9-15-2022

Active Simultaneous Localization and Mapping in Unstructured Environment with a Quadrotor

Siming He
University of Pennsylvania

Active Simultaneous Localization and Mapping in Unstructured Environment with a Quadrotor

Abstract

An agent performing Simultaneous Localization and Mapping (SLAM) constructs a map of the environment while estimating its location at the same time. SLAM algorithms primarily focus on finding the best estimate of the location of the agent, and by extension, that of the landmarks in the environment, given observations from sensors. These algorithms do not typically address how an agent should explore an unknown environment to build a map efficiently. This ability for active exploration is important for autonomous robots to work in unknown, unstructured environments such as forests or caves.

This paper proposes an active SLAM system that allows an agent to explore its surroundings, using visual-inertial data from an RGBD camera. We formalize this problem as taking actions that maximize the amount of information obtained from the scene. At each time step, a utility function that computes the incremental information gain is used to take actions. We conduct experiments using an Intel RealSense camera mounted on a custom-built quadrotor and show that we can explore indoor environments (while restricting the actions to choosing new viewpoints in SO(3) for practical reasons).

Keywords

Active Exploration, Simultaneous Localization and Mapping, Autonomous Robot, Quadrotor

Disciplines

Electrical and Computer Engineering

Active Simultaneous Localization and Mapping in Unstructured Environment with a Quadrotor

Siming He[1]

Abstract— **An agent performing Simultaneous Localization and Mapping (SLAM) constructs a map of the environment while estimating its location at the same time. SLAM algorithms primarily focus on finding the best estimate of the location of the agent, and by extension, that of the landmarks in the environment, given observations from sensors. These algorithms do not typically address how an agent should explore an unknown environment to build a map efficiently. This ability for active exploration is important for autonomous robots to work in unknown, unstructured environments such as forests or caves. This paper proposes an active SLAM system that allows an agent to explore its surroundings, using visual-inertial data from an RGBD camera. We formalize this problem as taking actions that maximize the amount of information obtained from the scene. At each time step, a utility function that computes the incremental information gain is used to take actions. We conduct experiments using an Intel RealSense camera mounted on a custom-built quadrotor and show that we can explore indoor environments (while restricting the actions to choosing new viewpoints in $\mathbf{SO}(3)$ for practical reasons).**

Index Terms— **Active Exploration, Simultaneous Localization and Mapping, Autonomous Robot, Quadrotor**

I. INTRODUCTION

The problem of SLAM mainly considers constructing a map of the environment while localizing the agent based on sensor data. However, SLAM algorithms do not address how an agent can actively gather sensor data to construct a detailed map. The capability of active exploration is important in many applications. For example, home robots need to have spatial awareness by exploring the environment before performing any tasks. Similarly, to understand forest ecology and assess carbon sequestration by the forests, we need robots to actively explore forests and estimate the number of trees, volume and species of each tree in the forests.

Researchers have investigated this active SLAM problem using various methods. The classical method is Next-Best-View planning: an agent samples potential actions, calculates the expected gain of each action and finally select the best action to execute [1]. Recently, belief-space planning and deep reinforcement learning are also used for active SLAM [1]. In this paper, we design a Next-Best-View planning system that samples and identifies the best action from $\mathbf{SO}(3)$. The system uses Kimera [2] and Voxblox [3] for SLAM and signed distance field (SDF) construction. The output of Kimera and Voxblox is then used in a planner to get the next best action. Then, we conduct experiments on a quadrotor and reflect on the challenges appeared during experiments. The quadrotor percepts the environment through a RealSense D435i camera and is controlled through a PID controller. The experiment result shows that the system is able to explore the environment. It also shows that future improvement is needed to incorporate localization and semantic uncertainty into utility functions, generalize the sample space from $\mathbf{SO}(3)$ to $\mathbf{SE}(3)$, predict the utility of unseen region, and optimize planning strategy. In section II, SLAM and Active SLAM problems are introduced and formulated. In section III and IV, we show the details of our system design and our experiments. In the end, we summarize this paper by discussing the pros and cons of the system and outlining possible future improvements.

II. BACKGROUND

A typical autonomous robot performs three basic tasks: 1) sense and model the world, 2) plan accordingly and control itself to interact with the world, 3) learn how to sense, plan, and control from data or feedback collected in a closed-loop system. SLAM mainly focuses on perception and learning in perception while active SLAM builds on SLAM and is able to intelligently plan and control itself. We will firstly discuss the problem of SLAM and then show how active SLAM can be built based on SLAM.

A. Simultaneous Localization and Mapping

In a SLAM system, an agent localizes itself and construct a map of the environment at the same time. The word ”Simultaneous” is highlighted because localization and mapping cannot be done separately. An agent cannot know the location of landmarks and cannot build an accurate global map without understanding its location. Similarly, without understanding the environment, an agent cannot know where it is in the environment.

In a SLAM problem, we want to estimate the most likely robot locations $X = \{x_0, x_1, \cdots, x_T\}$ for timestep 0 to T and landmarks locations $\Theta = \{\theta_0, \theta_1, \cdots, \theta_n\}$ given robot odometry data $U = \{u_0, u_1, \cdots, u_T\}$ and robot measurements $Z = \{z^0, z^1, \cdots, z^T\}$. Each $z^t \in Z$ contains a set of observations $\{z_0^t, z_1^t, \cdots\}$ of landmarks $\{\theta_0^t, \theta_1^t, \cdots\} \in \mathcal{P}(\Theta)$. To estimate robot locations, we have a motion model

$$P(x_t \mid x_{t-1}, u_{t-1}) \sim N(m(x_{t-1}, u_{t-1}), \Sigma_t) \qquad (1)$$

*This work is funded by the Wharton Summer Program of Undergraduate Research

*Advisor: Professor Pratik Chaudhari, Electrical and Systems Engineering department and the GRASP Robotics Laboratory, University of Pennsylvania

[1]Siming He is a Wharton and SEAS Undergraduate student in Statistics and Computer Science, University of Pennsylvania `siminghe@wharton.upenn.edu`

where m is a function to calculate x_t from x_{t-1} and u_{t-1}. In addition, we have an observation model

$$P(z_k^t \mid x_t, \theta_k^t) \sim N(l(x_t, \theta_k^t), S_t) \tag{2}$$

where l is a function to calculate z_k^t from x_t and θ_k^t. Overall, it is a smoothing problem of the joint belief distribution (Details in APPENDIX-A):

$$\begin{aligned} Bel(X, \Theta) &= P(X, \Theta \mid Z, U) \\ &\propto P(x_0) \cdot \prod_{i=1}^{T} P(x_i \mid x_{i-1}, u_{i-1}) \\ &\quad \cdot \prod_{i=0}^{n} P(\theta_i) \prod_{j=0}^{T} P(z^j \mid x_j, \theta_i) \cdot \mathbf{1}_{z_j}(\theta_i) \end{aligned} \tag{3}$$

where $\mathbf{1}_z(\theta)$ is an indicator function that equals to 1 if z is an observation of θ and 0 otherwise. Hence, the joint belief can be represented as a graphical model as in Fig. 1 and we can use graph-based optimization. GTSAM [4]

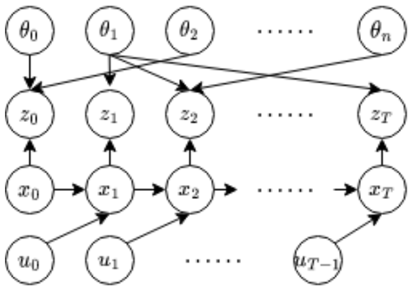


Fig. 1: Graphical Model of Joint Belief of States

is a library that can efficiently construct and optimize such graphs. GTSAM uses factor graphs to model the smoothing problem [4]. A factor graph is a bipartite graph where variables are random variables we want to estimate and factors are probabilistic constraints on the variables [4]. We can convert the above graph formulation into a factor graph formulation. We have two types of probabilistic constraints: binary motion factor $f_i(x_i, x_{i-1}; u_{i-1})$ and binary measurement factor $g_i(x_i, \theta_k^i; z_k^i)$ based on Eq. (1) and Eq. (2) respectively. Hence, we have

$$\begin{aligned} Bel(X, \Theta) = f'(x_0) &\cdot \prod_{i=0}^{n} g'(\theta_i) \cdot \prod_{i=1}^{T} f_i(x_{i-1}, x_i; u_{i-1}) \\ &\cdot \prod_{i=0}^{n} \prod_{j=0}^{T} g_j(x_j, \theta_i; z^j) \cdot \mathbf{1}_{z^j}(\theta_i) \end{aligned} \tag{4}$$

where $f'(\cdot)$ and $g'(\cdot)$ are unitary factors that represent priors about robot and landmarks locations. Therefore, we get the factor graph in Fig. 2. Hence, SLAM problems can be formulated as a maximum a posteriori estimation of the

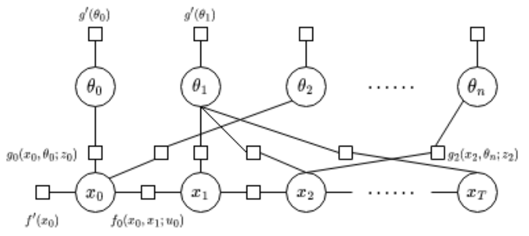


Fig. 2: Factor Graph of Joint Belief of States

factor graph, i.e.

$$\begin{aligned} X^*, \Theta^* &= \text{argmax}_{X,\Theta} \log f(X, \Theta) \\ &= \text{argmax}_{X,\Theta} \sum_{i=1}^{T} M^T \Sigma_{i-1}^{-1} M + \sum_{i=0}^{n} \sum_{j=0}^{T} L^T S_j L \end{aligned} \tag{5}$$

where $M = x_i - m(x_{i-1}, v_{i-1})$ and $L = \theta_i - l'(x_j, z_i)$ where l' is the inverse observation model of l. This estimation can be solved using the Gauss–Newton method or the Levenberg-Marquardt method [4].

Kimera SLAM library [2] adopted in our system uses IMU on-manifold preintegration [5] to get motion factors. Kimera [2] gets measurement factors by using Shi-Tomasi Corner Detection [6] to identify good landmarks, using Lucas-Kanade method [7] to calculate the motion of landmarks between frames, and using RANSAC [8] to verify the matching of landmarks in different frames. It also uses DBoW2 library [9] for loop-closure detection and loop-closure motion factors. The output factor graph is optimized by using iSAM structureless vision model in GTSAM toolbox [2],[4],[5].

B. Environment Representation

A dense point cloud is generated from RGBD data of the RealSense D435i Camera. We want to discretize the point cloud into some voxel world representation where voxel (smallest cube) in 3D space corresponds to pixel (smallest square) in 2D space.

In occupancy grid representation, each voxel is a binary random variable which shows the probability of the voxel being occupied. A point cloud is used to update the probability of each voxel being occupied through a binary Bayes filter. Another type of representation is signed distance field in which each voxel contains a distance from the voxel to the surface. Truncated signed distance field (TSDF) contains the distance from the voxel to the surface along the ray from the viewpoint to the voxel. The distance is calculated by

$$d(x, p, s) = \|p - x\| \text{sign}((p - x) \cdot (p - s)) \tag{6}$$

where $x, p, s \in \mathbb{R}^3$ are the center position of the voxel, the position of the point in the point cloud, and the position of the sensor respectively. It is called truncated SDF because we truncate the field and only voxels near the surface are kept (Fig. 3). The reason of truncating the field is that TSDF is used for describing the surface of the environment and voxels outside the truncated region are not useful. Euclidean signed distance field (ESDF) contains the distance from the voxel

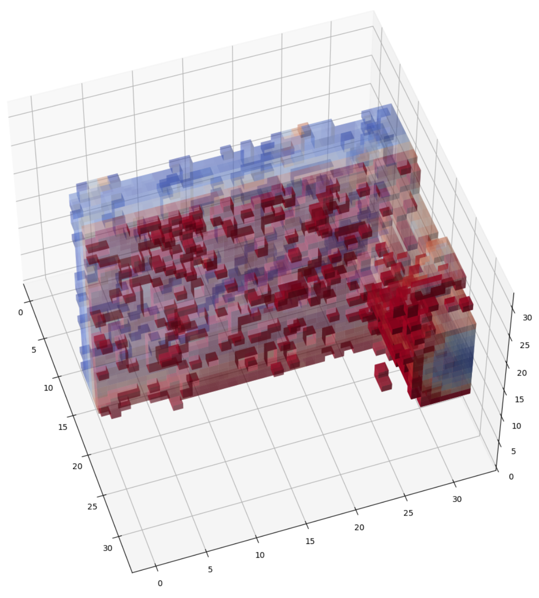


Fig. 3: TSDF truncated at 25 centimeters around the surface
Red voxels are outside and blue voxels are inside the surface.

to the closest surface and is used for obstacle avoidance. Signed distance field is a better representation of accurate maps since the distances allow interpolating the surface.

In our system, Voxblox [3] is used to get TSDF from point cloud for mapping. Voxel hashing [10] is used to save memory when constructing a map at scale. TSDF in Voxblox is different from the definition above because the distance in its TSDF is an estimation of the shortest distance from the surface. The estimation is

$$D_{i+1}(x,p,s) = \frac{W_i(x)D_i(x) + w(x,p)d(x,p,s)}{W_i(x) + w(x,p)} \tag{7}$$

where $w(x,p) = \frac{1}{z^2}$ for voxels near and outside the surface, z is the depth measurement of p, and $W_{i+1}(x,p) = \min(W_i(x) + w(x,p), W_{max})$ [3]. This estimation is good as the error reduced down below a quarter of the voxel size with $p = 0.95$ as the number of observations increases [3]. The weight W_i is similar to log odds since both log odds update and $w(x,p)$ update is inversely proportional to some power of z as shown in APPENDIX-B. The increase in weight means the increase of certainty about the voxel.

C. Active SLAM

Active SLAM can be formulated as a Partially Observable Markov Decision Process (POMDP) which contains $(S, A, T, R, \gamma, \Omega, O)$:

$$\begin{aligned}
&S : \text{set of states } X \times \Theta \\
&A : \text{set of actions of the agent} \\
&\Omega : \text{set of observations } U \times Z \\
&\gamma : \text{discount factor} \\
&T(s', r', s, a) = P(S_{t+1} = s', R_{t+1} = r' \mid S_t = s, A_t = a) \\
&R(s,a) = E(R_{t+1} \mid S_t = s, A_t = a) \\
&O(s,z) = P(Z_{t+1} = z \mid S_{t+1} = s)
\end{aligned}$$

Since we already have the SLAM algorithms that estimate the belief about states, we can convert the POMDP into a belief MDP:

$$\begin{aligned}
&B : \text{Bel}(x) = P(X, \Theta \mid Z, U) \\
&T_B(b', r', b, a) = P(B_{t+1} = b', R_{t+1} = r' \mid B_t = b, A_t = a) \\
&\qquad\qquad = \sum_{z \in \Omega} P(b', r' \mid b, a, z) \cdot P(z \mid b) \\
&P(z \mid b) = \sum_{s' \in S} O(s,z) \cdot \sum_{s \in S} T(s', r', s, a) \cdot \text{Bel}(s) \\
&R_B(b,a) = \sum_{s \in S} \text{Bel}(s) \cdot R(s,a)
\end{aligned}$$

with A and γ same as the ones in POMDP. We want to find a policy $\pi^*(b)$ that generates a sequence of actions $\{a_t, a_{t+1}, \cdots, a_{t+k}\}$ from belief states such that $\sum_{i=0}^{k} R_B(\text{Bel}(s_{t+i}), a_{t+i})$ is maximized. In a belief MDP, $T_B(b', r', b, a)$ can be considered as a world model that predicts the future belief states and rewards. In our problem, $R_B(b,a)$ can be considered as a competence map that predicts the map's competence/accuracy/certainty at given belief states [12]. An active SLAM system constantly learns the world model, the competence map, and the policy based on real-world data [12], [13] as demonstrated in Fig. 4. Hence, the choices of model, competence map, and policy

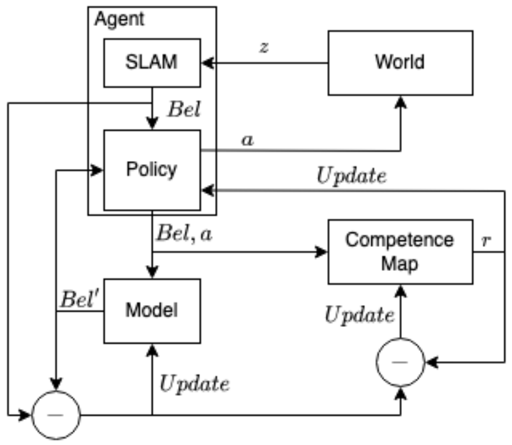


Fig. 4: Active SLAM System

are important for active SLAM.

III. SYSTEM DESIGN

A. Assumptions

The implementation of the whole system in Fig. 4 is too complicated for this paper. Hence, we made some assumptions to simplify the system:

- An agent has perfect knowledge of its pose i.e. there is no need to consider the quality of localization during exploration.
- An agent can only change its roll, pitch, and yaw at a fixed location (x, y, z).
- Sampling and going to the next-best-view is a good exploration strategy.
- Unseen regions are considered to have the same information gain.

We discuss why some of the assumptions don't work well and how to improve them in section V. Such surface-based next-best-view methods are used in several past research about active exploration. Papers [14], [15], [16], [17] all create some utility functions based on information gain of voxels and cost of movement. They sample viewpoints, evaluate them by the utility functions to decide the next best view, and move to the next best view.

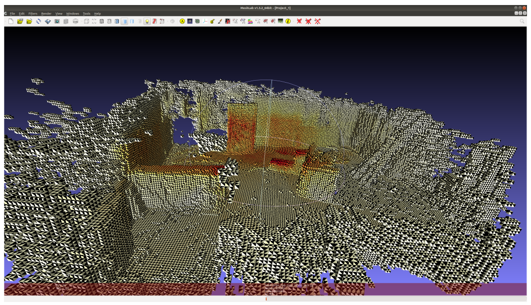


Fig. 5: Voxelblox TSDF Output

B. System

1) SLAM: Kimera-VIO [2] is used for SLAM given stereo camera data from Intel RealSense Depth Camera D435i.
2) Environment Representation: Dense point could data from RealSense D435i is converted to TSDF and ESDF by Voxblox library [3]. Fig. 5 visualizes the surface of the voxel representation of EuRoC V101easy visual-inertial data [18] where redder color means higher certainty of the voxel.
3) Ray Tracing: Firstly, we define three types of voxels that we are interested in based on the definition in [14]. New Voxels are voxels in regions that haven't been explored. Frontier Voxels are voxels that are on the surface of any explored objects with any New Voxels as neighbors. Surface Voxels are voxels that are on the surface of any explored objects and are not Frontier Voxels. Then, we use an efficient voxel traversal algorithm for ray tracing [19] to get all voxels that are in the current viewpoint. To speed up the process, we avoided tracing rays that hit the same voxel by checking ray and axis-aligned bounding box intersection [20]. A detailed explanation of this optimization is covered in APPENDIX-C.
4) Utility Function: We design a utility function similar to the past papers [14], [15], [16], [17]:

$$U_{surf}(V) = \sum_{v \in V_{surface}} \left(\frac{w_{new}(v)}{w_{new}(v) + W(v)}\right) \tag{8}$$

$$U_{front}(V) = |V_{frontier}| \cdot c_1 \tag{9}$$

$$U_{new}(V) = |V_{new}| \cdot c_2 \tag{10}$$

$$U(V) = \frac{U_{surf}(V) + U_{front}(V) + U_{new}(V)}{(|dYaw| + |dPitch|) \cdot |V|} \tag{11}$$

where V is the set of voxels in the viewpoint, $V_{surface}, V_{frontier}, V_{new}$ are the sets of surface, frontier, and new voxels respectively, $|dYaw|, |dPitch|$ are the change of yaw and pitch from the current viewpoint to the goal viewpoint, $w(v)$ is the same in Eq. 7, c_1, c_2 are constants which are set to 0.1 in this paper.
5) Sampling: We sample yaw uniformly from $[0, 2\pi]$ and sample pitch uniformly from $[-\frac{\pi}{6}, \frac{\pi}{6}]$ twenty times at each time step. Then, we pick the next viewpoint as the sampled viewpoint with the highest utility gain.
6) PID Controller: The goal viewpoint is the input of the PID Controller which outputs and sends roll, yaw, pitch, and throttle values to the flight controller of the drone.

IV. EXPERIMENT

A. Ray Tracing Results

Fig. 6 shows the result of ray tracing. The red pyramid represents the viewpoint and field of view. Cyan voxels, purple voxels, and yellow voxels are respectively surface, new, and frontier voxels visible from the viewpoint. Since the depth measurement error increases with distance, I limit the maximum distance of ray tracing to 2.5 meters. Such a maximum distance bounds the depth measurement error and runtime of ray tracing. After speeding up ray tracing through the method in APPENDIX-C, the ray tracing time in Fig. 6 is reduced by 52.13%.

B. Information gain

As shown in Fig. 7, different viewpoints have different information gain where redder color means higher gain. We observe that viewpoints that contain more voxels within 2.5 meters have higher information gain. Moreover, viewpoint B has higher information gain than viewpoint A, i.e. a viewpoint that contains voxels with higher uncertainty has higher information gain. Specifically, let V_A and V_B be the sets of voxels visible at viewpoint A and B. $V_A - V_B$ is the set of voxels in R and $V_B - V_A$ is the set of voxels in R'. We observe that $V_A - V_B$ has more certainty than $V_B - V_A$ since region R is redder than R' in Fig. 7. Therefore, V_B should have higher information gain than V_A which is aligned with the observation.

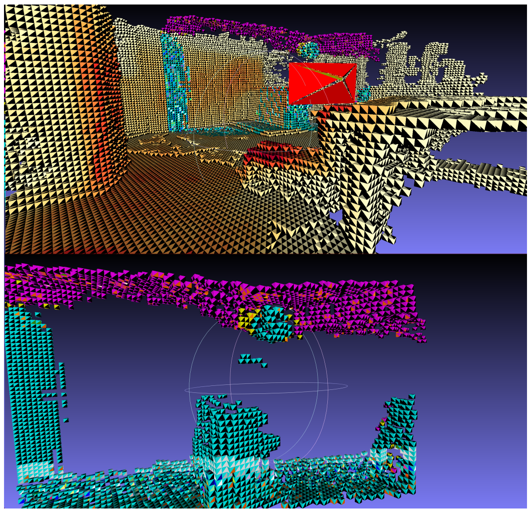


Fig. 6: Ray Tracing Result at Given Viewpoint

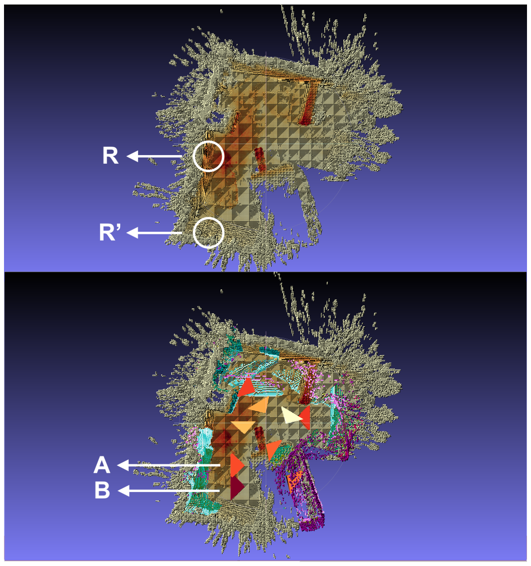
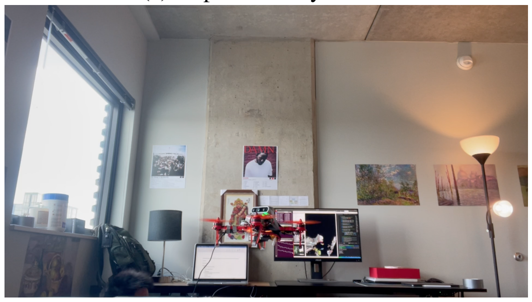


Fig. 7: Information Gain at Different Viewpoint

(a) Exploration by Drone



(b) Mapping Result

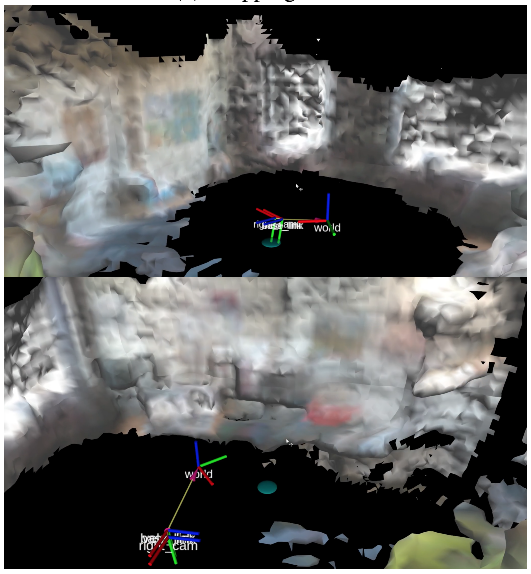


Fig. 8: Exploration and Mapping

C. *Exploration with Drone*

The PiDrone [21] is used as the actual robot to do exploration and mapping. In addition to the sensors of PiDrone, we mount Intel RealSense Depth Camera D435i on the drone to collect stereo and depth image data. Fig. 8 (a) shows the PiDrone during exploring and mapping and Fig. 8 (b) shows the mapping result. Fig. 9 shows the cumulative map surface gain and number of voxels during our experiment. The surface gain is the increase of weight of all surface voxels at the current viewpoint. The gain jumps up twice in the plot, possibly because the drone is mapping the area of windows which has high uncertainty. The log number of voxels seen at each viewpoint shows that the number of surface voxels is approximately the same as the drone rotating at a constant speed. The number of frontier voxels and new voxels changes when exploring different areas. After 150 seconds, there is almost no frontier and new voxel because all voxels have been explored. The gain is still increasing after 150 seconds because the explored map is improving as the agent re-visits the areas but the gain is lower.

V. CONCLUSION

In this paper, we implement an active SLAM system and test it on a drone. The experiment results show that the system is able to explore the environment and construct a map. However, we can see in Fig. 8 (b) that the map is still not very detailed and accurate and improvements are needed in the future. During explorations, the sampled next-best-views are sometimes not good as we can see in Fig. 9. These problems are mainly caused by some of the assumptions we made in Section III A. Hence, we need to change some of the assumptions in the future:

1) Our assumption that the agent has perfect knowledge of its pose is unrealistic since the actual robot is not able to know its exact location. Moreover, localization uncertainty needs to be considered in exploration since localization error could be larger than depth measurement error. Poor localization would cause inaccurate mapping results.
2) In this paper, we only consider the change of roll, yaw, and pitch with a fixed robot location. It is a good

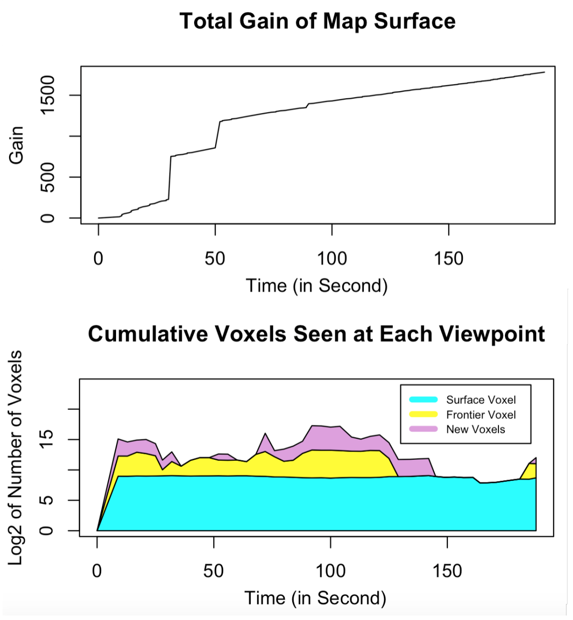


Fig. 9: Gains and Voxels at Each Timestep

simplification which reduces the state space that we sample from. However, with a fixed location, an agent is not able to move closer or to different angles to construct a more detailed and accurate map. We need to sample both rotation and translation and use ESDF for $\mathbf{SE}(3)$ trajectory planning to avoid collisions.

3) After expanding the state space to include different locations, methods of sampling would become a challenge because a small amount of samples would cause non-optimal planning and a large amount of samples would increase the time of sampling. One solution would be sampling around the frontier [14], [15], [16], [17] which is likely to have high information gain. However, with this solution, an agent is not able to re-explore explored areas that might have high information gain. Another solution would predict regions with high uncertainty and sample more from these regions.
4) A viewpoint may have low cumulative information gain but a set of voxels in the viewpoint has high information gain. In this case, we need a method to know that such a viewpoint may also be worth visiting.
5) Next-best-view sampling is sub-optimal even if we have a good sampling method. Other planning and control methods may be considered for better exploration results.
6) Unexplored regions have a fixed information gain in this system. It is not a good approximation since it doesn't show which unexplored region to go to especially in larger state space. Prediction of unseen regions or information gain of unseen regions would be needed.
7) This system only cares about constructing a geometric map. It would be important to construct a hybrid map that contains both geometric and semantic details, i.e. we need to incorporate semantic information and uncertainty into the belief state space.

References

[1] J. A. Placed, J. Strader, H. Carrillo, N. Atanasov, V. Indelman, L. Carlone, and J. A. Castellanos, "A Survey on Active Simultaneous Localization and Mapping: State of the Art and New Frontiers," *arXiv:2207.00254 [cs.RO].*

[2] A. Rosinol, M. Abate, Y. Chang and L. Carlone, "Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping," *in IEEE Intl. Conf. on Robotics and Automation (ICRA), 2020.*

[3] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning," *in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017.*

[4] F. Dellaert, "Factor Graphs and GTSAM: A Hands-on Introduction," *Technical Report number GT-RIM-CPR-2012-002.*

[5] C. Forster, L. Carlone, F. Dellaert and D. Scaramuzza, "On-Manifold Preintegration for Real-Time Visual–Inertial Odometry," *in IEEE Transactions on Robotics, vol. 33, no. 1, pp. 1-21, Feb. 2017, doi: 10.1109/TRO.2016.2597321.*

[6] J. Shi and C. Tomasi, "Good features to track," *in IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 1994, pp. 593–600.*

[7] J. Bouguet, "Pyramidal implementation of the Lucas Kanade feature tracker," *in Intel Corporation, Microprocessor Research Labs, 2000.*

[8] B. Horn, "Closed-form solution of absolute orientation using unit quaternions," *J. Opt. Soc. Amer., vol. 4, no. 4, pp. 629–642, Apr 1987.*

[9] D. Gálvez-López and J. D. Tardós, "Bags of binary words for fast place recognition in image sequences," *in IEEE Transactions on Robotics, vol. 28, no. 5, pp. 1188–1197, October 2012.*

[10] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-Time 3D Reconstruction at Scale Using Voxel Hashing," *in ACM Transactions on Graphics (TOG), vol. 32, no. 6, p. 169, 2013.*

[11] M. S. Ahn, H. Chae, D. Noh, H. Nam, and D. W. Hong, "Analysis and Noise Modeling of the Intel RealSense D435 for Mobile Robots," *in 2019 16th International Conference on Ubiquitous Robots (UR), pp. 707-711.*

[12] S. B. Thrun and K. Möller, "Active Exploration in Dynamic Environments," *1992.*

[13] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence, Volume 101, Issues 1–2, 1998, Pages 99-134.*

[14] Y. Kompis, L. Bartolomei, R. Mascaro, L. Teixeira and M. Chli, "Informed Sampling Exploration Path Planner for 3D Reconstruction of Large Scenes," *in IEEE Robotics and Automation Letters, vol. 6, no. 4, pp. 7893-7900, Oct. 2021, doi: 10.1109/LRA.2021.3101856.*

[15] L. M. Schmid, M. Pantic, R. Khanna, L. Ott, R. Y. Siegwart, and J. I. Nieto, "An Efficient Sampling-Based Method for Online Informative Path Planning in Unknown Environments," *IEEE Robotics and Automation Letters, 2020, vol. 5, pp. 1500-1507.*

[16] R. Border, J. D. Gammell, P. Newman, "Surface Edge Explorer (SEE): Planning Next Best Views Directly from 3D Observations," *IEEE Press, 10.1109/ICRA.2018.8461098.*

[17] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova and R. Siegwart, "Receding Horizon "Next-Best-View" Planner for 3D Exploration," *2016 IEEE International Conference on Robotics and Automation (ICRA), 2016, pp. 1462-1468, doi: 10.1109/ICRA.2016.7487281.*

[18] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. Achtelik and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *International Journal of Robotic Research, DOI: 10.1177/0278364915620033, 2016.*

[19] J. Amanatides and A. Woo, "A Fast Voxel Traversal Algorithm for Ray Tracing," *In Eurographics '87, 1987, pp. 3-10.*

[20] A. Williams, S. Barrus, R. Keith, and M. P. Shirley, "An efficient and robust ray-box intersection algorithm," *Journal of Graphics Tools, 2003, vol. 10, p.54.*

[21] I. Brand, J. Roy, A. Ray, J. Oberlin and S. Oberlix, "PiDrone: An Autonomous Educational Drone Using Raspberry Pi and Python," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 1-7, doi: 10.1109/IROS.2018.8593943.*

APPENDIX

A. Decompose Belief of States

$$
\begin{aligned}
Bel(X, \Theta) &= P(X, \Theta \mid Z, U) \\
&= P(\Theta \mid X, Z, U) \cdot P(X \mid Z, U) \\
&= P(x_0, \cdots, c_T \mid z^0, \cdots, z^T, u_0, \cdots, u_T) \\
&\quad \cdot \prod_{i=0}^{n} P(\theta_i \mid x_0, \cdots, x_T, z^0, \cdots, z^T) \\
&\propto P(x_0) \cdot \prod_{i=1}^{T} P(x_i \mid x_{i-1}, u_{i-1}) \\
&\quad \cdot \prod_{i=0}^{n} P(z^0, \cdots, z^T \mid x_1, \cdots, x_T, \theta_i) P(\theta_i) \\
&\propto P(x_0) \cdot \prod_{i=1}^{T} P(x_i \mid x_{i-1}, u_{i-1}) \\
&\quad \cdot \prod_{i=0}^{n} P(\theta_i) \prod_{j=0}^{T} P(z^j \mid x_j, \theta_i) \cdot \mathbf{1}_{z_j}(\theta_i)
\end{aligned}
$$

B. Log Odds Update and Weight Update

The log odds is

$$
\begin{aligned}
L(d_t \mid z_t) &= \log\left(\frac{f_{N(d_t,\sigma)}(d_t)}{1 - f_{N(d_t,\sigma)}(d_t)}\right) \quad \sigma = a z_t^2 + b z^t + c \\
&\propto \frac{d_t^2}{z_t^4} + \frac{d_t^2}{z_t^4 - z_t^5}
\end{aligned}
$$

The σ used in the calculation is based on the noise model of RealSense Depth Camera which is quadratic as a function of distance [11]. It is indeed similar to the weight update of W_t in Eq. (7).

C. Speeding Up Ray Tracing Algorithm

Since the accuracy of depth measurement decreases as distance increases, we only care about voxels that are less than 2.5 meters from the viewpoint. Another advantage of setting this maximum distance is that it bounds the runtime i.e. a fixed number of voxels would be traversed in the worst case. We call the plane that is 2.5 meters away the end plane. We sample rays from viewpoint to every voxel on the end plane. Then, we do ray tracing to find all hitted voxels. One observation is that several neighboring rays would be highly likely to hit the same voxel especially when the voxel is closer to the viewpoint. In this case, we waste time tracing all the neighboring rays without getting any new hitted voxel. To solve this problem, for each ray, we check if the ray intersects with any voxels hitted by its neighboring rays. There is at most eight neighbors to check. It takes much less time than traversing through the voxels (50 voxels if the hitted voxel is 2.5 meters from the viewpoint) for ray tracing. Another two observations are 1) a ray can only hit the same voxel of its neighbors if not hit a new voxel, 2) if a ray intersects with a voxel of its neighbor, it must hit such voxel. A 2D example is shown in the following figure.

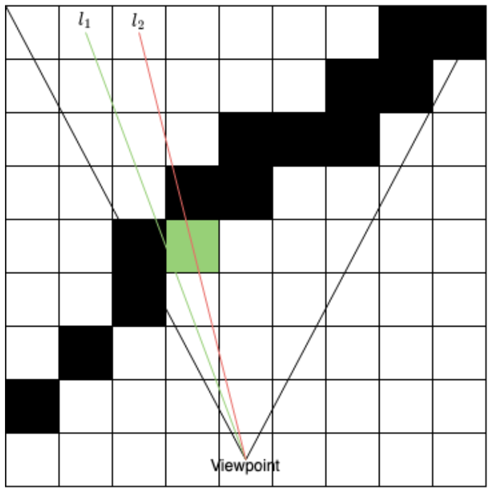


Fig. 10: 2D Ray Tracing Example

l_1 is the traced ray and we know from l_1 that the green voxel v is occupied. Before we trace $l2$, we check voxel v hitted by its neighbor $l1$. Since l_2 and v intersect, we can skip tracing $l2$.