Adaptive path planning for depth-constrained bathymetric mapping with an autonomous surface vessel

Troy Wilson | Stefan B. Williams

Australian Centre for Field Robotics, The University of Sydney, Sydney, Australia

Correspondence
Troy Wilson, Australian Centre for Field Robotics, The University of Sydney, Sydney, NSW 2006, Australia.
Email: t.wilson@acfr.usyd.edu.au

Abstract
This paper describes the design, implementation, and testing of a suite of algorithms to enable depth-constrained autonomous bathymetric (underwater topography) mapping by an autonomous surface vessel (ASV). Given a target depth and a bounding polygon, the ASV will find and follow the intersection of the bounding polygon and the depth contour as modeled online with a Gaussian process (GP). This intersection, once mapped, will then be used as a boundary within which a path will be planned for coverage to build a map of the bathymetry. Efficient methods are implemented enabling online fitting, prediction and hyperparameter optimization within the GP framework on a small embedded PC. New algorithms are introduced for the partitioning of convex polygons to allow efficient path planning for coverage. These algorithms are tested both in simulation and in the field with a small twin hull differential thrust vessel built for the task.

KEYWORDS
exploration, marine robotics, mapping, planning

1 | INTRODUCTION

Navigational maps are important for the safe passage of recreational and commercial boating traffic. Traditionally, these are created with sonar data collected from surveying vessels. The high cost of conducting these surveys impacts the frequency of re-surveying. An autonomous surface vessel (ASV) able to conduct these surveys autonomously could significantly reduce this cost and thus enable more frequent surveying to occur for a fixed budget due to lower equipment and personnel costs. An additional benefit arises from the fact that the smaller draught and reduced thrust of the ASV will allow surveying in shallower waters and create a smaller wake, resulting in less disturbance to the shoreline in sensitive estuarine environments such as mangroves. To behave in an autonomous manner in unknown environments requires the ability to sense and interpret the environment, which requires building a model of the environment in real time as it is explored, and making decisions based upon projections from this model. Using a modeling framework, which produces uncertainty estimates, such as Gaussian process (GP) regression, allows the maps produced to be used knowing the amount of confidence the model has in its projections across space.

This paper describes a system of algorithms tested in simulation and implemented in the field, which demonstrate the use of an autonomous system for bathymetric mapping. A small twin hull, differential thrust ASV with a single beam sonar for depth sensing and Global Positioning System (GPS) receiver and Inertial Measurement Unit (IMU) for localization is modeled and built. The vessel is shown to autonomously build a model of the bathymetry with a GP, where the model is continuously fit as data arrives and the parameters of the model are periodically reestimated online. A minimum depth target combined with a bounding polygon is used to both avoid static obstacles and define the area to be explored. A control algorithm plans a path to follow the intersection of the depth contour and bounding polygon. Once this boundary is mapped, a lawnmower path is then planned within it for coverage, which is shown to provide a low variance map. A new algorithm based on the Boustrophedon cellular decomposition (BCD) is developed to achieve this. To allow this system to run in real time on a small embedded system in the field, incremental algorithms have been implemented to update Cholesky decompositions of the covariance matrices required for the GP model. A multithreaded implementation has been used to share the computational load between the two physical (four virtual) cores on the ASV. The outcomes of this work are validated both in simulation and in a deployment of the ASV in an estuarine environment.

The remainder of this paper is organized as follows: Section 2 presents related work, reviewing the current state of the art in autonomous route planning. Section 3 provides a brief summary of GPs. Section 4 details the algorithmic suite developed to enable the
autonomous bathymetric surveying. Sections 5 and 6 then test these algorithms in simulation and the field. Section 7 summarizes the work with conclusions and avenues for future research.

2 | RELATED WORK

Current robotic surveying work often involves preplanned survey paths, which require prior information on the area to be surveyed and cannot react to information as it is received. Creating the optimal back and forth path for coverage of an area while staying within the workspace, which is referred to as a lawnmower path in the robotics literature or an axis parallel solution to the milling problem in computational geometry, has been shown to be related to the traveling salesman problem (TSP) and thus NP hard in general. By partitioning the complex workspace into a number of simpler shapes, which can easily be solved, and then joining these spaces together, it is possible to produce feasible paths for coverage in polynomial time. The joining together of these cells themselves optimally is also a TSP problem, and thus approximations must also be used here for polynomial time solutions.

A polygonal workspace can be split into its elemental trapezoids, known as the trapezoidal decomposition. These trapezoids are convex, and thus lawnmower paths in any direction can cover the space. This method while simple to implement can result in an excessive number of elemental cells, which can lead to a large number of inefficient transit paths to join these together. Some of these cells could be merged back together to create larger elemental convex polygons and then joined as shown in Oksanen and Visala. The authors of that work also implement a heuristic for optimizing the orientation of the tracks within each cell. Alternatively, a sweep direction can be chosen through the polygon, which is then split into elemental polygons that are monotone to this sweep direction. A polygon is monotone to a sweep direction if lines orthogonal to this sweep direction cross the polygon at most twice. This guarantees by construction that these elemental polygons can be covered by tracks orthogonal to the sweep direction; however, this also means that all cells must be covered by tracks in the same orientation. This method was introduced as the BCD. It results in a smaller number of elemental cells than even the merged version of the trapezoidal decomposition, thus reducing the number of transit paths between them. This is not guaranteed to reduce the total transit length, especially if using heuristics to solve the transit paths; however, reducing the number of cells is especially important for online implementations given the optimal solution is NP in relation to the number of cells. A graph of the elemental cells and their neighbors is created during the decomposition, and a simple search through the graph to the next antclockwise cell is conducted to join the cells together. This approach has been extended to nonpolygonal workspaces and nonlinear sweep lines as the Morse decomposition. The BCD has also been expanded to work online in unstructured environments. While some simple rules are used to allow for rejection of bad data, all other data points are assumed to be perfectly sensed.

The advantage of adaptively reacting to sensed information has been demonstrated. The authors used a chemical sensing threshold based on median differences in a small moving window compared to a large moving window to trigger a detour from a coarse survey path into a fine spiral for increasing coverage in areas likely to contain hydrothermal vents. While the trigger level was adapted throughout the mission with both the moving windows and a factor based on percentage of spirals used versus percentage of mission completed, the absolute level of the trigger was adjusted by a pretuned parameter. It is not clear this parameter would easily be set without extensive prior surveying and would significantly impact its ability to start spirals in the most informative regions.

Using the sensor data as it is collected to build a model of the parameter of interest to be studied and then planning within this model offers the potential for surveying in unknown environments under uncertainty and for optimizing paths taken to produce models based on levels of certainty and resource use. GPs offer a useful framework to deal with estimation under uncertainty and planning and can be conducted on both mean and variance information. There is a growing literature on planning within GPs, for the problem of sensor placement, which ignores travel times. Other methods such as Locally Weighted Projection Regression (LWRP) by Vijayakumar et al. could be used; however as discussed in Nguyen-Tuong et al., this method while potentially faster is more complex to implement and requires manually tuned parameters.

A GP is used to model bathymetry with an autonomous underwater vehicle in Hollinger et al. While adaptive planning is implemented based on the model predicted, this is all conducted off-line either between dives or postmission on segments of dives. The hyperparameters (HPs) of the GP are optimized once off an initial dive. GPs are fit online, with HPs optimized off prior data by Souza et al. and Marchant and Ramos to model terrain roughness and light distribution, respectively, with a ground vehicle. Greedy adaptive sampling is driven through a function based on the predicted mean plus uncertainty predictions, adjusted by a distance function, which is useful in situations where maximum values are the primary interest such as in environmental pollutants. In a similar vein, Gotovos et al. propose level set estimation, which uses the mean and uncertainty predictions from a GP to classify areas into above, below, or uncertain relative to a desired threshold. This was conducted on prior data, and the HPs were optimized from either a subset or all of the real data. A GP is modeled to an environmental data set of 2024 points in a vertical one-dimensional transect of chlorophyll-a measurements in Lake Geneva. From this 10,000 simulated points are created, though without any noise reintroduced. These simulated points are then added either sequentially individually or in batches based on uncertainty, and the resulting prediction performance of classification is compared by both number of samples and normalized travel time. Lawrance and Sukkarieh have shown that wind fields can be estimated with a GP and potential paths evaluated to on the joint objective of reducing the uncertainty in the map and retaining enough potential energy to keep a glider aloft. While the HPs are estimated online, this is done in simulation and the function used highly constrains the resulting parameters to an area close to a defined prior. Thus while GPs have been implemented online for planning purposes, their HPs are generally learnt off-line and
the authors are not aware of any which are learning their HPs online in the field.

There have been a number of published studies on the design, development, and testing of small ASVs for robotics research in recent years. Twin hull vessels have the advantage of being relatively stable in roll and have been implemented by a number of authors. All but the last of which used differential thrust for increased maneuverability, allowing rotation on the spot. Some examples of larger autonomous surfaces vessels are the full-scale catamaran used for methane sensing on an inland dam, the MIT AutoCat and Kayaks, Swordfish, Delfim, and the Springer Unmanned Surface Vessel. There has also been some commercial development in ASVs such as Swordfish, Delfim, and the Springer Unmanned Surface Vessel. Here we are using the convention of lower case letters for scalars or vectors and upper case for matrices. There are three Hyper-Parameters (HPs) \( \theta \), which must be set, namely \( \sigma_n^2 \), \( \sigma_i^2 \), and \( I \). Point estimates of these HPs can be obtained by maximizing the log marginal likelihood (LML) of the data given the parameters, which is defined as:

\[
\log(y|X, \theta) = -\frac{1}{2} y^T K_\theta^{-1} y - \frac{1}{2} \log |K_\theta| - \frac{n}{2} \log 2\pi
\]

Cholesky factorization is generally used instead of the direct matrix inversion required in Eqs. (3) and (4) for both numerical stability and speed. Even with this however, the calculation of \( K \) when fitting the GP is \( O(n^3) \). Once we have \( K \), the partial derivatives of the LML with respect to \( \theta \) are relatively inexpensive to calculate which allows fast numerical gradient descent techniques to be used. For the SE kernel we have:

\[
\frac{\partial}{\partial \theta_i} \log(y|X, \theta) = \frac{1}{2} \text{tr} \left( (aa^T - K_\theta^{-1}) K_\theta^{-1} \partial K_\theta / \partial \theta_i \right) \\
= a \cdot K_\theta^{-1}
\]

\[
\frac{\partial K_\theta}{\partial \sigma_i^2} = \exp \left( -\frac{\|r\|^2}{2\sigma_i^2} \right) \frac{\partial K_\theta}{\partial \sigma_i^2} = \delta_i \frac{\partial K_\theta}{\partial l} = \frac{\|r\|^2}{\beta^3} \sigma_i^2 \exp \left( -\frac{\|r\|^2}{2\sigma_i^2} \right)
\]

4 | AUTONOMY SUITE OF ALGORITHMS

The aim of our study is to provide a suite of algorithms to allow an ASV to operate in an unstructured environment, with minimal prior information, and to autonomously explore the area and return a map of the bathymetry.

We have chosen to use GPs to model the bathymetric contours. We initially tried bivariate splines as detailed in and implemented in the Fortran routine SURFIT, however we found large instabilities resulted very quickly short distances away from sampled points. Whilst we did not use the GP uncertainties in our algorithms, it is still useful to have these predictions to evaluate the quality of the resulting map, as can be seen in Figure 8. We chose to use a zero mean function for our GP. This was to both to keep maximum flexibility by not assuming a parametric model for the mean and simplicity by not introducing extra parameters to estimate. Specification of a mean function has most impact far from sampled points when the covariance kernel has little impact. As our algorithm only searches relatively close to currently sampled points this extra complexity would be unlikely to add much value.

There are three main components to the algorithm. Firstly we have the GP which is updated with data as it is collected by the ASV. For this to run online on a small embedded CPU care must be taken in how the GP is updated and we use analytical gradients for efficient estimation of the HPs. Secondly, we have developed an algorithm to follow the intersection of a bounding polygon and the depth contour as predicted by the GP. Thirdly, once this has concluded we propose an efficient
4.1 Online Gaussian process updates

The main computational load in fitting and predicting from GPs lies in the Cholesky factorization of the covariance matrix which is \(O(n^3)\). When we are adding data, or predicting \(m\) test points, instead of recalculating the entire Cholesky factorization, we can simply calculate the extra columns and rows related to the new data/test points, and add these to the matrix we have already calculated. For \(m\) new training/test points this reduces the update step from \(O((n + m)^3)\) to the larger of \(O(n^2m)\) or \(O(m^3)\), due to Eqs. (9) and (10), respectively. This can be done exactly for the case of additions and approximately for deletions as detailed in.\(^{34}\) This is applied in\(^{35}\) for fitting GPs to sensor data and\(^{15}\) in using GPs to model control of robotic joints. Outside of robotics GPs are generally used on a batch of data once it has been collected. Optimizations in software implementations of GPs focus on sparsifying or reducing the size of the covariance matrix to reduce computation time on one batch of data. The authors are not aware of any GP packages which implement incremental data updates and as such we will briefly detail this procedure for updating the Cholesky matrix as described in.\(^{35}\)

We have the positive semi-definite (p.s.d) covariance matrix \(K_{11}\) and its upper triangular Cholesky matrix \(L_{11}\). If we are adding new data points to the end of the covariance matrix we then have

\[
\begin{bmatrix}
K_{11} & K_{12} \\
K_{T} & K_{22}
\end{bmatrix}
\]

The new elements \(K_{12}\), its transpose \(K_{T}^{12}\) and \(K_{22}\) are calculated from the new data or test points using Eq. (1). We want to calculate the resulting Cholesky Matrix

\[
\begin{bmatrix}
S_{11} & S_{12} \\
0 & S_{22}
\end{bmatrix}
\]

For a triangular \(A\) we can use backwards substitution to solve \(AX = B\), defined as \(x = A \backslash B\). We thus find the following solutions for the elements of \(S\):

\[
S_{11} = L_{11} \tag{8}
\]

\[
S_{12} = L_{11}^T \backslash K_{12} \tag{9}
\]

\[
S_{22} = \text{Chol}(K_{22} - S_{12}^T S_{12}) \quad \text{or for } m = 1 \quad S_{22} = \sqrt{K_{22}} \tag{10}
\]

where \(\text{Chol()}\) signifies the Cholesky decomposition. The GP is run on its own thread and its covariance and Cholesky matrices are updated as sonar data arrives using the incremental method described above. When predictions are required from the GP, the current covariance matrix of training points is taken, the new covariance for the test points with themselves, \(K_{22}\), and with the training points, \(K_{12}\), are calculated and then the incremental method is used to update the Cholesky matrix to solve the GP for \(y^*\). Depending on the amount of data that is collected, there may still be issues with the size of this matrix. Methods for controlling the size of the covariance through sparsification are discussed in\(^{36}\) and the approximate Cholesky downdates can be used to efficiently achieve this.

The GP also requires determination of its HPs. As stated in Section 3, we use the analytical gradients calculated in Eqs. (6) and (7) to maximize the LML of the data given the parameters. This is run after an initialization period to collect some data and then at regular intervals. It is run on a separate thread, and when it returns new HPs these are then used by the GP. It is important to note that these HPs are used to define the covariance matrix, and thus when they change, both the covariance matrix and the Cholesky matrix need to be fully recalculated, before performing any new incremental data updates or predictions.

4.2 Find and follow the intersection of a depth contour and a boundary

The adaptive autonomy of the ASV is provided by the algorithm for finding the desired depth contour, defined by a target depth \(z_t\) and following the intersection of this with the bounding polygon, which is detailed in Algorithm 1. The depth contour is set at the intersection of a safe operating depth and a minimum depth of interest for the study. For a point sensor only sensing directly downwards we cannot avoid floating objects such as buoys, or very steep gradients such as vertical seafloor rises. The vessel should be able to both operate safely due to these bounds and obtain the sonar data required to create a bathymetric map of the area. The bounding polygon will keep the ASV both in an area of interest and away from obstacles unable to be predicted through modeling of the bathymetry. Inspired by Bug type algorithms such as DistBug,\(^{37}\) the algorithm follows the desired depth contour until it hits a boundary. Upon hitting the boundary it then follows this boundary until it finds the boundary taking it shallower than the target depth, at which point it leaves the boundary and again begins following the depth contour. This is continued until a circuit has been completed. A key difference in our algorithm to the bug algorithms is that we are not aiming for a single goal but always searching for a desired depth at a distance \(r\), the search radius from our current position. This search is detailed in Algorithm 2. In addition, the surface upon which we are searching, the GP model of the bathymetry, is changing as we obtain data and both reft the model and re-estimate the HPs of the model. This model changes faster in the initial stages when we have small amounts of localized data. This can lead to the initial path turning back on itself. To counteract this noise, and prevent early determination of boundary closure, we set a parameter on the boundary completion test for loop closure to ignore the most recent loopBuffer points. The value of loopBuffer should be set relative to the expected length of the boundary.
ALGORITHM 1  Find and follow contour within bounding polygon

1. procedure FFCB($poly, r, z_t, loopBuffer, \psi_{adj}$)
2.    mode $=$ Contour, $polyDir = \emptyset$
3. repeat at 10Hz
4.    Obtain current $GP$ and pose estimate $x, y, \psi$ from separate threads
5. if mode $=$ Contour then
6.        $\psi_s, \psi_e = \psi - \psi_{adj}, \psi + \psi_{adj}$
7.        $x_w, y_w = \text{roseSolve}(GP, z_t, \psi, x, y, r, \psi_s, \psi_e)$
8. if $x_w, y_w$ is not in $poly$ then $\rightarrow$ Crossing boundary
9.    mode $=$ Boundary
10. if $polyDir = \emptyset$ then
11.        Set $polyDir$ to index direction along edge to deeper water
12. Get vertex $x_w, y_w$ in direction $polyDir$ from edge of $poly$ crossed
13. else
14. Get $x_w, y_w$ from current vertex in direction $polyDir$
15. if Distance to $x_w, y_w < z_t$ then
16.    Get $x_w, y_w$ from next edge in direction $polyDir$
17. if Depth predicted at $x_w, y_w < z_t$ then $\rightarrow$ Leave boundary
18. mode $=$ Contour
19. Set $\psi_s, \psi_e$ to the intersection of an arc of radius $r$ centered at $x, y$ with $poly$
20. $x_w, y_w = \text{roseSolve}(GP, z_t, \psi, x, y, r, \psi_s, \psi_e)$
21. Calculate heading $\psi_d$ to $x_w, y_w$ from $x, y$
22. if boundary or contour has been found then
23. Append $x, y$ to $boundaryList$
24. Send $\psi_d$ to heading controller
25. until Boundary completed by more than $loopBuffer$
26. return $boundaryList$

ALGORITHM 2  Solve for heading to target depth $z_t$

1. procedure ROSETOLVE($GP, z_t, \psi, x, y, r, \psi_s, \psi_e$)
2. Set $\psi_*$ as a linearly spaced vector from $\psi_s$ to $\psi_e$ with 50 increments
3. Predict $z_*$ at distance $r$ from $x, y$ for headings $\psi_*$ from $GP$ in a batch
4. for each sequential pair of $\{z_*, \psi_*\}$ points do
5. Set $\psi_d$ as the $\psi$ with the minimum $\text{abs}(\psi - z_t)$, with ties broken by minimum $\text{abs}(\psi - \psi)$
6. Set $x_w, y_w$ as the cartesian coordinates of the ray projected from $x, y$ at $\psi_d$ for $r$
7. return $x_w, y_w$

The RoseSolve() algorithm searches for the heading leading to the target depth on the circumference of an arc on the compass rose of radius $r$ around the current position, as shown in Figure 1. Whilst in contour following mode, this arc is centered around the current heading and bounded by the start angle, $\psi_s$, and end angle, $\psi_e$. This range, $\pm \psi_{adj}$ can be chosen to reduce the search space so we are not wasting computation searching where we have just come from. When exiting boundary following mode, $\psi_s$ and $\psi_e$ for the search are set based on the boundary of the polygon, such that the arc is inside the polygon. This arc is then evenly split into a number of points, which are converted into Cartesian co-ordinates and the predicted depth for these positions is returned in a batch from the GP. Linear interpolation is conducted between each pair of points for the heading which returns the depth closest to the target depth. We then return the best of these solutions with ties broken based on the distance to our current heading. This behavior can be seen in Figure 1, where the algorithm is solving for a desired depth of 3m. For the position on the left and near the middle, an arc centered around the current heading is searched, whilst in the position on the right boundary it can be seen we are searching on the arc within the boundary. The depth for all the points on these arcs are queried from the GP, and then the best segment linearly interpolated to get the solution, indicated by the solid grey dot. For

![FIGURE 1](image_url)
the positions on the left and right, the algorithm finds the heading which will lead to the desired depth, whereas the position in the middle returns the heading which is closest to the desired depth.

Originally we implemented this as a recursive bi-section search on the arc, however the computational cost of repeatedly querying the GP for 1 prediction point method led us to implement a batch procedure. The number of splits is a design parameter. With 50 splits, we have a prediction point no more than $\frac{1}{40}\pi$ radians apart, which with a search radius $r = 5\text{m}$ equates to test point spacing of approximately 0.4m. The velocity and sampling frequency on our ASV resulted in spacing of the sampling of points of around 1m along the path of the vehicle. Thus the combination of this test point spacing with linear interpolation, the smooth surface provided by the GP on this scale given the sampling scale, and the frequency on the control loop speed was found to be a reasonable compromise for computational load and precision.

A key tunable parameter in this algorithm is the search radius. A number of factors come into play in the choice of the search radius. The smaller this radius is, the smaller the area in the GP around the current position of the vessel is searched for the desired heading. A larger radius will result in a smoother path, however, this will also result in a larger tracking error between the desired depth found on this radius, and the depth sensed directly below the vessel. The expected rate of change of the bathymetry should be taken into account such that the search radius is set to allow the vessel to follow these changes. The speed of the vessel and the sampling rate of the sonar should also be taken into account.

### 4.3 Discrete monotone polygonal partitioning and path generation

Upon completion of the intersection of the depth contour and the bounding polygon we have a new intersection polygon around which we have already sensed and which we now need to plan within for coverage. For the purpose of this study the track width is a design parameter. Similar to the BCD we have implemented a method to create elemental polygons which are monotone to a given sweep direction. There are some differences however which we have implemented to produce a more optimal path given the desired path spacing.

Under BCD, a sweep direction is chosen. A line orthogonal to this sweep direction is traced through the polygon. At any point where the number of crossings of this line with the polygon changes we have a connectivity event. These events are used to create the elemental polygons which are monotone to this sweep direction. This can be seen in Figure 2(a), where there are 3 changes in the connectivity count which is used to create the three elemental polygons (note this is using a horizontal sweep direction left to right). The union of these three polygons returns the original polygon. These polygons are then joined in an anti-clockwise order as shown by Figure 2(a), with lawnmower paths then drawn within them as shown by Figure 2(b).

The first issue with this partitioning is that the width of these polygons is not necessarily a multiple of the track width. The result of this can be seen in Figure 2(b), where the last up track in cell 1 is close to the cell boundary. The first vertical tracks in cells 2 and 3 are then closer to this track than our desired spacing which results in longer total path length and irregular coverage. For our application of a point sensor this results in some sensed points being closer than required. In the application of perfect sensing of a fixed width scanner, this would result in a significant amount of scan overlap on the cell edges, again a waste of resources. We have developed the Discrete Monotone Polygonal Partitioning (DMPP) method to explicitly deal with this, which will be discussed after detailing the other shortcomings of the BCD method.

The second issue with the BCD method is that the transit paths joining the cells are not optimized. The order of joining is simply conducted through an anti-clockwise search of the neighbors of the current

---

**Figure 2** Comparison of BCD (a,b) with DMPP (c,d,e) showing more efficient coverage with DMPP both in terms of number of vertical transects (BCD requiring 1 more in cells 2 and 3) and distance to join cells.
cell for the first cell that has not yet been covered, repeated until all cells are covered. It is not until all the transit paths are computed that the lawnmower path within the cell is computed. This is clearly sub-optimal. As can be seen in Figure 2(b), lawnmower coverage of cell 1 ends in the top right hand corner. Clearly the best cell to go to would be the upper right cell, however the BCD method is not aware of this and instructs a transit to the bottom right cell. Which corner of the cell to join to should also be dependent on which corner of the prior cell we exited from, however the BCD method does not actually detail how the lawnmower paths within the cell should be constructed, which in any case are only fit after the order of cell joining is determined. We deal with this second issue by calculating the lawnmower path within the closest cell, then solving an A* search from the exit point of this cell. The lawnmower paths within the cell should be constructed, which in any case are only fit after the order of cell joining is determined. We deal with this second issue by calculating the lawnmower path within the closest cell, then solving an A* search from the exit point of this cell to the nearest corner of an unexplored cell, and repeating until all cells are covered.

Finally the BCD is designed to handle polygonal objects within its boundary. The boundary is set up as a simple rectangle. Whilst there is nothing to stop their general approach being applied to more complex boundaries, they do not mention this case and how it would be approached. Our algorithm explicitly deals with this case.

Algorithm 3 details the DMPP. The input parameters for this algorithms are the boundary polygon poly, the desired track width $\delta$ and the sweep direction $\psi_d$. By constraining the sweep direction in the range $-\pi/2$ to $\pi/2$, without loss of generality, we can sweep from left to right from the bottom left corner of poly. Lines 4-6 of Algorithm 4.3 detail this sweeping process of producing a list of crossings of a line orthogonal to the sweep direction with the polygon which is sorted from the bottom. An example of this can also be seen in Figure 2(c) where the sweep lines are shown, and their corresponding crossing count shown on the top of the chart.

With this list of crossings, we then look for changes from one sweep line to the next, indicating a connectivity change event. When this happens we close our open cells in order from bottom to top, before we open new cells, again ordered from bottom to top. The coordinates of the individual cell corners created are ordered clockwise from the bottom left corner. These 4 points define the first and last tracklines in the cell. We then need to follow the boundary between point 2 to 3 and point 4 to 1 (as the other two sides are the tracklines), to fully define the elemental cell, completing our DMPP.

**Algorithm 3** Discrete Monotone Polygonal Partitioning (DMPP)

```
1: procedure DMPP(poly, $\delta$, $\psi_d$)
2:   Set angle of the line $\psi_1$ to $\psi_d + \pi/2$
3:   Set $x$, $y$ to the minimum values of poly in the $x$ and $y$ directions
4:   while $x$ or $y$ inside poly do
5:     Project ray from $x$, $y$ at angle $\psi_1$, append sorted list of crossings to sweepCrossings
6:     Adjust $x$, $y$ in direction $\psi_d$ by $\delta$
7:   for each list of crossing points in sweepCrossings do
8:     if there is a change in the number of crossing compared to the prior list then
9:       if there are open cells then
10:      for each pair of crossing points in the prior list of sweepCrossings do
11:         Set crossing points as closing corners of associated open cell, append cell to cells
12:     for each pair of crossing points do
13:        Open a new cell, set crossing points as opening corners
14:   for each cell in cells do
15:      Add additional required points on the non-trackline edges to trace the boundary poly
16: return cells
```
where $N_{DMPP}, N_{BCD}$ are the number of tracklines orthogonal to the sweep direction due to the DMPP and BCD, respectively. $R_{DMPP}, R_{BCD}$ are the sums of the remainders for each method, $W_p, W_s$ are the width of the polygon of the current segment in direction $\psi_d$, where the segment is defined by connectivity events in the relevant scheme and following39 we define $A \div B = \lfloor A/B \rfloor$, $A \mod B = A - B \lfloor A/B \rfloor$ where $\lfloor \cdot \rfloor$ is the floor function.

Since DMPP by definition creates a cells which are a multiple of the trackwidth $\delta$, $R_{DMPP} = 0$. BCD on the other hand does not consider $\delta$ which discretizing the cells. Thus $R_{BCD} \geq 0$, and in any realistic scenario in the field $R_{BCD} > 0$. If $R_{BCD} \geq \delta$ then BCD will create additional tracklines over and above DMPP. Thus we have:

$$N_{DMPP} \leq N_{BCD} \text{ with equality iff } R_{BCD} < \delta$$

There are then the additional inefficiencies in the BCD method which does not search for the most efficient way to join the cells and decides how to join the cells before knowing where the paths exit and enter them. For even the simple example shown in Figure 2 the BCD method results in total within cell path 6% longer and transit paths 98% longer for a total path which is 11% longer than our DMPP and path generation algorithm. An example for a more complex polygon can be seen in Figure 3. Unlike the BCD in which the union of the cells is the polygon, in the DMPP there are spaces between the cells. However these are designed such that the tracklines are exactly upon these edges and thus we achieve the even coverage we desire. This has the additional advantage that we do not need to worry about joining adjacent elemental cells as they will be covered efficiently. This can be seen in Figure 3 where the two cell on the bottom left could be combined into one cell, but it would make no difference in the planned path.

Whilst this is a greedy method in that it is only ever looking ahead one cell, it is superior to the BCD, which naively takes the nearest ant-clockwise cell regardless of transit distance. The resultant path length could be improved at the expense of computation time by a deeper search, though an exhaustive search would quickly become infeasible as the number of cells grows due to the NP nature of the problem. The sweep direction could also be optimized, perhaps through running a number of potential rotations similar to the trapezoidal sweep optimization in, though unlike their method, we must choose the same trackline orientation for all cells as they have been created monotone to the same sweepline orientation. Whilst we have only shown the DMPP and path fitting algorithms for polygons without loops or holes, as this was our use case, the algorithms themselves can easily be adapted to this.

5 | SIMULATION

A simulated bathymetry has been created to demonstrate these algorithms. A vessel with perfect localization, sensing and control is tested to focus on validating the performance of the coverage algorithm itself. The parameter settings can be seen in Table 1. From the start point the vehicle is driven in a circle of radius 5m for 50s to gain some initialization points for the GP. After this initialization, HPs are estimated, and then again every 30s.

As can be seen in Figure 4(a), the vessel follows the contour gradient it has discovered by searching on its GP model of the bathymetry and follows south until it arrives at the target depth. It then turns east and follows this contour until it gets to the western boundary at [0,237]. At this point, it follows the boundary south into deeper water until this boundary following would take it shallower than the target depth at which point it turns east again and follows the contour. After another boundary and contour following section, it completes tracing the intersection of the boundary and the depth contour.

The algorithm now moves on to partitioning this intersection and creating a path for coverage. Figure 4(b) shows the result. The intersection is split into four cells. From the start point the closest cell corner, on the south west, cannot be transited to in a straight line as this would take us out of the polygon. An A* path is generated to take us to this point. A lawnmower path is then generated to take us through this cell to the west. A transit path from the end of this cell is then generated to take us to the north west corner of the next cell, which can be done directly as this stays within the polygon. The lawnmower path is then generated for this cell, and the process repeated for the final two cells until we have a plan for coverage of the entire space.

### Table 1 Parameter settings for simulation and field trials

| Hyperparameter          | Simulation | Field Trial |
|-------------------------|------------|-------------|
| Velocity (m/s)          | 1.0        | ≈ 1.0       |
| $z_t$ (m)               | 4.5, 5.0, 7.5 | 5.0         |
| $\delta$ (m)           | 10.0       | 5.0         |
| $\psi_d$ (rad)          | 0.0        | 0.0         |
| Start point (m)         | 250E, 350N | 0E, 0N      |
| IMU (Hz)                | 1          | 50          |
| GPS (Hz)                | 1          | 1           |
| Control loop (Hz)       | 1          | 5           |
| $\lambda_{EMA}$ (s)    | 5          | 5           |
The main design parameters for these algorithms are the target depth $z_t$, the path spacing $\delta$, and the search radius $r$. The target depth should be chosen based on a combination of the safe operating depth of the vehicle and the depths of interest for the study. The path spacing should be chosen based on the coverage density desired. The search radius impacts the operation of the algorithm in a number of ways. A larger search radius expands the search horizon, though if this is too large it may move us away from where we have certainty in our model. As this search radius increases the ability of the vessel to smoothly follow tight turns in the contour is reduced, and a tracking error between what is directly under the vehicle compared to the depth at the planning horizon while following a curve is introduced. As such this parameter should be bounded from above based on an expectation of the minimum radius of curves in the contours it is following. On the lower range of this variable, we want the planning horizon to be longer than the distance covered by the vessel between planning points (in the simulation case this is 1 m due to a velocity of 1 m/s and a control loop of 1 Hz). Empirical testing has shown the solution to be robust to the choice of $r$ as can be seen in Figure 4(a) where setting $r$ at 2.5 m or 7.5 m results in a very similar path to $r = 5$ m.

6 | FIELD TESTS

To demonstrate the robustness of these algorithms to the noise introduced from uncertainty in sensing and localization from operating in the field and the computational limits imposed by an embedded CPU and real-time operation, a small ASV was built. The vessel, which can be seen in Figure 5, is a twin hull, differential thrust design. The twin hulls are made of medium density polyurethane foam with a resin coating joined by two aluminum crossbars. There are two waterproof electronics enclosures, one mounted in each hull. One box contains the batteries and battery controller, with the rest of the electronics in the other. It has a footprint of 1.2 m × 0.85 m and weighs approximately 10 kg. This small form size enables deployment by a single person. Localization is provided by a Flexpacs G6 GPS running at 1 Hz combined with a Vectornav VN-100 rugged IMU running at 50 Hz fused with a Kalman filter. Bathymetric sensing is provided by an Airmar single beam sonar operating at 1 Hz. An Embedded Solutions ADLN2000PC containing an Intel Atom N2600 1.6 GHz dual (four virtual) core processor with 2 Gb RAM provides the computational resources. Information relay and manual override control are handled with an Xbee Pro 2.4 GHz RF module. Power is supplied by an Ocean Server BBDC-02R dual battery controller connected to 2 BA-95HC 6.6 Ah Li-ion battery packs providing 6 h of operational time. Propulsion is provided by two Seabotix BTD150 thrusters. There are also leak and temperature sensors inside the payload boxes, which combined with a heartbeat published over the radio frequency channel trigger automatic kill switches for safety. The vessel has a maximum velocity of approximately 1 m/s. The vessel is very maneuverable given its lightweight and differential thrust design and is able to rotate on the spot.

The framework used for the code running on the ASV is robot operating system (ROS) Indigo, installed on Ubuntu 14.04. All code is written in Python for quick development; however, ROS also supports C++, so modules can be rewritten if processing speed becomes an issue. ROS inherently supports multithreaded operation with a single launch file initiating the various modules referred to as nodes. Communication between the nodes is handled by both a parameter server to read/write global variables and a publish/subscribe messaging system. A separate GUI has been written in Python and QT, which runs on a laptop and interfaces with the ASV over the Xbee link. This provides
both information on the ASV, a map of the ASV path/way points and manual control of the ASV. An example from the mission can be seen in Figure 6.

The area chosen for this task is a secluded part of the Port Hacking river, to the south of Sydney, Australia, called Cabbage Tree Basin. This is an interesting area for a number of reasons. The Port Hacking river was the first estuary in Australia closed to commercial fishing in the late 19th century. It is bordered on one side by a residential area and on the other by the Royal National Park, which is the second oldest national park in the world (after Yellowstone), established in 1879. Cabbage Tree Basin itself is a significant area of heritage value with a long history of Aboriginal occupation. It was the site of the first marine hatchery in Australia in 1900 and is one of the earliest described estuarine wetland areas in Australia. The long shallow entrance to the basin also significantly limits access to recreational boating traffic, enabling unobstructed operation of the ASV.

A bounding polygon 100–150 m wide by 40 m high was set, as can be seen by the white trapezoid in Figure 7. This area encompassed depths from less than 50 cm to 8 m. The parameters used can be seen in Table 1. The mission starts at the northeast corner of the white trapezoid, which can be seen in Figure 7. The vessel was manually driven south-west for approximately 10 m until it reached a depth of 1 m. It was then driven in an arc for 5 s (achieving a quarter of a circle). The GP hyperparameters are estimated from these initial points and then again every 30 s. The FFCB control loop, Algorithm 1, then started operation. Figure 8 shows some snapshots of the GP model, planning and path traveled during the mission. In each of the six pairs of figures, the upper figure shows the path of the vessel overlaid, at a given point in time, on the bathymetry estimated by the model, while the lower chart shows the confidence of the model at this time point through the standard deviation (in log scale), with one data point per second. A video showing the evolution of the model as each new data point arrives can also be seen online.

To increase the robustness of the algorithm in the field trials, we used an exponentially smoothed average of the current estimated heading, with a half-life of 5 s (equivalent to the time taken to cross the planning horizon). This was used in contour following mode to center the search space for the roseSolve() algorithm to smooth out the short-term effects of any environmental forcing on the instantaneous heading.

Figure 8(a) shows the path and model immediately after the initialization period and first HPs estimation. At this point, the algorithm does not have a very good model to work with when trying to follow the contour. Owing to this the vessel traces, a tight circle between $t = 30$ s and $t = 37$ s before continuing to follow the contour south until it hits the desired depth. It then follows this depth contour southeast until
it gets to the southern boundary. This boundary is followed along the bottom edge, up the western side and partially across the top until this would take it too shallow, as can be seen in Figure 8(c), at which point it switches back to contour following mode with the vessel then following the contour back toward where it first found the contour. This is achieved by \( t = 396 \) s. The intersection of the depth contour and boundary polygon is now used by the DMPP and path generation algorithm to plan a path. In this case, the intersection is already a monotone polygon with respect to the sweep direction and thus only one cell was created and a lawnmower path was fit to it. The vessel then followed this path for coverage. As can be seen from Figures 8(d)–8(f), the standard deviation of the map within the intersection falls to around 3 cm after the coverage task has been completed at point 1033.

The history of the HPs as they were optimized throughout the mission can be seen in Figure 9. The range of the noise standard deviation, whilst it looks large, is on a different scale and two orders of magnitude smaller than the model noise, so is irrelevant in terms of its implications for the control algorithm. The model standard deviation, \( \sigma_f \), relatively quickly stabilizes around 3–4 m by the third point. The characteristic length scale estimate slowly decreases as we get more data, though from the fourth point the estimate is in a range of 16.8–27.9 m. The final estimated HPs in the contour following phase are \( \sigma_f = 4.07 \) m, \( l = 18.45 \) m, and \( \sigma_n = 0.103 \) m.

![Figure 8](image_url)  
**Figure 8**  
Scanned points recording in field trial overlaid on online GP estimated depth contours (top) and standard deviations (bottom) at \( t = 30, 150, 270, 396, 700, \) and 1033

![Figure 9](image_url)  
**Figure 9**  
Iterative HP estimation

The level of the prior uncertainty and thus the asymptotic level that is approached as we move away from observed data points is controlled by \( \sigma_f \), while \( \sigma_n \) is the estimated noise. Neither of these parameters impacts our algorithm. Higher noise from our sensors or errors in localization would show up in an increased \( \sigma_n \). The length scale parameter is the key parameter here as it estimates how far away from a test point we consider other points. If this were to be solved at a value significantly larger than the space, we are sampling in, 1000 for instance, then we would essentially fit a flat plane through the data points. If
it was much smaller then we would only be using a very local range of points in prediction. If we were sampling in an area with rougher bathymetry, then we could expect a smaller length scale to model this. We did not experience any serious issues with instability of the HPs in the field trial or simulation leading to problems in contour following. This would be most likely to occur at the start of the mission with a small number of measurements. If this were to become an issue, potential solutions would be a longer initialization period, or to take a more Bayesian approach and impose prior distributions on the HPs. We can see that with the changes in the HPs throughout the mission, the algorithm showed its ability to follow an estimated contour, which appears reasonable when compared to the final estimate of the bathymetry as can be seen in Figure 8(f).

To estimate the accuracy of the online contour following algorithms, a GP was fitted to 3731 data points collected in the area on a number of missions in the 2 h prior to the mission shown. Postmission at each point where a sonar reading was taken, we have calculated the heading that would have been generated by the RoseSolve algorithm given the data and HPs that were in place at that time. For each of these headings, the depth at \( r \) is predicted from both the online model and the model generated from prior data. A histogram of these points overlaid with the estimated kernel density (using a Gaussian kernel with a bandwidth of 0.05) is shown in Figure 10. The mean of these errors is –0.21 m and the standard deviation is 0.23 m. This mean bias is likely due to the fact that the tide was receding and the tidal range that day was 0.74 m. The standard deviation of 0.23 m is reasonable given the estimated noise on the prior model was 0.18 m. The largest errors occurred in the northwest corner where we had the least coverage in prior data as can be seen in Figure 10. The mean of these errors is –0.21 m and the standard deviation is 0.23 m. This mean bias is likely due to the fact that the tide was receding and the tidal range that day was 0.74 m. The standard deviation of 0.23 m is reasonable given the estimated noise on the prior model was 0.18 m. The largest errors occurred in the northwest corner where we had the least coverage in prior data as can be seen in Figure 10. 

As we can see by comparing the sequential plots in Figure 8. In Figure 8(b), the model believes that the depth starts to rise about 7 m away from the current position, whereas once this area has been sampled, which we can see in Figure 8(c), we can see that it actually stayed constant. If we had tried to make a plan to follow the whole contour from the data and model we had at the time of Figure 8(b), we would not have achieved the result seen in Figure 8(c) by following the contour in a local region.

7 | CONCLUSIONS AND FUTURE WORK

In this paper, we have developed and implemented a suite of algorithms for autonomously finding and following the intersection of a bathymetric contour and a bounding polygon, and then fitting a path for coverage within this boundary for the purpose of producing a map of the bathymetry in an unknown area. We provide a new algorithm for the partitioning of complex polygonal workspaces and the planning of coverage paths within them, which is more efficient than the BCD method and explicitly handles the complex boundary shapes. We have implemented computationally efficient methods for the updating of Cholesky matrices used by the GP to allow online fitting and prediction of a bathymetric map, including online optimization of the HPs of the GP. Code for all algorithms and the implementation on the ASV are provided on GitHub.2

We have also developed and implemented a small autonomous surface vehicle with which we can test this and other robotics algorithms and conduct surveys in estuarine waterways. We tested these algorithms in simulation and in the field and through these experiments have shown the robustness of the platform and algorithms to uncertainty introduced by sensor noise and environmental forcing in a dynamic environment combined with the ability to run in real time on a small embedded system.

GPs inherently have scaling issues for fitting and prediction due to the inversions required of the covariance matrices which are \( O(n^3) \). This was not an issue for our surveys, but would need to be considered for significant increases in scale. Future work in this area could look at implementing methods for sparsification or reduction of the
covariance matrices. Specific to our algorithms, given the search space is relatively local, we could use only the data points relevant for making a map in the area searched. This could be guided by the model such that the area within which we should use the data points would be guided by the combination of the search range and the HPs which determine the range of predictive power of each data point. Computational speed for estimation of the HP is less of an impediment for the online autonomous operation as it can be run independently of the control loop in a separate thread, or potentially even off board the ASV if the data points are all transmitted across, with the results integrated as they arrive. We chose a fixed window for the reoptimization of the HPs. A more principled approach could potentially look at an estimate the uncertainty of HPs. Fisher information could be used here, the inverse of which through the Cramér Rao inequality provides a lower bound on variance of the parameters, with the window for reoptimization growing as the estimate of our uncertainty reduces.

In this study, the mean predictions from the GP were used for mapping the boundary and once mapped a lawnmower path was executed within for coverage. Natural extensions to this would be to use the variance predictions in addition to the mean for creating a safe boundary following algorithm and once this boundary is defined to use the variance information from the GP to drive the exploration within this boundary to create maps of a desired fidelity, with a minimum length path. A nonstationary covariance matrix could be fit which could allow heterogeneous spacing of sampling points dependent on the local variability of the bathymetry.

The single beam sonar, while likely to be replaced by a multibeam sonar for creating maps for navigational purposes, serves as a useful analogue for other variables of interest in estuarine environments which can only be measured with point sensors. We intend in future work to use point sensors for dissolved oxygen, salinity, water temperature, turbidity, pH and chlorophyll-a, to measure the health of estuaries. Spatiotemporal GPs could autonomously model these variables in an estuary over the cycle of an incoming tidal front. The path of the ASV will be planned online as the model is fit to create the best model of these variables.

ACKNOWLEDGMENTS

This work is supported by the Department of Education and Training through the Australian Postgraduate Awards scheme and the Australian Research Council through the Future Fellowships award. The authors would also like to thank Seatools Pty Ltd for support in construction of the platform used for the field trials.

ENDNOTES

1. https://www.youtube.com/watch?v=yyTFwQdxD9U.
2. https://zenodo.org/record/47963.

REFERENCES

1. Senet CM, Seemann J, Flampouris S, Ziener F. Determination of bathymetric and current maps by the method DISC based on the analysis of nautical x-band radar image sequences of the sea surface (november 2007). IEEE Trans Geosci Remote Sensing. 2008;46(8):2267–2279.
2. Clark CM, Hanke K, Xydes A, et al. Estimation of volumetric oxygen concentration in a marine environment with an autonomous underwater vehicle. J Field Robot. 2013;30(1):1–16.
3. Grasmueck M, Eberli GP, Viggiano DA, Correa T, Rathwell G, Luo J. Autonomous underwater vehicle (AUV) mapping reveals coral mound distribution, morphology, and oceanography in deep water of the straits of Florida. Geophys Res Lett. 2006;33(23).
4. Johnson-Roberson M, Pizarro O, Williams SB, Mahon I. Generation and visualization of large-scale three-dimensional reconstructions from underwater robotic surveys. J Field Robot. 2010;27(1):21–51.
5. Galceran E, Carreras M. A survey on coverage path planning for robotics. Robot Auton Syst, 2013;61(12):1258–1276.
6. Arkin EM, Held M, Smith CL. Optimization problems related to zigzag pocket machining. Algorithmica. 2000;26(2):197–236.
7. Latombe J-C. Robot Motion Planning. Berlin: Springer Science & Business Media; 2012;124.
8. Oksanen T, Visala A. Coverage path planning algorithms for agricultural field machines. J Field Robot, 2009;26(8):651–668.
9. Choquet H, Pignon P. Coverage path planning: The boustrophedon cellular decomposition. In: Roman C, ed. Field and Service Robotics. Berlin: Springer; 1998:203–209.
10. Acar EU, Choquet H, Rizzi AA, Atkar PN, Hull D. Morse decompositions for coverage tasks. Int J Robot Res. 2002;21(4):331–344.
11. Acar EU, Choquet H. Robust sensor-based coverage of unstructured environments. In: IEEE International Conference on Intelligent Robots and Systems. Piscataway, NJ: IEEE; 2001;1:61–68.
12. Ferri G, Jakuba MV, Yoerger DR. A novel method for hydrothermal vents prospecting using an autonomous underwater robot. In: International Conference on Robotics and Automation (ICRA), Piscataway, NJ: IEEE; 2008:1055–1060.
13. Krause A, Singh A, Guestrin C. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. J Mach Learn Res. 2008;9:235–284.
14. Vijayakumar S, D’souza A, Schaal S. Incremental online learning in high dimensions. Neural Comput. 2005;17(12):2602–2634.
15. Nguyen-Tuong D, Seeger M, Peters J. Model learning with local Gaussian process regression. Adv Robot. 2009;23(15):2015–2034.
16. Hollinger GA, Mitra U, Sukhatme GS. Active and adaptive dive planning for dense bathymetric mapping. In: Experimental Robotics. Berlin: Springer; 2013:803–817.
17. Souza JR, Marchant R, Ott L, Wolf DF, Ramos F. Bayesian optimisation for active perception and smooth navigation. In: International Conference on Robotics and Automation (ICRA). Piscataway, NJ: IEEE; 2014:4081–4087.
18. Marchant R, Ramos F. Bayesian optimisation for intelligent environmental monitoring. In: International Conference on Intelligent Robots and Systems. Piscataway, NJ: IEEE; 2012;2242–2249.
19. Gotovos A, Casati N, Hitz G, Krause A. Active learning for level set estimation. In: International Joint Conference on Artificial Intelligence (IJCAI), 2013. Wiley.
20. Lawrance NR, Sukkarieh S. Path planning for autonomous soaring flight in dynamic wind fields. In: International Conference on Robotics and Automation (ICRA). Piscataway, NJ: IEEE; 2011:2499–2505.
21. Hitz G, Pomerleau F, Garneau M-E, et al. Autonomous inland water monitoring: Design and application of a surface vessel. Robot Automat Mag. 2012;19(1):62–72.
22. Matos A, Cruz N. Positioning control of an underactuated surface vessel. In: Oceans 2008. Piscataway, NJ: IEEE; 2008:1–5.
23. dos Santos DS, Nascimento CL, Cunha WC. Autonomous navigation of a small boat using IMU/GPS/digital compass integration. In: International Systems Conference (SysCon). Piscataway, NJ: IEEE; 2013:468–474.

24. Tokekar P, Bhadauria D, Studenski A, Isler V. A robotic system for monitoring carp in Minnesota Lakes. J Field Robot. 2010;27(6):779–789.

25. Dunbabin M, Grinham A, Udy J. An autonomous surface vehicle for water quality monitoring. In: Australasian Conference on Robotics and Automation (ACRA). 2009;2–4.

26. Manley JE, Marsh A, Cornforth W, Wiseman C. Evolution of the autonomous surface craft AutoCat. In: Oceans 2000. Piscataway, NJ: IEEE; 2000;1:403–408.

27. Ferreira H, Martins R, Marques E, et al. Swordfish: An autonomous surface vehicle for network centric operations. In: Oceans 2007–Europe. Piscataway, NJ: IEEE; 2007;1–6.

28. Pascoal A, Oliveira P, Silvestre C, et al. Robotic ocean vehicles for marine science applications: The European ASIMOV project. In: Oceans 2000. Piscataway, NJ: IEEE; 2000.

29. Naeem W, Sutton R, Xu T. An integrated multi-sensor data fusion algorithm and autopilot implementation in an uninhabited surface craft. Ocean Eng. 2012;39:43–52.

30. Hine R, Willcox S, Hine G, Richardson T. The wave glider: A wave-powered autonomous marine vehicle. In: Oceans 2009. Piscataway, NJ: IEEE; 2009;1–6.

31. Saildrone. Saildrone. http://www.saildrone.com. Accessed January 18, 2016.

32. Rasmussen CE, Williams CKI. Gaussian Processes for Machine Learning. Cambridge, MA: MIT Press; 2006.

33. Dierckx P. Algorithms for smoothing data with periodic and parametric splines. Comput Vision Graph. 1982;20(2):171–184.

34. Seeger M. Low rank updates for the Cholesky decomposition. Technical report, University of California, Berkeley, CA; 2007.

35. Osborne M, Roberts S, Rogers A, Ramchurn S, Jennings N. Towards real-time information processing of sensor network data using computationally efficient multi-output Gaussian processes. In: International Conference on Information Processing in Sensor Networks. Piscataway, NJ: IEEE; 2008:109–120.

36. Quinonero-Candela J, Rasmussen CE, Williams CK. Approximation methods for Gaussian process regression. Large-Scale Kernel Mach. 2007:203–224.

37. Kamon I, Rivlin E. Sensory-based motion planning with global proofs. IEEE Trans Robot Automat. 1997;13(6):814–822.

38. Dechter R, Pearl J. Generalized best-first search strategies and the optimality of A*. J ACM. 1985;32(3):505–536.

39. Graham RL, Knuth DE, Patashnik O, Liu S. Concrete mathematics: A foundation for computer science. Comp Phys. 1989;3(5):106–107.

40. Quigley M, Conley K, Gerkey B, et al. Ros: An open-source robot operating system. In: International Conference on Robotics and Automation (ICRA) Workshop on Open Source Software, Kobe, Japan: 2009;3:5.

41. West R, West J. Cabbage tree basin: Natural values and options for management. Unpublished Report for the Hacking Catchment Management Committee. 2000:79.

42. Hensman J, Nicolo Fusi NDL. Gaussian processes for big data. In: Conference on Uncertainty in Artificial Intelligence (UAI-13). Bellevue, WA: AUAI Press; 2013:282–290.

43. Cover TM, Thomas JA. Elements of Information Theory. Hoboken, NJ: Wiley; 2012.

How to cite this article: Wilson T, Williams S. Adaptive path planning for depth-constrained bathymetric mapping with an autonomous surface vessel. J Field Robotics. 2018;35:345–358. https://doi.org/10.1002/rob.21718