Area Mesh Generation

Hang Li
Matriculation No.03749963
ge23zop@mytum.de

Tiantian Wei
Matriculation No.03711820
ge49rep@mytum.de

Ziting Huang
Matriculation No.0375493
ge95qus@mytum.de

Linya Ruan
Matriculation No.03728287
ge35wod@mytum.de

***Abstract*—This report explore the process of 3D slightly hilly terrain containing a polygon of n corners reconstruction using UAVs with down-pointing stereo-camera. A mesh reconstruction of area is generated from raw images after UAVs scans this area. The core modules are trajectory planning, feature matching, point cloud generation and merging, colored mesh generation.**

***Index Terms*—trajectory generation, feature matching, point cloud, mesh, 3D terrain reconstruction, opencv**

I. Introduction

In recent years, the use of unmanned aerial vehicles (UAVs) for aerial imaging has become increasingly popular in various applications such as environmental monitoring, agriculture, and urban planning [1]. In this technical report, we explore a comprehensive process for generating mesh from raw image data of stereo RGB cameras on drone. The raw images are processed using stereo_image_proc library in ROS and stereo SGBM algorithm to calculate the disparity image. The point cloud is then obtained from disparity image using camera parameters. The point clouds from each waypoint on the self-define trajectory have to be aligned in the world frame in order to create a complete 3D representation of the scanned terrain. We use Poisson reconstruction algorithm and greedy triangle algorithm in C++ to generate a mesh from the aligned point cloud. In addition, we explore the algorithm to color the mesh using RGB values from the original aligned point cloud. This project is achieved in simulation environment using Unity, which provides a realistic and interactive environment for testing and evaluating the proposed method.

II. trajectory generation

A. The requirement of trajectory generation

The movement trajectory should fulfill the following requirements:

1. Cover the terrain areas with distinct features (Figure 1).
2. Minimize overlap of the RGB images captured.
3. Prevent drone shake at each target point from affecting image quality by pausing for a few seconds at each point to capture the images.

B. The Explanation of trajectory generation

Following is the trajectory that we implement.

The process of path planning is illustrated in Figure 1. The input parameters consist of the initial position, maximum speed, maximum acceleration, fixed altitude, and target position. The drone takes off from the starting point and travels towards the first destination, where it hovers for 5 seconds.

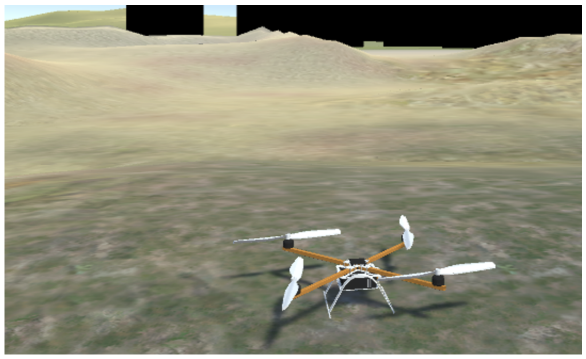

Fig. 1. Unity environment

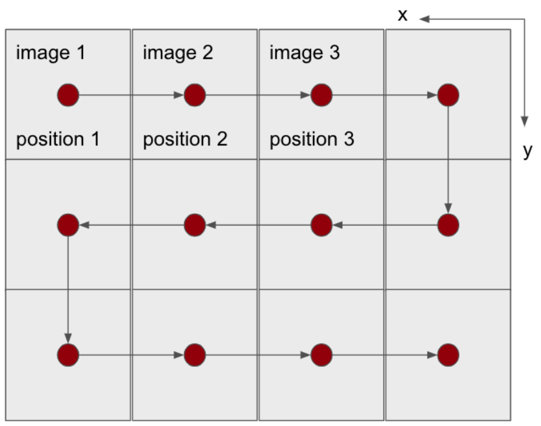

Fig. 2. trajectory planning

During the third second, it captures the first terrain map. The drone then moves towards the second destination, where it pauses again to take a second image. Ultimately, the drone returns to the starting position to complete the entire route.

III. Feature Machting and Point Cloud Generation

For point cloud generation the SGBM algorithm from OpenCV library and the ROS package stereo_image_proc are applied. SGBM algorithm can do feature matching and compute disparity. We can then use the intrinsic and extrinsic parameters of the camera to convert the disparity to point cloud. While stereo_image_proc can provide RGB point cloud directly from stereo images.

A. Point Cloud Generation using SGBM algorithm

In oder to apply SGBM method, we have to first convert ROS message to CV images using cv_bridge. The input are a pair of images and the output is a disparity image. The resulting disparity image is a grayscale image indicating the difference in pixels between corresponding points in the stereo images . It could be used to generate a point cloud, where each point corresponds to a pixel in the disparity image [2].

$$fx = 0.5 * width/tan(FOV/2) \tag{1}$$

$$fy = fx \tag{2}$$

$$cx = width/2 \tag{3}$$

$$cy = height/2 \tag{4}$$

Where f_x and $f_y$is the focal length of the camera, c_x and c_y are the principal point coordinates (the location of the optical axis on the image plane).

The next step involves computing the 3D position of each point in the camera image, which can be achieved by transforming the pixel coordinates of each point in the disparity image into 3D coordinates using the pinhole camera model. In our context of generating a point cloud from a disparity image, the pinhole camera model is used to transform 2D pixel coordinates in the disparity image to 3D coordinates in the camera frame.

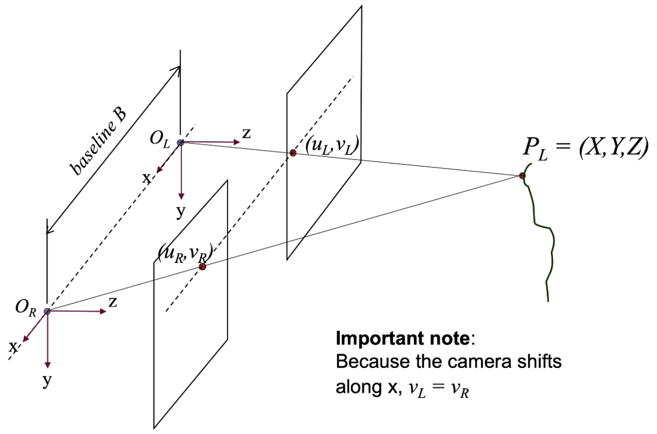

Fig. 3. Principle of Binocular Stereo Imaging [2]

$$u = (f * x/z) + c_x \tag{5}$$

$$v = (f * y/z) + c_y \tag{6}$$

These equations project a 3D point onto the image plane, and the (u, v) pixel coordinates can be used to locate the corresponding point in the disparity image. The resulting (x,y,z) image coordinates are as below [3]:

$$x = (z/f) * (u - c_x) \tag{7}$$

$$y = (z/f) * (v - c_y) \tag{8}$$

$$z = z \tag{9}$$

Once the 3D position of each point in the camera image is determined, it is added to the point cloud, which is represented as a collection of points, where each point is denoted by its (x, y, z) coordinates. Adding each point to the point cloud creates a 3D representation of the scene captured by the stereo camera, which can be effectively visualized and processed using software tools such as PCL and RViz.

B. RGB Point Cloud Generation using stereo_image_proc

The stereo_image_proc package [4] provides the ability to produce point clouds with distinct color properties. With the help of this function, the mesh can be colored. Colored mesh improves the visual appeal of the generated point clouds and can be used for further purpose. Figure 3 shows a RGB point cloud from stereo_image_proc We simply need to switch the names of the current ROS topics to those that this package can recognize. It provides stereo camera information and raw images. Due to the inaccuracy of the published camera parameters, we need to change some topic values and the topic name for the camera information topic. Flexibility and configurability are two of the stereo image processing package's key advantages. It offers a variety of controls and options that can be changed to enhance the speed and precision of the point cloud generation procedure. For instance, it offers options for adjusting the parameters for producing disparity maps and point clouds.

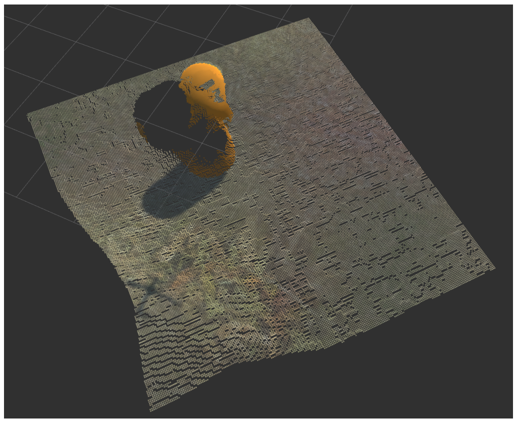

Fig. 4. RGB point cloud from stereo_image_proc

C. Transform to world frame

The generated point cloud are in 'camera' frame. In order to merge the point cloud, we have to transform them to world frame. Using `transformPointCloud` in PCL and `lookupTransform` in ROS TF library we can easily conduct transformation. The function `lookupTransform` function determine the transform between current frame and the desired frame. The function `transformPointCloud` accepts the point cloud, the transform, and the target frame's name as input. The transformation is then applied to each point, yielding a new point cloud in the target frame. It is possible to easily convert point clouds between different frames of reference in a ROS environment by combining these two functions.

IV. Point cloud Preprocessing

Preprocessing aims to enhance the quality of the point cloud data by removing noise, outliers, and redundant information, thus providing better and light weight point cloud data for efficient surface reconstruction. In our program, preprocess involves a sequence of filtering, smoothing, and down-sampling techniques, that are tailored to the specific requirements of the mesh generation process.

Down-sampling, the first step in preprocessing, reduces the number of points in the point cloud by sub-sampling, while still preserving the essential features of the point cloud. This step helps to reduce the computational load and allows for more efficient processing of the data. In our project, we set the leaf size as 0.2.

Smoothing is also an important preprocess method, which makes point cloud more smooth. In our project we use movingLeatSquares which is actually a resample method. PCL proide a function `RadiusOutlierRemoval` to filter outliers.

V. Mesh Generation

After getting a high quality point cloud we can start to generate mesh. Mesh generation is to convert a collection of points into a continuous surface model. In our project, we applied two distinct algorithms Greedy Triangle Mesh Generation and Poisson surface reconstruction.

A. Greedy Triangle Mesh Generation

The Greedy Triangle algorithm [7] involves repeatedly constructing triangles between neighboring points in the cloud. To start with the reconstruction, the algorithm selects a point at random from the cloud and locates the two points that are geographically closest to it. These three vertices come together to build the first triangle. After that, the algorithm chooses, for each new triangle, the closest point in the cloud. This point should be not part of a triangle, but it does have at least one neighbor who is part of a triangle. After that, the algorithm locates the two neighboring points to the new point. It builds a new triangle by connecting the new point with its two new neighboring points. This process of selecting points and building triangles is repeated until the surface reconstruction is finished. Figure 5 is the generated triangle mesh in our project. The greedy triangle algorithm has several advantages over other meshing algorithms, including Simplicity, efficiency, and the capability to generate a mesh with high quality. Furthermore, the algorithm is able to process point clouds of any topology or density.

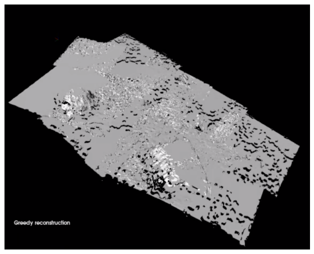

Fig. 5. Greedy Triangle Mesh

B. Poisson surface reconstruction

Poisson surface reconstruction algorithm is another approach to generating a mesh from a point cloud. This algorithm takes a set of unorganized points as input and generates a watertight mesh as output [6]. The algorithm works by constructing a scalar function over the input points using a set of local polynomial functions. This scalar function is then used to generate a continuous surface representation through a process of implicit surface reconstruction [6]. Figure 6 shows the generated poisson reconstruction. One major advantages of the Poisson reconstruction algorithm is its ability to handle noisy data and fill in missing information. The algorithm can also produce a watertight surface, which means that the resulting mesh is a closed surface without any holes.

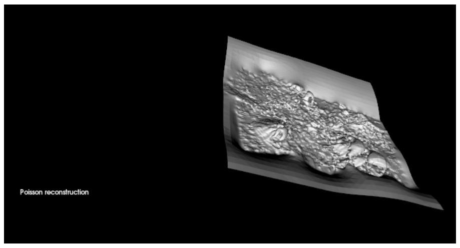

Fig. 6. Poisson surface reconstruction

C. Mesh Visualization and Color of the Mesh

After a successful surface regeneration, it is helpful to demonstrate the created mesh. This can be done in a number of ways. The Point Cloud Library (PCL) provides a powerful tool, the PCL Visualizer, for the visualization of point clouds, meshes, and other geometric objects. This tool also allows us to view the mesh in a 3D environment and interact with it using a variety of tools such as zooming, panning, and rotating.

Also, since the meshes are formed without any color specification, it might sometimes be challenging to identify their important features. In order to satisfy the demand of different visualisation options, two mesh coloring techniques are provided in the following part of this section.

1) ***Color from original point Cloud:*** It is often useful to color the mesh using the RGB values of the original point cloud in order to examine the mesh generation result. This can be achieved by assigning each vertex of the mesh the RGB value of the closest point in the original point cloud.

The algorithm to color the mesh involves finding the closest point in the original point cloud for each vertex of the mesh. To implement the algorithm, we first construct a k-d tree from the original point cloud. Then, for each vertex of the mesh, we query the k-d tree to find the closest point in the original point cloud. We then assign the RGB value of the closest point to the vertex. Figure 7 shows the generated poisson and greedy triangle reconstruction with the original RGB information.

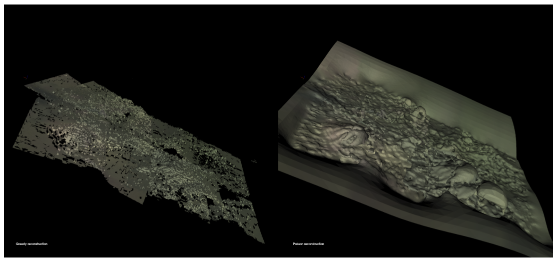

Fig. 7. Color from original point Cloud

The PCL (Point Cloud Library) provides a convenient way to perform this algorithm using the pcl::KdTreeFLANN class. This class implements a fast k-d tree search algorithm and can be used to efficiently find the closest point in the original point cloud for each vertex of the mesh [7].

Once the mesh is colored using the RGB values of the original point cloud, it provides a more realistic and visually appealing representation of the underlying scene [8].

*2) **Height Visualization**:* Since this project is developed to reconstruct a part of terrain, height visualization of the mesh can also be useful for the mesh analysis. In our program we define 2 medians between the minimal and the maximal height of the resulting fusion of point clouds and color every point according to their heights. The color varies then smoothly from blue (**low**) to red (**high**) depending on the height of each point. Using this coloring algorithm [9] the change of the height can now be visualized in the pcl-Visualizer in figure 8 and 9 [7].

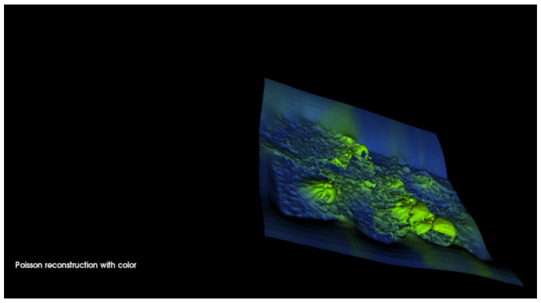

Fig. 8. Height Visualization of Poisson Mesh

CONCLUSIONS AND FURTHER POTENTIAL WORK

In this project, We can successfully generate mesh reconstruction as expected. The whole process that from raw stereo images to mesh goes smoothly. In addition we also add color information to point cloud successfully. However, there are still some problems. Firstly, the TF transform is the biggest problem of our project, two point cloud can never be aligned perfectly. Secondly feature matching is hard to conduct, because the ground does not have strong features, which can result in the inability to calculate disparity. As further work we can test this process in different unity environment.

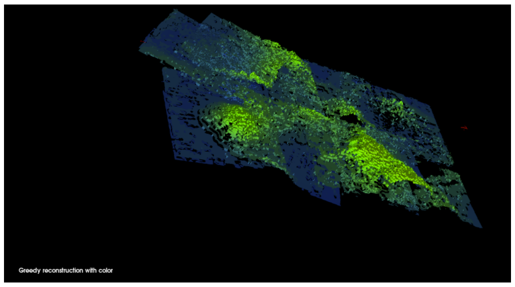

Fig. 9. Height Visualization of Triangle Mesh

REFERENCES

[1] Mohsan, S.A.H., Othman, N.Q.H., Li, Y. et al. Unmanned aerial vehicles (UAVs): practical aspects, applications, open challenges, security issues, and future trends. Intel Serv Robotics 16, 109–137 (2023). https://doi.org/10.1007/s11370-022-00452-4
[2] https://docs.ros.org/en/melodic/api/
[3] https://johnwlambert.github.io/stereo/
[4] https://learnopencv.com/depth-perception-using-stereo-camera-python-c/
[5] http://wiki.ros.org/rviz
[6] https://diglib.eg.org/handle/10.2312/SGP.SGP06.061-070
[7] https://pointclouds.org/documentation/tutorials/index.html
[8] http://t.csdn.cn/YpuzY
[9] http://t.csdn.cn/Dyq1a