International Journal of Multimedia Data Engineering and Management
Volume 14 • Issue 1

Adaptive Acquisition and Visualization of Point Cloud Using Airborne LIDAR and Game Engine

Chengxuan Huang, University of California, Davis, USA
Evan Brock, University of Tennessee at Chattanooga, USA
Dalei Wu, University of Tennessee at Chattanooga, USA*
https://orcid.org/0000-0002-9362-3906
Yu Liang, University of Tennessee at Chattanooga, USA
https://orcid.org/0000-0002-1844-9063

ABSTRACT

The development of digital twin for smart city applications requires real-time monitoring and mapping of urban environments. This work develops a framework of real-time urban mapping using an airborne light detection and ranging (LIDAR) agent and game engine. In order to improve the accuracy and efficiency of data acquisition and utilization, the framework is focused on the following aspects: (1) an optimal navigation strategy using Deep Q-Network (DQN) reinforcement learning, (2) multi-streamed game engines employed in visualizing data of urban environment and training the deep-learning-enabled data acquisition platform, (3) dynamic mesh used to formulate and analyze the captured point-cloud, and (4) a quantitative error analysis for points generated with our experimental aerial mapping platform, and an accuracy analysis of post-processing. Experimental results show that the proposed DQN-enabled navigation strategy, rendering algorithm, and post-processing could enable a game engine to efficiently generate a highly accurate digital twin of an urban environment.

KEYWORDS

Digital Twin, LIDAR, Point Cloud, Reinforcement Learning, Rendering, Surface Reconstruction

1. INTRODUCTION

The development of the digital twin of an urban environment requires live streaming of measurement data to maintain real-time accurate representation of the environment. It is preferable that the measurements are collected in a continuous and automated fashion. Machine learning enabled measurement instruments serve the purpose of urban data acquisition well as they need minimal labor work. For example, automated airborne LIDARs are desirable in real-time urban mapping as they provide continuous high-resolution ranging and depth information by illuminating the object/

DOI: 10.4018/IJMDEM.332881 *Corresponding Author

environment with laser light and measuring the reflection with a sensor component. However, it is usually costly to train such systems in a real-world urban setting.

The development of the digital twin also calls for real-time visualization of heterogeneous data, especially live streaming data (Saddik, 2018). Visualization plays a critical role in the development of digital twins. Game engines are often used to render large, detailed, 3D environments, the same kind that geospatial experts seek to replicate (Andrews, 2020). The coordinate system within any game engine can be used to replicate 3D localization of objects and terrain, while taking advantage of their optimization and portability. Both interactable and performant, game engines seem to be the perfect candidate to visualize and interact with the geographic environment, and thus are a near perfect candidate to visualize urban environment (Rusu, 2018). Industry clearly agrees on the aspect. For example, both Google and Mapbox have built APIs and SDKs to bring their infrastructure and frameworks into the Unity game engine (Google, 2021; Mapbox, 2021).

But game engines are simply not built to handle live streaming data from unsupported objects, nor are they built to render dynamically changing meshes defined by live streaming data. Most previous work with LIDARs seeks to localize an object through deducing their own location through LIDAR data (Chong, 2013a; Chong, 2013b). Other work uses a combination of telemetry sensors and LIDAR data to achieve the same purpose (Toroslu, 2018). While these approaches work well for object detection or short-term scans, they do not support collaborative scans where multiple scans can be stitched together automatically through the geographical significance of their vertices in any three-dimensional environment.

In this paper, we describe a framework connecting IoT devices to game engines with point cloud pre-processing and post-processing techniques for surface reconstruction in urban mapping. The framework allows for the reconstruction of an environment to be observed in real-time. Different from existing work on mapping using LIDAR data (Agrawal, 2017), our implementation of large-scale live maps inside a game engine has proven to be very intuitive in testing. We also propose to use a game engine to generate a virtual urban environment where an airborne LIDAR agent equipped with DQN reinforcement learning is trained. This offers a safe, efficient, and low-cost approach for the training of the LIDAR agent. When the trained LIDAR agent is deployed in real urban environments, the game engine is also able to visualize the collected LIDAR data in surface reconstruction.

The remainder of this paper is organized as follows: Section 2 presents the proposed framework and workflow. Section 3 addresses LIDAR data processing. Section 4 discusses game engine enabled virtual training of the LIDAR agent for optimal navigation. Section 5 discusses the LIDAR data error sources. Section 6 presents both simulation and field experimental results. Section 7 concludes this paper and lists the future work.

2. OVERVIEW OF THE PROPOSED FRAMEWORK AND WORKFLOW

This work aims to develop an urban data acquisition and visualization framework that is applicable for both simulation and in-situ operation of an airborne LIDAR for digital twin development. One objective is to achieve an autonomous drone carrying a LIDAR that can navigate and map an area without prior knowledge. As shown in Figure 1, we propose to adopt reinforcement learning to allow the drone to learn how to gather information efficiently without any collision. The DQN model is first trained in controlled virtual environments, and then deployed with a real airborne LIDAR in real-world environments.

The drone simulation in a virtual environment offers a simplified interface to control the drone and retrieve LIDAR data. We used AirSim (Microsoft, 2021), a simulator for various vehicles built on Unreal Engine. Unreal Engine's level editor also provides visualization and raycasting, as well as the simulation/virtual environment, as shown in Figure 2. We used the "Downtown West Modular Pack" created by PurePolygons as the environment, which is available in the Epic Games Marketplace.

Figure 1. The proposed urban data acquisition and visualization framework for digital twin development

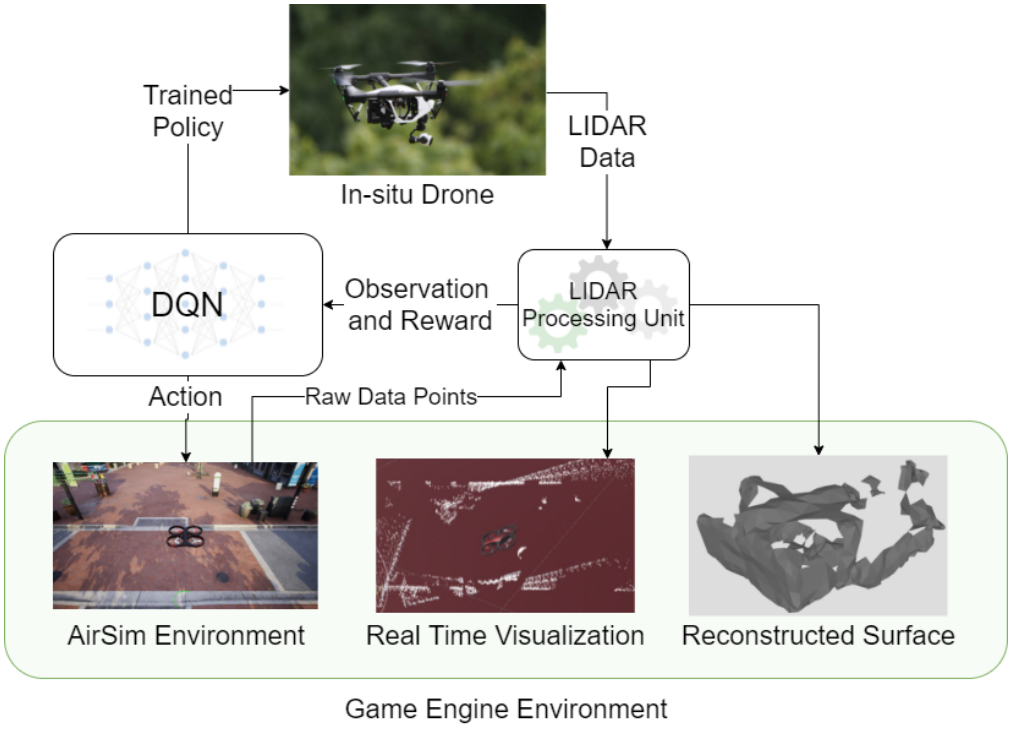


Figure 2. A virtual environment generated by AirSim

For each time step of the DQN, the virtual drone sends its received LIDAR data to a LIDAR processing unit that analyzes the observation and calculates the reward. The LIDAR processing unit is built on the Computational Geometry Algorithms Library (CGAL). For each incoming LIDAR

data point to be considered as new valid information, it needs to be at least somewhat distant from the rest of the already collected points. Furthermore, we discard the points that are very far from the origin of the virtual drone because we are mainly interested in the surroundings of the drone. To process and represent the observations from the drone, we propose two generic maps that describe the surroundings of the drone. One map describes the density of points and the other describes the distance of the closest point in each region. The observation also includes the position of the drone since the drone is rewarded for acquiring the points that are closer from the origin. The observation and reward of each time step are sent to the replay memory module of the DQN. The DQN decides the action of the virtual airborne LIDAR of the next time step based on a balanced exploitation and exploration strategy and sends the action back to the AirSim environment.

After the training is done in the virtual environment, we deploy the learned DQN algorithm with a real airborne LIDAR in a real environment. The collected on-site field data include the geographical data from the drone and the relative scan from the LIDAR carried by the drone. The drone records the offset of each scan, which is the vertical, horizontal and orientation components different from the user-defined geographical zero point. The LIDAR detects all surrounding objects within range and records their relative positions to the drone. The on-site data are processed every frame, which allows taking multiple scans that are aligned automatically if there is overlap between them. The data acquired by the LIDAR forms a point cloud and is sent to and processed within the LIDAR processing unit and rendered back in the game engine for real time visualization.

The LIDAR processing unit is also responsible for processing real time data into reconstructed surface, albeit at slower intervals. It first removes outliers from the point cloud, normalizes the points with WLOP simplification, and then reconstructs the surface using Poisson surface reconstruction.

3. LIDAR DATA PROCESSING

The LIDAR processing unit implements a set of methods that are built on the CGAL. The unit contains helper functions that are useful for the processing of the collected data.

3.1 Point Filtering

Since the goal of the LIDAR agent is to gather as much information in the environment as possible, we need to define what is valid information, and how one piece of information might contain more value than others. Table 1 shows the mathematical notation used in this paper.

Algorithm 1 Updating of point cloud $\mathcal{O}_t$

Require: Newly captured points: $\mathcal{P}_t$ at time step t, the point cloud at previous time step: $\mathcal{O}_{t-1}$, and the predetermined cut-off distance δ_{ob}

1: Initialize the valid newly captured point set: $\mathcal{V}_t \coloneqq \{\}$;
2: for $\mathbf{p} \in \mathcal{P}_t$ do
3: $min\_dist = \infty$
4: for $q \in \mathcal{O}_{t-1} \cup \mathcal{V}_t$ do
5: $min\_dist = \min(min\_dist, \|\mathbf{p} - \mathbf{q}\|)$ // $\|\cdot\|$ indicates Euclidean distance
6: endfor
7: if $min\_dist \geq \delta_{ob}$ then
8: $\mathcal{V}_t \leftarrow \mathcal{V}_t \cup \{\mathbf{p}\}$;
9: endif

Table 1. Mathematical notation

Drone States	
$\mathbf{x} = (\mathbf{x}, \mathbf{y}, \mathbf{z})$	The position of the drone
$\mathbf{M} \in \mathbb{R}^{\mathbf{n}\times\mathbf{m}}$	Projection map
$\mathbf{M}_d \in \mathbb{R}^{\mathbf{n}\times\mathbf{m}}$	Density map
$\mathbf{M}_c \in \mathbb{R}^{n\times m}$	Closeness map
(u, v)	The polar-coordinate position of a point
(i, j)	The index of a point on $\mathbf{M}$
DQN Configuration	
$s_t \in \mathcal{S}$	The state of drone/environment
$a_t(s_t) \in \mathcal{A}$	The action taken WRT s_t
$Pr_t(S' \mid S_t)$	State transition likelihood
$r_t(s_t, a_t) \in \mathbb{R}$	Immediate reward of a_t
$\gamma \in [0,1]$	Discount factor for the return
$\pi(a' \mid s_t)$	Policy function
$Q_\pi(s_t, a_t)$, $Q_\pi^*(s_t, a_t)$	Return and optimal return
LIDAR Data Processing Unit	
$\mathcal{O}_t$	The point cloud at t
$\mathcal{P}_t$	The newly captured points at t
$\mathcal{V}_t$	The newly captured valid points at t
δ_{ob}	Cut-off distance for valid points
$R(\mathbf{p}) : \mathbb{R}^3 \rightarrow \mathbb{R}$	Scaled reward function of newly captured point $\mathbf{p}$
Accuracy Analysis	
σ_x, σ_y	GPS error

continued on following page

Table 1. Continued

σ_z	Barometer error
ρ (pitch), κ (roll), ψ (yaw)	Drone orientation
σ_ρ (pitch), σ_κ (roll), σ_ψ (yaw)	Drone orientation error
θ	The angle of LIDAR
σ_θ	LIDAR angle error
σ_d	LIDAR range error

```
10:  endfor
```
$\mathcal{O}_t = \mathcal{O}_{t-1} \cup \mathcal{V}_t$

Algorithm 1 shows the generation of point cloud. The data processing unit stores the environment observations $\mathcal{O}_t$ as a set of existing LIDAR points, which should increase in size over time. For each time step t, the LIDAR collects a new set of points $\mathcal{P}_t$. Then, the newly collected points $\mathcal{P}_t$ are stored into the point cloud with only the valid information preserved.

We construct $\mathcal{O}_t$ from the previous observation $\mathcal{O}_{t-1}$ and the newly collected points $\mathcal{P}_t$. Intuitively, for each collected point $\rho \in \mathcal{P}_t$, we check if there are any points in $\mathcal{O}_{t-1}$ that are in the vicinity of ρ. If none, we conclude that ρ is indeed newly acquired information about the environment. We define a hyperparameter δ_{ob} as the smallest distance for point ρ to be away from the closest point in $\mathcal{O}_{t-1}$ so that ρ can be considered as newly acquired information.

To speed up the process of finding points, we use the k-d tree data structure to store $\mathcal{O}_t$. As a result, for each point ρ, the computational complexity of finding the closest point in $\mathcal{O}_t$ to ρ is only $\mathcal{O}\left(\log\left|\mathcal{O}_t\right|\right)$.

3.2 Information Gain by Newly Captured Points

We propose a new evaluation method that measures the information gain of each new point. It is less challenging for the drone to move with constant velocity than to change velocity; therefore, the agent is incentivized to move in one direction during the finite horizon, collecting points along the way. However, in this paper, we aim to have a holistic picture of the surrounding environment, rather than a corridor of LIDAR points stretching in one direction. To counter this incentive, we introduce an exponentially scaled point evaluation system that assigns more reward to the points that are closer to the origin. As shown in Figures 3 and 4, adding scaling changes the behaviors of the agent dramatically.

For any point ρ, the reward given by that point is defined as:

$$R\left(p\right) = \frac{a}{\xi \left\|p\right\| / \delta_{td}} \tag{1}$$

Figure 3. Points collected without scaling

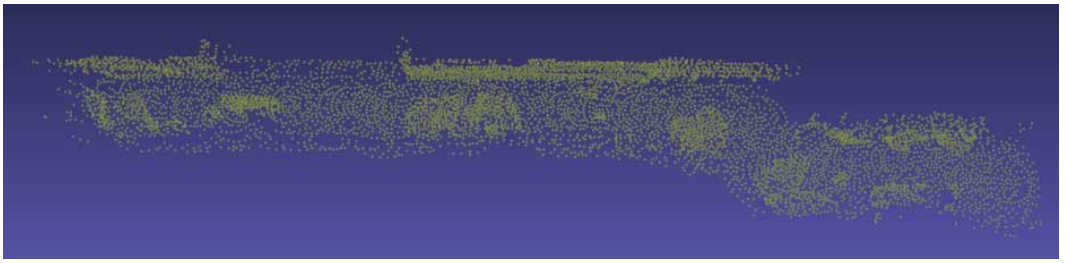
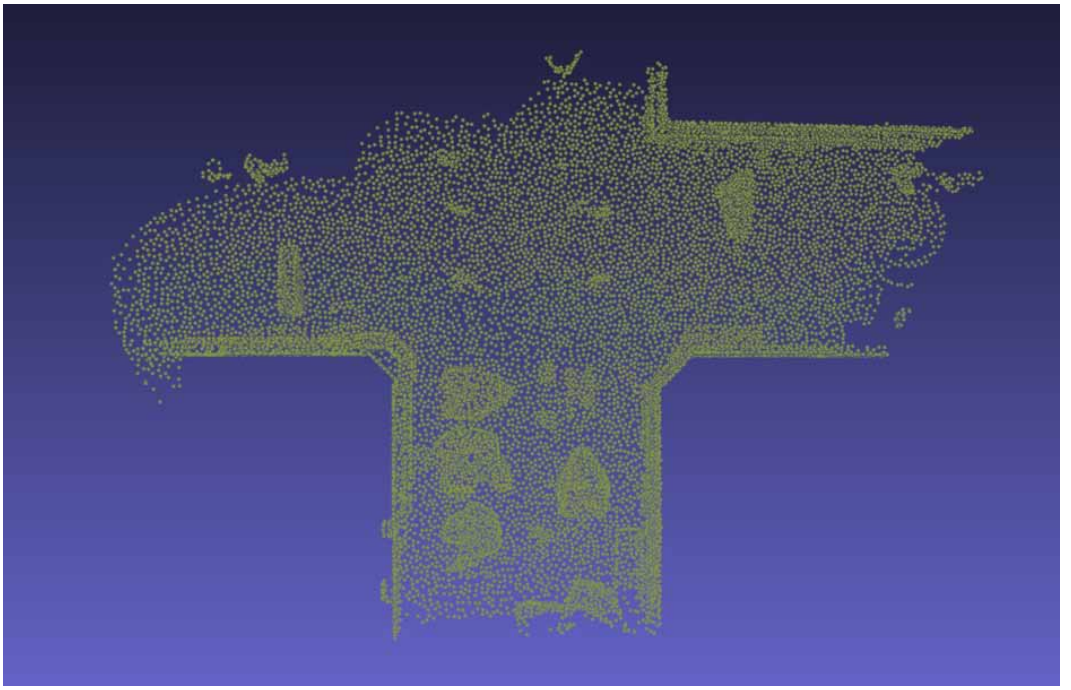


Figure 4. Points collected with scaling



where a denotes the unscaled reward for each point. In this paper, it is assumed that a is a constant that represents the information gain of a point without scaling. ξ denotes the regression factor, and δ_{td} denotes the threshold distance. For example, if $\xi = 2$ and $\delta_{td} = 10$, then the information gain given by one point 10 units away is equivalent to 2 points 20 units away, which is also equivalent to about 1000 points 100 units away.

3.3 Observation Representation

In this paper, we propose a novel method of representing the surrounding point cloud of the drone. The observation of the drone should only consist of the points that are in the range of its LIDAR camera. Furthermore, as shown in (Guo, 2020), most deep learning techniques applied to point clouds need to extract features from those point clouds, instead of using the original data set. The reason is that some nature of point cloud data, such as the high dimensionality, the sparseness of the data, and the irregularity of its shapes, makes point cloud data inefficient to be represented. Therefore, it

is preferred to transform the point cloud data from the high-dimensional space to a space of fewer dimensions for better performance in deep learning.

In this paper, we construct two $n \times m$ dimensional maps to represent the points surrounding the drone. For the first map, we represent the density of the surrounding point in each region. The points that are used to construct the two maps will be from the point cloud $\mathcal{O}_t$ at time t. Thus, the point clusters that are further away from the drone will be denser. This will incentivize the drone to find a distance that is optimal for point collection. The second map represents the closest point of the surrounding points in each region. The rationale behind this map is to make the drone avoid moving in certain directions when it detects some points that are too close to it.

To project the points onto the maps, we employ the Gall-Peters projection (Gall, 1885), which is a rectangular map projection that preserves the size of each shape on the sphere.

Let $E_t \subseteq \mathcal{O}_t$ be the points that are at most C_{range} away from the drone. Assume the drone is at point $\mathbf{x}$. For each point $\mathbf{y}$, let $\mathbf{y} - \mathbf{x} = (x, y, z)$. Then the point's position on the map $\left(u, v\right)$ can be derived as:

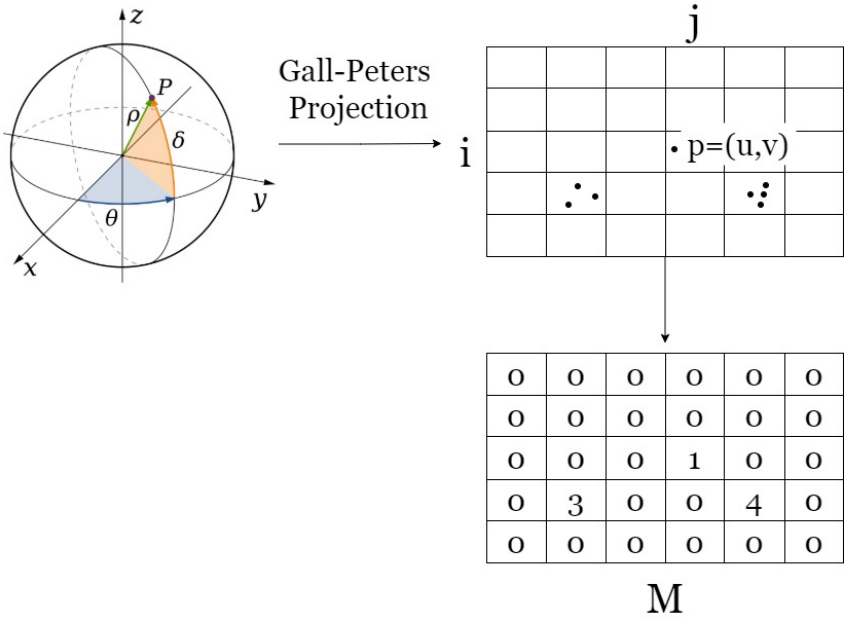
$$u = \arctan 2(y, x) \tag{2}$$

$$v = \frac{2z}{\left\|\mathbf{y} - \mathbf{x}\right\|} \tag{3}$$

where arctan2() is the two-argument function that gives the unambiguous angle for the polar coordinates when converting from Cartesian coordinates. Then, the width of the map would be the range of the function $\arctan 2()$, which is 2π; and the height of the function would be 4.

To apply machine learning, we further convert the map with points into a matrix where each element corresponds to a point. As shown in Figure 5, to construct the grids, we need to determine which row and column will each point be in. Let the position of point p on the map be (u, v). Then the index of the point on the map (i, j) is:

Figure 5. Projection of points to the projection map



$$i = \left\lceil \frac{u + \pi}{2\pi / n} \right\rceil \tag{4}$$

$$j = \left\lceil \frac{v + 2}{4 / m} \right\rceil \tag{5}$$

which indicates that point p is in the ith row and the jth column in the grids. As Figure 5 shows, this process results in a projection map $\boldsymbol{M} \in \mathbb{R}^{n \times m}$, where each grid contains the list of points. Then the density map $\boldsymbol{M}_d$ can be constructed as:

$$\boldsymbol{M}_d(i, j) = \frac{| \boldsymbol{M}(i, j) |}{C_{max}} \tag{6}$$

and the closeness map $\boldsymbol{M}_c$ can be constructed as:

$$\boldsymbol{M}_c(i, j) = \min(\{| p |: p \in \boldsymbol{M}(i, j)\}) \tag{7}$$

where C_{max} is a constant that can be calculated using the detecting range of the drone C_{range}, the minimum closeness of each point d_{ob}, and the dimensions (n and m) of the map.

3.4 Data Preprocessing and Postprocessing

Given the nature of the real-time mapping process, there are different sources introducing data error. Preprocessing and postprocessing need to be applied to the raw data to reduce noise and improve image reconstruction accuracy. Postprocessing also needs to be performed in real-time to keep up with the rest of the application's processes (Guarda, 2017; CGAL, 2020; Javaheri, 2017; Haider 2019).

3.4.1 Data Preprocessing

Position Smoothing: Depending on network conditions, the position of the drone maybe updated at inconsistent frequencies in the virtual environment. This can lead to points being reported at incorrect locations, which results in an inaccurate scan. This effect to the server is that the position of the drone appears to jump around instead of moving at some velocity. We use a linear Kalman filter to process the noisy, Gaussian data into more accurate telemetry data in real-time, at the same frequency that the telemetry (200Hz) data is fed into the filter (Slowak, 2019).

Based on our preliminary experiment, it is observed that Kalman filter can smooth the generation of point cloud and provide better data quality (Brock, 2022a) compared with the scenario without using Kalman filter (Brock, 2022b).

Outlier Removal: If the average Euclidean distance e between a point D and its nearest neighbors is greater than the threshold of outliers Ω, i.e., $\sum_{e=1}^{\omega} D_e / \omega > \Omega$, where ω is a hyperparameter that represents the number of neighbor points that are used to calculate the average distance, the point should be removed from the scan.

3.4.2 Postprocessing

While the following steps for post-processing occur at a lower frequency than most of the other functions of the system, they still produce meaningful display results while a scan is being recorded. Using a variety of techniques, data is processed during the scan to replace the raw data simultaneously.

The postprocessing algorithms can build more meaningful associations with the surrounding data and preserve storage when points are converted into surfaces.

Computation of Normals: The Poisson surface reconstruction algorithm needs each point's normal vector to be pointed inside the surface. Usually, the unoriented normals can be oriented by constructing a Riemannian graph over the points, deciding an initial orientation, and drawing a minimum spanning tree over the graph (Hoppe, 1992). However, this method is very computationally expensive as the time complexity of the operation is $O(n^2)$. For a point set of 10,000 points, the method used 4 hours to complete the operation in our initial experiment.

As one of the technical contributions of this paper, we propose a new method of orienting the normals by utilizing the drone's positional data and the LIDAR data. The new method reduces the time spent dramatically compared to the contextual approach. As the acquisition of the point cloud is associated with light reflection from the surface of an object to the LIDAR device, we can add an additional vector to each point called the orientation vector

$\mathbf{w} = \mathbf{p} - \mathbf{x}$ where $\mathbf{p}$ is the position of the point and $\mathbf{x}$ is the position of the drone. Then the oriented normal $\mathbf{v}'$ of $\mathbf{p}$ is

$$\mathbf{v}' = \begin{cases} \mathbf{v}, \text{if } \mathbf{v} \cdot \mathbf{w} \geq \mathbf{0} \\ -\mathbf{v}, \text{if } \mathbf{v} \cdot \mathbf{w} < \mathbf{0}. \end{cases} \tag{8}$$

With the additional information from the camera, the normals can be oriented in linear complexity. With the same data set of 10,000 points, our proposed method reduced the time spent to orient the normals from 4 hours to 20 seconds.

WLOP Simplification: The collected data points have various sources of noise and inconsistency. To remove most of the errors in the point cloud, we apply the Weighted Locally Optimal Projection (WLOP) to the data set (Lipman, 2007).

Surface Reconstruction: The processed point cloud will be visually recognizable as recorded object; however, it would be resource-demanding to load the entire point cloud, especially for big objects or objects with many details. To solve this issue, surface reconstruction is considered. The desired output should represent the point cloud well with less points and surfaces than the original point cloud. Furthermore, the constructed surface should be ``denser'' in more detailed areas. These criteria can be satisfied using Poisson Surface Reconstruction. The Poisson Surface Reconstruction method takes as input point-set $\mathcal{P}$ and each point's normal vector: $p.n(p \in \mathcal{P})$. The reconstruction algorithm assumes that the point-set is taken from the surface of a solid and the normals points to the inside of the solid tangent to the surface. The reconstruction method then solves for an approximate indicator function of the inferred solid with gradient that best matches with the normals. To convert the indicator function (which is a scalar function) into meshes, adaptive marching cubes were used to iso-contour the gradient of the indicator function.

4. NAVIGATION OPTIMIZATION BASED ON DEEP REINFORCEMENT LEARNING

Given the environment and the drone, we want to construct a policy such that, given the state of the drone and its surroundings, it will act to maximize the information acquired about the environment through LIDAR.

Since the state and action spaces are finite, we can model the optimization of the movement of the digital drone as a sequential decision-making problem, which can be further modeled as a Markov decision process (MDP).

Then, for each time step t, we can describe the MDP model as follows:

- State space $\mathcal{S}$: the set of the states about the drone's operational status and the observed environment. Let $s_t = (\kappa_t, \Psi_t) \in \mathcal{S}$ represent the state of the drone and its observation about the environment at time t. Here κ_t represents the newly acquired LIDAR points, and Ψ_t represents the operating state of the drone, which may include the drone's orientation, position, velocity, and the throttle of each fan.
- Action space $\mathcal{A}$: a set of actions of the drone. Let $a_t = (i_t, j_t, k_t)$ denote the drone's action of moving in at most one of three orthogonal directions in the three-dimensional space at any time t. Thus, only up to one of i_t, j_t, and k_t can be ± 1. Then the position of the drone at time $t+1$ can be denoted as $P_t + a_t \cdot C_{step}$, where P_t represents the position of the drone at time t, and C_{step} is a constant that represents the drone's flight distance for each time step.
- State transition probability $\{ Pr_t(s, a, s') = Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$: the probability of transition from state s to state s' under action a.
- K: the horizon over which the drone will act.

Then, the goal of the optimization problem is to find a policy for the drone: a function $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that, given the state s_t, outputs an action a_t that maximizes the accumulative knowledge about the environment in the given finite horizon. Mathematically, we need to maximize $\mathbb{E}[\sum_{t=0}^{T} \gamma^t r_t(s_t, a_t)]$, where $\mathbb{E}[\cdot]$ is the expectation taken over $s_{t+1} \sim Pr(s_{t+1} \mid s_t, a_t)$ and $\gamma (0 \leq \gamma \leq 1)$ is the discount factor of the reward r_t at different time steps. Due to the extreme curse of dimensionality in the state space $\mathcal{S}$ and the immense challenge of identifying transition probability $P(s_{t+1} \mid s_t, a_t)$, it is impractical to use exact methods such as linear programming and dynamic programming to solve the MDP problem. To address this challenge, we investigate a deep reinforcement learning framework where the digital drone is reinforced to learn a policy in a game engine generated virtual environment.

4.1 State Definition

In this paper, we assume that the drone has no prior knowledge of the environment. Therefore, the state of the drone should only be derived by collected LIDAR points and operational information the drone itself, such as the position and the throttle of each fan.

The state is characterized by two parts: the information about the environment through the collected LIDAR points, κ, and the operating state of the drone, Ψ. Then, at each time step t, the state of the DRL agent s_t is defined as $s_t = (\kappa_t, \Psi_t)$, where κ_t depends on the observation $\mathcal{O}_t$ described in Section 3.3. Specifically, $\kappa_t = (\boldsymbol{M}_d, \boldsymbol{M}_c)$. The operating state Ψ_t of the agent consists of two parts: its position and an indicator of collision, that is, $\Psi_t = (\mathbf{x}_t, \mathbf{1}_{\mathcal{S}}(\mathbf{x}_t))$, where $\mathbf{1}_{\mathcal{S}} : \mathbb{R}^3 \rightarrow \{0,1\}$ is a collision indicator function that depends on the environment and the position of the drone.

4.2 Action Definition

The set of actions include the movement of the drone in one of the three orthogonal directions in the three-dimensional space. To reduce the action space, action values are discretized. In addition, we may restrain the drone from moving up and down and control the drone to stay at a constant height if it is unnecessary for the drone to move vertically in collecting LIDAR points.

Given the position of the drone $\mathbf{x}_t = (x_t, y_t, z_t) \in \Psi_t$ .at time t, the position of the drone for the next time step can be generally derived as:}

$$\mathbf{x}_{t+1} = \begin{cases} (x_t + C_{step}, y_t, z_t) & \text{or} \\ (x_t, y_t + C_{step}, z_t) & \text{or} \\ (x_t, y_t, z_t + C_{step}) & \text{or} \\ (x_t - C_{step}, y_t, z_t) & \text{or} \\ (x_t, y_t - C_{step}, z_t) & \text{or} \\ (x_t, y_t, z_t - C_{step}) & \text{or} \\ (x_t, y_t, z_t) \end{cases} \tag{9}$$

4.3 Reward Function

The drone is rewarded based on the newly observed information gain but penalized if it collides with the environment. The reward is calculated by the data processing unit. Specifically, the reward function $r(s_t, a_t) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is decided by the number of scaled, newly acquired points. Thus, the reward function at time t is

$$r_t(s_t, a_t) = \sum_{\mathbf{p} \in \mathcal{V}_t} R(\mathbf{p}) + C_{col} \cdot \mathbf{1}_{\mathcal{S}}(\mathbf{x}_t)) \tag{10}$$

where $\mathcal{V}_t$ is the newly acquired valid points and defined in Algorithm 1; $R(\mathbf{p})$ is the scaled information gain function and is defined in Eq. (1); and $\mathbf{1}_{\mathcal{S}}$ is the collision indicator function and defined in Section 4.1. C_{col} is a negative constant representing the penalty when the agent is collided with the environment.

4.4 Optimal Q-value Approximation by DQN

With the defined state, action, and reward, the considered deep reinforcement learning (DRL) process can be further described. The expected accumulated discounted reward of policy π is defined as $\eta(\pi)$:

$$\eta(\pi) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t r_t(s_t, a_t)] \tag{11}$$

where $\mathbb{E}[\cdot]$ is the expectation taken over $s_{t+1} \sim Pr(s_{t+1} \mid s_t, a_t)$ and $\gamma (0 \leq \gamma \leq 1)$ is the discount factor of the reward at different time steps.

The goal of the learning algorithm is to determine the optimal policy π^* by estimating the optimal Q-function, which is defined as:

$$Q^*(s, a) = \mathbb{E}[\eta(\pi^*) \mid s_t = s, a_t = a]. \tag{12}$$

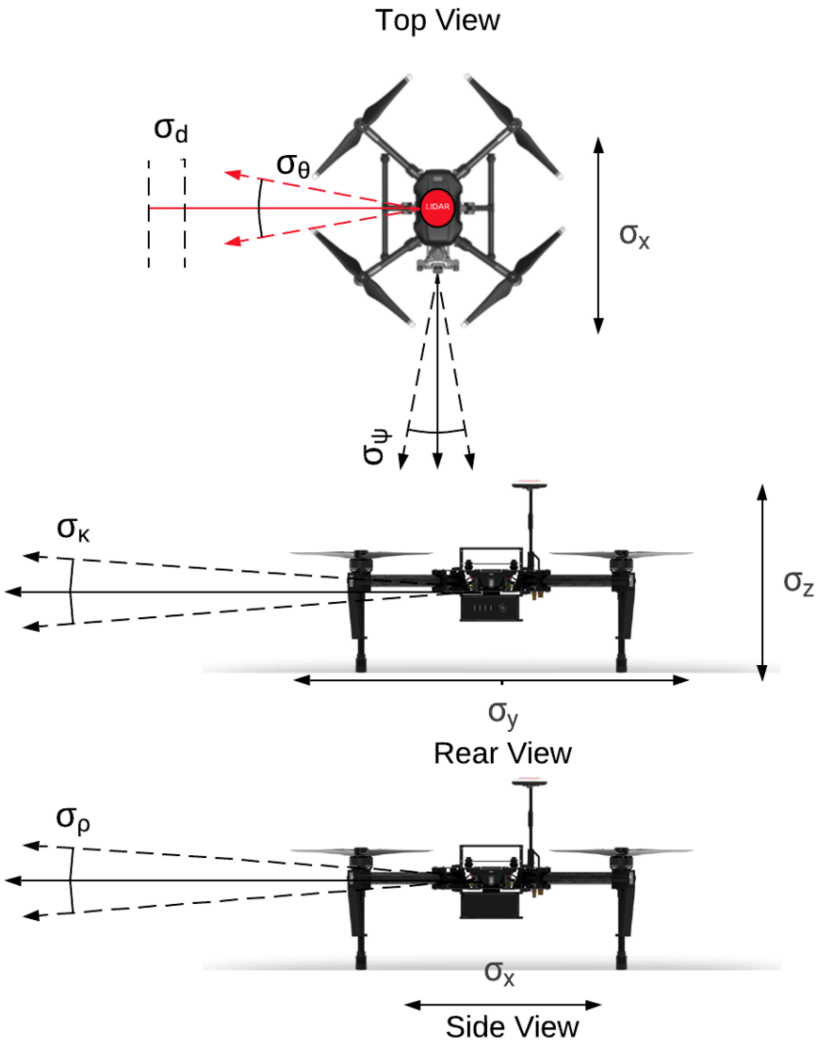
To approximate the optimal function, we use Deep Q-learning with experience replay and DQN (Mnih, 2013).

5. ACCURACY ANALYSIS

To further characterize the errors introduced during the drone-positioning process, Figure 6 provides a visual representation of all possible sources of error relative to the orientation of the drone, presented from three different views of the drone: top view, rear view, and side view, respectively. The drone has lateral error on all three axes, as well rotational error on all three axes. In addition to this, it has rotational error from the LIDAR, and distance error from the LIDAR's laser. All of these are defined relative to the airframe of the drone and are listed in Table 1 (May, 2007).

$$\begin{bmatrix} \sigma_{tx} \\ \sigma_{ty} \\ \sigma_{tz} \end{bmatrix} = \begin{bmatrix} \sigma_x & w_x & \cos(\theta) \\ \sigma_y & w_y & \sin(\theta) \\ \sigma_z & w_z & r_z \end{bmatrix} \begin{bmatrix} 1 \\ d \\ \sigma_d \end{bmatrix} \tag{13}$$

Figure 6. Error components



where

$$w_x \triangleq \sin(\theta)\sin(\sigma_\theta)+\sin(\theta)\sin(\sigma_\phi)+\cos(\theta)\sin(\sigma_\phi) \quad (14)$$

$$w_y \triangleq \cos(\theta)\sin(\sigma_\theta)+\cos(\theta)\sin(\sigma_\phi)+\sin(\theta)\sin(\sigma_\kappa) \quad (15)$$

$$w_z \triangleq \sqrt{(sin(\theta)sin(\sigma_\kappa))^2 + (cos(\theta)sin(\sigma_\rho))^2} \quad (16)$$

$$r_z \triangleq \sqrt{(sin(\theta)sin(\kappa))^2 + (cos(\theta)sin(\rho))^2} \quad (17)$$

Equations (13) - (17) can be described as a breakdown of the significance of certain error sources under certain circumstances. As such, many of the sources of error are amplified or reduced depending on the recorded angle of the scanned point. One linear source of error is distance, as all sources of error except σ_x, σ_y, σ_z, and σ_d are increased by the distance of the scanned point. The quantification for σ_{tz} is slightly different because the effects of pitch and roll both manipulate the vertical position of a scanned point, while the yaw of the drone, when compensated for in pre-processing, does not.

6. EXPERIMENTAL EVALUATION

6.1 Simulation With AirSim

We evaluated the DRL-based drone navigation optimization method by conducting simulation using AirSim (Microsoft, 2021). AimSim is a simulator platform for AI research to experiment with deep learning, computer vision and reinforcement learning algorithms for autonomous vehicles such as drones and cars. AirSim provides application programming interfaces (APIs) to retrieve data and control drones/vehicles in a platform independent way. Table 2 shows the configuration of DQN for the optimal navigation of the LIDAR drone.

Figure 7 shows the point cloud acquired by the agent during an episode of the early learning phase. Figure 8 shows the acquired point cloud during the agent training with high scale.

Figure 9 shows the immediate reward obtained with the proposed reinforcement learning method and random walk, respectively, at different learning episode. Figure 10 shows that the comparison of the immediate rewards of random-walk and the DQN-enabled navigation with respect to learning steps. It can be observed that the RL method has achieved more reward than random walk.

6.2 Field Evaluation

6.2.1 Configuration of LIDAR Sensor

The physical setup of our aerial mapping platform consists of a modified DJI M100 equipped with a RPLIDAR A2 for recording 2D scans and a Raspberry Pi micro controller for all necessary networking

Table 2. Configuration of DQN for optimal navigation

Parameter	Value
Learning rate	0.0001
Batch size	64
Training frequency	5 episodes
Buffer size	10^6
Polyak update	1
Discount factor	0.99

Figure 7. Acquired point cloud during an episode of the early learning phase of the agent

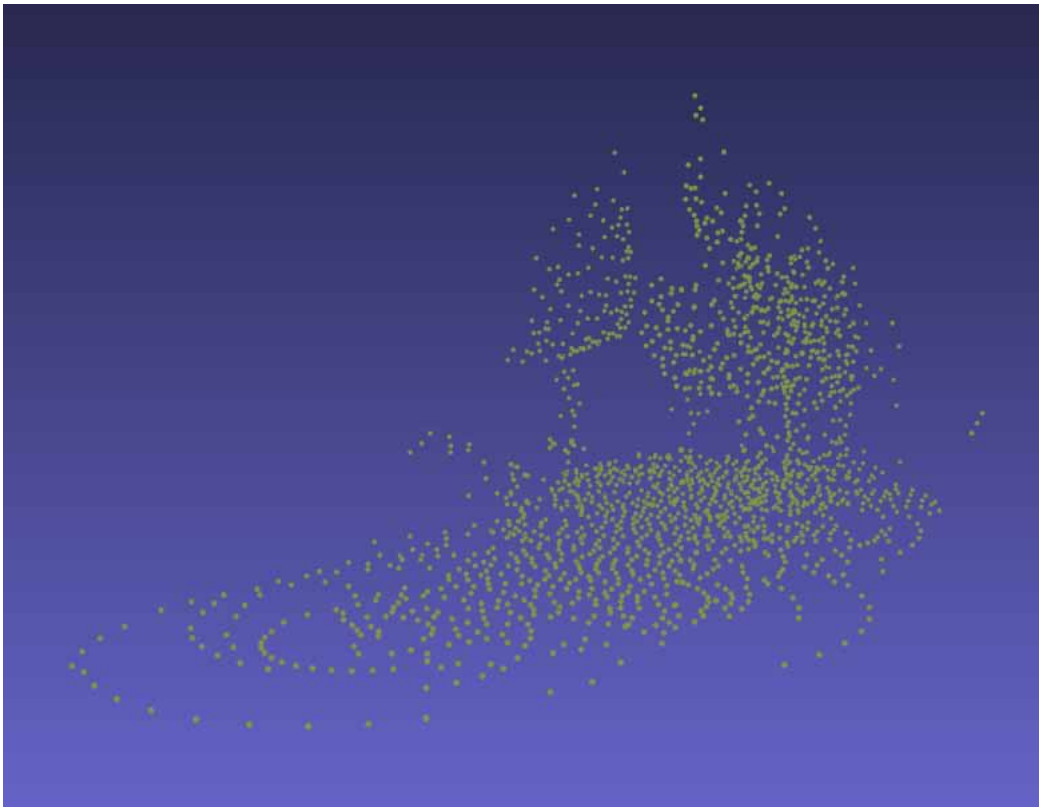
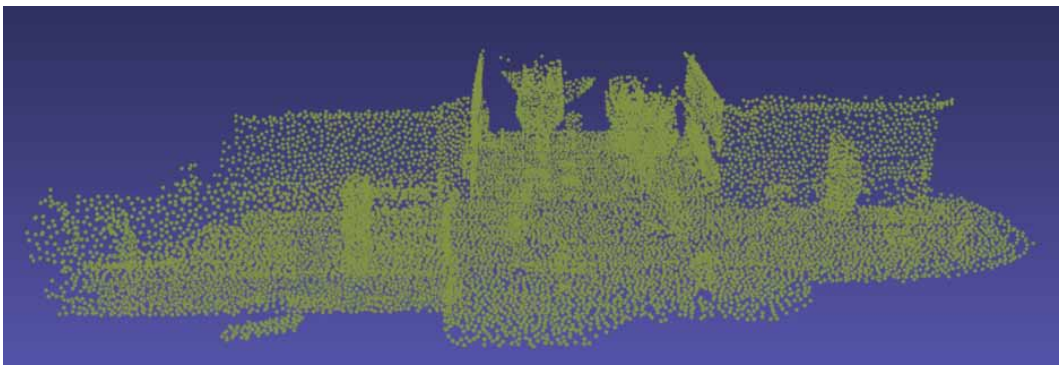


Figure 8. Acquired point cloud during the agent training with high scale



functions, as shown in Figure 11. A generalized diagram of how all the components of the drone relate to each other can be seen in Figure 12. Specifically, a GPS is used for absolute localization, and telemetry sensors are used for precise movements to store scans with respect to geographical coordinates.

Table 3 shows the configuration of the LIDAR used for field evaluation.

Figure 9. Immediate rewards with different learning episode

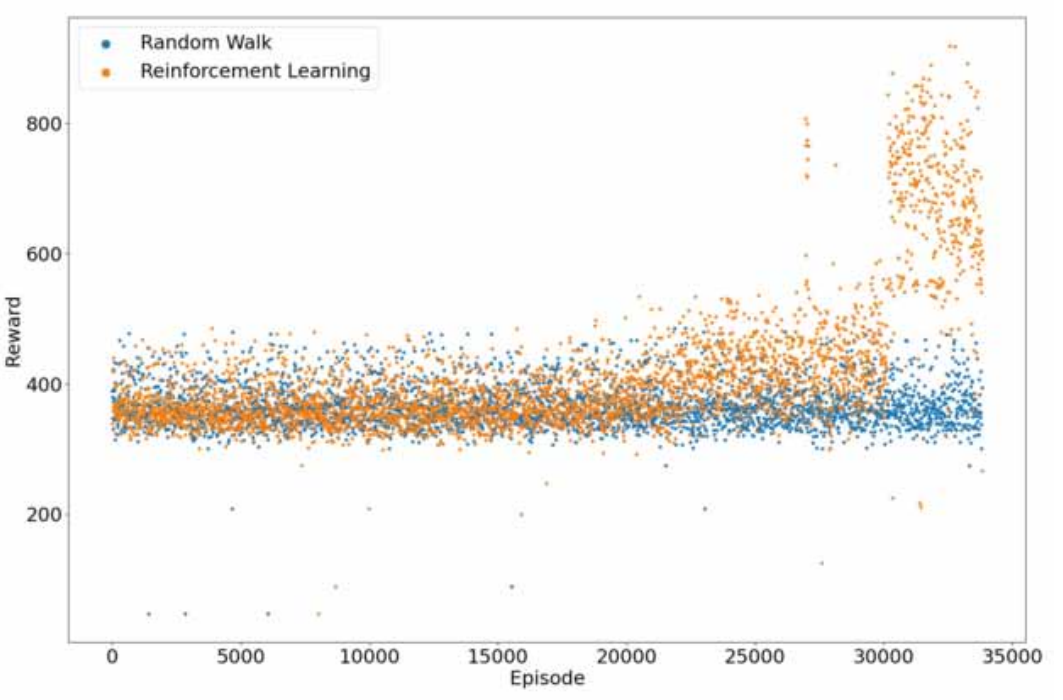
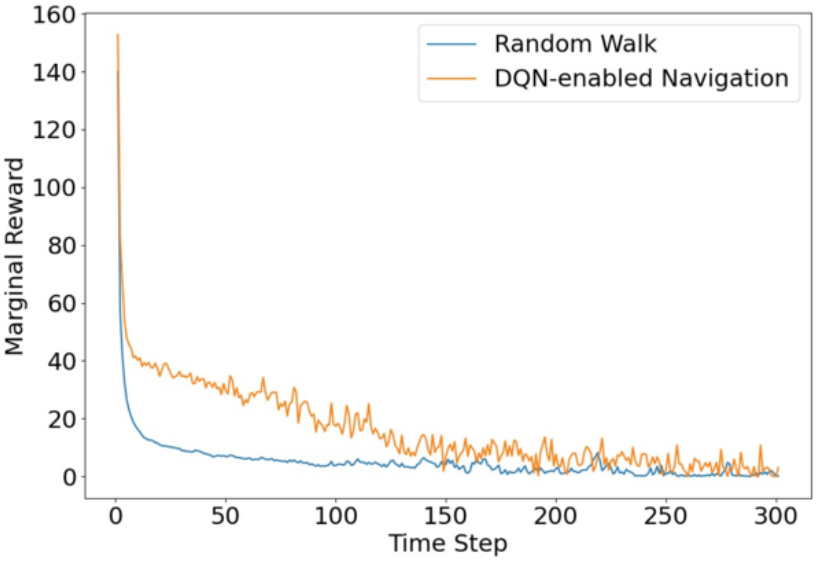


Figure 10. Comparison of immediate rewards of random-walk and DQN-enabled navigation at different learning time steps



6.2.2 Visualization Results Based on Game Engine

The combination of geographic localization and relative mapping results in a powerful application that can be used to update current geographic databases easily or construct a new one from scratch.

Figure 11. The LIDAR mapping drone platform employed in the experiment

Figure 12. High-level infrastructure of drone

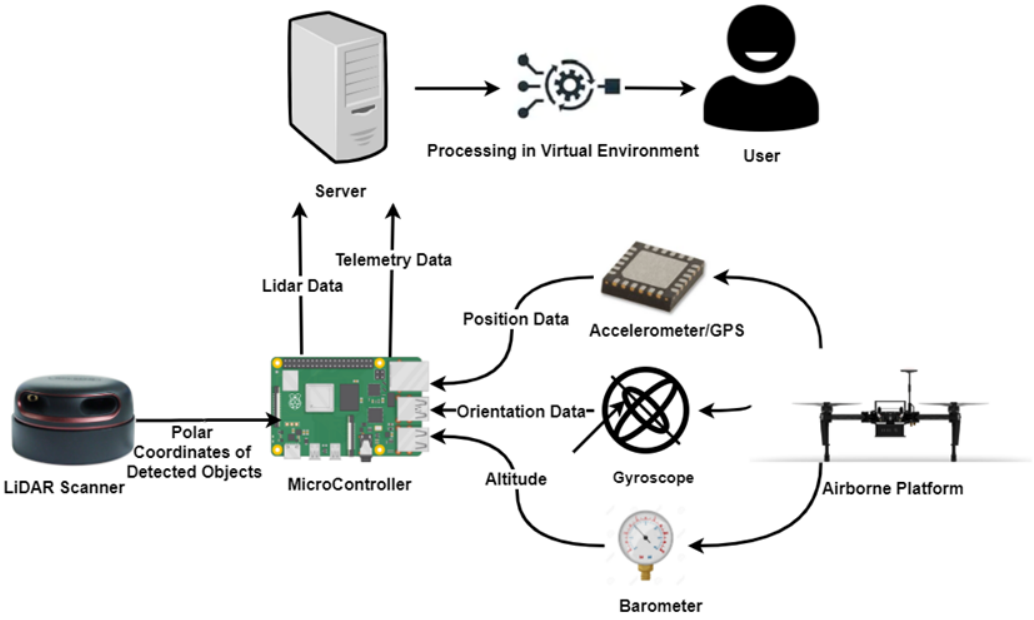


Table 3. Configuration of LIDAR

Parameter	Value
Range	10
Captured points per second	3000
Rotations per second	10
Number of channels	16
Position of LIDAR	(0, 0, -1)
VerticalFOVLower	0
VerticalFOVUpper	360

Figure 13. Building and nearby field for reference

This is due to all scans containing data relating to their real-world coordinates, which makes them very easy to locate in a 3D environment.

When we take the real-world environment in Figure 13 from Google Earth, and scan it in two separate sessions, as shown in Figure 14, the result is still a single large scan, with a seamless border between the two. This test was limited in range due to networking limitations but could easily be improved by using a dedicated antenna aboard the drone to transmit the data back to the host machine instead of a micro-controller over WIFI. However, the results still show that the system can put scans in their respective location within a virtual environment with no further operator input other than the initial altitude zeroing. This makes it a highly efficient system for recording and rending large scale scans of real-world environments, a useful tool for either updating or creating mapping databases.

6.2.3 Surface Reconstruction Using 3D Mesh

Our post-processing method is capable of turning nearly incomprehensible scans represented as point clouds into much more tangible polygons. Figure 15 and Figure 16 show a scan of an indoor environment before and after post-processing. Our method is capable of turning sparse vertices into solid planes that much more accurately portray the area scanned.

Figure 14. The building and the field are scanned separately but are combined automatically in a 3D environment

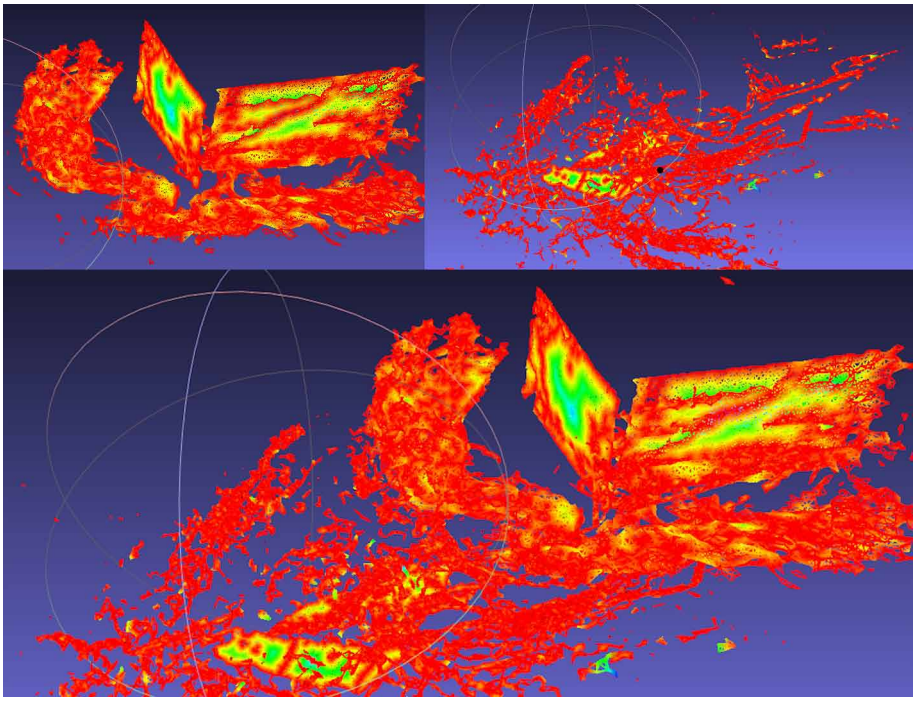
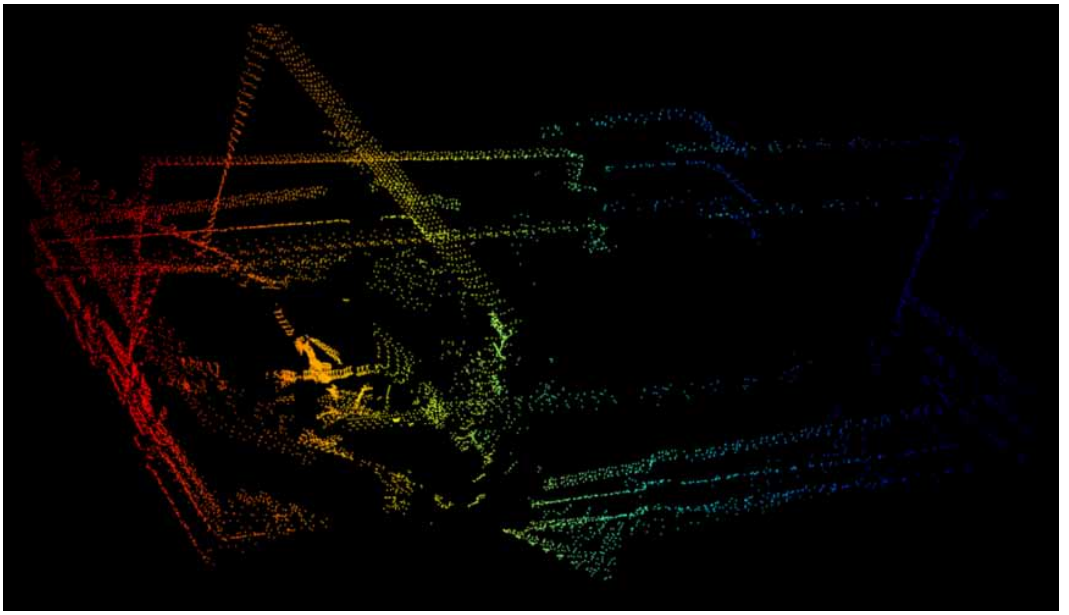
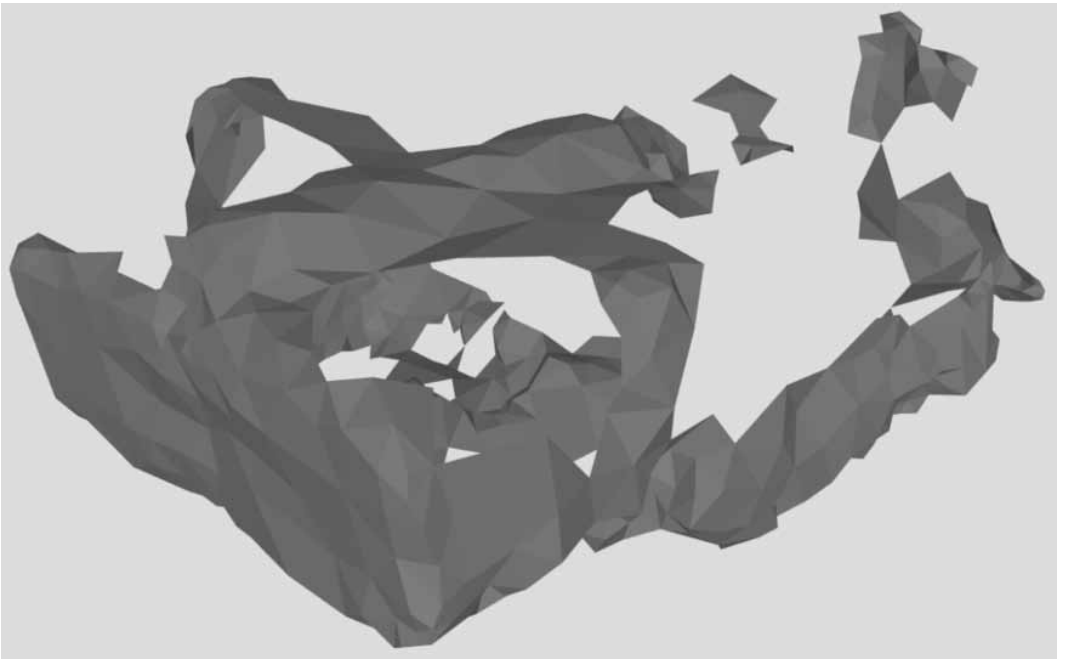


Figure 15. Raw point cloud data



For point clouds with greater density that are already comprehensible, our method is still helpful for improving accuracy along uniformly defined surfaces. Using this method on such surfaces reduces

Figure 16. Post-processed data with surface reconstruction



the error produced by the many possible error components from airborne LIDAR scanning. The system will inherently produce a jagged surface with the raw point cloud but possesses enough data density that post-processing can fit the data much more accurately to the actual plane or uniform surface that the points were scanned on. As shown in Table 4, the average distance from a point to the plane of best fit is reduced by about 40% after processing. The graphical representation of the plane fitting can be seen in Figure 17.

Figure 17. Planes of best fit over a cropped portion of raw (left) and over a cropped portion of processed data (right), respectively

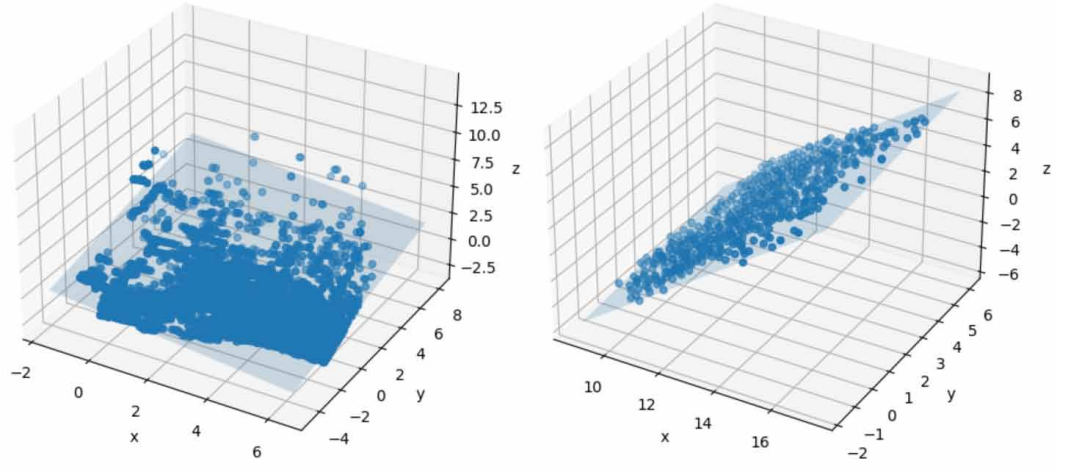


Table 4. Average distance to plane of best fit

	Before Processing	After Processing
Average Distance	2.86	1.73
Point Count	4760	531

Our results were recorded with a two-dimensional LIDAR. The quality and density of the scan could be improved greatly by using a three-dimensional LIDAR, especially since the horizontal configuration of the LIDAR makes it difficult to scan the ground or other horizontal surfaces. However, the concept and functionality can be applied to most hardware.

7. CONCLUSION

In this paper, we proposed machine learning and data processing methods for LIDAR-based real-time urban mapping using game engine. Specifically, we utilize digital environment to train a DRL-based airborn LIDAR agent to facilitate field LIDAR data acquisition. Methods for improving the accuracy of LIDAR data recorded by airborne platforms were studied. Both preprocessing and postprocessing techniques for rendering real time dynamic meshes in a game engine and surface reconstruction were developed. Those methods and techniques provide insight into overcoming the challenges associated with dynamic mesh rendering in real-time urban mapping by using game engine for digital twin development.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation under grant numbers 1647175 and 1924278.

REFERENCES

Agrawal, P., Iqbal, A., Russell, B., Hazrati, M. K., Kashyap, V., & Akhbari, F. (2017). PCE-SLAM: A real-time simultaneous localization and mapping using LiDAR data. *Proceedings of IEEE Intelligent Vehicles Symposium (IV).* doi:10.1109/IVS.2017.7995960

Andrews, C. (2020). *Gamification in GIS and AEC*. Retrieved from https://www.esri.com/arcgis-blog/products/arcgis/3d-gis/gamification-in-gis-and-aec/

Brock, E., Clark, S., Wu, D., & Liang, Y. (2022a). *Technical report - collected point cloud with kalman filter.* Retrieved from https://drive.google.com/file/d/17qLu0KvosD33C3JK2vpVurjePRXtJ-pi/view?usp=sharing

Brock, E., Clark, S., Wu, D., & Liang, Y. (2022b). *Technical report - collected point cloud without kalman filter*. Retrieved from https://drive.google.com/file/d/1OORt IRIISLdaIzXL0xne4G3odQqIbW1/view?usp=sharing

CGAL. (2020). *CGAL User and Reference Manual*. Retrieved from https://doc.cgal.org/Manual/

Chong, Z. J., Qin, B., Bandyopadhyay, T., Ang, M. H., Frazzoli, E., & Rus, D. (2013). Mapping with synthetic 2D LIDAR in 3D urban environment. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems.* doi:10.1109/IROS.2013.6697035

Chong, Z. J., Qin, B., Bandyopadhyay, T., Ang, M. H., Frazzoli, E., & Rus, D. (2016). Synthetic 2D LIDAR for precise vehicle localization in 3D urban environment. *Proceedings of IEEE International Conference on Robotics and Automation*.

Gall, J. (1885). Use of cylindrical projections for geographical, astronomical, and scientific purposes. *Scottish Geographical Magazine*, *1*(4), 119–123. doi:10.1080/14702548508553829

Google. (2021). *Maps SDK for Unity Overview - Google Maps Platform Gaming Solution*. Retrieved from https://developers.google.com/maps/documentation/

Guarda, A. F. R., Bioucas-Dias, J. M., Rodrigues, N. M. M., & Pereira, F. (2017). Improving point cloud to surface reconstruction with generalized tikhonov regularization. *Proceedings of IEEE 19th International Workshop on Multimedia Signal Processing (MMSP).* doi:10.1109/MMSP.2017.8122287

Guo, Y., Wang, H., Hu, Q., Liu, H., Liu, L., & Bennamoun, M. (2020). Deep learning for 3d point clouds: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*(12), 4338–4364. doi:10.1109/TPAMI.2020.3005434 PMID:32750799

Haider, A., & Tan, S. (2019). Improvement of LiDAR data classification algorithm using the machine learning technique. In J. M. Craven, J. A. Shaw, & F. Snik (Eds.), *Polarization Science and Remote Sensing IX, International Society for Optics and Photonics* (Vol. 11132, pp. 232–240). SPIE. doi:10.1117/12.2525415

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., & Stuetzle, W. (1992). Surface reconstruction from unorganized points. *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, 71-78. doi:10.1145/133994.134011

Javaheri, A., Brites, C., Pereira, F., & Ascenso, J. (2017). Subjective and objective quality evaluation of 3d point cloud denoising algorithms. *Proceedings of IEEE International Conference on Multimedia Expo Workshops*. doi:10.1109/ICMEW.2017.8026263

Lipman, Y., Cohen-Or, D., Levin, D., & Tal-Ezer, H. (2007). Parameterization-free projection for geometry reconstruction. *ACM Transactions on Graphics*, *26*(3), 22. doi:10.1145/1276377.1276405

Mapbox. (2021). *Maps for Unity*. Retrieved from https://www.mapbox.com/unity/

May, N., & Toth, C. (2007). Point positioning accuracy of airborne LiDAR systems: A rigorous analysis. Proceedings of Photogrammetric Image Analysis.

Microsoft. (2021). *AirSim*. Retrieved from https://microsoft.github.io/AirSim/

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *Proceedings of NIPS Deep Learning Workshop*.

Rusu, C. (2021). *AR, VR, gamification: cutting-edge technologies applied in smart cities*. Retrieved from http://citisim.org/ar-vr-gamification-cutting-edge-technologies-applied-in-smart-cities/

Saddik, A. E. (2018). Digital twins: The convergence of multimedia technologies. *IEEE MultiMedia*, *25*(2), 87–92. doi:10.1109/MMUL.2018.023121167

Slowak, P., & Kaniewski, P. (2019). LiDAR-based SLAM implementation using Kalman filter. *Proceedings of Radioelectronic Systems Conference*.

Toroslu, I., & Dogan, M. (2018). Effective sensor fusion of a mobile robot for SLAM implementation. *Proceedings of the 4th International Conference on Control, Automation and Robotics*. doi:10.1109/ICCAR.2018.8384648