

Efficient Visual Exploration and Coverage with a Micro Aerial Vehicle in Unknown Environments

Lionel Heng, Alkis Gotovos, Andreas Krause, and Marc Pollefeys

Abstract—In this paper, we propose a novel and computationally efficient algorithm for simultaneous exploration and coverage with a vision-guided micro aerial vehicle (MAV) in unknown environments. This algorithm continually plans a path that allows the MAV to fulfil two objectives at the same time while avoiding obstacles: observe as much unexplored space as possible, and observe as much of the surface of the environment as possible given viewing angle and distance constraints. The former and latter objectives are known as the exploration and coverage problems respectively. Our algorithm is particularly useful for automated 3D reconstruction at the street level and in indoor environments where obstacles are omnipresent. By solving the exploration problem, we maximize the size of the reconstructed model. By solving the coverage problem, we maximize the completeness of the model. Our algorithm leverages the state lattice concept such that the planned path adheres to specified motion constraints. Furthermore, our algorithm is computationally efficient and able to run on-board the MAV in real-time. We assume that the MAV is equipped with a forward-looking depth-sensing camera in the form of either a stereo camera or RGB-D camera. We use simulation experiments to validate our algorithm. In addition, we show that our algorithm achieves a significantly higher level of coverage as compared to an exploration-only approach while still allowing the MAV to fully explore the environment.

I. INTRODUCTION

Micro aerial vehicles are soaring in popularity due to their compact size and affordability. They are used in an increasing number of applications such as aerial photography, aerial surveillance, environmental monitoring, and search and rescue. An up-and-coming application is automated 3D reconstruction of large environments. The generated 3D models are used for many purposes such as visualization, geological assessments, and model-based localization with a camera. In outdoor scenarios, automated 3D reconstruction is achieved by commanding a MAV equipped with a downward-looking camera to follow a preset flight path at a high altitude [14]. The MAV records GPS and image data during the flight, and afterwards, a 3D reconstruction pipeline converts the recorded data to a 3D model. However, the assumption of an obstacle-free environment makes this technique ill-suited for use at the street level and in indoor environments where obstacles are omnipresent. In these settings, the MAV has to detect obstacles, usually with a forward-looking camera, and factor them in its planning.

We attempt to simultaneously solve the exploration and

L. Heng {hjianyon@dso.org.sg} is with the Information Division, DSO National Laboratories, Singapore. A. Gotovos, A. Krause, and M. Pollefeys {alkisg@inf.ethz.ch, krausea@ethz.ch, marc.pollefeys@inf.ethz.ch} are with the Department of Computer Science, ETH Zürich, Switzerland. Most of the work was done when the first author was at ETH Zürich.

coverage problems for unknown 3D environments using a vision-guided MAV. In the *exploration* problem, we want to explore as much of the environment as possible, so that the reconstructed 3D model is as large as possible. In the *coverage* problem, we want to observe the entire surface of the environment given viewing angle and distance constraints so that the reconstructed 3D model is complete and distortion-free.

In this paper, we describe our algorithm for efficient visual exploration and coverage with a MAV in unknown environments. Our algorithm is designed to run on-board a MAV with limited computational resources. We assume that the MAV’s pose is already known and that the MAV is equipped with a forward-looking depth-sensing camera in the form of either a stereo camera or a RGB-D camera. We do not see these assumptions as restrictive, as the pose can be provided by a SLAM system, and a depth-sensing camera is low-cost and easy to mount on a MAV.

For efficient visual exploration and coverage, we use the state lattice [17] as a discrete graphical representation of the state space. This state lattice representation has two important advantages with respect to computational efficiency:

- 1) the problem of path planning with motion constraints reduces to an unconstrained graph search problem, and
- 2) translational invariance: any motion or edge that connects two states also connects all other pairs of identically arranged states.

We exploit this translational invariance property by pre-computing several aspects of the exploration and coverage algorithm such that this algorithm is able to run in real-time and on a computationally-constrained MAV.

As the quadrotor dynamics are differentially flat [15], the control inputs can be expressed as a function of four flat outputs $[x, y, z, \psi]$ and their derivatives, where $[x \ y \ z]^T$ are the coordinates of the center of mass of the quadrotor and ψ is the yaw angle. Hence, we define the MAV’s state to be $[x \ y \ z \ \psi]^T$ with zero roll and pitch, which results in a 4D state space.

Our proposed algorithm alternately solves the exploration and coverage problems. At each step of our proposed algorithm, we first solve the exploration problem by looking for goals located on the edges of currently known free space and choosing the one with the highest information gain weighed exponentially by its cost to reach. Subsequently, we solve the coverage problem by planning a path to the selected goal such that coverage is maximized subject to budgets on the total path cost and planning time. We repeat this

procedure each time the path is blocked or the information gain associated with the goal drops significantly.

Our exploration and coverage algorithm is novel in the sense that, to the best of our knowledge, there is no other existing work that solves both the exploration and coverage problems in real-time. Exploration algorithms [18, 19] attempt to fully explore an environment and run in real-time but do not optimize coverage. Coverage planning algorithms [3, 5, 6] attempt to fully cover an environment but require a known map that accurately represents the environment of interest. Next-best-view planning algorithms [4, 13] are most similar to our work but assume a single-object scene. Thus, they sample candidate views within a sphere around the object before selecting the view with the highest information gain. These algorithms do not scale well to multi-object scenes in which more objects correlate to more sampling, and in turn, a higher computational cost. Our algorithm is able to run in real time on a computationally-constrained MAV equipped with a forward-looking depth-sensing camera. Furthermore, path planning with motion constraints is integrated into our algorithm, which allows us to keep CPU and memory usage to a minimum by removing the need for a separate path planner.

A. Related Work

There has been work on exploration in 3D space [18, 19] with a depth-sensing camera. Shade and Newman [18] formulated the 3D exploration problem as a partial differential equation (PDE). The solution is in the form of a vector field, and the path of steepest descent through the vector field is chosen. However, the chosen path may not respect the motion constraints of the MAV, and furthermore, solving the PDE in real time requires a GPU, which is not available on a MAV. Shen et al. [19] demonstrated 3D exploration running on-board a MAV. They use particles both as a memory-efficient sparse representation of free space and for identifying frontiers. The latter is achieved by simulating the expansion of a system of particles with Newtonian dynamics, and detecting regions of greatest expansion. These exploration algorithms do not take coverage into account. By not doing so, the reconstructed model will not be complete, and distortion and holes will be prevalent.

On the other hand, existing approaches to the coverage planning problem typically require a prior 3D model of the environment and plan a path offline. Cheng et al. [3] simplifies a 2.5D model of the environment as a set of hemispheres and cylinders, and plans a path that completely covers the surfaces. However, this approach does not generalize well to 3D models and to cluttered environments in which objects not modelled well by hemispheres and cylinders may not be fully covered. Englot and Hover [5] generate a set of redundant view configurations that completely cover the model, and find a feasible path that connects a subset of the view configurations. To deal with imperfections in the a priori map, Galceran et al. [6] iteratively optimizes the initial planned coverage path using current sensor measurements. In contrast, we do not require a prior model, and at the same

time, we plan a path in an online fashion and that explicitly considers the motion constraints of the robot.

Next-best-view planning algorithms [4, 13] iteratively determine in real-time the best viewing configuration for a camera to go to such that the reconstructed model is as complete as possible. However, to maintain computational tractability, they assume a single-object environment that is free of obstacles, and that the object location is known. They exploit these assumptions by orienting all viewing configurations towards the center of the object. These assumptions are not applicable to our case of modelling an entire unknown environment. Furthermore, these algorithms are myopic, and as a result, there is no upper bound on the overall path length.

In the context of informative path planning, the problem of maximizing the information gathered subject to a cost budget constraint has been previously formulated as a submodular orienteering problem [20, 1]. The algorithms used in these cases are based on the recursive-greedy algorithm for submodular orienteering proposed by Chekuri and Pal [2], which has provable approximation guarantees, but is prohibitively slow for real-time use on graphs with hundreds or thousands of nodes like ours. We propose a linearized approximation, based on the general framework proposed by Iyer et al. [12] for submodular function optimization, which drops theoretical guarantees, but performs fast enough to be used in our setting.

II. EXPLORATION AND COVERAGE

We use a state lattice \mathcal{L} to discretize the 4D state space; the states are arranged in a regular 3D grid pattern while the yaw angle is discretized non-uniformly. The presence of an edge between two states indicates that it is feasible for the robot to move between these two states given its motion constraints. Each edge is assigned a weight that indicates the cost of moving along the edge and the cost of a path between two states is simply the sum of edge weights along the path. We construct a set of edges in the form of primitive motions that allows the MAV to turn on the spot, and move within the camera's field of view. In this way, the MAV will not collide with an unseen object.

Furthermore, the state lattice allows us to precompute several aspects of the exploration and coverage algorithm such that the algorithm can efficiently run in real-time and on-board the MAV. Ray-casting has been used to compute the information gain [21] in 2D exploration with a laser-guided robot. However, ray-casting is not computationally feasible for stereo and RGB-D sensors which generate more depth measurements by 2 orders of magnitude. In contrast, by making use of precomputed data based on the state lattice, we are able to efficiently compute the information gain without the need for ray-casting. In addition, computing the coverage typically requires computationally-expensive z -mapping, however the use of precomputed data circumvents the need for that. We only use a combination of memory lookup operations and integer arithmetic operations to compute information gain and coverage.

To choose a goal state, we compute a list of candidate states located along frontiers and select the one that maximizes information gain and allows for rapid map expansion. Once a goal is chosen, we aim to compute a path to it that maximizes the total coverage provided by states in the path, while constraining the maximum path length, as well as the computation time. If the MAV reaches the goal, the path is blocked, or the information gain associated with the goal significantly decreases, we repeat both goal selection and path planning.

A. Precomputation

1) *Swath*: Each edge in the state lattice encodes an element in a predetermined set of primitive motions \mathbf{M} . The swath of a primitive motion is a set of voxels occupied by the MAV’s footprint during the execution of the motion. We precompute the swath for each primitive motion in \mathbf{M} . By precomputing these swaths, we avoid the use of computationally-expensive simulation to check if the MAV collides with an obstacle. Instead, we efficiently detect a collision by checking if any voxel in the swath is occupied.

2) *View Frustum*: For each discrete yaw angle $\psi \in \{\psi_1, \dots, \psi_n\}$, we construct a view frustum which is parameterized by:

- 1) the view frustum origin which is equivalent to the camera pose derived from the camera-MAV transform and the MAV’s state $[0 \ 0 \ 0 \ \psi]^T$, and
- 2) six planes whose normals point towards the center of the view frustum.

We denote the set of voxels falling within the view frustum as \mathcal{V}_ψ . To compute \mathcal{V}_ψ , we first find an axis-aligned minimum bounding box that contains the six corners of the view frustum. Subsequently, for each voxel v in the bounding box, we check whether v falls within the view frustum by computing the signed distance between v ’s center and each plane and checking whether that signed distance is positive. If so, we add v to \mathcal{V}_ψ . All voxels in \mathcal{V}_ψ are sorted in order of increasing distance from the camera center. For each voxel $v \in \mathcal{V}_\psi$, we store the following information in a file:

- 1) \mathbf{p}_v , the local coordinates of v ’s center with respect to the view frustum origin,
- 2) the indices of all voxels that would be occluded by v if it were occupied, and
- 3) the image coordinates of the projection of each face on the image plane only if the projection meets certain conditions discussed below.

For any voxel face, we store the image coordinates of its projection only if two conditions are met: the ray from the camera center through v ’s center is incident on the face with a viewing angle less than θ_{max} , and the image area of the projected face is at least p pixels. These two conditions ensure distortion-free and highly-detailed texture mapping, and in turn, a high-quality 3D textured model of the environment.

Each instance of the exploration and coverage algorithm loads the contents of this file into a lookup table. This view

Algorithm 1: Algorithm for computing $I(s_g)$.

input : environment map, lookup table, $\mathbf{p}_g, \mathcal{V}_{\psi_g}$

output: $I(s_g)$

$I(s_g) = 0$

for $v \in \mathcal{V}_{\psi_g}$ **do**

$\mathbf{p} = \mathbf{p}_v + \mathbf{p}_g$

 set the label of v equal to that of the voxel located at \mathbf{p} in the environment map

if v is labeled as occupied **then**

 query the lookup table to find $\mathcal{O} \subset \mathcal{V}_{\psi_g}$ where

\mathcal{O} is the set of voxels occluded by v

 mark all voxels in \mathcal{O} as occluded

for $v \in \mathcal{V}_{\psi_g}$ **do**

if v is labeled as unknown and not marked as occluded **then**

$I(s_g) = I(s_g) + 1$

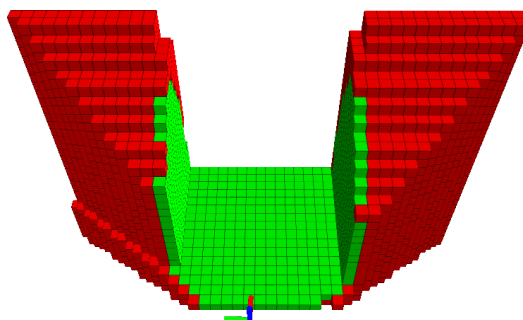


Fig. 1: The camera pose is shown as a set of 3 perpendicular axes. Occupied voxels are represented by green cubes. Voxels corresponding to unknown space and occluded by occupied voxels are represented by red cubes. These voxels are not considered in the computation of the information gain.

frustum precomputation allows us to avoid computationally-expensive floating-point arithmetic operations when evaluating utility functions for both exploration and coverage.

B. Choosing a Goal to Go To

When choosing a goal to go to, the MAV chooses a goal at which it maximizes a utility function that rewards exploration. Such goals are located on frontiers in the environment map.

First, we construct a set of candidate goals \mathbf{G} by including all states located on frontiers. For any candidate goal $s_g \in \mathbf{G}$, we define the information gain $I(s_g)$ to be the number of unexplored voxels that are enclosed in the corresponding view frustum and are not occluded by occupied voxels. Fig. 1 illustrates an example of unexplored voxels that are occluded by occupied voxels. Assume without loss of generality that s_g has coordinates \mathbf{p}_g and a discrete yaw angle ψ_g . Then \mathcal{V}_{ψ_g} contains the voxels encapsulated in the view frustum that corresponds to s_g . We efficiently compute $I(s_g)$ using Alg. 1; this computation typically takes a few milliseconds.

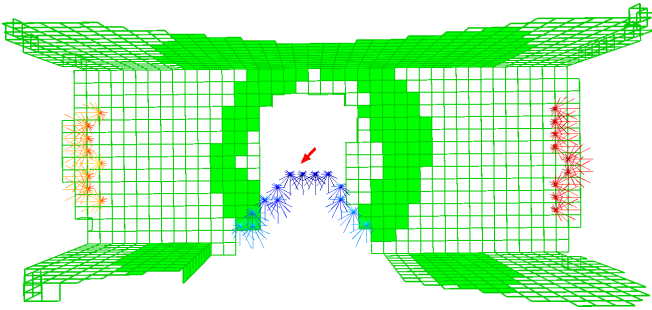


Fig. 2: Candidate goals are shown as lines. These goals are located on frontiers. The longer the line, the higher the associated information gain. The more red the line, the higher the path cost to the corresponding goal. A red arrow marks the current pose of the MAV.

Given the current state of the MAV $a \in \mathcal{L}$, we choose the candidate goal that maximizes the utility function [7]: $U_1(s_g) = I(s_g)e^{-\lambda \ell_{min}(a, s_g)}$, where λ is a parameter that determines the trade-off between rapid exploration and filling in details, and $\ell_{min}(a, s_g)$ is the cost of the shortest path from a to s_g . Fig. 2 shows an example of candidate goals.

For purposes of efficiency, we use lazy evaluation such that we do not have to compute $I(s_g)$ and $\ell_{min}(a, s_g)$ for all s_g . We know that $\max(I) = \max\{|\mathcal{V}_{\psi_1}|, \dots, |\mathcal{V}_{\psi_n}|\}$. Using Dijkstra’s algorithm, we evaluate each state s in order of increasing $\ell_{min}(a, s)$ from the start state. Given the currently evaluated state s , we compute the maximum attainable score $\max(I)e^{-\lambda \ell_{min}(a, s)}$. If this maximum attainable score is less than the maximum score $m = \max(\{I(s_{g_1}), \dots, I(s_{g_n})\})$ over all previously evaluated candidate goals s_{g_1}, \dots, s_{g_n} , we know that we cannot get a score higher than m for a not-yet-evaluated candidate goal which is guaranteed to have a higher path cost than s , and thus, stop the evaluation. We then plan a path to the candidate goal which corresponds to the maximum score of m .

C. Planning a Path to the Goal

Given the current state of the MAV $a \in \mathcal{L}$ and the goal state $b = \arg \max_{s_g \in \mathbf{G}} U_1(s_g)$ obtained in the previous section, we would like to plan a path from a to b , such that a utility function U_2 that expresses coverage is maximized. More formally, if \mathcal{P}_{ab} is the set of all paths from a to b in \mathcal{L} , then we would like to solve

$$\begin{aligned} \max. \quad & U_2(P) \\ \text{s.t.} \quad & P \in \mathcal{P}_{ab} \\ & \ell(P) \leq B, \end{aligned} \quad (1)$$

where $\ell(P)$ denotes the cost of the path as a sum of edge weights and B is our path cost budget.

Given a set \mathcal{S} of previously traversed states and a state s , we define the marginal gain of the coverage function $C(s|\mathcal{S}) := U_2(\mathcal{S} \cup \{s\}) - U_2(\mathcal{S})$ as the number of voxel faces that:

- 1) are associated with occupied voxels,

Algorithm 2: Algorithm for computing $C(s|\mathcal{S})$.

input : environment map, lookup table, \mathbf{p} , \mathcal{V}_ψ

output: $C(s|\mathcal{S})$

initialize a zero 2D buffer with the same dimensions as the camera image

$C(s|\mathcal{S}) = 0$

for $v \in \mathcal{V}_\psi$ **do**

$\mathbf{p}' = \mathbf{p}_v + \mathbf{p}$

 set the label of v equal to that of the voxel located at \mathbf{p}' in the environment map

if v is labeled as occupied **then**

for each face f of v **do**

if f is marked as previously observed **then**
 | continue

 query the lookup table to find \mathcal{J} , the set of image coordinates of the projection of f on the image plane

for each $\mathbf{j} \in \mathcal{J}$ **do**

if the pixel with coordinates \mathbf{j} in the buffer has a zero value, set the pixel’s value equal to v ’s index

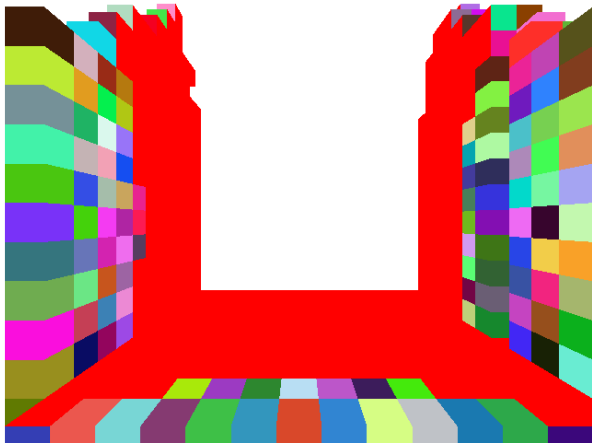
for $v \in \mathcal{V}_\psi$ **do**

if any pixel in the buffer has a value equal to v ’s index **then**

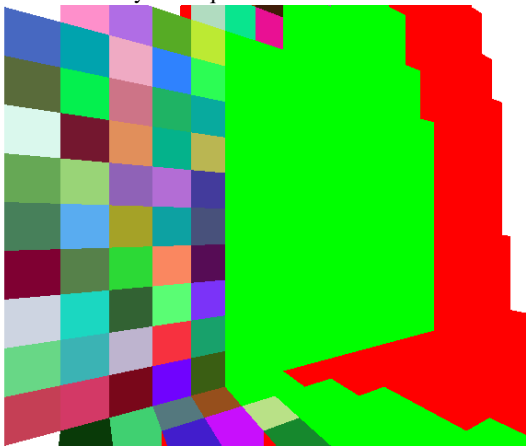
 | $C(s|\mathcal{S}) = C(s|\mathcal{S}) + 1$

- 2) have not been observed by the MAV at any state $t \in \mathcal{S}$ given a maximum viewing angle of θ_{max} and a minimum projection area of p pixels, and
- 3) are observed by the MAV at state s given a maximum viewing angle of θ_{max} and a minimum projection area of p pixels.

Assume without loss of generality that s has coordinates \mathbf{p} and a discrete yaw angle ψ . Then \mathcal{V}_ψ contains the voxels encapsulated in the view frustum that corresponds to s . We efficiently compute $C(s|\mathcal{S})$ using Alg. 2 that is similar to the z -buffering technique commonly used in computer graphics. The 2D buffer computed in Alg. 2 is identical to a z -buffer in the sense that each pixel in the buffer corresponds to the closest voxel with respect to the camera. However, for each pixel, we store the index of the closest voxel instead of storing the distance to the closest voxel. We note that the lookup table only contains image coordinates for voxel face projections that meet the two conditions of a maximum viewing angle and minimum projection area. As a result, when computing the 2D buffer, we do not have to explicitly consider these two conditions which are computationally expensive. Although we do not compare depth values, it is guaranteed that Alg. 2 records the index of the closest voxel for each pixel in the 2D buffer as the voxels are stored in order of increasing distance from the camera in the lookup table. We can see that the precomputation step allows for



(a) A visualization of the computed 2D buffer at the beginning of the flight. Red pixels correspond to faces of occupied voxels whose incidence angle exceeds θ_{max} or whose projection area is below p pixels. Other non-white pixels correspond to voxel faces whose incidence angle is at most θ_{max} , and are considered as new observations. Each voxel is identified by a unique non-red color.



(b) A visualization of the computed 2D buffer several seconds later as the MAV turns on the spot towards the left. Green pixels correspond to voxel faces that are marked as having been observed previously given the viewing angle and projection area constraints. The computed marginal coverage only includes new observations (colors other than red and green) and not previous observations (green).

Fig. 3: Visualization of the 2D buffers computed in Alg. 2.

several optimizations that allow us to efficiently compute $C(s|\mathcal{S})$ within a few milliseconds. Note that the dependence on previous observations \mathcal{S} is implicit in the environment map that is given as input to the algorithm. Fig. 3 illustrates two examples of our 2D buffer.

If the coverage function is modular, that is, if it can be written as a sum of weights over the states in the path $U_2(P) = \sum_{s \in P} w_s$, then Eq. (1) is an instance of the *orienting problem* [22]. Unfortunately, the coverage function we are using does not satisfy modularity; intuitively, while a state by itself may provide significant coverage, its marginal coverage benefit is heavily reduced when we have already visited “nearby” states in the lattice. However, the coverage

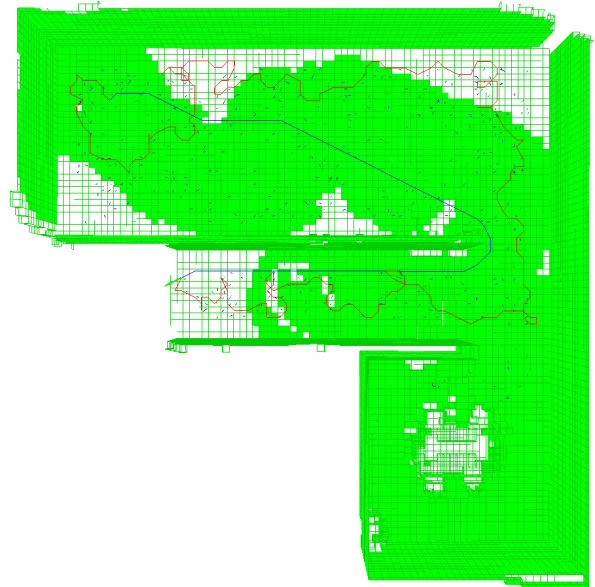


Fig. 4: A blue line represents the shortest path to the goal state while a red line represents the path to the goal state that maximizes coverage subject to path cost and planning time budgets. Note that the planned path attempts to cover areas with a high number of unobserved voxel faces. Cube face outlines represent faces of unoccluded occupied voxels in the environment map while solid cube faces represent voxel faces that have been previously observed subject to the viewing angle and minimum projection area constraints. Thin small blue lines represent sampled states.

function satisfies a natural “diminishing returns” property known as *submodularity* [16], which can be formally stated as follows: for any two sets of already traversed states $\mathcal{S} \subseteq \mathcal{T}$ and any state $s \notin \mathcal{T}$, it holds that $C(s|\mathcal{T}) \leq C(s|\mathcal{S})$.

This makes Eq. (1) an instance of the *submodular orienting problem* which has been proven to be hard to approximate up to a logarithmic factor of the optimum [2]. To efficiently obtain an approximate solution, we use a two-step approach, based on first obtaining an orienting problem by approximating the submodular function by a modular one, and then computing an approximate solution for that problem. More precisely, we first linearize the coverage function by randomly subsampling N states from the lattice and assigning to each sampled state, s_i , $i = 1, \dots, N$, a weight equal to its marginal coverage given all previously sampled states, that is, $w_{s_i} := C(s_i|\{s_1, \dots, s_{i-1}\})$. We use the modular surrogate function $\hat{U}_2(P) := \sum_{s \in P} w_s$ to formulate the resulting orienting problem instance as a mixed integer program [22], which we solve using the Gurobi Optimizer [8]. Since the orienting problem is itself NP-hard, Gurobi might fail to find the optimal path in reasonable time; however, it allows us to specify a path cost limit and time limit, and returns an approximate solution in the form of the best feasible path computed up to that point.

Fig. 4 shows an example of such a path, and in addition, the shortest path to the goal for purposes of comparison. We

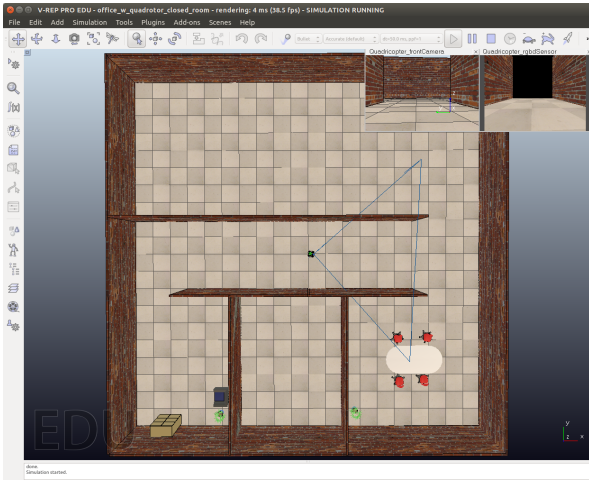


Fig. 5: A simulated office-like environment in the v-rep simulator.

observe that the planned path incorporates a wide variety of yaw angles in order to maximize coverage, and visits areas with a high number of unobserved voxel faces. As the MAV visits each state along the path until either the goal is reached or replanning is requested, the environment map is updated with the new voxel face observations at that state.

III. EXPERIMENTS AND RESULTS

We use the v-rep simulator from Coppelia Robotics for our simulation experiments. We simulate a quadrotor with a RGB-D camera in an office-like environment shown in Fig. 5. This RGB-D camera outputs 640×480 color and depth images.

In each experiment, we incrementally map the environment in 3D using the tiled octree-based occupancy map implementation [9] with dynamic tile caching. This implementation uses constant space regardless of the volume of the environment to be mapped. As we use a RGB-D camera, we utilize a beam-based inverse sensor measurement model to update the occupancy map. In this model, cells within σ m of the measurement are assigned a constant probability $p_{occupied}$ corresponding to occupied space, cells at least σ m in front of a measurement are assigned a constant probability p_{free} corresponding to free space, and cells at least σ behind the measurement are assigned a constant probability $p_{unknown}$ corresponding to unknown space. Using the full-resolution depth images as input, we update the occupancy map at 5 Hz using a single Intel 2.1 GHz core.

We parameterize the edge budget B as a multiple of the shortest path cost from the current state a to the goal state b , that is, $B = \kappa \ell_{min}(a, b)$, and set $\kappa = 1.5$. Furthermore, we use a planning time budget of 3 seconds. We constrain the MAV to fly at a specific height which is 1 m.

We compare our algorithm and a variant thereof with the frontier-based exploration algorithm [11], which involves going to the nearest frontier cell. Our algorithm involves finding the goal state that maximizes U_1 , and computing a path to that state by approximately solving (1). The variant of

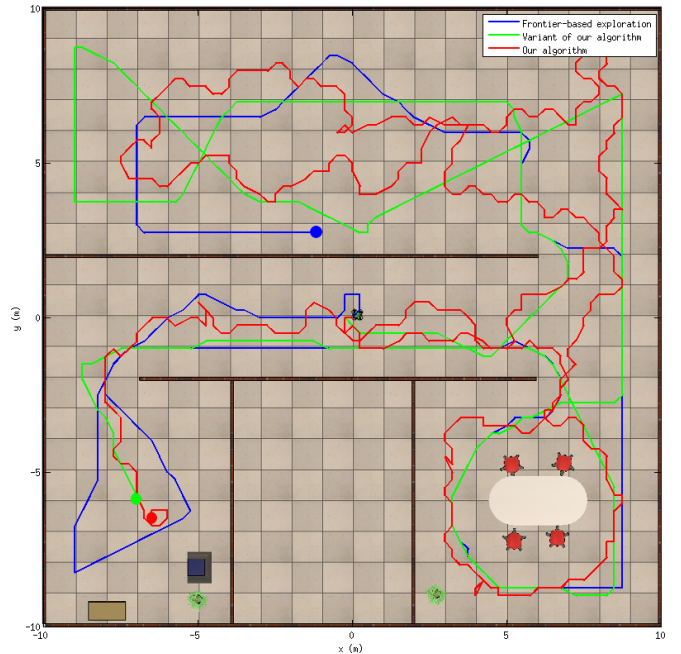


Fig. 6: The paths taken by the MAV with the frontier-based exploration algorithm, the variant of our algorithm, and our algorithm are colored blue, green, and red respectively. Each path starts at the center of the map. A circle marks the end of the path.

TABLE I: For each algorithm, we compute the average over 10 runs of the path length and percentage of voxel faces that are observed.

Algorithm	Path Length (m)	% voxel faces that are observed
Frontier-based exploration [11]	98.4	74.1%
Variant of our algorithm	112.0	79.0%
Our algorithm	136.1	89.9%

our algorithm finds the goal state in an identical manner, but just uses the shortest path to that state. Fig. 6 shows the path taken by the MAV when we run each of the three algorithms: the frontier-based exploration algorithm, the variant of our algorithm, and our algorithm. We observe from this figure that the path computed by our algorithm enforces a constant change in yaw angle as the MAV moves throughout the environment. In this way, we maximize coverage at the expense of a longer path length.

We execute 10 runs of each algorithm. All algorithms assume an initially unknown environment, and terminate when there are no more frontiers left. For each algorithm, we compute the average path length and percentage of voxel faces that are observed over all runs; the results are shown in Table I.

The results show that although our exploration and coverage algorithm incurs the longest path length, it achieves the highest coverage by a significant margin. Fig. 7 shows the resulting coverage maps after the execution of both the frontier-based exploration and our algorithm. We observe that

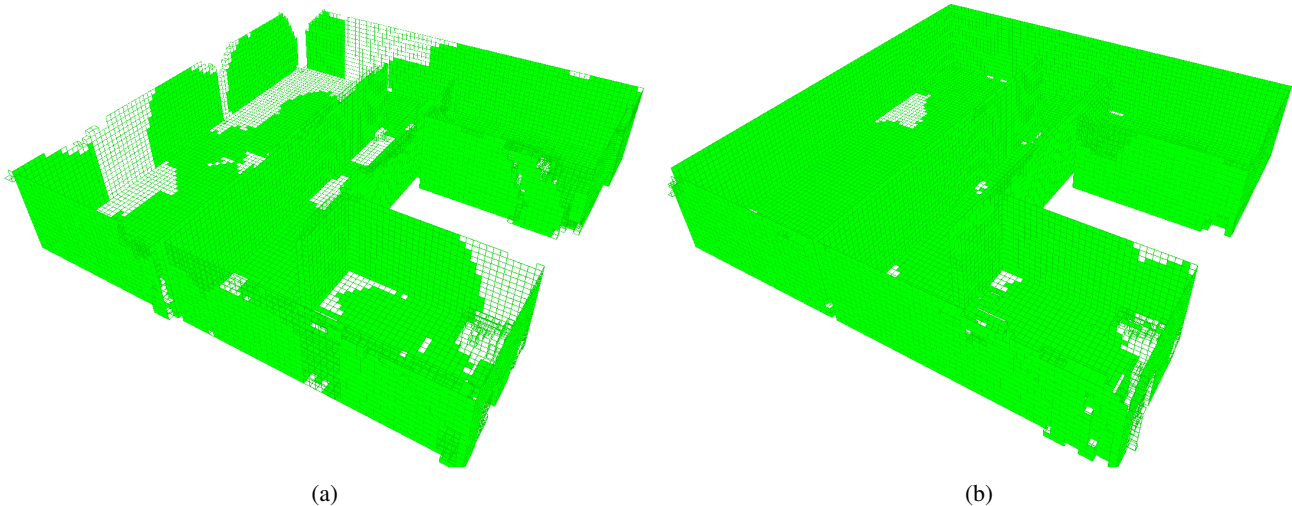


Fig. 7: The resulting coverage maps after the completion of the frontier-based exploration (a) and our algorithm (b). From visual inspection, we can see that our algorithm produces a significantly more complete map.

our algorithm generates a significantly more complete map. For 3D reconstruction, coverage is of high importance, as we seek a 3D model that is as complete as possible, and thus, our exploration and coverage algorithm is suitable for automated 3D reconstruction.

IV. CONCLUSIONS

We have proposed an algorithm, which is the first to simultaneously solve the exploration and coverage problems in real-time. Our algorithm follows a two-step approach: (1) choose the goal state that maximizes information gain weighed by the cost to get there, and (2) plan a path to that goal state that maximizes coverage given path cost and planning time budgets. By combining efficient exploration and coverage, we facilitate automated 3D reconstruction in cluttered environments with a MAV. Simulation experiments show our algorithm to work well and in real-time on a single CPU core that is similar to that found on MAVs equipped with Intel Core i7 single-computer boards and sold by Ascending Technologies.

In the near future, we plan to conduct real-world experiments with our algorithm running on-board a MAV. However, there is one obstacle we have to overcome before real-world demonstrations can occur. Although we have developed a real-time on-board implementation for SLAM [10] that is able to close the loop, our mapping implementation [9] is not able to close the loop. Currently, no CPU-based technique exists for dealing with loop closures for 3D occupancy maps in real-time.

V. ACKNOWLEDGEMENTS

The first author is funded by the DSO National Laboratories Postgraduate Scholarship. This work is partially supported by the SNSF V-MAV grant (DACH framework), and an ERC Starting Grant.

REFERENCES

- [1] J. Binney, A. Krause, and G. S. Sukhatme. Informative path planning for an autonomous underwater vehicle. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4791 – 4796, 2010.
- [2] C. Chekuri and M. Pal. A recursive greedy algorithm for walks in directed graphs. In *Foundations of Computer Science (FOCS)*, pages 245–253, 2005.
- [3] P. Cheng, J. Keller, and V. Kumar. Time-optimal uav trajectory planning for 3d urban structure coverage. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2750–2757, 2008.
- [4] E. Dunn, J. van den Berg, and J. Frahm. Developing visual sensing strategies through next best view planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4001–4008, 2009.
- [5] B. Englot and F. S. Hover. Three-dimensional coverage planning for an underwater inspection robot. *International Journal of Robotics Research (IJRR)*, 32(9-10): 1048–1073, 2013.
- [6] E. Galceran, R. Campos, N. Palomeras, M. Carreras, and P. Ridao. Coverage path planning with realtime replanning for inspection of 3d underwater structures. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6586–6591, 2014.
- [7] H. Gonzalez-Banos and J.-C. Latombe. Navigation strategies for exploring indoor environments. *International Journal of Robotics Research (IJRR)*, 21(10-11): 829–848, 2002.
- [8] I. Gurobi Optimization. Gurobi optimizer reference manual, 2014. URL <http://www.gurobi.com>.
- [9] L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Autonomous visual mapping and exploration with a micro aerial vehicle. *Journal of Field Robotics (JFR)*, 31(4):654–675, 2014.

- [10] L. Heng, G. H. Lee, and M. Pollefeys. Self-calibration and visual slam with a multi-camera system on a micro aerial vehicle. In *Proceedings of Robotics: Science and Systems (RSS)*, 2014.
- [11] D. Holz, N. Basilico, F. Amigoni, and S. Behnke. Evaluating the efficiency of frontier-based exploration strategies. In *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, pages 1–8, 2010.
- [12] R. Iyer, S. Jegelka, and J. Bilmes. Fast semidifferential-based submodular function optimization. In *International Conference on Machine Learning (ICML)*, pages 855–863, 2013.
- [13] M. Krainin, B. Curless, and D. Fox. Autonomous generation of complete 3d object models using next best view manipulation planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5031–5037, 2011.
- [14] O. Küng, C. Strecha, P. Fua, D. Gurdan, M. Achtelik, K.-M. Doth, and J. Stumpf. Simplified building models extraction from ultra-light uav imagery. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXVIII-1/C22:217–222, 2011.
- [15] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2520–2525, 2011.
- [16] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1):265–294, 1978.
- [17] M. Pivtoraiko, R. A. Knepper, and A. Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics (JFR)*, 26(3):308–333, 2009.
- [18] R. Shade and P. Newman. Choosing where to go: Complete 3d exploration with stereo. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2806–2811, 2011.
- [19] S. Shen, N. Michael, and V. Kumar. Stochastic differential equation-based exploration algorithm for autonomous indoor 3d exploration with a micro-aerial vehicle. *International Journal of Robotics Research (IJRR)*, 31(12):1431–1444, 2012.
- [20] A. Singh, A. Krause, C. Guestrin, and W. Kaiser. Efficient informative sensing using multiple robots. *Journal of Artificial Intelligence Research (JAIR)*, 34(1):707–755, 2009.
- [21] C. Stachniss, G. Grisetti, and W. Burgard. Information gain-based exploration using Rao-Blackwellized particle filters. In *Proceedings of Robotics: Science and Systems (RSS)*, Cambridge, USA, 2005.
- [22] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research (EJOR)*, 209(1):1–10, 2011.