

# Informative Path Planning for an Autonomous Underwater Vehicle

Jonathan Binney, Andreas Krause and Gaurav S. Sukhatme

**Abstract**—We present a path planning method for autonomous underwater vehicles in order to maximize mutual information. We adapt a method previously used for surface vehicles, and extend it to deal with the unique characteristics of underwater vehicles. We show how to generate near-optimal paths while ensuring that the vehicle stays out of high-traffic areas during predesignated time intervals. In our objective function we explicitly account for the fact that underwater vehicles typically take measurements while moving, and that they do not have the ability to communicate until they resurface. We present field results from ocean trials on planning paths for a specific AUV, an underwater glider.

## I. INTRODUCTION

### A. Informative Path Planning

We are interested in the problem of collecting data about a scalar field, within a certain area. In particular, we care about collecting data about coastal areas of the ocean *e.g.*, temperature, salinity, and chlorophyll content of the water. Ideally, we would like to have high spatial resolution measurements over the entire area that we care about.

One simple approach would be to deploy static sensors through the area of interest, and use them to collect data. Unfortunately, this would require a huge number of sensors to achieve reasonable spatial resolution over a large area. Instead, we can take advantage of one or several autonomous underwater vehicles (AUVs), which can move throughout the area, taking samples in many locations. In this way we can trade some temporal resolution for higher spatial resolution at a relatively low cost.

The mobility of an AUV requires us to make decisions about where to sample. There are an endless number of possible paths which we could choose for the AUV. Ignoring depth for the moment, the simplest, and most often utilized choice in ocean sampling would be a “lawnmower” pattern, which would take the AUV back and forth across the region of interest in parallel, evenly spaced legs. While this would give us a large amount of information, the amount of time and energy required to execute such a path is often prohibitive.

Jonathan Binney (binney@usc.edu) and Gaurav S. Sukhatme (gaurav@usc.edu) are with the Robotic Embedded Systems Laboratory and Department of Computer Science at the University of Southern California (USC). Andreas Krause is with the Computer Science Department at Caltech. This work was supported in part by the NOAA MERHAB program under grant NA05NOS4781228 and by NSF as part of the Center for Embedded Network Sensing (CENS) under grant CCR-0120778, by NSF grants CNS-0520305 and CNS-0540420, by the ONR MURI program (grants N00014-09-1-1031 and N00014-08-1-0693) by the ONR SoA program and a gift from the Okawa Foundation. Jonathan Binney was supported by a doctoral fellowship from USC. This research was partially supported by ONR grant N00014-09-1-1044 and a gift from Microsoft Corporation.

Often we have some data about the area of interest before we deploy our AUV. This data may come from other AUVs that have already traversed the region, from remote sensing data, or from a small set of static sensors in the area. We would like to use this data to choose a new path for the AUV which gives us the most additional information. This is called the *informative* path planning problem.

There has been a significant amount of work done recently on the general problem of informative path planning for mobile robots, including [1], [2], [3], and [4]. Each of these approaches perform path planning by maximizing or minimizing an objective function related to the quantity being measured.

Using AUVs in coastal areas, however, creates additional issues which must be dealt with in order to plan practical paths. First, there is the problem of boat traffic. Our experimental AUVs surface periodically to check their position using GPS, and to transmit data. While they are at the surface, these expensive vehicles can be damaged in collisions with boats. To minimize this danger we must keep the AUVs out of high traffic areas during certain times of day. Second, our AUVs take samples while moving underwater, but typically do not change direction underwater. This means that when choosing a path for the AUV, the set of possible waypoints along the path is not the same as the set of sample locations along that path.

The contribution of this work is to extend a known path planning algorithm, recursive greedy [5], to work well with an AUV. Specifically, we have modified the algorithm to handle samples taken as the robot moves. We also explain how to use the time-window concept from recursive greedy to keep the AUV out of high traffic areas at certain times.

### B. Gliders

The specific AUV which we use most often is the Slocum Glider [6], which can be seen in Fig. 1. The underwater glider is an extremely low power underwater vehicle. By modifying its ballast, it alternately descends and ascends through the water. Small wings on either side give it forward motion as it does this, causing it to “glide” through the water. Although this method of locomotion is quite slow (on the order of 0.4 m/s), it allows the glider to operate at sea continuously for weeks at a time.

The path of the glider is a sawtooth pattern, moving up and down in the water as it travels horizontally between two waypoints. Instead of parametrizing the actual path and solving for the optimal trajectory in three dimensions, we ignore depth and solve a two dimensional problem. We use only the measurements from a specific slice of depths; for



Fig. 1. The Slocum Glider, resting at the surface before diving.

example measurements taken between a depth of 10 meters and a depth of 15 meters.

We work with the gliders as part of USC CINAPS [7]. Other researchers in CINAPS are working on related problems, such as tracking Harmful Algal Blooms (HABs) [8], and improving trajectory design using ocean models [9].

## II. PROBLEM DESCRIPTION

We describe the possible mission plans for the glider as a graph  $G = E, V$ , where each node  $v \in V$  is a possible waypoint for the glider, and each edge  $e \in E$  is one possible leg of the glider's path. A possible path for the glider is then described as a list  $P = [v_1, v_2, \dots, v_k]$  of nodes, in the order in which they would be visited. We wish to choose the glider's path such that it maximizes a function  $f(P)$  of the path.

The goal of the path planning is to model a scalar field such as temperature or salinity, represented by a random variable  $X$  at each location in the graph. Following [10], for  $f(P)$  we use the mutual information,  $I(\cdot)$ , between the sampled and unsampled locations. This can be defined in terms of the reduction in entropy,  $H(\cdot)$ , of the unsampled locations after taking the new samples into account,

$$I(X_{V_P}; X_{V-V_P}) = H(X_{V-V_P}) - H(X_{V-V_P} | X_{V_P}). \quad (1)$$

Here  $X_{V_P}$  are the random variables for the locations at which we collect samples along the path, and  $X_{V-V_P}$  are the variables for locations where we do not collect samples.

In order to calculate the entropy, we model the underlying scalar field using a Gaussian process [11]. In a Gaussian process, given samples at a set of points  $X_{V_P}$  we can calculate the covariance and mean at another set of points  $X_{V-V_P}$ . The entropy can then be calculated using the covariance.

In a Gaussian process, the kernel function used to create the kernel matrices is parametrized by hyper-parameters, which are typically learned from the data. Because of this, it is useful to have some pilot data that can be used to make an initial estimate of the hyper-parameters. In the simple kernel function which we use, the hyper parameters describe how quickly the function varies spatially.

### A. Submodularity

In order to allow an efficient solution, in this work we assume that the function  $f(P)$  is submodular. Submodularity

expresses the idea of diminishing returns: a new sample provides less information once a large number of samples have been taken. Mutual information is shown to be submodular in [12]. For a set function to be submodular, it must satisfy

$$f(A \cup \{s\}) - f(A) \geq f(B \cup \{s\}) - f(B), \quad (2)$$

for all sets  $A \subseteq B \subseteq V$  and elements  $s \in V - B$  [10]. We do not give more detail here, but [13] has a good explanation of why to use mutual information in sensor placement problems.

## III. ALGORITHM

### A. Recursive Greedy

We make use of the recursive-greedy algorithm for the submodular orienteering problem presented in [5]. It provides a logarithmic approximation guarantee with much better running time than brute force methods. The running time is  $O((2nB)^i \cdot T_f)$  where  $n$  is the number of possible waypoints,  $B$  is the maximum length for the path,  $i$  is the number of recursions which increases logarithmically with the length of the optimal path, and  $T_f$  is the maximum time required to evaluate the objective function (mutual information in our case).

The algorithm works recursively. When looking for a path from node  $s$  to node  $t$ , it splits the problem in half, and tries all possible middle points on the path. For each possible middle point, it tries all possible lengths  $b$  for the first half (thereby leaving  $B - b$  for the second half.) The algorithm commits to the best solution it can find for the first half of the path before starting the second half, which is why it is only an approximation. Still, it provides a solution with a reward of at least  $f_x(P^*) / [1 + \log k]$ , where  $f_x(P^*)$  is the reward obtained by the optimal solution and  $k$  is the length of the optimal path, as shown in [5].

Because the algorithm works by enumerating all possible values for the length allocated to the first half of the path, it requires  $B$  to be an integer value. The quantities which make sense to use for path length, distance and time, are actually continuous. This turns out to be only a small issue however. If we want to be able to represent path lengths from 0.0 to 10.0 with a resolution of 0.1, for instance, we can simply multiply all lengths by 10 and use a value of 100 for  $B$ . This illustrates the trade off that we face: increasing  $B$  allows better resolution, but increases running time.

To handle this problem, [5] also presents a way to use bounds on the reward function to make the running time depend only logarithmically on  $B$ . Although we did not need to do this for the results in this paper, it may be useful when looking at much longer paths.

### B. Avoiding High Traffic Areas Using Time Windows

Because we operate our AUVs in a coastal area, we must take care to avoid boat traffic. Specifically, we need to keep the AUV out of busy areas during times of day when boat traffic is high, because otherwise it could be run over while at the surface. This includes areas near marinas where boats

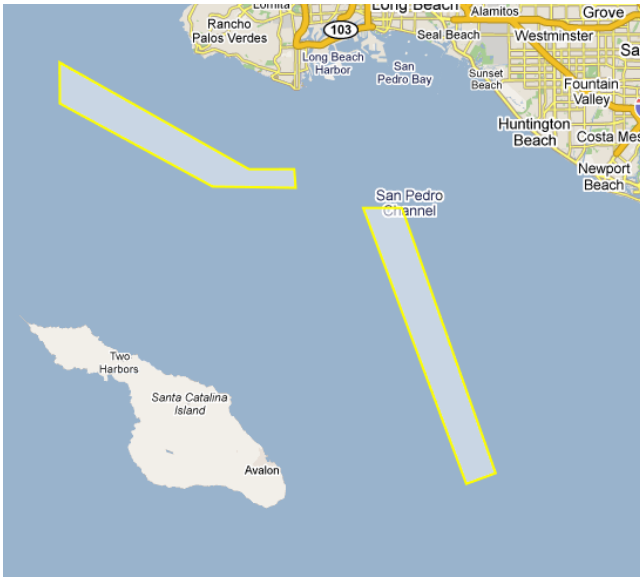


Fig. 2. Shipping lanes between Santa Catalina island and Los Angeles.

often go in and out during the day, as well as shipping lanes. Fig. 2 shows the two major shipping lanes between Santa Catalina island and Los Angeles in the Southern California Bight (SCB)<sup>1</sup>, the area in which we typically operate.

In [5], a time-window variant of the recursive greedy algorithm is also discussed. Adding time windows requires checking to make sure that each time the path is split, the middle node is visited within some allowed time window. As shown in [5], the same approximation guarantee and running time hold for the time window variant. We use these time windows to keep the glider out of dangerous areas during high traffic times.

### C. Edge based objective

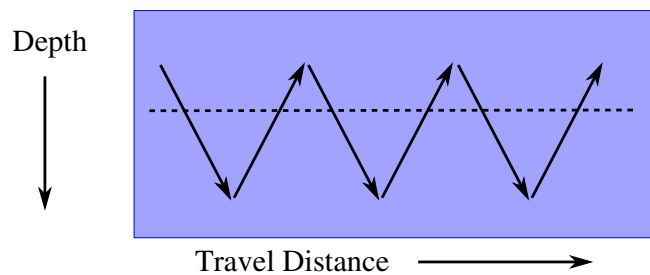


Fig. 3. An example of a glider trajectory through the water. It glides forward and downward, then glides back up. The dotted line shows one depth slice; the intersection of this line with the glider path gives the set of points at that depth that the glider would obtain.

The glider is moving up and down in the water in a sawtooth pattern as it moves between each waypoint, and so takes samples at many different depths, as can be seen in Fig. 3. If we restrict ourselves to only a thin slice of depths, such as the ones in the shaded box, then the set

<sup>1</sup>The SCB is the oceanic region contained within 32° N to 34.5° N and -117° E to -121° E

of samples are taken at approximately evenly spaced points along a straight line. If we place candidate waypoints in a regular grid over the area in which the glider is operating, and add edges between nearby waypoints, then we get a graph like the one in Fig. 4(a). We then place a set of evenly spaced points on each edge, approximating the positions of the samples that the glider would take as it moved between those two waypoints, as shown in Fig. 4(b).

In order to ensure that the original proof of the approximation guarantee in [5] still holds when using these edge points, we construct a new graph  $G'$ . Graph  $G'$  has all the nodes of  $G$ , plus one node for each point that we added along the edges. The graph  $G'$  in Fig. 4(c) corresponds to the graph  $G$  shown in Fig. 4(a) and 4(b).

Normally, adding so many extra nodes to the graph would greatly increase the running time of the recursive-greedy algorithm. In this case, however, we note that since the glider cannot actually change course at any of these points along each edge, they do not increase the number of possible paths. Because of this we are able to run the recursive greedy algorithm using only the nodes from the original graph  $G$ , but using  $G'$  to determine the sample points that need to be used to evaluate the objective function for each path.

When we calculate the objective function  $f(P)$  during recursive-greedy, instead of using the reward that would be obtained from sampling at each node on the path, we use the reward that would be obtained by sampling at the points along each edge traveled in the path. Because this does not change the number of times which  $f(P)$  is evaluated, the only term that is affected in the running time is  $T_f$ , the maximum time to evaluate the objective function for a path.

---

#### Algorithm 1 Algorithm: RG-EB( $s, t, \sigma, \tau, X, i$ )

---

```

if  $\tau < R(t)$  or  $l(s, t) + \sigma > D(t)$  then return Infeasible
 $m \leftarrow 0$ 
 $P \leftarrow s, t$ 
if  $i = 0$  then return  $P$ 
for  $v \in V$  do
  for  $\sigma < b < \tau$  do
     $P_1 \leftarrow \text{RG-EB}(s, v, \sigma, \sigma + b, X, i - 1)$ 
     $V_1 \leftarrow \text{expand-edges}(P_1, G, G')$ 
     $P_2 \leftarrow \text{RG-EB}(v, t, \sigma + b, \tau, X \cup V_1, i - 1)$ 
     $V_2 \leftarrow \text{expand-edges}(P_2, G, G')$ 
    if  $f_X(V_1 \cup V_2) > m$  then
       $P \leftarrow P_1 \cdot P_2$ 
       $m \leftarrow f_X(V_1 \cdot V_2)$ 
    end if
  end for
end for
return  $P$ 

```

---

The pseudocode for our algorithm is given in algorithm 1. The algorithm is based on the recursive-greedy algorithm with time windows, but has been modified to use an objective function that is based on the edges, instead of the vertices. To plan a path, the function *RG-EB* is called, with  $s$  as the initial waypoint of the AUV,  $t$  as the desired end waypoint,

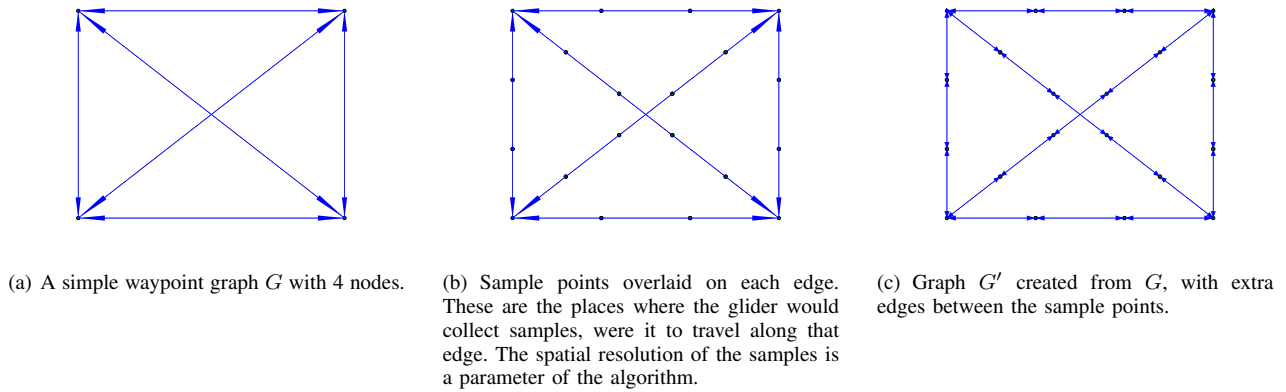


Fig. 4.

$\sigma$  as the start time,  $\tau$  as the end time,  $X$  as the set of pilot measurements, and  $i$  as the maximum level of recursion.

The algorithm searches for a path on the graph  $G$ , but each time it needs to evaluate the objective function on a path  $P$ , it first calls the function `expand_edges( $P, G, G'$ )`, which takes a path on  $G$  and returns all of the corresponding vertices which would be visited in the graph  $G'$ .

#### IV. RESULTS

To see whether this algorithm can realistically be used with the glider, we tested it on field data from an ocean deployment in the SCB. We start by giving the glider several static waypoints to visit. We then model the sensed variable in a rectangular area of interest using a Gaussian process, incorporating the samples already taken. The data points collected for pilot data are plotted in Fig. 5(a).

In a typical usage scenario, we might deploy the glider during the day and let it run for 24 hours to collect pilot data. The following day, we can run the path planning algorithm, and send the new set of waypoints to the glider. The glider can then follow the new set of waypoints for the next 24 hours.

For the results discussed here, we model the water conductivity. Normally this quantity is used to calculate the salinity of the water, but because conductivity is what we actually get from the sensor, we model it directly using the Gaussian process. We note that for oceanographic purposes, conductivity (and salinity) may not vary in interesting ways in the area which we have chosen. For testing purposes, we care only that it is a quantity for which the variations are greater than the sensor noise. In the future, we would like the path planning technique to be used in conjunction with oceanographers. In that usage scenario, the path planning would be combined with a user interface, so that a scientist could specify a sensed variable and an area of interest, along with high-traffic areas, and paths could be generated automatically.

Because this work is about our path planning algorithm, we do not provide an analysis of how well a Gaussian process models water conductivity in the ocean. We treat the objective function as a black box, and use Gaussian

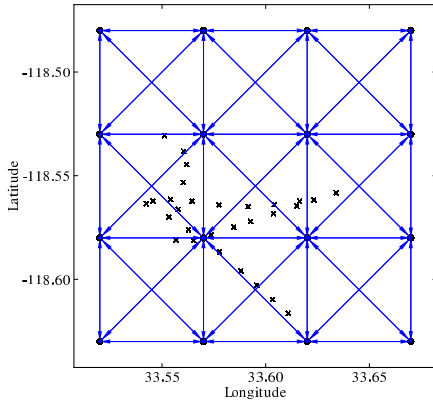
processes because they are simple and fairly well understood. Any other submodular objective function could be used; for example the mutual information between variables in some more complicated model.

We give the path planning algorithm a regular grid of possible waypoints, with possible edges between neighboring waypoints. Then we use our modified recursive greedy algorithm to find the best path from the glider's current position to one of the corners of the area of interest. We give it a budget  $B$  to work with, which is a limit on the length of the path. This budget can be interpreted as the amount of time that we want the path to take. We could also choose the start and end points to be the same, so that the glider returns to its original position.

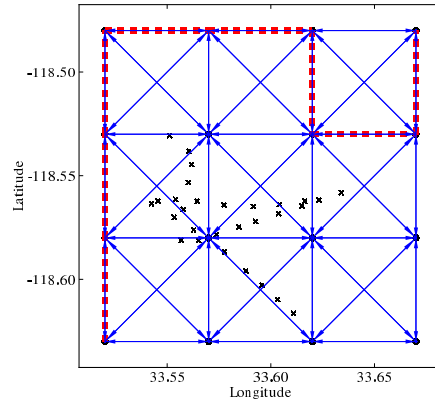
The resulting path can be seen in Fig. 5(b). Intuitively, the glider tries to choose a path that visits regions which have not yet been sampled, since these areas will give the most information. After reaching the desired end node, we have a number of choices. We could move on to another area, or we could add the measurements taken to the original pilot measurements, and use these as the pilot measurements to plan a new path for the region, reducing the uncertainty further.

In order to get a better idea of the effect that the path planning has on the uncertainty in the field being measured, we plot the variance. Fig. 5(c) shows the variance of the field when only the pilot measurements are used. Dark areas represent higher variance, and light areas represent lower variance. As expected, the variance is much lower near places where samples have been taken. It increases fairly quickly in areas farther from samples. The hyper-parameters of the kernel of the Gaussian process determine how quickly it increases.

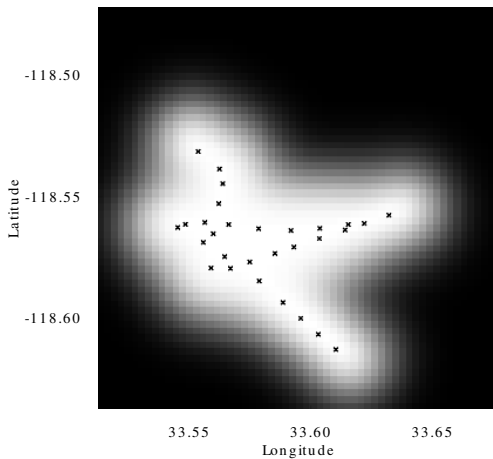
In Fig. 5(d) we can see the effect of adding samples from the planned path. Many of the areas for which the covariance was high after incorporating the pilot measurements were sampled by the chosen path. Because the maximum path length is limited, some areas are still missed. This could be improved by extending the path length, or by incorporating the path measurements into the pilot data, planning a new



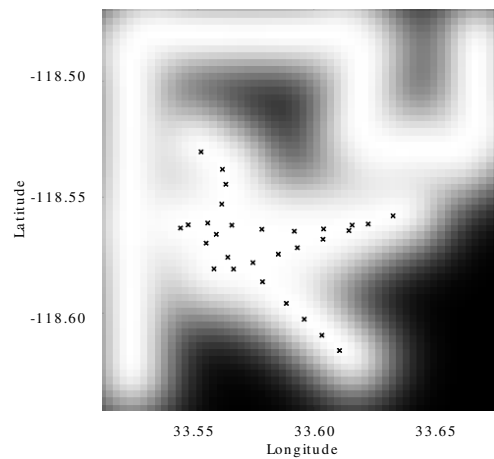
(a) The pilot data collected before path planning. Each 'x' represents one sample. Here the pilot data was collected using an initial run of the glider, but we could also use data from static buoys or remote sensing data.



(b) Recursive-greedy solution



(c) Covariance with only the pilot data



(d) Covariance after samples from chosen path are incorporated

Fig. 5.

path, and taking more samples.

Next we show the results for the case where we create a hypothetical “high-traffic” area for the glider to avoid during peak traffic hours. The shaded region in Fig. 6 is the area that the glider must avoid. In the case where we give the glider a new path to follow each day, we often do not want it to visit waypoints in high traffic areas until nighttime.

In the resulting path in Fig. 6 the glider waits to enter the high-traffic area, continuing to sample in the safe area. Then it enters the high traffic area, collects data there “at night”, and exits the area before the end of the safe time window.

## V. CONCLUSIONS AND FUTURE WORK

We have presented a algorithm for path planning for underwater gliders which maximizes mutual information. Our algorithm adapts and extends a previous method based on the recursive greedy algorithm. In addition to generating near-optimal paths, the algorithm ensures that the vehicle stays out of high-traffic areas during predesignated time

intervals. Results from trials with a single glider at sea are encouraging.

While the recursive greedy algorithm is considerably more efficient than exhaustive search, it still takes a long time to get results on medium or large sized datasets. Because the glider typically goes long distances between waypoints, we are able to use a rough discretization of the space of possible waypoints, but computational time still limits the scalability of our approach. In order to scale up further, we could take advantage of the fact that the values at locations separated by large distances are almost entirely uncorrelated, as was done in [4]

In this work we assume that the quantity which we are measuring is time-invariant; this is only reasonable over short periods of time. The recursive greedy technique could be extended to work with time-varying quantities by adding many copies of each node in the graph, one for each possible time that it could be visited. This would greatly increase the running time of the algorithm however, so another approach



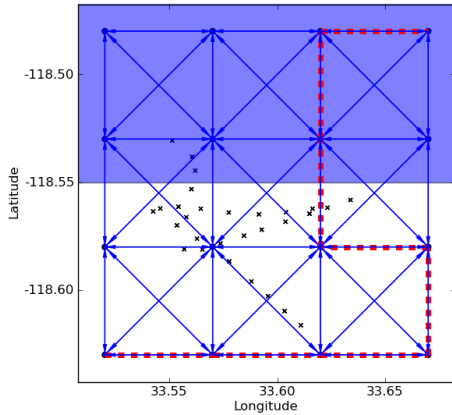


Fig. 6. Recursive-greedy solution with time windows.

is needed.

We reported on path planning for one glider here, but a technique that builds on the recursive greedy algorithm to iteratively do path planning for multiple robots can still have good approximation guarantees, as shown in [4]

#### A. Distance metric

One other assumption that we make is that the cost of traveling between nodes is proportional to the distance between them. In reality, because the glider moves so slowly, ocean currents can play a large role in how long it takes to travel between two points. Fortunately, there is a computational model of the currents in the Southern California area called the Regional Ocean Modeling System (ROMS) which gives estimated currents for the areas in which we are working [14], [15].

ROMS gives us the ability to get an estimated current velocity for each location in a grid. In the future we would like to use these values to calculate the expected cost (in time) of traveling between two waypoints, and use these costs as edge lengths when constructing the waypoint graph.

#### B. Communication

When it surfaces at each waypoint, the glider can send sensor data back to shore using either its satellite phone or using its Freewave radio. The satellite phone works from almost everywhere, but is slow and very expensive. The Freewave radio communicates directly with base stations on shore, and so is relatively cheap and fast, but has limited range.

Ideally, we would like to get data from the glider every time that it surfaces. Unfortunately, since sending so much data using the satellite phone would be prohibitively expensive, and the glider must sometimes take readings in areas which have limited radio connectivity with the shore, we must find a compromise. In the future, we would like to be able to incorporate communication constraints into the path planning algorithm; for example by requiring that data be transmitted back to a base station within 24 hours of it

being taken. This would force the glider to periodically seek out areas with good radio connectivity.

#### C. Implementation

All of the source code for this work is available online at [16]. The implementation is written in Python using the Numpy library, except for the creation of the Gaussian process kernel function, which is written in C for efficiency reasons.

#### REFERENCES

- [1] K. H. Low, J. M. Dolan, and P. Khosla, "Adaptive multi-robot wide-area exploration and mapping," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 1*, (Estoril, Portugal), pp. 23–30, International Foundation for Autonomous Agents and Multiagent Systems, 2008.
- [2] B. Zhang and G. S. Sukhatme, "Adaptive sampling for estimating a scalar field using a robotic boat and a sensor network," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, (Rome, Italy), pp. 3673–3680, 2007.
- [3] N. Yilmaz, C. Evangelinos, P. Lermusiaux, and N. Patrikalakis, "Path planning of autonomous underwater vehicles for adaptive sampling using mixed integer linear programming," *IEEE Journal of Oceanic Engineering*, vol. 33, no. 4, pp. 522–537, 2008.
- [4] A. Singh, A. Krause, C. Guestrin, and W. Kaiser, "Efficient informative sensing using multiple robots," *Journal of Artificial Intelligence Research*, vol. 34, pp. 707–755, 2009.
- [5] C. Chekuri and M. Pal, "A recursive greedy algorithm for walks in directed graphs," in *46th Annual IEEE Symposium on Foundations of Computer Science, 2005.*, pp. 245–253, 2005.
- [6] "Webb research corporation - slocum glider." <http://www.webbresearch.com/slocumglider.aspx>.
- [7] R. N. Smith, J. Das, H. Heidarsson, A. A. Pereira, F. Arrichiello, I. Cetinic, L. Darjany, M. Garneau, M. D. Howard, C. Oberg, M. Ragan, E. Seubert, E. C. Smith, B. Stauffer, A. Schnetzer, G. Toro-Farmer, D. A. Caron, B. H. Jones, and G. S. Sukhatme, "USC CINAPS builds bridges: Observing and monitoring the southern california bight," *IEEE Robotics and Automation Magazine*, Mar. 2010. Special Issue on Marine Robotic Systems.
- [8] J. Das, K. Rajan, S. Frolov, J. Ryan, F. Py, D. A. Caron, and G. S. Sukhatme, "Towards marine bloom trajectory prediction for AUV mission planning," in *IEEE International Conference on Robotics and Automation*, (Anchorage, USA), 2010.
- [9] R. N. Smith, A. A. Pereira, Y. Chao, P. P. Li, D. A. Caron, B. H. Jones, and G. S. Sukhatme, "Autonomous underwater vehicle trajectory design coupled with predictive ocean models: A case study," in *IEEE International Conference on Robotics and Automation*, (Anchorage, USA), 2010.
- [10] A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg, "Near-optimal sensor placements: maximizing information while minimizing communication cost," in *The Fifth International Conference on Information Processing in Sensor Networks*, (Nashville, USA), pp. 2–10, 2006.
- [11] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. the MIT Press, 2006.
- [12] A. Krause, A. Singh, and C. Guestrin, "Near-Optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies," *J. Mach. Learn. Res.*, vol. 9, pp. 235–284, 2008.
- [13] A. Krause and C. Guestrin, "Near-optimal observation selection using submodular functions," in *Proc. of 22nd Conference on Artificial Intelligence*, 2007.
- [14] A. F. Shchepetkin and J. C. McWilliams, "The regional oceanic modeling system (ROMS): a split-explicit, free-surface, topography-following-coordinate oceanic model," *Ocean Modelling*, vol. 9, no. 4, pp. 347–404, 2005.
- [15] Z. Li, Y. Chao, J. C. McWilliams, and K. Ide, "A Three-Dimensional variational data assimilation scheme for the regional ocean modeling system," *Journal of Atmospheric and Oceanic Technology*, vol. 25, pp. 2074–2090, Nov. 2008.
- [16] <http://robotics.usc.edu/~binney/icra2010/>.