ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Πλοήγηση και χαρτογράφηση ρομπότ με χρήση αισθητήρα βάθους
Robot navigation and mapping using depth sensor

Κουτούλας Αναστάσιος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υπεύθυνος
Δελήμπασης Κωνσταντίνος
Επίκουρος Καθηγητής

Λαμία, 2016

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Πλοήγηση και χαρτογράφηση ρομπότ με χρήση αισθητήρα βάθους
Robot navigation and mapping using depth sensor

Κουτούλας Αναστάσιος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων/σα
Δελήμπασης Κωνσταντίνος
Επίκουρος καθηγητής

Λαμία, 2016

Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [1], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. *Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί* ***χωρίς να τα περικλείω σε εισαγωγικά*** *και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

2. *Δέχομαι ότι η αυτολεξεί* ***παράθεση χωρίς εισαγωγικά****, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. *Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια*

4. *Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.*

Ημερομηνία: ……/…../20……

Ο – Η Δηλ.

(Υπογραφή)

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.

Πλοήγηση και χαρτογράφηση ρομπότ με χρήση αισθητήρα βάθους
Robot navigation and mapping using depth sensor

Κουτούλας Αναστάσιος

Τριμελής Επιτροπή:

Δελήμπασης Κωνσταντίνος, Επίκουρος καθηγητής

Πλαγιανάκος Βασίλειος, Αναπληρωτής καθηγητής

Μάρκου Ευριπίδης, Επίκουρος καθηγητής

Contents

Abstract 7

1 Introduction and related work 8

1.1 Related work 8

1.2 Purpose of this work 10

1.3 Equipment and tools 11

1.3.1 Kinect specifications 11

1.3.2 Installation of Kinect 13

1.3.3 Kinect Fusion Explorer D2D 13

1.3.4 The PLY File Format 14

2 Methodology 16

2.1 Mathematical operators 17

2.2 Decimation of point set 17

2.3 Identifying planar surfaces in the captured frame 20

2.3.1 Identifying one best fitting plane 20

2.3.2 Identifying multiple planes 20

2.4 Region Growing Algorithm 21

2.4.1 Automating region growing algorithm 29

2.5 Merging world points from different frames 30

2.6 Contour extraction 31

2.7 Quantification of error 33

2.8 Removal of common points between frames 35

3 Experiments and results 37

3.1 Experiment 1 37

3.1.1 Initialization 37

3.1.2 Decimation Results 37

3.1.3 Plane extraction and rotation estimation 38

3.1.4 Floor identification and merging results 41

3.1.5 Contour extraction results 43

3.2 Experiment 2 44

3.2.1 Initialization 44

3.2.2 Decimation Results 45

3.2.3 Plane extraction and rotation estimation 45

3.2.4 Floor identification and merging results 51

3.2.5 Contour extraction results 53

3.2.6 Experimenting with parameters 55

4 Conclusion and future work..56
5 References...57

Abstract

Mapping of real and complex environments is a commonly researched issue as methods for robot navigation are constantly improved. It can be useful to robots in order to move in the environment avoiding collisions which may harm them. Also it can be useful in mapping hazardous areas in which humans cannot enter. In this work we will present a method of mapping a real environment using Kinect depth sensor. Kinect sensor takes depth frames which are useful to both mapping the environment and setting a path. The experiments conducted for the purpose of this work took place in indoor environments (classroom and another room) and the translation between the captured frames was measured manually. In the first chapter we will make an introductive reference on mapping and robot navigation and why mapping is essential for navigation. Also we will discuss some works and papers which present methods for indoor mapping using Kinect-like sensors or other sensors. Furthermore we will talk about the equipment and tools used for the purpose of this work. In the second chapter we will describe the methodology used for implementing the method presented and take a look into the algorithms used for this purpose. In the third chapter we will discuss some experiments we have conducted and present the results of each of them which have been produced by using the methodology described in chapter 2. The results show that the method presented can map the environment with a small error. In chapter 4 we will discuss the meaning of the results and the efficiency of our method. Also we will talk about future work, which may improve this method. In the last chapter we show the references used in chapter 1 and generally in this work.

1 Introduction and related work

In modern robot navigation, robots need to have data regarding their surroundings so they are able to track themselves in the environment and plan their path avoiding obstacles in order to survive. This problem is known as SLAM (Simultaneous Localization And Mapping) and can be solved using different methods. For solving this problem researchers use laser sensors, depth sensors (like Kinect), cameras and other types of devices. Depth sensors are commonly used because knowing the depth value in a frame can help in the identification of possible obstacles. Although data taken from a depth sensor can be useful, the sensor itself has some drawbacks such as the field of view which is relatively small (0.8m to 3.5m usually) compared to laser sensors. A combination of sensors is also commonly used (e.g. camera and laser sensor). In the past few years there has been a major research interest in this field and many methods were presented, some of which work in real-time and in complex environments.

1.1 Related work

There are many papers and experiments already published which are discussing and/or experimenting in SLAM algorithms. Some of them carry out experiments using Kinect depth sensor or similar sensors.

- **Map generation**

As described in a paper [4] there are two representations of the environment when talking about mapping: the metric and the topological. The metric can be either geometric or grid-based. In geometric representations features like walls or corridors are directly mapped in respect of the world coordinate system whereas in grid-based representations [8] each grid has its own position in the environment which is given by the cell coordinates (x, y). Topological representations [7] aim at presenting the environment as a set of regions which are usually represented as nodes. Metric approach is faster and easier to write and update but are unreliable in large environments. On the other hand topological approach is more complex [9] but is better for position estimation and path planning [6]. In response to proposals of combining metric and topological representations the authors suggest a new method. Their method is related to proposals by Thrun [9], Arleo [5] or Zelinsky [10] but in addition their method extracts the topological map from the metric representation on-line.

- **Plane extraction and segmentation**

In a paper published in 2012 [14] the authors carry out experiments for indoor mapping using planes extracted by a method described in the paper. The authors preprocess their data by estimating the normal vectors and edge pixels taken by Kinect sensor. Then they apply the RANSAC algorithm if a group of points forming a plane is bigger than a predetermined threshold. After that they extract the boundary of the plane. When they have extracted all the planes in a frame they use a method in order to merge the different planes which represent the same region. From

experiments they have conducted we can see that this whole process is very fast and considered real-time.

Another paper published in 2011 [28] suggests a method for real-time plane segmentation. The authors present a fast process for the computation of local surface normals. Then they cluster these normals and segment all the planes formed by the point clouds. The authors also refer to some related with their work papers. One of them [27] uses 2D virtual range scans for both collision avoidance and localization. Another features two types of virtual scans, virtual structure and obstacle maps. The first type models environmental structures such as walls in a virtual 2D laser range scan and the second information about closest obstacles [26]. From the experiments carried out by the authors we can see that the processing times of their method are very low.

A different method for range image segmentation is presented in a paper published in 2012 [29]. The authors discuss several approaches on segmentation such as RANSAC-based segmentation and segmentation using Region Growing. In RANSAC-based segmentation a method suggests sequentially removing inliers from the original data set, and continuing the segmentation with the residual points [30]. Another approach is to first identify connected regions and apply RANSAC region-wise [31]. A very interesting and efficient approach is to decompose unorganized point clouds using an octree subdivision and apply RANSAC only to subsets of the original point cloud [32]. In region growing segmentation a method connects neighboring points in 3D laser range scans to a mesh-like structure and then those scans are segmented recursively by merging connected patches that are likely to lie on the same planar surface [33]. The authors then talk about their approach. They first use a fast mesh construction and then perform segmentation. The mesh construction uses a quad mesh and several triangulations and the segmentation is done by a region growing segmentation method which has not many differences with the other region growing algorithms used elsewhere.

- **2D Indoor Mapping using Kinect**

Another paper which was published in 2013 [15] suggests a method for converting Kinect's 3D depth data to 2D map in order to use it for indoor SLAM. The authors describe this process and prove it is better than using 2D or 1D sensors because those sensors may miss some objects in the scene due to its shape and position. The production of the real-time map was carried out by a base station on which the robot transmitted the data collected from Kinect.

A different method of mapping indoor spaces is suggested in a paper published in 2016 [16]. The authors of this paper use a Coarse-to-Fine registration method of the RGB-D data taken from Kinect. More specifically they use methods to extract and detect 2D visual features and use them to perform coarse registration. Furthermore they use an image-based segmentation technique for detecting regions in RGB images. Their results show that this method can map an indoor environment with a small error.

- **3D modeling of indoor environments using Kinect-like sensors**

In a paper published in 2012 [19] the authors introduce RGB-D Mapping. This paper is an extension of their previous work (2010). They use a frame-to-frame alignment RANSAC and demonstrate that it performs better than the Euclidean-space RANSAC. Then they use FAST features and Calonder descriptors instead of scale-invariant feature transform (SIFT) for feature identification. Last step in their work is to use sparse bundle adjustment (SBA) and show how to incorporate ICP constrains into SBA. Among others the authors refer to techniques for 3D mapping using range scanners [20][21][22] and stereo cameras [23][24][25].

- **Indoor SLAM using laser sensors**

There are several other sensors like Kinect which can provide similar data. A paper published in 2010 [17] suggests using a 2D laser range finder and an omnidirectional camera for indoor SLAM. The authors present a fully automatic process to do that. They calibrated the laser and camera using Matlab Omnidirectional Calibration Toolbox and used Polar Scan Matching (similar to Iterative Closest Point but less computationally expensive) for matching the different scans. The final step was to extract vertical lines from camera images and laser scans.

- **Indoor Mapping with short range sensors**

A paper published in 2010 [18] proposes a method of indoor mapping using bumper sensors and a wall sensor that enables wall-following for odometry. The method first performs trace segmentation by fitting line segments to the noisy trajectory taken by the wall sensor. Then the authors apply a probabilistic rectification process to the segmented traces to obtain the orthogonal wall outlines. The map result is a list of line segments representing the wall outlines. The experiments they have conducted show that the method is robust to odometry noise and non-rectilinear obstacles along the walls.

1.2 Purpose of this work

The purpose of this work is to present a method of mapping indoor environments and identifying the regions in which a robot can navigate. This identification is done by processing a series of consecutive depth frames taken by Kinect sensor. Kinect provides 3D data which are processed in order to produce a 2D map of the regions which represent the floor. To produce this map we use plane extraction (we calculate the rotation using the planes extracted), manually measured translation between the frames (simulating robot odometry), a region growing algorithm and contour extraction methods.

1.3 Equipment and tools

For the purpose of this work, we used Kinect for Windows. Furthermore several tools and programs were used to aid this effort. Those are: MatLab, Kinect SDK and Kinect Developer Toolkit.

1.3.1 Kinect specifications

Kinect is an RGB depth sensor, which was designed and developed by Microsoft for the video game console X-BOX 360 and for computers. It features an RGB camera, a depth sensor and an array of microphones as shown in figure 1 [1]. Microsoft has supplied the device with open source software, which can track movement and faces. Kinect contains a rotor (tilt) just above its base, which allows it to move the angle of its field of view. The range in which objects can be tracked by Kinect is from 0.8 to 3.5 meters. In order to compute depth, Kinect uses an infrared ray emitter. The infrared rays are reflected onto any object they meet in their way, and then return to the device. The depth image produced by Kinect is 640x480 pixels. Table 1 shows some more specifications [1] of Kinect for Windows sensor.

Kinect	Array Specifications
Viewing angle	43° vertical by 57° horizontal field of view
Vertical tilt range	±27°
Frame rate (depth and color stream)	30 frames per second (FPS)
Audio format	16-kHz, 24-bit mono pulse code modulation (PCM)
Audio input characteristics	A four-microphone array with 24-bit analog-to-digital converter (ADC) and Kinect-resident signal processing including acoustic echo cancellation and noise suppression
Accelerometer characteristics	A 2G/4G/8G accelerometer configured for the 2G range, with a 1° accuracy upper limit.

Table 1 – Kinect specifications

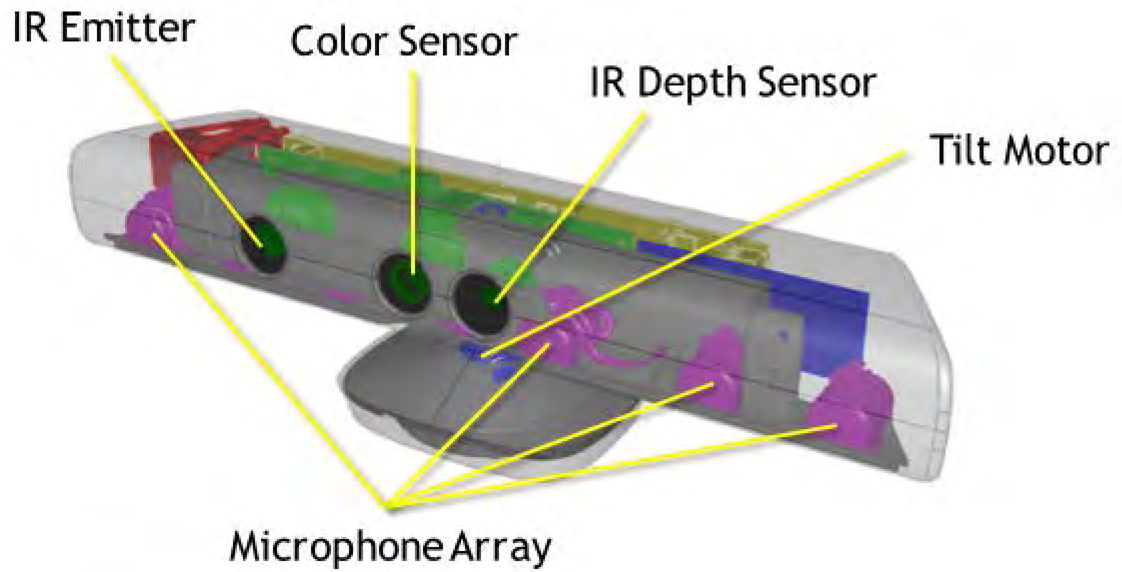

Figure 1 - Kinect features

Kinect uses its own coordinate system. The camera is considered as the start of the system (0, 0, 0). X is the horizontal axis, Y is the vertical axis and Z is the depth axis. The sensor measures the distances in front of it as negative. The coordinate system of the sensor is shown in figure 2 [2].

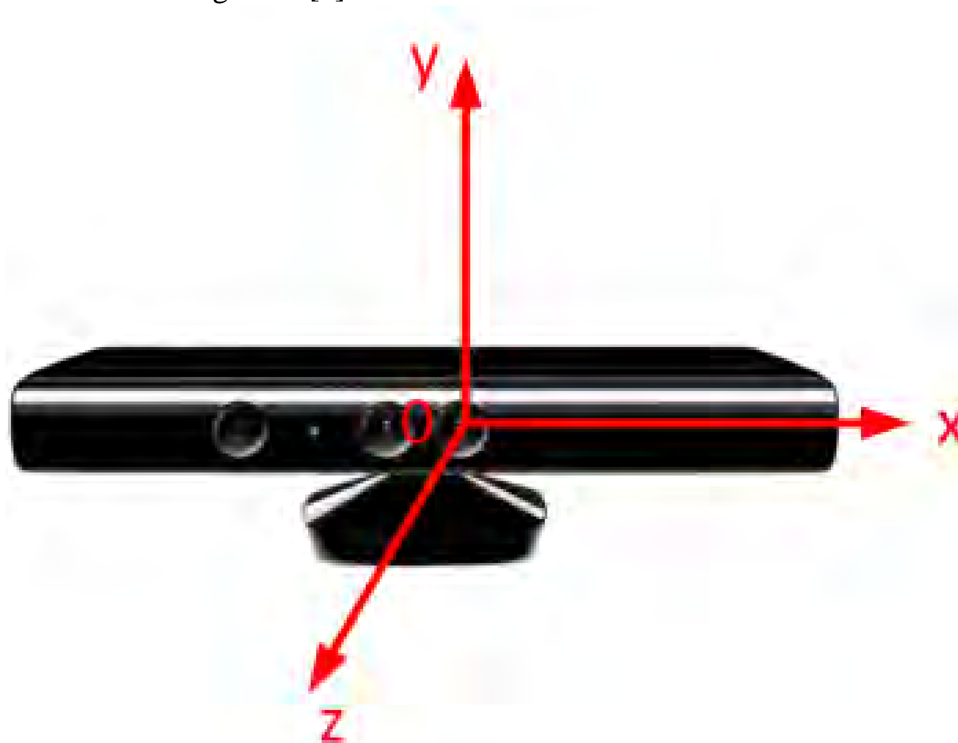

Figure 2 - Kinect Coordinates

1.3.2 Installation of Kinect

In order to install Kinect, some packages are required. First we need to install Kinect SDK, which can be downloaded through Microsoft's official site. Then we must install the Kinect Developer Toolkit in order to gain access to the documentation and various open source implementations. As soon as the installation is completed, we can plug our device into our computer and let Windows automatically install Kinect's drivers. If they fail to install the drivers, we can download them through Microsoft's official site and install them manually.

1.3.3 Kinect Fusion Explorer D2D

Kinect Fusion Explorer is an open source implementation which is provided by Kinect Developer Toolkit. This software can be used to capture frames and make a 3D reconstruction model. There is a UI which gives the user plenty of choices (figure 3).

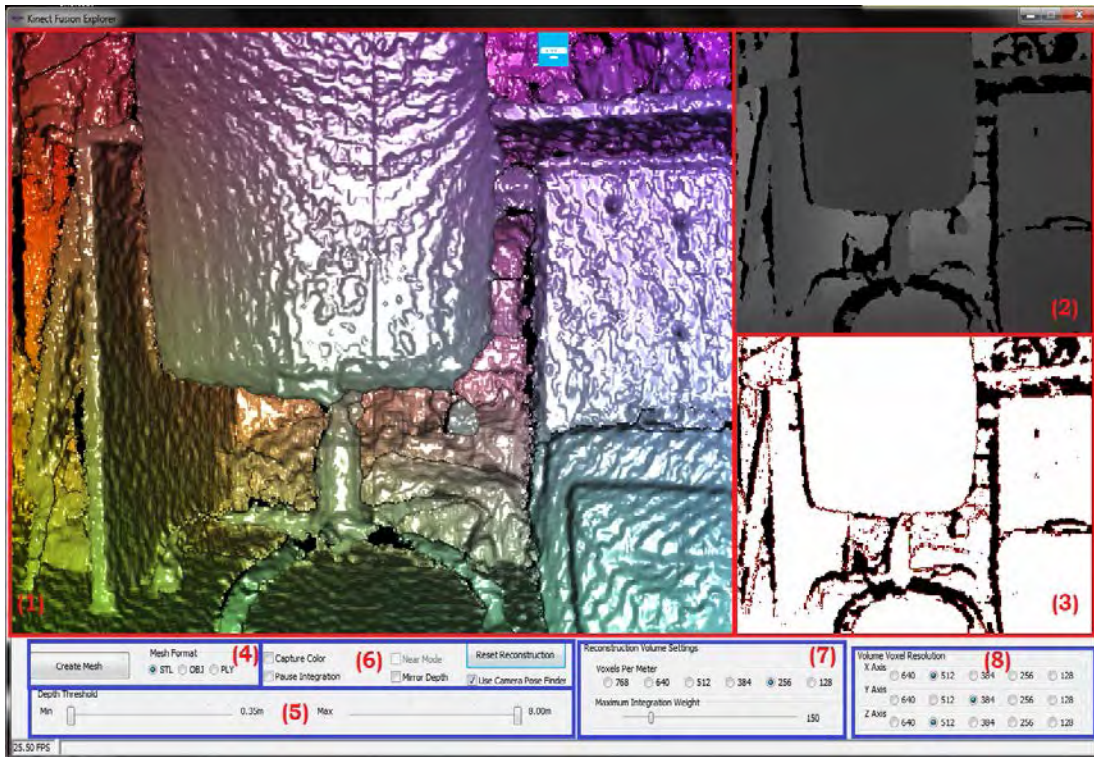

Figure 3 - Kinect Fusion Explorer UI

The UI is divided in two regions. The first region (marked as red) is the view region whereas the second region (marked as blue) is the control region.

In view region, there are 3 windows:

1. Live 3D reconstruction (marked as 1): Real time 3D triangle reconstruction of area using Kinect measurements. These measurements are being processed in order to create live feedback.
2. Depth image (marked as 2): Depth image produced by Kinect measurements. Light colors (closer to white color) represent small distances and dark colors (closer to black color) represent big distances.

3. Raw data (marked as 3): Raw point coordinates without showing depth.

In control region there are 5 sections:

1. Mesh section (marked as 4): Provides the button *Create a Mesh*, which saves the frames captured by Kinect, into a file. Three options are available for the file format:
   - STL (STereoLithography)
   - OBJ (Object File)
   - PLY (Polygon File Format)

2. Depth threshold section (marked as 5): Provides options for setting minimum and maximum distance the Kinect can see (0.35 to 8 meters).

3. Capture Section (marked as 6): Provides options for color capture, mirror capture, use of camera pose finder and pause capture (pause integration). Pause capture is only available when camera pose finder is disabled. This section also provides a button for resetting the reconstruction.

4. Reconstruction volume settings section (marked as 7): Provides options for setting the number of voxels per meter as well as the maximum weight they have on the reconstruction process.

5. Volume voxel resolution section (marked as 8): Provides options for setting the resolution of a frame for every axis.

1.3.4 The PLY File Format

PLY (Polygon File Format) is a computer format also known as *Stanford Triangle Format*. It is designed for saving 3D data coming out of 3D scanners (e.g. Kinect). As soon as Kinect is plugged into our computer, we can run *Kinect Fusion Explorer D2D* in order to capture a frame. We choose *PLY format* from mesh section and leave everything else in the default options. We press the *Create Mesh* button and save the file. PLY contains data for a point cloud, as well as the polygons formed by them.

A PLY file always starts with the keyword:

```
Ply
```

The keyword PLY is followed by three lines, which gives us information on the data encryption (binary, ASCII):

```
format ascii 1.0
format binary_little_endian 1.0
format binary_big_endian 1.0
```

The following lines show how many vertices are included in the file and how they are described (x, y, z axis):

```
element vertex 12
property float x
property float y
property float z
```

Similar lines follow in order to show how many faces are included in the file:

```
element face 10
property list uchar int vertex_indices
```

End of header follows:

```
end_header
```

The lines that follow are the actual data: point coordinates (vertices) in 3-column ascii format and the indices of the vertices that form the triangles (3 indices per line for each triangle).

2 Methodology

The goal of this work is to create a map which shows the regions in which a robot can navigate. This map will be produced by processing a series of frames taken by Kinect sensor. The Kinect sensor is manually moved. To read these data from the PLY files taken by Kinect in order to process it in Matlab, we use a method [3] we found on Mathworks. Figure 4 shows the process used in order to produce this map. First we need to reduce the number of points and faces included in a frame, which will lead to better performance. The next step is to use a variation of region growing algorithm in order to mark the region that represents the floor. After this process, we will find the contour of this region. Having already measured the ground truth contour manually we will compare it with the output contour and calculate the mean error in centimeters.

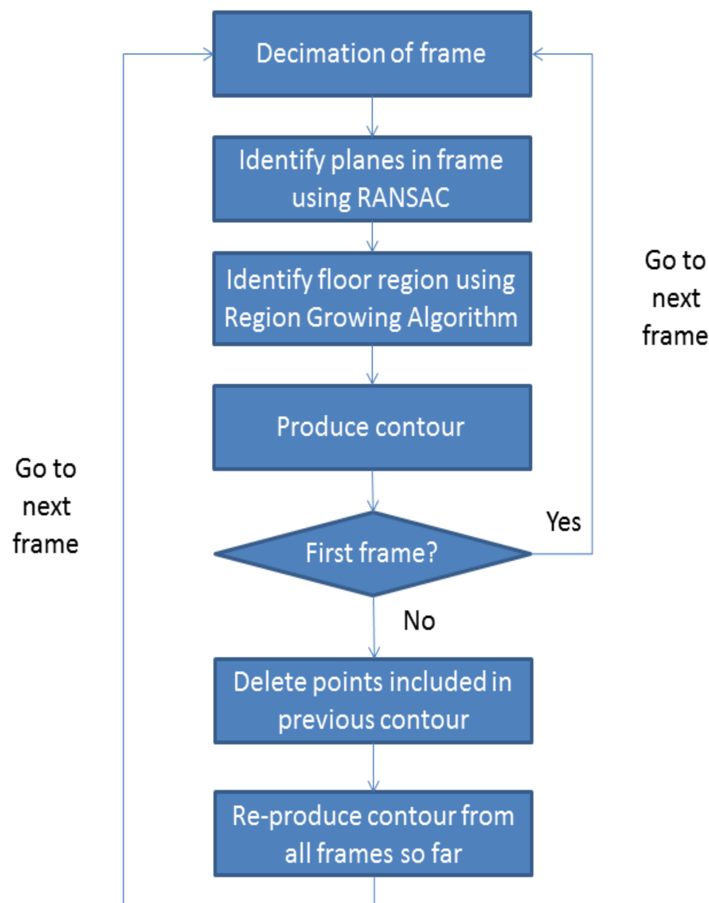

Figure 4 - Methodology

2.1 Mathematical operators

The cross product between two vectors is a vector which is normal to the plane defined by those two vectors. The symbol used to describe this operation is:

$$\vec{\boldsymbol{a}} \times \vec{\boldsymbol{b}}$$

When it comes to triangles, the cross of the vectors of two of its edges is the normal vector of the triangle itself. The vector of a triangle's edge is computed by subtracting the two points forming the edge. Given a triangle consisted of the points:

$$v1 = (x1, y1, z1)$$
$$v2 = (x2, y2, z2)$$
$$v3 = (x3, y3, z3)$$

The vectors of its edges are:

$$\overrightarrow{\boldsymbol{e1}} = (x2 - x1, y2 - y1, z2 - z1)$$
$$\overrightarrow{\boldsymbol{e2}} = (x3 - x2, y3 - y2, z3 - z2)$$
$$\overrightarrow{\boldsymbol{e3}} = (x3 - x1, y3 - y1, z3 - z1)$$

The normal vector of the triangle is:

$$\vec{\boldsymbol{V}} = \overrightarrow{\boldsymbol{e1}} \times \overrightarrow{\boldsymbol{e2}}$$

To compute the angle between two vectors $\vec{\boldsymbol{a}}, \vec{\boldsymbol{b}}$, thus to determine if they can be considered as parallel we calculate the dot product between the two normalized vectors:

$$\vec{\boldsymbol{a}} \cdot \vec{\boldsymbol{b}} / |\vec{\boldsymbol{a}}|.|\vec{\boldsymbol{b}}|$$

We will use this product later in region growing algorithm to check if this value is higher than a predetermined threshold.

2.2 Decimation of point set

The frame captured by Kinect consists of a set of points and a set of the faces formed by those points. Due to the size and complexity of data, triangle decimation is required. Decimation algorithms are used very often in computer graphics-related operations, since they drastically reduce the computational burdin of these operations. Several criteria are used for selecting which triangles to eliminate, as described in [35] and elsewhere. In our application, we apply decimation to significantly reduce the number of faces and points of a given PLY file (generated for each Kinect frame) and make data processing faster and easier.

MatLab has a built-in function called *reducepatch*. This function's parameters are the set of points, the set of faces formed by those points and the number of faces needed after the decimation. Then *reducepatch* reduces the number of faces, while attempting to preserve the overall shape of the original scene/object. After finishing that process, *reducepatch* returns a structure containing the reduced points and faces.

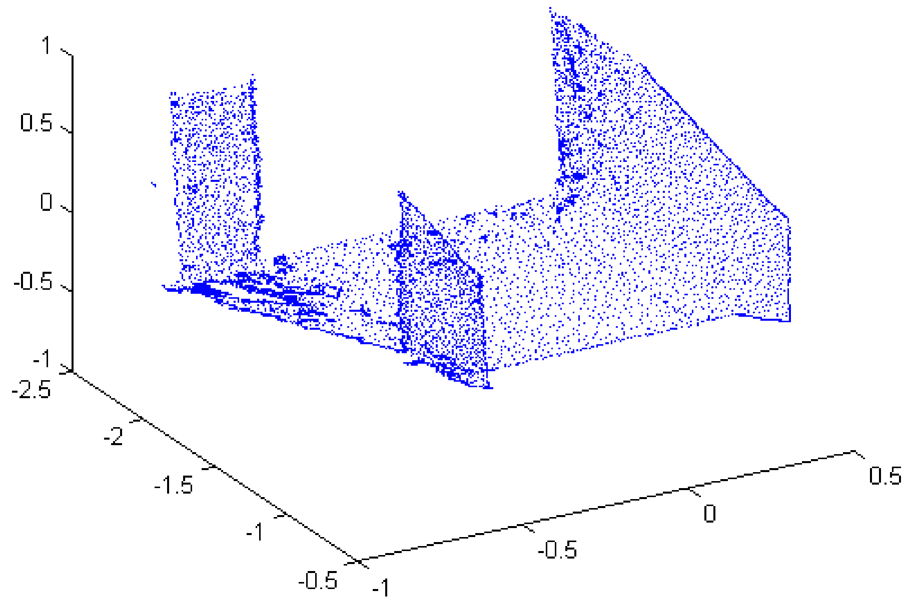

Figure 5 - Pre-decimation points

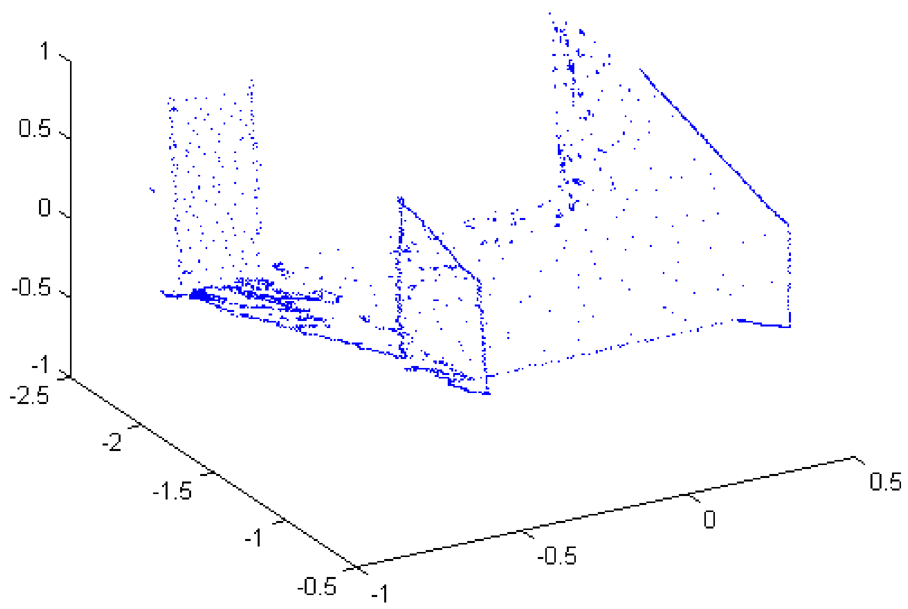

Figure 6 - Post-decimation points

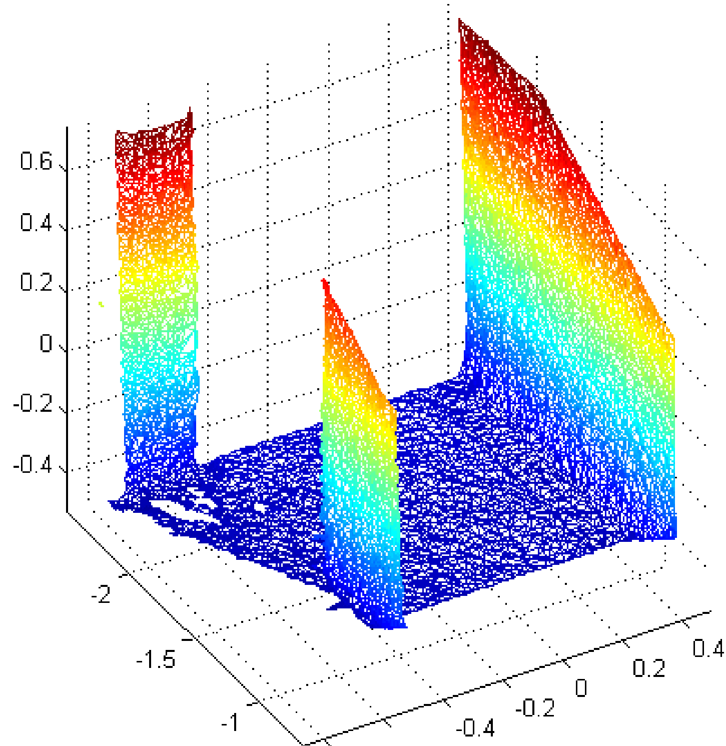

Figure 7 - Pre-decimation faces

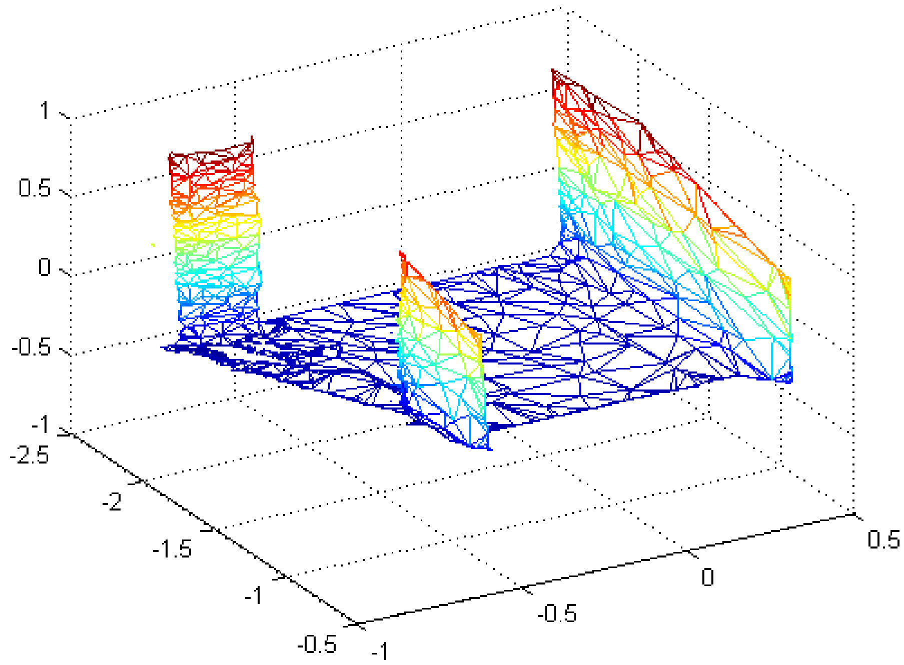

Figure 8 - Post-decimation faces

The figures above show the original frame (figure 5 for points and figure 7 for faces) of a corridor and the results after decimation (figure 6 for points and figure 8 for faces). The original frame had about 12000 points and 15000 faces. Using the *reducepatch* function the number of points was reduced to about 2000 and the number

of faces to 3000. As we can see in figures 2.b and 2.d the overall shape of the corridor was well preserved. The function can also work efficiently in cases of much bigger sets of points and faces (even millions).

2.3 Identifying planar surfaces in the captured frame

In order to identify the planar surfaces in a captured frame we must first extract all the planes by using RANSAC algorithm. RANSAC (Random Sample Consensus) [12] is an iterative method to estimate parameters of a mathematical model from a set of points. It is a non-deterministic algorithm in the sense that it produces a reasonable result only with a certain probability, which can be increased if the number of iterations is increased. In this case it can be used for estimating the parameters of all the planes on a points set. Those parameters are *A, B, C*, and *D*. Given the points (x, y, z) forming the plane, the equation of the plane is shown below:

$$A \cdot x + B \cdot y + C \cdot z + D = 0$$

2.3.1 Identifying one best fitting plane

The pseudo-code of RANSAC algorithm is shown below. *Consensus set* is the set of points fitting a plane and data set the set of all the points in the scene.

<u>Step 1:</u>
Select three random points and compute the parameters (*A, B, C* and *D*) of the best fitting plane.

<u>Step 2:</u>
Calculate the distance of every point from the best fitting plane. If the distance is smaller than a predetermined threshold, add the point to the *consensus set*. The distance [13] is calculated by the equation:

$$distance = \frac{A \cdot x + B \cdot y + C \cdot z + D}{\sqrt{A^2 + B^2 + C^2}}$$

<u>Step 3:</u>
If the number of points in *consensus set* is considerably high, the plane is considered good.

<u>Step 4:</u>
If plane is good, the plane parameters are calculated again based on the points in the consensus set.

<u>Step 5:</u>
Repeat steps 2, 3 and 4 until consensus set length stays the same.

2.3.2 Identifying multiple planes

The process described above can be used for every plane in the scene. To do so, some further steps must be followed:

<u>Step 6:</u>
Remove **consensus set** points from data set.

<u>Step 7:</u>
Call RANSAC algorithm (2.3.1) until all the points fit a plane or the number of iteration exceeds a predefined value.

The parameters *A, B, C* and *D* are normalized in order to give better planes. To normalize the parameters we need to compute the plane's normal vector value as shown below and then divide parameters *A, B, C* and *D* with it. If we define **V** = [*A, B, C*] and **S** = [*A, B, C, D*], then:

$$|\mathbf{V}| = \sqrt{A^2 + B^2 + C^2}$$

$$\mathbf{S}_{\mathbf{normal}} = \frac{\mathbf{S}}{|\mathbf{V}|}$$

Except the parameters of each plane RANSAC can also give a random point (x, y, z) which is part of the plane and therefore it can be used as a seed point for the Region Growing algorithm. With RANSAC the process can be used not only for mapping the floor, but instead for mapping the whole scene.

This whole process identifies all the best fitting planes in a scene. To determine which one of them represents each wall on the scene, we can check the normal vector of the planes. The normal vector's orientation shows the rotation of the plane in the scene.

We ran RANSAC on a test set of points forming 2 vertical planes. The results are show in figure 8.a.

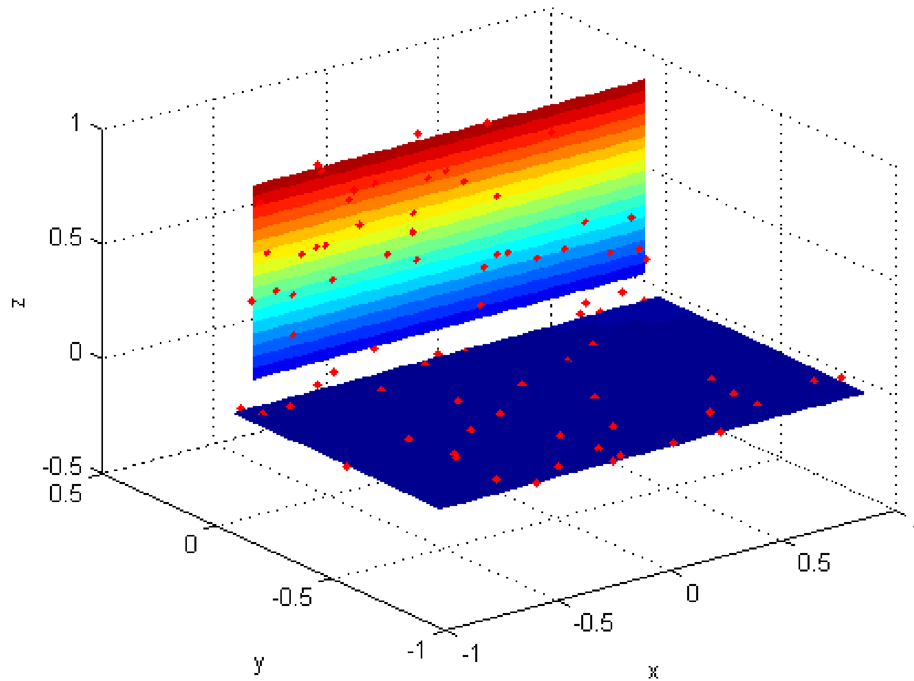

Figure 8.a – Points (red) and planes identified by RANSAC

2.4 Region Growing Algorithm

Region Growing [11] is a simple region-based image segmentation method which is very similar to clustering algorithms. The algorithm examines neighboring points of initial seed point(s) and determines whether these neighbors should be added to the

same region as the initial point(s). More specifically it uses a queue which contains the indexes of the points. In each loop the algorithm extracts the head of the queue, identifies its neighbors and determines whether it meets the criteria specified by user. The efficiency of this algorithm is depended on the correct selection of the initial seed point(s).

Advantages :

1. Can correctly separate the regions that have the same properties we define.
2. Can provide the original images which have clear edges with good segmentation results.
3. It is simple, as it only needs a small number of seed points to represent the property we want.
4. It is flexible. It works well with multiple criteria at the same time.

Disadvantages :

1. Computationally expensive
2. Sensitive to noise.

Region Growing Algorithm variation

For the purpose of this work, a variation of Region Growing algorithm was used. The purpose of this variation is to make Region Growing algorithm work with triangles. This variation gets a seed triangle, identifies its neighbor triangles and then determines whether to put them in the same region as the seed triangle. The criteria based on which the algorithm determine that, is whether the vertical vector of the triangle is parallel to the mean vector of the triangles which already are in the region. To fully enlighten the process a pseudo-algorithm is given below:

Step 1:

Get seed triangle by user. Compute normal vector of the seed triangle, set it as mean and insert the triangle in queue *Q (neighbors queue).*

Step 2:

Extract the first triangle from queue *Q* and insert it in queue *area (region's queue).*

Step 3:

Identify the triangle's neighbors.

Step 4:

For each neighbor:

Step 4.a:

Compute normal vector.

Step 4.b:

If normal vector is parallel or quite parallel to the mean normal vector, insert it in queues *area* and *Q*.

Step 4.c:

Re-compute mean vector by taking into account every triangle in queue *area*.

Step 5:

Repeat steps 2, 3 and 4 until queue ***Q*** is empty.

Step-by-step example:

To better understand this example, here is some clarification of the colors used in the figures below:

Green: Triangles which are in *region's queue (area)* and whose neighbors the algorithm is about to identify.

Gray: Triangles which are neighbors of the triangle investigated at that time (the extracted head of the queue *Q*), but not yet compared to the mean normal vector of the *region's queue (area)* triangles.
Yellow: Triangles which are neighbors of the triangle investigated at that time (the extracted head of the queue *Q*) and also pass the criteria to be considered part of the region. Their neighbors are not yet identified.

The algorithm starts from a seed triangle which is given either by user (by hand) or by RANSAC algorithm (random point of the plane found by RANSAC which is part of a triangle). This triangle is put into the *region's queue (area)* and marked green as shown in figure 9. At this time the queues status is:

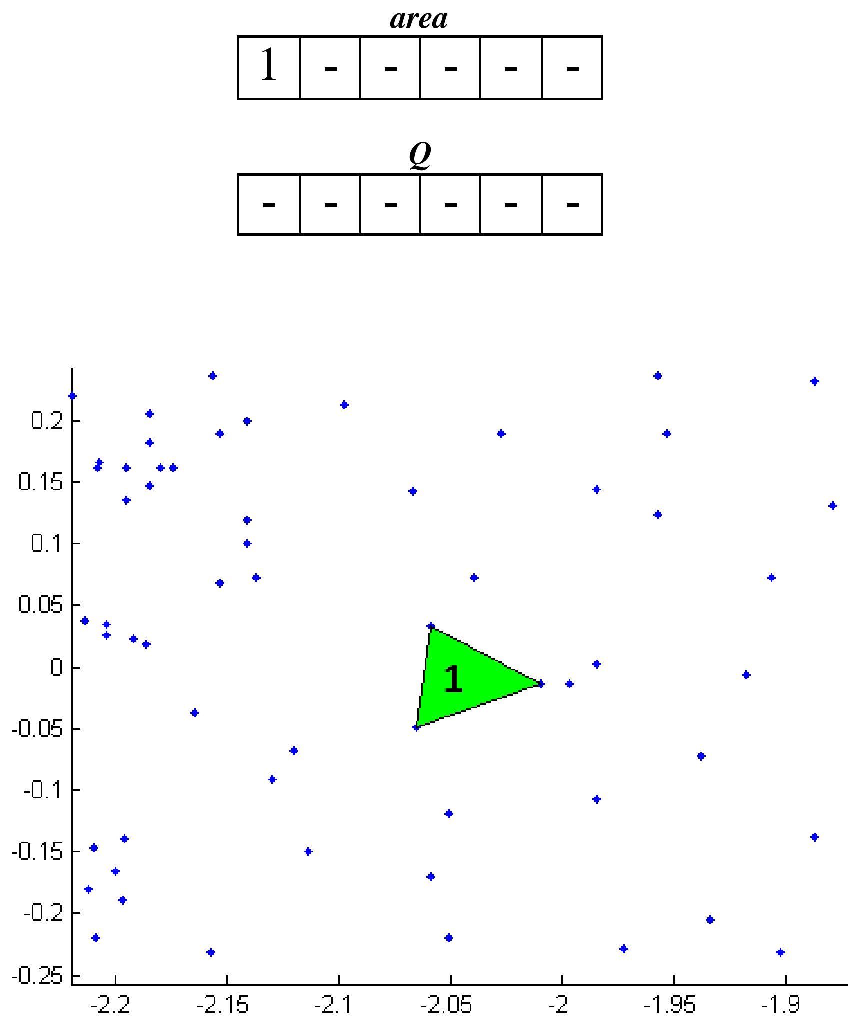

Figure 9

The algorithm will now start identifying this triangle's neighbors one by one. In figure 10 one of its neighbors is marked gray.

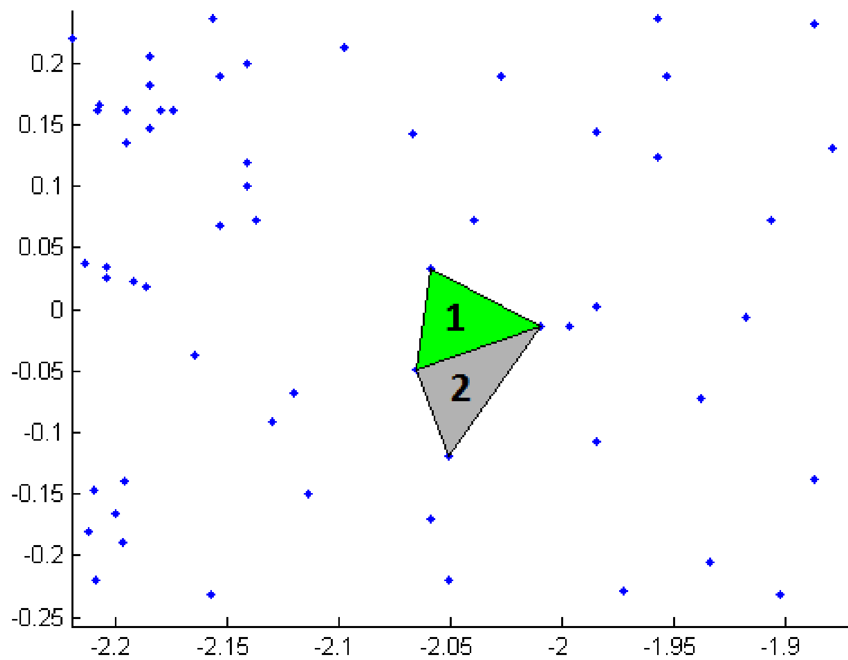

Figure 10

Triangle's (2) normal is quite parallel to the mean normal vector of all the triangles marked green, so it is marked yellow in figure 2.g. If the triangle's normal vector was not acceptably parallel to the mean of the green triangles normal vectors it would remain gray. The triangle (2) is now put to the *neighbors queue (Q)***.** The updated status of the queues is:

area

1	-	-	-	-	-

Q

2	-	-	-	-	-

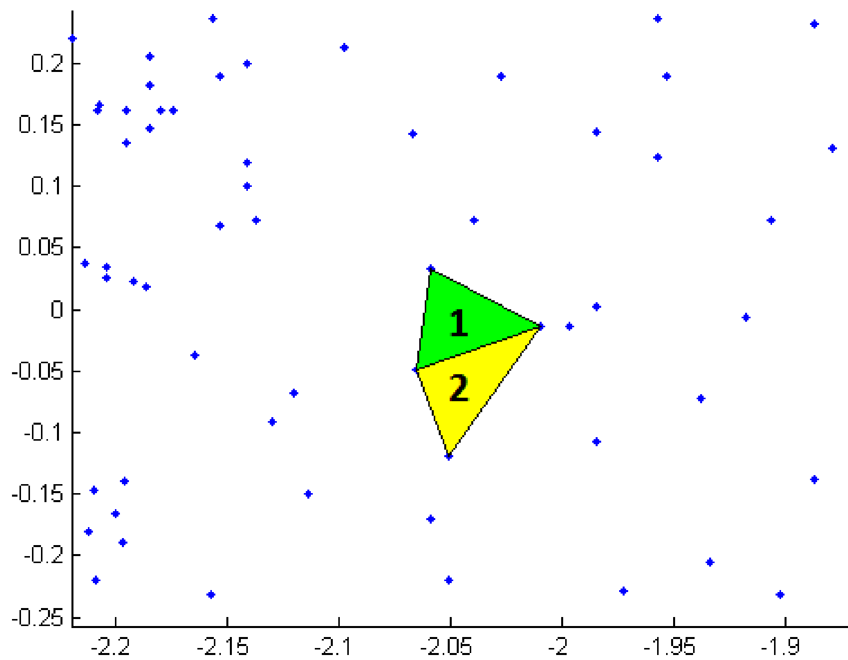

Figure 11

The same process is used for the other neighbors of the triangle (1). The algorithm identifies the neighbor triangle 3 (figure 11) and then puts it in *neighbors queue (Q)* as it passes the pre-defined criteria (figure 12). After these steps the status of the queues is:

area

1	-	-	-	-	-

Q

2	3	-	-	-	-

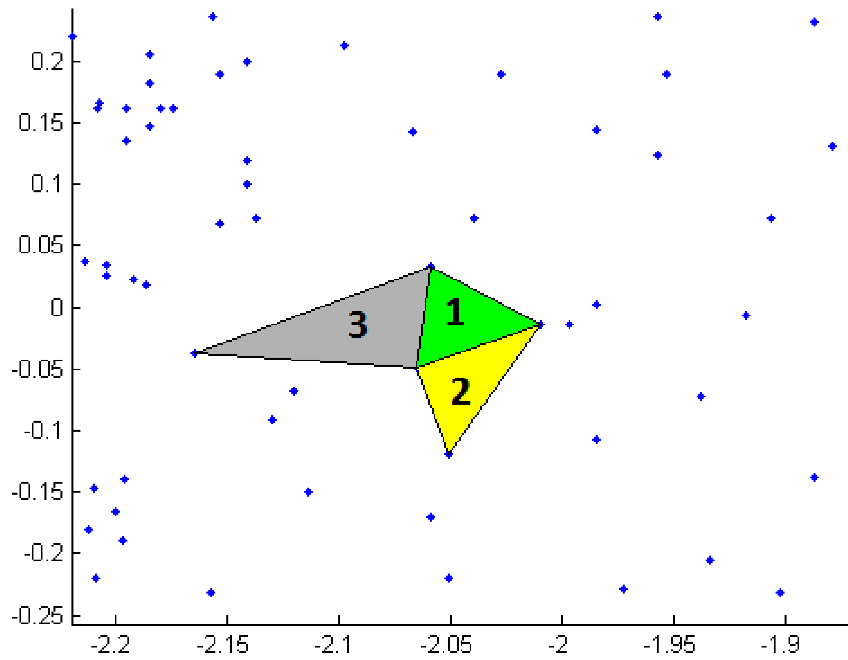

Figure 12

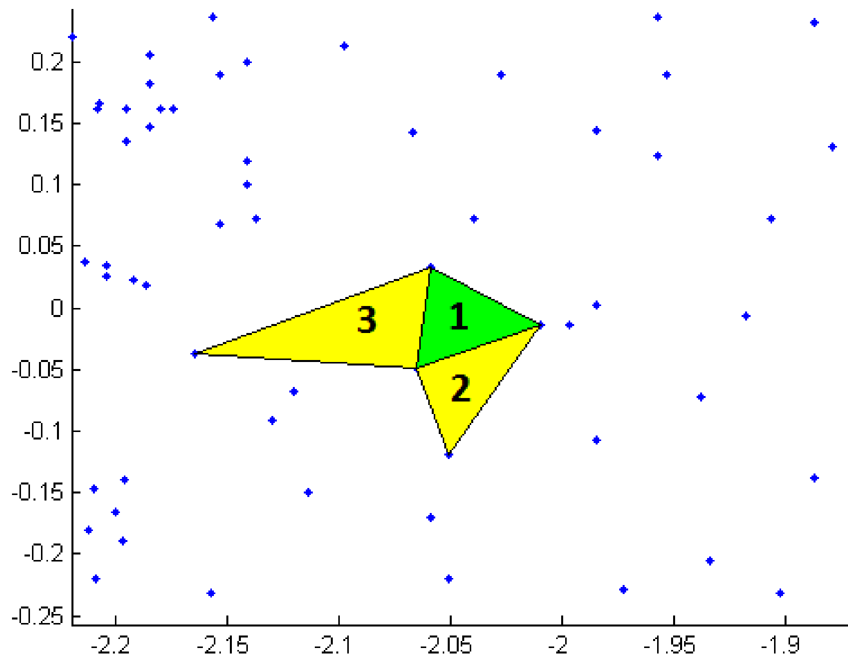

Figure 13

The algorithm identifies the last neighbor of triangle 1 using the same process (figure 2.j) and then puts it in the *neighbors queue (Q)* (figure 13). The updated status of the queues is:

area

1	-	-	-	-	-

Q

2	3	4	-	-	-

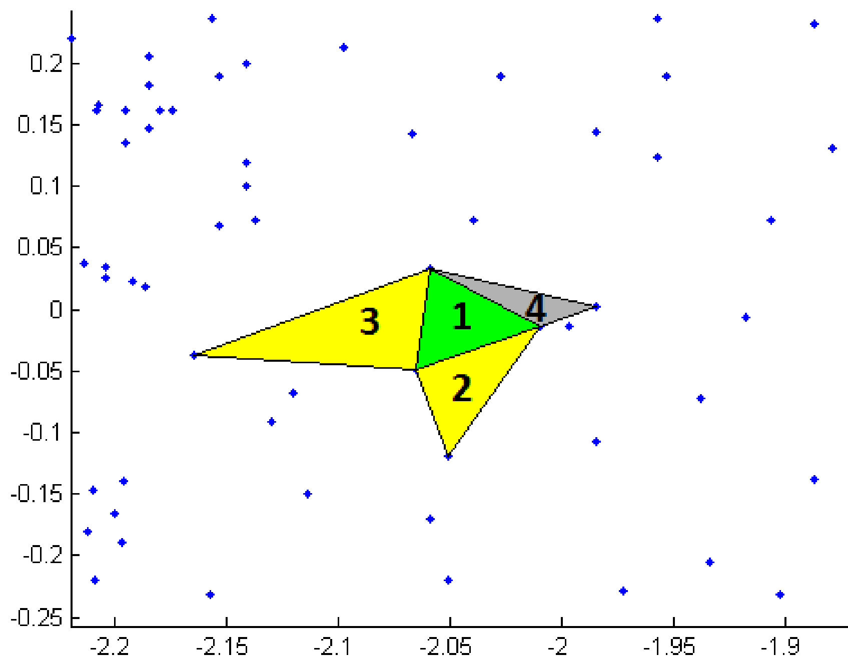

Figure 14

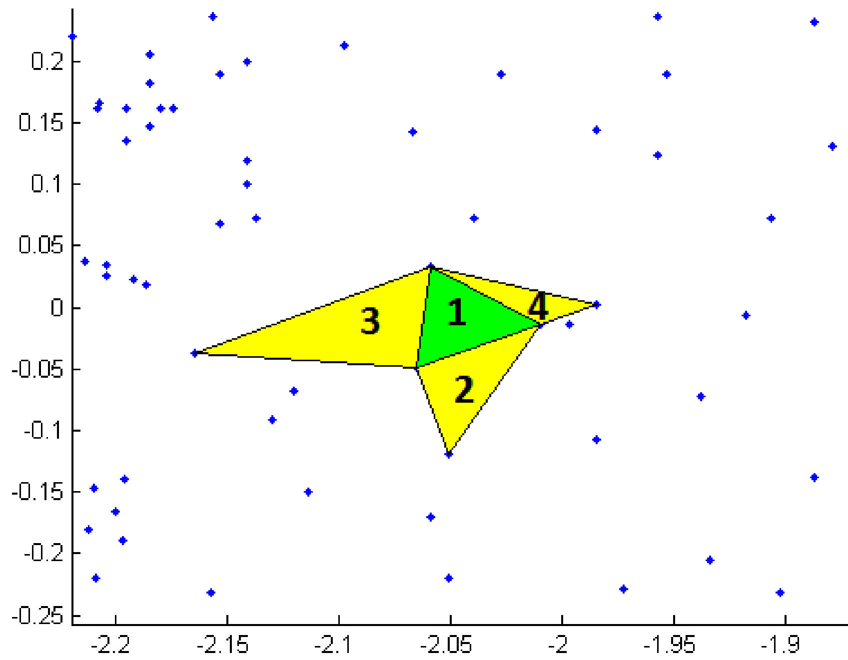

Figure 15

At this point the algorithm has identified all neighbors of triangle 1 and determined whether they are part of the region. Now the algorithm extracts the head triangle of the *neighbors queue (Q)* marks it green and puts it in the *region's queue (area)***.** So at this point the queues status is:

area

1	2	-	-	-	-

Q

3	4	-	-	-	-

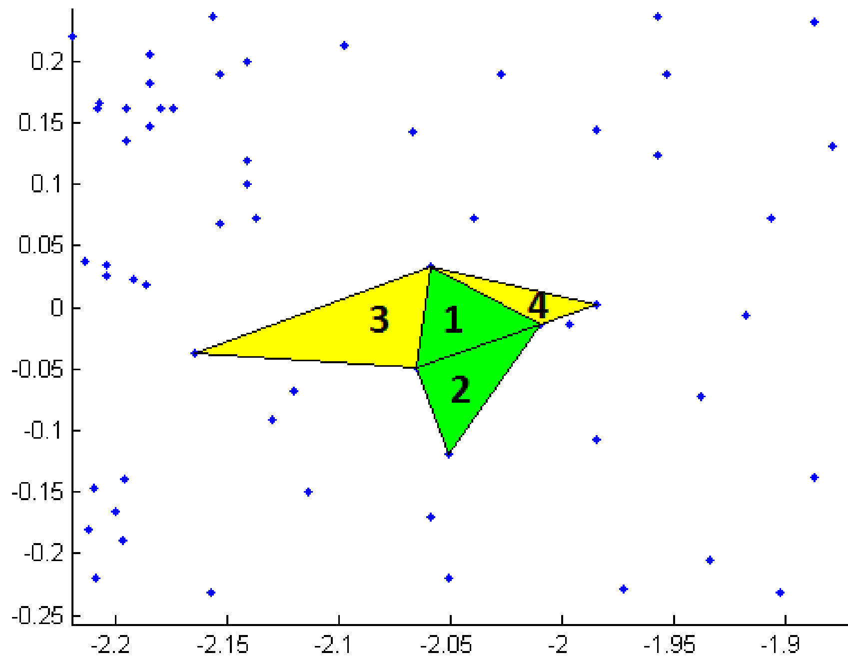

Figure 16

This whole process is repeated until there are no triangles left in *neighbors queue (Q)*.

2.4.1 Automating region growing algorithm

Except for the parameters of a plane, RANSAC also identifies the subset of Kinect frame points fitting in the plane, according to a tolerance. From that subset we can select a single point to play the role of the seed point for the Region Growing Algorithm. This point should be located near in the center of mass of the points than lay on the plane in order to avoid problems caused by the points located near the outline of the plane.

Given that $A \cdot x + B \cdot y + C \cdot z + D = 0$ defines a plane, x_{mid}, y_{mid} and z_{mid} are the mean coordinates of the points belonging to that plane (with a tolerance) and *value* is a predetermined number (in cm), the selected point (x_0, y_0, z_0) must meet the criteria below:

Case 1:
The plane is defined mostly by axis X and axis Y (equivalently, $C > A \text{ and } C > B$):

$$x_0 > x_{mid} - value \text{ and } x_0 < x_{mid} + value$$
$$and$$
$$y_0 > y_{mid} - value \text{ and } y_0 < y_{mid} + value$$

Case 2:
The plane is defined mostly by axis X and axis Z (equivalently, $B > A \text{ and } B > C$):

$$x_0 > x_{mid} - value \text{ and } x_0 < x_{mid} + value$$
$$and$$
$$z_0 > z_{mid} - value \text{ and } z_0 < z_{mid} + value$$

Case 3:
The plane is defined mostly by axis Y and axis Z (equivalently, $A > B \text{ and } A > C$):

$$y_0 > y_{mid} - value \text{ and } y < y_{mid} + value$$
$$and$$
$$z_0 > z_{mid} - value \text{ and } z_0 < z_{mid} + value$$

We run the automated region growing algorithm in the same test set used in RANSAC (2.3.2) and the results are shown in figure 16.a.

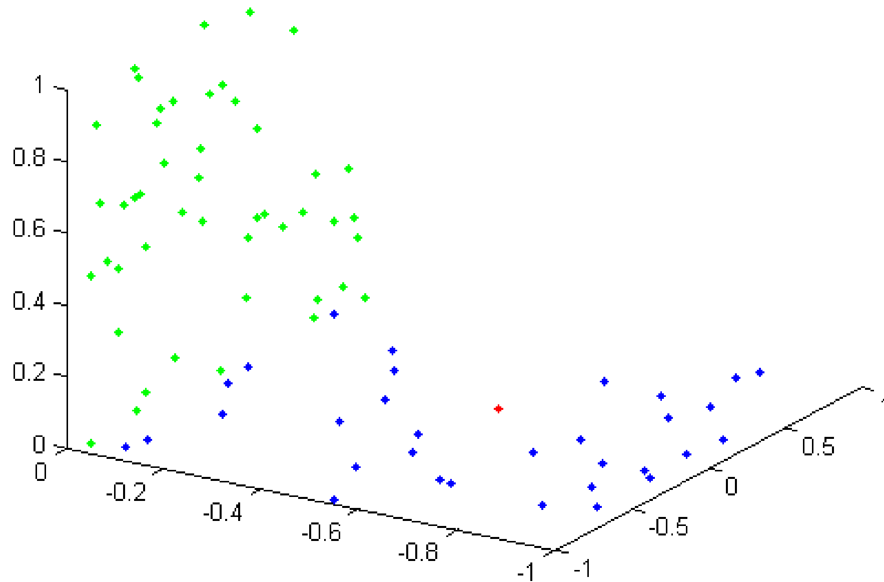

Figure 16.a – Points of the floor (blue), seed point given by RANSAC (red) and points of the wall (green)

2.5 Merging world points from different frames

After performing region growing algorithm in every frame, we need to merge the frames in order to complete the mapping of the floor. To do this we need to know how much the Kinect sensor has moved between two frames. We can obtain this information manually (simulating robot odometry) or by the information extracted by planes (given by RANSAC). To better understand this information we will not use the coordinates system used by Kinect. Instead we will use the common coordinate system. The camera of the sensor is considered to be the center of the coordinates system (point 0, 0, 0). The sensor is moved only forward (-X axis), backwards (X axis), to the left (-Y axis) and to the right (Y axis). The sensor can also rotate left or

right only around Z axis. We define our first frame to be the coordinate system and then transform the following frames to match with it.

As soon as we receive the translation and/or rotation of the sensor, using manually measurements and plane information, we can transform the points of each frame to make them match the first frame. The types of point transformation are described below:

Type 1: Translation

The Kinect sensor can move forward/backward/left/right. Given that $\mathbf{T} = [T_x, T_y, 1]$ is the movement matrix (Tx represents movement in cm on X axis, Ty represents movement in cm on Y axis and on Z axis there is no movement), the transformation of the points is given by the equation below:

$$\text{points}_{\text{transformed}} = \text{points}_{\text{original}} + \mathbf{T}$$

Type 2: Rotation

We assume that the Kinect sensor can freely rotate around Z axis. Given that θ is the angle of the sensor rotation, the rotation matrix R is calculated as shown below:

$$\begin{matrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{matrix}$$

The transformation of the points can be either one of the two types or a combination of them. If it is a combination of movement and rotation, the transformation is given by the equation below:

$$\text{points}_{\text{transformed}} = (\text{R} \cdot \text{points}_{\text{original}}) + \mathbf{T}$$

Using the planes which RANSAC gives as output we can estimate the rotation between two consecutive frames. Knowing the planes which represent the same surface before and after the rotation we can estimate the rotation angle as shown below.

Given that $[A_1, B_1, C_1, D_1]$ represent the surface (plane) in the frame before the rotation and $[A_2, B_2, C_2, D_2]$ represent the same surface (plane) in the frame after the rotation, the formula for estimating the rotation angle is:

$$\text{angle} = \cos^{-1}([A_1, B_1, C_1] \cdot [A_2, B_2, C_2])$$

Note that '·' in this case is the scalar product between two vectors.

2.6 Contour extraction

Now that we have merged the frames, we have a set of points describing the floor-like region. The next step is to find the contour of this region. Although the identification of the convex hull of a set of points is a well-studied problem with efficient solutions [36], finding the non-convex hull is a problem hard to define

eg.[37]. Our approach is a practical and efficient solution of extracting the contour after embedding the real world points of the contour into a binary discrete image. Thus the proposed algorithm is parameterized by the size of the pixel used in converting the set of real world points into a binary image. Then we will find the pixels forming the contour of the binary image and convert them again to coordinates.

- **World coordinates to pixels**

To convert the coordinates of points to binary image, we will use a simple algorithm. At the end of this process the region will be white (1) and the background will be black (0). Given that *pixel size* is a predetermined value in cm, the size of this binary image is computed based on the given pixel size:

$$\text{number of rows} = \frac{|max(y) - min(y)|}{(pixel\ size)}$$

$$\text{number of columns} = \frac{|ma\,x(x) - min(x)|}{(pixel\ size)}$$

A simple linear algorithm:

<u>Step 1</u>

For every pixel ***(i, j)*** in binary image where ***i*** represents rows and ***j*** represents columns and *pixel size* is a predetermined value, compute the x and y coordinates forming the pixel:

$$x0 = min(x) + (j - 1) \cdot (pixel\ size)$$
$$x1 = x0 + (pixel\ size)$$

$$y0 = max(y) - (i - 1) \cdot (pixel\ size)$$
$$y1 = y0 + (pixel\ size)$$

<u>Step 2:</u>

If there are one or more points which are between the pixels coordinates, then that pixel in the binary image is marked as white (1).

<u>Step 3:</u>

Repeat steps 1 and 2.

- **Extraction of binary image contour**

In a binary image it is much easier to find the contour of a region. Several algorithms exist for extracting the contour of a connected component in a binary image [34], such as the Pavlidis algorithm. MatLab includes a built in function called *bwtraceboundary* which implements the track the contour of region / object in binary image. The algorithm starts from a pixel specified by the user and then follows its neighbors in order to complete the contour. The starting pixel must be part of the

contour of the region (white (1)). To consider a pixel as contour at least 3 of its neighbors must not be part of the region (black (0)). Connectivity can be parameterized by the user, thus the algorithm checks for 4 or 8 neighbors in each current pixel. User can also specify which neighbor will be the first to be checked by the function as well as the direction the function will follow which can be clockwise or counterclockwise.

As soon as *bwtraceboundary* has found the contour it returns a matrix containing the rows and columns of the pixels forming the contour.

- **World pixels to coordinates**

In order to find the coordinates of the pixels forming the contour of the binary image, we will use a reverse algorithm similar to the one we used for converting the coordinates to a binary image.

The coordinates of the pixels forming the contour can be computed using the pixel size which was used to convert them to binary. Given that i represents the rows and j represents the columns, the coordinates are computed using the equations below:

$$x = min(x) + (\, pixel\ size) \cdot j$$

$$y = max(y) - (pixel\ size) \cdot i$$

2.7 Quantification of error

As soon as we have found the point forming the contour we need to check if the result is close enough to the original contour. An easy way to compare the output contour with the true contour is to employ the Distance Transform (DT) of the ground-truth contour (which is embedded in a discrete binary image, as described above). Several efficient algorithms exist for computing the DT of a binary image [38]. We will use the built-in function *bwdist* scans a binary image and for every pixel computes the Euclidean distance between that pixel and the nearest non-zero (white) pixel of the binary image.

We use the function *bwdist* on the true contour binary image. The contour binary image will be created with a predetermined small pixel size. The binary image of the contour may differ in size with the output binary image because we choose a relatively smaller pixel size to catch all the details of the ground truth contour. As a result the ground binary contour image will be always equal to or bigger than the output binary image. To compare the two images we need to transform the second image. This transformation can be done with Matlab's built-in function *imresize*, which takes as parameters a binary image, the desired size of the image after transformation (rows and columns) and the name of the method to use for resizing the image. Then for every pixel of the output contour binary image we check its value in the true contour image. To find the error of the output contour we sum those values and then find their mean value. Given that *pixel size* is a predetermined value in cm we use the following formula to find the error in cm:

$$error_{cm} = error_{pixels} \cdot (pixel\ size)$$

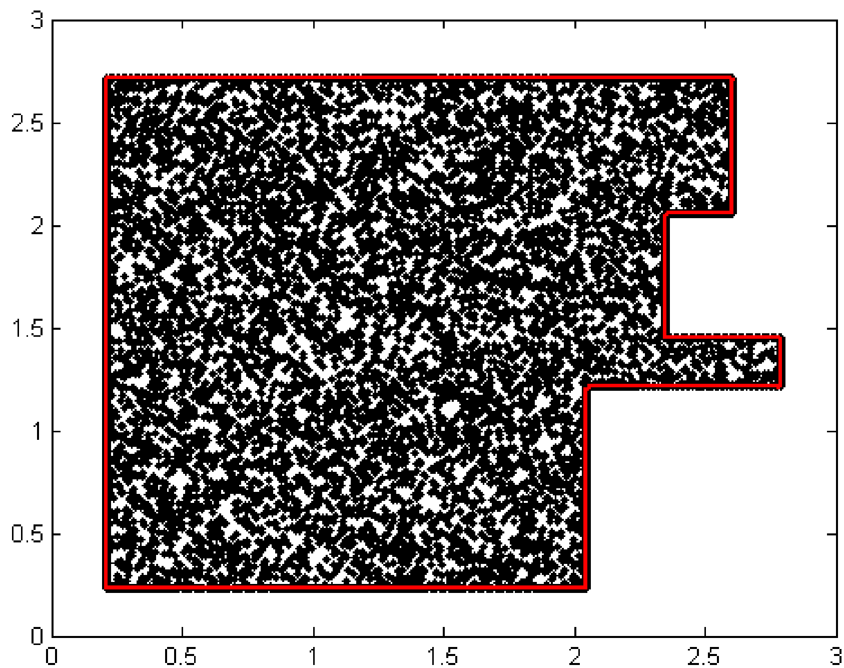

Figure 17 - Point set and ground truth contour

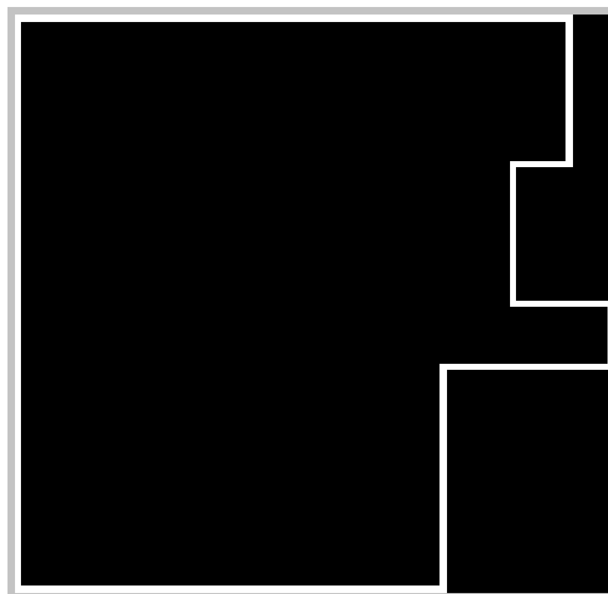

Figure 18 - Binary ground truth contour

Given the set of points shown at figure 17 (figure 18 binary contour), we will use this process to show how the algorithm corresponds to different pixel sizes. The ground binary contour will be calculated using a 3cm pixel size.

We calculated the contour using 5cm and 20cm pixel sizes (figure 19 and figure 20) and as we can see the accuracy of the algorithm is decreased if we increase the pixel size. Furthermore the mean error is increased. Setting the pixel size to 5cm gives

a contour with 0.66cm mean error whereas using a 20cm pixel gives a contour with 4.34cm mean error.

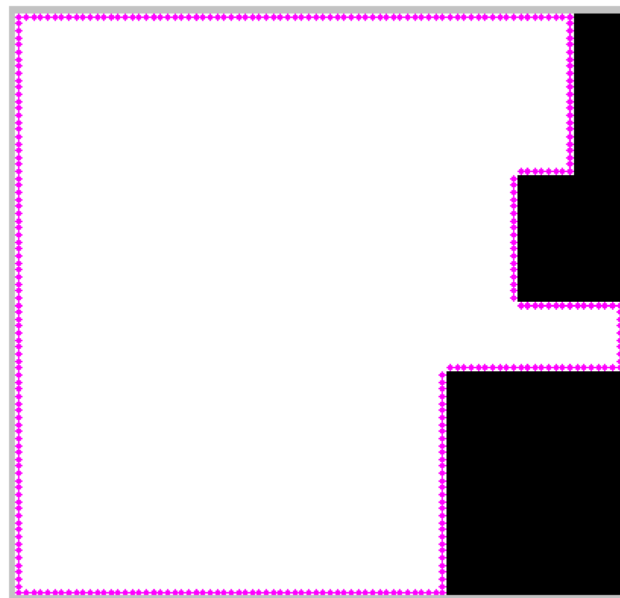

Figure 19 - Binary contour (5cm)

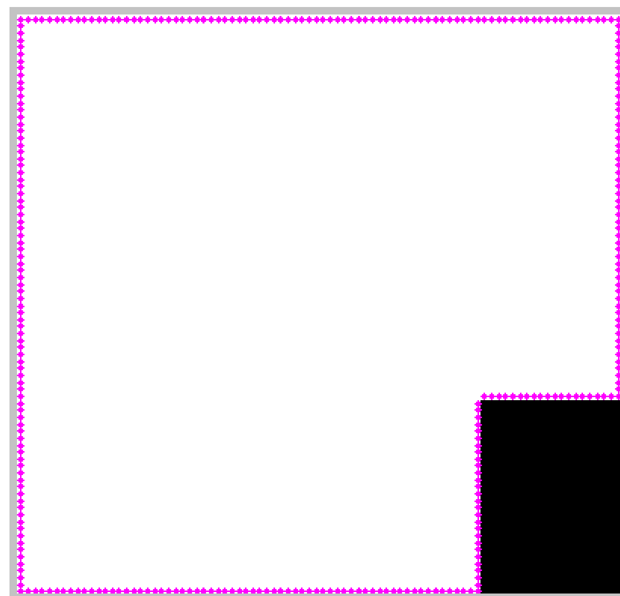

Figure 20 - Binary contour (20cm)

2.8 Removal of common points between frames

Although there is a translation and/or rotation between the points of two consecutive frames, some points are inevitably common in both frames. Taking them into account twice would be both more computationally expensive and also not a very logic experiment. Because of that we must remove those points from one of the two frames (the second). To do so we will use Matlab's built-in function *inpolygon*. This function takes as parameters a set of points which form a polygon and a set of points for testing and returns a matrix (same size with the test point set) containing 0 and 1.

The value 0 represents that the points in the specific index are not part of the polygon and the value 1 represents the opposite. The simple steps of this process are shown below:

<u>Step 1:</u>
Calculate the contour of all the previous frames so far.
<u>Step 2:</u>
Use *inpolygon* function to identify which points of the following frame are included in the contour calculated in step 1 and remove those points
<u>Step 3:</u>
Re-calculate the contour using all previous frames and the following frame.
<u>Step 4:</u>
Repeat steps 1,2 and 3 until all frames are processed.

3 Experiments and results

Using the methodology we described in chapter 3, we carried out some experiments. We used Kinect sensor to capture consecutive frames of an area and then using the process described in chapter 3, we managed to map the floor and calculate the error between the true contour and the output contour.

3.1 Experiment 1

3.1.1 Initialization

In the first experiment we captured three consecutive frames of a region in which there are two vertical walls and the floor. We considered the first frame to be the reference coordinate system and we transformed the points of the following frames to match with it. In this experiment we only had rotation between the frames. The experiment's initialization is shown in Table 2. The values shown in column Translation are in cm.

Frame	Translation	Rotation
1	[0, 0]	0°
2	[0, 0]	$\sim 12^o$
3	[0, 0]	$\sim 12^o$

Table 2 – Experiment 1 initialization

3.1.2 Decimation Results

For decimation we used the same settings for each frame. The faces were reduced to 6000 which was empirically a good choice in order to avoid both holes and computational complexity. Choosing a smaller number of faces for decimation may result to "holes" in some regions of the frame and therefore will give wrong outputs. These "holes" are created because the decimation algorithm reduces points from regions which are linear in order to keep as many edge points as possible so that the overall shape is preserved. In case we choose a larger number of faces we may have better results in the shape of floor, but the cost of complexity and processing would be much higher. The frame shown in figure 21 shows the points of the first frame after decimation.

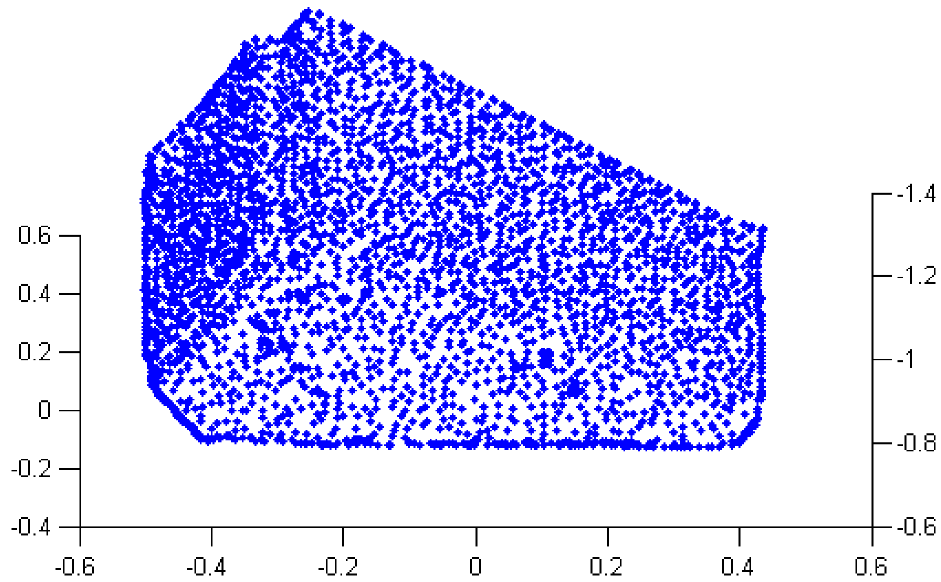

Figure 21 - Post-decimation points

3.1.3 Plane extraction and rotation estimation

RANSAC algorithm was used in every frame in order to identify all the planes in each of them. For this experiment the threshold for the maximum distance of a point from the plane was set to 0.8cm. The threshold of the percent of points needed to consider a plane good was set to 20%. The results of the seven consecutive frames are shown in figures 22 – 24 where the points of the frame are marked red. Note that the planes shown in the figures are not boundary correct but represent the set of points fitting them.

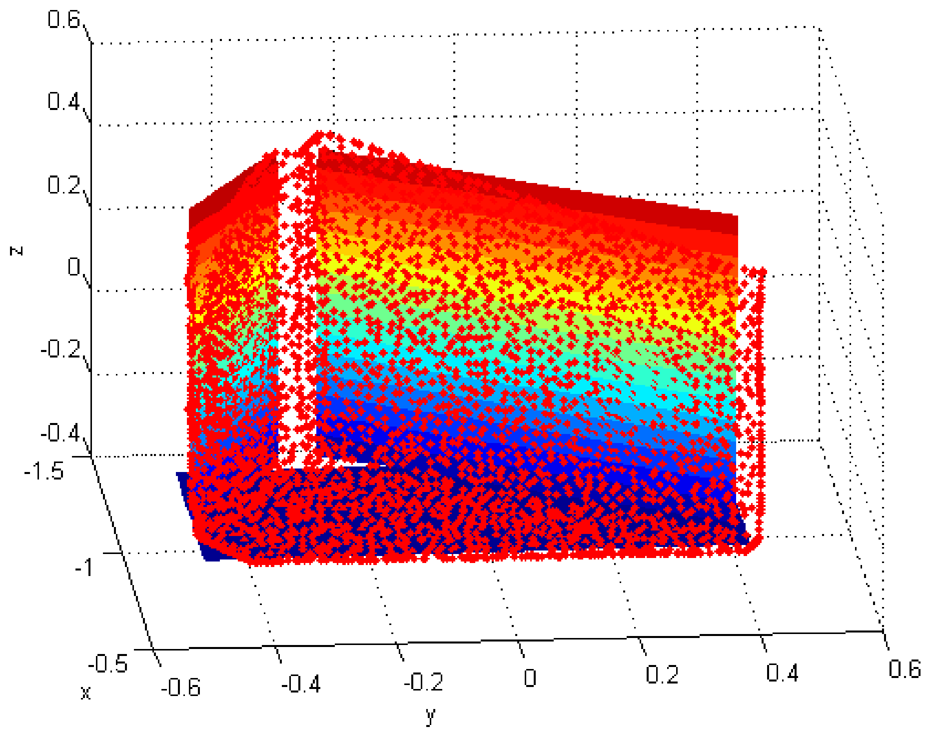

Figure 22 - Frame 1 planes and points

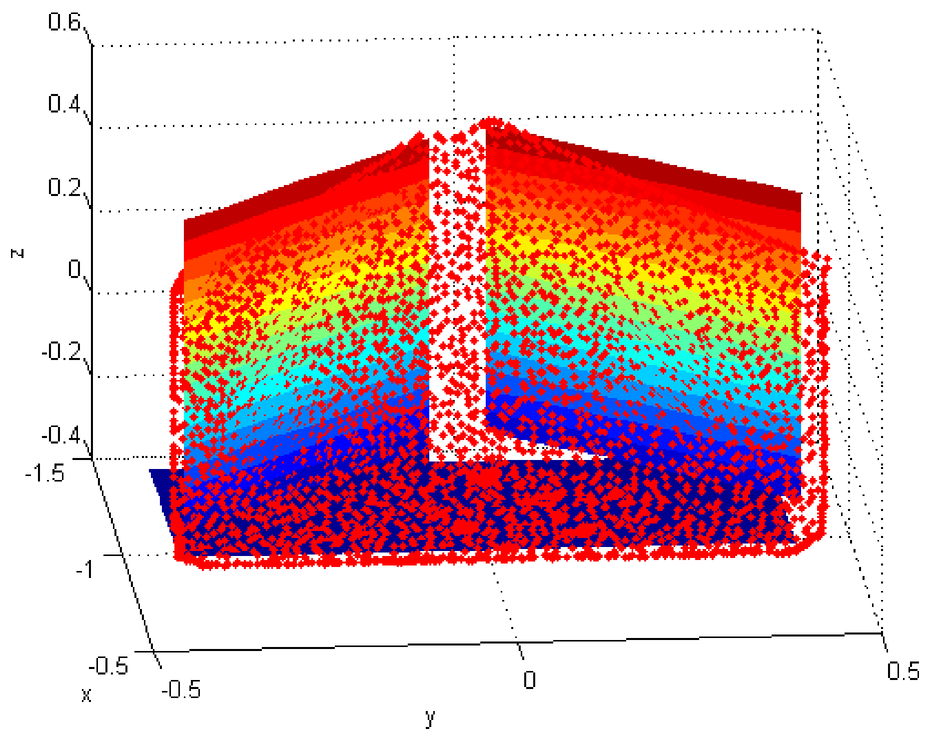

Figure 23 - Frame 2 planes and points

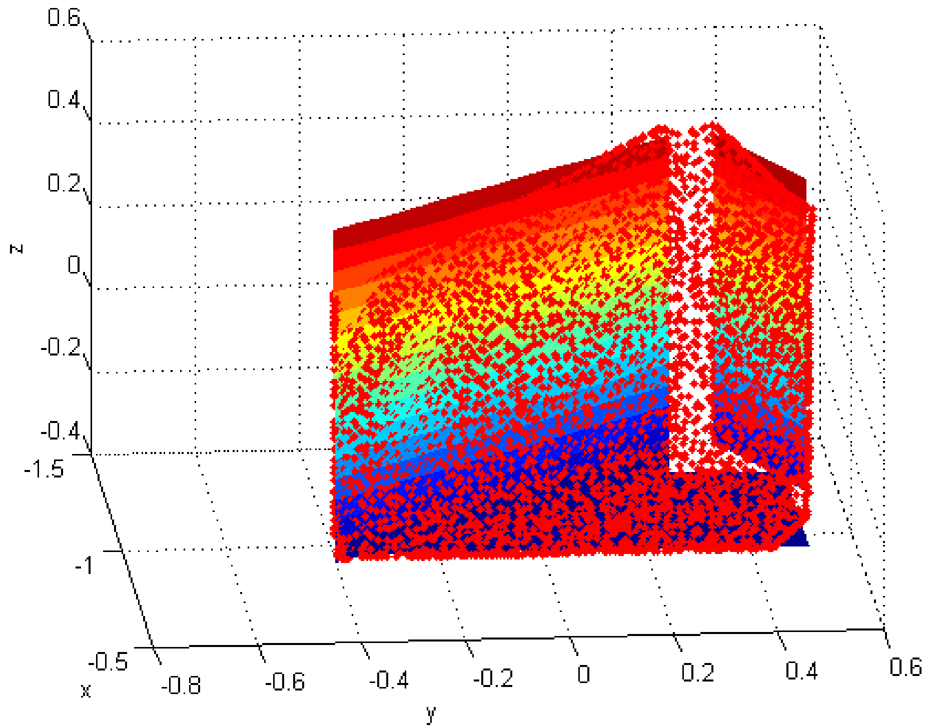

Figure 24 - Frame 3 planes and points

Table 3 shows all the planes found by RANSAC algorithm. The values shown are the normalized values of *A*, *B*, *C* and *D* of the planes in each frame and Table 3 the real and the algorithm's estimation angle between every two frames.

Frame	Planes in frame	Plane parameters [*A*, *B*, *C*, *D*]
1	3	Plane 1: [-0.8512, 0.5246, 0.0151, -0.9047] Plane 2: [-0.0124, 0.0112, -0.9999, -0.3144] Plane 3: [-0.5050, -0.8631, -0.0021, -0.8747]
2	3	Plane 1: [-0.6832, -0.7302, 0.0001, -0.8696] Plane 2: [-0.7171, 0.6967, 0.0156, -0.9130] Plane 3: [0.0085, -0.0010, 1.0000, 0.3101]
3	3	Plane 1: [-0.8407, -0.5414, -0.0098, -0.8569] Plane 2: [0.0071, -0.0085, -0.9999, -0.3137] Plane 3: [-0.5082, 0.8612, -0.0032, -0.9201]

Table 3 – Experiment 1 plane parameters

Frames	Real angle	Estimated angle
1 to 2	$\sim 12^{o}$	12.5254^{o}
2 to 3	$\sim 12^{o}$	15.3176^{o}

Table 4 – Experiment 1 estimated rotation angle

3.1.4 Floor identification and merging results

We used the variation of region growing algorithm on each of the three frames in order to find the region of the floor. The algorithm identified the points and faces that form the region of the floor. The result is shown in the figure below (figure 25). For this experiment we set the threshold (for identifying whether normal vectors are parallel enough to mean normal vector) to be 0.3. Note that values closer to 0 mean that the normal vector is closer to becoming parallel to the mean normal vector.

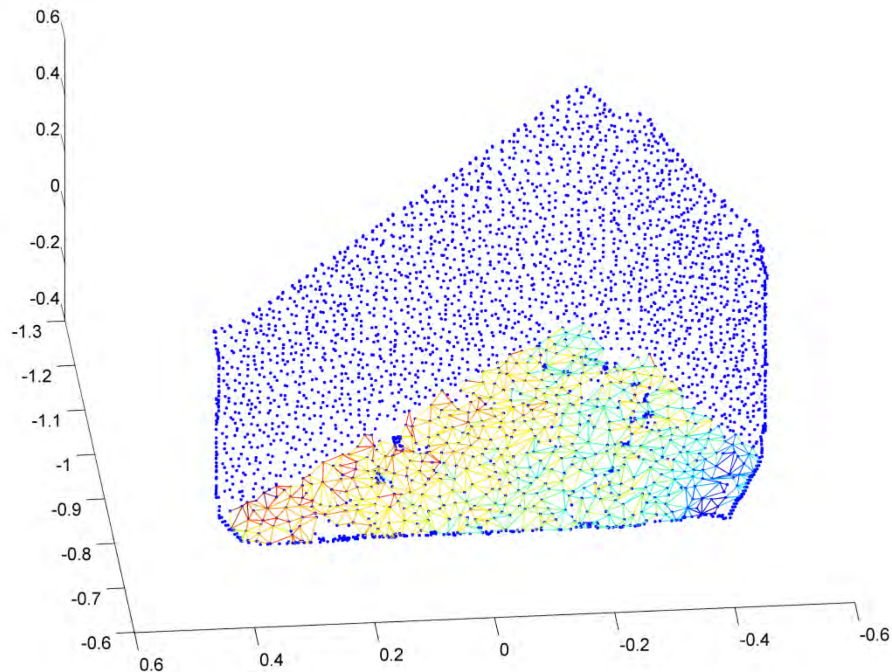

Figure 25 – Points (blue) and triangles found by region growing

Using the planes extracted by RANSAC we know the rotation angle between each frame and we can therefore calculate the rotation matrices. The rotation angle is $\mathbf{-12.5254^{o}}$ for the second frame and $\mathbf{-15.3176^{o}}$ for the third frame. The rotation matrices are shown below.

$$\begin{matrix} 0.9753 & 0.2209 & 0 \\ -0.2209 & 0.9753 & 0 \\ 0 & 0 & 1 \end{matrix}$$

$$\begin{matrix} 0.8918 & 0.4524 & 0 \\ -0.4524 & 0.8918 & 0 \\ 0 & 0 & 1 \end{matrix}$$

In figure 26 we can see the merged frames after their transformation (blue for frame 1, red for frame 2 and green for frame 3) and in figure 27 we can see the merged floor (blue for frame 1, red for frame 2 and green for frame 3).

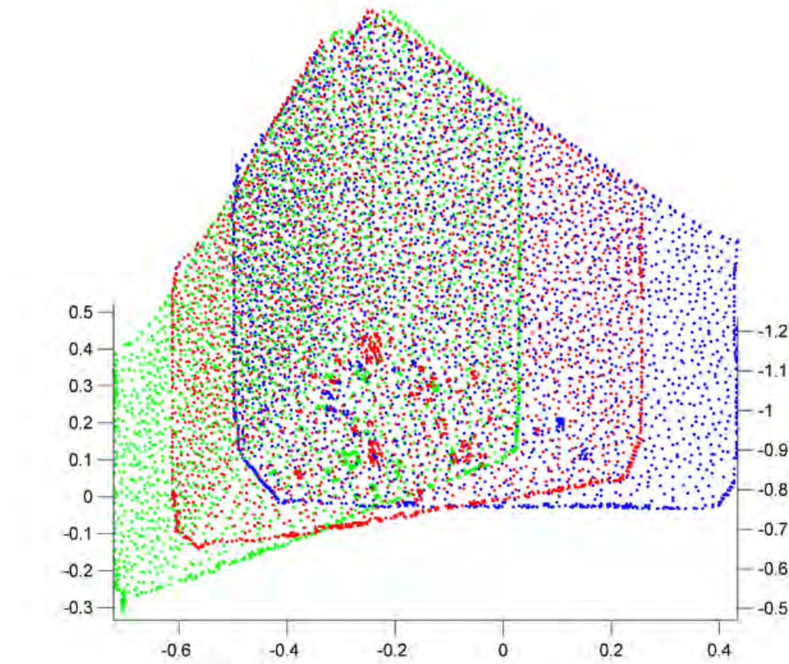

Figure 26 - Merged frames after transformation of points

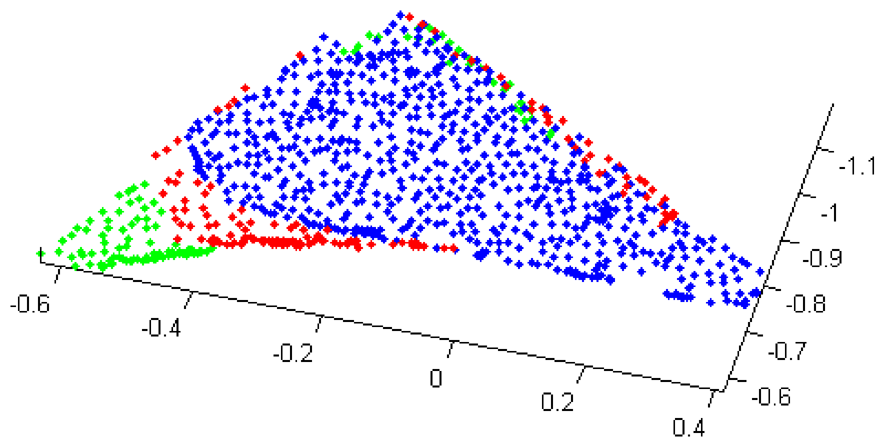

Figure 27 - Merged floor after performing region growing in all three frames (common points removed)

3.1.5 Contour extraction results

Now that we have the merged floor we need to find its contour. In order to do so we must first convert the coordinates to binary. We choose the pixel size to be 2.5cm and the result is a binary image in which white represents the pixels which have at least one point in them and black represents the pixels which have no points in them. The output binary image is shown in figure 28. To find the contour of this binary image we will use *bwtraceboundary*. The algorithm automatically selects a starting pixel to start the process. The output returned by the function is shown in figure 29 (marked in pink). The contour shown in figure 29.i is in pixels so the next step is to convert it to coordinates. To do that we will use the reverse procedure using the same pixel size (2.5cm). The result is shown in figure 30.

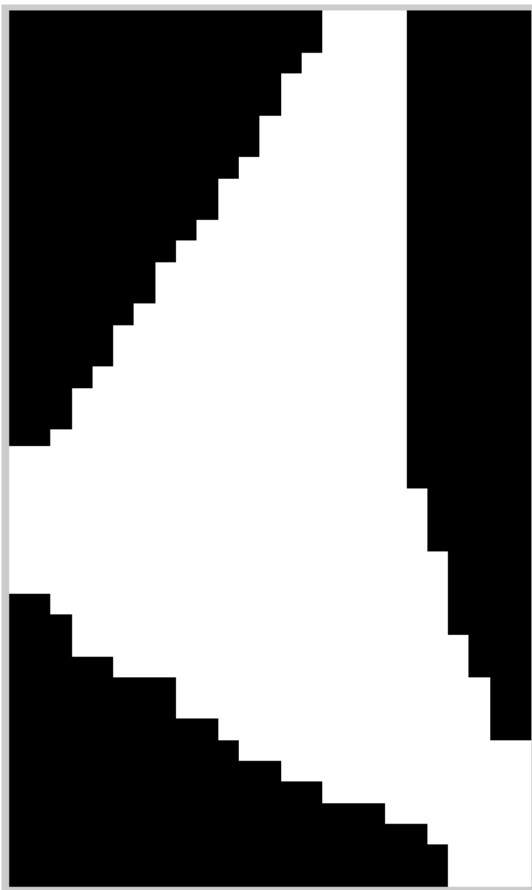

Figure 28 - Binary image of floor

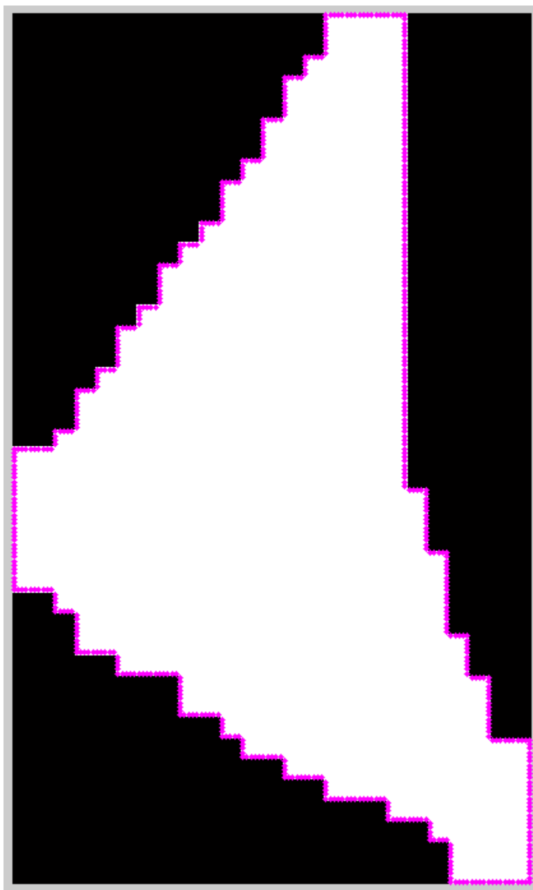

Figure 29 - Binary image of floor

The result seems quite good as most of the points are within the contour (marked red in figure 30). Using the methodology described in chapter 2.7 we calculated the mean error between the original contour and the output contour shown in figure 30. The mean error in this experiment is 0.98cm.

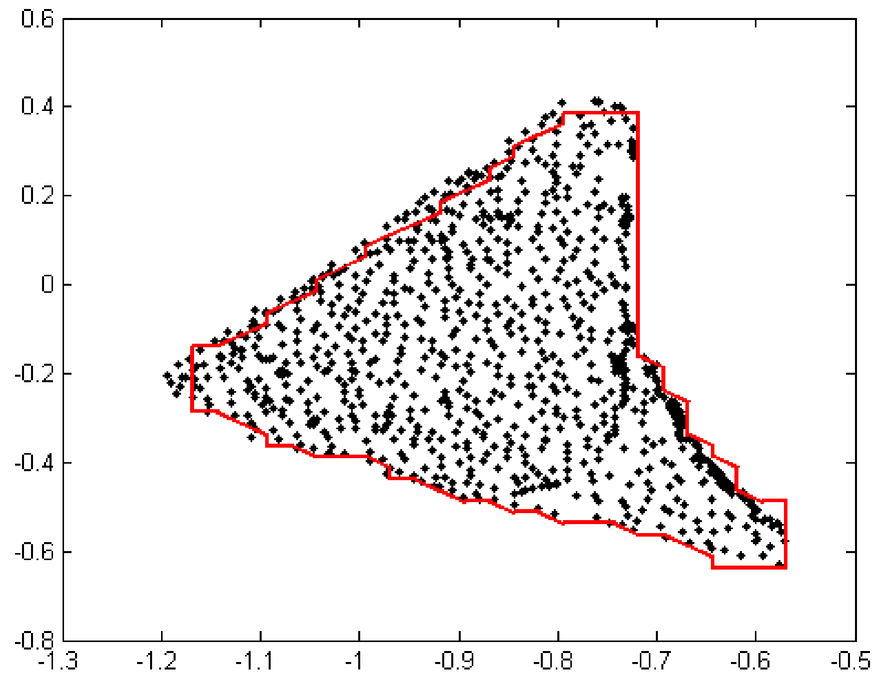

Figure 30 - Floor contour coordinates (red) and points of the floor (black)

3.2 Experiment 2

3.2.1 Initialization

In this experiment we captured seven consecutive frames of a classroom. We once again considered the first frame to be the reference coordinate system and we transformed the points of the following frames to match with it. In this case we had translation or rotation between the frames. The experiment's initialization is shown in Table 5. The values shown in column Translation are in cm.

Frame	Translation	Rotation
1	[0, 0]	0°
2	[-70, 0]	0°
3	[-140, 0]	0°
4	[-210, 0]	28°
5	[-210, -70]	28°
6	[-210, -140]	28°
7	[-210, -210]	28°

Table 5 – Experiment 2 initialization

3.2.2 Decimation Results

First we decimated the points and faces of a frame. We again used the same method in every frame so each frame's faces were reduced to 6000. The frame shown in figure 31 shows the points of the first frame after decimation.

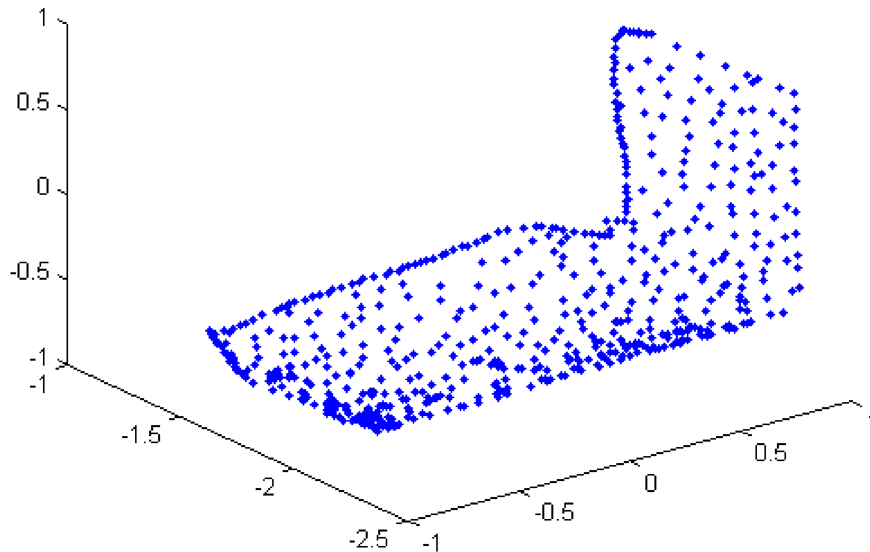

Figure 31 - Post-decimated points

Decimation reduced the points and faces by a huge amount preserving the overall shape of the frame perfectly.

3.2.3 Plane extraction and rotation estimation

Once again we used RANSAC algorithm every frame in order to identify all the planes in each of them. In this case the threshold for the maximum distance of a point from the plane was set to 1.5cm. The threshold of the percent of points needed to consider a plane good was set to 12%. The results of the seven consecutive frames are shown in figures 32 – 38 where the points of the frame are marked red. Note that the planes shown in the figures are not boundary correct but represent the set of points fitting them.

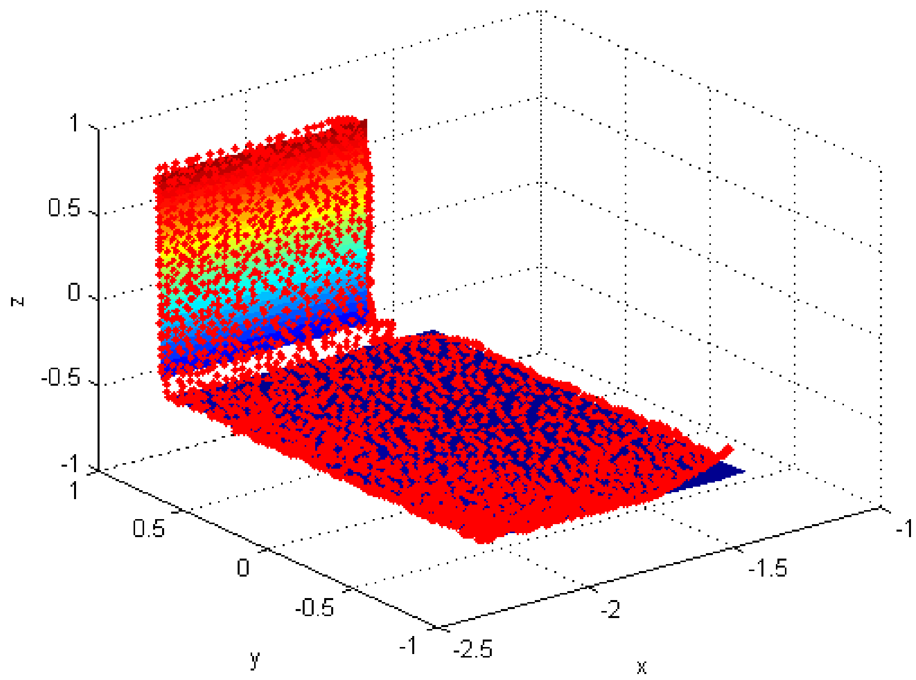

Figure 32- Frame 1 planes and points

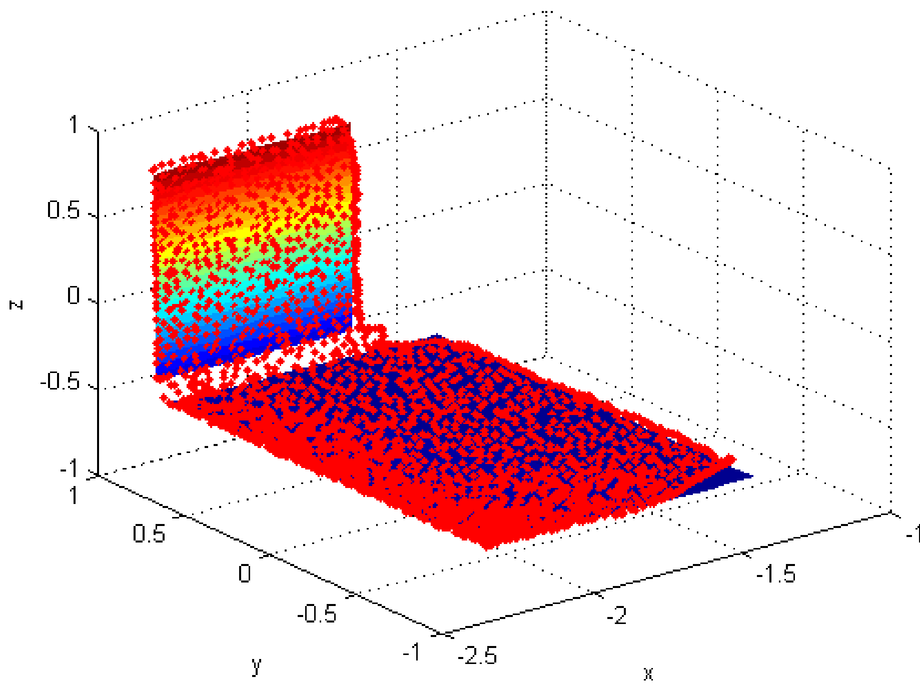

Figure 33 - Frame 2 planes and points

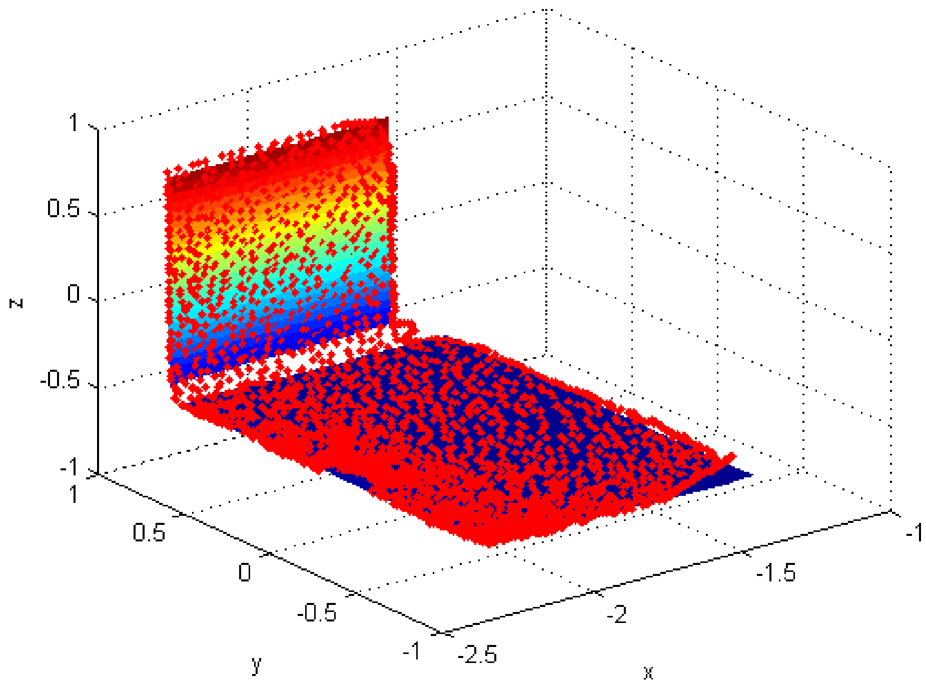

Figure 34 - Frame 3 planes and points

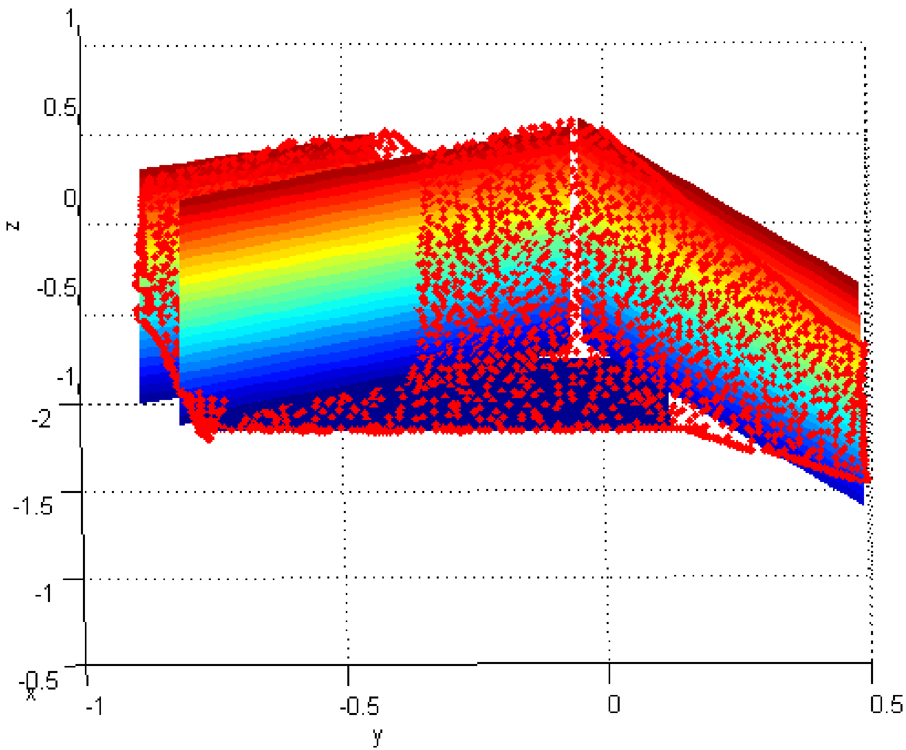

Figure 35 - Frame 4 planes and points

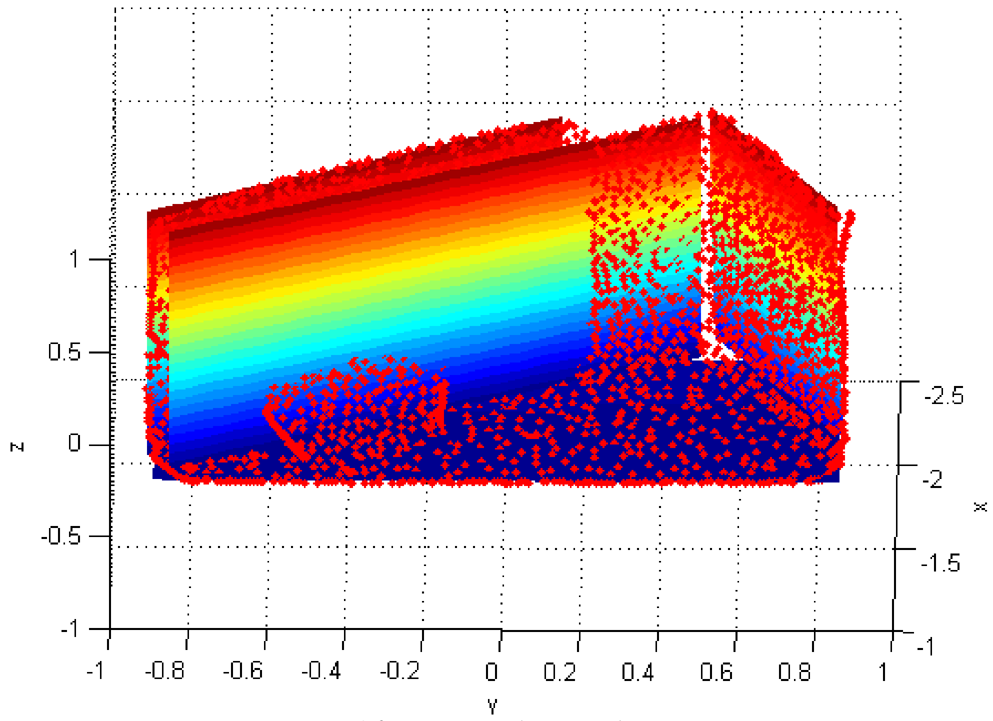

Figure 36 - Frame 5 planes and points

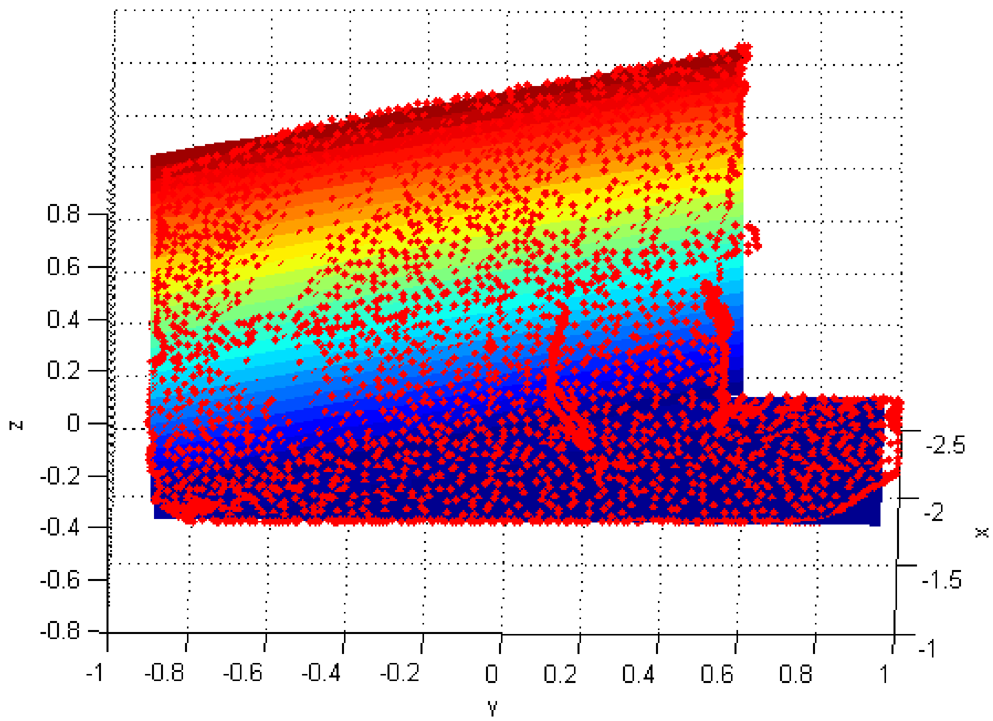

Figure 37 - Frame 6 planes and points

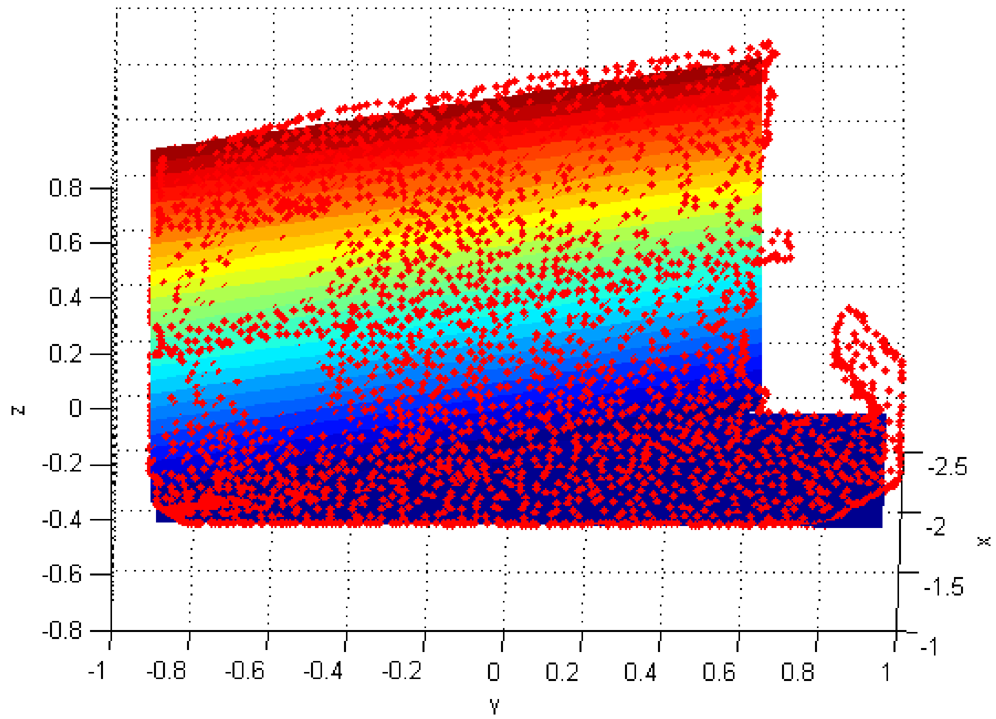

Figure 38 - Frame 7 planes and points

RANSAC has found all the planes in each frame. Table 6 shows the normalized values of *A*, *B*, *C* and *D* of the planes in each frame and Table 7 the real and the algorithm's estimation angle between every two frames.

Frame	Planes in frame	Plane parameters [*A*, *B*, *C*, *D*]
1	2	Plane 1: [-0.0272, -0.0042, -0.9996, -0.6363] Plane 2: [0.0105, 0.9999, -0.0094, -0.8790]
2	2	Plane 1: [0.0265, 0.0072, 0.9996, 0.6371] Plane 2: [0.0245, 0.9996, -0.0105, -0.8768]
3	2	Plane 1: [-0.0236, -0.0015, -0.9997, -0.6288] Plane 2: [-0.0150, 0.9998, -0.0100, -0.8942]
4	4	Plane 1: [0.8671, 0.4968, -0.0347, 1.8084] Plane 2: [-0.4879, 0.8729, 0.0074, -0.8625] Plane 3: [0.8921, 0.4507, -0.0327, 1.6700] Plane 4: [-0.0345, -0.0041, -0.9994, -0.6524]
5	4	Plane 1: [-0.8759, -0.4818, 0.0246, -1.8337] Plane 2: [0.8616, 0.5062, -0.0360, 1.6351] Plane 3: [-0.4861, 0.8739, 0.0094, -1.5352] Plane 4: [0.0355, 0.0046, 0.9994, 0.6609]
6	2	Plane 1: [-0.8814, -0.4715, 0.0295, -1.8089] Plane 2: [0.0207, 0.0109, 0.9997, 0.6401]
7	2	Plane 1: [-0.8912, -0.4521, 0.0373, -1.8104] Plane 2: [0.0299, 0.0115, 0.9995, 0.6543]

Table 6 – Experiment 2 plane parameters

Frames	Real angle	Estimated angle
1 to 2	0°	0.8024°
2 to 3	0°	2.2609°
3 to 4	28°	28.3604°
4 to 5	0°	0.1650°
5 to 6	0°	0.7235°
6 to 7	0°	1.3244°

Table 7 – Experiment 2 estimated rotation angle

3.2.4 Floor identification and merging results

Using once again the variation of region growing algorithm on each of the seven frames we identified the region of the floor. The algorithm identified the points and faces that form the region of the floor. The result is shown in the figure below (39). For this experiment we set the threshold (for identifying whether normal vectors are parallel enough to mean normal vector) to be 0.3.

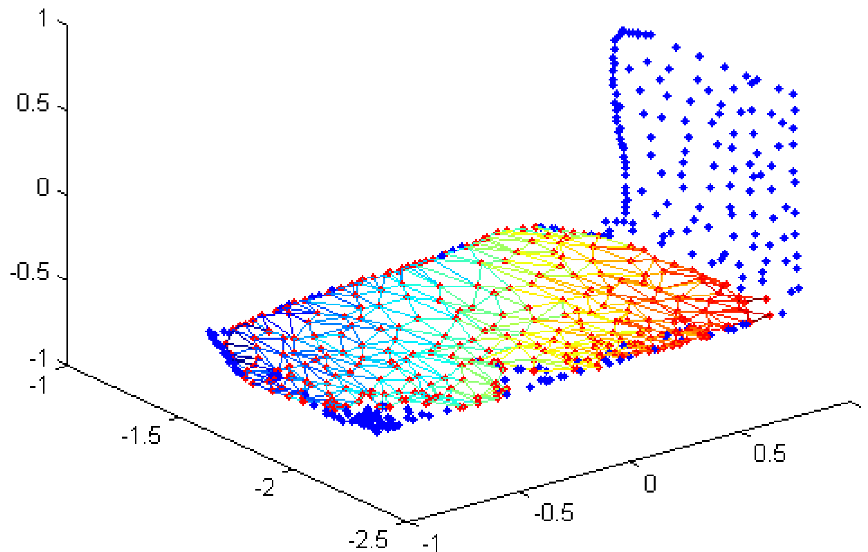

Figure 39 - Points (blue) and triangles found by region growing

Next step is to merge the seven frames together. Using the odometer we know that the distance between the first 2 frames is **70cm** on the X axis. Kinect realizes the coordinates of X axis in front of it as negative, so in order to merge the first two frames we need to just subtract **70cm** from the X axis coordinates of the second frame's points. Frame 3 follows the same pattern so to merge it with the other two we need to subtract **140cm** from the X axis coordinates of the third frame's points. The fourth frame is a little different as it features both translation and rotation. The rotation is +**28.3604°** (as estimated in Table 7) around Z axis and in order to apply it to the points we must first create the rotation matrix which is shown below:

$$\begin{matrix} 0.8775 & -0.4795 & 0 \\ 0.4795 & 0.8775 & 0 \\ 0 & 0 & 1 \end{matrix}$$

To apply the rotation to the points we multiply the rotation matrix with the points of the fourth frame. The movement is the same as in frames 2 and 3, so we subtract **210cm** from the X axis coordinates of the fourth frame's points. It is critical that we first rotate the points and then move them. Frame 5 also has both movement and rotation. The rotation is the same and the movement is on Y axis this time. The translation is **70cm** to the left so it will be negative. We first apply the rotation by multiplying the points with the rotation matrix. Then we subtract **70cm** from the Y axis coordinates of the fifth frame's points. To make it match with the other frames we must also subtract **210cm** form the X axis coordinates of the fifth frame's points. Frames 6 and 7 follow the same pattern as frame 5 so we first rotate its points using the same rotation matrix and then subtract **210cm** form the X axis coordinates. Lastly we subtract **140cm** from the Y axis coordinates of the sixth frame's points and **210cm** from the Y axis coordinates of the seventh frame's points. The result of the merged frames is shown in figure 40 (blue (right) for frame 1, red for frame 2, green for frame 3, yellow for frame 4, purple for frame 5, black for frame 6 and blue (left) for frame 7) and the merged floor after removing common points in each frame, in figure 41 (blue (right) for frame 1, red for frame 2, green for frame 3, yellow for frame 4, purple for frame 5, black for frame 6 and blue (left) for frame 7).

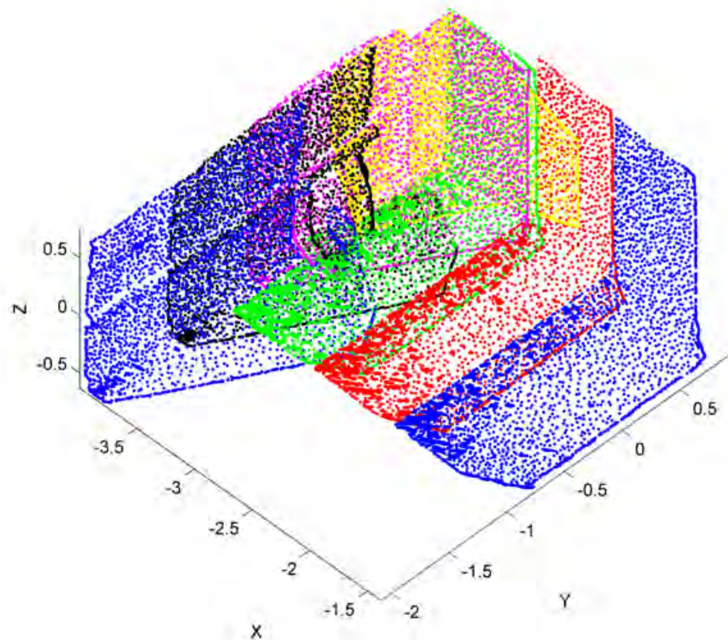

*Figure **40** - Merged frames after transformation of points*

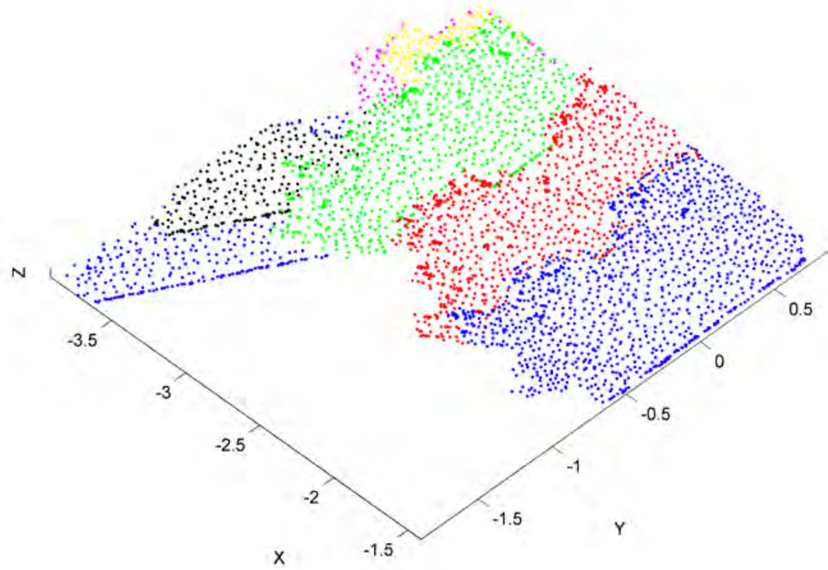

*Figure **41** - Merged floor after performing region growing in all three frames (common points removed)*

3.2.5 Contour extraction results

Now that we have the merged floor we need to find its contour. In order to do so we must first convert the coordinates to binary. We choose the pixel size to be 5cm and the result is a binary image in which white represents the pixels which have at least one point in them and black represents the pixels which have no points in them. The output binary image is shown in figure 42. To find the contour of this binary image we will again use *bwtraceboundary*. The output returned by the function is shown in figure 43 (marked in pink). The contour shown in figure 43 is in pixels so the next step is to convert it to coordinates. To do that we will use the reverse procedure using the same pixel size (5cm). The result is shown in figure 44.

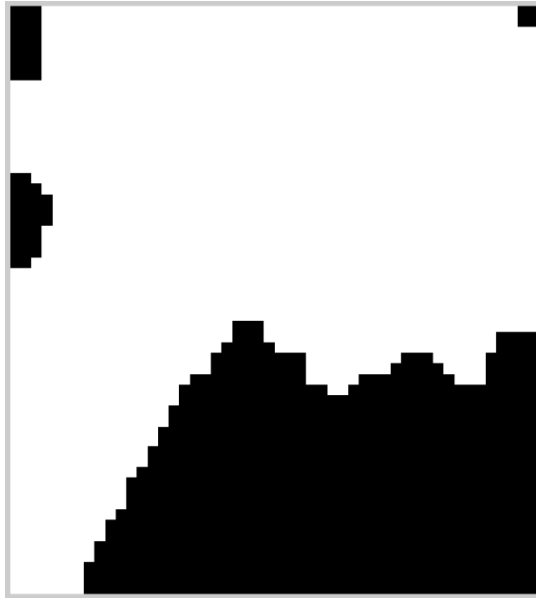

*Figure **42** - Binary image of floor*

*Figure **43** - Binary image contour (pink pixels)*

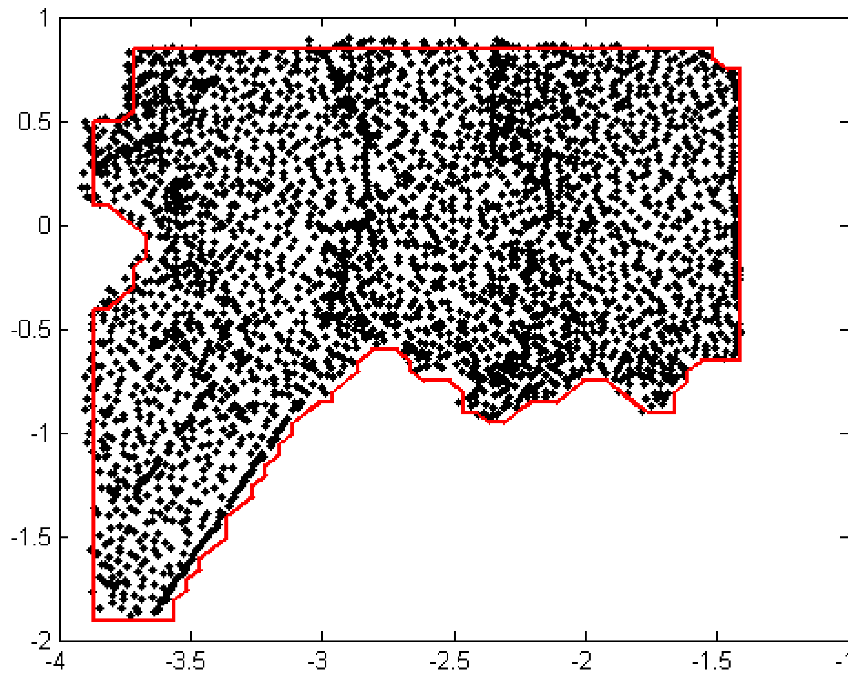

*Figure **44** - Floor points (black) and contour (red) coordinates*

The result seems quite good as most of the points are within the contour (marked red in figure 44). Using the methodology described in chapter 2.7 we calculated the mean error between the original contour and the output contour shown in figure 44. The mean error in this case is about 1.34cm.

3.2.6 Experimenting with parameters

The mean error of the output contour may vary with different parameters. The parameters which can be changed are the pixel size used for the binary image creation and the number of faces after decimation. To further analyze the process and to show the different results the algorithm can give as output, we experimented with those parameters. Table 8 shows the results in different values of these parameters. Looking at the values of Table 8 we can easily understand that bigger pixel size value results in greater error even if the number of faces is big. Furthermore we can see that a higher number of faces can result in less error which seems legit because the more faces and points there are in a frame the less "holes" will be present.

Number of faces	Pixel size	Error
1000	5cm	3.75cm
1000	10cm	4.68cm
1000	20cm	7.67cm
3000	5cm	3.91cm
3000	10cm	4.29cm
3000	20cm	6.19cm
6000	5cm	1.34cm
6000	10cm	3.30cm
6000	20cm	5.13cm

Table 8 – Experiments with parameters

4 Conclusion and future work

The results from the experiments conducted show that our method maps the environment efficiently with a relatively small error. In Table 9 we present the running times of each step of our method. The process is not real time as some steps take several seconds to complete. The main speed problems can be seen in reading process, in points / faces decimation process and in region growing process.

Experiment	Reading of file	Decimation	RANSAC	Region Growing	Total time
1	~3sec	~1.5sec	~15ms	~10sec	~15.5sec
2	~10sec	~6sec	~40ms	~22sec	~40sec

Table 9 – Running times for every step / frame in each of the two experiments. The process was performed on an AMD FX-6300 processor

The process for each step takes ~15.5 seconds / frame to complete in the first experiment and ~40 seconds / frame in the second experiment. Table 10 shows the process of the overall contour extraction and calculation of error in each experiment.

Experiment	Contour extraction and error calculation
1	~1sec
2	~2sec

Table 10 – Running times in each of the two experiments. The process was performed on an AMD FX-6300 processor

Future work

Future work on this method can improve its efficiency and its speed. Some of them are presented below:

- Implementation of a more efficient and faster method for reading the data captured by Kinect. A way to do this is to read directly from stream without saving data into a file.
- Implementation of a faster method for decimating the points/faces of frames.
- Calculation of translation between frames from the info of the planes extracted by RANSAC algorithm.
- Calculation of rotation not only in z axis but also in x and y axis and translation on z axis.
- Implementation of a method for merging planes representing the same surface in consecutive frames.

5 References

[1] Kinect Specifications (Microsoft) - https://msdn.microsoft.com/en-us/library/jj131033.aspx

[2] Kinect coordinates image - http://aivarastumas.weebly.com/kinervo.html

[3] PLY read method (Gabriel Peyr) - https://www.mathworks.com/matlabcentral/fileexchange/5355-toolbox-graph/content/toolbox_graph/read_ply.m

[4] Cristina Urdiales, Antonio Bandera, Eduardo Perez, Alberto Poncela, and Francisco Sandoval "Hierarchical planning in a mobile robot for map learning and navigation", Volume 116 of the series Studies in Fuzziness and Soft Computing pp 165-188

[5] Arleo, A., Millan, J.R. and Floreano, D. (1999) "Efficient learning of variable-resolution cognitive maps for autonomous indoor navigation", IEEE Transac-tions on Robotics and Automation, Vol. 15, No. 6, pp. 990-1000

[6] Borenstein, J., Everett, H.R. and Feng, L. (1996) "Navigating mobile robots: systems and techniques", Wellesley, Massachusetts: A.K. Peters, Ltd.

[7] Matarí c, M.J. (1994) "Interaction and intelligent behavior", Technical Report AI-TR-1495, MIT, AI-Lab, Cambridge-USA.

[8] Moravec, H. P. (1988) "Sensor fusion in certainty grids for mobile robots", AIMagazine, Vol. 9, No. 2, pp. 61-74.

[9] Thrun, S., Bucken, A., Burgard, W., Fox, D., Frohlinghaus, T., Hennig, D., Hofmann, T., Krell, M., and Schimdt, T. (1998) "Map learning and high-speed navigation in RHINO", MIT/AAAI Press, Cambridge.

[10] Zelinsky, A. (1992) "A mobile robot navigation exploration algorithm", IEEE Transactions on Robotics and Automation, Vol. 8, pp. 707-717.

[11] Region Growing (Wikipedia) - https://en.wikipedia.org/wiki/Region_growing

[12] RANSAC (Wikipedia) - https://en.wikipedia.org/wiki/Random_sample_consensus

[13] Point to plane distance (Mathworld) - http://mathworld.wolfram.com/Point-PlaneDistance.html

[14] Tae-kyeong Lee, Seungwook Lim, Seongsoo Lee, Shounan An, and Se-young Oh, *Senior Member, IEEE* (2012) "Indoor Mapping Using Planes Extracted from Noisy RGB-D Sensors*", Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference

[15] Kamarulzaman Kamarudin, Syed Muhammad Mamduh, Ali Yeon Md Shakaff, Shaharil Mad Saad, Ammar Zakaria, Abu Hassan Abdullah and Latifah Munirah Kamarudin (2013) "Method to Convert Kinect's 3D

Depth Data to a 2D Map for Indoor SLAM ", Signal Processing and its Applications (CSPA), 2013 IEEE 9th International Colloquium

[16] Daniel R. dos Santos, *Member, IEEE*, Marcos A. Basso, Kourosh Khoshelham, Elizeu de Oliveira, Jr., Nadisson L. Pavan, and George Vosselman (2016) "Mapping Indoor Spaces by Adaptive Coarse-to-Fine Registration of RGB-D Data", IEEE Geoscience and Remote Sensing Letters (Volume: 13, Issue: 2, Feb. 2016)

[17] Gabriela Gallegos and Patrick Rives (2010) "Indoor SLAM Based on Composite Sensor Mixing Laser Scans and Omnidirectional Images", Robotics and Automation (ICRA), 2010 IEEE International Conference

[18] Ying Zhang, Juan Liu, Gabriel Hoffmann, Mark Quilling, Kenneth Payne, Prasanta Bose, Andrew Zimdars (2010) "Real-Time Indoor Mapping for Mobile Robots with Limited Sensing", Mobile Adhoc and Sensor Systems (MASS), 2010 IEEE 7th International Conference

[19] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Renand Dieter Fox (2012) "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments", The International Journal of Robotics Research 0(0) 1–17

[20] Thrun S, Burgard W and Fox D (2000) "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping", In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).

[21] Triebel R and Burgard W (2005) "Improving simultaneous mapping and localization in 3D using global constraints", In Proceedings of the National Conference on Artificial Intelligence (AAAI).

[22] Newman P, Sibley G, Smith M, Cummins M, Harrison A, Mei C, et al. (2009) "Navigating, recognizing and describing urban spaces with vision and laser", The International Journal of Robotics Research 28(11–12): 1406–1433.

[23] Nister D (2004) "An efficient solution to the five-point relativepose problem", IEEE Transactions on Pattern Analysis and Machine Intelligence 26: 756–777.

[24] Akbarzadeh A, Frahm JM, Mordohai P, Clipp B, Engels C, Gallup D, et al. (2006) "Towards urban 3D reconstruction from video", In Pro-ceedings of the Third International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT).

[25] Konolige K and Agrawal M (2008) "FrameSLAM: From bundle adjustment to real-time visual mapping", IEEE Transactions on Robotics 25(5): 1066–1077.

[26] D. Holz, C. Lörken, and H. Surmann "Continuous 3D Sensing for Navigation and SLAM in Cluttered and Dynamic Environments", In Proc. of the International Conference on Information Fusion (FUSION), 2008.

[27] O. Wulf, K. O. Arras, H. I. Christensen, and B. Wagner "2D Mapping of Cluttered Indoor Environments by Means of 3D Perception", In Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA), 2004.

[28] Dirk Holz, Stefan Holzer, Radu Bogdan Rusu, and Sven Behnke (2011) "Real-Time Plane Segmentation using RGB-D Cameras*", Volume 7416 of the series Lecture Notes in Computer Science pp 306-317

[29] Dirk Holz and Sven Behnke (2012) "Fast Range Image Segmentation and Smoothing using Approximate Surface Reconstruction and Region Growing", Volume 194 of the series Advances in Intelligent Systems and Computing pp 61-73

[30] K.-M. Lee, P. Meer, and R.-H. Park "Robust adaptive segmentation of range images", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, pp. 200–205, 1998.

[31] L. Silva, O. Bellon, and P. Gotardo "A global-to-local approach for robust range image segmentation", In Proc. of the Int. Conference on Image Processing (ICIP), Rochester, NY, USA, 2002, pp. 773–776.

[32] R. Schnabel, R. Wahl, and R. Klein " Efficient RANSAC for point-cloud shape detection. ", Computer Graphics Forum, vol. 26, no. 2, pp. 214–226, 2007.

[33] D. Hahnel, W. Burgard, and S. Thrun "Learning compact: 3D modelsof indoor and outdoor environments with a mobile robot", Robotics and Autonomous Systems, vol. 44, no. 1, pp. 15–27, 2003.

[34] Ren, Mingwu, Jingyu Yang, and Han Sun. "Tracing boundary contours in a binary image", *Image and vision computing* 20.2 (2002): 125-131.

[35] Schroeder, William J., Jonathan A. Zarge, and William E. Lorensen. "Decimation of triangle meshes", *ACM Siggraph Computer Graphics*. Vol. 26. No. 2. ACM, 1992

[36] Toussaint, Godfried T. "A historical note on convex hull finding algorithms", *Pattern Recognition Letters* 3.1 (1985): 21-28.

[37] Fadili, Mohamed-Jalal, Mahmoud Melkemi, and Abderrahim Elmoataz. "Non-convex onion-peeling using a shape hull algorithm", *Pattern recognition letters* 25.14 (2004): 1577-1585.

[38] Breu, Heinz, et al. "Linear time Euclidean distance transform algorithms", *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.5 (1995): 529-533.