})$
8: $n_{nearest} \leftarrow Nearest(\mathbf{x}_{samp}, V \setminus V_{closed})$
9: $\mathbf{x}_{feasible} \leftarrow Steer(\mathbf{x}_{n_{nearest}}, \mathbf{x}_{samp}, \Delta)$
10: % Find near points to be extended
11: $N_{near} \leftarrow Near(\mathbf{x}_{feasible}, V \setminus V_{closed}, r)$
12: **for** all $n_{near} \in N_{near}$ **do**
13: % Extend towards new point
14: $\mathbf{x}_{new} \leftarrow Steer(\mathbf{x}_{n_{near}}, \mathbf{x}_{feasible}, \Delta)$
15: **if** $NoCollision(\mathbf{x}_{n_{near}}, \mathbf{x}_{new}, \mathcal{X}_{free})$ **then**
16: % Calculate new reward and cost
17: $R_{new} \leftarrow EstimateReward(R_{n_{near}}, \mathbf{x}_{new}, P(x))$
18: $c(\mathbf{x}_{new}) \leftarrow EvaluateCost(\mathbf{x}_{n_{near}}, \mathbf{x}_{new})$
19: $C_{new} \leftarrow C_{n_{near}} + c(\mathbf{x}_{new})$
20: $n_{new} \leftarrow \langle \mathbf{x}_{new}, C_{new}, R_{new} \rangle$
21: **if** PRUNE(n_{new}) **then**
22: Delete n_{new}
23: **else**
24: % Add edges and node
25: $E \leftarrow E \cup \{(n_{near}, n_{new}\}$, $V \leftarrow V \cup \{n_{new}\}$
26: % Add to closed list if budget exceeded
27: **if** $C_{new} > B$ **then**
28: $V_{closed} \leftarrow V_{closed} \cup \{n_{new}\}$
29: **end if**
30: **end if**
31: **end if**
32: **end for**
33: **end while**
34: % Extract trajectory tree
35: $\mathcal{T}_f = (V, E)$

the expected reward (Alg 2, Line 17) and cost (Alg 2, Line 18) are estimated for the new trajectory. The new trajectory is generated by taking the new point and traversing backwards down the tree back to the starting point. We note that the cost here represents the amount of budget used and, unlike in many motion planning problems, is not being directly optimized (i.e., expected reward is being optimized). As described above, it is not straightforward to calculate this expected reward efficiently because it requires iterating over an exponential number of possible observations. We describe an approximation method below in Section IV-C.

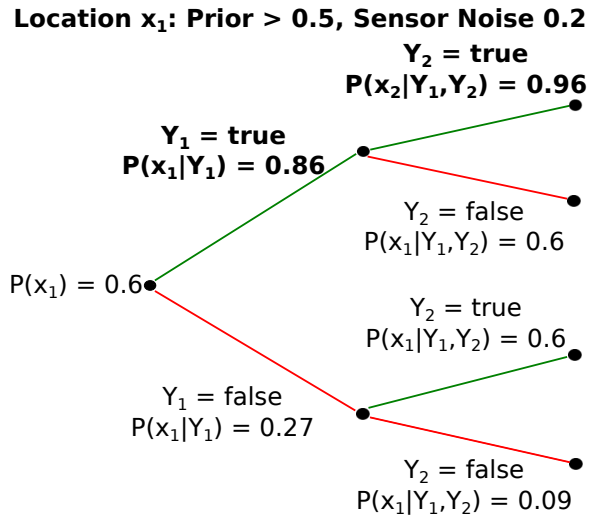
Once a new node in the tree is generated, we check if an existing node already exists nearby with lower cost and higher expected reward. If such a node exists, the new node is pruned because it is unlikely to be included in the final trajectory (Alg 2, Line 21). For alternative methods of pruning, including those that lead to asymptotic optimality, readers are directed to our prior work [13]. If a node is not pruned, it may also have cost exceeding the budget, in which case it is added to the "closed list" of nodes that should no longer be extended (Alg 2, Line 28). If the node is valid, it is connected to its parent node by updating the edge list (Alg 2, Line 25). The resulting tree formed by these nodes and edges encodes a rich space of trajectories for optimizing the expected reward function. The maximum reward trajectory can easily be extracted by looking at expected reward on the leafs of the tree and tracing the trajectory back to the root.

Once the motion planner is run, and the vehicle has taken a step in the environment, the trajectory tree does not need to be thrown away. Instead, it is straightforward to prune away all nodes that stem from the original vehicle configuration that are no longer valid. To do this, we simply set the updated tree to the subtree beginning at the new vehicle location $\mathbf{x}_{cur}$. In addition, the expected reward estimates on all nodes need to be reset and recalculated using the updated belief. This operation only requires a single traversal of the tree if the optimistic approximation of expected reward is used (see Section IV-C), and thus only requires $O(N)$ computation. This operation provides the *UpdateTree* function in Line 12 of Algorithm 1.

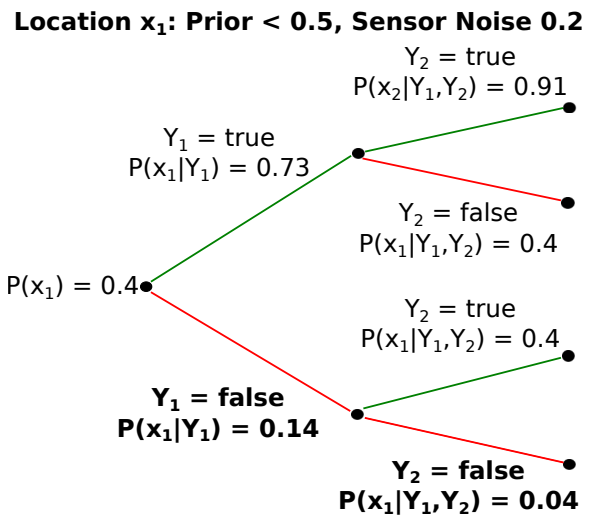
C. Optimistic Approximation of Expected Reward

A key step in executing the ReASC algorithm is efficiently estimating the expected reward for a trajectory (Alg 2, Line 17). When a new node is generated, the algorithm must reconstruct the trajectory $\mathcal{P}$ back to the root of the tree by traversing the edge list. This trajectory is then used to estimate the value of $R(\mathcal{P})$, which is then stored with the node. The exact solution to $R(\mathcal{P})$ (shown in Equation 2) requires calculating an expectation over all possible measurements. Such an expectation would require enumerating over an exponentially increasing number of measurement outcomes as the length of the trajectory (i.e., the nodes along the path back to the root) increases. To see this, note that at each point in the trajectory, the vehicle can receive either a false or true measurement of the target location, which creates a branching factor of two. Thus, the exact solution would require up to $O(2^L)$, where L is the number of nodes tracing back to the root. As the density of the tree and the length of the trajectories increases, this computation will quickly grow infeasible.

Since our goal is to generate long-horizon trajectories, we need an approximation strategy for the expected reward that reduces the necessary computation. We employ an optimistic approximation of the expected reward that provides an a practical approximation (similar to [26]). This upper bound is found by choosing only *optimistic* measurements that move the target beliefs towards higher certainty about the locations of the targets. If the vehicle moves to a location x where the probability of a target $P(x) > 0.5$, we assume that the vehicle will receive a positive measurement. The posterior probability



(a) Prior above 0.5 (assume positive measurements)



(b) Prior below 0.5 (assume negative measurements)

Fig. 2. Visualization of the optimistic approximation of expected reward. The proposed approximation method chooses the emboldened path through the measurement tree, which provides an optimistic estimate of the final uncertainty. This method reduces the required computation from exponential to linear in the number of nodes in the trajectory.

$P(x \mid Y)$ is then calculated based on a Bayesian update from the corresponding measurement:

$$P(x \mid Y) = \eta P(x) P(Y \mid x), \tag{4}$$

where η is a normalizing constant. Conversely, if $P(x) < 0.5$, we assume that the vehicle will receive a negative measurement:

$$P(x \mid \neg Y) = \eta P(x) P(\neg Y \mid x). \tag{5}$$

By applying this approximation to all nodes in the trajectory, we can calculate an estimate of the expected reward gained by trajectory $\mathcal{P}$ in $O(L)$ time. Letting $\mathcal{X}_{\mathcal{P}+} \subseteq \mathcal{X}$ be the subset of locations visited by $\mathcal{P}$ where the posterior $P(x \mid Y)$ is above the decision threshold, we have an estimate of the increase in expected reward $\bar{R}$ for trajectory $\mathcal{P}$:

$$\bar{R}(\mathcal{P}) = \sum_{x \in \mathcal{X}_{\mathcal{P}+}} R_s P(x \mid Y) - R_f P(\neg x \mid Y). \tag{6}$$

This approximation does not require looking at the exponential number of possible of outcomes (see Figure 2 for a visualization). The intuitive justification of this approximation is that it effectively estimates the expected reduction of probability mass around the high uncertainty situation where $P(x) = 0.5$. In the case of $P(x) = 0.5$, the vehicle is forced to make a risky decision that could result in an high penalty. Conversely, cases where $P(x) \approx 0$ or $P(x) \approx 1$, the vehicle is making a near-certain decision, which will result in high likelihood of reward and low likelihood of penalty. In the following section, we demonstrate that this method provides improved performance over greedy methods in both simulations and experiments.

D. Discrete and Continuous Variants

The general formulation of the ReASC algorithm is capable of operating in continuous space, similar to other sampling-based motion planning algorithms. However, it is important to note that the discrete planning problem is also challenging in search and classification domains (i.e., still at least NP-hard [29]). This is an important distinction from the classical shortest path problem, which can be solved efficiently using Dijkstra's algorithm or A* search. For search and classification objectives, it is not possible to use A* heuristics to reduce the search space. This is due to the dependence of the reward on the prior trajectory. In contrast, the ReASC algorithm uses the optimistic approximation to estimate the reward on each node and can be applied to objectives with dependencies on the prior trajectory.

In some scenarios it may be beneficial to plan in a discrete space (e.g., when the possible locations of the vehicle are limited by environmental conditions). In this case, a discrete variant of the ReASC algorithm can be formulated by restricting the nodes on the tree to discrete locations. When a sample is taken, nodes on the tree are extended towards the new node. In the discrete variant, these extended nodes then have their locations rounded to the nearest discrete point. The discrete variant also has the advantage that the tree will not become overly dense and lead to a blowup in computational cost.

V. Simulations and Experiments

A. Data-driven Simulations

We now discuss simulations and experiments validating the proposed ReASC search and classification algorithm. The simulations were run on a single desktop with a 3.2 GHz Intel i7 processor with 9 GB of RAM. The ReASC algorithm was implemented in C++ on Ubuntu Linux. Nearest neighbor

queries were provided by the Open Motion Planning Library (OMPL) [6]. The algorithm was terminated after taking 1000 samples, which led to completion in approximately 0.01 seconds.

The simulations use depth and temperature data from a portion of Puddingstone Lake in San Dimas, CA (Lat. 34.088854°, Lon. -117.810667°). Measurements of depth and temperature were collected using an autonomous aquatic vehicle (described in Section VI). The vehicle ran a 45 minute survey of the lake, and linear interpolation was used to generate a temperature and depth map for the entire area of interest (see Figure 5). The vehicle's goal was to identify target regions within a pre-specified depth and temperature. Locating areas within a pre-specified interval has direct relevance to scientific research objectives (e.g., organisms of interest typically live in areas of certain depth and temperature) and surveillance applications (e.g., locating submarines at pre-specified depths).

For these experiments, the target depth was set to $d = 7 \pm 1$ meters, and the target temperature was set to $T = 9.6 \pm 1$ degrees C. The area of interest was divided into a 10×10 discrete map of cells, and a total of 16 of these cells fit within the temperature and depth threshold. Thus, performance is determined by how many correct identifications the vehicle can make (16 being perfect). One unit of reward is given for a correct target classification, and one unit is subtracted for an incorrect classification.

An initial probability map was determined based on the 45 minute survey (shown in Figure 5). Areas that were not observed by the survey were estimated using linear interpolation. Each of the discrete areas was then assigned a probability of being within the threshold based on an exponential decay from the target threshold. The resulting probability map provided an initial estimate for $P(x)$.

A simulated autonomous vehicle was tasked with moving through the area of interest and refining the initial probability map. The vehicle received simulated measurements of the target depth and temperature. We set false positive and false negative rates to different values to model a sensors with varying noise levels. The proposed ReASC algorithm was compared to (1) a random method that moves to each adjacent cell with uniform probability, and (2) a greedy method that calculates the exact 1-step reward and moves to the cell with the highest reward [32].

The results in Figure 3 demonstrate that the ReASC algorithm outperforms the greedy and random methods for varying budgets and false positive/negative rates. The advantage over greedy is particularly pronounced with larger budgets because the greedy method becomes trapped in a local maximal reward and is unable to escape to areas of higher reward. In such cases, as much as a 36% improvement can be seen with the proposed methods. These simulations demonstrate the advantage of long-horizon planning using sampling-based methods with reward approximation.

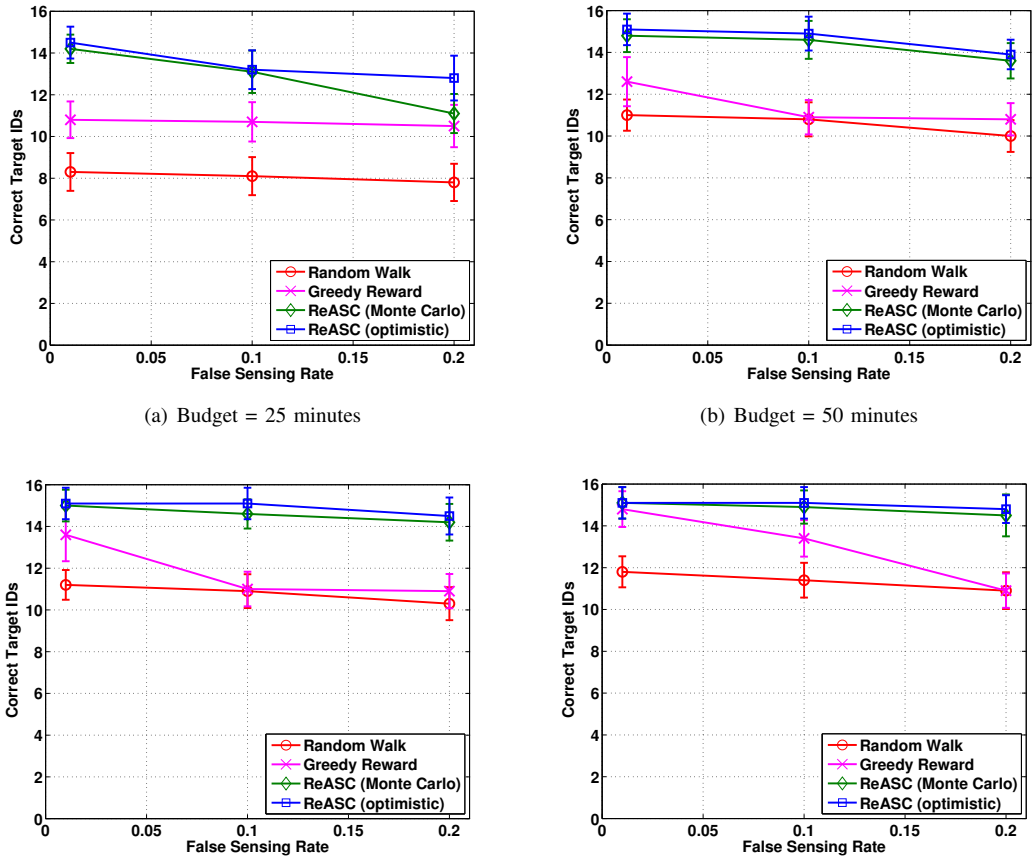
To decouple the benefit of long-horizon planning from the accuracy of the approximation method, we also compare to a variant of ReASC that uses Monte Carlo sampling to estimate the expected reward. The Monte Carlo method simulates K possible measurement combinations along a candidate trajectory and estimates the expected reward by averaging rewards over these measurement configurations. This method provides an estimate of expected reward that becomes more accurate with increasing running time. In these simulations, we set $K = 200$, which yielded an average replanning time of 0.98 seconds (approximately the upper limit on what would be reasonable replanning time in the aquatic monitoring scenario).[1] In contrast, ReASC with the optimistic approximation had an average replanning time of 0.01 seconds. Figure 3 shows that the optimistic approximation slightly outperforms the Monte Carlo approximation, which demonstrates the effectiveness of the proposed approximation approach.

VI. Experiments with Autonomous Aquatic Surface Vehicle

We now demonstrate our proposed approach using a lake monitoring scenario with an autonomous aquatic surface vehicle. The Ecomapper vehicle (www.ysi.com/ecomapper) shown in Figure 4 is propeller driven and moves at speeds up to 2 knots. It is equipped with a GPS unit and a Doppler Velocity Log (DVL), which provide localization and depth sensing capabilities for the vehicle. The vehicle communicates with a ground station through a standard 802.11 wireless connection. The vehicle measures the temperature using a thermometer and the depth using the DVL at its current location. The Ecomapper is capable of diving to depths of 100 meters; however, in these experiments, it acted as a surface vehicle to provide safer testing of the proposed algorithm. The vehicle houses a front-end computer running Windows and a back-end Linux computer with a 1.6 GHz Intel Atom processor and 1 GB of RAM. The back-end computer ran the proposed algorithm onboard in real time. Experiments were conducted at Puddingstone Lake in San Dimas, CA.

Implementing the proposed algorithm required interfacing with the vehicle's front-end and back-end computers and translating the waypoints given by the algorithm to command actions for the propellers. The back-end computer, running Robot Operating System (ROS) [27], received measurements from the sensors (using a Python parser) and ran the ReASC planner (implemented in C++). The ReASC planner on the back-end computer generated waypoints and sent them via ethernet to the front-end computer. The front-end computer, packaged standard with the YSI Ecomapper vehicle, used YSI's proprietary waypoint following algorithm to execute the waypoints. The YSI waypoint following algorithm adjusts for winds and currents that would otherwise force the vehicle to deviate from its specified trajectory. This system architecture allows ReASC to be integrated with existing control software operating in real-time on the aquatic vehicle.

[1] We note that an exhaustive method for calculating the expected reward is not feasible because the total number of possible observation combinations for observing N points would be 2^N. Typically, greater than 25 points are observed along a candidate trajectory in the lake.



(a) Budget = 25 minutes

(b) Budget = 50 minutes

(c) Budget = 75 minutes

(d) Budget = 100 minutes

Fig. 3. Simulation results for the proposed algorithm compared to a myopic (greedy) heuristic and a random walk method. The proposed ReASC sampling-based algorithm consistently outperforms the baseline algorithms for varying budgets and false sensing rates. Variants of ReASC are shown with a Monte Carlo reward approximation method and with the proposed optimistic reward approximation method. Error bars are one SEM.

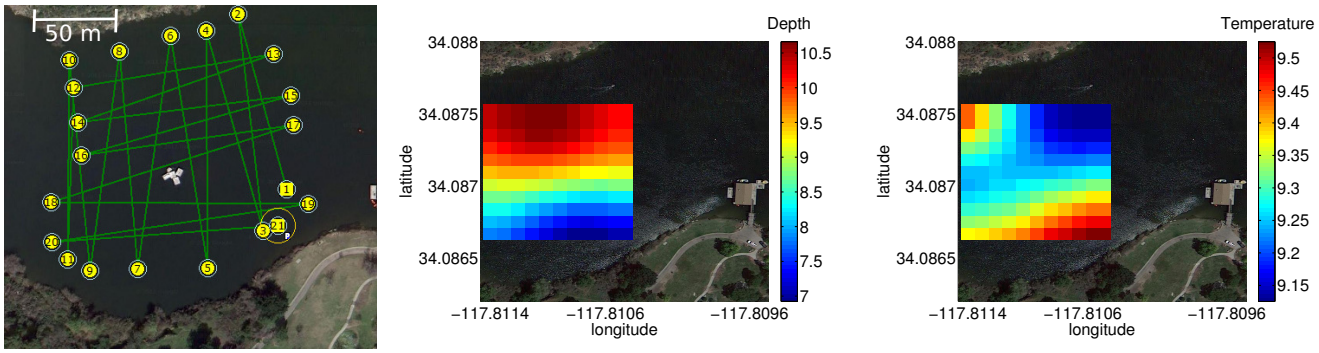
Fig. 4. Ecomapper propeller-driven AUV used as a surface vehicle for lake monitoring experiments with the proposed ReASC adaptive classification algorithm.

As in the simulated experiments, an initial probability map was generated, and the vehicle was tasked with refining that map. In these experiments, we only use the depth target of $d = 7 \pm 1$ meters due to better reproducibility on the vehicle. The initial and final probability maps, along with the trajectory executed by the vehicle, is shown in Figure 6. The vehicle trajectory has some circular turns, which is an artifact of the waypoint following controller.

The initial probability map predicted that 38 regions were within the target depth range. After executing the 10 minute mission on the vehicle, 10 of those target regions were found by the sensor to be outside the target range. The remaining 28 regions were determined by the sensor to be within the target range. With a relatively short mission (compared to the 45 minute survey), the proposed algorithm was able to determine that 26% of the locations predicted to be within the target range by the survey were likely erroneous. This proof-of-concept experiment demonstrates the proposed ReASC algorithm running onboard an autonomous aquatic vehicle using existing hardware.

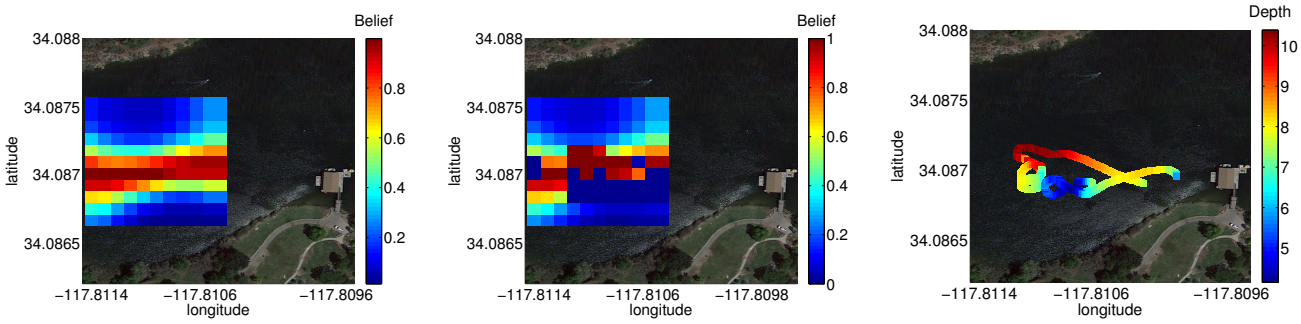
VII. Conclusions and Future Work

This paper has shown that it is possible to implement long-horizon planning for search and classification tasks in real time on a fielded aquatic vehicle. The proposed ReASC algorithm combines motion planning techniques with an optimistic approximation strategy for expected reward to allow for real-time optimization of search and classification objective functions.



(a) Initial Survey (b) Interpolated Depth (c) Interpolated Temperature

Fig. 5. Initial 45 minute survey performed by an autonomous aquatic vehicle (left). The vehicle measured depth and temperature at its current location. The depth (center) and temperature (right) maps are interpolated for areas that were not directly observed.



(a) Initial Belief (b) Final belief (c) Executed Trajectory

Fig. 6. Probability map generated from the initial survey (left), refined probability map after executing a 10 minute mission (center), and vehicle trajectory executed by the ReASC algorithm (right). The vehicle is able to identify that 10 of the locations predicted by the initial survey to be within the target range are outside this range (based on the observed sensor data). Image is best viewed in color.

The resulting techniques were validated both in simulation and on an autonomous surface vehicle performing a target classification task in a lake.

There are a number of lessons learned from the experiments. It is clear that the computation on existing aquatic vehicles is sufficient to run sampling-based planners. One issue that remains is translating waypoints in 2D or 3D space into control actions for the vehicle. A number of existing waypoint controllers are not designed for densely packed waypoints, and additional work is needed to ensure that the trajectories are executed in an efficient manner (e.g., do not contain unnecessary loops). Another important observation from the experiments is that testing becomes challenging when the vehicle is performing adaptive behaviors. In some cases, the vehicle did not properly execute the trajectory, and the reasons were difficult to debug. Methods for verification and validation of controllers that include replanning would help to diagnose these problems before they occur.

The results in this paper open up a number of interesting areas for algorithmic future work. It would be worthwhile to explore the applicability of state-of-the-art POMDP solvers that utilize sampling-based methods (e.g., [20]) to the domains of interest. We showed that naive Monte Carlo estimation methods do not perform competitively relative to running time, but more sophisticated approaches may yield improved results. Of theoretical interest is an analysis of the asymptotic optimality of the proposed approaches. Algorithms like RRT* [17] and RIG [13] are asymptotically optimal given conservative pruning strategies. To show similar properties for the ReASC algorithm, it would be necessary to bound the approximation accuracy of the optimistic strategy.

Experimental future work includes implementation of the proposed algorithm on autonomous underwater vehicles. A particularly interesting scenario to examine is the use of the proposed algorithm for underwater inspection and intervention in marine structures. Overall, the proposed techniques have opened the door to scalable, long-term planning in domains with high uncertainty.

ACKNOWLEDGEMENTS

The author gratefully acknowledges Gaurav Sukhatme from the University of Southern California for providing access to the autonomous aquatic vehicle used in experiments. This research has been funded in part by the following grant from the Office of Naval Research Science of Autonomy program: ONR-N00014-14-1-0509.

References

[1] A. Agha-mohammadi, S. Chakravorty, and N. M. Amato. FIRM: Sampling-based feedback motion planning under motion uncertainty and imperfect measurements. *Int. J. Robotics Research*, 33(2):268–304, Feb. 2014.

[2] A. Bachrach, S. Prentice, R. He, and N. Roy. RANGE - robust autonomous navigation in GPS-denied environments. *J. Field Robotics*, 28(5):646–666, Sept. 2011.

[3] J. Binney and G. S. Sukhatme. Branch and bound for informative path planning. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 2147–2154, May 2012.

[4] A. Bry and N. Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 723–730, May 2011.

[5] M. Bryson, A. Reid, F. Ramos, and S. Sukkarieh. Airborne vision-based mapping and classification of large farmland environments. *J. Field Robotics*, 27(5):632–655, Sept. 2010.

[6] I. A. Şucan, M. Moll, and L. E. Kavraki. The open motion planning library. *IEEE Robotics & Automation Mag.*, 19(4):72–82, Dec. 2012.

[7] B. Dean, M. Goemans, and J. Vondrak. Adaptivity and approximation for stochastic packing problems. In *Symp. Discrete Algorithms*, pages 395–404, Jan. 2005.

[8] B. Dean, M. Goemans, and J. Vondrak. Approximating the stochastic knapsack: the benefit of adaptivity. *Mathematics of Operations Research*, 33(4):945–964, Nov. 2008.

[9] H. Feder, J. J. Leonard, and C. M. Smith. Adaptive mobile robot navigation and mapping. *Int. J. Robotics Research*, 18(7):650–668, July 1999.

[10] M. Goemans and J. Vondrak. Stochastic covering and adaptivity. In *Latin American Symp. Theoretical Informatics*, pages 532–543, Mar. 2006.

[11] D. Golovin and A. Krause. Adaptive submodularity: A new approach to active learning with stochastic optimization. In *Proc. Conf. Learning Theory*, pages 333–345, June 2010.

[12] E. Hernandez, M. Carreras, and P. Ridao. A path planning algorithm for an AUV guided with homotopy classes. In *Proc. Int. Conf. Automated Planning and Scheduling*, pages 82–89, June 2011.

[13] G. Hollinger and G. Sukhatme. Sampling-based robotic information gathering algorithms. *Int. J. Robotics Robotics*, 33(9):1271–1287, Aug. 2014.

[14] G. Hollinger, S. Singh, J. Djugash, and A. Kehagias. Efficient multi-robot search for a moving target. *Int. J. Robotics Research*, 28(2):201–219, Feb. 2009.

[15] G. Hollinger, U. Mitra, and G. Sukhatme. Active and adaptive dive planning for dense bathymetric mapping. In *Proc. Int. Symp. Experimental Robotics*, pages 803–817, June 2012.

[16] L. P. Kaelbling and T. Lozano-Perez. Integrated task and motion planning in belief space. *Int. J. Robotics Research*, 32(9–10):1194–1227, Aug. 2013.

[17] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Robotics Research*, 30(7):846–894, June 2011.

[18] A. Krause and C. Guestrin. Submodularity and its applications in optimized information gathering. *ACM Trans. Intelligent Systems and Technology*, 2(4), July 2011.

[19] H. Kurniawati, D. Hsu, and W. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proc. Robotics: Science and Systems Conf.*, Zurich, Switzerland, June 2008.

[20] H. Kurniawati, Y. Du, D. Hsu, and W. Lee. Motion planning under uncertainty for robotic tasks with long time horizons. *Int. J. Robotics Research*, 30(3):308–323, Mar. 2011.

[21] X. Lan and M. Schwager. Planning periodic persistent monitoring trajectories for sensing robots in gaussian random fields. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 2407–2412, May 2013.

[22] J. Leonard et al. A perception-driven autonomous urban vehicle. *J. Field Robotics*, 25(10):727–774, Oct. 2008.

[23] D S. Levine. Information-rich path planning under general constraints using rapidly-exploring random trees. Master's thesis, Massachusetts Institute of Technology, June 2010.

[24] K. H. Low, J. M. Dolan, and P. Khosla. Information-theoretic approach to efficient adaptive path planning for mobile robotic environmental sensing. In *Proc. Int. Conf. Automated Planning and Scheduling*, pages 233–240, Sept. 2009.

[25] R. R. Murphy. *Disaster robotics*. MIT Press, 2014.

[26] R. Platt, R. Tedrake, L. Kaelbling, and T. Lozano-Perez. Belief space planning assuming maximum likelihood observations. In *Proc. Robotics: Science and Systems Conf.*, Zaragoza, Spain, June 2010.

[27] M. Quigley et al. ROS: an open-source Robot Operating System. In *Proc. ICRA Workshop on Open Source Software*, Kobe, Japan, May 2009.

[28] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proc. IEEE Symp. Foundations of Computer Science*, pages 421–427, Oct. 1979.

[29] A. Singh, A. Krause, C. Guestrin, and W. Kaiser. Efficient informative sensing using multiple robots. *J. Artificial Intelligence Research*, 34:707–755, Apr. 2009.

[30] A. Singh, A. Krause, and W. Kaiser. Nonmyopic adaptive informative path planning for multiple robots. In *Proc. Int. Joint Conf. Artificial Intelligence*, Pasadena, CA, July 2009.

[31] R. N. Smith et al. USC CINAPS builds bridges: Observing and monitoring the Southern California Bight. *IEEE Robotics and Automation Mag.*, 17(1):20–30, Mar 2010.

[32] D. Thompson and D. Wettergreen. Intelligent maps for autonomous kilometer-scale science survey. In *Proc. Int. Symp. Artificial Intelligence, Robotics and Automation in Space*, Los Angeles, CA, Feb. 2008.