BIM-Aided Scanning Path Planning for Autonomous Surveillance UAVs with LiDAR

Changhao Song [a], Kai Wang [a] and Jack C.P. Cheng [a]

[a]Department of Civil and Environmental Engineering, The Hong Kong University of Science and Technology, Hong Kong SAR
E-mail: csongae@connect.ust.hk, kwangaw@connect.ust.hk, cejcheng@ust.hk

Abstract –

An Unmanned Aerial Vehicle (UAV), equipped with a Light Detection And Ranging (LiDAR) scanner, can collect high-accuracy point cloud data of facilities in cluttered indoor environment. Recent developments in aerial robotics have demonstrated navigation through designated waypoints, yet little has been investigated on the trajectory to complete a full scan of the environment. This study develops an automated approach to integrate scan planning and trajectory generation of a LiDAR-carrying UAV. The proposed approach converts an as-designed Building Information Model (BIM) into an occupancy map, where a set of waypoints are generated with a greedy algorithm. The shortest collision-free path to traverse all the waypoints is computed with the A* algorithm and Genetic Algorithm (GA). After that, the straight-line segments are transformed into a minimum snap trajectory formed of piecewise polynomials. The planned trajectory is validated with both a MATLAB numerical solver and a Hardware-In-the-Loop (HIL) simulation in the Unreal Engine 4.

Keywords –

UAV; LiDAR; BIM; Hardware-in-the-loop; Motion planning

1 Introduction

Terrestrial Laser Scanning (TLS) is commonly used in the Architecture, Engineering, Construction and Facility Management (AEC/FM) industry for site inspection, progress tracking and model generation. Traditional ways of TLS involve selection of scanner locations and registration of multiple point clouds. This process is usually conducted manually by surveyors, which is time-consuming and subject to coverage issues [1]. A wise approach is to integrate a Light Detection And Ranging (LiDAR) scanner with a ground or aerial robot, making it a versatile and efficient tool for Mobile Laser Scanning (MLS). An Unmanned Aerial Vehicle (UAV), owing to its autonomy and flexibility, is a good choice for the platform, especially in cluttered environments where the walkability is poor. In recent years, research in LiDAR-carrying UAVs has demonstrated robustness in Simultaneous Localization And Mapping (SLAM), as well as autonomous navigation in unknown environments [2]. However, most existing methods focus on navigation through a sequence of designated waypoints, while it is difficult to achieve an autonomous flight. The missing segment is planning for the essential waypoints to explore the environment and complete a full scan. This problem can be effectively addressed when prior knowledge of the environment is available, such as the design information of buildings and facilities.

This study is proposed to close the loop of scan planning and UAV path finding, with the aim of facilitating a fully autonomous flight for LiDAR-carrying UAVs. It starts from an as-designed Building Information Model (BIM), which retains the geometric and semantic information of a facility and is compatible with various data formats [3]. The sensor model is constructed based on an existing product [4] to represent the perception range, Field of View (FOV), and Level of Detail (LOD). A greedy algorithm is designed to iteratively maximize the coverage of the planned waypoints, followed by a Traveling Salesman Problem (TSP) to solve for an optimal path that consists of straight-line segments. This serves as a guiding path, which is transformed into a minimum snap trajectory with Quadratic Programming (QP).

The planned trajectory is validated first with a MATLAB numerical solver, before conducting a Hardware-In-the-Loop (HIL) simulation with the Robot Operation System (ROS) [5] and Unreal Engine 4 (UE4) [6]. The HIL simulation builds on our previous work [7] with an extension of automated control input. The intended scene is a cluttered indoor environment filled with Mechanical, Electrical and Plumbing (MEP) components.

This paper is organized as follows: section 2 provides a comprehensive review of the related studies

on LiDAR-carrying UAVs. Section 3 illustrates the proposed methodology on scan planning and trajectory generation, followed by the simulation environments and results in section 4. In closing, the conclusion and future work will be presented in section 5.

2 Literature Review

2.1 Scan Planning

The purpose of scan planning is to ensure coverage of the target objects while minimizing the cost of time or energy. Compared with traditional TLS, a mobile LiDAR scanner automatically enforces overlapping between consecutive scans for registration. However, similar techniques can be applied for visibility analysis and occlusion handling. For example, Argüelles-Fraga et al. [8] developed a method to parametrize the influencing factors of scan accuracy, based on a circular cross-section tunnel. Biswas et al. [1] proposed a BIM-oriented approach to determine optimal scanner locations that maximize the covered surfaces while considering occlusions between components. Wang et al. [9] presented a greedy algorithm to iteratively generate scanner locations around concrete specimens. While the above-mentioned studies aim at planning for fixed scanner locations, an integrated framework was proposed in [10] to generate waypoints of LiDAR-carrying UAVs and connect them with the shortest path. This study provides a clear outline to plan for scanning paths, yet little has been discussed on trajectory generation and flight control for detailed implementation.

2.2 Localization and Mapping

Localization is a critical issue for UAVs in an indoor environment, where GPS signals are not available. It is often combined with mapping to form a SLAM problem. The objective of localization is to obtain the 6-DoF (Degrees of Freedom) state estimation, including 3 positions (x, y, z) and 3 orientations (yaw, pitch, roll). This can be achieved with internal sensor suites, including Inertial Measurement Units (IMUs), LiDAR scanners, monocular and stereo cameras. IMUs are easily accessible, lightweight sensors that measure acceleration and angular velocity at high frequency. They are usually fused with other sensory data to produce odometry. LiDAR-based solutions, such as LOAM [11], outperform vision-based methods in terms of cumulative drifts. However, the demand for payload limits their application in Micro Aerial Vehicles (MAVs). There are also methods that take advantage of both types, such as V-LOAM [12], which is ranking the top on the KITTI odometry benchmark [13].

SLAM problems can be solved with filter-based algorithms, such as the commonly used Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF) [14] that deals with highly nonlinear models and particle filter [15] that can handle non-Gaussian distributions. In recent years, it is a growing trend to switch from filters to graph optimization, integrated with loop closure to reduce cumulative drifts. Such examples include ORB-SLAM [16] and VINS [17].

2.3 Motion Planning

In UAV motion planning, the term “path” and “trajectory” are often used interchangeably. According to [18], a path can be a continuous curve or discrete line segments connecting two positions, while a trajectory refers to a path parametrized with time *t*. In this paper, we use “path” to denote straight-line segments connecting waypoints, and “trajectory” for high-degree polynomials to be followed by a UAV.

The objective of UAV path planning is to determine the shortest collision-free path that connects the take-off position, a sequence of waypoints, and the landing position. Path planning problems can be solved efficiently with sampling-based methods, which generate random samples and connect to a search graph. Examples include Probabilistic Road Map (PRM), Rapid-exploring Random Tree (RRT), and RRT* [19] which converges to optimality as samples increase. Search-based methods, such as Dijkstra’s algorithm and A* algorithm, are also frequently used to find the optimal path from a search graph. A* is an extension of Dijkstra’s algorithm that evaluates each search node with a heuristic function before accessing it. As shown in [20], search algorithms can be combined with sampling methods to enable real-time processing.

To transform a path into a trajectory, the simplest approach is to solve for the polynomial parameters with respect to *t*, by setting waypoint constraints. However, a UAV trajectory is supposed to be safe and feasible, at least twice differentiable to produce velocity and acceleration. Mellinger and Kumar [21] formulated the trajectory as an optimization problem to minimize energy consumption and solved with Quadratic Programming (QP). Richter et al. [19] extended their work to an unconstrained QP and ensured safety by adding intermediate waypoints. A different technique was proposed by Chen et al. [22] that generates flight corridors formed by safe regions, and constrain the trajectory within it.

3 Methodology

This section describes the proposed methods and implementation. An overview of the framework is illustrated in Figure 1. The planning phase consists of four steps: (1) map construction, (2) waypoint

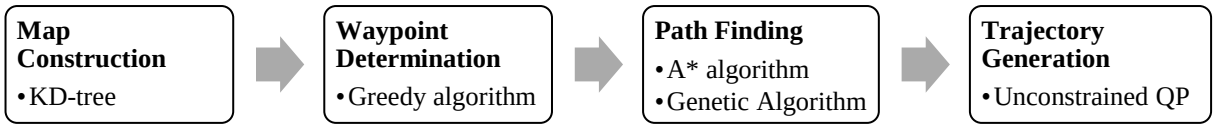

Figure 1. Workflow of the planning phase

determination, (3) path finding and (4) trajectory generation, which are detailed in 3.1-3.4, respectively. The target facility is a water treatment plant located in Tai Po, Hong Kong SAR. It is a cluttered indoor area filled with MEP components, such as pipelines and valves. One set of the pipelines, which is part of the duplicated layouts, is extracted as the test area. The planning phase is implemented in MATLAB, followed by the simulations in section 4.

3.1 Map Construction

The UAV motion planning is performed on an occupancy grid map, where each cell is attached with a label, indicating obstacles or free space. The map is constructed based on an as-designed BIM of the target object, as shown in Figure 2. The model is exported as an OBJ file with an add-in [23] of Revit, and meshed in CloudCompare [24] to produce a reference point cloud, displayed in Figure 3. The reference point cloud consists of evenly distributed points, covering the surface of the objects. It is represented with the KD-tree data structure, which enables fast K-Nearest Neighbor (KNN) search for computing safe distance. An empty voxel grid is created with resolution s, and a KNN search is applied between the grid points and the reference point cloud. Depending on the nearest neighbor distance d, and the safe threshold θ_d, each cell in the voxel grid is labeled as obstacle ($d < 0.5s$), safe region ($d > \theta_d$), or buffer zone. Considering the scale of the facility, s and θ_d are determined as 0.5m and 1m, respectively. This completes the construction of the occupancy grid map.

Figure 2. The as-designed BIM of the target components in Revit

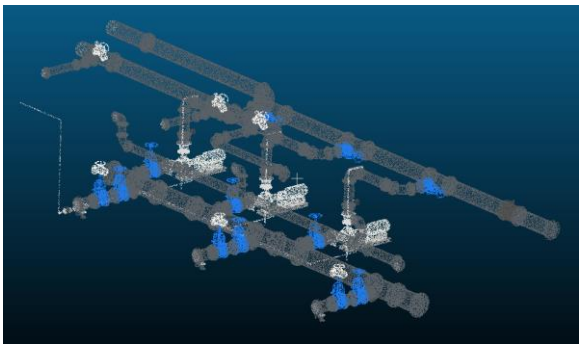

Figure 3. The meshed point cloud in CloudCompare. Surface density = 1000. RGB entries are obtained from the texture.

3.2 Waypoint Determination

The objective of waypoint determination is to bridge the problem of traditional TLS with that of LiDAR-carrying UAVs. A greedy algorithm was proposed in [9] that maximizes the number of covered surfaces at each selection. However, this method is not model-based and works only for concrete specimens. In this study, we adopt a similar idea that tries to achieve the local optimum when generating each waypoint and iterate until the requirements are satisfied. Besides, we describe the coverage based on the LiDAR model and consider occlusion handling.

3.2.1 LiDAR Model

The sensor model in this paper is constructed based on an existing product, RS-LiDAR-32 [4], of which the specifications are listed in Table 1. Geometrically, the sensor coverage can be described as the volume between two conic surfaces, bounded by the sphere of perception range (Figure 4). Given the LOD requirement δ and angular resolution θ, the perception range L should be reduced according to Equation (1):

$$L = \delta/\theta \quad (1)$$

In this case, the smallest element has a diameter of 5cm. Considering the typical measurement error of ±2cm and the worst-case resolution 0.4°, the perception range is determined as 4m to ensure detection of the pipes. As for the FOV, UAVs will not stay horizontal during the flight because the pitch, roll angles change according to the x-y motion. Therefore, the vertical FOV can be reduced to account for the inclination. In this study, the

reduction is taken as 10%.

Table 1. RS-LiDAR-32 specifications

Horizontal FOV	360°
Vertical FOV	–25° ~ +15°
Horizontal Resolution	0.1°/ 0.2°/ 0.4°
Vertical Resolution	≥ 0.33°
Range	200m
Range accuracy (typical)	±2cm

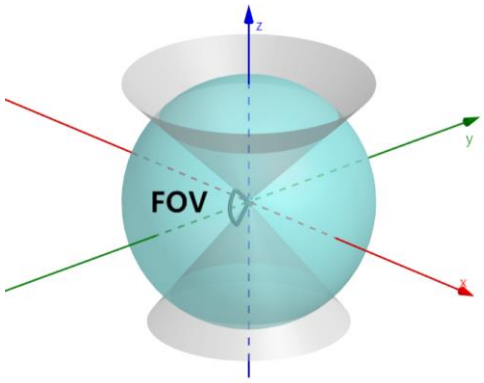

Figure 4. Coverage of the LiDAR model. The donut shape bounded by two conic surfaces (FOV) and a sphere (perception range)

3.2.2 Greedy Algorithm

The greedy algorithm is designed to generate a set of waypoints by maximizing the number of newly covered cells at each iteration, until the stopping criteria are met. The algorithm takes all the occupied cells (i.e. obstacles) as the target to be covered, and the safe regions as potential waypoints. A greedy search is applied to determine the safe cell with maximum coverage and add to the waypoint list. Detailed procedures are illustrated in Algorithm 1. The coverage examination is explained as follows.

For each safe cell, the range search is applied to return all the target cells within the sphere of perception range. This is also achieved with the KD-tree structure. After that, the sphere is reduced according to the FOV with Equation (2), where γ is the angle between the vector **u** from the safe cell to the target cell, and the body frame z-axis **z**. This represents the volume sandwiched between two conic surfaces from the vertical FOV, as illustrated in Figure 4.

$$\gamma = \cos^{-1}\left(\frac{\boldsymbol{u}\cdot\boldsymbol{z}}{\|\boldsymbol{u}\|\|\boldsymbol{z}\|}\right) \in \frac{\pi}{2} - FOV \tag{2}$$

The occlusion handling is realized with similar techniques. The vector **l**, which connects the safe cell with an obstacle cell, is checked against **u**. When Equation (3) and (4) are both satisfied, the ray to the target cell is considered as occluded by the obstacle. These equations indicate the cylindrical volume centered around **u**, with radius 1/2 of the grid size. An illustration is available in Figure 5. The occlusion check is applied for the obstacle cells within the spherical range. When all the obstacles return false, the target cell is considered as within the coverage.

$$h = \frac{\|\boldsymbol{l}\times\boldsymbol{u}\|}{\|\boldsymbol{u}\|} < \frac{1}{2}s \tag{3}$$

$$p = \frac{\|\boldsymbol{l}\cdot\boldsymbol{u}\|}{\|\boldsymbol{u}\|} \in (0, \|\boldsymbol{u}\|) \tag{4}$$

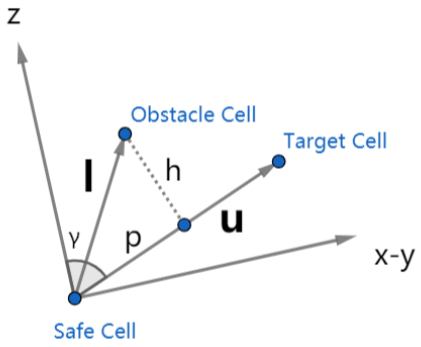

Figure 5. Illustration of Equation (2), (3) and (4). γ is the angle between **u** and **z**, h is the point-to-line distance and p is the projection of **l** on **u**

Algorithm 1 Greedy algorithm

```
i ← 0
waypoints ← empty()
uncovered_map ← obstacles
while i < max_iteration
   max_coverage ← empty()
   max_cell ← null
   for cell in safe_region
      sphere ← range_search(cell, uncovered_map, L)
      cone ← reduce_by_FOV(sphere)
      coverage ← reduce_by_occlusion(cone)
      if coverage > max_coverage
         max_coverage ← coverage
         max_cell ← cell
      end if
   end for
   uncovered_map ← uncovered_map \ max_coverage
   waypoints ← waypoints ∪ max_cell
   i ← i+1
   if max_coverage < quit_threshold
      break
   end if
end while
return waypoints
```

3.3 Path Finding

After a set of waypoints are determined, the problem is to compute an optimal path to traverse all of them. According to [10], this can be formulated as a TSP and solved efficiently with Genetic Algorithm (GA). We adopt similar techniques to solve for a collision-free path that ensures the shortest summed Euclidean distance. However, this path contains straight-line segments and sharp turns that are unsuitable for flight control. Therefore, it serves as the waypoint constraints for trajectory generation.

To form the TSP, a cost matrix is required to represent the pairwise path cost between the nodes. In this problem, it is constructed with the A* algorithm, which is applied on every pair of the waypoints. The occupancy map is treated as a search graph, where the successors of a cell are generated from the 26 adjacent cells. The heuristic function is taken as the Euclidean distance to the goal, which is guaranteed to be admissible.

After the cost matrix is obtained, the GA is implemented as follows:

1. The initial population is generated with random permutation of the waypoints, in bit arrays.
2. The tournament selection is applied to obtain a set of parents based on the fitness function, which is the summed path cost of the ordered sequence.
3. Three operations are applied to produce the next generation: (1) copy: select a member and copy directly to the next generation. (2) crossover: select two parents to produce offspring. Here, the order crossover operator is used, which takes a subset from parent 1, and arrange the remaining bits according to their order in parent 2. (3) mutation: select a member and randomly switch two bits in it.
4. Iterate from step 2, until N generations.

For this study, a population size of 1000 is applied on a set of 10 waypoints. The rate of copy, crossover and mutation are 9%, 90% and 1%, respectively. The optimal solution first appeared after 9 generations.

3.4 Trajectory Generation

The UAV trajectories are usually represented as piecewise polynomials parametrized with time *t*, in three dimensions, respectively. To ensure kinodynamic feasibility, the trajectory is subject to the derivative constraints which come from the specified end derivatives, and the continuity constraints which ensures smoothness. The expression in one dimension is shown in Equation (5) and (6), where the trajectory is an N-degree polynomial with M pieces:

$$f(t) = \begin{cases} f_1(t) = \sum_{i=1}^{N} p_{1,i}(t-T_0)^i & T_0 \le t \le T_1 \\ f_2(t) = \sum_{i=1}^{N} p_{2,i}(t-T_1)^i & T_1 \le t \le T_2 \\ \vdots & \\ f_M(t) = \sum_{i=1}^{N} p_{M,i}(t-T_{M-1})^i & T_{M-1} \le t \le T_M \end{cases} \quad (5)$$

$$\text{s.t.} \quad \begin{cases} f_j^{(k)}(T_j) = x_{T,j}^{(k)} \\ f_j^{(k)}(T_j) = f_{j+1}^{(k)}(T_j) \end{cases} \quad (6)$$

To solve for the polynomial parameters $p_{j,i}$, we refer to the method in [21], which formulates the trajectory as an optimization problem: the objective is to minimize the fourth order derivative (i.e. snap) of the trajectory, subject to the continuity constraints and derivative constraints. Equation (7) and (8) illustrate the problem definition in vector form, where **p** is the collection of polynomial parameters and **Q** is the Hessian matrix, $\mathbf{A_{eq}}$ and $\mathbf{d_{eq}}$ are the collection of constraints. The minimum snap trajectory is a seventh degree (N=7) piecewise polynomial, which optimizes the least energy consumption. The variable **p** can be solved with QP.

$$\begin{aligned} \text{min.} \quad & J = \int_o^T \left(f^{(4)}(x)\right)^2 dx \\ & = \begin{bmatrix} \boldsymbol{p_1} \\ \vdots \\ \boldsymbol{p_M} \end{bmatrix} \begin{bmatrix} \boldsymbol{Q_1}(T_1) & & \\ & \ddots & \\ & & \boldsymbol{Q_M}(T_M) \end{bmatrix} \begin{bmatrix} \boldsymbol{p_1} \\ \vdots \\ \boldsymbol{p_M} \end{bmatrix}^T \\ & = \boldsymbol{p^T Q p} \end{aligned} \quad (7)$$

$$\text{s.t.} \quad \boldsymbol{A_{eq}} \begin{bmatrix} \boldsymbol{p_1} \\ \vdots \\ \boldsymbol{p_M} \end{bmatrix} = \boldsymbol{d_{eq}} \quad (8)$$

However, direct optimization of polynomial parameters is numerically unstable, because the values are usually very small as time *t* increases. Therefore, we adopt the method in [19] to reformulate the problem as an unconstrained QP. A mapping matrix **M** is constructed to transform the variable from polynomial parameters **p** into the end derivatives **d**. Additionally, a binary selection matrix **C** containing 1s and 0s is constructed to separate the fixed derivatives $\mathbf{d_F}$ and free derivatives $\mathbf{d_P}$. The derivative constraints and continuity constraints are automatically enforced by the selection matrix. In this way, the problem is transformed into an unconstrained QP. The composition matrix in the middle can be further split according to the size of $\mathbf{d_F}$ and $\mathbf{d_P}$, as shown in Equation (9) and the close-form solution is obtained with Equation (10). The optimized end derivatives are transformed back to polynomial parameters to compute the trajectory.

$$J = \begin{bmatrix} \boldsymbol{d_1} \\ \vdots \\ \boldsymbol{d_M} \end{bmatrix} \boldsymbol{M^{-T} Q M^{-1}} \begin{bmatrix} \boldsymbol{d_1} \\ \vdots \\ \boldsymbol{d_M} \end{bmatrix}^T = \begin{bmatrix} \boldsymbol{d_F} \\ \boldsymbol{d_P} \end{bmatrix}^T \boldsymbol{C M^{-T} Q M^{-1} C^T} \begin{bmatrix} \boldsymbol{d_F} \\ \boldsymbol{d_P} \end{bmatrix} = \begin{bmatrix} \boldsymbol{d_F} \\ \boldsymbol{d_P} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{R_{FF}} & \boldsymbol{R_{FP}} \\ \boldsymbol{R_{PF}} & \boldsymbol{R_{PP}} \end{bmatrix} \begin{bmatrix} \boldsymbol{d_F} \\ \boldsymbol{d_P} \end{bmatrix} \quad (9)$$

$$\boldsymbol{d_P^*} = -\boldsymbol{R_{PP}^{-1} R_{FP}^T d_F} \quad (10)$$

To construct the Hessian matrix **Q** and mapping matrix **M**, the time duration T_i for each segment is required. In this study, the time is allocated according to the Euclidean path cost, based on the predefined average velocity at 1m/s. Another critical issue in trajectory generation is that the piecewise polynomial may deviate from the collision-free guiding path. To reinforce safety, collision check is performed along the trajectory with KNN search. The positions at each timestamp are checked against the nearest obstacle cell. The midpoint of guiding path segments will be added as an intermediate waypoint if collision is detected.

4 Validation

This section describes two separate experiments to validate the planned trajectory. The first one is performed in MATLAB with the numerical solver ode45. A simple PID controller is implemented to realize the motion control. The second experiment is an HIL simulation in the UE4 environment, where a physical flight controller is employed to communicate with the simulator. Details are explained in 4.1 and 4.2, respectively.

4.1 MATLAB simulation

The UAV trajectories are executed with a flight controller, which takes in the desired states and true states of the UAV at each moment to produce the desired motor output. The planned trajectory is published in a stream of state vectors $[x, y, z, v_x, v_y, v_z, \psi, \theta, \varphi, \omega_x, \omega_y, \omega_z]^T \in R^{12}$, with a fixed frequency. It was demonstrated in [21] that the full state vector can be reduced to the 3D position and yaw angle $[x, y, z, \psi]^T$, due to the differential flatness property. For this study, only the 3D position is enforced while the yaw planning is left as future work. A PID controller is constructed following the nested loops in [25], where the position control lies in the outer loop and influence the attitude control in the inner loop. The governing equation is shown in Equation (11), where **e**(t) is the error between the desired state and true state. The control output **u**(t) is

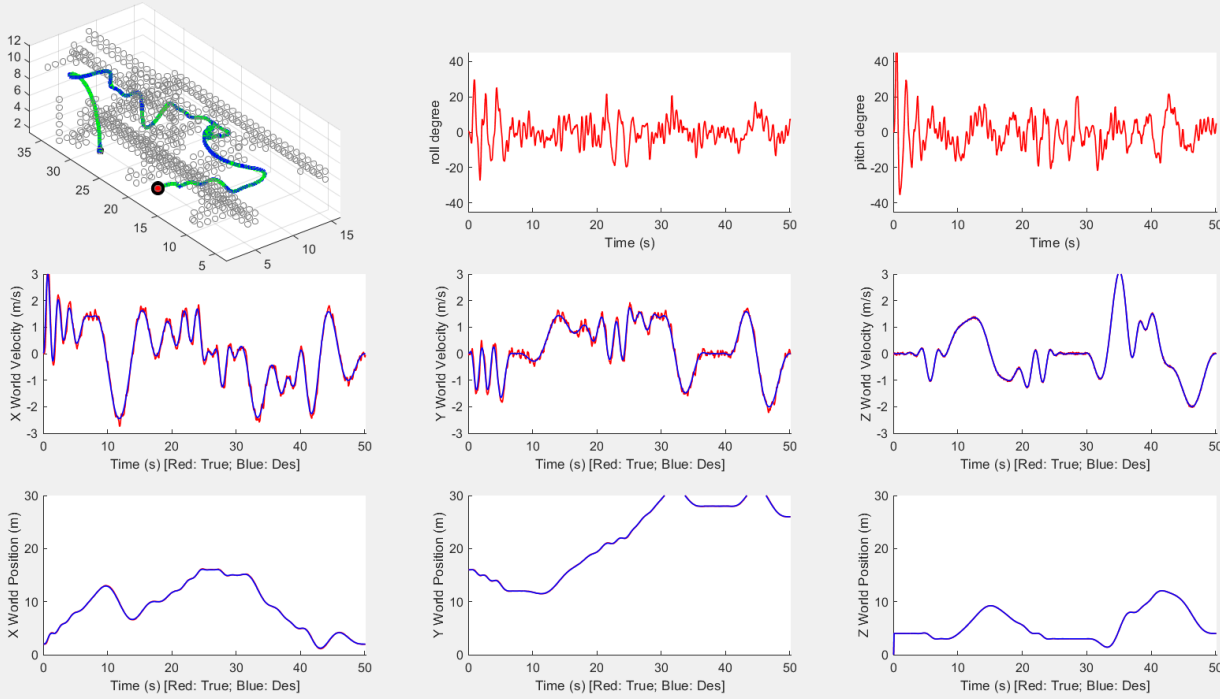

Figure 6. Simulation results in MATLAB. In subplot (1), the planned trajectory and the ground truth trajectory are plotted with green and blue curves, respectively. The grey cells indicate the obstacles. In (2)-(9), the desired states and the true states are plotted with blue and red curves, respectively. For the roll and pitch angle, the desired values are not commanded, but determined by the horizontal motion, instead.

related to the Force and Moment with the Newton-Euler equations, as shown in Equation (12) and (13). The weight *mg* and moment of inertia matrix **I** are determined according to the UAV model in [26]. The gain parameters K_p and K_d are tuned manually, while the Integral term is omitted for simplicity. During the simulation, the desired states are the input from the trajectory generator, while the true state is computed with the ode45 solver, based on rigid body dynamics.

$$\boldsymbol{u}(t) = \ddot{\boldsymbol{x}}^{\boldsymbol{des}}(t) + K_d\dot{\boldsymbol{e}}(t) + K_p\boldsymbol{e}(t) \tag{11}$$

$$m\ddot{\boldsymbol{p}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + \boldsymbol{R} \cdot \begin{bmatrix} 0 \\ 0 \\ \Sigma F_i \end{bmatrix} \tag{12}$$

$$\boldsymbol{I} \cdot \begin{bmatrix} \ddot{\Phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} + \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \times \boldsymbol{I} \cdot \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \Sigma M_x \\ \Sigma M_y \\ \Sigma M_z \end{bmatrix} \tag{13}$$

The simulation result is plotted in Figure 6, where subplot (1) is a 3D view of the trajectory and (2)-(9) are the UAV states against time.

4.2 HIL simulation

The UE4 is an advanced game engine that provides highly realistic virtual environment. Besides, it enables a wide range of robotics applications, such as navigation, computer vision, deep learning, etc. This is the main reason why it is selected as the platform for experiment. The HIL simulation is carried out based on our previous work [7], in which a software pipeline was developed to integrate UE4 with ROS. The former provides the physics engine and high-quality sensor data, while the latter contains abundant packages for robotics perception and odometry. The communication between them is realized with the ROS master.

In this simulation, the as-designed BIM of the target facility is exported in FBX format and then imported into UE4 to create a scene, as shown in Figure 7. The planned trajectory is coded into a script as a stream of messages with timestamp, position and orientation. The script is passed into the UE4 server through an API layer, AirSim [27]. The trajectory is executed with a hardware flight controller, Pixhawk 4 [28], which subscribes desired states and ground truth states from the UE4 environment and publish motor outputs to the UAV model. A screenshot during the simulation is displayed in Figure 8.

5 Conclusion and Future Work

This paper presents an integrated framework to bridge the robotics problem of UAV navigation with the civil engineering application of as-built point cloud generation. The framework comprises BIM-aided map construction, waypoint-based scan planning, static path planning, and dynamic trajectory generation. These techniques are verified with a numerical simulation and a highly realistic HIL simulation. The results demonstrated that the motion planning algorithms can deal with complex environments with MEP components.

The proposed framework can be improved in several aspects. First, the yaw planning for trajectory can be completed, which is critical for vision-based odometry and navigation. Secondly, the quality of scanning is not evaluated. This can be addressed by setting a LiDAR sensor model into the UE4 environment to perform a virtual scan. Furthermore, it is promising if the framework can run in parallel with ROS to achieve real-time application. This is possible with the MATLAB ROS Bridge and it is expected to conduct a real-world experiment in future.

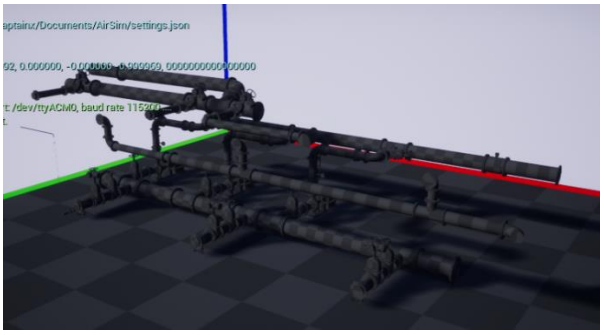

Figure 7. The scene constructed from the FBX file in UE4.

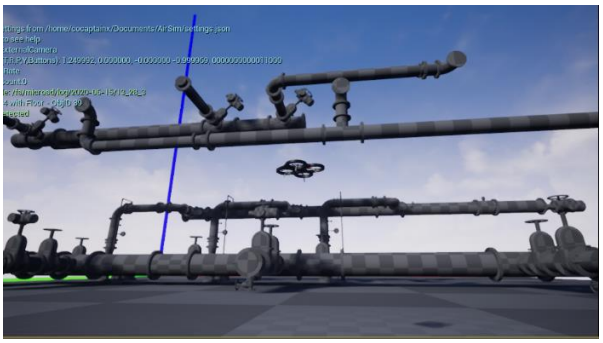

Figure 8. Trajectory following with the Pixhawk 4 flight controller.

References

[1] Biswas H. K., Bosché F., and Sun M. Planning for scanning using building information models: A novel approach with occlusion handling. In *Symposium on Automation and Robotics in Construction and Mining (ISARC 2015)*, 2015.

[2] Gao F. et al. Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments. *Journal of Field Robotics*, vol. 36, *(4)*, pp. 710-733, 2019.
[3] Shirowzhan S. et al. BIM compatibility and its differentiation with interoperability challenges as an innovation factor. *Autom. Constr.*, vol. 112, pp. 103086, 2020.
[4] RS-LiDAR-32. Online: https://www.robosense.ai/rslidar/rs-lidar-32. Accessed: 12/06/2020
[5] Quigley M. et al. ROS: An open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
[6] Unreal Engine 4. Online: https://www.unrealengine.com/en-US/?lang=en-US. Accessed: 12/06/2020
[7] Wang K. and Cheng J. C. Integrating hardware-in-the-loop simulation and BIM for planning UAV-based as-built MEP inspection with deep learning techniques. In *Proceedings of the 36th International Symposium on Automation and Robotics in Construction*, 2019.
[8] Argüelles-Fraga R. et al. Measurement planning for circular cross-section tunnels using terrestrial laser scanning. *Autom. Constr.*, vol. 31, pp. 1-9, 2013.
[9] Wang Q., Sohn H., and Cheng J. C. Automatic as-built BIM creation of precast concrete bridge deck panels using laser scan data. *J. Comput. Civ. Eng.*, vol. 32, *(3)*, pp. 04018011, 2018.
[10] Bolourian N. and Hammad A. LiDAR-equipped UAV path planning considering potential locations of defects for bridge inspection. *Autom. Constr.*, vol. 117, pp. 103250, 2020.
[11] Zhang J. and Singh S. LOAM: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, 2014.
[12] Zhang J. and Singh S. Visual-lidar odometry and mapping: Low-drift, robust, and fast. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015.
[13] Geiger A. et al. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, vol. 32, *(11)*, pp. 1231-1237, 2013.
[14] Jung J. et al. Development of kinematic 3D laser scanning system for indoor mapping and as-built BIM using constrained SLAM. *Sensors*, vol. 15, *(10)*, pp. 26430-26456, 2015.
[15] Özaslan T. et al. Inspection of penstocks and featureless tunnel-like environments using micro UAVs. In *Field and Service Robotics*, 2015.
[16] Mur-Artal R., Montiel J. M. M., and Tardos J. D. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, vol. 31, (5), pp. 1147-1163, 2015.
[17] Qin T., Li P., and Shen S. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, vol. 34, *(4)*, pp. 1004-1020, 2018.
[18] Yang L. et al. A literature review of UAV 3D path planning. In Proceeding of the 11th World Congress on Intelligent Control and Automation, 2014.
[19] Richter C., Bry A., and Roy N. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research* (pp. 649-666), Springer, Cham, 2016.
[20] Gao F. and Shen S. Online quadrotor trajectory generation and autonomous navigation on point clouds. Presented at the 2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), 2016, pp. 139–146.
[21] Mellinger D. and Kumar V. Minimum snap trajectory generation and control for quadrotors. Presented at the 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 2520–2525.
[22] Chen J., Liu T., and Shen S. Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. Presented at the 2016 IEEE International Conference on Robotics and Automation (ICRA), 2016, pp. 1476–1483.
[23] Revit OBJ converter. Online: https://visionworkplace.com/products/obj-converter-for-autodesk-revit. Accessed: 12/06/2020.
[24] CloudCompare. Online: https://www.danielgm.net/cc/. Accessed: 12/06/2020.
[25] Michael N. et al. The grasp multiple micro-uav testbed. *IEEE Robotics & Automation Magazine*, vol. 17, *(3)*, pp. 56-65, 2010.
[26] Mellinger D., Michael N., and Kumar V. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *The International Journal of Robotics Research*, vol. 31, *(5)*, pp. 664-674, 2012.
[27] Shah S. et al. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2018.
[28] Meier L. et al. PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, vol. 33, *(1-2)*, pp. 21-39, 2012.