Basic Implementation and Measurements of Plane Detection in Point Clouds

A project of the 2017 Robotics Course of the School of Information Science and Technology (SIST) of ShanghaiTech University

https://robotics.shanghaitech.edu.cn/teaching/robotics2017

Chen Chen
School of Information and Science Technology
chenchen@shanghaitech.edu.cn
ShanghaiTech University

Wentao Lv
School of Information and Science Technology
lvwt@shanghaitech.edu.cn
ShanghaiTech University

Yuan Yuan
School of Information and Science Technology
yuanyuan1@shanghaitech.edu.cn
ShanghaiTech University

***Abstract*—In practical robotics research, plane detection is an important prerequisite to a wide variety of vision tasks. FARO is another powerful devices to scan the whole environment into point cloud. In our project, we are working to apply some algorithms to convert point cloud data to plane mesh, and then path planning based on the plane information.**

I. *Introduction*

In practical robotics research, plane detection is an important prerequisite to a wide variety of vision tasks. Plane detection means to detect the plane information from some basic disjoint information, for example, the point cloud. In this project, we will use point clouds from FARO devices which contains a set of position(x,y,z, and) and the RGB color. At first, we will try to implement some of algorithms to extract plane information from point cloud data and do measurement between them. And then we might do some path planning based on this data and try to find a state-of-art solution on point cloud environment information path planning on indoor situation.

II. *State-of-the-Art*

A. *Papers*

Rusu and Cousins (2011) is the original paper that introduces the PCL, and it tells some basic information and creative points about PCL.

Schnabel et al. (2010) introduces an automatic algorithm to detect basic shapes in unorganized point clouds. Its application areas include measurement of physical parameters, scan registration, surface compression, hybrid rendering, shape classification, meshing, simplification, approximation and reverse engineering.

Cignoni et al. (2008) is the first introduction of the project MeshLab from Visual Computing Lab of the ISTI-CNR. In this paper, the authors describe the architecture and the main features and design objectives discussing what strategies have been used to support its development, as well somem useful examples of the practical uses.

Corsini et al. (2012) is working on dealing with the problem of taking random samples over the surface of a 3D mesh describing and evaluating efficient algorithms for generating different distributions. In this paper, the author we propose Constrained Poisson-disk sampling, a new Poisson-disk sampling scheme for polygonal meshes which can be easily tweaked in order to generate customized set of points such as importance sampling or distributions with generic geometric constraints.

Kazhdan and Hoppe (2013) is talking about poisson surface reconstruction, which can create watertight surfaces from oriented point sets.

Poisson surface reconstruction is a well known technique for creating watertight surfaces from oriented point samples acquired with 3D range scanners. The technique is resilient to noisy data and misregistration artifacts. However, it suffers from a tendency to over-smooth the data.

In this paper, the authors explore modifying the Poisson reconstruction algorithm to incorporate positional constraints, extend the technique to explicitly incorporate the points as interpolation constraints. This approach differs from the traditional screened Poisson formulation in that the position and gradient constraints are defined over different domain types. Whereas gradients are constrained over the full 3D space, positional constraints are introduced only over the input points, which lie near a 2D manifold.

Moreover the authors present several algorithmic improvements that together reduce the time complexity of the solver to linear in the number of points, thereby enabling faster, higher-quality surface reconstructions.

Deschaud and Goulette (2010) presents a fast and accurate algorithm to detect planes in unorganized point clouds using filtered normals and voxel growing. Their voxel growing method has a complexity of $O(N)$ and it is able to detect large and small planes in very large data sets and can extract them directly in connected components.

Poppinga et al. (2008) presents a very fast but nevertheless

accurate approach for surface extraction from noisy 3D point clouds. Instead of processing a global model, the sequential nature of the 3D data acquisition on mobile robots is exploited.

The Hough Transform is a well-known method for detecting parameterized objects. It is the de facto standard for detecting lines and circles in 2-dimensional data sets. For 3D it has attained little attention so far. Even for the 2D case high computational costs have lead to the development of numerous variations for the Hough Transform. Borrmann et al. (2011) evaluates different variants of the Hough Transform with respect to their applicability to detect planes in 3D point clouds reliably. Apart from computational costs, the main problem is the representation of the accumulator. Usual implementations favor geometrical objects with certain parameters due to uneven sampling of the parameter space. They present the accumulator ball as an accumulator design. The advantage of this design is that it does not unjustifiably favor planes with specific parameters, due to the equal size of the patches. The evaluation of the different Hough methods shows clearly that the Randomized Hough Transform is the method of choice when dealing with 3-dimensional data due to its exceptional performance as far as runtime is concerned. The randomized selection of points does not diminish but rather improve the quality of the result. Because points are removed from the data set once they have been found to lie on a plane, accuracy for the next plane increases. Comparison with the region growing by Poppinga et al. (2008) and the hierarchical fitting of primitives by Attene et al. (2006) shows that the RHT also competes with other plane extraction methods.

B. *Software*

1) ***Point Cloud Library (PCL)****:* Point Cloud Library (PCL) is a standalone, large scale, open project for 2D/3D image and point cloud processing, which is released under BSD license and free for research use. It is crossplatform, and has been successfully compiled and deployed on Linux, macOS, Windows and Android/iOS. The PCL framework contains numerous state-of-the art algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. These algorithms can be used, for example, to filter outliers from noisy data, stitch 3D point clouds together, segment relevant parts of a scene, extract key points and compute descriptors to recognize objects in the world based on their geometric appearance, and create surfaces from point clouds and visualize them. To work as a multi-functional software, the PCL has some independent parts which could be compiled separately, to serve as a module. The modularity design is important for distributing PCL on different platforms with reduced computational or size constraints, and could possibly gain a better performance. The PCL has its ROS package version, and we can utilize its functions with other ROS packages together.

Ying and Frstner (2010) introduces a new approach on plane detecting method by integrating RANSAC and MDL (minimum description length). The method could avoid detecting wrong planes due to the complex geometry of the 3D data. This paper tests the performance of proposed method on both synthetic and real data. It first introduced the principle of MDL encoding using a simple example for interpreting a set of points in a plane. Then the authors derived the description length of interpreting points in 3D space as a generalization of the previous part. After this, they recalled the basic approach of RANSAC algorithm for plane detection, which was introduced as a fore work to this paper. Finally, they give the proposed plane detection method by integrating RANSAC and MDL, together. With their implementation of algorithms, the experimental result was given in the last part of this paper. The key scheme to detecting planes are: The point cloud is partitioned into small rectangular blocks to make sure that there will be a maximum of three planes in one block. RANSAC is applied to extract planes in each block. The MDL principle is employed to decide how many planes are in each block. Eventually, there are zero to three planes in each block. This method could be seen as an enhanced version of RANSAC plane detection with the utilization of MDL.

2) ***MeshLab****:* MeshLab is an open source advanced 3D mesh processing software system,and is oriented to the management and processing of unstructured large meshes and provides a set of tools for editing, cleaning, healing, inspecting, rendering, and converting these kinds of meshes. MeshLab is free and open-source software, with lincense GNU General Public License (GPL), version 2 or later.

MeshLab is developed by the ISTI - CNR research center; initially MeshLab was created as a course assignment at the University of Pisa in late 2005. It is a general-purpose system aimed at the processing of the typical not-so-small unstructured 3D models that arise in the 3D scanning pipeline.

3) ***3D Builder****:* 3D Builder is a 3D computer-aided design tool for Microsoft Windows that makes it easy to create, view, edit, and 3D print 3D objects. It is developed by Microsoft and available for free in the Windows Store for all Windows platforms including Desktop, Phone, Holographic/HoloLens, and XBox. The user interface supports touch and uses a ribbon similar to the Office Mobile tablet apps. The app is easy for beginners but contains powerful tools available in higher end CAD software. Although the app supports a wide range of common 3d file formats, it is the primary viewer for the 3MF file format. The app is included with Windows 10 desktop and is widely viewed as one of the most used 3D apps worldwide.

3D Builder loads to a welcome page similar to the ones in Office Mobile, with options for loading 3D objects, scanning 3D objects, or choosing a 3D object to edit from a catalog of preinstalled items. Only one project can be edited at a time per window, but multiple windows can be opened simultaneously. Additionally multiple objects can be added to a scene.

Views for objects include Center view, X-ray, Shading, Wire frame, and Shadows. A view applies to all objects and cannot be customized.

Editing tools for objects include Simplify, Split, Smooth, Emboss, Extrude Down, Merge, Intersect, and Subtract buttons. Objects can be managed with traditional copy, paste, delete, rotate, and drag-and-drop buttons and keyboard short-

cuts. They can also be moved by finger and stylus movements. It is also possible to see the units of measure on the grid background and change the units of measure. This app does not use any visible scrollbars in the scene.

3D Builder includes a built-in mini tutorial and links to online resources.

Features

- 3D Builder provides everything you need to make any 3D content printable.
- Open 3MF, STL, OBJ, PLY, and WRL (VRML) files.
- Clean up models by smoothing and simplifying.
- Automatically repair models so you can print them.
- Use the 3D Scan app to scan yourself in full color.
- Take pictures with your webcam and make them 3D, or use BMP, JPG, PNG, and TGA files.
- Emboss any model with text or images.
- Drag-and-drop to build with simple shapes.
- Merge, intersect, or subtract objects from each other, or slice them into pieces.
- Add a base to uneven objects.
- Print directly to supported 3D printers using multiple materials or order your model through our preferred online printing service: i.materialise.com.
- Supported 3D printers: https://microsoft.com/3d and online 3D printing service i.materialise.com
- Print images of your 3D objects on paper.
- Save as 3MF, STL, PLY, or OBJ files.

III. *System Description of 1st half*

A. *Proposal*

As is suggested by the contents above, we have known that there are some algorithms which have proved the point cloud plane detection is feasible. We are going to implement one of these algorithms. If possible, we will also do some optimization on these algorithms. After the implementation part, we get the planes in a point cloud. Thus we could use the plane data to do some work like robot path planning, or tagging the areas for better robot mapping. The plane extraction is feasible, one potential problem is the system complexity. Since the sensor we use is the FARO laser scanner, the point cloud itself contains much information. We could possibly down sampling the data, or find a way to compress data without information loss, to make the plane detection faster.

B. *xyz2pcd*

We get the FARO data from Prof. Schwertfeger. The data is xyz format. For data containing position and color information, a line in an xyz format is as the following:

- index in the first dimension
- index in the second dimension
- x position
- y position
- z position
- red color in uint8 type
- green color in uint8 type
- blue color in uint8 type

However, PCL works on pcd format. For data containing position and color information, a line in an pcd format is as the following:

- x position
- y position
- z position
- rgb color in uint32 type

Hence, we write a python code to transform xyz to pcd. In our test, for an xyz file with size 106 MB (containing information of about 2.39 million points) , the transformation can be done with in about 7 seconds.

C. *Plane Extraction*

We write a cplusplus code with PCL to extract planes from point clouds. Here is the pseudo-code:

```
Read the pcd file.
Downsample the data. cloud_filtered = the points
    cloud after downsampling
Initialize n = N = the number of points in
    cloud_filtered.
Initialize r = some ratio subject to 0 < r < 1 (e.g.
     r = 0.3).
while(n > r * N)
  Detect planes in cloud_filtered with RANSAC
    algorithm.
  Add the inlier points to a new plane points cloud
    and save the new palne as a new pcd file.
  Remove the inliers from cloud_filtered.
  Update n = the number of points in cloud_filtered.
```

After running the program, several pcd files of planes will be written if some planes are found.

We write a cplusplus code to show the result. It reads pcd files of planes and set different colors for points belonging to different planes. Then it shows the result with pcl_viewer.

D. *Result and Analysis*

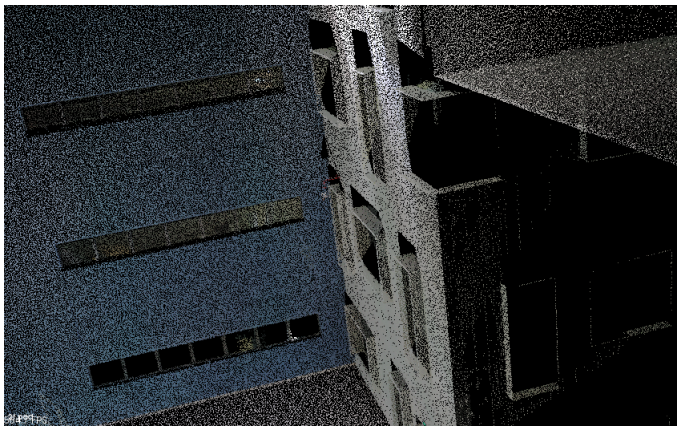

Fig. 1. 2nd floor ground truth

We use pcl_viewer to show the point clouds in pcd format. Figure 1 and figure 4 are about the raw data and we regard them as the ground truth.

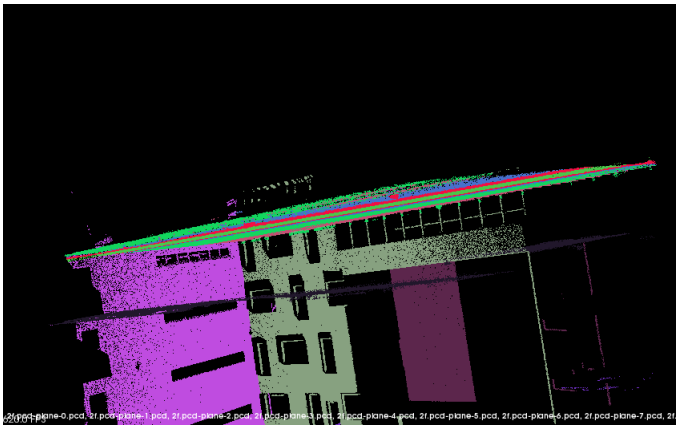

Fig. 2. 2nd floor result

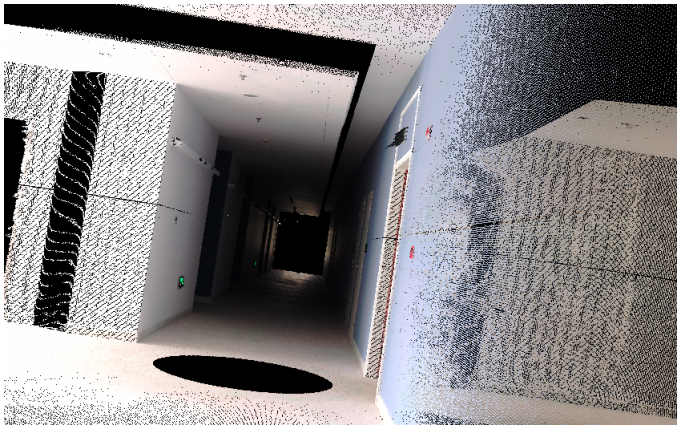

Fig. 4. 5th floor ground truth

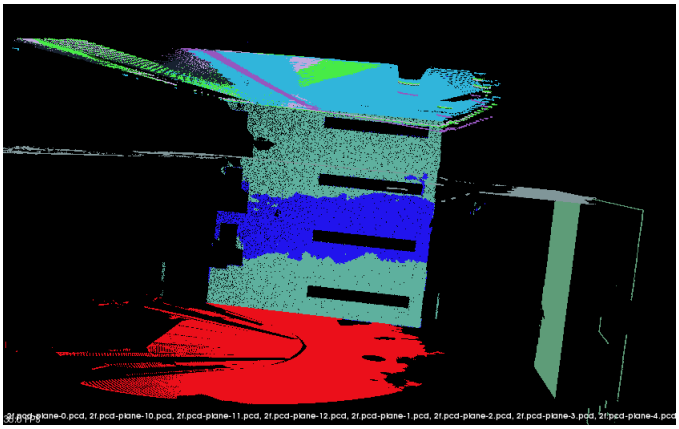

Fig. 3. 2nd floor result

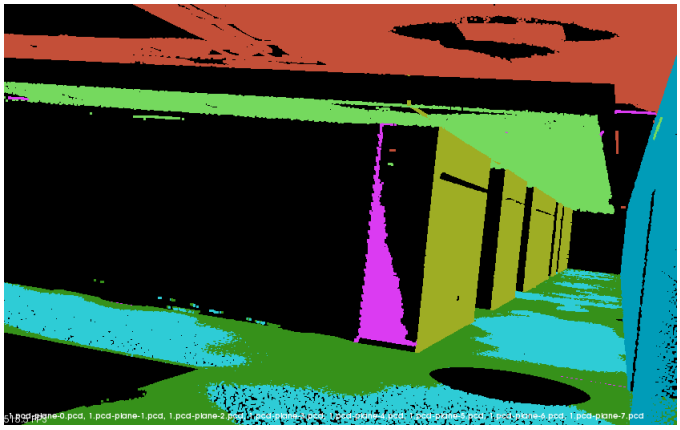

Fig. 5. 5th floor result

For processed data, we set different colors for points belonging to different planes. Figure 2 and figure 3 are results of data shown in figure 1. Since the two main planes are extracted, the result shown in figure 2 is good . The result shown in figure 3 is terrible because the wall is extracted as two lanes.The result shown in figure 5 has the similar problem.

In short, our program works but not very well. We will try to fix the problem in the 2nd half.

E. Work to be done in the 2nd half

We want to introduce a new interposition layer after the processing of the algorithm, so solve the current inconsistent problem. After the algorithm output each planes, our layer will detect each planes attributes, then automatically fix the error as general as possible.

Currently the planes are just set of points, it is hard to process. One problem is the memory bottleneck. The current plane is finite. It is not determined by a little points, but visually determined. If we want to load a bigger plane, the points quantity would be greater, which is not necessary. Secondly, points themselves are not continuous, which is hard for process. We will transfer those planes to a "real" plane, which is represented by some key points. Upon this work, we will construct a 2-D image presentation of planes.

Our third work is to do the plane alignment. If we want a global plane map of SIST building, one approach is to run the algorithm on those pre-aligned data. But this approach could potentially have some problems. First, the aligned data size is big. A single process could take a long time. Second, the data output could possibly not be accurate. Two planes which have do relation could be recognized as one same plane. So our plan is to run our algorithm on single scan, which is more accurate, and then manually align those outputs.

IV. *System Description of 2nd half*

In the latter half stage of our project, we achieved some progress based upon our previous work. In the first stage, we used PCL to extract planes within a point cloud. The performance of the build-in algorithm of plane extraction in PCL gains a kind of good performance with proper attributes set. The result that we could get from the first stage is a set of points which represent a plane set. Though this result has achieved the goal that we want to extract planes from a point cloud, it is not useful. Because:

1) The plane is finite, not a plane in math, just points.

2) The extracted information has a big size, which is not easy to utilize.
3) The result is still not accurate, as may due to the error of scanning or alignment.

In the 2nd half, we worked to try to get these problems resolved. First, we want to extract the edge information of every plane (hull). Second, we want to in some aspects compress the data to represent same plane with smaller space. What's more, we want to compensate the "hole" which is caused by the scanner's blind area.

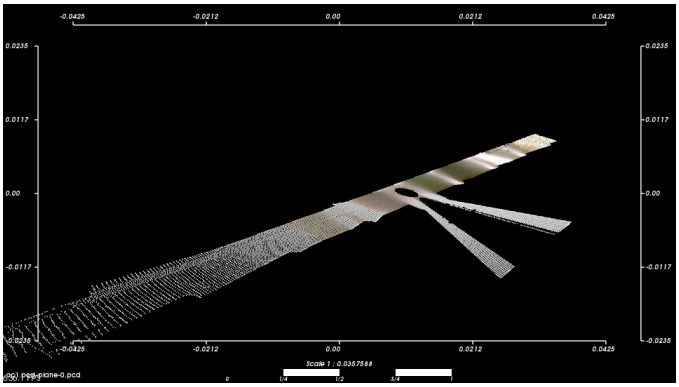

Fig. 6. ground plane (points)

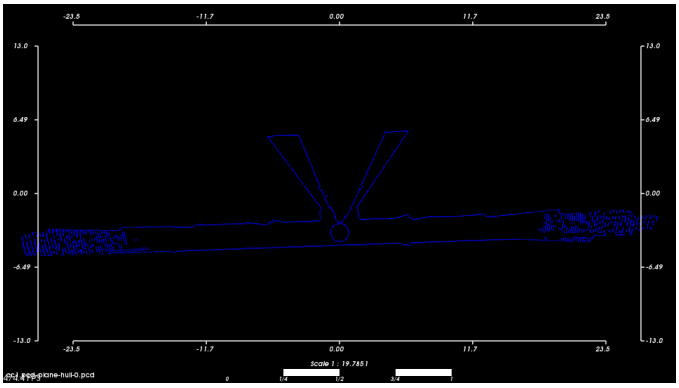

Fig. 7. hull of the ground plane

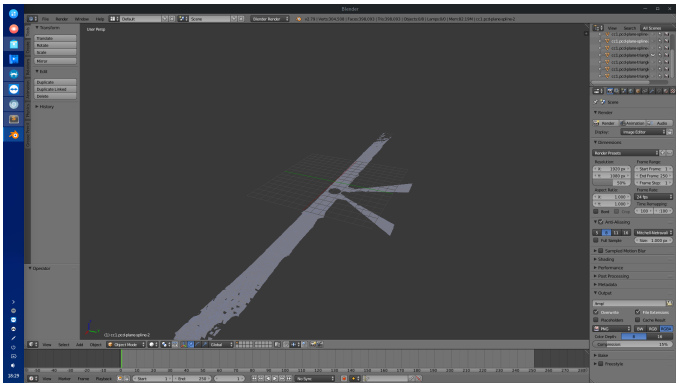

Fig. 8. triangulated ground plane

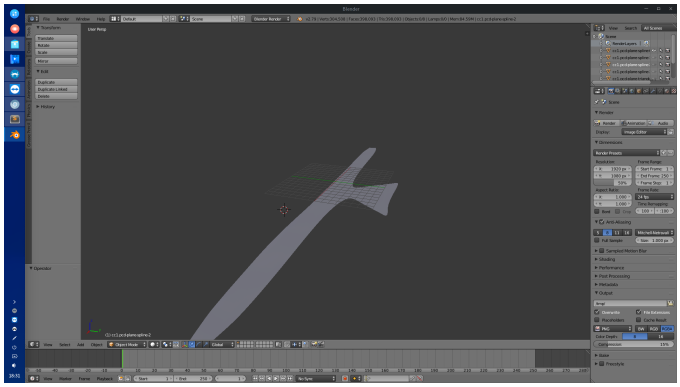

Fig. 9. flat plane of the ground plane

Figure 6 is the point cloud extracted from the 5th floor scan of the SIST building. It consists of many points, which represents our result from the previous stage. With this information, we can now get the hull of the point cloud.

A. *Hull extraction*

The hull extraction result of points in figure 6 is shown in figure 7. The "wave" pattern on the two sides are caused by the precision of the scanner, as there are fewer points. This work is done by the function "segment_hull" in our cpp code.

B. *Triangulation*

Figure 8 is the triangulation result of figure 6. This plane represents the ground, but in reality they are not on the same height. Thus we got some small triangular planes, with errors. This work is done by the function "fast_triangles" in our cpp code.

C. *B_spline*

We fit the triangular planes to a single flat plane, as is shown in figure 9. This work is done by the function "B_spline" in our cpp code. The iteration runs for around hours, but the result is not bad. With better parameters, we possibly could get better results.

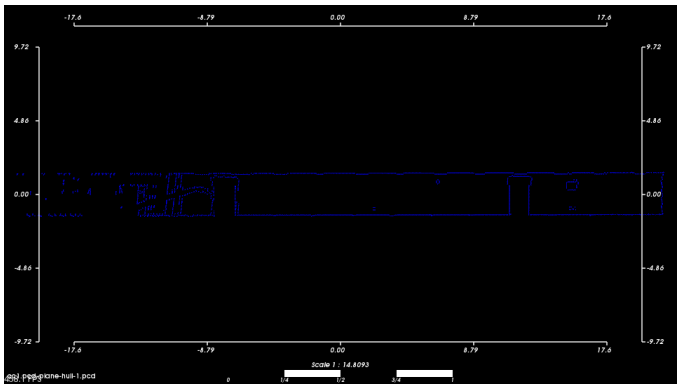

Fig. 10. hull of the wall plane

Here is another result. Figure 10 is the hull of the wall plane. Figure 11 is the triangulation result of the wall plane.

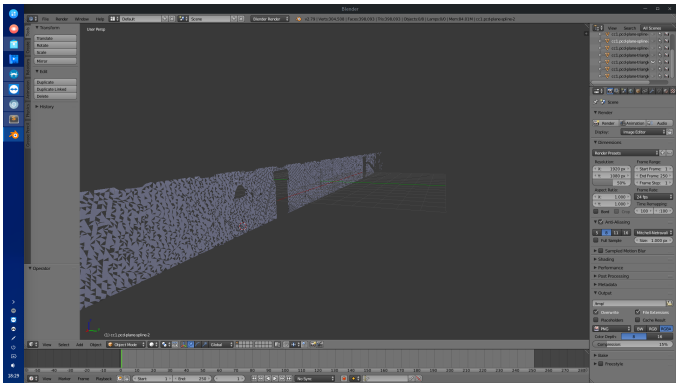

Fig. 11. triangulated wall plane

V. *System Evaluation*

First we have to check the correctness of our algorithm implementation as there could be some wrongly recognized plane in the output data. Once we can determine that weve got the correct planes, we optimize it.

We can compare the optimized result and the raw output of the algorithm. With the information that we have got, the infinite plane could be generated. We can compare the aligned planes map with the raw point cloud data, and see if every obvious planes have been extracted. A better evaluation method like visualization (texture mapping) or some other things could also possibly be introduced, too.

VI. *Conclusions and Future Work*

In this project we implement the plane detection in point clouds scanned by a Faro Focus 3D X330. We present a method to measure the quality of the plane detection. In the future, the result of plane detection will be used to robot navigation.

References

Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *IEEE International Conference on Robotics and Automation*, pages 1–4, 2011.

R Schnabel, R Wahl, and R Klein. Efficient ransac for pointcloud shape detection. *Computer Graphics Forum*, 26 (2):214–226, 2010.

Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. Meshlab: an open-source mesh processing tool. In *Eurographics Italian Chapter Conference 2008, Salerno, Italy*, pages 129–136, 2008.

M Corsini, P Cignoni, and R Scopigno. Efficient and flexible sampling with blue noise properties of triangular meshes. *IEEE Transactions on Visualization & Computer Graphics*, 18(6):914–924, 2012.

Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *Acm Transactions on Graphics*, 32 (3):1–13, 2013.

Jean Emmanuel Deschaud and Franois Goulette. A fast and accurate plane detection algorithm for large noisy point clouds using filtered normals and voxel growing. *3dpvt*, 2010.

J Poppinga, N Vaskevicius, A Birk, and K Pathak. Fast plane detection and polygonalization in noisy 3d range images. In *Ieee/rsj International Conference on Intelligent Robots and Systems*, pages 3378–3383, 2008.

Dorit Borrmann, Jan Elseberg, Lingemann Kai, and Andreas Nchter. The 3d hough transform for plane detection in point clouds: A review and a new accumulator design. *3d Research*, 2(2):3, 2011.

Marco Attene, Bianca Falcidieno, and Michela Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. *Visual Computer*, 22(3):181–193, 2006.

Michael Ying and Wolfgang Frstner. Plane detection in point cloud data. 2010.