The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

Inspection Planning for Sensor Coverage of 3D Marine Structures

Brendan Englot and Franz Hover, *Member, IEEE*

***Abstract*— We introduce an algorithm to achieve complete sensor coverage of complex, three-dimensional structures surveyed by an autonomous agent with multiple degrees of freedom. Motivated by the application of an ocean vehicle performing an autonomous ship hull inspection, we consider a planning problem for a fully-actuated, six degree-of-freedom hovering AUV using a bathymetry sonar to inspect the complex structures underneath a ship hull. We consider a discrete model of the structure to be inspected, requiring only that the model be provided in the form of a closed triangular mesh. A dense graph of feasible paths is constructed in the robot's configuration space until the set of edges in the graph allows complete coverage of the structure. Then, we approximate the minimum-cost closed walk along the graph which observes 100% of the structure. We emphasize the embedding of observations within the edges of the graph as a means of utilizing all available sensor data in planning the inspection.**

I. INTRODUCTION

There are numerous practical applications requiring periodic or persistant surveillance of the indoor/outdoor environment and structures within the environment. To automate these tasks, a variety of planning algorithms have been developed to solve the robot coverage problem. For applications in 2D workspaces (e.g., demining, snow plowing, lawn mowing) in which the agent possesses relatively few degrees of freedom, a cellular decomposition of the free space is typically sufficient to achieve coverage [1]. Approaches in which the inspection of a structure is the goal have often modeled the task as an "art gallery problem", which requires computation of the minimum number of guards who can together observe the entire gallery [2]. In addition, some coverage algorithms have implemented planning strategies based on prior knowledge of the specific geometry of the structure being covered, as in car painting [3] and building inspections [4]. Finally, various challenging issues remain unsolved from a computational geometry perspective if highly accurate 3D models are a desired product of the inspection, especially in the absence of *a priori* knowledge [5].

Our application of interest is the inspection of a ship hull by an autonomous underwater vehicle (AUV), in which acoustic sensing is required to characterize the surrounding environment and complete sensor coverage is necessary to locate foreign objects that may have been placed on the hull. A fully-actuated, hovering AUV has been developed for the inspection task, equipped with imaging and bathymetry sonar to perform the inspection and an inertial measurement unit (IMU) and Doppler velocimetry log (DVL) for underwater navigation [6]. Efforts are also underway to use the sonar data itself to improve navigation via simultaneous localization and mapping [7]. With respect to the motion planning problem, a large portion of the hull is relatively flat and can be inspected suitably by an imaging sonar and a planar vehicle trajectory, which a cellular decomposition approach would solve easily. It is the complex structures under the hull, such as the propellers, rudders, and portions with severe curvature, which require nontrivial inspection paths and the use of bathymetry sonar to accurately observe their geometries. To model these structures a discrete representation of the ship hull will be used, allowing our algorithm to utilize a CAD model or a preliminary range scan (such as the bathymetry scan depicted in Figure 1) for the planning task. In addition, five degrees of freedom are available for planning the inspection (surge, sway, heave, yaw, and sonar pitch).

Given the dimensionality of the problem, a sampling-based motion planning algorithm, in which robot configurations are sampled quasi-randomly and used to catalog feasible paths through the workspace, is an appealing alternative to an exhaustive deterministic search of a multi-dimensional configuration space (C-Space). Prior work in sampling-based planning of inspection paths has solved the problem by adopting the aforementioned "art gallery" approach of choosing a set of configurations which offer complete coverage and finding a feasible tour among these nodes [8], [9], [10]. This approach plans using a graph in which sensor information is embedded in the nodes and the edges are used as a means of transit between nodes. We propose instead to use the full set of sensor information obtained from traversing the edges of the graph, modeling the inspection task as a postman problem rather than a traveling salesman problem. The postman framework has been used previously to solve deterministically-designed inspections for point robots in a 2D workspace [11], [12], in which the robot possesses a small field of view relative to the size of the workspace. We adopt this information-efficient approach both to accomodate the small sensor footprint of the bathymetry sonar and to ensure that all views of the structure are utilized in planning the inspection.

In Section II we introduce the algorithm used to construct a dense graph whose traversal is guaranteed to offer complete sensor coverage of the structure of interest. In Section III the method for finding a minimal-cost closed walk on the graph which offers complete sensor coverage is presented. A closed walk is sought to allow cyclical traversal of the inspection path for persistent surveillance. Section IV offers results of the algorithm using a simulation of the Bluefin-

This work was supported by the Office of Naval Research under Grant N00014-06-10043, monitored by Dr. T.F. Swean

B. Englot and F. Hover are with the Department of Mechanical Engineering, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge MA 02139 USA benglot@mit.edu hover@mit.edu

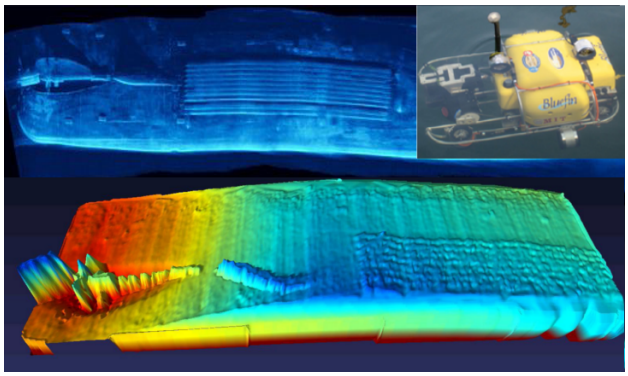

Fig. 1. Imaging and bathymetry sonar data from a survey of a 65' boat are depicted. Included is a photograph of the Bluefin-MIT HAUV, version 1B, which was used to perform the survey. Imaging sonar mosaic courtesy of AcousticView. Bathymetry sonar reconstruction courtesy of Hordur Johannsson, MIT.

MIT Hovering Autonomous Underwater Vehicle (HAUV) and triangle meshes of the marine structures of interest to our application.

II. GRAPH CONSTRUCTION ALGORITHM

Given a triangle mesh which represents the structure to be inspected and the obstacles in the surrounding environment, an iterative algorithm is used to construct a dense graph of feasible configurations (nodes) and feasible paths between them (edges). Although this approach draws inspiration from the well-known Probablistic Roadmap algorithm (PRM) [13], some distinctly different design choices are made to develop a dense graph instead of an expansive, sparse graph.

We denote a robot configuration by $\mathbf{q} \in \Re^N$, where N is the dimension of the configuration space, and a discrete node of the structure mesh by $\mathbf{s} \in \Re^3$. Given a feasible initial configuration $\mathbf{q}_0$, the set of mesh nodes in the structure, referred to hereafter as *Structure*, and the set of triangles belonging to the structure, Algorithm 1, *BUILD_GRAPH*, begins constructing a graph in configuration space using *ADD_TO_GRAPH*(), which is described by Algorithm 2. Configurations $\mathbf{q}_{rand}$ are sampled quasi-randomly (using a Sobol sequence) and Algorithm 4, *FIND_NEIGHBORS*(), is used to connect $\mathbf{q}_{rand}$ to a user-designated number of neighboring configurations which are already on the graph.

Taking $\mathbf{q}_{rand}$ as input, *FIND_NEIGHBORS*() draws straight-line paths through C-Space between $\mathbf{q}_{rand}$ and the designated number of nearest neighbors. Each candidate edge is then checked for collisions with obstacles, and all nodes $\mathbf{s}$ in the structure mesh are checked by *OBSERVATION*() to determine whether they lie inside the workspace volume swept by the robot sensor along the traversal of each collision-free candidate edge. This swept volume will differ depending on the constraints of the robot, and specifically in the case of the HAUV we define a sequence of steps by which the agent travels from one configuration to another. The user may allow the agent to change its position and orientation simultaneously, but for the purpose of obtaining a swept volume which is easy compute and to implement during the underwater inspection we first require the agent to adjust its orientation to that of the adjacent node, and then to move to the position of the node in a separate step. In addition, the swept volume may vary depending on the direction of travel along the edge, and if this is the case the observations along both directions of travel must be cataloged by *OBSERVATION*().

For all $\mathbf{s}$ that lie within the sensor's swept volume, *OBSERVATION*() proceeds to check the line of sight between the mesh node $\mathbf{s}$ and the robot configuration along the candidate edge which sees $\mathbf{s}$. This is achieved using a ray-tracing algorithm which checks for intersection between the line of sight and the triangles of the structure mesh. All structure nodes $\mathbf{s}$ that are observed by an edge of the graph are cataloged, and any structure nodes being observed for the first time are removed from the set *Unobserved*. After the observations of collision-free edges are checked, the collision-free edges are returned by *FIND_NEIGHBORS*().

Once the size of *Unobserved* is diminished to a certain percentage ε of the size of *Structure*, Algorithm 1 begins calling *ADD_MISSING_VIEW*() instead of *ADD_TO_GRAPH*(). *ADD_MISSING_VIEW*(), which is described by Algorithm 3, is designed to locate the remaining handful of structure nodes that have not been seen, once the vast majority of the structure has already been observed by the graph. It does so using *NEWTON_SEARCH*(), which applies Newton's method for root-finding using the robot's Jacobian pseduoinverse to find a configuration $\mathbf{q}_{view}$ which sees the missing structure node $\mathbf{s}_{missing}$. The configuration $\mathbf{q}_{view}$ returned by *NEWTON_SEARCH*() is planted at the center of a new edge in C-Space, and if the swept volume of the edge sees a member of *Unobserved*, the edge's endpoints $\mathbf{q}^1_{view}$ and $\mathbf{q}^2_{view}$ are subjected to the same collision check and observation check used in *FIND_NEIGHBORS*(). To ensure that a consistent diversity of good configurations is returned by *NEWTON_SEARCH*(), every call to this algorithm is initialized with a randomly sampled configuration $\mathbf{q}_{rand}$.

After the entirety of the discrete structure has been observed by the graph's edges, graph construction stops and a closed walk is found which selects an efficient subset of the graph to use as an inspection path.

III. PATH-FINDING ALGORITHM

A. Exact Formulation for the Optimal Inspection Path

First we present the integer programming formulation which solves exactly for the minimum-cost subtour that observes the entire structure. The graph of feasible paths constructed in C-Space is treated as a flow network, in which a unit flow represents a single traversal of an edge by the robot, and each edge of the C-Space graph is modeled as a pair of directed edges. Associated with each directed edge is the set of observations gathered by the agent along the edge, which in some cases may be the same for both edges in the pair, and in other cases each member of the pair may collect different information (depending on the geometry of the robot's sensor and its kinematic constraints). The constraints of the graph are enforced on the flow variables

Algorithm 1 $BUILD\_GRAPH(q_0, Structure)$

$G.init(q_0)\ Unobserved \leftarrow Structure$
while $Unobserved \neq \emptyset$ **do**
  if $(Unobserved.size()/Structure.size()) < \varepsilon$ **then**
    $ADD\_MISSING\_VIEW(G)$
  else
    $ADD\_TO\_GRAPH(G)$
  end if
end while
return G

Algorithm 2 $ADD\_TO\_GRAPH(G)$

$FeasibleNeighbors \leftarrow \emptyset$
while $FeasibleNeighbors = \emptyset$ **do**
  $q_{rand} \leftarrow DRAW\_SAMPLE()$
  $FeasibleNeighbors \leftarrow FIND\_NEIGHBORS(q_{rand})$
  if $FeasibleNeighbors \neq \emptyset$ **then**
    $G.addNode(q_{rand})$
    for $q_i \in FeasibleNeighbors$ **do**
      $G.addEdge(q_{rand}, q_i)$
    end for
  end if
end while

through the incidence matrix A, which contains a row for each node in the graph and a column for each flow variable, with $A_{ij} \in \{0, 1, -1\}$. The observations made along the graph are enforced on the flow variables by the constraint matrix B, which contains a row for each flow variable x and a column for each discrete node **s** in the structure being inspected. Entry B_{jk} is 1 if edge j observes mesh node k, and 0 otherwise. Using this modeling framework, the problem is stated by the following:

$$\begin{aligned} minimize\ & \sum_j c_j x_j \\ subject\ to\ & \sum_j A_{ij} x_j = 0\ \ \forall\, i \\ & \sum_j B_{jk} x_j \geq 1\ \ \forall\, k \\ & \sum_j (A_{1j})^2 x_j \geq 2 \\ & x_j\ integer\ \ \forall\, j \end{aligned} \tag{1}$$

The indices i, j, and k refer to the graph's nodes, flow variables, and the discrete structure nodes, respectively. The cost c_{ij} applied to each flow variable may represent the time or energy consumed by the agent while traversing the edge represented by flow variable x_j. The first constraint enforces the structure of the C-space graph, the second requires that every node in the structure is observed by the agent at least once, and the third requires that part of the closed walk must pass through the node on which the graph was initialized (designated $i = 1$), which is assumed to be the configuration from which the agent starts the inspection.

In its current form, (1) may allow the minimum-cost flow

Algorithm 3 $ADD\_MISSING\_VIEW(G)$

$foundMissingView \leftarrow false$
$FeasibleNeighbors_1 \leftarrow \emptyset, FeasibleNeighbors_2 \leftarrow \emptyset$
$s_{missing} \leftarrow Unobserved.getFirst()$
while $foundMissingView = false$ **do**
  $q_{rand} \leftarrow DRAW\_SAMPLE()$
  $q_{view} \leftarrow NEWTON\_SEARCH(q_{rand}, s_{missing})$
  $\{q^1_{view}, q^2_{view}\} \leftarrow DRAW\_EDGE(q_{view})$
  if $FIND\_VIEW(q^1_{view}, q^2_{view}) = true$ **then**
    $FeasibleNeighbors_1 \leftarrow FIND\_NEIGHBORS(q^1_{view})$
    $FeasibleNeighbors_2 \leftarrow FIND\_NEIGHBORS(q^2_{view})$
    if $FeasibleNeighbors_i \neq \emptyset\ \forall\, i \in \{1, 2\}$ **then**
      $foundMissingView \leftarrow true$
      $G.addEdge(q^1_{view}, q^2_{view})$
      for $q^i_{view} \in \{q^1_{view}, q^2_{view}\}$ **do**
        $G.addNode(q^i_{view})$
        for $q_j \in FeasibleNeighbors_i$ **do**
          $G.addEdge(q^i_{view}, q_j)$
        end for
      end for
    end if
  end if
end while

Algorithm 4 $FIND\_NEIGHBORS(q_{rand})$

$FeasibleNeighbors \leftarrow NEAREST\_NEIGHBORS(q_{rand})$
for $q_i \in FeasibleNeighbors$ **do**
  if $COLLISION(q_{rand}, q_i) = false$ **then**
    for $s_j \in Structure$ **do**
      if $OBSERVATION(q_{rand}, q_i, s_j) = true$ **then**
        $Unobserved \leftarrow Unobserved \backslash \{s_j\}$
      end if
    end for
  else
    $FeasibleNeighbors \leftarrow FeasibleNeighbors \backslash \{q_i\}$
  end if
end for

to consist of several unattached closed walks on the graph rather than a single, continous closed walk. To eliminate this possibility, additional constraints must be included in the problem formulation. Unfortunately, the quantity of constraints required is exponential in the number of nodes in the C-Space graph, since the flow through every combination of nodes in the graph must be inspected to eliminate subtours. For this reason it is impractical to enumerate all of these constraints to solve the problem exactly, and an approximate solution will be found using only (1) and not the exponentially many subtour elimination constraints.

The closest relative of this problem which has been classified in the operations research literature is known as the Prize-Collecting Rural Postman Problem (PRPP) [14], an NP-hard problem in which the agent must find the minimum-cost closed walk when there is a prize located on each edge of the graph that is collected only on the first traversal. Our

Algorithm 5 $CONNECT\_SUBTOURS(Subtours, q_{init}, G)$

$q_1 \leftarrow q_{init}$
$t_1 \leftarrow GET\_SUBTOUR\_CONTAINING(q_1)$
$EligibleSubtours \leftarrow Subtours \backslash t_1$
$ConnectingPaths \leftarrow \emptyset$
while $ConnectingPaths.size() < Subtours.size() - 1$ **do**
  $\{t_2, q_2\} \leftarrow NEAREST(t_1, q_1, EligibleSubtours)$
  $ConnectPath \leftarrow DIJKSTRA(q_1, q_2, G)$
  $ConnectingPaths.add(ConnectPath)$
  $EligibleSubtours \leftarrow EligibleSubtours \backslash t_2$
  $q_1 \leftarrow q_2$, $t_1 \leftarrow t_2$
end while
$ConnectFinalPath \leftarrow DIJKSTRA(q_2, q_{init}, G)$
$ConnectingPaths.add(ConnectFinalPath)$
return $ConnectingPaths$

agent also collects a one-time prize from certain edges when it sees a structure node for the first time, but in our case the collection of all prizes is a hard constraint and not simply an addition to the reward. This leaves the inspection problem with a larger number of constraints than the PRPP.

B. Approximate Formulation for the Optimal Inspection Path

An efficient closed walk for the structure inspection may be found using (1) only, but to do so the resulting unattached subtours must be connected together into a continuous closed walk. For this purpose, we have designed Algorithm 5, $CONNECT\_SUBTOURS()$, which iteratively fuses each subtour to a neighboring subtour. This algorithm is called when there are two or more subtours returned by the solution to (1), and it takes as input a list of the subtours, the configuration $\mathbf{q_{init}}$ from which the inspection is initialized, and the C-Space graph. Given a starting configuration $\mathbf{q_1}$, this algorithm chooses the nearest configuration $\mathbf{q_2}$ on a yet unvisited neighboring subtour. Dijkstra's algorithm is used to find a feasible path connecting $\mathbf{q_1}$ and $\mathbf{q_2}$, and $\mathbf{q_2}$ is set to $\mathbf{q_1}$ for the next iteration. This forces a subtour to connect to two neighbors through a single point of attachment, which simplifies the execution of the closed walk. In this manner, the inspection will begin at $\mathbf{q_{init}}$, carry out the subtour to which $\mathbf{q_{init}}$ belongs, and once the agent arrives back at $\mathbf{q_{init}}$, it will proceed to the next subtour. This pattern repeats, carrying out the next subtour and departing from the same configuration through which it arrived.

IV. INSPECTION PLANNING FOR THE HAUV

A. Platform-Specific Constraints

To implement the inspection planning algorithms for structure inspection by the Bluefin-MIT HAUV, some platform-specific constraints must be formulated. Although the vehicle is capable of motion in surge, sway, heave, yaw, and sonar pitch, we assume that the vehicle always translates using a combination of surge and heave only, since this constraint will always sweep the largest possible volume with the bathymetry sonar, which is mounted with the beam horizontal at zero sonar pitch. Configurations will be sampled in x, y, z, and sonar pitch, with the size of the sampling space in x, y, and z dictated by the size of the inspection environment, and sampling in sonar pitch dictated by the allowable limits of sensor motion (sensor can pitch 90 degrees above or below the horizontal plane). The graph edge connecting any two configurations will dictate the vehicle yaw, since translation is constrained to occur in the surge and heave directions only. For every edge in C-Space added to the graph, we construct two directed edges, each in which the vehicle starts at a node, adjusts its sonar pitch angle, turns in yaw to face the adjacent node, then translates in surge and heave to the adjacent node. Each directed edge sweeps out a different volume of Euclidean space than its oppositely directed partner. For the purposes of the minimum-cost flow optimization problem, the cost of each edge of the graph is set to the length of the edge in Euclidean space. Pitching the sonar is modeled as a cost-free action since it can occur quickly and with low energy expenditure. Figure 2 illustrates the observation of some structure mesh nodes during an edge traversal. We assume the sonar has a 30 degree field of view, a minimum range of 1m, and a maximum range of 10m.

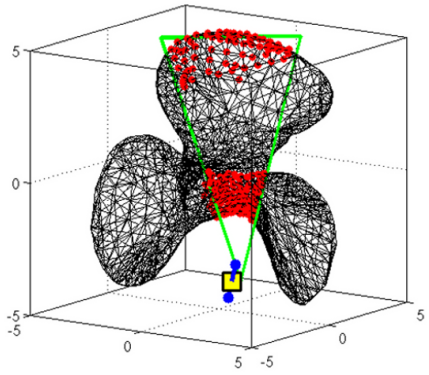

Fig. 2. Visual depiction of the sweeping of the bathymetry sonar footprint across an edge of the graph. The sensor footprint is drawn in green, and the discrete structure nodes observed by the traversal of the blue graph edge are plotted in red.

If we approached this problem from the "art gallery" perspective, sensor observations collected at the nodes of the C-Space graph would be the only observations stored. This could be a viable strategy for the HAUV if a nodding maneuver is performed with the sonar at each graph node, but it discards the valuable sensor information gathered while traveling between nodes. Because of the limited field of view of the bathymetry sonar and the increased expense of planning a tour on a larger graph, we have chosen not to discard this information.

B. Results

The inspection planning algorithms were applied to a variety of structure meshes, which are pictured in Figure 3. The meshes were scaled to compare with the largest-sized structures we will inspect with bathymetry sonar during an autonomous ship hull inspection. In the cases of the actual ship structures (the propeller with nothing attached

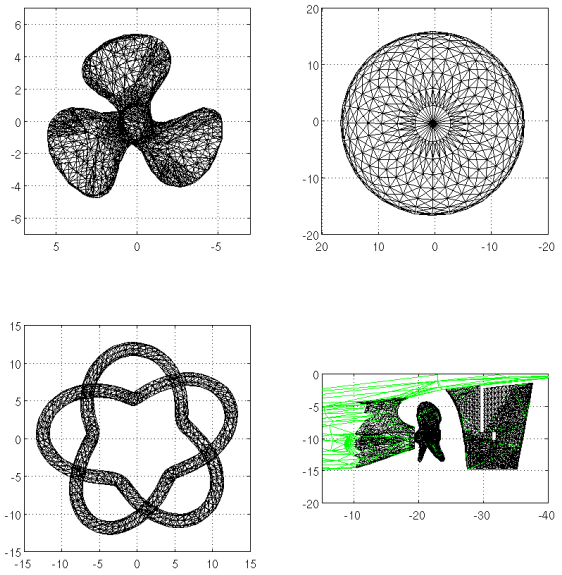

Fig. 3. Four discrete structure models to which the inspection planning algorithms are applied. The structures are a ship propeller, a sphere, a set of interwoven tubes, and the propeller and rudder of a large ship, which includes the complete ship model (in green) for the purposes of obstacle avoidance. Dimensions are in meters.

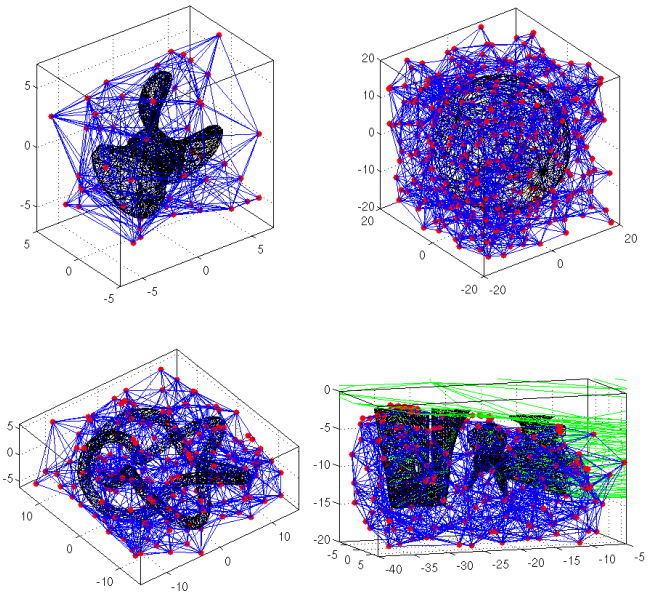

Fig. 4. The dense graphs constructed for the four example structures of Figure 3. Each graph is constructed in a 4D configuration space, and is projected here in 3D Euclidean space. The limits of the space in which samples were drawn are identical to the boundaries of the four plots.

and the propeller and rudder attached to a hull), the nodes of the structure mesh were increased in density such that the maximum distance between structure nodes occupying the same triangle was about 0.5m. This spacing between the discrete nodes of the structure model should be chosen based on the size of objects the inspection is intended to detect. Although we increased the density of nodes, the original, larger triangles were used for collision checking and ray tracing. For the larger ship mesh, we only require inspection of the structures possessing severe curvature, and the remainder of the ship is stored for the purpose of collision detection and ray tracing. Figure 4 displays the graphs that were constructed to achieve 100% coverage of each structure. For each graph, the minimum-cost closed walk selected as the inspection path is illustrated in Figure 5.

The only tuning of the algorithm which varied between structures were the boundaries in C-Space from which we sampled the x, y, and z coordinates of each configuration. Otherwise, the same algorithm settings, including a choice of ten nearest neighbors for the number of connections to make in *FIND_NEIGHBORS*(), and the use of Algorithm 3 for the final 5% of the structure, were used for all results depicted here. IBM's CPLEX solver was used to solve the integer programming problem for the minimum-cost flow. Data on computation time and the sizes of the structure meshes and the C-Space graphs is presented in Table 1. The number of nodes in each graph scaled approximately with the size of each respective structure. Although the propeller mesh was densely populated with discrete nodes, the algorithm solved for an inspection path quickly and required a relatively sparse graph to achieve 100% sensor coverage. The other structures required dense graphs by comparison, and the sphere, the most sparsely populated structure model, produced the graph of greatest density due to the ease of connecting each node to its ten closest neighbors without obstruction. The large ship mesh, because of the narrow passages between structures, required significantly more time than the other examples to complete graph construction. Most of this time was spent using Algorithm 3 to search for feasible configurations which observed the structure nodes in the narrow cracks in and around the rudder, which are approximately 0.5m wide in most locations. Although the tight spaces in this ship model posed a challenge, as they do for most planning algorithms, the resulting inspection path yielded an intuitively curved trajectory to explore the narrow space between the upper part of the propeller and the hull.

To offer a benchmark for comparison with the results of [8], the prior work most similar to ours, we also apply the algorithm to a set of randomly chosen cube structures, each of which is comprised of five nodes and four triangles on each face, the result of which is plotted in Figure 6. Although the sensor used in our inspections has a rather limited field of view compared with that of [8], an inspection of comparable path length and structure is obtained when inspecting the simpler, dispersed 3D structures assessed in this prior work.

V. CONCLUSION

We have introduced an algorithm for planning inspection paths which iteratively constructs a dense graph in the agent's

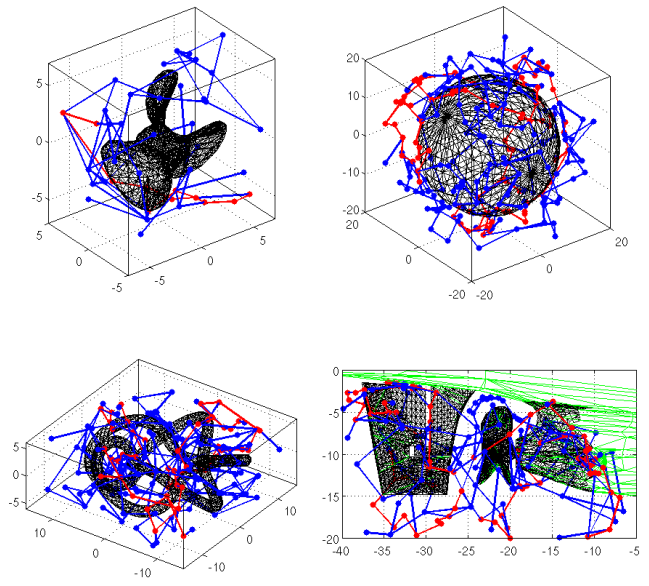

Fig. 5. Inspection paths planned for the four example structures of Figure 3. Blue edges represent the subtours required for coverage, and red edges represent the paths selected by Algorithm 5 to connect them.

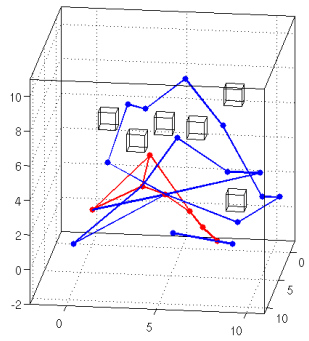

Fig. 6. An inspection path planned for six randomly planted 1-m cubes, intended for comparison of our algorithm with the result of [8]. Blue edges represent the subtours required for coverage, and red edges represent the paths selected by Algorithm 5 to connect them.

configuration space until the set of edges observes the entire structure. A sampling-based approach is favored due to the need to explore four dimensions comprehensively, and the desire to avoid doing so exhaustively. To accomodate 3D structures of arbitrary complexity we assume a discrete model of the structure, which can utilize models produced by CAD software or produced by previously collected data. After constructing the graph, we approximate the minimum-cost closed walk along the graph which observes the entire structure.

Although this algorithm successfully achieves coverage planning over arbitrary discrete 3D structures and does so using an information-efficient postman formulation, there are desired improvements and extensions. We would like to

TABLE I

ALGORITHM PERFORMANCE FOR THE EXAMPLE STRUCTURES

	Propeller	Sphere	Tubes	Ship
Structure Nodes	1922	482	1280	3771
Structure Triangles	712	960	2562	2460
Graph Nodes	194	785	585	958
Graph Edges	806	6014	2786	4078
95% Coverage Time [sec]	32	70	56	187
100% Coverage Time [sec]	62	108	142	2792
IP Solution Time [sec]	2	110	97	50
Alg. 5 Solution Time [sec]	3	71	24	61
# Steps in Subtours Only	74	295	256	269
# Steps in Inspection Path	84	352	305	330
Length of Inspection Path [m]	351	1779	1331	1005

reduce computation time by searching more efficiently for feasible views of the last few unobserved structure nodes, the step which is usually most time-consuming. We also wish to quantify and, if possible, optimally manage the division of complexity between the graph construction and path-finding steps in planning problems of this type, in which an agent must connect with many thousands of targets rather than execute a simple point-to-point path.

REFERENCES

[1] H. Choset, "Coverage for robotics - A survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, 2001, pp. 113-126.

[2] J. O'Rourke, *Art Gallery Theorems and Algorithms*, New York: Oxford University Press, 1987.

[3] P. Atkar, A.L. Greenfield, D.C. Conner, H. Choset, and A. Rizzi, "Uniform Coverage of Automotive Surface Patches" *Int. J. Robotics Research*, vol. 24(11), 2005, pp. 883-898.

[4] P. Cheng, J. Keller, and V. Kumar, "Time-Optimal UAV Trajectory Planning for 3D Urban Structure Coverage," *Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, Nice, France, 2008, pp. 2750-2757.

[5] W. Scott and G. Roth, "View Planning for Automated Three-Dimensional Object Reconstruction and Inspection," *ACM Computing Surveys*, vol. 35(1), 2003, pp. 64-96.

[6] F. Hover, et al., "A Vehicle System for Autonomous Relative Survey of In-Water Ships," *Marine Technology Society Journal*, vol. 41(2), 2007, pp. 44-55.

[7] H. Johannsson, M. Kaess, B. Englot, F. Hover, and J. Leonard, "Imaging Sonar-Aided Navigation for Autonomous Underwater Harbor Surveillance," *Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, 2010, to appear.

[8] T. Danner and L. Kavraki, "Randomized Planning for Short Inspection Paths," *Proc. IEEE Int. Conf. on Robotics and Automation*, San Francisco, 2000, pp. 971-976.

[9] M. Saha, G. Sanchez-Ante, T. Roughgarden, and J.C. Latombe, "Planning Tours of Robotic Arms Among Partitioned Goals," *Int. J. Robotics Research*, vol. 25(3), 2006, pp. 207-223.

[10] P. Wang, R. Krishnamurti, and K. Gupta, "View Planning Problem with Combined View and Traveling Cost," *IEEE Int. Conf. on Robotics and Automation*, Rome, 2007, pp. 711-716.

[11] K. Easton and J. Burdick, "A Coverage Algorithm for Multi-robot Boundary Inspection," *IEEE. Int. Conf. on Robotics and Automation*, Barcelona, 2005, pp. 727-734.

[12] K. Williams and J. Burdick, "Multi-robot Boundary Coverage with Plan Revision," *IEEE. Int. Conf. on Robotics and Automation*, Orlando, 2006, pp. 1716-1723.

[13] L. Kavraki, P. Svestka, J.C. Latombe, and M. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Trans. on Robotics and Automation*, vol. 12(4), 1996, pp. 566-580.

[14] J. Araoz, E. Fernandez, and O. Meza, "Solving the Prize-collecting Rural Postman Problem," *European J. of Operational Research*, vol. 196, 2009, pp. 886-896.