Hierarchical Coverage Path Planning in Complex 3D Environments

Conference Paper · May 2020

CITATIONS
0

READS
217

4 authors, including:

Ji Zhang
Carnegie Mellon University
35 PUBLICATIONS **1,526** CITATIONS
SEE PROFILE

Matthew Travers
Carnegie Mellon University
59 PUBLICATIONS **517** CITATIONS
SEE PROFILE

Howie Choset
Carnegie Mellon University
295 PUBLICATIONS **6,607** CITATIONS
SEE PROFILE

Hierarchical Coverage Path Planning in Complex 3D Environments

Chao Cao, Ji Zhang, Matt Travers and Howie Choset

***Abstract*— State-of-the-art coverage planning methods perform well in simple environments but take an ineffectively long time to converge to an optimal solution in complex three-dimensional (3D) environments. As more structures are present in the same volume of workspace, these methods slow down as they spend more time searching for all of the nooks and crannies concealed in three-dimensional spaces. This work presents a method for coverage planning that employs a multi-resolution hierarchical framework to solve the problem at two different levels, producing much higher efficiency than the state-of-the-art. First, a high-level algorithm separates the environment into multiple subspaces at different resolutions and computes an order of the subspaces for traversal. Second, a low-level sampling-based algorithm solves for paths within the subspaces for detailed coverage. In experiments, we evaluate our method using real-world datasets from complex three-dimensional scenes. Our method finds paths that are constantly shorter and converges at least ten times faster than the state-of-the-art. Further, we show results of a physical experiment where a lightweight UAV follows the paths to realize the coverage.**

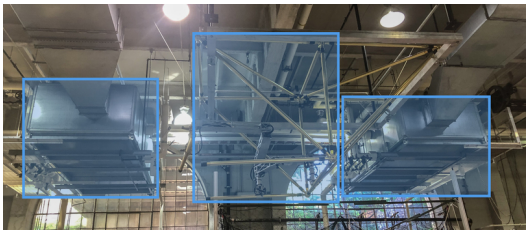
I. Introduction

We present an algorithm to determine a path for a robot with a limited-filed-of-view sensor, e.g. a camera, to perceive all surface areas in a target environment. Such a problem is often called a coverage planning problem because the "sensory footprint" covers the entire reachable areas in the space, as if we were painting the surfaces with the sensor. The problem seeks the shortest path, which will be followed by the robot to realize the coverage.

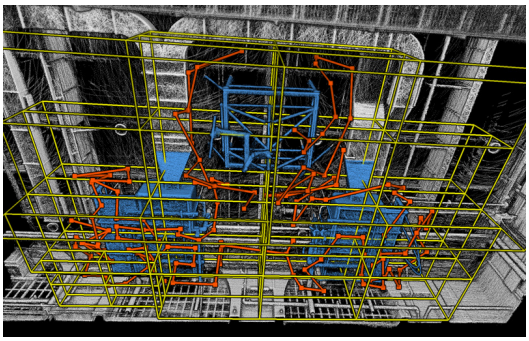
Coverage planning problems have been addressed in the plane and generally use a pattern, e.g. a raster scan [1], and a decomposition [2] to ensure complete coverage is obtained. Three-dimensional coverage approaches have also been considered for applications such as three-dimensional printing of objects [3]. Finally, 2.5-D coverage or surface coverage has been considered for application such as paint deposition where the challenge lies in ensuring uniform coverage of a material on a non-flat surface embedded in three-dimensions, hence 2.5-D [4]. At the risk of abusing nomenclature, we define the work in this paper as three-dimensional because the robot truly has to fly through three dimensions to ensure coverage of the target surfaces.

The state-of-the-art methods [5], [6] in solving this type of problems benefit from random sampling, which gives the shortest collision-free path found within a given and relatively short amount of time. The methods draw random samples along the surfaces and connect the samples to form a path, by solving a Traveling Salesman Problem (TSP) [7].

All authors are with the Robotics Institute at Carnegie Mellon University, Pittsburgh. Emails: {ccao1, zhangji, mtravers, choset}@andrew.cmu.edu



(a)



(b)

Fig. 1: Example result. (a) shows a photo of the environment. The structures to cover are highlighted by the blue rectangles. (b) presents the coverage planning result where the structures to cover are in blue. The yellow boxes are the boundaries of the subspaces. The orange path is the coverage path.

Due to redundant occlusion/collision checks in the sampling process and the time on TSP solving, these methods slow down significantly in large, complex environments.

Our method extends the state-of-the-art by employing a multi-resolution hierarchical framework. In this framework, a high-level algorithm separates the space into multiple subspaces at different resolutions (see an example result in Fig. 1). If there is a large number of structures in a given volume, the space is further subdivided at a finer resolution. A low-level, sampling-based algorithm solves for a path within each subspace for detailed coverage. In experiments, the method finds paths that are constantly shorter than the state-of-the-art and converges at least ten times faster as well. Further, we use a lightweight UAV to follow the paths and record image streams to realize the coverage. We make available a video of our experimental results[1].

II. Related Work

Conventional coverage planning frameworks typically employ pre-defined path patterns e.g. raster scans, to traverse the environment [1], [8]–[10]. These primitive-based methods

[1]Experiment video: https://youtu.be/u6UYarMNPcA

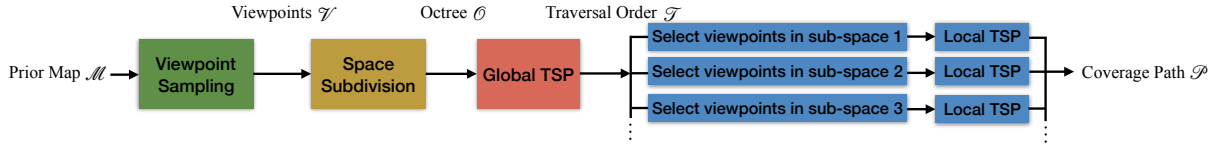


Fig. 2: Processing flowchart.

work well in 2D [11], e.g. for agriculture and cleaning robots, and later are extended to surface coverage [4], [12], e.g. for robotic painting. To deal with obstacles, the Boustrophedon decomposition method models obstacles as polygons and separates the space into multiple parallel subspaces at the vertices of the polygons [2], [13]. Overall, due to the usage of pre-defined path patterns, these methods are limited to structured and relatively simple environments.

The coverage planning problem can also be solved as an orienteering problem [14]. The problem solves for sensor poses and a tour connecting the sensor poses to realize the coverage. For example, the method of Roberts et al. solves the problem in two steps – first compute the sensor orientation for each candidate sensor pose, then select a subset of the sensor poses and find the order by maximizing an award function as an integer linear programming problem [15]. Arora and Scherer propose a random sampling-based method combining the constraint satisfaction problem and TSP [16]. In summary, these methods can handle unstructured environments. However, due to the fact that the problem does not model occlusions and collisions, they have difficulty with highly complex 3D structures.

The method put forward in this work is most related to the work of Hollinger et al. [5] and Bircher et al. [6]. The two methods share great similarity and are considered state-of-the-art. The methods first select a set of sensor poses for a complete coverage through a random sampling process. Then, a tour is formed by solving TSP. This process repeats while the cost of the tour reduces through iterations. In addition, the method of Dornhege et al. solves the problem as a set cover problem and TSP [17]. Because of the complex nature of the problem, the runtime of these methods increases drastically w.r.t. the complexity of the environment.

Our work draws inspiration from [5] and addresses issues by adopting a hierarchical framework. The problem is solved at two levels. The overall traversal is solved at a high level and the detailed coverage is solved at a low level. This way, the problems of detailed coverage planning are kept small, improving the efficiency. The method is significantly faster than the state-of-the-art in complex 3D environments.

III. Methodology

Our method starts with sampling camera viewpoints in the overall environment (Fig. 2). Then, the environment is divided into multi-resolution subspaces and a global TSP is run to solve for a tour through the subspaces. Later, viewpoints are selected and the detailed coverage paths are computed in the individual subspaces to form the path.

A. Viewpoint Sampling

A viewpoint is described as a 6-tuple $\vec{v} = [x_c, y_c, z_c, \phi_c, \theta_c, \psi_c]$, where x_c, y_c, z_c denote the camera position, and ϕ_c, θ_c and ψ_c denote the roll, pitch, and yaw angles. Given a surface patch, the camera poses for viewing the surface patch is set at distance D away, determined by specifications of the camera and dimensions of the vehicle. Let $\vec{n} = [n_x, n_y, n_z]$ be the surface normal. We set the roll angle $\phi_c = 0$ in practice which limits the size of the search space. The pitch and yaw angles oriented orthogonal to the surface patch are given by

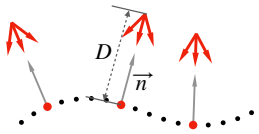


Fig. 3: Sampling camera viewpoints. The dotted curve indicates a surface. The red dots are sub-sampled points. The grey arrows show the normal directions on the surface. The red arrows are the sampled viewpoints which are D away from the surface in a neighborhood region of θ^* and ψ^*

$$\begin{aligned} \theta^* &= \arctan(n_z, \sqrt{n_x^2 + n_y^2}) \\ \psi^* &= \arctan(-n_y, -n_x) \end{aligned} \tag{1}$$

Viewpoints are sampled in a neighborhood region of θ^* and ψ^* along the surfaces, as illustrated in Fig. 3.

We use a pin-hole camera model to project the 3D points from the surfaces onto the image plane. We explicitly reason about the range, occlusions, and field of view of the camera. Points that appear on the image plane are then aggregated to calculate the coverage area of v. After sampling all viewpoints, collision check is run to eliminate the viewpoints colliding with structures in the environments. The collision check considers the vehicle dimensions. The set of feasible viewpoints is denoted as $\mathcal{V}$. An example is shown in Fig. 4 where the green arrows indicate the feasible viewpoints. Note that this is the same environment as in Fig. 1.

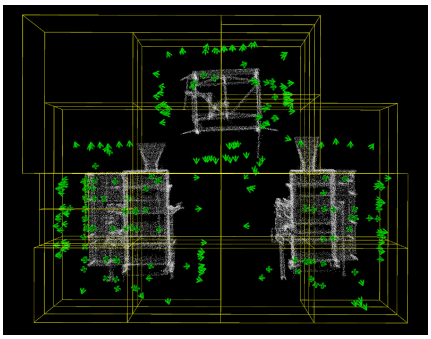


Fig. 4: Feasible viewpoints (green arrows) and subspaces (yellow boxes) in the same environment as in Fig. 1.

Algorithm 1: $SubdivideSpace$

input : Feasible viewpoints $\mathcal{V}$

$\mathcal{V}_{subspace} \leftarrow$ InsertViewpointsToLeaves($\mathcal{V}$);
if $|\mathcal{V}_{subspace}| \leq N$ **then**
  $is\_leaf \leftarrow true$;
  Return;
else
  $is\_leaf \leftarrow false$;
  for $i \in [0, 7]$ **do**
    $children[i] \leftarrow$ new OctreeNode(i);
    $children[i].SubdivideSpace(\mathcal{V}_{subspace})$;
  end
end

B. Space Subdivision

The feasible viewpoints $\mathcal{V}$ are stored in a multi-resolution Octree [18] which is used to divide the environment into subspaces. Each leaf in the Octree is associated with a local cluster of viewpoints. We adopt a multi-resolution scheme in building the Octree because in an irregular environment, evenly dividing the environment can result in certain subspaces to have significantly more viewpoints than other subspaces. The Octree construction is presented in Algorithm 1. We use $N \in \mathbb{Z}^+$ to denote the maximum number of viewpoints allowed in a subspace. The depth of the Octree varies depending on the complexity of the enclosed structures. Areas with complex structures are subdivided further until all subspaces contain no more than N viewpoints.

C. Global TSP

With the environment divided into subspaces, the global TSP seeks to solve for a tour through the subspaces. The detailed coverage paths in the subspaces are then computed based on the global TSP tour. Algorithm 2 presents the coverage planning algorithm. On line 2-3, we construct an Octree to obtain a list of non-empty leaves where each leaf contains a set of viewpoints. On line 5, we use the TSP solver in Google OR-Tools [19] to determine the visiting sequence of the subspaces. The TSP solver takes as input a distance matrix containing pairwise distances between the subspaces. The distance matrix is calculated on line 4 where the distance between two subspaces is the Euclidean distance between the centroids of the enclosed viewpoints. If no feasible path is found, however, the distance is set to infinity.

D. Local Coverage Planning

We solve a coverage planning problem within in each subspace to find the shortest path that covers the surfaces. Similar to [15], the problem exhibits submodularity where the marginal reward of selecting a new viewpoint decreases as more viewpoints have been selected. Specifically, let $\mathcal{A}^i = \{\bigcup_{v_j \in \mathcal{V}_i} \mathcal{A}^i_j\}$ be the area covered after traversing the i-th subspace following the traversal order $\mathcal{T}$. Depending on the viewpoint distribution, part of $\mathcal{A}^i$ may have been covered

Algorithm 2: CoveragePlanning

input : Prior map $\mathcal{M}$
output: Coverage path $\mathcal{P}$

1 $\mathcal{V} \leftarrow$ SampleViewpoints($\mathcal{M}$);
2 $\mathcal{O} \leftarrow$ new OctreeNode();
3 $\mathcal{O}$.SubdivideSpace($\mathcal{V}$);
4 $\mathcal{D} \leftarrow$ ComputeDistanceMatrix($\mathcal{O}$);
5 $\mathcal{T} \leftarrow$ SolveTSP($\mathcal{D}$);
6 $t \leftarrow \mathcal{T}$.begin();
7 **while** $t \neq \mathcal{T}$.*end()* **do**
8 $\mathcal{T}^t_{best} \leftarrow \emptyset$, $l^t_{best} \leftarrow \infty$;
9 $i \leftarrow 0$;
10 **while** $i < K$ **do**
11 $\hat{\mathcal{V}}^t_{subspace} \leftarrow$ SelectViewpoints($\mathcal{V}^t_{subspace}$);
12 $\hat{\mathcal{D}}^t \leftarrow$ ComputeDistanceMatrix($\hat{\mathcal{V}}^t_{subspace}$);
13 $\mathcal{T}^t \leftarrow$ SolveTSP($\hat{\mathcal{D}}^t$);
14 **if** Length($\mathcal{T}^t$) $< l^t_{best}$ **then**
15 $\mathcal{T}^t_{best} \leftarrow \mathcal{T}^t$, $l^t_{best} \leftarrow$ Length($\mathcal{T}^t$);
16 **end**
17 $i \leftarrow i + 1$;
18 **end**
19 $\mathcal{P} \leftarrow \mathcal{P}$.append($\mathcal{T}^t_{best}$);
20 $t \leftarrow \mathcal{T}$.next();
21 **end**

before traversing $\mathcal{A}^i$. The coverage area of the i-th subspace is adjusted to $\hat{\mathcal{A}}^i = \mathcal{A}^i - \cup(\mathcal{A}^j \cap \mathcal{A}^i)$, $j \in \mathcal{T}$ and $j < i$.

The problems inside the subspaces are solved on lines 7-21 in Algorithm 2. The processing has two main steps: 1) select a min-cardinality set of viewpoints on line 11, and 2) compute a TSP tour to connect the selected viewpoints on line 13. The first step is further explained in Algorithm 3. The algorithm uses a priority queue to keep track of candidate viewpoints. On line 4, when a viewpoint is pushed into the queue, its priority is set to the size of its coverage area. On line 6, viewpoints are selected at the probabilities proportional to their priorities. Due to the submodularity of this problem, when a viewpoint is selected, the rewards of selecting other viewpoints need to be properly decreased on lines 8-10. The processing adds more viewpoints as neighbors of the current viewpoint on lines 11-18 until no more viewpoint with admissible reward is available on lines 19-21. In the second step, when computing the TSP tour, the start and end viewpoints are chosen to be the closest to the end viewpoint of the previous subspace and the start viewpoint of the next subspace, respectively. The two steps of processing repeats for a number of $K \in \mathbb{Z}^+$ iterates. Then, the shortest paths from the subspaces are concatenated to form the overall coverage path on line 20 in Algorithm 2.

To warrant a collision-free path, we use the insight of lazy collision checking [20] where collision check is performed after a path is found. If an edge connecting two viewpoints are in collision with structures in the environment, a new viewpoint is added at the point of collision. The process repeats until a collision-free path is found or the maximum

Algorithm 3: SelectViewpoints

input : Viewpoints in subspace $\mathcal{V}_{subspace}$
output: Subset $\hat{\mathcal{V}}_{subspace}$ with a full coverage

$Q \leftarrow$ new PriorityQueue();
$v \leftarrow$RandomPick($\mathcal{V}_{subspace}$);
$\mathcal{A}_v \leftarrow$ ComputeCoverageArea(v);
Q.AddWithPriority(v, $\mathcal{A}_v$);
while $Q \neq \emptyset$ **do**
  $v, \mathcal{A}_v \leftarrow$ ProbabilisticPick(Q);
  $\hat{\mathcal{V}}_{subspace} \leftarrow \hat{\mathcal{V}}_{subspace} \cup v$;
  for $q \in Q$ **do**
    Q.UpdatePriority(q, $\mathcal{A}_v$)
  end
  for $u \in$ *Neighbor*(v) **do**
    if *NotSelected*(u) **then**
      $\mathcal{A}_u \leftarrow$ ComputeCoverageArea(u);
      if $\mathcal{A}_u \geq \mathcal{A}_{min}$ **then**
        Q.AddWithPriority(u, $\mathcal{A}_u$);
      end
    end
  end
  if $Q.front().priority < \mathcal{A}_{min}$ **then**
    break;
  end
end

iteration number is met. In the second case, the distance between the two viewpoints is set to infinity.

E. On Complexity

Let us inspect the time complexity of the proposed method. We use space subdivision to keep the coverage planning problems in small scales. Given that each subspace contains no more than N viewpoints, the runtime of solving TSP in a subspace is within a constant time. With $k \in \mathbb{Z}^+$ non-empty subspaces, the accumulative runtime of all subspaces becomes $O(k)$. In addition, the TSP solver based on the LinKernighan heuristic to solve the global TSP runs in $O(k^{2.2})$ time [21]. Considering both, the time complexity of solving all TSPs is $O(k^{2.2})$. In compassion, if without space subdivision, solving TSP with all viewpoints in the environment takes $O(n^{2.2})$ time, where $n \in \mathbb{Z}^+$ is the number of viewpoints, $n >> k$. Another consideration is that the viewpoint selection process takes redundant occlusion and collision checks. The process is more efficient if kept in a small scale. In Section IV, our results show that the runtime of our method is significantly faster than the state-of-the-art in both the viewpoint selection and TSP solving processes.

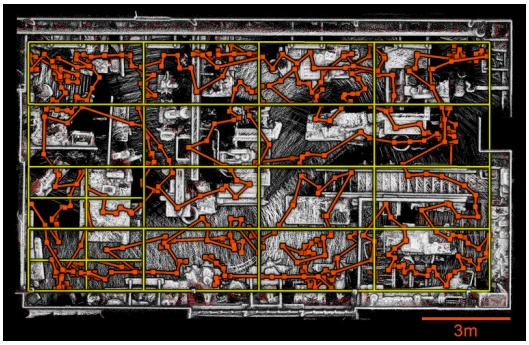
IV. Experiments

A. Path Generation Tests

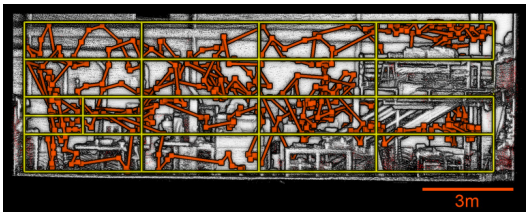
The experiment evaluation compares the proposed method to three other methods listed as follows.

- *Baseline*: This the method of Hollinger et al. [5] which is considered the state-of-the-art.
- *No Heuristic*: This is a configuration of our method that does not use the priority queue in Algorithm 3 to associate the viewpoints with different probabilities. All viewpoints are selected at an equal probability.
- *No Space Subdivision*: This is another configuration of our method that does not use space subdivision. The problem is solved as a whole in the entire space.

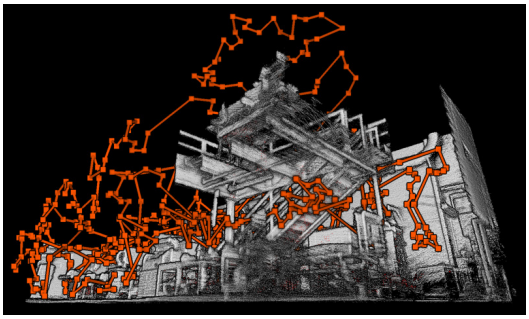
We show results of three path generation tests in Fig. 5-Fig. 7, with a machine shop, an aircraft, and a bridge, respectively. We set the camera field of view at 60° and keep it 1m away from the surfaces. The yellow boxes represent



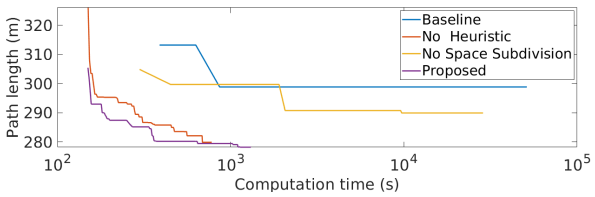
(a)



(b)

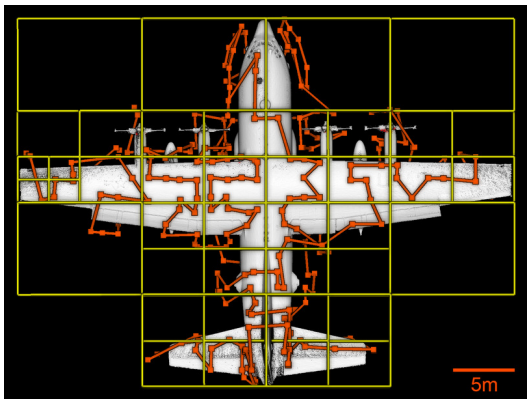


(c)

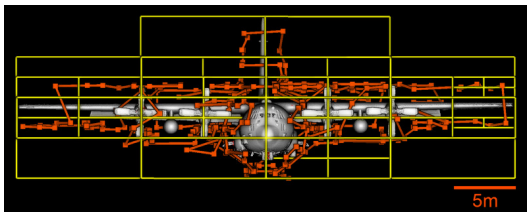


(d)

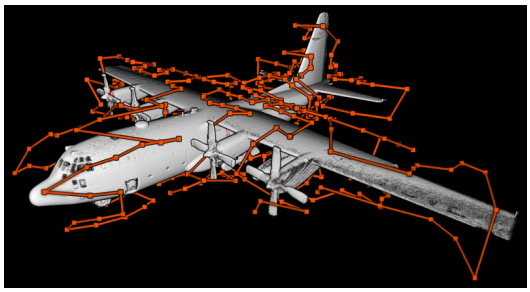
Fig. 5: Machine shop test result. In (a)-(c), the yellow rectangles indicate the subspaces. The orange path is the coverage path and the dots on the path are viewpoints. A number of 1484 viewpoints are selected from 12160 sampled viewpoints. The red points represent uncovered areas due to occlusions in tight areas. (d) is runtime comparison.



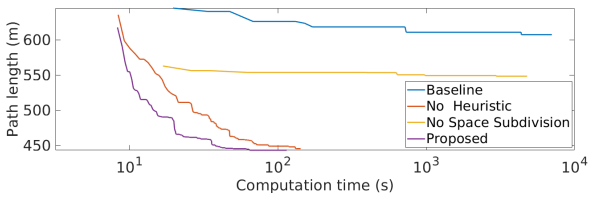
(a)



(b)



(c)



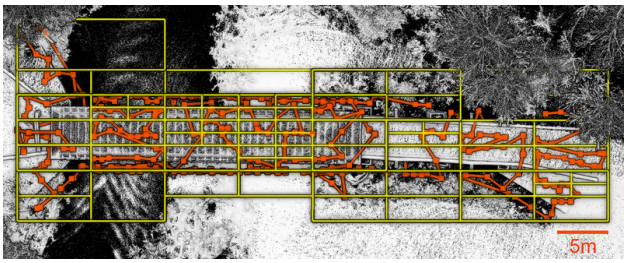
(d)

Fig. 6: Aircraft test result. The figure shares the same layout as Fig. 5. 1089 out of 3675 viewpoints are selected.

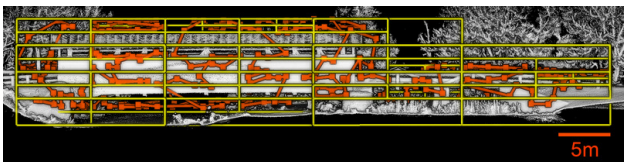
the sub-spaces. The orange paths are the coverage paths and the dots on the paths are viewpoints. At the bottom of each figure, we compare the runtime of our method to the other three methods as listed above. This uses a laptop computer with a 4.4GHz CPU running Linux. The processing consumes a single CPU thread. Our method converges over a magnitude faster than the baseline method and at the same time generates shorter paths. Further, we see that using heuristics to bias the viewpoint selection produces marginal differences. The space subdivision creates major impacts.

B. UAV Flight Test

Our UAV platform is shown in Fig. 8(a). This is a DJI Mavic Air UAV built with a gimbal camera that covers the front and downward areas. Two additional cameras are

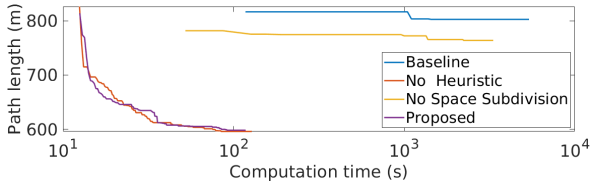


(a)



(b)

(c)



(d)

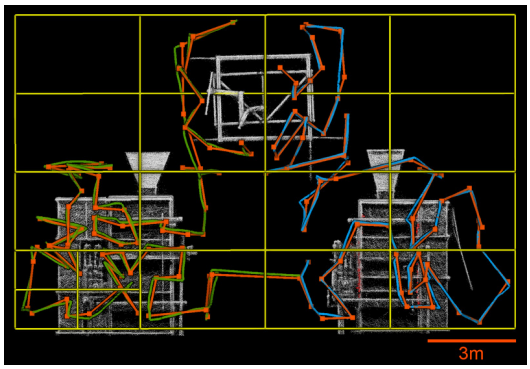
Fig. 7: Bridge test result. The figure shares the same layout as Fig. 5. 2681 out of 6695 viewpoints are selected.

mounted on the top to cover the upward areas. During flight tests, the positioning of the UAV is provided by a LiDAR-based tracking system (Fig. 8(b)) based on our previous work [22]. We first build a prior map of the environment before the flight. The tracking system then localizes on the prior map and subtracts the map from the scan data. The points left are from the UAV. We apply a simple filtering process to remove outlier points not from the UAV but other moving objects in the environment. To facilitate tracking, a foam pole is attached to the top of the UAV to increase the height of the tracking target. The control commands are sent to the

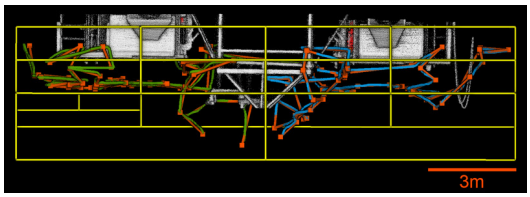
(a)

(b)

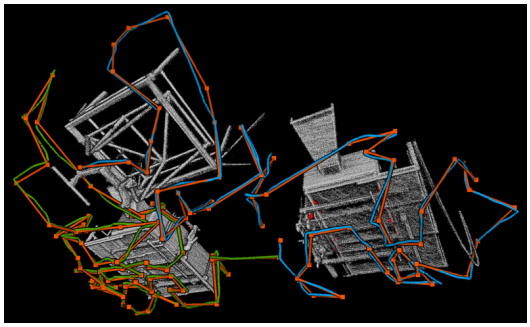
Fig. 8: (a) DJI Mavic Air UAV. The onboard gimbal camera covers the front and downward areas. Two additional cameras are mounted on the top to cover the upward area. (b) LiDAR-based tracking system to provide positioning of the UAV during flight tests. The system tracks the foam pole attached to the top of the UAV in the scan data.



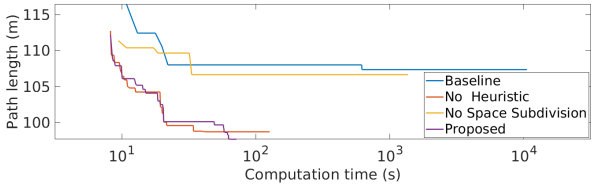
(a)



(b)



(c)



(d)

Fig. 9: UAV flight test result. The figure shares the same layout and convention as Fig. 5. 290 out of 1090 viewpoints are selected to form the coverage path. The green path and the blue path in (a)-(c) are the trajectories of the UAV. Due to battery limitation, the UAV flight test is conducted in two runs starting at the two ends of the coverage path (orange). One run covers the left side and the other covers the right side. Both runs last for about 10 minutes. The maximum speed of the UAV during the flight test is 0.4m/s.

UAV from a laptop computer over WiFi.

The result of the UAV flight test is shown in Fig. 9. The green path and the blue path in Fig. 9(a)-Fig. 9(c) are the trajectories of the UAV. Due to limitation of battery life, the flight test is separated into two runs starting at the two ends of the coverage path (orange). Each run lasts for about 10 minutes. The maximum speed of the UAV during the flight test is 0.4m/s. Fig. 9(d) gives the runtime comparison. Further, we show three representative images

(a)

(b)

(c)

Fig. 10: Representative images recorded in the UAV flight test. (a) is from the gimbal camera on the UAV. (b) and (c) are from the two upward cameras.

recorded by the onboard cameras during the flight test in Fig. 10. Specifically, Fig. 10(a) is from the gimbal camera and Fig. 10(b)-Fig. 10(c) are from the upward cameras.

Finally, let us examine the runtime breakdown for the four tests in Table I. The runtime on space subdivision is negligible in compassion to the other tasks. Viewpoint sampling is shared by both methods. The runtime on viewpoint selection and TSP solving is for one algorithm iteration. Thanks to the multi-resolution hierarchical framework, our method runs 20+x faster in both viewpoint selection and TSP solving.

V. Conclusion and Future Work

The paper proposes a method solving for coverage planning with a multi-resolution hierarchical framework. Paths are computed at two different levels – at a high level as the order of the subspaces for traversal through the subspaces and at a low level as detailed coverage paths within the subspaces. The resulting method runs significantly faster than the state-of-the-art. The current method uses geometry-based metrics to separate the environment into subspaces without tacking into account the semantic meanings of the objects. In the future, we plan to employ semantic segmentation in separating the environment. We expect the future method to cover the environment object by object. An advantage of such a method is that a human executive can determine the set of objects to cover.

Acknowledgment

Special thanks are given to M. Mousaei for facilitating UAV experiments.

TABLE I: Comparison of runtime breakdown. The viewpoint sampling is for both methods. The viewpoint selection and TSP solving are for one algorithm iteration.

Test	Viewpoint sampling	Baseline		Proposed	
		Viewpoint selection	TSP	Viewpoint selection	TSPs
Machine shop	147.8s	163.0s	64.8s	1.9s	0.20s
Aircraft	7.5s	6.3s	7.7s	0.15s	0.37s
Bridge	12.1s	16.2s	39.1s	0.10s	0.36s
UAV flight	8.1s	1.6s	0.67s	0.044s	0.017s

REFERENCES

[1] H. Choset, "Coverage for robotics–a survey of recent results," *Annals of mathematics and artificial intelligence*, vol. 31, no. 1-4, pp. 113–126, 2001.

[2] ——, "Coverage of known spaces: The boustrophedon cellular decomposition," *Autonomous Robots*, vol. 9, no. 3, pp. 247–253, 2000.

[3] M. K. Micali and D. Dornfeld, "Fully three-dimensional toolpath generation for point-based additive manufacturing systems," in *Solid Freeform Fabrication Symposium*, vol. 27, 2016.

[4] P. N. Atkar, A. Greenfield, D. C. Conner, H. Choset, and A. A. Rizzi, "Uniform coverage of automotive surface patches," *The International Journal of Robotics Research*, vol. 24, no. 11, pp. 883–898, 2005.

[5] G. A. Hollinger, B. Englot, F. S. Hover, U. Mitra, and G. S. Sukhatme, "Active planning for underwater inspection and the benefit of adaptivity," *The International Journal of Robotics Research*, vol. 32, no. 1, pp. 3–18, 2013.

[6] A. Bircher, M. Kamel, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel, and R. Siegwart, "Three-dimensional coverage path planning via viewpoint resampling and tour optimization for aerial robots," *Autonomous Robots*, vol. 40, no. 6, pp. 1059–1078, 2016.

[7] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the operations research society of America*, vol. 2, no. 4, pp. 393–410, 1954.

[8] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous systems*, vol. 61, no. 12, pp. 1258–1276, 2013.

[9] L. Paull, S. Saeedi, M. Seto, and H. Li, "Sensor-driven online coverage planning for autonomous underwater vehicles," *IEEE/ASME Transactions on Mechatronics*, vol. 18, no. 6, pp. 1827–1838, 2012.

[10] N. Gyagenda, A. K. Nasir, H. Roth, and V. Zhmud, "Coverage path planning for large-scale aerial mapping," in *Annual Conference Towards Autonomous Robotic Systems*. Springer, 2019, pp. 251–262.

[11] M. F. Jensen, D. Bochtis, and C. G. Sørensen, "Coverage planning for capacitated field operations, part ii: Optimisation," *Biosystems Engineering*, vol. 139, pp. 149–164, 2015.

[12] E. U. Acar, H. Choset, A. A. Rizzi, P. N. Atkar, and D. Hull, "Morse decompositions for coverage tasks," *The international journal of robotics research*, vol. 21, no. 4, pp. 331–344, 2002.

[13] I. Rekleitis, A. P. New, E. S. Rankin, and H. Choset, "Efficient boustrophedon multi-robot coverage: an algorithmic approach," *Annals of Mathematics and Artificial Intelligence*, vol. 52, no. 2-4, pp. 109–142, 2008.

[14] A. Gunawan, H. C. Lau, and P. Vansteenwegen, "Orienteering problem: A survey of recent variants, solution approaches and applications," *European Journal of Operational Research*, vol. 255, no. 2, pp. 315–332, 2016.

[15] M. Roberts, D. Dey, A. Truong, S. Sinha, S. Shah, A. Kapoor, P. Hanrahan, and N. Joshi, "Submodular trajectory optimization for aerial 3d scanning," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 5324–5333.

[16] S. Arora and S. Scherer, "Rapidly exploring random orienteering," 2016.

[17] C. Dornhege, A. Kleiner, A. Hertle, and A. Kolling, "Multirobot coverage search in three dimensions," *Journal of Field Robotics*, vol. 33, no. 4, pp. 537–558, 2015.

[18] Y. You, L. Fan, K. Roimela, and V. V. Mattila, "Simple octree solution for multi-resolution lidar processing and visualisation," in *2014 IEEE International Conference on Computer and Information Technology*. IEEE, 2014, pp. 220–225.

[19] L. Perron and V. Furnon, "Or-tools," Google. [Online]. Available: https://developers.google.com/optimization/

[20] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 2951–2957.

[21] C. H. Papadimitriou, "The complexity of the lin–kernighan heuristic for the traveling salesman problem," *SIAM Journal on Computing*, vol. 21, no. 3, pp. 450–465, 1992.

[22] J. Zhang and S. Singh, "Laser-visual-inertial odometry and mapping with high robustness and low drift," *Journal of Field Robotics*, vol. 35, no. 8, pp. 1242–1264, 2018.