MARSIM: A Light-Weight Point-Realistic Simulator for LiDAR-Based UAVs

Fanze Kong, Xiyuan Liu, *Graduate Student Member, IEEE*, Benxu Tang, Jiarong Lin, Yunfan Ren, *Graduate Student Member, IEEE*, Yixi Cai, *Graduate Student Member, IEEE*, Fangcheng Zhu, *Graduate Student Member, IEEE*, Nan Chen, and Fu Zhang, *Member, IEEE*

***Abstract*—The emergence of LiDAR sensors have brought new opportunities for autonomous unmanned aerial vehicles (UAVs) by advancing navigation safety and computation efficiency. Yet the successful developments of LiDAR-based UAVs must rely on extensive simulations. Existing simulators can hardly perform simulations of real-world environments due to the requirements of dense mesh maps that are difficult to obtain. Therefore, we develop a point-realistic simulator of real-world scenes for LiDAR-based UAVs. The key idea is the underlying point rendering method, where we construct a depth image directly from the point cloud map and interpolate it to obtain realistic LiDAR point measurements. Our developed simulator is able to run on a light-weight computing platform and supports the simulation of LiDARs with different resolution and scanning patterns, dynamic obstacles, and multi-UAV systems. Developed in the ROS framework, the simulator can easily communicate with other key modules of an autonomous robot, such as perception, state estimation, planning, and control. Finally, the simulator provides 10 high-resolution point cloud maps of various real-world environments, including forests of different densities, historic building, office, parking garage, and various complex indoor environments. Evaluation results show that the developed simulator achieves superior performance in terms of time and memory consumption against Gazebo and that the simulated UAV flights highly match the actual one in real-world environments. We believe such a point-realistic and light-weight simulator is crucial to bridge the gap between UAV simulation and experiments and will significantly facilitate the research of LiDAR-based autonomous UAVs in the future.**

***Index Terms*—Aerial systems: simulator, lidar, perception and autonomy.**

I. Introduction

RECENT developments of LiDAR technologies have significantly lowered the cost and weight of LiDAR sensors, which creates many opportunities for unmanned aerial vehicle (UAV) applications, such as mine exploration [1], biological data statistics [2], mapping [3], high-speed navigation [4], and obstacle avoidance, etc. However, deploying UAVs to these widespread applications requires extensive tests, which are often cost-demanding since the system under test are still in active development and hence may have a noticeable failure rate (e.g., collision with the environment). A simulator that resembles the reality can significantly reduce the time and equipment cost occurred in UAV tests and has become a crucial component of UAV developments.

Existing simulators (e.g., Gazebo [5], Webots [6], Airsim [7]) have difficulties meeting the demand of high-resolution realistic scene simulation for LiDAR-based UAVs due to the following

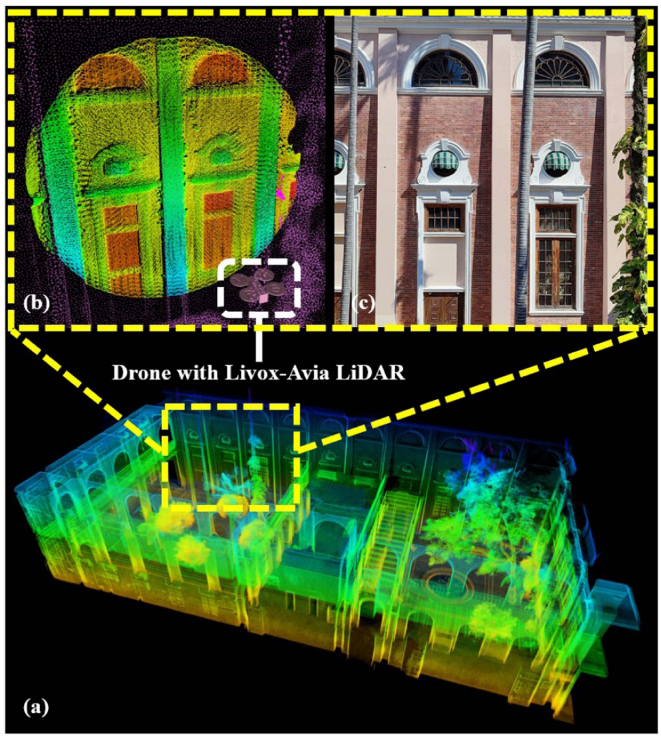


Fig. 1. A demo of MARSIM. (a) The point cloud map of the HKU main building (one of ten real-world scenes of MARSIM). (b) A scan of points of a Livox Avia LiDAR rendered directly from the point cloud map by MARSIM. (c) The photo of the corresponding scene in the real world. It is seen our proposed simulator can restore the structural details of the real scene with high quality (more details can be found on https://youtu.be/dVUi9jQled0).

Manuscript received 16 November 2022; accepted 16 March 2023. Date of publication 3 April 2023; date of current version 11 April 2023. This letter was recommended for publication by Associate Editor I. Karamouzas and Editor A. Bera upon evaluation of the reviewers' comments. This work was supported by the University Grants Committee of Hong Kong through the General Research Fund scheme under Grant 17206421 and a DJI Donation Fund. *(Corresponding author: Fu Zhang.)*

Fanze Kong, Xiyuan Liu, Jiarong Lin, Yunfan Ren, Yixi Cai, Fangcheng Zhu, Nan Chen, and Fu Zhang are with the Department of Mechanical Engineering, University of Hong Kong, Hong Kong (e-mail: kongfz@connect.hku.hk; xliuaa@connect.hku.hk; jiarong.lin@hku.hk; renyf@connect.hku.hk; yixicai@connect.hku.hk; zhufc@connect.hku.hk; cnchen@connect.hku.hk; fuzhang@hku.hk).

Benxu Tang is with the School of Mechanical Engineering and Automation, Harbin Institute of Technology, Harbin 150001, China (e-mail: 180320222@stu.hit.edu.cn).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2023.3264163, provided by the authors.

Digital Object Identifier 10.1109/LRA.2023.3264163

limitations: (i) their simulated environments are mostly virtual, unrealistically simple, and man-made, which possess a considerable gap from complex real-world scenes; (ii) they only import mesh maps, which are difficult to obtain from real-world environments that are often measured in 3D point clouds by laser scanners or LiDARs. To the best of our knowledge, there are no open-source and mature tools available for generating high-resolution and high-fidelity mesh maps out of point cloud data. The commonly-used Poisson reconstruction [8] method is time-consuming and has low-quality meshes on real point cloud data captured by LiDARs due to occlusions and point density variations in large scene scanning; (iii) they often rely on high-performance GPUs to achieve real-time simulations in large complex mesh maps, which puts a high requirement for computing platforms.

Motivated by these gaps, in this paper, we propose a light-weight LiDAR-based UAV simulator, which has the following features:

1) Directly utilizing point cloud maps reconstructed from real environments for LiDAR scan rendering. The point cloud map contains fine details of the environments and could be easily obtained with a LiDAR.
2) High efficiency in computation and memory consumption, and the ability to run on personal computers without a dedicated graphics processing unit (GPU).
3) Versatility in supporting the simulation of three types of dynamic obstacles, multi-UAV systems (with configurable flight control modules), and various solid-state and mechanical spinning LiDAR models with adjustable parameters (angular resolutions, scanning patterns, field-of-views, sensing ranges, etc.).
4) We open-source our code on GitHub,[1] which is ROS-compatible and can be easily integrated with SLAM and path-planning algorithms to benefit the community.

II. Related Works

LiDAR-based UAV simulation consists of UAV motion simulation and LiDAR simulation. Compared to the LiDAR simulation, the UAV motion simulation is rather straightforward and mature. MATLAB can support many types of UAVs' motion simulation and controller design, such as quadrotors [9] and VTOLs [10]. Common simulators such as Gazebo [5] and Airsim [7] are also able to simulate the movement of UAVs at high frequencies. Wei et al. [11] utilize the Gazebo simulation to develop a multi-UAV path planning algorithm, and Han et al. [12] simulate a UAV SE(3) planning in Airsim and provide a benchmark for autonomous UAV racing.

The main challenge of LiDAR-based UAV simulation lies in the LiDAR simulation. There are many existing simulators that support LiDAR simulations, such as Gazebo [5], Webots [6], Airsim [7] and, SVL [13]. Gazebo has been widely used for the simulation of mobile robots and the verification of autonomous LiDAR-based exploration algorithms, such as GBP [1], MBP [14], TARE [15] and Splatplanner [16]. SVL simulates a LiDAR mounted on a vehicle rooftop in an urban environment for the application of autonomous driving. The main drawback of these simulators is that they import maps in the form of mesh models, which are often available for artificial environments modeled by 3D modeling software (e.g., Sketchup, Blender, 3DS Max, etc.) or built based on Gazebo model library. For instance, Splatplanner made eleven artificial maps in Gazebo [16], and TARE made five larger artificial maps in Gazebo for algorithm verification. Besides, DARPA subterranean competition [17] provides a set of tunnel and mine maps where most of them are manual-made. Artificial, manual-made maps can meet some basic simulation requirements, but most of them are relatively simple and unrealistic, leading to a large gap from complex real-world environments.

A more common representation form of real-world environments is a point cloud map, which can be collected by devices such as 3D laser scanners or LiDAR sensors. To fill the gap between simulation and reality, some existing works attempt to provide realistic mesh maps from point clouds. For example, LiDARsim [18] generates realistic mesh maps using surfel-based method [19] for realistic self-driving simulation. Other methods have also been developed to construct realistic mesh maps from point clouds, such as Poisson reconstruction [8], Truncated Signed Distance Field (TSDF) volume method [20], and Delaunay triangulation method [21]. However, these methods are not robust to occlusion and point density variation that often occur in point clouds collected by LiDARs in large-scale scenes. In this case, non-existent surfaces may be falsely generated which require extra labor work to fix or tune the parameter. To overcome these issues, FlightGoggles [22] separately construct each individual object in the scene in a commercial software *Reality Capture* and then synthesize them with the background to build a highly realistic simulation environment. This approach is not very scalable since is very time- and labor-consuming to model each object and synthesize the scene (FlightGoggles [22] provides only two scenes). Finally, high-resolution mesh models are also challenging to render in real time without high-performance GPUs.

A question arises whether we really need mesh maps for LiDAR simulation. Mesh models have the advantage to attach material texture recovered from camera images, which can then render images for camera simulation. However, for LiDAR simulation, such texturing is not necessary. Moreover, thanks to the recent developments of low-cost, high-precision LiDAR sensors and simultaneous localization and mapping (SLAM) algorithms [23], [24], obtaining high-precision point clouds of real-world environments are becoming much more affordable and accessible. Motivated by this trend, we choose to directly use point cloud maps for simulation instead of mesh maps. A similar idea has been preliminarily explored in FUEL [25], but the presented simulator has limited resolution and accuracy that is suitable for only depth cameras in small scenes due to the high computation cost.

III. System Overview

As shown in Fig. 2, our UAV simulator is mainly composed of three submodules: a built-in flight controller module, a dynamics and kinematics simulation module, and a LiDAR simulation module (modules in black, see Fig. 2). The simulator is able to interact with planners, SLAM algorithms, and visualization modules in the ROS framework, forming a complete LiDAR-based UAV simulation system.

To use the simulator, users should first choose a LiDAR model and supply a point cloud map of the environment. Users can then plug in their own SLAM (or use ground-truth odometry) and Planner algorithms to the UAV simulator via the ROS topic

[1][Online]. Available: https://github.com/hku-mars/MARSIM.git

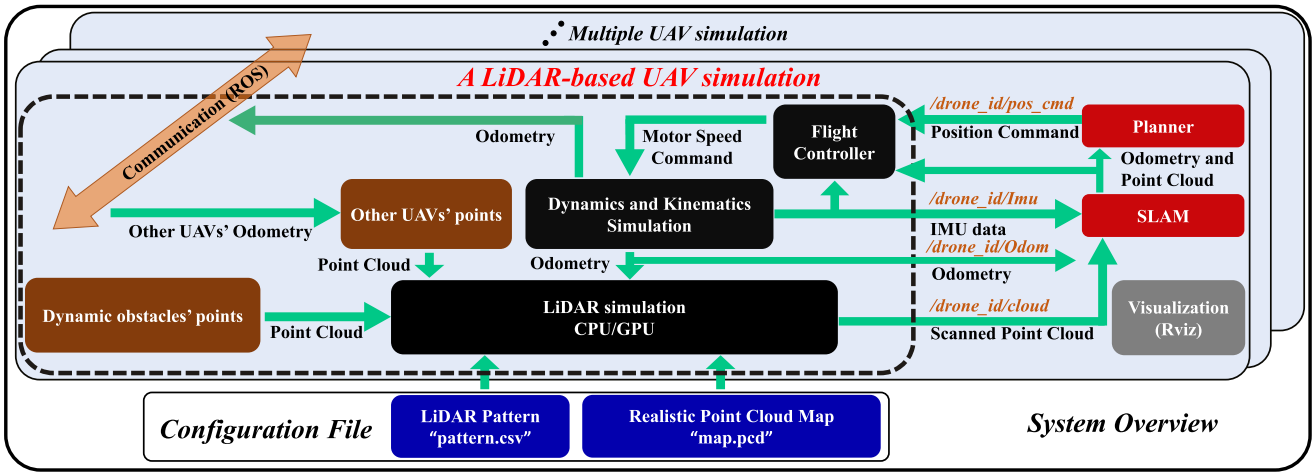


Fig. 2. The overall framework of our simulator (black dashed box) and how it interacts with external modules in ROS.

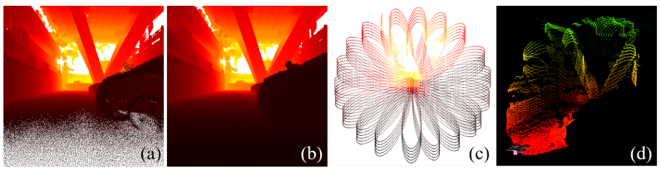


Fig. 3. Illustration of LiDAR simulation. (a) Is the sparse depth image which directly projects map points to the image. (b) Is the dense image after interpolation. (c) Shows the depth image masked with the LiDAR scanning pattern (e.g., a Livox Avia LiDAR). (d) Shows the final output LiDAR point clouds.

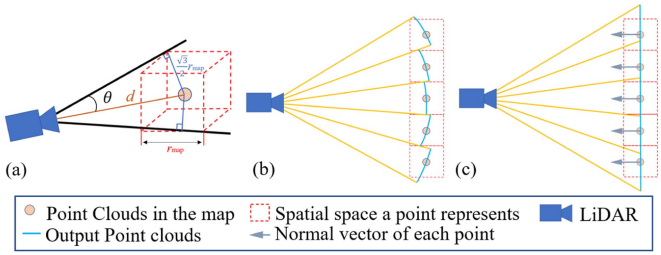


Fig. 4. The principle of occlusion culling and plane correction. (a) Demonstrates the geometric principle to compute the interpolation range of each point. (b) Demonstrates the problem of occlusion culling on large planes and (c) demonstrates the plane correction.

names shown in Fig. 2 for verification and visualization. Once the simulator starts, the dynamics and kinematics simulation module starts to compute the UAV's odometry and IMU data, according to which the LiDAR simulation module then renders the LiDAR scanned point cloud. The simulated IMU data and LiDAR scans are published in ROS, which could be used by the SLAM and then by the planner module. Besides the static environments represented by the point cloud map, the LiDAR simulation also simulates point measurements on dynamic obstacles and other UAVs in real time.

IV. Methodology

A. LiDAR Simulation

Given a point cloud representation of the environment (see Section V) and the current LiDAR pose, the LiDAR simulation module aims to render the points that should be measured in the current LiDAR scan. To do so, we first project all points of the point cloud map into the current LiDAR FoV and perform interpolation to obtain a dense depth image of θ_{res} angular resolution and then mask those points that are not on the scanning pattern. The remaining depth pixels are finally added with LiDAR measurements noises and transformed to point clouds for publishing (see Fig. 3). To accelerate the point projection process, we perform two key preprocessing when importing the point cloud map: 1) to limit the number of map points, we conduct spatial downsampling with a spatial resolution r_{map} on all map points; 2) we cut the map space into equal large voxels (with cube length l) and save the map points contained in each voxel in its respective point list. When rendering the LiDAR point cloud, we screen out voxels having intersection with the current LiDAR FoV, only points in these voxels are projected.

1) Occlusion Culling: While effectively limiting the number of map points, spatial downsampling will cause two problems. One is that the projected points do not uniformly populate the depth image due to the projection principle: points close to the LiDAR sensor are very sparse while those that at far are very dense, resulting in many empty depth pixels (as shown in Fig. 3(a)). Another problem is that after projected to the depth image, points on background objects will intervene in points on foreground objects and hence cause false depth measurements. To address these issues, we perform an occlusion culling process as follows.

Due to the spatial downsampling with resolution r_{map}, a point in the map should really represent a cube of substance with length r_{map} (see Fig. 4(a)). Therefore, the interpolation range θ_{max} around a map point $\mathbf{p}$ on the depth image is the projected area of the point's corresponding cube:

$$\theta_{\text{max}} = \arcsin\left(\frac{\frac{\sqrt{3}}{2} r_{\text{map}}}{d}\right) \tag{1}$$

where d is the depth of the map point $\mathbf{p}$. Pixels within the interpolation range θ_{max} will have their depth values set to d. If

the interpolation ranges of different map points have overlaps, the overlapped pixels will retain the smallest depth value.

2) Plane Correction: The occlusion culling works well when the LiDAR laser ray is perpendicular to the object's surface. However, when the laser ray is not perpendicular to the surface, the interpolated points will go out of the surface (see blue point clouds in Fig. 4(b)), which causes unexpected false point measurements. To address this issue, when interpolating the neighbor of a map point, we view the map point as a small plane and interpolate pixels in the interpolation range $\theta_{\max}$ by calculating the intersection point between the pixel ray and the plane (see Fig. 4(c)). The small plane around each map point is fitted from its neighboring points in the map during the preprocessing stage and the estimated plane normal is saved along with the map point. Since the depth calculation from a plane is more time-consuming, we perform such plane correction only for map points genuinely on a plane. This is achieved by introducing a new variable γ of each map point, which indicates the plane quality of the point. The value γ is computed as the plane thickness during the plane fitting and is saved along with the plane normal and map point. During online rendering, only map points with γ below a certain threshold will perform the plane correction.

3) GPU Acceleration: When the map resolution increases, the size of point clouds expands gradually to millions, and the time for projecting map points onto the depth image will increase considerably. In this case, only using the central processing unit (CPU) can no longer meet the real-time requirements. To address this issue, we utilize Graphics Processing Unit (GPU) hardware to accelerate the LiDAR simulation process. Affordable CPU-integrated or standalone GPUs are widely available in standard personal computer and hence does not degrade the generality of our simulator. They are also well supported by Open Graphics Library (OpenGL) [26], which is a multi-platform general graphics rendering library that can efficiently use GPU resources to project point clouds onto depth image in our task. We propose a point cloud parallel renderer based on OpenGL to accelerate the depth image construction. With GPU acceleration, the simulator is able to run in real time (more than 10 Hz) on point cloud maps with tens of millions of points in the map.

B. Dynamics and Kinematics Simulation

In order to simulate a realistic UAV flight, the simulator provides a dynamics and kinematics simulation of UAV according to the standard rigid-body model [27]. The thrust and torque in the dynamic model are generated from a second-order motor model whose command is the expected motor speed [28]. The model parameters (e.g., inertia matrix, mass, propeller torque, thrust coefficients, motor KV value) are drawn from [27] and are summarized in a separate configuration file where users may modify as needed. The computed angular velocity and special acceleration are added with measurement noises and biases to obtain the IMU measurements for publishing. The complete ground-true UAV state is also published for external reference.

C. Flight Controller Design

After the motion simulation of a UAV, a controller is needed to accurately control the UAV flight. Our simulator adopts a cascaded dual-loop PID controller as shown in [29], where the inner loop is an attitude controller, and the outer loop is a position controller. According to the UAV dynamic and kinematic model parameters, the controller gains are tuned based on the expected natural frequency and system damping ratio to achieve a good position control performance. Users can replace our position and attitude controllers with their own controllers according to their needs.

D. Dynamic Obstacles Simulation and Collision Detection

To mimic the real environment where dynamic obstacles may appear, the simulator supports the simulation of dynamic obstacles. We randomly generate a certain number of three types of dynamic obstacles: (i) UAVs moving at a constant speed in random directions; (ii) spherical objects moving in free falling trajectories; and (iii) small cubes moving in random walk trajectories. When one object moves out of the map boundary, a new one is generated at a random location in the map. The number, size, and speed of the objects can be adjusted by the users if needed.

To best simulate reality, the simulator conducts a high-precision collision check at each simulation step to detect if the UAV collides with obstacles (static or dynamic) in the environment. To achieve this, we establish a cube of the same size as the UAV within the UAV's body frame. We then divide the cube into voxels of equal size using the simulated map resolution. The state of each voxel is classified as either occupied or free, with voxels containing the UAV model's point clouds labeled as occupied and the others as free. To ensure the efficient indexing of the voxel information, we store the voxel information in a static hash table. During each collision check process, the neighboring points stored in two KD-Trees (one for the static environmental point cloud and the other for dynamic obstacles' point clouds) are searched and project into the UAV's body frame. If the neighboring points are projected into a voxel labeled as occupied, a collision event is considered to have occurred.

E. Decentralized Multiple UAV Simulation

A notable trend in UAV research is swarm navigation and control. To enable such research, our simulator supports the simulation of UAV swarm systems. To distribute the computation load and make the simulator more scalable to swarm size, the simulation is completely decentralized, where each UAV is simulated in one separate thread (as a ROS node) or computer. Different threads or computers communicate via ROS communication. This way, multiple UAV simulations can be distributed to multiple light-weight computers with local area network (LAN) connection. In order to simulate the interaction between multiple UAVs more realistically, the simulator adds a mutual observation function: at each simulation step, the point clouds of each realistic UAV model are added to the global point cloud map to render the LiDAR scans of the remaining UAVs.

V. Realistic High-Resolution Point Cloud Map Construction

To enable realistic interaction between the UAV and the environment in the simulator, we construct point cloud maps from actual scenes. In order to restore the realistic environment as much as possible, we put forward two requirements for the point cloud map: high resolution and high precision. To fulfill these requirements, we use a hand-held device carrying

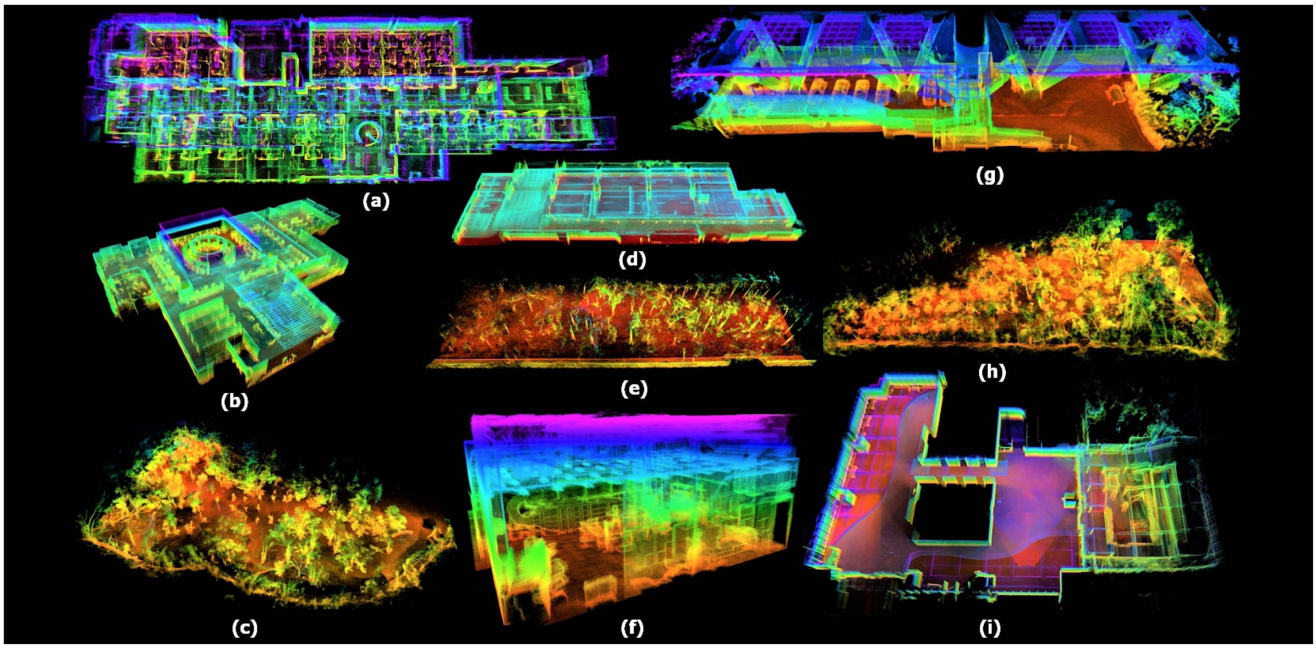


Fig. 5. High-resolution point cloud maps provided by the simulator with detailed information shown in Table I. (a), (b), (c), (d), (e), (f), (g), (h), and (i), respectively correspond to large office, Indoor-2, common forest, simple parking garage, simple forest, Indoor-1, complex parking garage, dense forest, Indoor-3 environments.

TABLE I
TEN REALISTIC POINT CLOUD MAPS AND THEIR FEATURES

MARSIM Maps	Size (m)	Large-Scale	Multi-Layer	Cluttered Obstacles	Narrow Corridor	Thin Structures	Real-world Environments
Historical Building (Fig. 1 (c))	47×20×23		✓	✓		✓	✓
Large Office (Fig. 5 (a))	45×16×6	✓	✓	✓	✓	✓	✓
Indoor-2 (Fig. 5 (b))	27×40×7			✓	✓		✓
Common Forest (Fig. 5 (c))	48×27×19	✓		✓		✓	✓
Simple Parking Garage (Fig. 5 (d))	45×15×5	✓					✓
Simple Forest (Fig. 5 (e))	45×16×6	✓		✓		✓	✓
Indoor-1 (Fig. 5 (f))	17×13×9			✓	✓	✓	✓
Complex Parking Garage (Fig. 5 (g))	62×12×10	✓		✓			✓
Dense Forest (Fig. 5 (h))	30×72×19	✓		✓	✓	✓	✓
Indoor-3 (Fig. 5 (i))	21×48×4			✓		✓	✓

a Livox Avia sensor (detailed in [30]) to scan the environment. The non-repetitive scanning of Livox Avia LiDAR enables the map points to be accumulated to a high resolution even when the LiDAR is placed at a stationary position. This reduces the effect of point density variation caused by LiDAR motion. To register all LiDAR scans under the same global frame, we use FAST-LIO2 [23] to construct a rough map and then utilize [24] to globally refine the map quality by LiDAR bundle adjustment. The globally registered point cloud map is imported to the CloudCompare software, where statistical outlier removal (SOR) filter and spatial downsampling are used to generate a uniform clean point cloud map. SOR filter is a filter that computes each point's average distance to its neighbors and removes the points having a relatively large distance (i.e., isolated noisy points). After filtering the outlier points, we manually fill some points in corners that the LiDAR scan cannot reach in real world. Using the above methods, we scanned ten real-world environments and obtained high-precision point cloud maps for the use of the simulator.

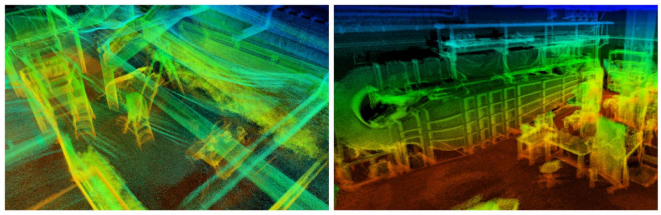
VI. RESULTS

A. High-Resolution Realistic Point Cloud Maps

This paper provides high-resolution (0.01 m) point cloud maps of ten real scenes for users to simulate, as shown in Fig. 5. Some of their environmental features can be seen in Table I, which can be used for reference when choosing a simulation map. The 0.01-m resolution map here refers to the original point cloud processed by the 0.01-m spatial downsampling. The scenes of the ten maps are three forests, three indoor scenes, a historical building (the HKU main building), two parking garages, and a large office. Detailed structures in real environments can be clearly seen in the close-up point clouds shown in Fig. 6.

B. Breakdown of Computation Resources Consumption

We compare the time consumption between MARSIM simulator and the Gazebo simulator. Since Gazebo can only use



(a) (b)

Fig. 6. Detailed structure of Indoor-1 and Indoor-2 maps. (a) Shows several clear ladders in Indoor-1 map and (b) shows many complex equipment in a machine workshop.

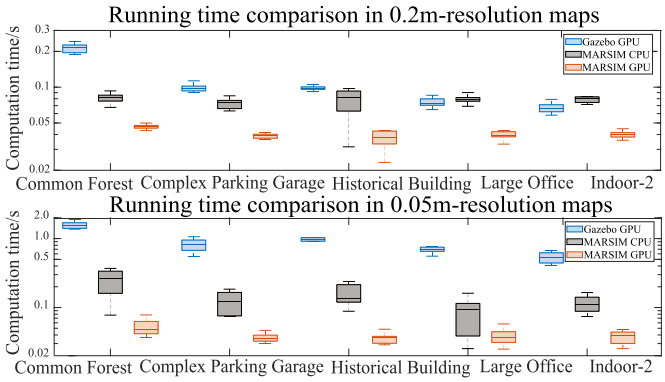


Fig. 7. Time consumption for rendering one Livox AVIA scan on a light-weight computation platform (NUC).

mesh models, we transform the point cloud maps of respective resolutions (see below) to mesh models using Poisson reconstruction method [8]. We select five typical scenes and compare the time and memory consumption of rendering one scan of a Livox AVIA LiDAR (77° × 70° FoV, 385 × 350 resolution, 30-m sensing range), respectively. For each map, we test two cases: a high-resolution map (0.05-m resolution) and a low-resolution map (0.2-m resolution). The data is generated by randomly selecting 10 positions and yaw angles of the UAV. The running time comparison on a light-weight computing platform NUC 10 Kit (with an i7-10710 U max frequency 4.70-GHz CPU, 32-GB RAM) is shown in Fig. 7. It can be seen that in low-resolution maps, even the CPU version of MARSIM can achieve slightly less computation time than the GPU-accelerated Gazebo simulation. With GPU acceleration, MARSIM is two times faster than Gazebo. In high-resolution maps, the difference is even more obvious: the CPU version of MARSIM is two times faster than the GPU-accelerated Gazebo simulation while the GPU version of MARSIM is ten times faster. The reason why Gazebo performed poorly in the experiments is because of the large number (over 2 million) of triangular faces in the generated mesh maps, which is necessary to retain a level of detail similar to the corresponding point cloud. In contrast, most existing robot simulations use very simple mesh maps, which are mainly composed of large planes and have a small number of triangular faces, which can be simulated in real time. Moreover, as a simulator specifically designed for point cloud, MARSIM does not need to process the whole render pipeline to render meshes (e.g., reducing the process of fragment shader, ray tracing, etc.) and complex physics simulation (like collision simulation), which decrease the consumption of computation

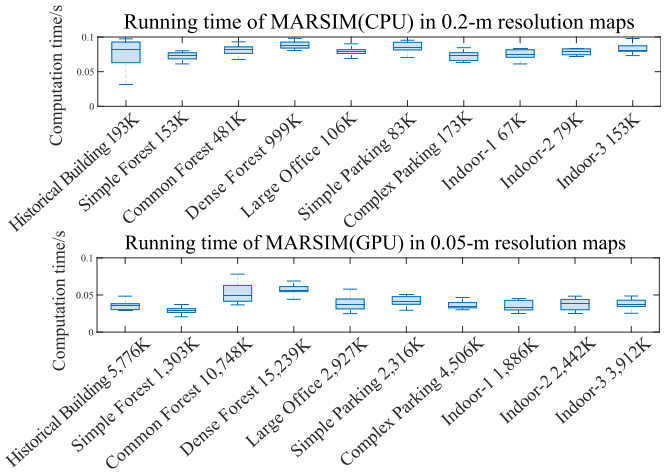


Fig. 8. Time consumption for rendering one Livox AVIA scan on all 10 realistic point cloud maps. The X-axis labels the name and size of the point cloud map.

TABLE II
MEMORY CONSUMPTION COMPARISON WITH GAZEBO ON A LIGHT-WEIGHT COMPUTATION PLATFORM (NUC)

Resolution	Map	RAM Consumption (GB)		
		Gazebo	MARSIM	
			CPU	GPU
0.2 m	Historical building	1.64	1.01	1.31
	Complex Parking Garage	1.46	1.1	1.25
	Large Office	1.68	1.04	1.13
	Common Forest	1.48	1.37	1.73
	Indoor-2	2.09	0.94	1.14
0.05 m	Historical building	6.97	4.04	3.43
	Complex Parking Garage	6.72	3.51	3.06
	Large Office	4.93	2.87	2.17
	Common Forest	16.88	7.35	3.53
	Indoor-2	3.73	2.51	1.84

resources significantly. Finally, we perform time consumption for all ten maps in 0.05-m and 0.2-m resolution with the same sensor. As shown in Fig. 8, in all cases, the simulator is able to run in real time at 10 Hz.

In addition to the time consumption comparison, we also collected the RAM consumption as shown in Table II. The RAM consumption of our simulator is about half that of the Gazebo simulator in both the CPU and GPU versions, which also demonstrates the light-weight characteristics of our simulator.

C. Accuracy Evaluation

We further conduct an experiment to assess the accuracy of the simulated point cloud in comparison to a real LiDAR-scanned point cloud. To collect the data, we hand-hold a Livox-AVIA LiDAR and walk through an environment with ample large planes. We then generate a set of simulation point clouds using the same trajectory in the simulator and the reconstructed map. The accuracy of the simulated point cloud is evaluated by comparing the reconstructed map from the real LiDAR with the accumulated map from the simulated point clouds. The accuracy indicator, defined as the average minimum distance between the two point cloud maps, is used to quantify the comparison. As

TABLE III
THE ACCURACY INDICATOR OF DIFFERENT INTERPOLATION METHOD AND DIFFERENT RESOLUTIONS

Method	Resolution (m)	Accuracy (m)
Only occlusion culling	0.1	0.0523
With plane correction	0.1	0.0323
With plane correction	0.01	0.0136

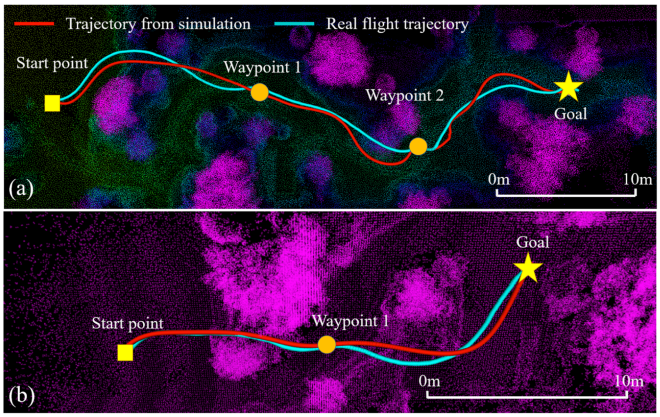


Fig. 9. Comparisons between flight trajectories in simulation and actual environment. (a) and (b) represent two experiments in different environments. The red lines are the trajectory results from the simulation, the blue lines are the actual flight trajectories.

shown in the results reported in Table III, the accuracy of the simulated point cloud is at the same level as the resolution of the simulated map. Furthermore, the plane correction algorithm significantly improves the accuracy.

D. Experiment Verification

To demonstrate that our simulator can provide flight simulation similar to real experiments, we verify the simulation of a UAV planning method, the Bubble planner [4], in an actual environment. To do so, we build a quadrotor UAV equipped with a Livox Mid360 LiDAR as used in [4]. Then, we handheld the UAV to manually scan the actual environment, which is a forest scene, and build its point cloud map. We simulate the Bubble planner in our simulator using the collected point cloud map and compare the simulated UAV flight trajectory with that of the real experiment with the same start position and target position. The comparison is shown in Fig. 9. As can be seen, the trajectories from the simulation are very close to the actual ones, which verifies the practicability of this simulator.

E. Support of Different Types of LiDARs and Other Functions

In order to increase the simulator's versatility, a variety of common LiDAR and depth camera models are also provided in the simulator. As shown in Fig. 10, the simulator supports sensors such as Livox Avia, Livox Mid-360, VLP-32, VLP-64, OS1-32, and Intel realsense D455. The simulator can reproduce the scanning patterns of these sensors so that users can use them directly without tuning any parameters. Moreover, dynamic obstacles and multi-UAV simulations are also supported, as shown in Figs. 11 and 12.

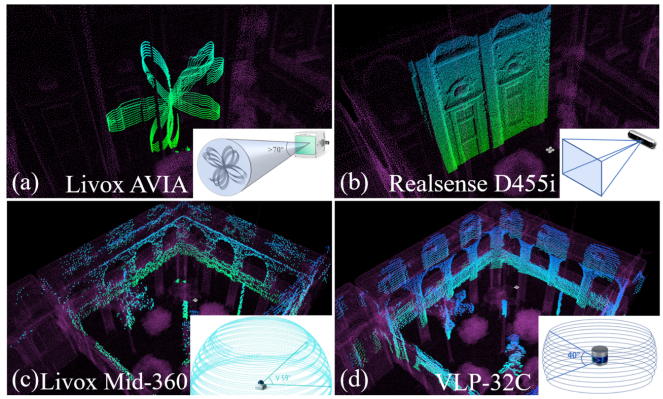


Fig. 10. Various LiDAR scan pattern support including Livox Avia (a), D455 (b), Livox Mid-360 (c), and VLP 32 (d), respectively.

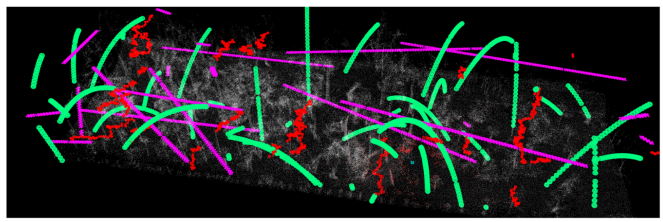


Fig. 11. Dynamic obstacles simulation in the simple forest map. The purple lines represent UAV models moving in constant speeds, the green curves represent spherical obstacles moving in free falling trajectories, and the red points represent cubes moving in random walk.

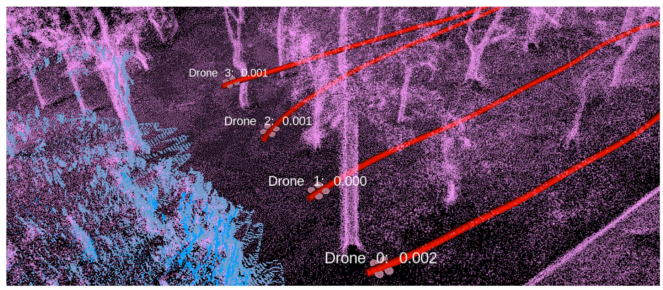


Fig. 12. Multi-UAV planning simulation. The pink models are the UAVs, and the red curves are the trajectories of the UAVs, avoiding the obstacles of a realistic forest map.

F. Practical Applications of the Simulator

This simulator is mainly used to provide a testing and verification platform for the algorithm development of LiDAR-based UAVs, especially motion planning and autonomous exploration algorithms that require interaction with the environments. While previous experiments have shown the application of our simulator in UAV motion planning, we also carried out simulation experiments of autonomous UAV exploration. Fig. 13 shows the autonomous exploration process of a UAV carrying Livox Avia using FUEL [25] algorithm in the Indoor-2 map. It is worth mentioning that the simulator has been successfully used to assist the development of multi-UAV mutual location in [31] and motion planning algorithm in [4], [32].

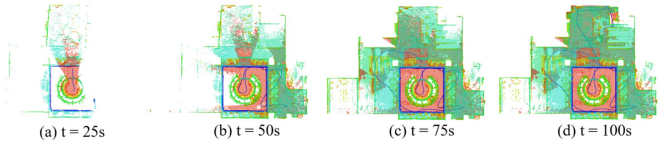


Fig. 13. Demonstration of a UAV autonomous exploration simulation in Indoor-2 map, utilizing FUEL algorithm. The area scanned by the UAV after different executing times are shown.

VII. Conclusion and Discussion

This paper proposes a LiDAR-based UAV simulator for real environment simulation on light-weight computing platforms. The simulator renders LiDAR scans directly on point cloud maps, which is way easier to capture for real environments than mesh models used by existing simulators. Moreover, due to the high accuracy of modern 3D LiDARs and laser scanners, a point cloud map scanned from real environments can truthfully represent the environment, which dramatically bridges the gap between simulation and reality. To maximize the practicality of the simulator, we further provide ten high-resolution point cloud maps and support the simulation of various types of LiDAR sensors, dynamic obstacles, and multi-UAV simulation. These features can meet the research and development needs of motion planning algorithms and autonomous exploration algorithms of single or multiple UAVs.

Since the simulator is based on point cloud maps, when the accuracy of the map is not high enough or there are noise points, the simulator cannot restore the correct details of the real environments. Also, the advantage in the computation efficiency of the simulator may degrade if a LiDAR scan is sparse, where the time for ray-casting on mesh models used by Gazebo (and other existing simulators) is reduced significantly. In contrast, the rendering module of our simulator has to generate a dense depth image not just on the scanning pattern, which leads to a waste of computing resources to calculate unnecessary depth image pixels. It could be an improvement direction of the simulator in the future.

References

[1] T. Dang, M. Tranzatto, S. Khattak, F. Mascarich, K. Alexis, and M. Hutter, "Graph–based subterranean exploration path planning using aerial and legged robots," *J. Field Robot.*, vol. 37, no. 8, pp. 1363–1388, Dec. 2020.

[2] K. Shah, G. Ballard, A. Schmidt, and M. Schwager, "Multidrone aerial surveys of penguin colonies in antarctica," *Sci. Robot.*, vol. 5, no. 47, 2020, Art. no. eabc3000.

[3] E. Karachaliou, E. Georgiou, D. Psaltis, and E. Stylianidis, "UAV for mapping historic buildings: From 3D modelling to BIM," *Int. Arch. Photogrammetry, Remote Sens. Spatial Inf. Sci.*, vol. 42, pp. 397–402, 2019.

[4] Y. Ren et al., "Bubble planner: Planning high-speed smooth quadrotor trajectories using receding corridors," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 6332–6339.

[5] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2004, pp. 2149–2154.

[6] O. Michel, "Cyberbotics Ltd. Webots: Professional mobile robot simulation," *Int. J. Adv. Robotic Syst.*, vol. 1, no. 1, 2004, Art. no. 5.

[7] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field Serv. Robot.*, 2018, pp. 621–635.

[8] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proc. 4th Eurograph. Symp. Geometry Process.*, 2006, pp. 61–70.

[9] F. Şenkul and E. Altuğ, "Modeling and control of a novel tilt–roll rotor quadrotor UAV," in *Proc. Int. Conf. Unmanned Aircr. Syst.*, 2013, pp. 1071–1076.

[10] L. Ximin, H. Gu, J. Zhou, Z. Li, S. Shen, and F. Zhang, "Simulation and flight experiments of a quadrotor tail-sitter vertical take-off and landing unmanned aerial vehicle with wide flight envelope," *Int. J. Micro Air Veh.*, vol. 10, pp. 303–317, 2018.

[11] W. Luo, Q. Tang, C. Fu, and P. Eberhard, "Deep-Sarsa based multi-UAV path planning and obstacle avoidance in a dynamic environment," in *Proc. 9th Int. Conf. Swarm Intell.*, 2018, pp. 102–111.

[12] Z. Han, Z. Wang, N. Pan, Y. Lin, C. Xu, and F. Gao, "Fast-racing: An open-source strong baseline for SE(3) planning in autonomous drone racing," *IEEE Robot. Automat. Lett.*, vol. 6, no. 4, pp. 8631–8638, Oct. 2021.

[13] G. Rong et al., "LGSVL simulator: A high fidelity simulator for autonomous driving, in *Proc. IEEE 23rd Int. Conf. Intell. Transp. Syst.*, 2020, pp. 1–6.

[14] M. Dharmadhikari et al., "Motion primitives-based path planning for fast and agile exploration using aerial robots," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 179–185.

[15] C. Cao, H. Zhu, H. Choset, and J. Zhang, "TARE: A hierarchical framework for efficiently exploring complex 3D environments," *Robot.: Sci. Syst.*, vol. 5, 2021.

[16] A. Brunel, A. Bourki, C. Demonceaux, and O. Strauss, "SplatPlanner: Efficient autonomous exploration via permutohedral frontier filtering," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 608–615.

[17] N. Koenig, "Darpa subt virtual competition software," 2019. [Online]. Available: https://github.com/osrf/subt/tree/master/subt_ign/worlds

[18] S. Manivasagam et al., "LiDARsim: Realistic lidar simulation by leveraging the real world," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 11167–11176.

[19] H. Pfister, M. Zwicker, J. Baar, and M. Gross, "Surfels: Surface elements as rendering primitives," *Proc. ACM SIGGRAPH Conf. Comput. Graph.*, 2000, pp. 335–342.

[20] A. Millane, Z. Taylor, H. Oleynikova, J. Nieto, R. Siegwart, and C. Cadena, "C-blox: A scalable and consistent TSDF-based dense mapping approach," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 995–1002.

[21] J. Lin and F. Zhang, "R3live: A robust, real-time, RGB-colored, lidar-inertial-visual tightly-coupled state estimation and mapping package," in *Proc. Int. Conf. Robot. Automat.*, 2022, pp. 10672–10678.

[22] W. Guerra, E. Tal, V. Murali, G. Ryou, and S. Karaman, "Flightgoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 6941–6948.

[23] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "FAST-LIO2: Fast direct LiDAR-inertial odometry," *IEEE Trans. Robot.*, vol. 38, no. 4, pp. 2053–2073, Aug. 2022.

[24] X. Liu, Z. Liu, F. Kong, and F. Zhang, "Large-scale LiDAR consistent mapping using hierarchical LiDAR bundle adjustment," *IEEE Robot. Automat. Lett.*, vol. 8, no. 3, pp. 1523–1530, Mar. 2023.

[25] B. Zhou, Y. Zhang, X. Chen, and S. Shen, "FUEL: Fast UAV exploration using incremental frontier structure and hierarchical planning," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 779–786, Apr. 2021.

[26] D. Shreiner et al., *OpenGL Programming Guide: The Official Guide to Learning OpenGL Versions 3.0 and 3.1*. London, U.K.: Pearson Education, 2009.

[27] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," in *Proc. IEEE 49th Conf. Decis. Control*, 2010, pp. 5420–5425.

[28] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *IEEE Robot. Automat. Mag.*, vol. 19, no. 3, pp. 20–32, Sep. 2012.

[29] K. Ogata et al., *Modern Control Engineering*, vol. 5. Englewood Cliffs, NJ, USA: Prentice Hall, 2010.

[30] C. Zheng, Q. Zhu, W. Xu, X. Liu, Q. Guo, and F. Zhang, "Fast-livo: Fast and tightly-coupled sparse-direct lidar-inertial-visual odometry," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 4003–4009.

[31] F. Zhu et al., "Decentralized lidar-inertial swarm odometry," 2022, *arXiv:2209.06628*.

[32] Y. Ren, S. Liang, F. Zhu, G. Lu, and F. Zhang, "Online whole-body motion planning for quadrotor using multi-resolution search," 2022, *arXiv:2209.06761*.