PURDUE ENGI

CE 597 – UAS-Based LiDAR Mapping
Spring 2022

LYLES SCHOOL
OF CIVIL
ENGINEERING

ASSIGNMENT 2: LIDAR POINT CLOUD PROCESSING (PART 1)

Due date: April 17th

Objective:

Get hands-on experience with LiDAR point cloud processing and data characterization. You will be working on the following task: point density estimation.

Given:

1. Point clouds acquired from a (a) Velodyne VLP-32C and (b) Riegl miniVUX-1DL LiDAR unit. Figure 1 show the two LiDAR units and the UAV-based mobile mapping systems (MMS). Figure 2 shows sample point clouds acquired from the two LiDAR units.

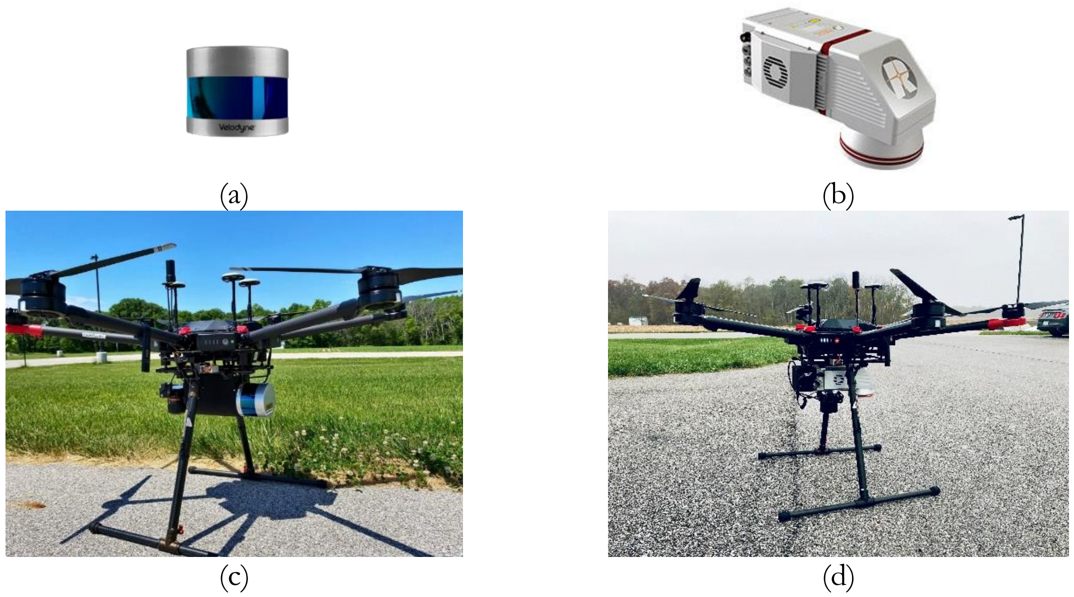
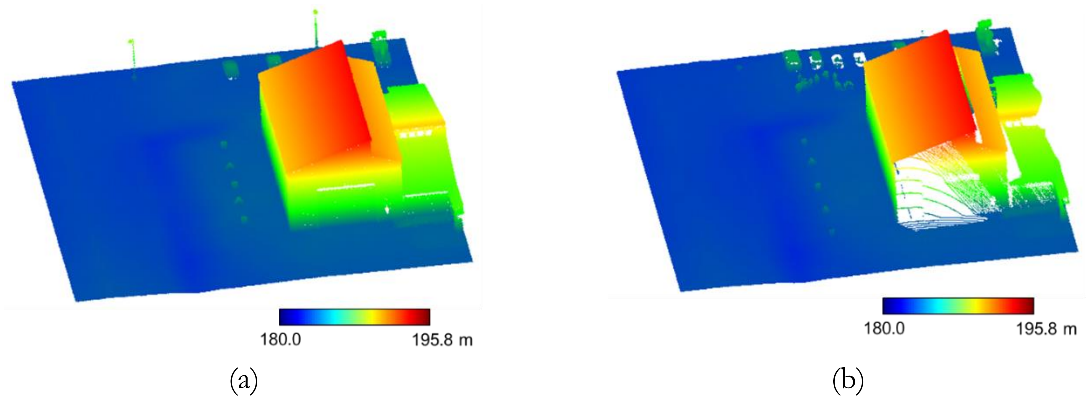

(a) (b) (c) (d)

Figure 1: LiDAR units showing: (a) Velodyne VLP-32C and (b) Riegl miniVUX-1DL. UAV-based MMS carrying a (c) Velodyne VLP-32C and (d) Riegl miniVUX-1DL.

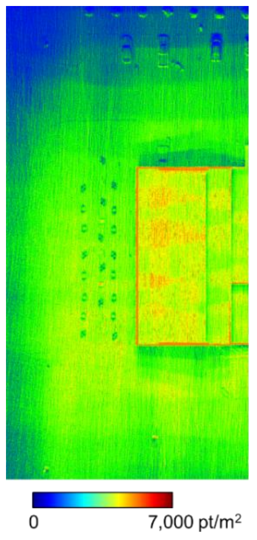


(a) (b)

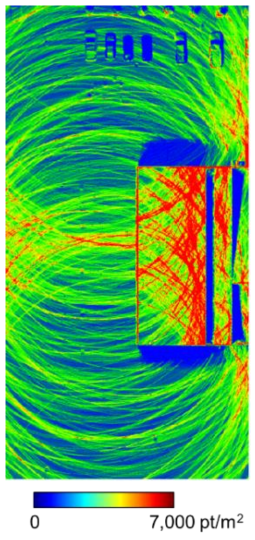
Figure 2: Point clouds derived from: (a) Velodyne VLP-32C and (b) Riegl miniVUX-1DL.

Task I: Point density estimation

For this task, you need to estimate the local point density within the area of interest using the box-counting method (refer to slide 9 in Lec.7 LiDAR Data Characterization slides) and generate a point density map that shows the point distribution variation along the XY plane. Figure 3 shows sample point density maps generated using Velodyne VLP-32C and Riegl miniVUX-1DL point clouds.



(a)



(b)

Figure 3: Sample point density maps for: (a) Velodyne VLP-32C and (b) Riegl miniVUX-1DL point clouds.

<u>Steps:</u>

1. Create a uniform 2D tessellation using square cells with a user-defined cell size (r) along the XY plane over the area of interest.
2. Calculate the number of points (n) in each cell.
3. Calculate the local point density of each cell (LPD_i) based on Equation 1.

$$LPD_i = \frac{n}{r^2} \tag{1}$$

4. Visualize the local point density as a color map.

<u>Requirement:</u>

1. Implement point density estimation using C/C++, Matlab*, or Octave. Note: Octave is a scientific programming language whose syntax is largely compatible with Matlab. Free software can be downloaded from: https://www.gnu.org/software/octave/index.
2. Generate point density maps for the two point clouds with the following cell sizes: 0.5 m and 1 m.

<u>Deliverables:</u>

Your report should include the following:

- The point density maps covering the area of interest;

<u>* Hints to improve the performance of your MATLAB code: see Appendix A</u>

- A summary of your observations, with the help of figures, of (a) the difference between the point density from the two LiDAR units and (b) the impact of cell size on the point density map;
- Explanation of any problems encountered, and
- Well-documented code.

Appendix A.

Objective:

The key objective of this appendix is providing some hints to improve the efficiency when processing large point clouds in MATLAB. Given that the size of LiDAR point cloud data is large, MATLAB code execution will be long due to inefficient code structure. Below are some useful tools you may need when working with point cloud data.

Preallocation:

for and while loops that incrementally increase the size of a data structure each time through the loop can adversely affect performance and memory use. Repeatedly resizing arrays often requires MATLAB® to spend extra time looking for larger contiguous blocks of memory, and then moving the array into those blocks. You can improve code execution time by preallocating the maximum amount of space required for the array.

The following code displays the amount of time needed to create a scalar variable, x, and then to gradually increase the size of x in a for loop.

```
tic
x = 0;
for k = 2:1000000
   x(k) = x(k-1) + 5;
end
toc
```

If you preallocate a 1-by-1,000,000 block of memory for x and initialize it to zero, then the code runs much faster because there is no need to repeatedly reallocate memory for the growing data structure.

```
tic
x = zeros(1,1000000);
for k = 2:1000000
   x(k) = x(k-1) + 5;
end
toc
```

Vectorization:

MATLAB® is optimized for operations involving matrices and vectors. The process of revising loop-based, scalar-oriented code to use MATLAB matrix and vector operations is called vectorization. Vectorizing your code is worthwhile for several reasons:

- Appearance: Vectorized mathematical code appears more like the mathematical expressions found in textbooks, making the code easier to understand.

- Less Error Prone: Without loops, vectorized code is often shorter. Fewer lines of code mean fewer opportunities to introduce programming errors.

- Performance: Vectorized code often runs much faster than the corresponding code containing loops.

Vectorizing Code for General Computing

This code computes the sine of 1,001 values ranging from 0 to 10:

```
i = 0;
for t = 0:.01:10
    i = i + 1;
    y(i) = sin(t);
end
```

This is a vectorized version of the same code:

```
t = 0:.01:10;
y = sin(t);
```

The second code sample usually executes faster than the first and is a more efficient use of MATLAB. Test execution speed on your system by creating scripts that contain the code shown, and then use the tic and toc functions to measure their execution time.

More information can be found in the following link:

- https://www.mathworks.com/help/matlab/matlab_prog/preallocating-arrays.html
- https://www.mathworks.com/help/matlab/matlab_prog/vectorization.html