PERFORMANCE COMPARISON OF POINT AND PLANE FEATURES FOR SLAM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MÜCAHİT YÖRÜK

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

DECEMBER 2014

Approval of the thesis:

PERFORMANCE COMPARISON OF POINT AND PLANE FEATURES FOR SLAM

submitted by **MÜCAHİT YÖRÜK** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver ___________
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Gönül Turhan Sayan ___________
Head of Department, **Electrical and Electronics Engineering**

Assoc. Prof. Dr. İlkay Ulusoy Parnas ___________
Supervisor, **Electrical and Electronics Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Aydan Erkmen ____________________
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. İlkay Ulusoy Parnas ____________________
Electrical and Electronics Engineering Dept., METU

Prof. Dr. Gözde Bozdağı Akar ____________________
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Afşar Saranlı ____________________
Electrical and Electronics Engineering Dept., METU

Salih Eren Balcı (Msc.) ____________________
ASELSAN

Date: 02.12.2014

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name	: Mücahit YÖRÜK
Signature	:

ABSTRACT

PERFORMANCE COMPARISON OF POINT AND PLANE FEATURES FOR SLAM

Yörük, Mücahit
M. S., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. İlkay Ulusoy Parnas
December 2014, 163 pages

Simultaneous Localization and Mapping (SLAM) is an indispensable capability for mobile robots that explore unknown environments. This advanced method is now widely employed since the development of improvements in sensor technology, such as 3D depth cameras. To avoid the risk of the human interaction in dangerous environments, various SLAM algorithms have been developed and proposed in the literature. The aim of this study, is to develop a landmark vector that improves the SLAM performance using the planar features of objects. In order to achieve this goal we generated a fastSLAM algorithm and two different feature extraction methods. The first feature extraction method is SURF, which gives responses at the edges of the depth images and the second feature extraction method is plane detection, which gives a compact representation of the environment. Throughout this thesis, four different landmark vectors are defined *(SURF point, plane as point, plane as oriented point and plane as surface)* and compared the effects on the SLAM. The advantages of using planar features are shown with both the RGBD SLAM dataset and the real time application.

Keywords: 3D fastSLAM, SURF, Plane Detection, RGBD Kinect Camera

ÖZ

SLAM UYGULAMASINDA NOKTA VE DÜZLEM ÖZELLİKLERİNİN PERFORMANS KARŞILAŞTIRMASI

Yörük, Mücahit
Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. İlkay Ulusoy Parnas
Aralık 2014, 163 sayfa

SLAM algoritmaları bilinmeyen ortamlarda çalışan robotlar için vazgeçilmezdir. Gelişen almaç kabiliyetleri sayesinde 3D derinlik bilgisinin alınabilmesi ile bu konu üzerinde geniş ölçüde çalışmalar başlamıştır. Tehlikeli ortamlarda insan etkileşiminden kaynaklı riskleri ortadan kaldırmak için çeşitli SLAM algoritmaları önerilmiştir. Bu tezin amacı düzlemsel özelliklerin kullanılarak SLAM algoritmasının performansını arttıracak işaret vektörlerinin oluşturulmasıdır. Bu amaç doğrultusunda fastSLAM algoritması uygulanmış ve iki farklı özellik çıkarma algoritması kullanılmıştır. İlk algoritma derinlik görüntüsünde nesnelerin sınırlarında cevap veren SURF algoritmasıdır. İkinci algoritma ise objeleri iyi tanımlayan düzlemsel özelliklerin çıkarılmasıdır. Tez boyunca dört farklı işaret vektörü kullanılmış ve SLAM performansına etkileri karşılaştırılmıştır. Düzlemsel özelliklerin kullanılmasının avantajları yayınlanmış olan bir veritabanı ile ve gerçek zamanlı uygulama ile gösterilmiştir.

Anahtar Kelimeler: 3D fastSLAM, SURF, Düzlemsel Özelliklerin Çıkarılması, 3D Kamera

To my family

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor Dr. İlkay Ulusoy Parnas for her guidance, advice, criticism, encouragement and insight throughout the completion of the thesis.

I am indebted to all of my friends and colleagues for their support and encouragements. I am also grateful to ASELSAN Inc. for the facilities that made my work easier.

Finally, I am grateful to my wife for her continuous support and encouragements.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Problem Definition and Motivation

With the evolution of technology, it has become possible to use intelligent agents in many areas even those where there is high risk thus, intelligent agents offer an alternative dangerous tasks being undertaken by human beings. When carrying out these tasks the location and map information provides the agents with a coherent way to interact with surrounding objects and people. These interaction agents can navigate safely, identify surrounding objects and deal with unexpected situations.

For all these applications (called tasks or missions), intelligent agents have to localize and if not given they have to generate the map. This kind of application may appear to be very easy to implement with powerful sensors *(such as GPS)*, however, because of environmental conditions it is not possible to use these sensors for every task. In closed environments GPS does not give true information and its resolution for outdoor applications may not be satisfactory. On the other hand, range sensors, laser scanners, 3D Time of Flight ToF cameras and Microsoft Kinect sensors have measurement errors and intelligent agents have to take these errors into account.

Generated algorithms to handle this kind of problem are called simultaneous localization and mapping *(SLAM)* algorithm. There are many different types of SLAM algorithms but the probabilistic approach which is relatively new is considered to be the best solution. In the probabilistic approach, the location of the intelligent agent *(vehicle pose)* and the surrounding objects *(landmarks, feature points, sensor measurements)* are defined with a probabilistic distribution function instead of a single point. Intelligent agents can handle sensor measurement errors, motion errors and algorithmic errors with the help of this probabilistic definition [1].

SLAM applications are convenient for use in both indoor and outdoor environments. For example, Minerva, an autonomous robot, is used as a tour guide in the Smithsonian National Museum of American History. The robot gives information to the visitors and navigates in the museum. During the navigation, it performs collision avoidance and uses path planning algorithms [2]. Also there are important SLAM implementations for outdoor environments. The Stanford Racing Team's vehicle *"Stanley"* is an important example of an outdoor SLAM application. Stanley won the DARPA (Defense Advanced Research Projects Agency) Grand Challenge in 2005 competing against 195 teams. The race was conducted over 142 miles in the Mojave Desert [3].

Over the last 20 years, various solutions have been proposed to resolve the SLAM problems encountered in indoor and outdoor environments. The sensor technology has the most important effect on the consistency of SLAM applications and due to technical limitations, researchers used sonar sensor or 2D laser scanners in earlier solutions [4]. Subsequently, 2D laser scanners were used with the addition of a mechanism to alter the sensor pitch angle and define the 3D environment [5]. Also, others researchers used stereo vision algorithms to determine the 3D location of landmarks *(objects)*, but these stereo vision algorithms were sensitive to lightning conditions. Later, camera companies produced TOF cameras which give the point cloud data directly with 30 fps;, however, these cameras are not extensively used because of their high cost. Nowadays, low cost RGBD Kinect sensors are extensively used for indoor SLAM applications. The use of 3D point cloud data allows the representation of environment with compact feature vectors.

This thesis reports on the experimental investigation into the contribution of planar features in an indoor SLAM. Different experiments were performed and comparisons were made using simulation parameters. Real datasets [6] were used in the experiments together with collected data from the METU Computer Vision laboratory.

1.2 Literature Survey

In the literature, there are different solution methods to the SLAM problems. These solution methods can be grouped according to the applied algorithm and sensor type.

1.2.1 Algorithms

The popular Iterative Closest Point (ICP) Scan Matching method was proposed by Besl and McKay in 1992 [7]. The idea was to align consecutive scans *(taken by external sensors)* iteratively and to estimate the transformation matrix between these scans. This ICP algorithm can be used with raw or processed data. These registrations can be undertaken with point sets, line segments, implicit curves, parametric curves, triangle sets, implicit surfaces and parametric surfaces [7].

Another implementation of ICP scan matching was proposed by Biber in 2003 [8]. Biber subdivided a 2D plane into cells and defined a normal distribution *(mean and variance)* to each cell that defines the probability of the measuring point [8]. For each cell that has at least three points, the following computation was carried out.

1 – Collect all 2D points $x_{i=1,\dots,n}$ contained in a cell.

2 – Calculate the mean $q = \frac{1}{n}\sum_i x_i$

3 – Calculate the Covariance matrix $\Sigma = \frac{1}{n}\sum_i (x_i - q)(x_i - q)^t$.

Normal distribution $N(q, \Sigma)$ defines the probability of measuring a sample point in that cell. The ICP algorithm determines the corresponding normal distributions between consecutive scans according to normal distributions and odometer data. The algorithm computes the sum of probabilities for each point. According to the result *(score)* the algorithm ends the computation or continues until convergence [8].

Another important implementation of ICP Scan Matching was proposed for 3D point sets. The idea was based on the human behavior. Humans look at the big picture first and then concentrate on the details. The Multi-Layered NDT algorithm adopts the same perspective as a human being to speed up the pairing process. The algorithm first tries to pair with lower layer which has only 8 cells and if the pairing is not satisfactory then it tries to pair with second layer that has 64 cells. In practice, using the first four layers is sufficient to make a good pairing between consecutive scans [9]. One of the 3D scans and its NDT representation for the top layer is given in Figure 1-1. The red points define the point cloud data, and blue shapes define the fitted distributions to these points for the first layer of the NDT representation.

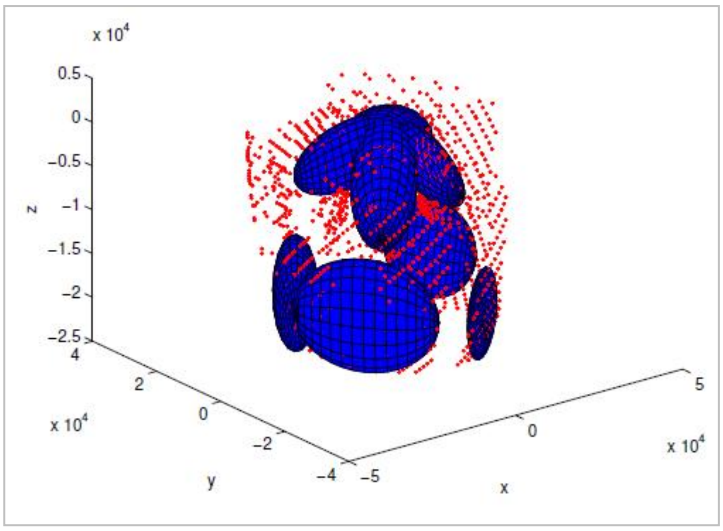


Figure 1-1 3D scan and its first layer representation with NDT [9].

On the other hand, other researchers used ICP algorithms at the object level for SLAM applications. The algorithm, detects the objects from the 3D point cloud data and then pairs the detected objects between consecutive scans with the ICP algorithm [10]. Figure 1-2 shows a sample scene and detected objects for an ICP-based SLAM application.

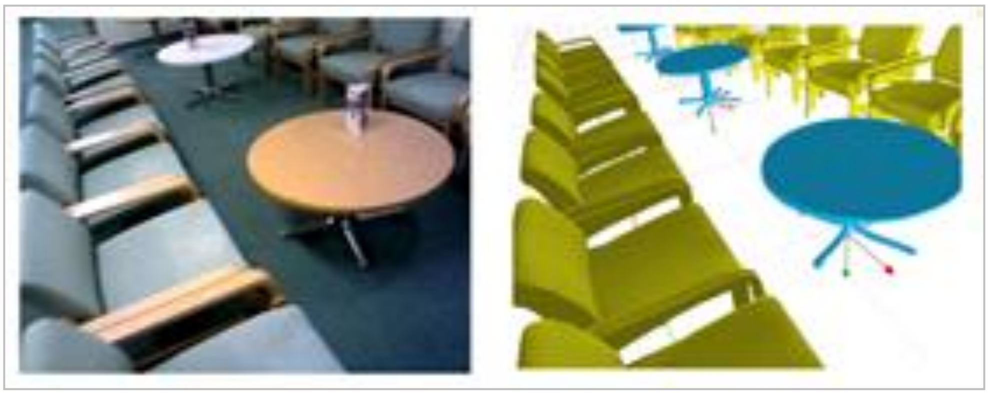


Figure 1-2 Sample scene and detected objects [10].

The Extended Kalman Filter (EKF) is another important SLAM method which uses probabilistic approach. It was published by Smith, Self and Cheeseman in 1990 [11]. Probabilistic approaches take into account all the expected errors in probabilistic manner and define landmarks or objects with a distribution. This probabilistic approach makes SLAM applications more robust than scan-matching based methods. Furthermore, probabilistic solutions are the only solution for the kidnapped robot problem [11]. Also, probabilistic approaches need least requirements according to the scan-matching based methods [1]. On the other hand, these advantages come at the price of computational inefficiency and approximation [1]. Different types of the EKF- SLAM methods were published for different environments and conditions. Efficient SLAM algorithms were generated for real time applications and large scale environments with EKF [12].

Another important method is the fastSLAM *(a factored solution to the SLAM problem)* algorithm which uses a sampling method *(particle filters)* instead of defining the probabilistic distribution function. The fastSLAM algorithm was developed in response to the time limitations *(computational complexity)* of the EKF based approaches. The most important feature of this method is its robustness to sensor failures [13]. There are many different implementations of fastSLAM in the literature for different environments with different types of sensors.

In 2007, Tanaka and Ito developed a walking aid system for a handicapped or elderly person. The system generated a 2D map of the environment with fastSLAM and laser scanners. They compared the fastSLAM algorithm with an ICP-based method, and explained that for slow motion both work but if the motion is not slow then the ICP based algorithm gives the wrong correspondences [14].

Arbeiter et al. proposed an algorithm to construct the 3D environment in 2010. The application uses a fastSLAM approach with TOF and color cameras. SURF features are extracted from the RGB camera and 3D correspondences are founded with ToF camera data. The final map only includes the raw point cloud data and the extracted features [15].

As explained above, there are 3 main types of SLAM algorithms; EKF based methods and Particle Filter (PF) based methods. All these methods have advantages and disadvantages; so it is important to choose the most appropriate method for different conditions *(sensor type, environment)*.

1.2.2 Sensor and Feature Types

In order to make the correct selection from the large number of different feature types the data source and environmental conditions must be taken into account.

For LIDAR data there are two main simultaneous localization and mapping approaches; feature extraction, and scan matching methods. Features depend on the environments. Lines, corners and curvatures are appropriate features for indoor environments. For an outdoor environment, tree features are important and they can be detected with a special feature detector. Tree features are not suitable for an office environment and corner features are not appropriate for a forest. There is no general purpose feature detector for varied environments [16].

Scan matching can also be used with LIDAR data since it directly uses raw data and the SLAM performance does not depend on surrounding objects *(lines, corners, trees…etc.)*. On the other hand, scan matching based methods tend to create dense pose graphs that significantly increase the computational cost [16].

Feature based methods are computationally less expensive than scan matching based methods. The computational cost of the scan matching method increases if the prior translational uncertainty increases. The computational complexity of the feature based matching is nearly independent of the initialization error [16]. If feature based methods are able to offer the same robustness and broad applicability to different environments, they would be more preferable than scan matching based methods [16].

Vision based 3D SLAM algorithms have been proposed with stereo vision, but these algorithms are very sensitive to the lightning conditions of the environment and the processing time of these methods make them useless in real time applications.

Nowadays, working with the 3D point cloud data is very popular. Weingarten used 3D features for feature-based SLAM in structured 3D environments. Weingarten extracted features directly from 3D point cloud data which was taken from Swiss ranger ToF camera and these 3D planar features are used to survey the performance of an EKF-SLAM [17]. Hedlund worked with ToF cameras to register consecutive scans and generated the 3D map of environment [18]. Ying Yang and Förstner proposed an algorithm to extract compact features and planes from 3D point cloud data in 2010 testing the performance of the proposed method with synthetic and real data [19]. Rusu proposed a method to generate an object map of indoor environments using 3D point cloud data [20]. Also Turunc investigated the contribution of planar segments to 3D EKF-SLAM using IRSCAN *(IR distance measurement based scanner system)* [21].

As in the research reported in this thesis other research used the Freiburg dataset and investigated the effect of SLAM performance with different feature types *(SURF, SIFT, ORB, SURF+Shi Tomasi),* only using low cost Kinect sensor data [22]. One study compared the localization performances of filter types *(EKF,PF)* in an indoor known environment [23]. The findings showed that the PF approach gives better result than the EKF and the performance results of using low cost Kinect sensor in localization was comparable with most state-of-the-art methods [23]. In 2012, Hartmann et al. presented a paper about Visual SLAM using a Kinect Camera with Oriented FAST and Rotated BRIEF *(ORB)* features with the fastSLAM algorithm. The performance of the ORB features were investigated in a room environment with 30 particles showing that the ORB features were better than the SIFT and SURF features [24]. On the other hand, some researchers preferred to use hybrid methods combining point *(SURF)* and plane features using Kinect sensor with a real time hand held SLAM application. They claimed that this hybrid method gives faster and more accurate registration than only using points [25].

In 2013 Taguchi et al. investigated the accuracy using planes or points for a SLAM application. This is same idea as the investigation detailed in this thesis. This algorithm provides faster correspondence with plane features and compact representation Furthermore, an application with planar features does not suffer from the local minima and robot converges to the true pose [26].

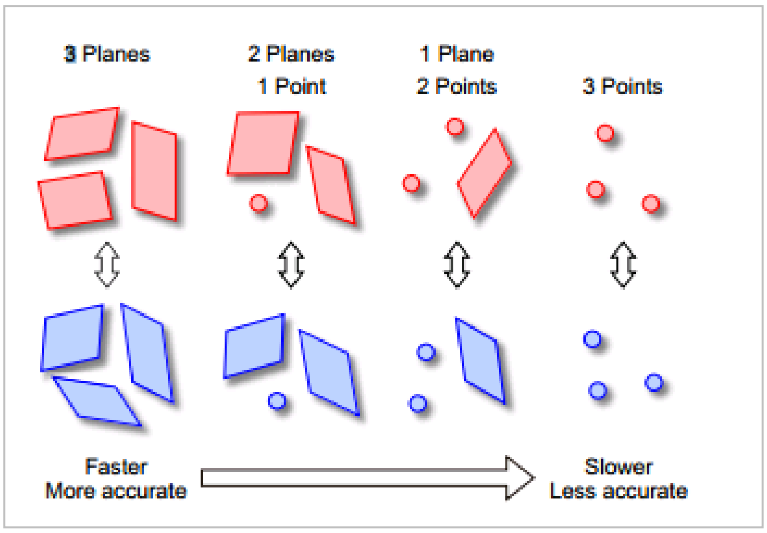


Figure 1-3 Performance comparison for accuracy and time [26].

To summarize, the researchers investigated the SLAM performance in various environments and conditions with different types of sensors. There are successful approaches to SLAM problem but none of them can be generalized to every condition. Currently, researchers are working on the generalized solution for the 3D SLAM with the compact representation of the map *(objects, surfaces, planes, curvatures)*.

1.3 Thesis Contribution

The major contributions of this thesis are related to constructing and running a fastSLAM algorithm with planar features. In this study, feature extraction algorithms implemented and the contribution of planar features to the fastSLAM performance for indoor applications was investigated. The navigation environment of the robot is planar and the robot moves along an x and y direction and rotates around the z direction. Throughout the study four different landmark vectors were used and a comparison was made in relation to the effect on SLAM performance. The four different landmark vectors are ;

- SURF Points (x, y, z, scale, laplace)
- Plane as Point (x, y, z,)
- Plane as Oriented Point (x, y, z, n_x, n_y, n_z)
- Plane as Surface (x, y, z, n_x, n_y, n_z, area)

The work presented here is constructed as follows.

The organization of the remainder of this thesis is as follows. **Chapter 2** introduces the theoretical background information about the SLAM algorithms. In **Chapter 3** details about the navigation environment and sensors are presented. **Chapter 4** explains the feature extraction methods together with the details of plane detection and SURF feature extraction. The details of fastSLAM algorithm and our SLAM implementation are contained in **Chapter 5**. In **Chapter 6** the results of the experiments are given together with a graphical representation of simulations and comments on the final results.

The principal contributions of this thesis are as follows.

- Implementation of feature extraction *(plane extraction, SURF feature extraction)* methods.
- The fastSLAM implementation with different landmark vectors *(SURF point, plane as point, plane as oriented point, plane as surface)* and performance comparison.
- Performance comparison under the effect of below parameters.
  - Effective Particle Rate
  - Linear Velocity Error Rate
  - Angular Velocity Error Rate
  - Particle Number
  - Re-sampling
- Data collection from METU Computer vision laboratory with the Pioneer 2 robot and Microsoft Kinect Sensor.

CHAPTER 2

THEORETICAL BACKGROUND

2.1 Localization

Localization is the problem of determining the pose of a robot according to the environment and it is the main precondition for robot navigation *(planning and motion)*. For a given map, it may appear easy to find the pose of a robot using sensor measurements however, since there are no noise free sensor readings the robot has to infer the pose from the sensor measurements [1].

The localization can be separated into three types according to the initial information obtained. If the robot knows the initial pose the problem is called position tracking. In this case, the main effort is to handle motion errors. Secondly, if the robot does not know where it is, this a global localization problem which includes position tracking and matching. In the global localization problem, the algorithm has to run through the entire map. The last localization type is the kidnapped robot problem. This is a type of global localization problem. The robot thinks that it knows the location, but it does not. A robot can manage a sensor failure problems if it can handle the kidnapped robot problem [1].

The localization problems can be grouped according to the environment as ; static or dynamic. In static environments, the only variable quantity *(state)* is the robot pose. In dynamic environments there could be other robots, objects or people which have location or configuration changes over time [1].

Another grouping is passive or active localization. In the passive approach the localization module only observes the robot while it moves randomly or performs routine tasks. However, in active approach the localization module controls the robot and tries to minimize the localization error [1].

The last group of the localization problem concerns the number of robots. The most common approach is single robot localization problem in which there is no communication issue. However, in the multi-robot localization problem all robots localize themselves and share information about their location. This information sharing makes resolution of the localization problem easier [1].

2.2 Map Building and Map Types

Mapping is the representation of the navigation environment. This map representation allows agents to plan their actions in order to reach the goal which can be anything such as; finding an object, navigating safely to anywhere place. However, in many cases, the agents do not have an accurate map of the environment and need to generate the map through the navigation. In fact, navigating around the environment and mapping can be the goal.

The maps can be separated into two types according to the map information. Topological maps contain the free spaces of the environment and metric maps contain the surrounding objects and features.

2.2.1 Topological Maps

Topological maps can be generated by nodes and edges *(links, arcs)*. These nodes and edges define free spaces. Topological maps can be modeled with graph representation which uses less memory usage and provides a compact representation of the map. Topological map representation is appropriate for large areas such as the roads that connect cities [27].

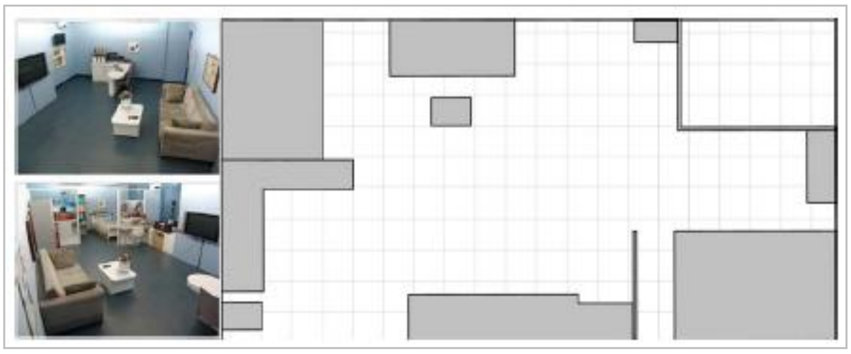


Figure 2-1 Layout of an indoor environment for topological mapping [27].

Figure 2-2 shows the result of the topological mapping for the navigation environment containing tables, sofa, bed and other household items given in Figure 2-1.

The topological map of the indoor environment shown in Figure 2-2 contains 7 nodes and 6 links. The representation of the map and its generation is very easy for topological maps however, this simple representation has some disadvantages as listed below.

- Routes described by links are not always the optimal ones.
- Topological maps do not include any accurate geometric description of the environment.
- Path planning algorithms are not appropriate for topological maps [27].

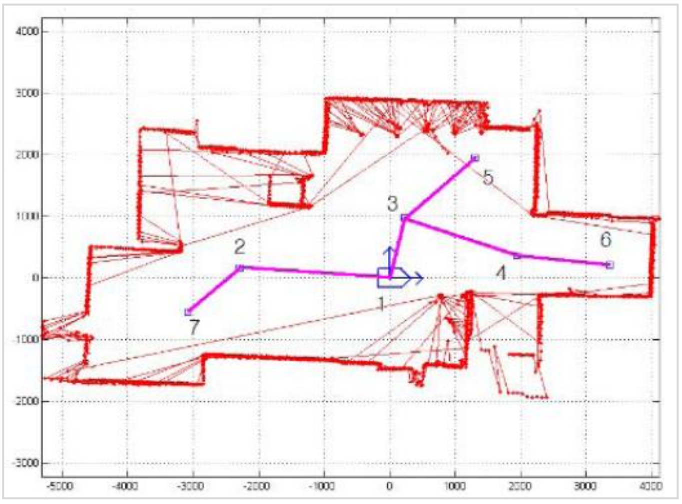


Figure 2-2 Topological map of the indoor environment [27].

2.2.2 Metric Maps

Metric maps describe the navigation environment. According to the representation method metric maps can be divided into 2 types.

2.2.2.1 Feature-based Maps

Feature based maps represent the environment with global location, orientation and parametric features. These features are corners, edges, SURF features, SIFT features, planes and any kind of object such as doors, trees or lights. In 2D and 3D SLAM applications the algorithms extract and integrate these features into the robot system with mono or stereo camera systems using the corner features to represent the environment. In Kalay’s work, features have been extracted from a 2D image and mapped to a 3D space with stereo vision [28]. Weingarten extracted features directly from 3D point cloud data. These 3D planar features have been used to survey the performance of EKF-SLAM with 3D planar features [17]. In another survey, the researchers represented their own general purpose feature detector for indoor and outdoor environments and compared their detector with corner detectors in an indoor environment and a tree detector in outdoor environments [16].

In all work presented above the features have been represented with their parameter set. For example, SURF features can be defined by 5 parameters *(x, y, z, sigma, laplace)* in 3D while corner point features are defined with only 3 parameters (x, y, z). x, y, and z represents the center of the location and these are also directly useful for SLAM but other parameters *(sigma, laplace)* also assist in corresponding landmarks [29].

Feature based maps do not show the free spaces of the environment. Instead, these maps are generated by certain parameters which define the surrounding objects. Feature-based maps have memory advantage according to the occupancy of grid maps. This advantage comes with the compact representation. Feature based maps represent the environment with only *nxm* matrix *(n: feature vector size, m: number of features (landmarks))*. Especially in the 3D SLAM, this compact representation decreases the memory usage and the computational cost.

On the other hand, feature based SLAM algorithms must include pre-defined robust feature extractors. If feature extractors are not sufficiently robust, this results in erroneous correspondences and the SLAM algorithm fails.

2.2.2.2 Occupancy Grid Maps

An occupancy grid representation was initially proposed by Elfes [30]. The continuous space of the environment is separated into cells in this representation. These cells marked as occupied, empty or unexplored [30].

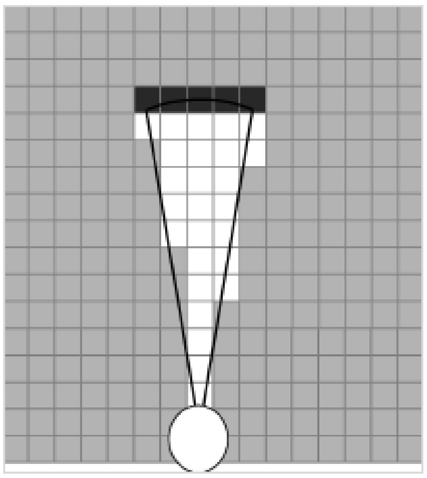


Figure 2-3 Representation of the environment with occupancy grids [1].

Figure 2-3 shows the representation of the environment with occupancy grids. Black color defines occupied grids, white color defines empty grids and gray color defines unexplored grids. Figure 2-4 shows an example of an occupancy grid map.

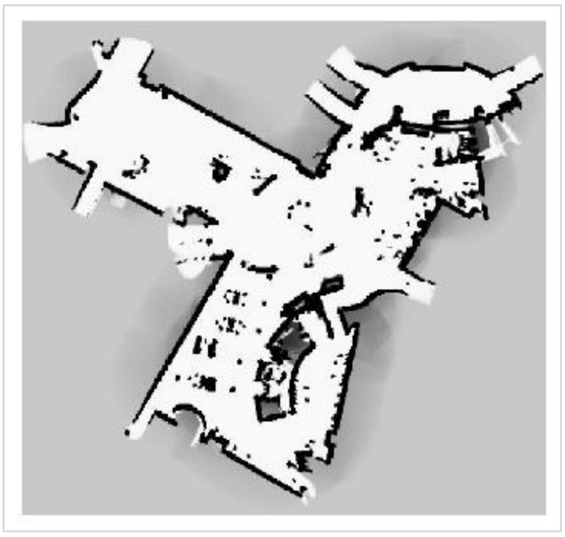


Figure 2-4 A sample map with occupancy grids [1].

The supporting idea for this method is to find the most likely map for a given sensor and actuator data.

$$m^* = argmax_m P(m|u_{1:t}, z_{1:t}) \tag{2.1}$$

Modeling dynamic obstacles and large uncertainty around found obstacles are two main disadvantages of occupancy grid maps [1].

2.3 Simultaneous Localization and Map Building (SLAM)

The SLAM problem is also known as CML *(Concurrent Mapping and Localization).* This problem arises when the agent does not know the map of the environment and its true pose. Generally, in real time applications, the agent does not have information about either and has to deal with this problem [1].

SLAM algorithms can be categorized according to many parameters such as features, environment, vehicle model, sensor model, and matching algorithm. There are many different approaches for different environments and setups. The simplest grouping should be determined according to the sensor technology.

In the earlier research sonar sensors or 2D laser scanners were used in the SLAM applications because of the technical limitations. Then the researchers needed to define 3D world and they began to use 2D laser scanners with addition of a mechanism to alter the sensor pitch angle [5]. Other researchers used stereo vision to determine 3D location of landmarks.

Later, the invention of the TOF cameras which directly give the 3D point cloud, provided an easier way to obtain 3D data from the environment but this camera is expensive and therefore was not extensively used.

Nowadays, low cost RGBD Kinect sensors are very popular in robotic applications. In comparison with earlier solutions it is easy to use but since the resolution and reliability are not sufficient for SLAM applications greater than 3 meters it is only appropriate for indoor use.

Some researchers use visual information *(stereo vision)* while some of them use depth data *(point cloud data)*. The use of different types of information is in an effort to find a better way to define the environment around the robot. Nowadays, the low cost RGBD sensor *(Microsoft Kinect)* is available for SLAM applications and because of its efficiency 3D-SLAM algorithms have become the most popular subject for indoor environments.

2.4 Probabilistic Approach: Bayesian Filter

Estimating the state of the robot and its environment using sensor data is the core of probabilistic robotic applications. An efficient state estimator can compute the current state of the robot recursively based on the previous state. This computation can be undertaken using a Bayesian Filter.

In a Bayesian filter, the probability of state can be found with the help of measurement and control data. The Bayesian filter contains two consecutive stages. First, finding the prediction and then with sensor measurements updating prediction and find the belief.

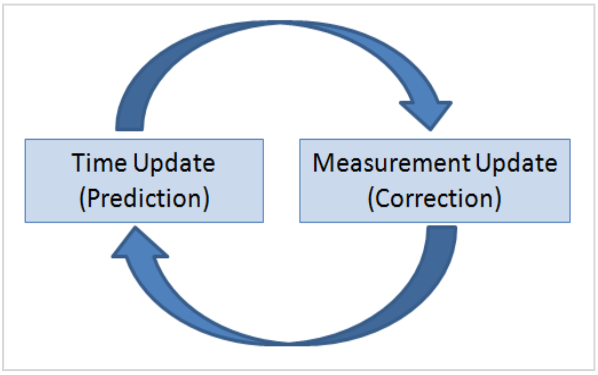


Figure 2-5 Prediction - correction sequence [31].

2.4.1 Derivation of Bayesian Filter

Figure 2-6 shows the dynamic Bayesian network for SLAM applications. In the figure, the edges define the relationship between nodes. The state at time t (x_t) depends on the sensor measurements ($z_{1:t}$) and control data ($u_{1:t}$) and also the sensor measurements at time t (z_t) depends on the state at time t (x_t).

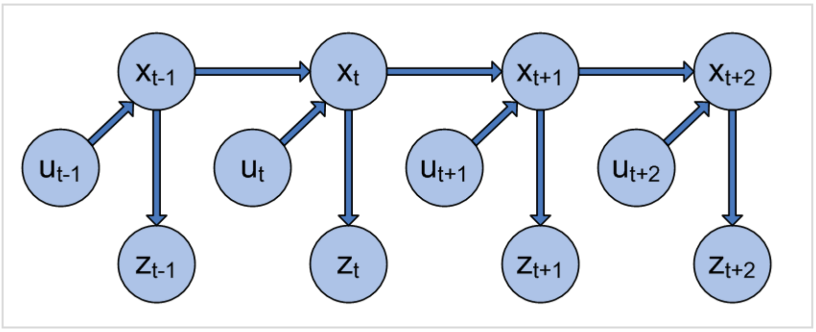


Figure 2-6 Dynamic Bayesian network for SLAM [1].

Using Bayesian formula the conditional probability of the state vector (x_t) can be written as;

$$p(x_t \mid z_{1:t}, u_{1:t}) = \frac{p(z_t \mid x_t, z_{1:t-1}, u_{1:t})\, p(x_t | z_{1:t-1}, u_{1:t})}{p(z_t \mid z_{1:t-1}, u_{1:t})} \tag{2.2}$$

In equation 2.2, the denominator is only a normalizing constant and then the following equation can be written;

$$p(x_t \mid z_{1:t}, u_{1:t}) = \eta . p(z_t \mid x_t, z_{1:t-1}, u_{1:t})\, p(x_t | z_{1:t-1}, u_{1:t}) \tag{2.3}$$

For a given parent node x_t the measurement vector becomes independent from the previous sensor measurements and control data. Figure 2-7 shows this independence. The given node is colored green and the removed edges are shown in red.

According to Figure 2-7 the conditional probability becomes;

$$p(z_t \mid x_t, z_{1:t-1}, u_{1:t}) = p(z_t \mid x_t) \tag{2.4}$$

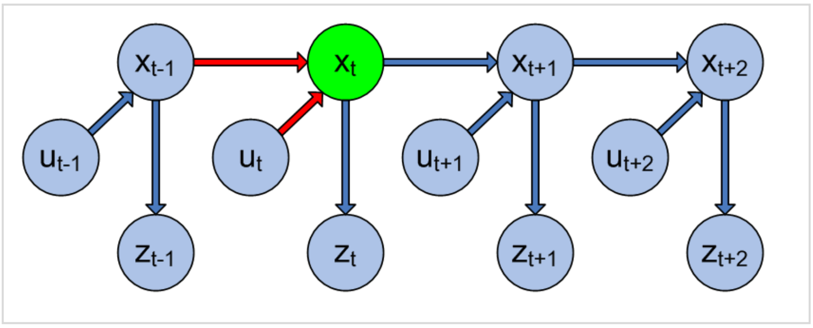


Figure 2-7 Removed edges in a graph model [1].

Then equation 2.3 is simplified with equation 2.4.

$$p(x_t \mid z_{1:t}, u_{1:t}) = \eta . p(z_t \mid x_t)\, p(x_t | z_{1:t-1}, u_{1:t}) \tag{2.5}$$

Equation 2.5 shows the belief and it can be written in terms of the **prediction ($\overline{bel}(x_t)$).**

$$bel(x_t) = \eta . p(z_t \mid x_t)\, \overline{bel}(x_t) \tag{2.6}$$

Calculating the belief is the second step of the Bayesian filter estimation. The first step is calculating the prediction. Now write x_t should be written according to one state before x_{t-1} to calculate prediction.

$$\overline{bel}(x_t) = p(x_t | z_{1:t-1}, u_{1:t}) \tag{2.7}$$

This is achieved by writing a joint probability $p(x_t, x_{t-1})$ and summing-out over x_{t-1}.

$$\sum_{x_{t-1}} p(x_t, x_{t-1} \mid z_{1:t-1}, u_{1:t}) = \sum_{x_{t-1}} p(x_t \mid x_{t-1}, z_{1:t-1}, u_{1:t}) . p(x_{t-1} \mid z_{1:t-1}, u_{1:t}) \tag{2.8}$$

For the given parent node x_{t-1} and control data u_t the prediction of the state is independent from previous sensor measurements $z_{1:t-1}$ and previous control data $u_{1:t-1}$. Figure 2-8 shows this independence. In Figure 2-8 the given nodes are green and removed edges are red.

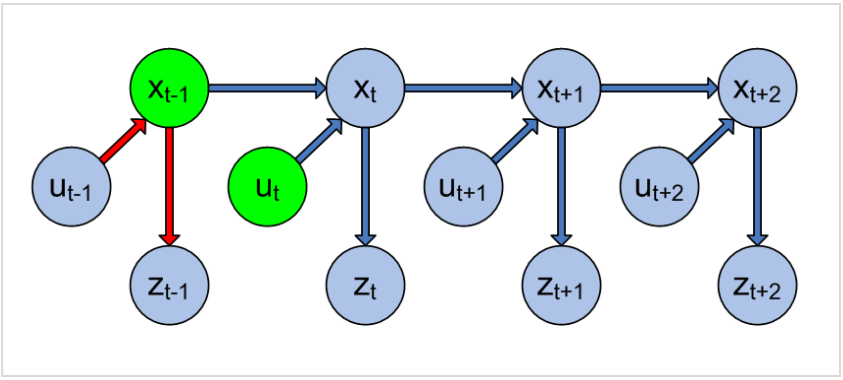


Figure 2-8 Independence from the past [1].

Then as shown in Figure 2-8 the first term of equation 2.8 becomes,

$$p(x_t \mid x_{t-1}, z_{1:t-1}, u_{1:t}) = p(x_t \mid x_{t-1}, u_t) \tag{2.9}$$

Also, the previous state vector x_{t-1} is independent from the current control data u_t while the current state x_t is not known. The yellow nodes in Figure 2-9 are independent from each other but their equal child node is not known. If the child node is known, independence will be removed. However, since the vehicle state at time t is not known it can be found using the past state and control data.

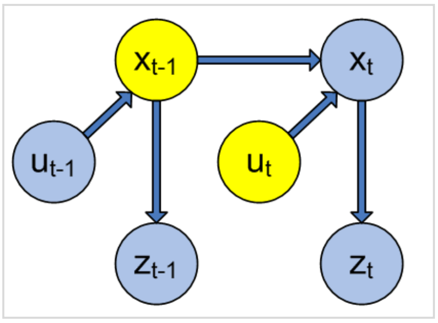


Figure 2-9 Independence of parent nodes [1].

Using independence in the second term of the prediction equation becomes:

$$p(x_{t-1} \mid z_{1:t-1}, u_{1:t}) = p(x_{t-1} \mid z_{1:t-1}, u_{1:t-1}) \tag{2.10}$$

Then final prediction equation can be written as below using equations 2.9 and 2.10:

$$\overline{bel}(x_t) = \sum_{x_{t-1}} p(x_t \mid x_{t-1}, u_t).p(x_{t-1} \mid z_{1:t-1}, u_{1:t-1}) \tag{2.11}$$

According to equation 2.11 **prediction** depends on the prior belief and motion model.

Now the belief in terms of prediction as follows;

$$bel(x_t) = \eta.p(z_t \mid x_t)\,\overline{bel}(x_t) \tag{2.12}$$

The Belief depends on the prediction (*prior belief, motion model*) and sensor measurements. This second step *(finding belief)* is the correction step. The full Bayesian algorithm is shown below.

```
1:  $\boldsymbol{Algorithm\ Bayes\_filter}(bel(x_{t-1}), u_t, z_t)$:
2:     for all $x_t$ do
3:         $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1})\, dx$
4:         $bel(x_t) = \eta\, p(z_t|x_t)\overline{bel}(x_t)$
5:     end for
6:     return $bel(x_t)$
```

Algorithm 1 Bayesian Filter algorithm [1]

2.5 Kalman Filter

The Kalman filter is a type of Gaussian filter which constitute the earliest tractable implementations of the Bayesian filter for continuous space. According to Gaussian techniques beliefs are represented by multivariate normal distributions. The probability of any state in Gaussian can be calculated by this formula:

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}}. e^{\left\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right\}} \qquad (2.13)$$

The density over the state x is characterized by two parameters μ (*mean value*) and Σ (*symmetric and positive semi-definite quadratic matrix*). The number of elements in the covariance matrix depends quadratically on the number of elements in the state vector. Gaussians have a single maximum and are appropriate for robotic implementations. In robotics, the true posterior is focused around the true state with a small margin of uncertainty [1].

Kalman filters implement belief computations for continuous states and this computation is not appropriate for discrete or hybrid spaces. This computed belief is shown with the mean and covariance(μ_t, Σ_t) at time t.

There are 3 important preconditions for Kalman filter SLAM applications. Linear transition model, linear measurement model and initial Gaussian belief are these three preconditions. These three properties guarantee that a distribution at any time will be a Gaussian.

2.5.1 Linear State Transition Model

A state transition model $p(x_t \mid x_{t-1}, u_t)$ must be a linear function. In matrix form it is:

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t \tag{2.14}$$

*(**x**: nx1 state vector, **u**: mx1 control vector, **A**: nxn matrix, **B**: nxm matrix, $\boldsymbol{\varepsilon_t}$: nx1matrix)*

ε_t is a random variable which models the uncertainty introduced by the state transition. Its mean is zero and covariance is R_t .

Then, the mean of the posterior is given by $\boldsymbol{A_t x_{t-1}} + \boldsymbol{B_t u_t}$ and the covariance byR_t.

$$p(x_t|u_t, x_{t-1}) = \det(2\pi R_t)^{-\frac{1}{2}}.e^{\left\{-\frac{1}{2}(x_t - \boldsymbol{A_t x_{t-1}} - \boldsymbol{B_t u_t})^T {R_t}^{-1}(x_t - \boldsymbol{A_t x_{t-1}} - \boldsymbol{B_t u_t})\right\}} \tag{2.15}$$

If the state transition model is not linear, then the posterior becomes nonlinear distribution and prediction fails.

2.5.2 Linear Measurement Model

The measurement model $p(z_t|x_t)$ must be a linear function. In matrix form it can be written as:

$$z_t = C_t x_t + \delta_t \tag{2.16}$$

($\boldsymbol{C} : k \times n$ Matrix, $\boldsymbol{\delta} : k \times 1$ measurement noise)

δ_t describes the measurement noise and its mean is zero and its covariance is Q_t .

Then the measurement probability is:

$$p(z_t|x_t) = \det(2\pi Q_t)^{-\frac{1}{2}}.\, e^{\left\{-\frac{1}{2}(z_t - \boldsymbol{C_t x_t})^T Q_t^{-1}(z_t - \boldsymbol{C_t x_t})\right\}} \tag{2.17}$$

If the measurement model is not linear, the algorithm defines detected landmarks with nonlinear distributions. So, the update equations fail for the landmarks in the remaining process.

2.5.3 Normally Gaussian Initial Belief

The initial belief $N(\mu_0\,,\Sigma_0)$ must be normally Gaussian.

$$bel(x_0) = p(x_0) = \det(2\pi\Sigma_0)^{-\frac{1}{2}}.\, e^{\left\{-\frac{1}{2}(x_0 - \boldsymbol{\mu_0})^T \Sigma_0^{-1}(x_0 - \boldsymbol{\mu_0})\right\}} \tag{2.18}$$

If the initial belief is not Gaussian the posterior will not be the Gaussian even if the state transition model and measurement model are linear.

2.6 Extended Kalman Filter

The Kalman Filter is applicable to SLAM problem when the initial belief is Gaussian and has linear transition and measurement models. In the real world state transitions and sensor measurements are rarely linear and the algorithm should take into account these nonlinearities and this method is called as Extended Kalman Filter (EKF).

The EKF formulation of SLAM was first introduced by Smith, Self and Cheeseman in 1990. The world is represented with landmarks and, covariances have been approximated by Gaussian distributions because of linearization [31].

The SLAM algorithm handles these nonlinearities through the Taylor Expansion. On the other hand, we can find the distribution of the vehicle state with Monte Carlo sampling (*i.e. 50000 samples*) which gives more realistic approximation than linearization by Taylor expansion. However, Taylor expansion is preferred because of the time efficiency.

Linearization approximates the nonlinear function g to the linear function that is tangent to g at the mean of the Gaussian. Approximation with Taylor expansion causes a linearization error. The linear approximation to a function g, is found using g's value and slope. The slope comes from partial derivative.

$$g'(u_t, x_{t-1}) = \frac{\partial g(u_t, x_{t-1})}{\partial x_{t-1}} \tag{2.19}$$

Both the value of g and its slope depends on the argument of g. The logical choice is to select the most likely value at the time of linearization. The most likely state is the mean value μ_{t-1} for Gaussian distributions. So the g function is approximated by its value at μ_{t-1}. The general linearization equation is given below.

$$y = f(a) + f'(a)(x - a) \tag{2.20}$$

In this work we applied the Taylor expansion at both the prediction and correction steps. The linearization formulas for the prediction step for velocity motion model are given in the equations below.

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}(x_{t-1} - \mu_{t-1}) \tag{2.21}$$

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1}) \tag{2.22}$$

G_t is a *nxn* size matrix *(n is dimension of state)* and called the **Jacobian matrix**. The same linearization is applied at the correction step to the h function around μ.

$$h(x_t) \approx h(\bar{\mu}_t) + \frac{\partial h(\bar{u}_t)}{\partial x_t}(x_t - \bar{\mu}_t) \tag{2.23}$$

$$h(x_t) \approx h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t) \tag{2.24}$$

EKF based approaches have suffered from the performance-limiting issues of linearization, computational problems and Gaussian assumption [11].

The main purpose of the EKF is not to calculate the exact posterior. Instead, it focuses on efficiently estimating the approximate mean and the covariance with some acceptable errors. There are many successful EKF-SLAM applications.

2.7 FastSLAM (Factored Solution to SLAM)

The FastSLAM algorithm uses a particle filter approach which is a type of non-parametric filter and an alternative to Gaussian filters. Non-parametric filters approximate posteriors by a finite number of values instead of a fixed functional form. Each approximation roughly corresponds to a region in the state space. The quality of the approximation depends on the number of parameters used to represent the posterior [1].

FastSLAM calculates the belief update in constant time, while the EKF-SLAM requires quadratic time. The total operation for updating all landmark states takes ***O(M.K)*** time for K landmarks and M particles [13]. The EKF-SLAM approach is not appropriate if there are large numbers of features *(i.e. 50.000)* in the environment, but fastSLAM can handle this situation and the algorithm converges to the true state vector.

The particle filter approach represents the posteriors by samples and ingredients of every sample are shown in Figure 2-10 for 3D SLAM.

We have **K samples** for any state in time and every sample has **1 weight**, **1 vehicle state** and **N landmark location estimates** *(μ and Σ).*

$$X_t^{[k]}, w_t^{[k]}, \quad \mu_{t,1}^{[k]}, \dots, \mu_{t,N}^{[k]}, \quad \textstyle\sum_{t,1}^{[k]}, \dots, \sum_{t,N}^{[k]}, \tag{2.25}$$

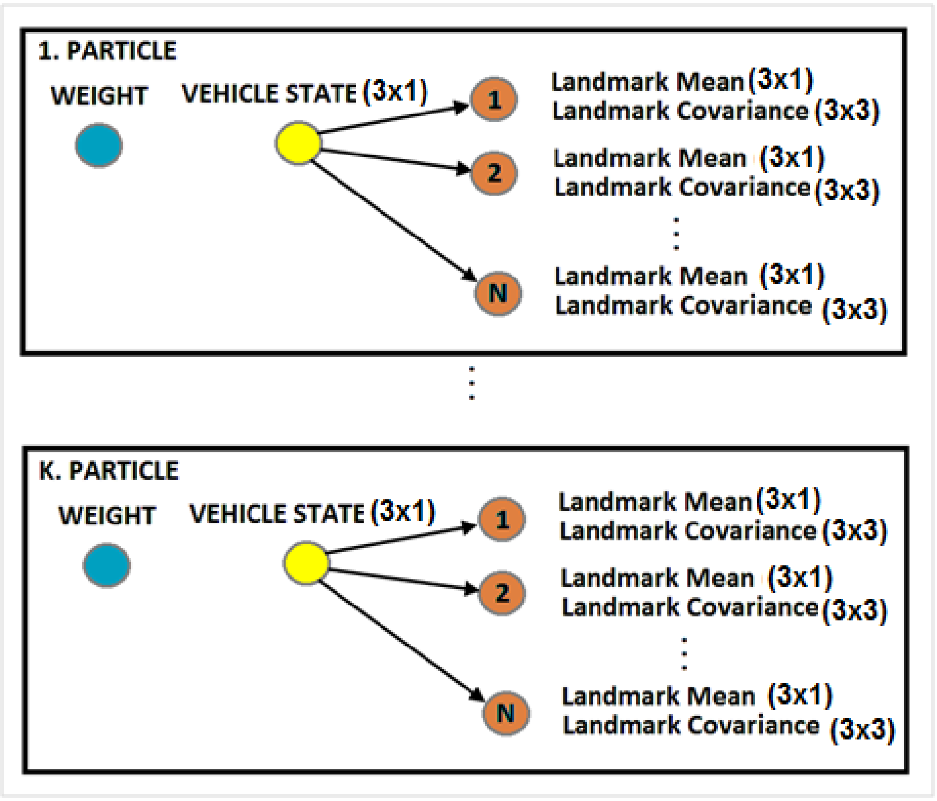


Figure 2-10 Particle representation for planar 3D SLAM.

The FastSLAM algorithm is quite easy to implement. Sampling from the motion model *(i.e. velocity motion model or odometer motion model)* and calculation of the importance weight is also straight-forward [13].
The other important properties and steps of particle filters and the fastSLAM algorithm are listed below.

2.7.1 Prediction of Vehicle State

An intelligent agent *(robot)* makes prediction for all the state vectors of a particle set when it obtains the control data *(linear velocity and angular velocity or odometer data)*. This prediction step includes the total **K** *(number of particle)* prediction.
In the prediction process the intelligent agent does not directly use the noisy control data. Instead, the agent takes samples from the Gaussian space of the control data and then uses this data to make predictions. Each particle represents a probabilistic guess of the robot path at time t and this feature is called a **multi-hypotheses** [13].

$$x_t^{[k]} \sim p\ (x_t | x_{t-1}^{[k]}, u_t) \tag{2.26}$$

This vehicle state calculation is easy if the motion model of vehicle is known (*i.e. the velocity motion model or odometer motion model as in EKF*).

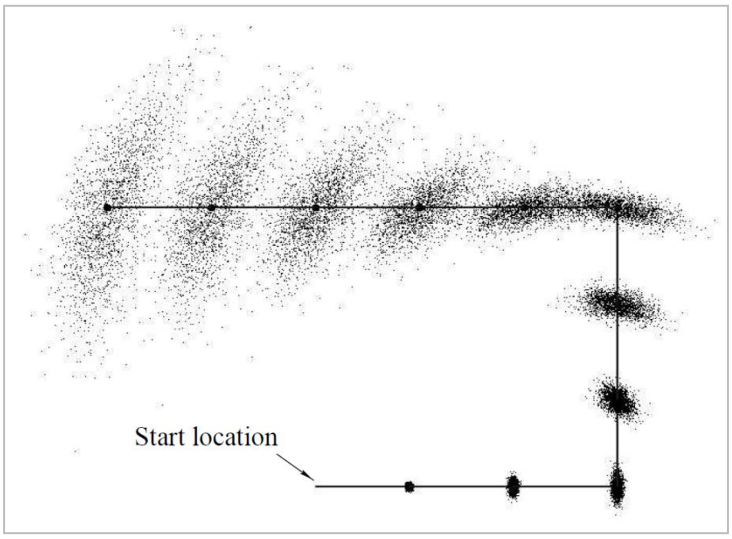


Figure 2-11Prediction step [1].

Figure 2-11 presents the sampling approximation of the position belief for a non-sensing robot. Without sensing there is no belief update through the navigation. All the particles represent different predictions [1].

2.7.2 Landmark Location Estimation and Update

The FastSLAM algorithm represents conditional landmark estimates using Kalman Filters *(EKF)*. Each landmark is defined with a Gaussian distribution $\mu_{t,n}^{[k]}, \Sigma_{t,n}^{[k]}$. Through the navigation process the algorithm makes linearization with Taylor expansion as in EKF because of the nonlinearities of the sensor measurement.

$$p(\mu_{t,n}^{[k]}, \Sigma_{t,n}^{[k]} \mid x_t^{[k]}, z_t^{[k]}, u_t^{[k]}) \tag{2.27}$$

$\mu_{t,n}^{[k]}$ defines the mean and $\Sigma_{t,n}^{[k]}$ defines the covariance of landmarks at time **t** for $\boldsymbol{k^{th}}$ particle [13]. For the 3D robot navigation scenario each mean $\mu_t^{[k]}$is a three-element vector, and $\Sigma_t^{[k]}$ is a 3x3 matrix.

In the update step the vehicle updates feature state vectors using sensor measurements. The update step is the same as the EKF update sequence. The algorithm updates the mean of the landmark locations and their covariances. This estimation and update step evaluates the pose of the landmarks for all particles *(for every different pose estimate of vehicle)* [13].

2.7.3 Weight Calculation

The algorithm computes the probability of the sensor measurement z_t for each particle. If the index of the sensed landmark is n, then the probability of the sensor measurement can be defined as,

$$w_t^{[k]} := \mathcal{N}\left(z_t; \middle| x_t^{[k]}, \mu_{t,n}^{[k]}, \Sigma_{t,n}^{[k]}\right) \tag{2.28}$$

$w_t^{[k]}$ is the importance weight, which shows how important the particle is. The intelligent agent computes the importance weight according to the sensor measurement and the expected sensor measurement. The importance weights of all particles are normalized and their sum is 1. These importance weights will be effective in the re-sampling step. Calculated weights became meaningless if there is not any re-sampling step.

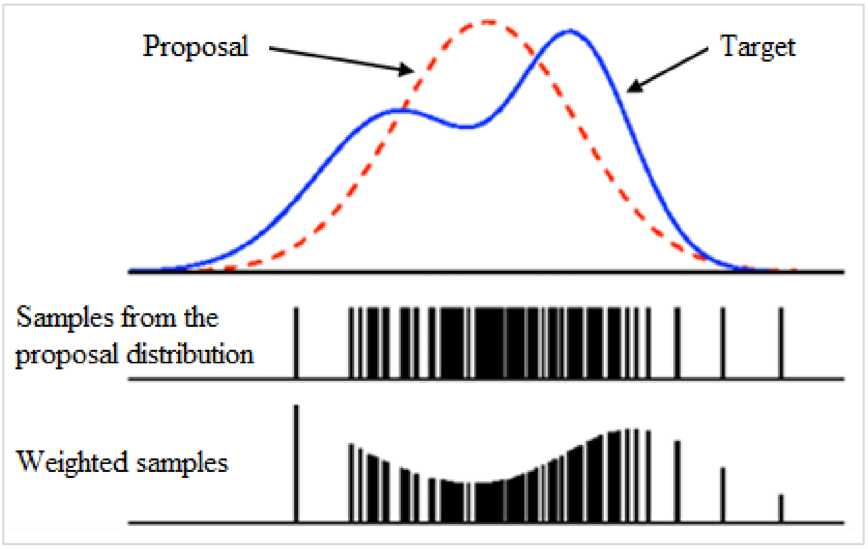


Figure 2-12 Weight representation [1].

In Figures 2-12, samples were taken from proposal distribution which is a Gaussian distribution, however, the real distribution (target) is not Gaussian and it is nonlinear. In Figure 2-12 the lengths of the bars represent the importance weights.

2.7.4 Re-sampling Process

The probability of drawing a particle concerns its normalized importance weight. The intuition behind this process is simple; the low weighted particles are deleted and systematically a successful particle is copied in place of the deleted particle. This property obtains information passing to the backward as indirectly [13].

The re-sampling process is shown Figure 2-13. M defines the total number of particles and first the r value is defined. The r value can be selected randomly but the algorithms used in this thesis are defined with formula 2.29. The aim was to achieve the middle of the weight value if all the weights are equal.

$$r = \frac{1}{2\,.NUMBER\ OF\ PARTICLES} \tag{2.29}$$

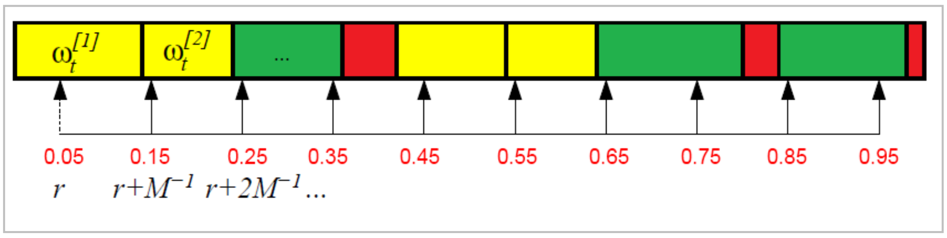


Figure 2-13 The re-sampling process [1].

According to Figure 2-13, the red particles (4th, 8th and 10th) are deleted and green particles are copied instead of the red particles. The algorithm takes the 3rd, 7th and 9th particles instead of the 4th, 8th and 10th particles.

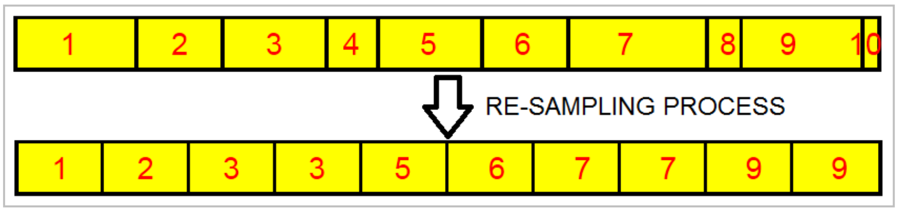


Figure 2-14 Before and after the re-sampling process [1].

2.7.5 Rao-Blackwellization

The joint probability distribution $p(a, b)$ is found for the arbitrary random variables a and b. However, if the conditional probability $p(b|a)$ can be described in closed form it is equally legitimate to only draw the particles from $p(a)$. The algorithm attaches to each particle a closed form description of $p(b|a)$.

This trick is called as **Rao-Blackwellization** and it yields better results than sampling from the joint. fastSLAM applies this technique and samples from the path posterior $p(x_t^{[k]} \,|u_t, z_t\,)$ and represents the map $p(m \mid x_t^{[k]} \mid u_t, z_t)$ in Gaussian form [13].

2.7.6 Factored Representation

In the fastSLAM application the problem is to find the map and vehicle pose.

$$p\ \left(x_t^{[k]}, m \middle| u_t, z_t \right) \tag{2.30}$$

The posterior can be factored as in equation 2.31.

$$p(x_t^{[k]}, m \mid u_t, z_t) \ = \ p(x_t^{[k]} \mid u_t, z_t) \ \prod_k p(m_k \,|x_t^{[k]}, u_t, z_t\,) \tag{2.31}$$

With this factorization the problem is decomposed to K+1estimation problems. One problem is estimating the posterior for robot paths and K problems for estimating the K landmarks conditioned on the path estimate [13].

2.7.7 Conditional Independence

In fastSLAM, knowledge of the robot path renders all landmark estimates independent. The location variables separate the individual features in the map from each other. If the state vector is a known variable, all the landmark estimates become independent from each other. Any dependence between two landmark estimates is mediated through the robot path [13].

The algorithm estimates a robot path with particle filter (PF) and estimates landmark positions with EKF. EKF is applied to every landmark for each particle and all landmarks defined with Gaussian distribution. For M particles and K landmarks the problem includes the **K.M** EKF calculation**.** This EKF calculation is in dimension 3 and does not grow as in EKF-SLAM.

CHAPTER 3

DATASET AND SIMULATION ENVIRONMENT

3.1 RGBD SLAM Kinect Dataset

Part of the large RGBD SLAM dataset was used to verify the proposed ideas during the research. The dataset was generated by Sturm et al. with a Microsoft Xbox Kinect Sensor [6].

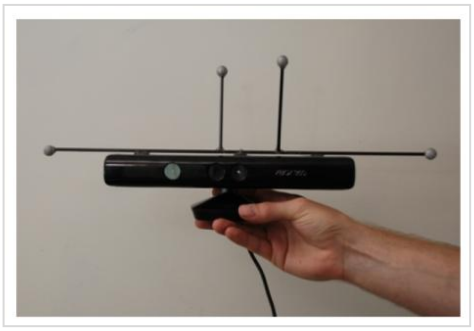


Figure 3-1 Microsoft Xbox Kinect Sensor with Reflection Sensors [6].

The Kinect sensor consists of a near-infrared laser, an infrared camera and a color camera between them. The near-infrared laser projects a refraction pattern on the scene and an infrared camera observes this pattern. Using block matching techniques it is possible to compute the disparity if the projected pattern is known. All the image rectification and block matching happens internally in the sensor [6].

The Kinect sensor has advantages and disadvantages in relation to the laser scanners. The technical specifications and performances of these sensors are compared for the map building, localization and obstacle avoidance using only one of the 480 lines of the Kinect output for the 2D SLAM simulation [32]. The technical comparison of the sensors is given in Table 3-1.

As shown in the Table 3-1, the SICK laser scanner gives a higher performance than the other two and also it is configurable *(range, resolution, measurement angle)* for specific applications. On the other hand, the cost of the SICK laser scanner is very high in relation to the two scanners. Taking into these properties, the Kinect sensor and Hokuyo laser scanner are more comparable [32]. However, the important difference between Kinect and Hoyuko concerns dead zones. The Kinect sensor could not separate the infrared dots on close obstacles due to blooming effects [32].

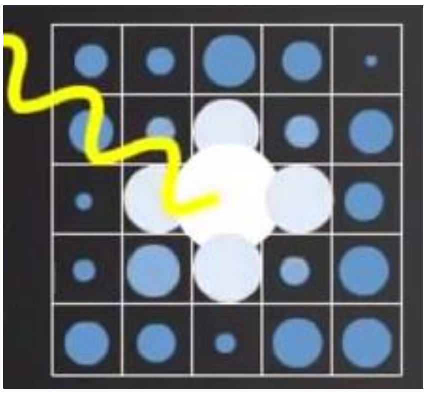


Figure 3-2 Visualization of blooming effect for a CCD camera [33].

Figure 3-2 shows the blooming effect on a CCD camera created by a high charge in one area which can influence the pixels next to it. In Kinect cameras this blooming effect makes the boundaries noisy [33].

Another important feature, is that accuracy is better with the Kinect sensor when the distance is less than 1.5 meters. Above this distance the Hokuyo laser scanners give better results [32]. The depth resolution of the Kinect sensor varies between 0.25 cm *(d=0.8 m)* and 4.8 cm *(d=4 m)*. The minimum distance was fixed at 0.8/0.4 meters but the maximum distance is not fixed. Different authors recommend rejecting distances of more than 3 meters or 4.6 meters [32].

Kinect produces a gap in the point cloud data if camera monitors a surface that cannot be scanned by the infrared pattern. Also, a contrary case produces the same result. This is called a parallax problem and is amplified by reflective surfaces. However, since laser scanners combine the receiver and transmitter the parallax problem does not occur [32].

Both sensors have problem in perceiving reflective surfaces, dark areas and transparent materials. The material of the obstacles has important effect on the perception of data [32].

The most important disadvantage is the smaller monitoring angle of the Kinect sensor in terms of laser scanner systems. This small view angle limits the capabilities of mapping and localization. On the other hand, Kinect is more reliable in obstacle detection than the Hokuyo laser scanner [32].

Table 3-1 Technical specification comparison of sensors [32].

	SENSORS		
	Kinect	Hoyuko	SICK
Maximum range [m]	3-6	4	8-80
Dead range [m]	0.8/0.4	0.06	0.07
Horizontal angle [°]	57	240	100-180
Distance resolution [mm]	2.5-48	1	1-10
Angular resolution [°]	≈0.097	0.3515	0.25-1
Accuracy [mm]	±6 (1m) ±130 (4m)	±30 (1m) ±120 (4m)	±10 (10m)
Geometry [mm]	65x290x70	50x50x70	155x156x210
Weight [kg]	0.55	0.16	4.5
Power voltage [V]	12	5	24
Power consumption [W]	5	4	30
Refresh rate [Hz]	30	10	18-75
Output Data [kB/s]	18000	5.4	500
Interfaces	USB	USB	RS-232 RS-422
approx. Costs $	150	1000	5000

The Kinect sensor is still good choice for indoor SLAM applications with fast 3D data perceiving and low cost [32].

The RGBD Kinect dataset consists of two different environments, a typical office *(fr1)* and large industrial workspace *(fr2)*. The latter is appropriate for robot-SLAM applications [6].

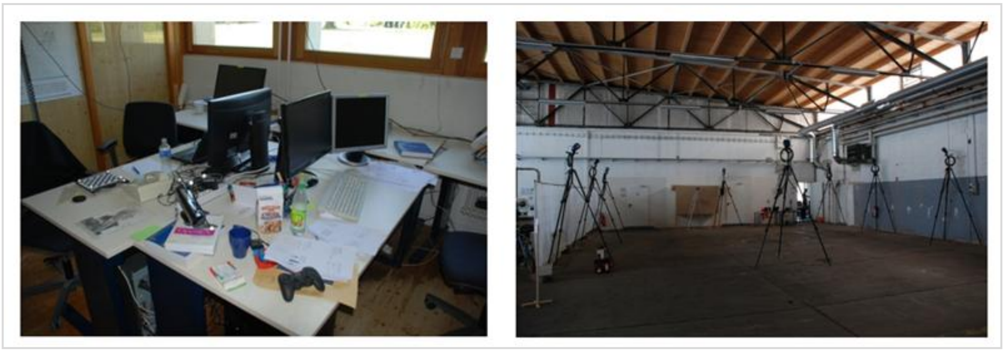


Figure 3-3 Typical office environment and large industrial workspace. [34]

Available robot SLAM sequences are shown in Table 3-2.

Table 3-2 List of available RGB-D SLAM sequences [34]

Sequence Name	Duration [s]	Avg. Trans. Vel. [m/s]	Avg. Rot. Vel.
fr2/pioneer_360	73	0.23	12.05
fr2/pioneer_slam	156	0.26	13.38
fr2/pioneer_slam2	116	0.19	12.21
fr2/pioneer_slam3	112	0.16	12.34

Table 3-2, gives the sequences and their basic information *(duration, average angular and translational velocity)*. The Kinect sensor is mounted on a Pioneer 3 robot which is controlled manually with joystick for the robot-SLAM sequences (Figure 3-4).

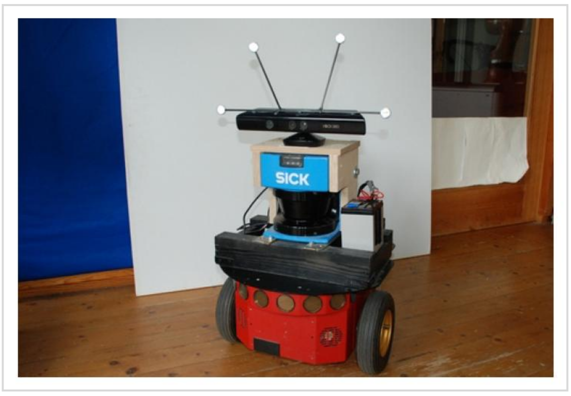


Figure 3-4 Pioneer r**obot** with Kinect sensor [34].

The ground truth of the dataset is created with 8 high speed tracking cameras *(Raptor-E from Motion Analysis)* working at 100 Hz [35].

Figure 3-5 Motion analysis Raptor-E capture cameras [34].

The dataset creates color *(8 bit RGB- each pixel value differs between 0-255)* and depth *(16 bit monochrome-each value differs between 0-65025)* images for every scan of the environment instead of point cloud data *(Point cloud data after conversion is nearly 20 GB).* The dataset contains a conversion algorithm to create point cloud data from the RGB and depth images. The frame rate of the dataset is 30 Hz and the sensor resolution is 640 x 480.The depth images are scaled to 5000. A pixel value of 5000 in the depth image corresponds to a distance of 1 meter from the camera [34]. The sequences include the ground truth data for the vehicle state. Each line in the ground truth data includes a **timestamp** *(number of seconds)* $\boldsymbol{t_x}, \boldsymbol{t_y}, \boldsymbol{t_z}$ *(position of the optical center of the color camera),* $\boldsymbol{q_x}, \boldsymbol{q_y}, \boldsymbol{q_z}, \boldsymbol{q_w}$ *(orientation of the optical center of color camera with respect to the motion capture system in the form of unit quaternions)* [34].

Calibration is another important point to consider for Kinect cameras. The default calibration values are not true values for each different Kinect camera. In Table 3-3, $\boldsymbol{f_x}, \boldsymbol{f_y}$ values define the focal length of the cameras, $\boldsymbol{c_x}, \boldsymbol{c_y}$ values define the optical center of the cameras and $\boldsymbol{d_s}$ value define the correction factor of the depth values [34].

Table 3-3 Calibration Parameters for the Freiburg 2 dataset [34].

Camera	f_x	f_y	c_x	c_y
Color	520.9	521.0	325.1	249.7
Infrared	580.8	581.8	308.8	253.0
Depth	d_s = 1.031			

In the current study, the **'freiburg2_pioner360'** and **'freiburg2_pioner_slam3'** datasets were used to verify the proposed ideas. These datasets are generated with **one complete turn** in a hall with a pioneer robot and satisfies the performance comparison after the loop-closing.

3.2 Real Time Application

The contribution of plane features for navigation (SLAM) are validated with the published dataset. After this validation the performance of algorithm is verified with a real time application with the pioneer robot and the Microsoft Kinect Sensor. Figure 3-6 shows the pioneer robot with a computer and a Kinect sensor.

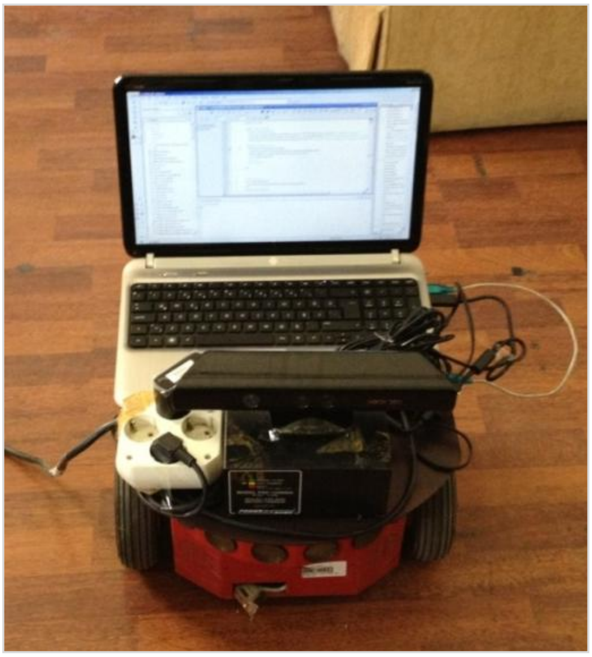


Figure 3-6 The robot system *(Image was taken in the METU Lab.)*

The computer communicates with the robot and sensor camera through different communication protocols. The first connection between the computer and the kinect sensor is through a USB cable. A RS-232 serial communication provides the second connection between computer and robot.

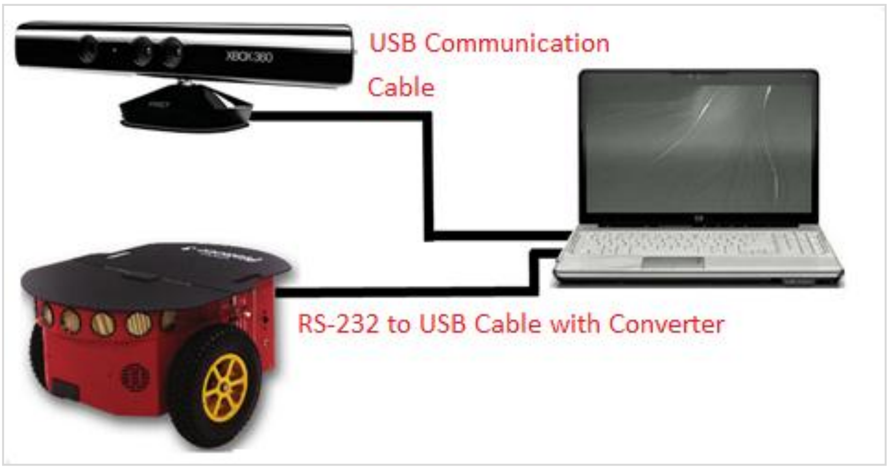


Figure 3-7 The robot system, computer and sensor connection

The computer controls the robot and satisfies the motion through the defined trajectory. The computer waits at some steps and takes sensor measurements throughout the motion.

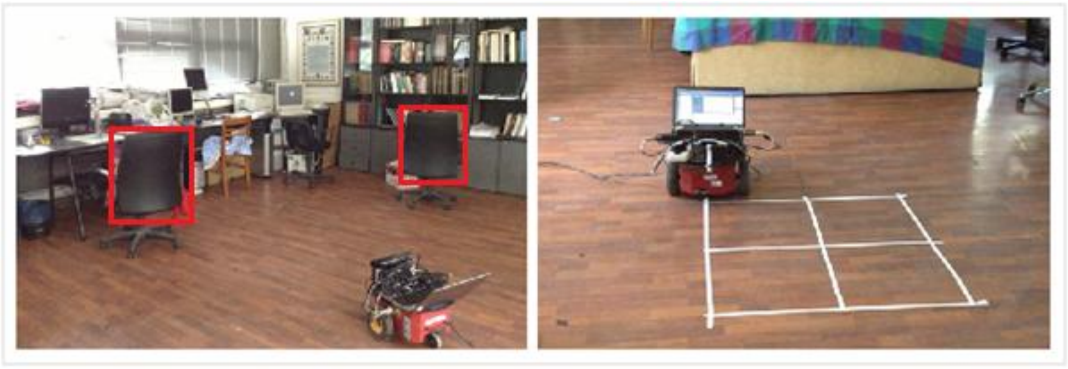


Figure 3-8 The robot system and map environment *(Images were taken in the METU Lab.)*

The navigation environment of the robot is the METU EE Computer Vision Laboratory. The backs of the chairs *(outlined in red)* are used as a landmark.

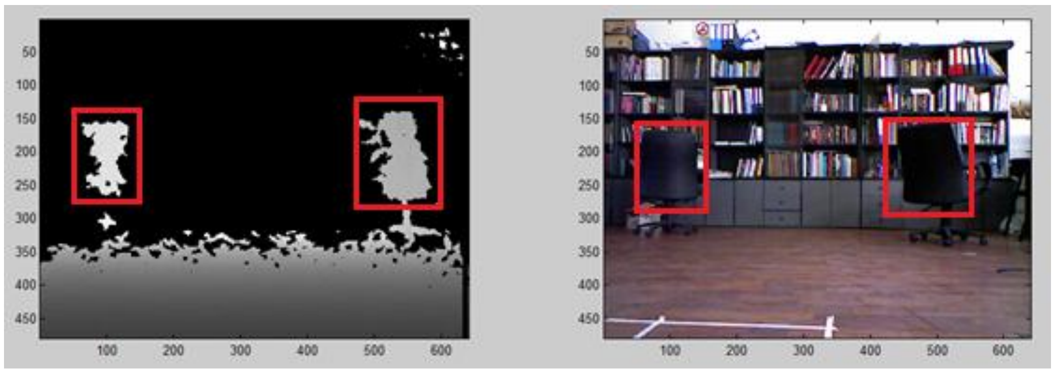


Figure 3-9 Sample depth and RGB image from real-time application *(Edited MATLAB images were taken in the METU Lab.)*

The original depth and RGB image are shown in Figure 3-9 and the extracted feature points (back of the chair) are shown in Figure 3-10. In the point cloud data the green points define the plane candidates and red points are the remaining ones.

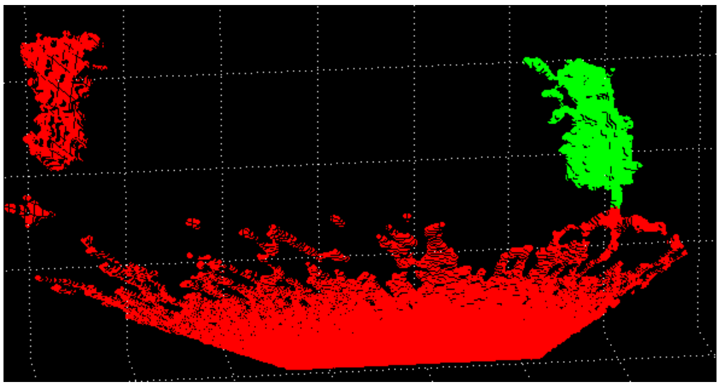


Figure 3-10 Sample 3D point cloud from a real time application *(MATLAB Image)*

The feature extraction methods are discussed in detail in chapter 4.

CHAPTER 4

FEATURE EXTRACTION

4.1 Filtering Process

Filtering process is an important step in the 3D SLAM algorithm. Every 3D scan of the environment contains nearly 307200 (640x480) points. The mathematical calculation takes too much time with this large number of points. To reduce the process time there should be a filtering process. Also, raw data can include noisy or unnecessary points for example the background or ground. To prevent erroneous data associations, robust data should be used and the foreground data should be extracted from the background. This process is explained in the following sections.

4.1.1 Sampling

3D point cloud data can include thousands of points and this extremely large number of points increases the process time. Sampling from raw data reduces the number of points and the process time. This sampling process reduces computation time while maintaining the accuracy of measurements as shown in [5]. In the figure 4-1 it can be seen that the number of points was decreased from 56000 to 2300 and process time of 96 hours was reduced to 5 seconds while maintaining a significant level of accuracy [5].

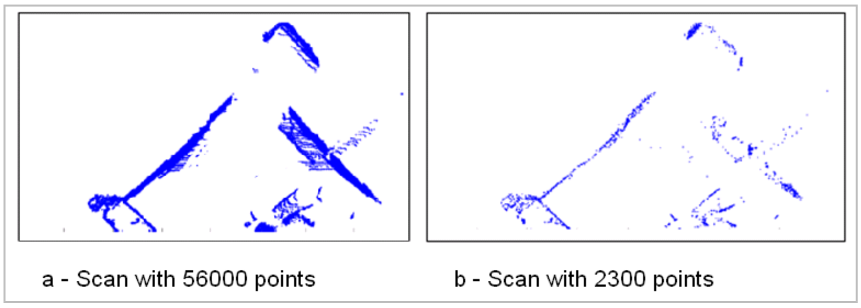


Figure 4-1 Original and sampled data [5].

Figure 4-2 shows the original and sampled versions of the raw data. Only 10% of original data can define the environment. Reducing the number of points decreases the computation time for the remaining algorithms.

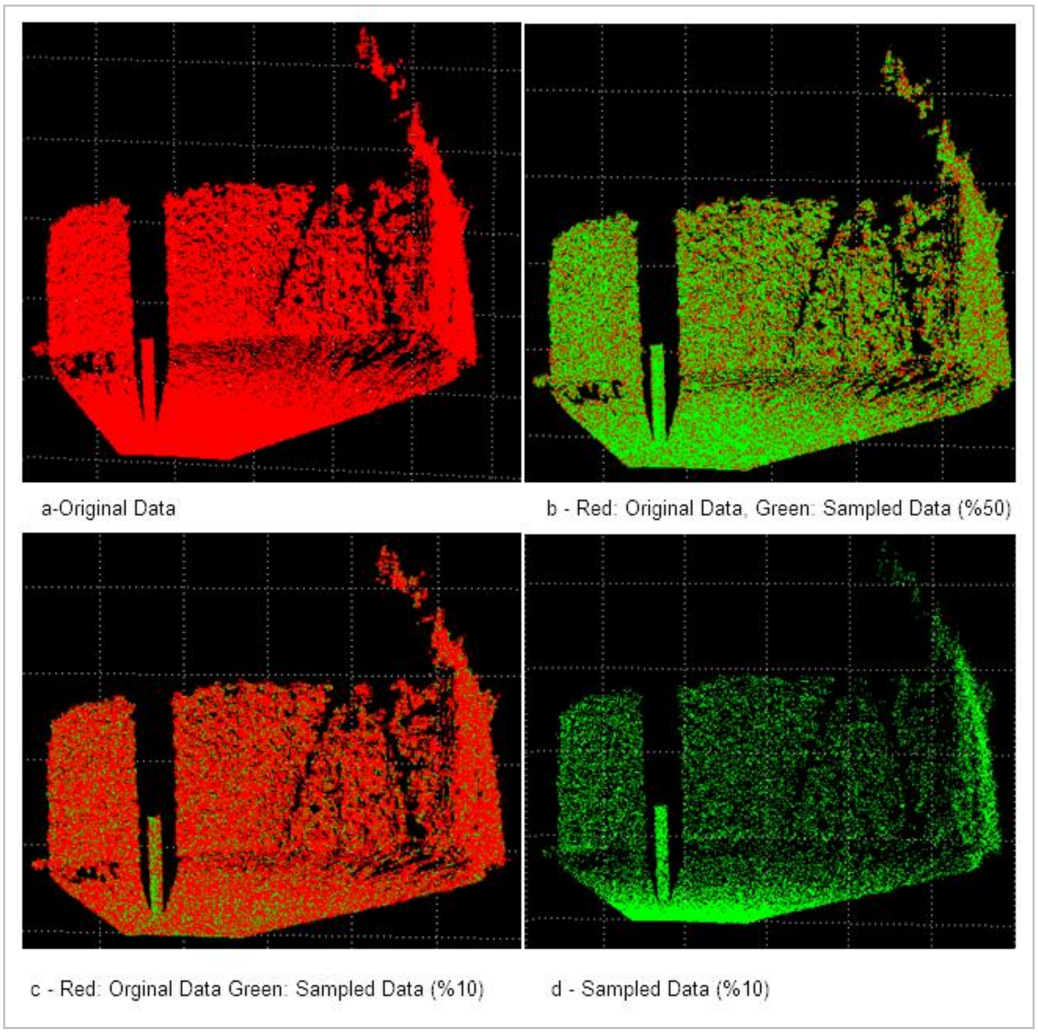

a-Original Data
b - Red: Original Data, Green: Sampled Data (%50)
c - Red: Orginal Data Green: Sampled Data (%10)
d - Sampled Data (%10)

Figure 4-2 Filtering results *(MATLAB Images)*

Through the sampling process there is a risk of losing the important part of the data however, the fastSLAM approach can handle this kind of problem.

The algorithm used in this study undertakes random sampling but, intelligent selection methods also exist to protect the important part of the point cloud data as in Chen's application. However, intelligent sampling comes at a computational price [5]. Chen uses an effective edge feature extraction method to extract the edge points and this reduces the redundant points [5]. The results are provided in Figures 4-3 and 4-4.

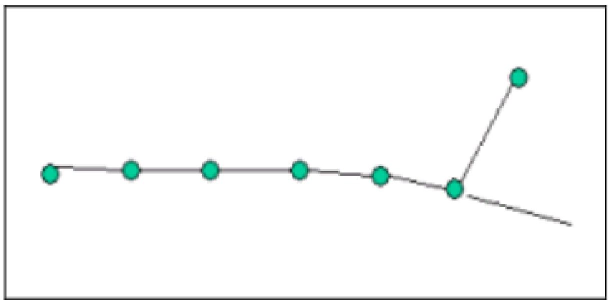


Figure 4-3 Original scan data points [5].

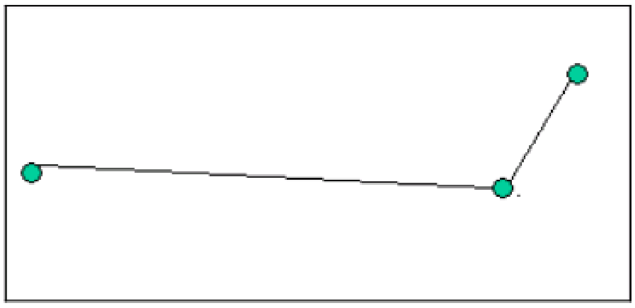


Figure 4-4 Data points after feature extraction filter [5].

After the removing process Chen defined the scan data with only 3 points instead of 7. The algorithm, detects the edge of objects and according to this edge information removes unnecessary points from the data. This process decreases the time consumption to the half for remaining processes [5].

4.1.2 Removing Background

All SLAM methods *(feature based and scan based)* suffer from data association errors. Incorrect sensor readings and observing the same features from different locations lead to errors in the resulting maps [36]. In current work, we removed background data and used only foreground data to prevent incorrect data associations. A foreground object can occlude portions of a background object and this results in abrupt boundaries and the suppression of those features that are close to these boundaries [16].

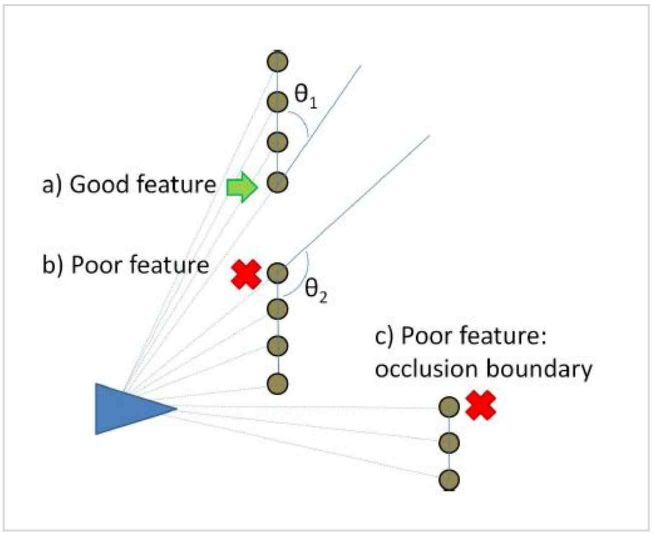


Figure 4-5 Poor features are suppressed. [16]

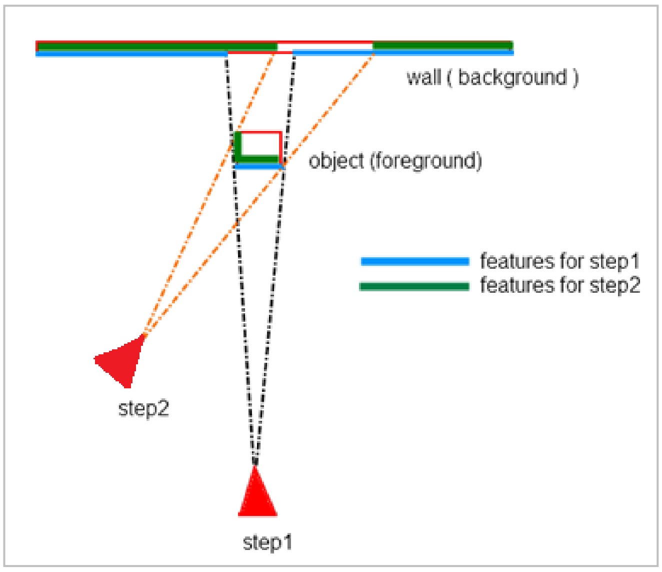


Figure 4-6 Removing the background to prevent false data association.

In Figure 4-6 the same object *(wall)* can be detected as different feature according to the view angle because of the occlusion by foreground object. This can result in a false data association and increases the final error in the SLAM application. The backgrounds of scenes are filtered according to the object sizes *(point count)* and this prevents false data associations.

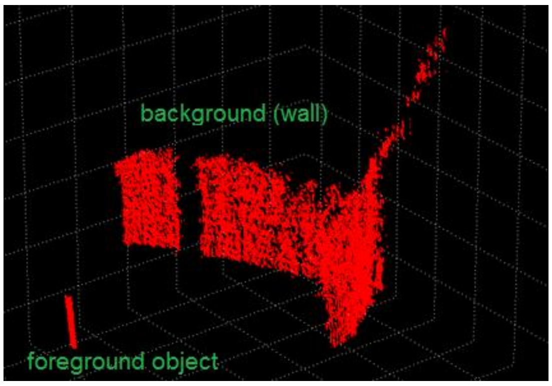


Figure 4-7 All data before background removal *(MATLAB Image).*

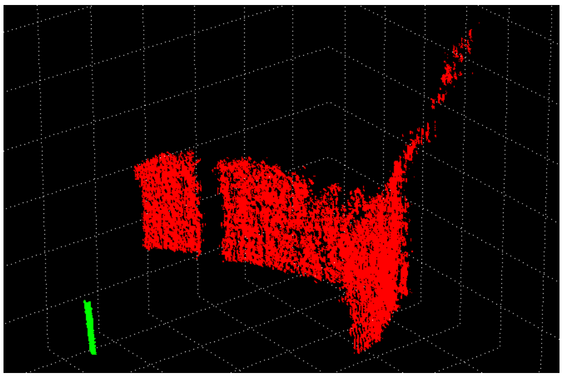


Figure 4-8 Green points are foreground and red points are background. *(MATLAB Image).*

Figure 4-7 and Figure 4-8 shows the background removal sample from the application. The foreground is easily and clearly separated from the background.

4.1.3 Filtering the Ground Floor

Horizontal plane segments are not essential for planar SLAM applications and can be discarded to eliminate the ground effect. Ground points cannot be modeled as distinctive features [37].

In the current application, in order to eliminate ground points, horizontal planes are detected and the points that fit these horizontal planes are filtered. These horizontal planes are detected according to the normal vector of randomly selected points. It is expected that the normal vector should be in the z direction. If the normal vector is not in the z direction, the algorithm iterates until it finds the normal vector that defines the ground points in the z direction. In fact, all the points were not selected randomly. Only the first point are selected randomly and remaining points are selected according to the Euclidian distance to the mean of selected points iteratively.

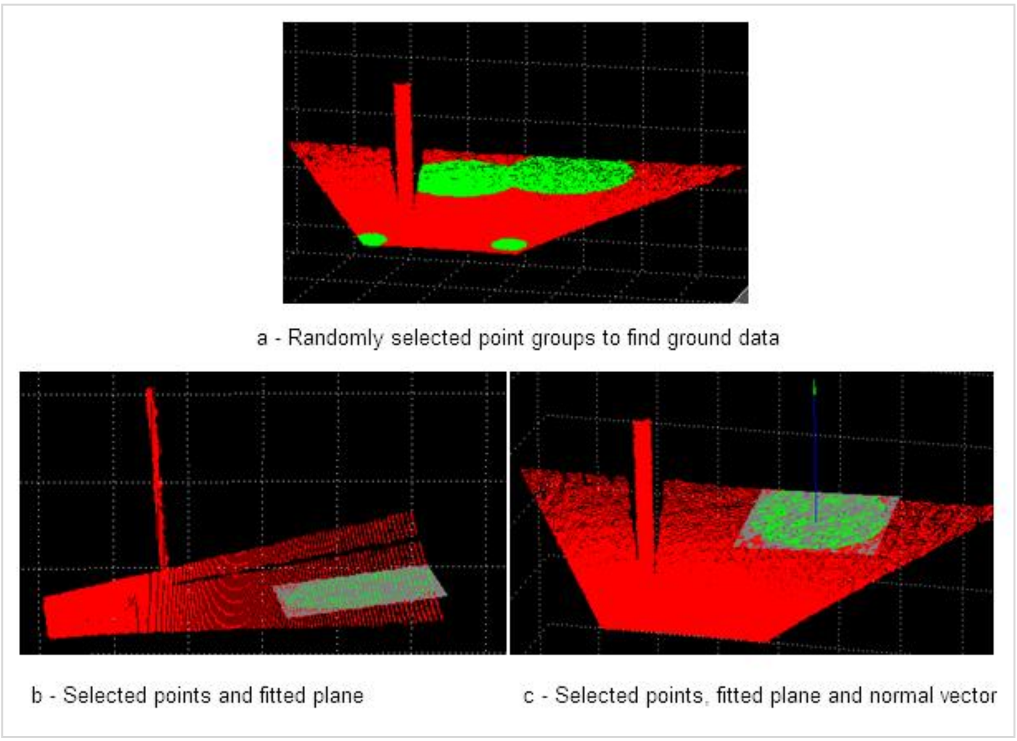

a - Randomly selected point groups to find ground data
b - Selected points and fitted plane
c - Selected points, fitted plane and normal vector

Figure 4-9 Filtering ground data *(MATLAB Images)*

Figure 4-9 (a) shows 4 different point groups. In our algorithms, different point groups were not found for iteration. The purpose is to show the different alternatives that help to find the ground points. The fitted plane is shown in (b) and in (c) the founded normal vector can be seen. The points that define the ground were cleared from the scan according to the normal vector information.

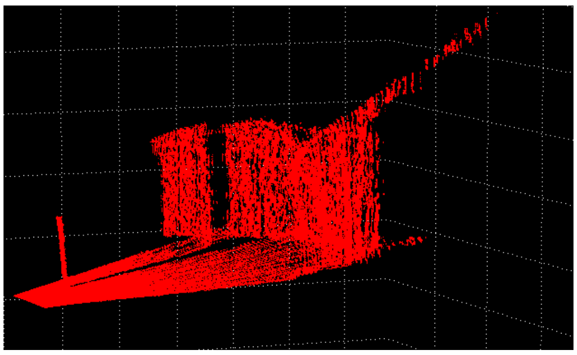


Figure 4-10 Raw scan data *(MATLAB Image)*

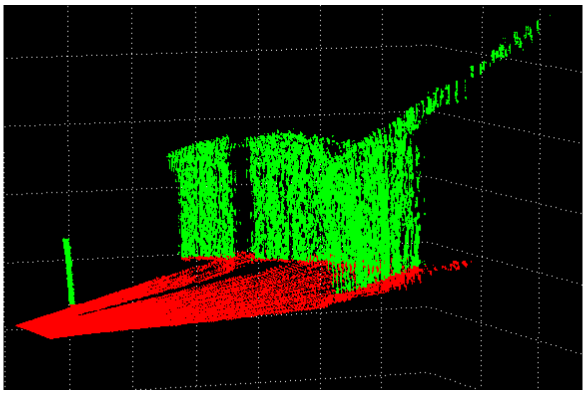


Figure 4-11 Filtering ground data. *(MATLAB Image)*

Figure 4-10 shows the raw scan data and Figure 4-11 shows the filtering. In Figure 4-11 the red points are assigned as ground points and eliminated. The remaining points are shown in green.

4.1.4 Hierarchical Clustering (Segmentation)

Removing the background and the ground floor data can be sufficient if the environment is not complicated. In Figure 4-12, the red points define the eliminated points and green points define the remaining ones.

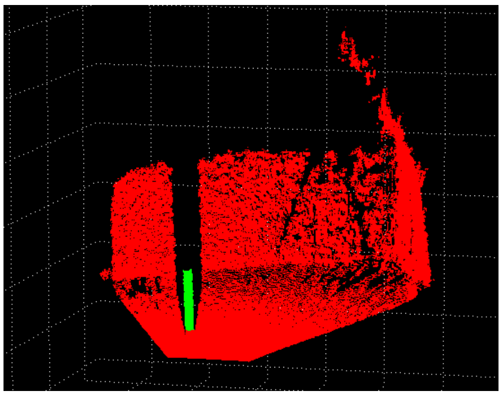


Figure 4-12 Raw data and final data after filtering *(MATLAB Image)*

The environment and the 3D scan data could be more complex. If there is more than one object plane fitting the process may fail. For this kind of scene, the algorithm splits the objects and then applies the plane fitting algorithm separately to all different point groups.

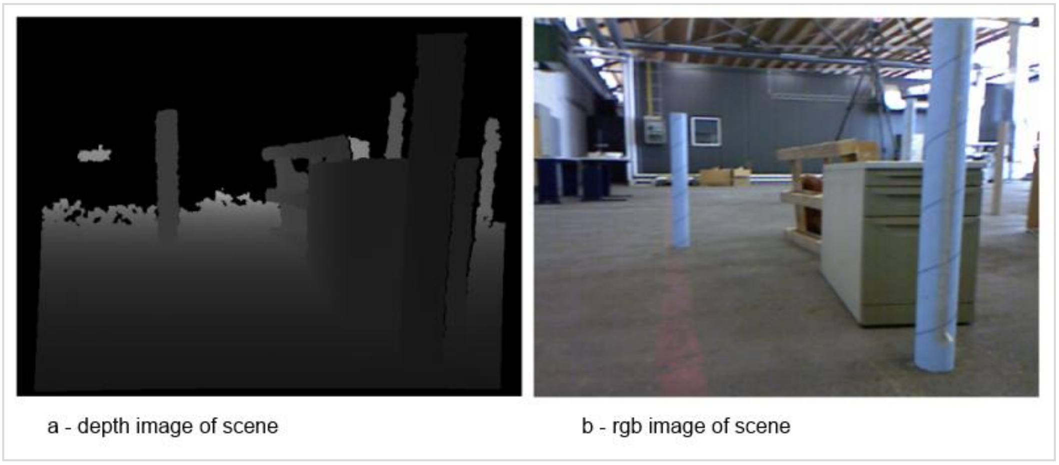


Figure 4-13 Complex 3D environment. *(Depth and RGB image for scan 281 from fre2_SLAM3)* [34].

Figure 4-12 shows the first process of filtering which is removing the ground. The red points are the removed ground floor points and green points are remaining points.

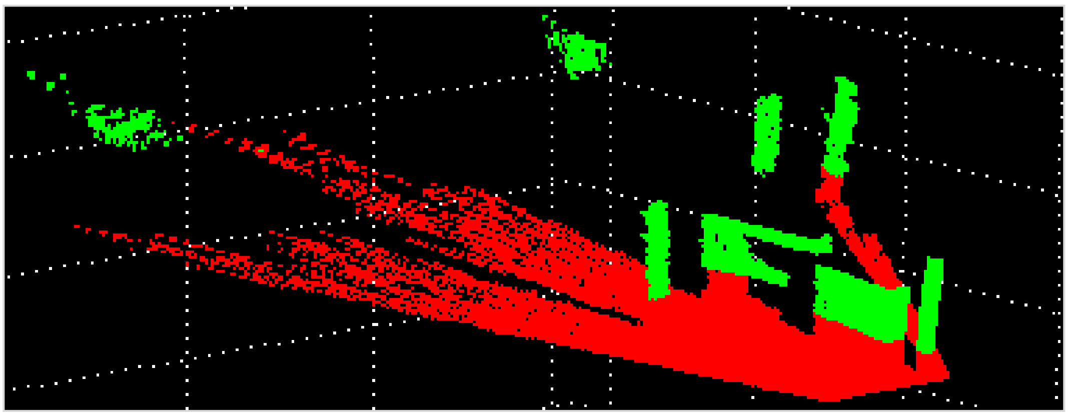


Figure 4-14 3D view of scan 281 of fre2_SLAM3 map. *(Green points are remaining parts after red points (ground points) removed) (MATLAB image)*

The algorithm can easily separate the point groups that define objects as shown in Figure 4-14 and Figure 4-15. The remarkable spatial distances between groups facilitate this easy separation.

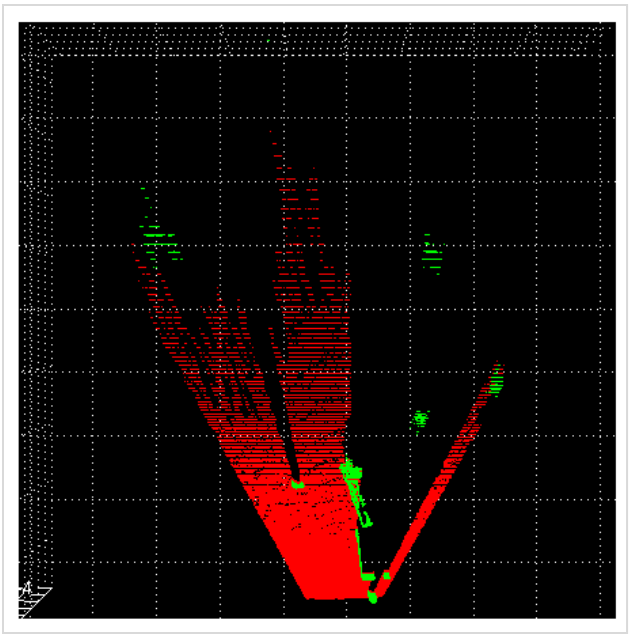


Figure 4-15 Birds eye view of 3D scan 281 *(MATLAB image).*

K-means clustering is a widespread clustering method but has some preconditions since it requires listed information.

- A number of clusters An initial assignment of data to clusters
- A distance measure between data $d(x, x')$

In the current research, the robot does not know the number of clusters for any scan and cannot make any initial assignment. Thus, the algorithm uses unsupervised clustering method. The selected unsupervised clustering method is hierarchical clustering. The hierarchical clustering method only requires a similarity function. Defining the similarity function is the most important element in the hierarchical clustering method. In 3D space objects are well separated from each other and the Euclidian distance function defines the satisfactory similarity function. Also, for hierarchical clustering, for each 3 points $(x, x', x")$ in the dataset the following properties should be satisfied [38].

- Non-negativity: $d(x, x') \geq 0$
- Reflexivity: $d(x, x') = 0 \; if \; and \; only \; if \; x = x'$
- Symmetry: $d(x, x') = d(x', x)$

- Triangle inequality: $d(x, x') + d(x', x'') \geq d(x, x'')$

In the method given in this thesis these properties are satisfied and robot can safely use hierarchical clustering method through the navigation.

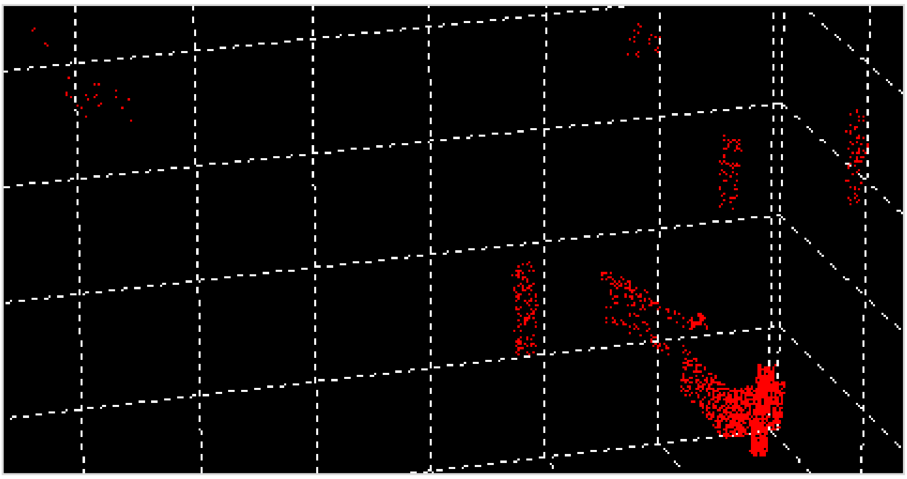


Figure 4-16 Randomly selected 2000 points for clustering *(MATLAB image)*

The number of points is very important for hierarchical clustering and before clustering, the algorithm randomly samples 2000 points from the point cloud data. The sampled data can be seen in Figure 4-16.

Hierarchical clustering according to spatial distance between the groups shown in Figure 4-17. This visualization is called dendogram which is a tree diagram that illustrates the hierarchical clustering process[38].

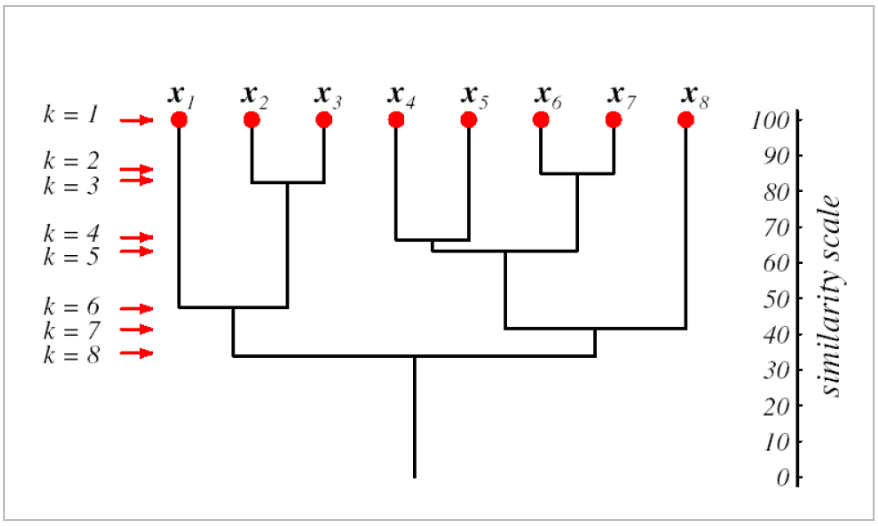


Figure 4-17 Dendogram representation of hierarchical clustering [38].

At the beginning of the clustering, at step 1 *(k=1)* all the points define a cluster thus the similarity scale is 100. After step 3 *(k=3)* the total number of clusters is 6 and the similarity decreases. The number of clusters and the similarity are inversely proportional. After step 8 *(k=8)* the total number of cluster becomes 1 and the similarity decreases to 30.

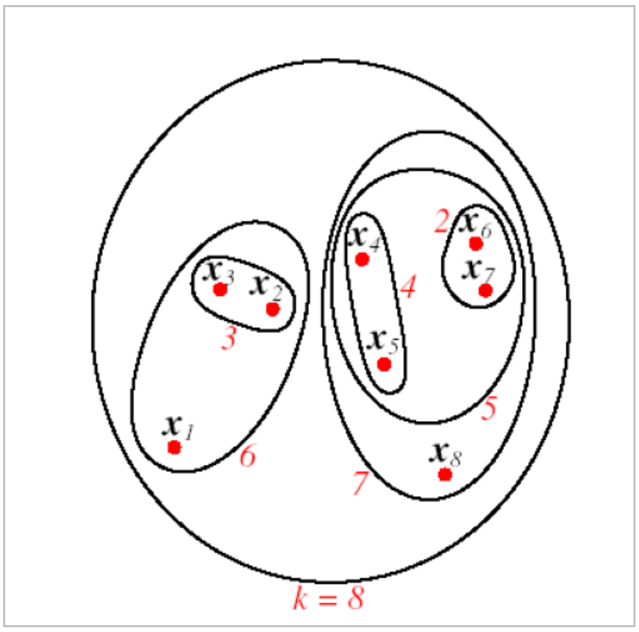


Figure 4-18 Alternative representation of the hierarchical clustering [38].

Understanding where to stop *(cut-off)* and criterion function are two important properties of hierarchical clustering. The criterion function of in this study algorithm is the Euclidian distance.

Cut-off value is important in order to split different objects *(plane segments)*. Figure 4-19 shows the dendogram visualization of the clustering process. For CUT_OFF = 3.0 meters, the algorithm only finds two clusters and the irrelevant point groups constitutes a cluster. On the other hand, selecting CUT_OFF = 0.01 results in hundreds of classes. The algorithm splits one object into many point groups and this causes erroneous feature detection and incorrect data associations.

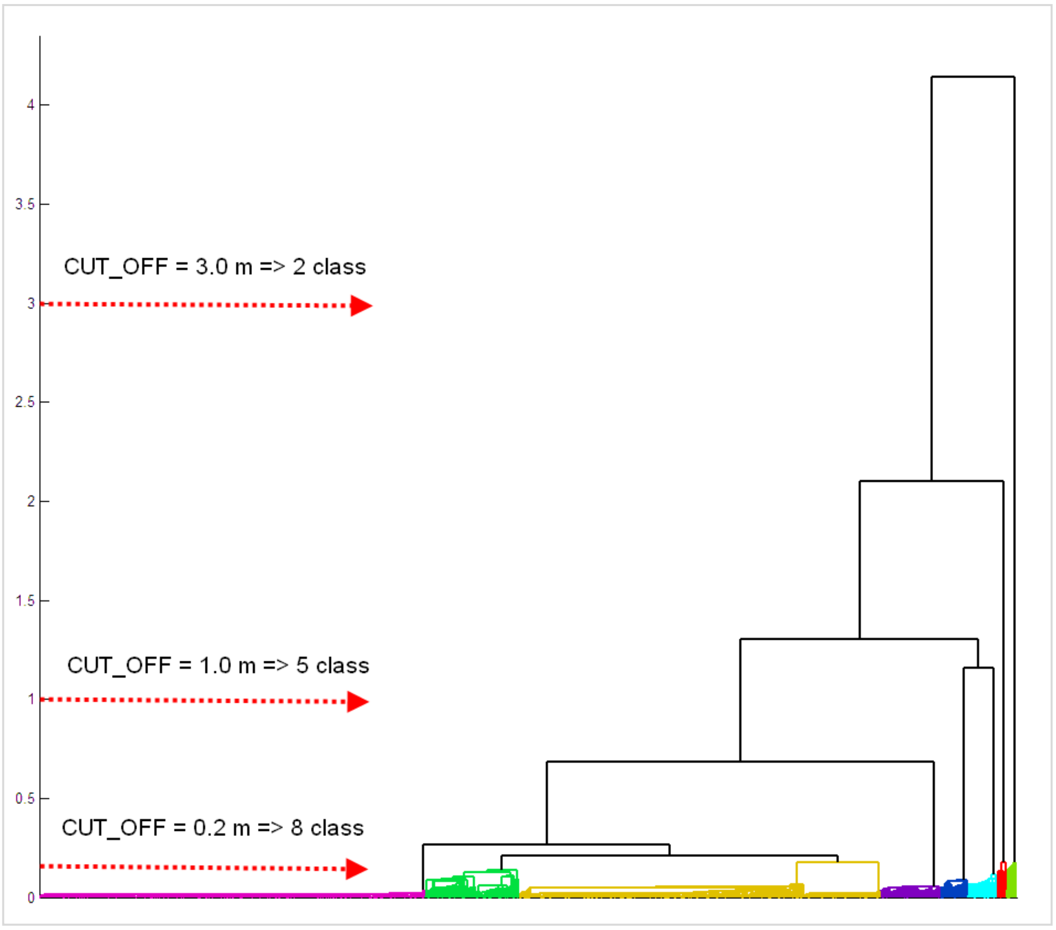


Figure 4-19 Dendogram of the 3D point cloud data *(MATLAB Image).*

Hierarchical clustering is very important part of the plane detection process. If different objects are defined as one this may result in false plane detection and erroneous data association. Selecting the optimal cut-off value is important to prevent this wrong data association. This cut-off value is selected according to visual check and cluster number-cut-off value graphs.

In Figure 4-14 and Figure 4-15 there are 8 point groups. For this scene the cluster number-cut off value relationship was analyzed. The clustering algorithm was run 1000 times for each cut-off value. The results are provided in Figure 4-20 and Figure 4-21.

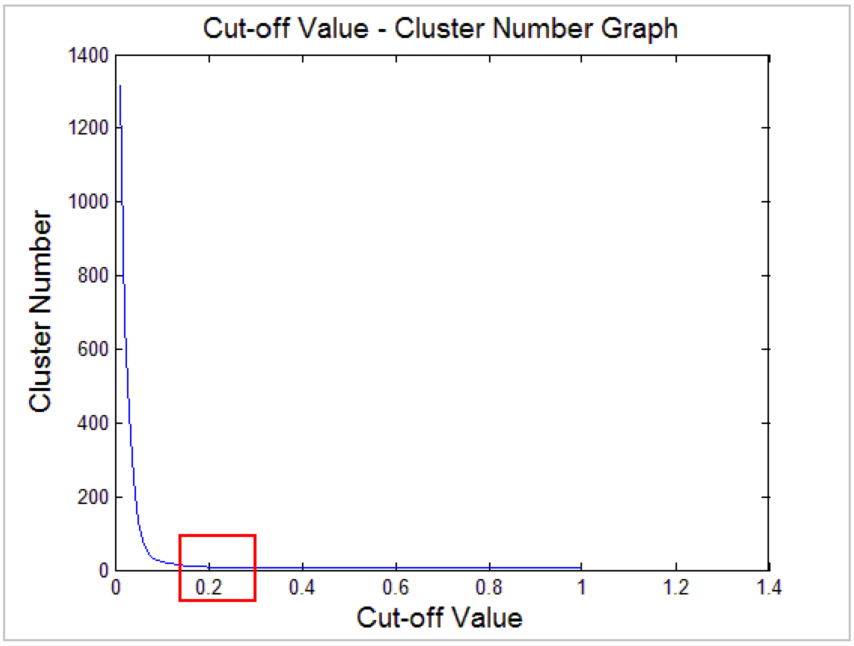


Figure 4-20 Distribution of the cut-off value *(MATLAB Image).*

Figure 4-20 shows the founded cluster number for cut-off values between 0.01 and 1. The expected number of clusters is 8 and this was founded around 0.2. The detailed graph is shown in Figure 4-21.

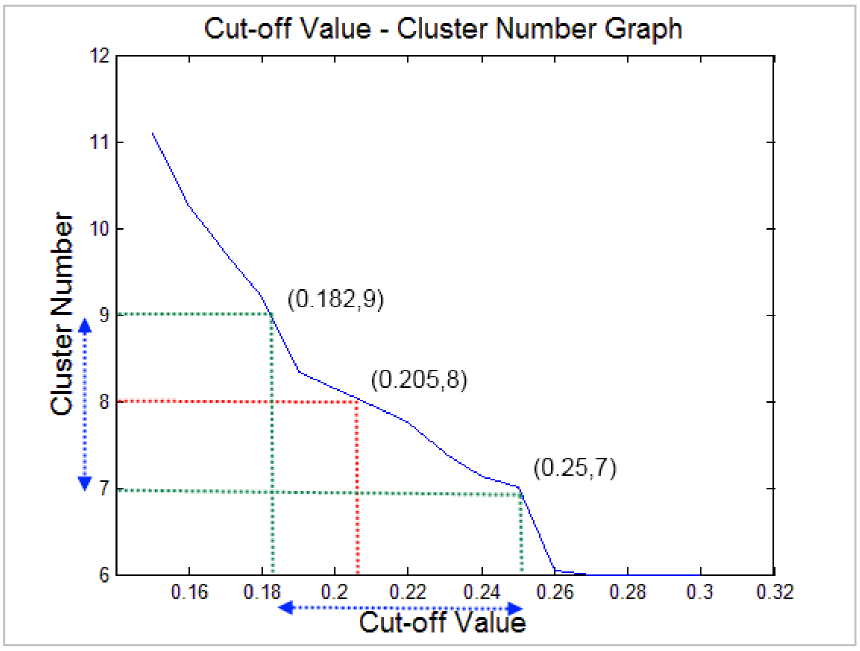


Figure 4-21 Distribution of the cut-off values *(MATLAB Image).*

The optimal value is 0.205 meters and in simulations this value was used.

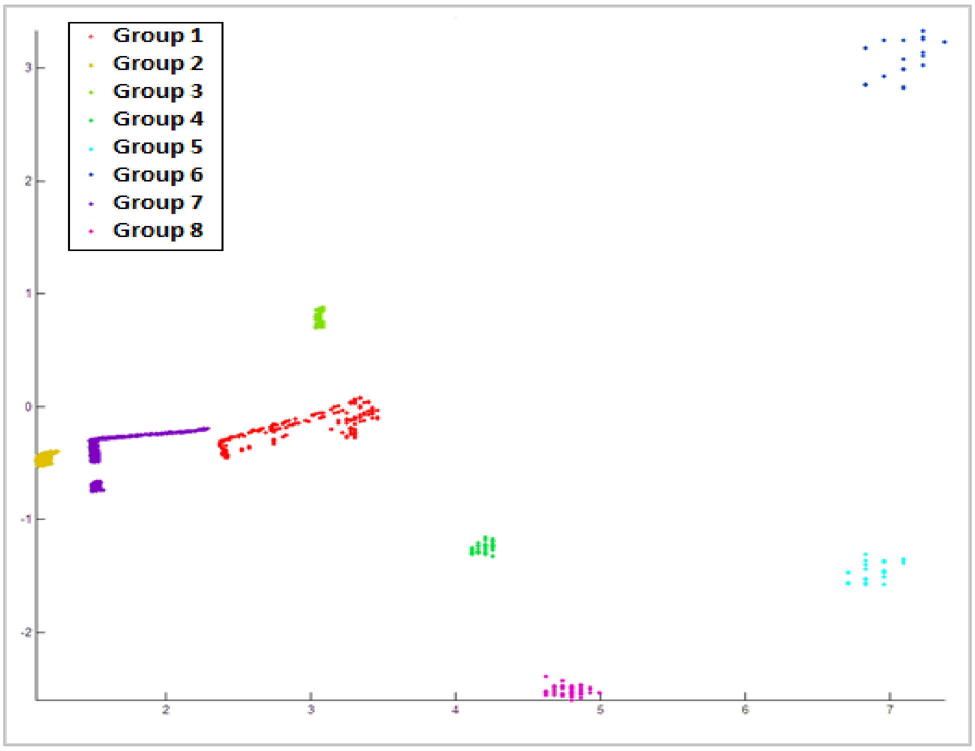


Figure 4-22 Results of the clustered data *(MATLAB image).*

Not all these clusters were used to extract the plane. The aim was to find reliable point cloud groups which exactly define an object *(plane)*.

The following criteria are used to eliminate clusters.

- The number of points should be greater than 50 and less than 1000. *(For 2000 points)*
- The mean point of cluster should be less than 3 m.
- The variance should be small *(less than 0.1)*
- The maximum angle between point groups and the vehicle should be 28 degrees.

The clustering process applied to randomly selected 2000 points because of the time limitations. The algorithm finds the cluster number of remaining points according to the Euclidian distance criteria. Every point is added to the closest clustered points' cluster.

Figure 4-23 shows sample data removal according to the maximum angle criteria. If the angle of any point in the group exceeds the 28 degree algorithm this classifies this group as unusable. The algorithm decides that the point group is a part of the object and the fitting plane of this point group injects erroneous landmarks into the algorithm

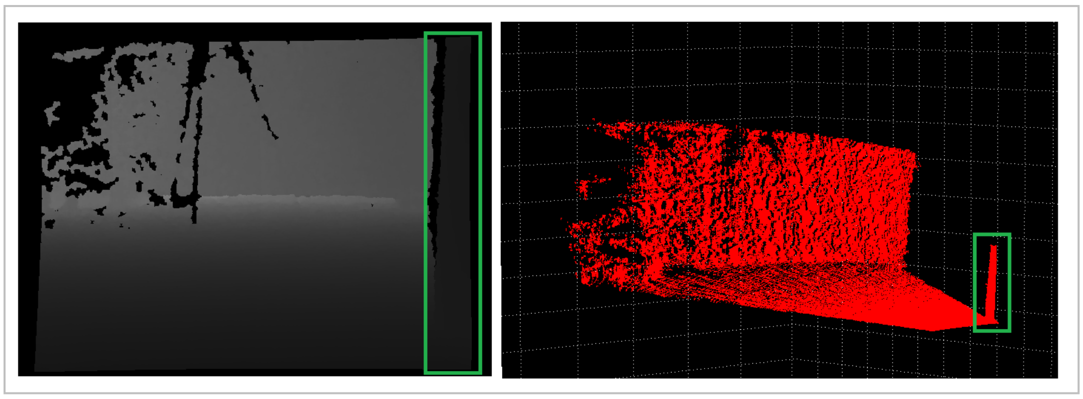


Figure 4-23 Removed edge point groups with angle criteria *(MATLAB Image).*

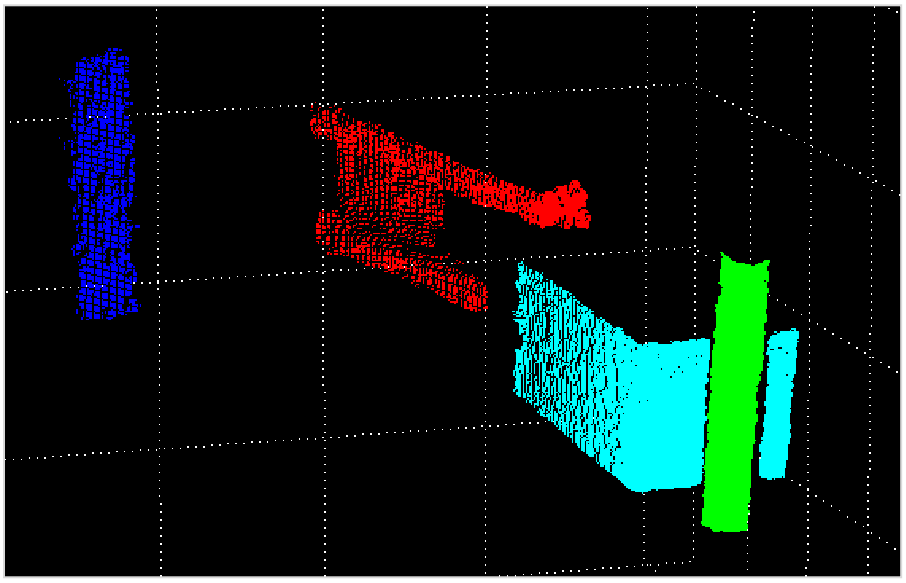


Figure 4-24 Plane candidates *(MATLAB Image).*

In figure 4-24, the point groups are shown after elimination according to the defined criteria *(distance and object size)*. However, there is one more criterion which is that the point groups can be eliminated if standard deviation is very high. Figure 4-25 shows the plane candidates after elimination.

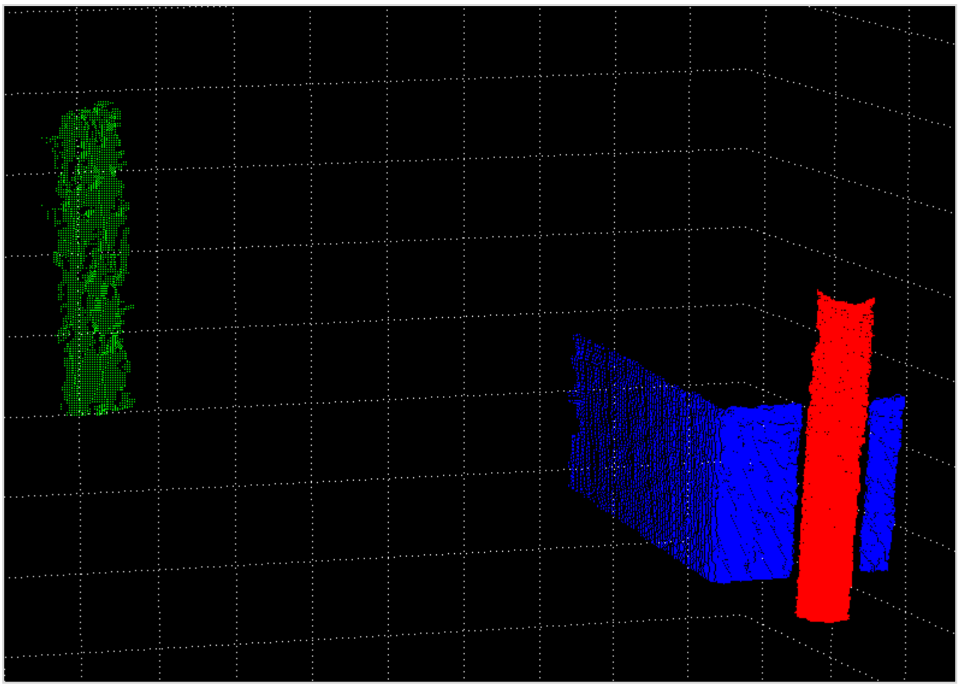


Figure 4-25 Plane candidates after the clusters with high variances were eliminated *(MATLAB Image).*

There are 3 point groups and all these point groups are candidates for the plane extraction algorithm.

4.1.5 Region Growing with Connected Component Analysis

Region growing segmentation according to connectivity is a well-known segmentation method for binary and grayscale images. However, a priori knowledge is required concerning the intensity of the target objects [39]. In the current study, there was no a priori knowledge about the target objects' intensity values since these values change with the motion.

Figure 4-26 presents a histogram of the depth image of scan 281 in which the connected component analysis is not appropriate for the depth image. No meaningful intensity threshold to discriminate the objects can be found because of the ground floor data. The ground floor data may contain values between 0-65025.

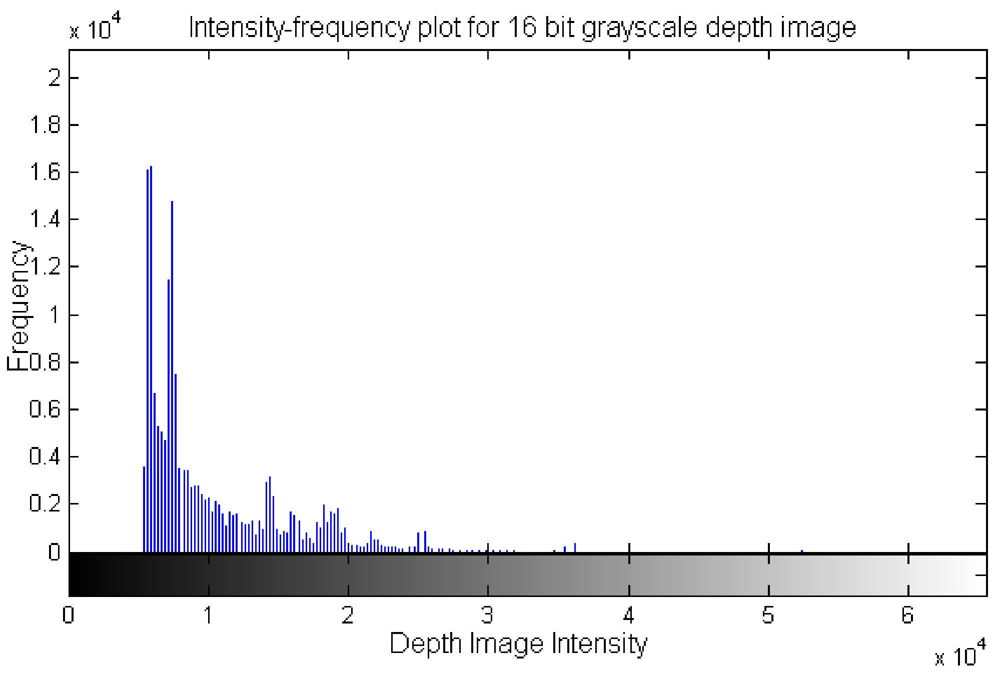


Figure 4-26 Histogram of depth image *(scan 281- Figure 4-13) (MATLAB Image).*

On the other hand there are 3D extensions of the connected component analysis method. A 26-connected component analysis has been used for traffic monitoring from the helicopter for 3D aerial data in [40]. In the current study, the connectivity was defined with a threshold value of 0.205 which is given in 4.1.4 for the hierarchical clustering cut-off. It is easy to discriminate the connected components with region growing according to 26-connectivity and this gives the same result with hierarchical clustering. On the other hand, the connected component analysis takes longer compared with hierarchical clustering as shown in Table 4-1.

Table 4-1 Segmentation Methods Time Comparison

Method	**Time (seconds)**
Hierarchical Clustering	0.7247
Region Growing with Connected Component Analysis	14.3285

Region growing with connected component analysis is more expensive than the hierarchical clustering method and throughout the current research hierarchical clustering method for object segmentation was applied.

4.2 Plane Feature Extraction

4.2.1 Applied Plane Feature Extraction Method

After removing the unnecessary data, the remaining points are candidates of the plane *(foreground objects).* Figure 4-27 includes the environment (a), RGB image (b) and 3D point cloud (c).

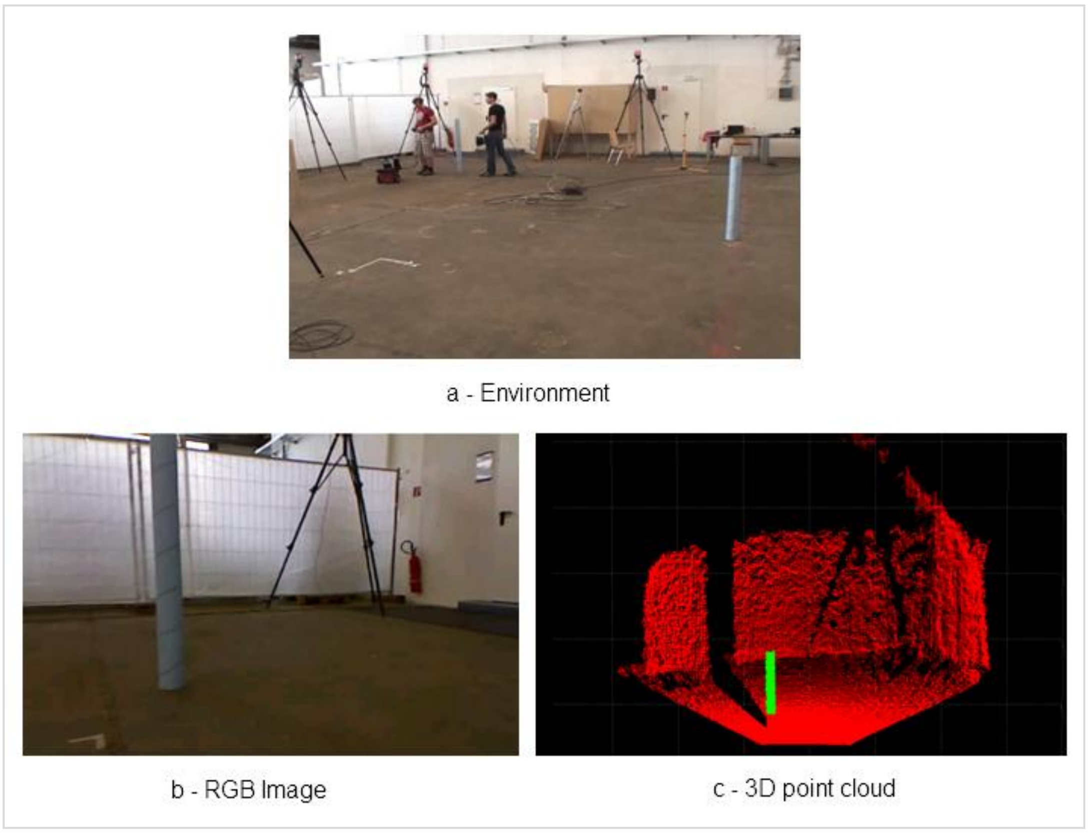


Figure 4-27 Vehicle and 3D scan. (a) SLAM environment from external camera (b) RGB image of scan from Kinect camera (c) 3D point cloud data from Kinect camera *(The red points have been removed, only the green points are used for the landmark (plane) extraction). (MATLAB images)*

The algorithm used in the current study is a process like the RANSAC algorithm, in fact, the proposed application is the modified RANSAC algorithm. **RANdom SAmple Consensus** *(RANSAC)* is a method to determine which points in a point cloud data satisfy a certain mathematical model *(i.e. plane, curvature ...)*. To achieve this aim RANSAC randomly chooses a few points to describe the specified model, for a plane this is 3. Then the point cloud data is separated to 2 classes called inliers and outliers. Inliers are points which lie within a certain accepted distance threshold, *t*, to specified model. The search ends with a maximum iteration count or having found a sufficient number of inliers [18].

In the application presented in this thesis the updated version of Tim Zaman's plane fitting algorithm *(RANSAC and LSE)* [41]. The proposed algorithm does not select only three points to fit a plane. Instead it uses a specified percentage of the data *(thousands of points)*.

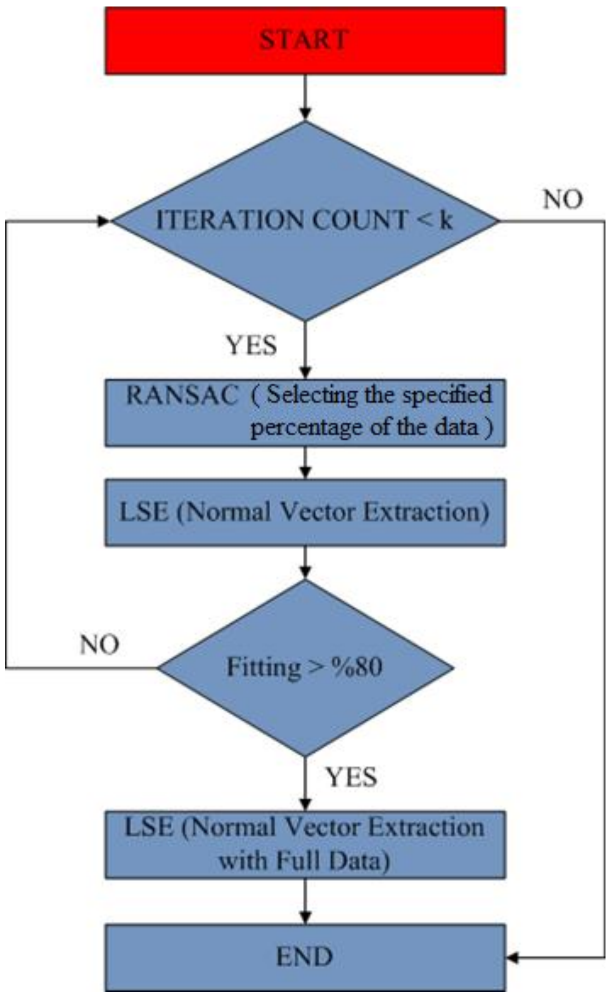

START
ITERATION COUNT < k
NO
YES
RANSAC (Selecting the specified percentage of the data)
LSE (Normal Vector Extraction)
NO
Fitting > %80
YES
LSE (Normal Vector Extraction with Full Data)
END

Figure 4-28 Plane extraction algorithm.

In the LSE function the algorithm first finds the scatter matrix for all selected points.

$$mean = m = \frac{\sum_{i=1}^{n} p_i}{n} \tag{4.1}$$

$$S = \sum_{i=1}^{n} (m - p_i)' * (m - p_i) \tag{4.2}$$

m : mean of all selected points

n : total number of selected points

p_i :ith point from selected point group.

In the proposed algorithm the scatter matrix is 3x3. Then an eigenvector is computed that corresponds to the smallest eigenvalue required to find the normal vector of selected points.

$$Sx = \lambda x \tag{4.3}$$

S : Scatter matrix

x : eigenvector

λ : eigenvalue

For zero eigenvalue $\lambda = 0$:

$$Sx = \lambda x = 0 \tag{4.4}$$

The scatter matrix and eigenvector becomes perpendicular for a zero eigenvalue therefore, the eigenvector corresponding to the minimum eigenvalue is the best definition of normal vector.

Now, the algorithm finds the best fitted normal vector to the selected point group. Then the algorithm controls the fitting performance of points to the estimated normal vector. For all points the following equation is calculated.

$$dist = n_{est}.m - n_{est}.p_i \tag{4.5}$$

$dist$: determined distance to fitted plane.

n_{est}: estimated normal vector

m : mean point of inliers *(mean of selected half of the data)*

p_i : ith point from the selected point group.

The geometrical definition is shown in Figure 4-29.

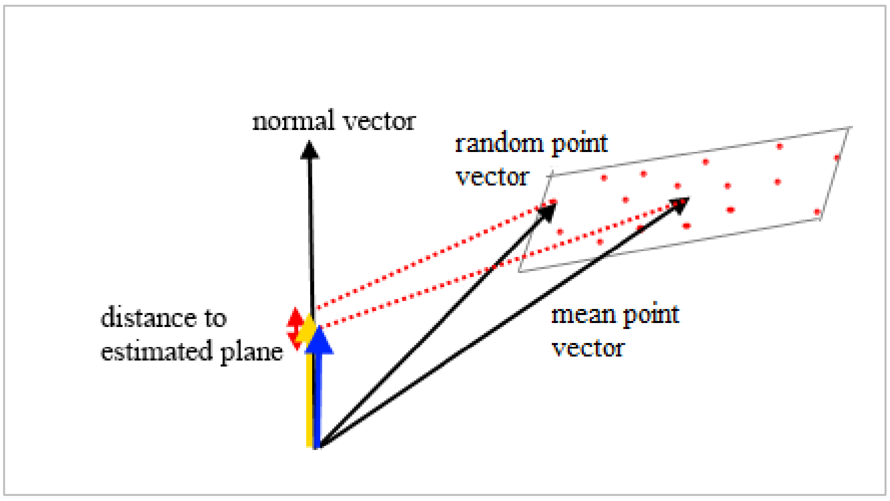


Figure 4-29 Finding the point distance to the estimated plane.

In figure 4-29 the blue vector is the dot product of the mean point vector and normal vector. The yellow vector is the dot product of the ith point vector and normal vector. The difference between these two vectors gives the perpendicular distance of ith point to the estimated plane.

After the distance calculation for all points, the algorithm checks the percentage of the inliers. If the distance is less than 0.1 meters, the relevant point is selected as an inliers. The algorithm accepts the found normal vector if the inlier points are higher than the 80% of the point cloud data.

Then the algorithm finds the final estimation of normal vector according to the full data. Selected points are shown in Figure 4-30(b) in green and the selected inliers which fits the small plane are shown in Figure 4-30(c) and the final plane that fits to the inliers is given in Figure 4-30(d).

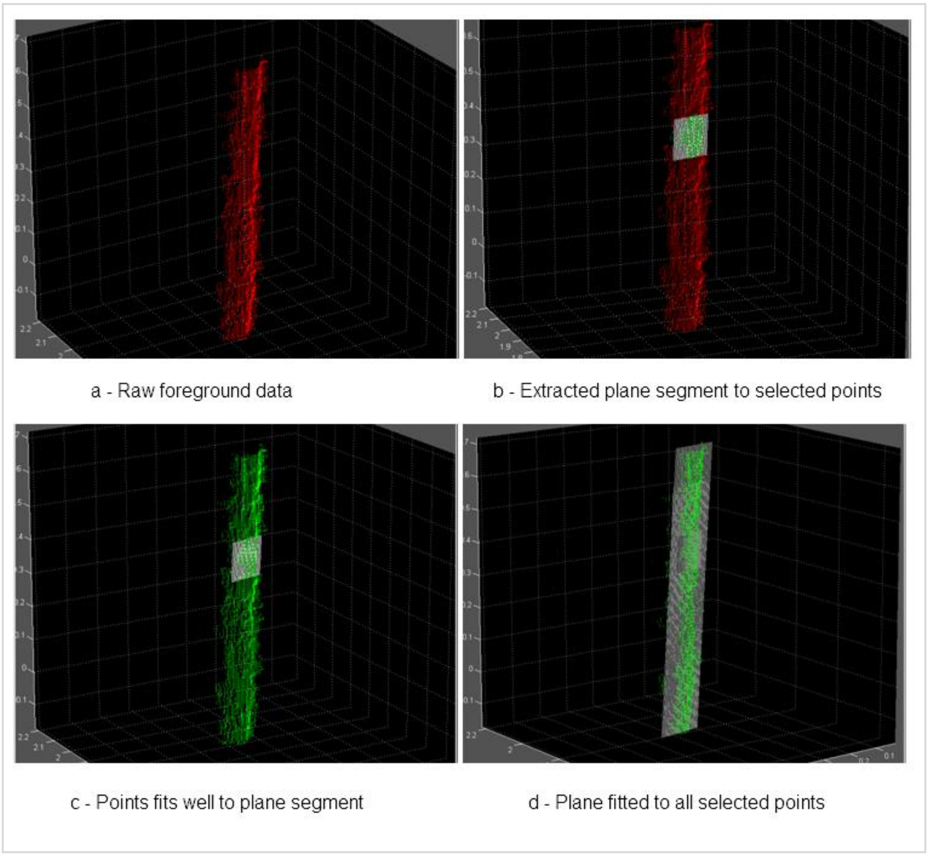


Figure 4-30 Plane fitting *(MATLAB Images).*

The map environment will not always be as simple as presented in Figure 4-27. Generally environments will be more complex as shown in the plane extraction given in Figure 4-31.

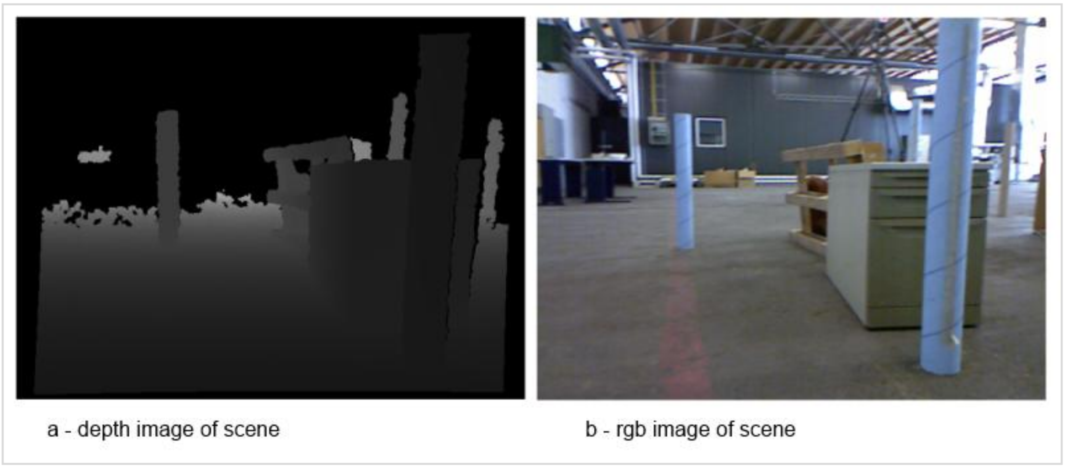


Figure 4-31 RGB and depth image of scan 281 of fre2_slam3 map [34].

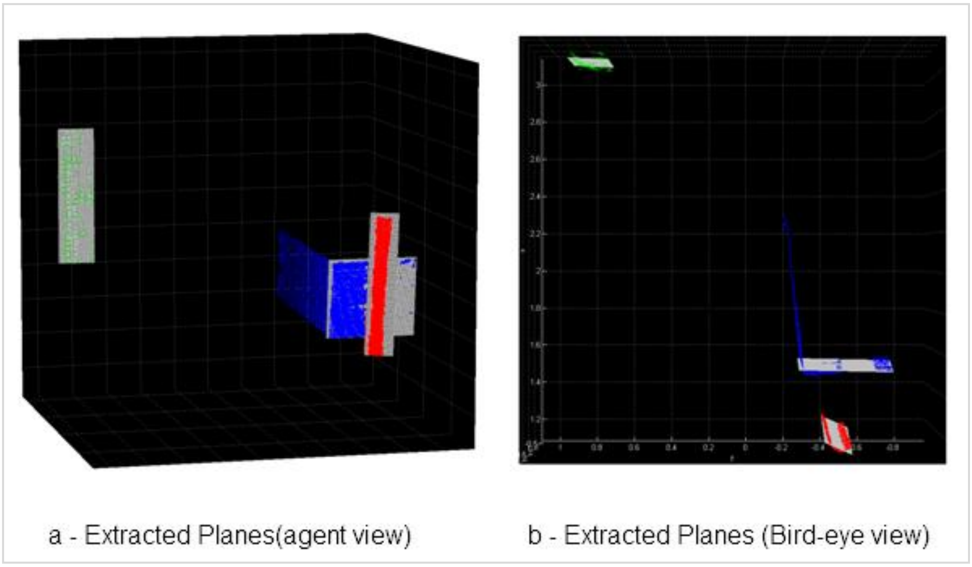

a - Extracted Planes(agent view)
b - Extracted Planes (Bird-eye view)

Figure 4-32 Extracted planes of scan 281 of fre2_slam3 map (*MATLAB Images).*

Even though the environment is complex, the proposed algorithm finds sufficient plane features.

4.2.2 Plane Feature Descriptors

An example of a detected plane feature is given in Figure 4-33. For planar surface resolution with 0.05 meters, the algorithm maintains a 19 x 4 array for each of the x, y and z values, each providing a76 point location and a normal vector.

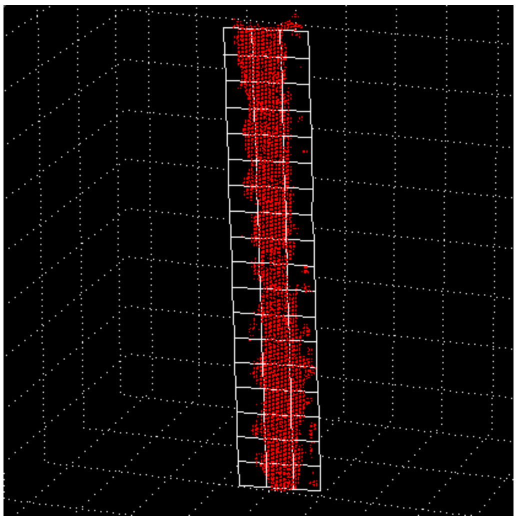


Figure 4-33 Extracted plane feature (*MATLAB Image)*

The size of the arrays change according to the surface size and resolution of the plane. The robot in this study does not directly use this plane definition but it extracts the compact properties from this plane definition. The properties are the center point, normal vector and area of the plane.

Throughout the research the plane features have been described as having 3 different feature descriptors. These feature descriptors are called landmark vectors in SLAM applications. Therefore, the **plane as point** landmark vector can be defined with the center point of plane.

$$Plane\ as\ Point = [x \quad y \quad z]^T \tag{4.6}$$

We can define the plane with a point feature and a normal vector. This landmark vector is called a **plane as oriented point**.

$$Plane\ as\ Oriented\ Point = [x \quad y \quad z \quad n_x \quad n_y \quad n_z]^T \tag{4.7}$$

The plane can be defined with a point, normal vector and area information. This landmark vector named; **plane as surface**.

$$Plane\ as\ Surface = [x \quad y \quad z \quad n_x \quad n_y \quad n_z \quad area]^T \tag{4.8}$$

4.2.3 Alternative Plane Feature Extraction Methods

There are different plane detection techniques in the literature. The Hough-transform and Random Sample Consensus *(RANSAC)* paradigm are the two main and common methods. The performance comparison for plane detection has been applied to the construction of roof planes using a LIDAR scanner. According to the comparison, the RANSAC algorithm provides higher quality in a shorter time [42].

There are also different plane extraction *(PE)* methods based on the random sample consensus *(RANSAC)*. The standard RANSAC plane extraction method attempts to maximize the number of inliers. The disadvantage of the standard RANSAC plane extraction method is that it may fail when a scene contains multiple intersecting planar surfaces with limited sizes [43].

The CC-RANSAC *(Coherence Check RANSAC)* plane extraction method successfully solves the straddling-plane problem when the scene contains simple steps. Figure 4-34 shows a curb and ramp. These types of features must be identified for safe parking [44].

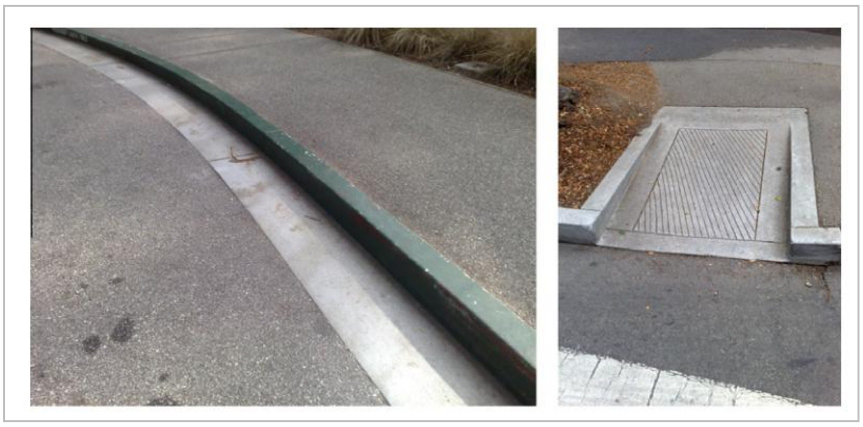


Figure 4-34 An example of a curb (a) and ramp (b) [44].

CC-RANSAC methods solve the plane extraction problem for these kinds of simple scenes. However, if the scene contains stairway with more than 6 steps the CC-RANSAC algorithms may fail. To solve this problem the Normal Coherence Check RANSAC (NCC-RANSAC) method is proposed [43]. This method checks the normal coherences for all the data points of the inlier patches *(on the fitted plane)* and removes the data points whose normal directions are contradictory to that of the fitted plane. This process avoids erroneous plane extractions [43].

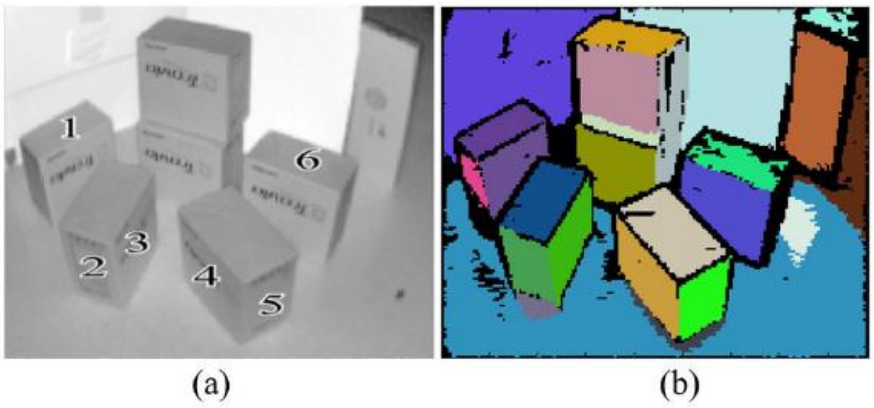


Figure 4-35 NCC-RANSAC with SR-4000 data. (a) Intensity Image (b) Extracted planes [43].

Objects are well-separated from each other in our navigation environment; therefore, there is no need to use the CC-RANSAC or NCC-RANSAC methods. Instead, the modified version of classical RANSAC method is implemented which has the computational advantage over the CC-RANSAC and NCC-RANSAC methods.

In implementation in this study the raw data was clustered using the hierarchical clustering method according to connectivity. This clustering step is the precondition for plane extraction functions. With the help of this clustering process, the modified version of classical RANSAC plane extraction function is satisfactory. In classical method the algorithm selects 3 random points and fits a plane to these selected points. The algorithm searches until the fitting performance exceeds the specified threshold. In our modified RANSAC method 1 random point and the specified percentage of the data were selected, according to the spatial distance to the selected random point. The details are presented in section 4.2.1.

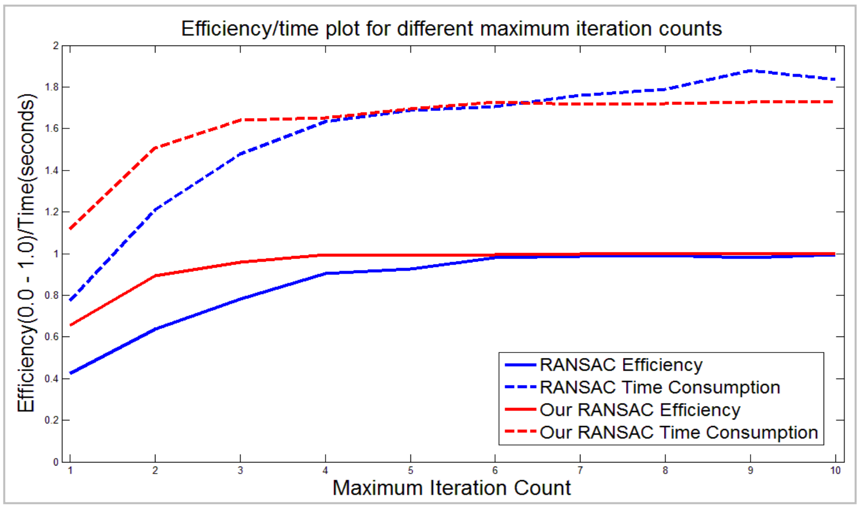


Figure 4-36 Comparison of the RANSAC algorithms *(MATLAB Image)*

Figure 4-36 shows the comparison of the classical RANSAC method and the implementation in the current study. The time and the performance comparison changes according to the maximum iteration count.

Maximum iteration count < 6: The proposed method satisfies good plane extraction performance with the higher time consumption.

Maximum iteration count ≥ 6: Both methods satisfy 100% plane extraction performances. However, the time consumption of the proposed method is smaller than the classical RANSAC method.

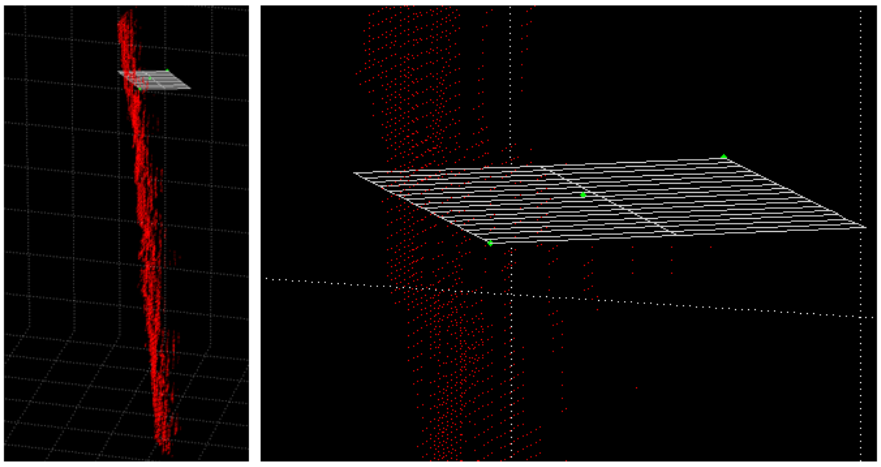


Figure 4-37 Initial plane extraction with RANSAC *(MATLAB Image).*

Classical RANSAC plane extraction may fail if the initially selected random points lie on wrong axis and this classical method may require high number of iterations to achieve the successful result. Figure 4-37 shows this kind of initial false estimation and the extracted plane. This kind of initial random point selection result in a large amount of time spent with a lower performance according to the proposed implementation. In the current study, the robust feature detection is important and to achieve this the modified version of RANSAC algorithm was used to detect the robust plane features.

4.3 SURF Feature Extraction

SURF is one of the most used rotation and scale invariant interest point detectors and a descriptor which helps to find matches between two images. The ***R(x, y, σ)*** function is the interest point criteria function and computes the response of every pixel in the image for different scales [45].

$$R(x,y,\sigma) = f\big(I(x,y,\sigma)\big) \tag{4.9}$$

The interest point detection algorithm considers the input image as an image stack *(image pyramid)* which is a collection of the input image in different scales. The algorithm can generate this image pyramid by smoothing and down-sampling the image. However, the SURF interest point detection algorithm increases the filter size and maintains a stable image size instead of down-sampling the image size. The integral image speeds up the interest point detection and convolution process. The details of the integral image are introduced in Appendix B. This process provides computational efficiency and avoids aliasing [45].

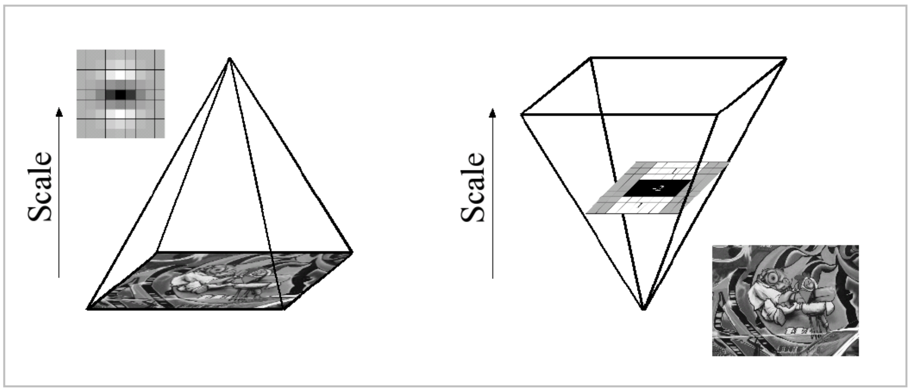


Figure 4-38 Instead of iteratively reducing the image size *(left)*, the use of integral images allows the up-scaling of the filter at constant cost (*right*) [45].

The response function $R(x, y, \sigma)$ is the determinant of the scale normalized Hessian Matrix [45].

$$R(x, y, \sigma) = \det\big(H(x, y, \sigma)\big) \tag{4.10}$$

The scale normalized Hessian Matrix is:

$$H_k(x, y, \sigma) = \frac{1}{\sigma^2}\begin{bmatrix} \frac{\partial^2}{\partial x^2} S_k(x, y, \sigma) & \frac{\partial^2}{\partial x \partial y} S_k(x, y, \sigma) \\ \frac{\partial^2}{\partial x \partial y} S_k(x, y, \sigma) & \frac{\partial^2}{\partial y^2} S_k(x, y, \sigma) \end{bmatrix} \tag{4.11}$$

In the formula (4.11), H_k represents the Hessian of the k^{th} frame and S_k represents the intensity for the image stack $\boldsymbol{I(x, y, \sigma)}$.

The SURF feature detection algorithm uses box filters and approximates the determinant of the Hessian Matrix. The approximation formula is given below.

$$\det\left(H_{approx}\right) = D_{xx}D_{yy} - (0.9D_{xy})^2 \qquad (4.12)$$

Ideally, as the definition of determinant obligates, the factor in front of D_{xy} should be 1.0 rather than 0.9. However, according to the error introduced by the discretization approximation, 0.9 is used as an ad-hoc compensation [45].

In SURF, these derivatives in the Hessian determinant are approximated with 3 box filters. A 9 x 9 version of the box filters are given in Figure 4-39 and Figure 4-40 [45].

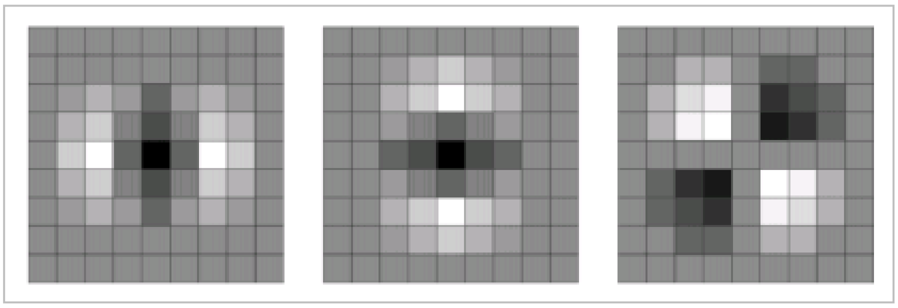


Figure 4-39 The 9 x 9 Gaussian second order partial derivatives in the x, y and xy directions [46].

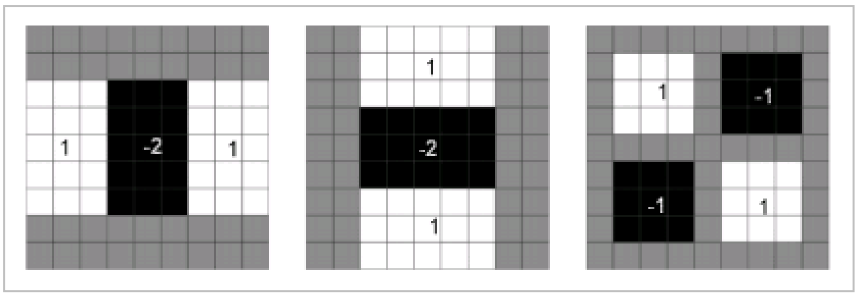


Figure 4-40 The weighted 9 x 9 box filter approximation of the 9 x 9 Gaussian filters. *(+1,-1, -2 are the weights assigned to those regions and the grey regions are of value 0)* [46].

The scale-octave representation is shown in Figure 4-41.

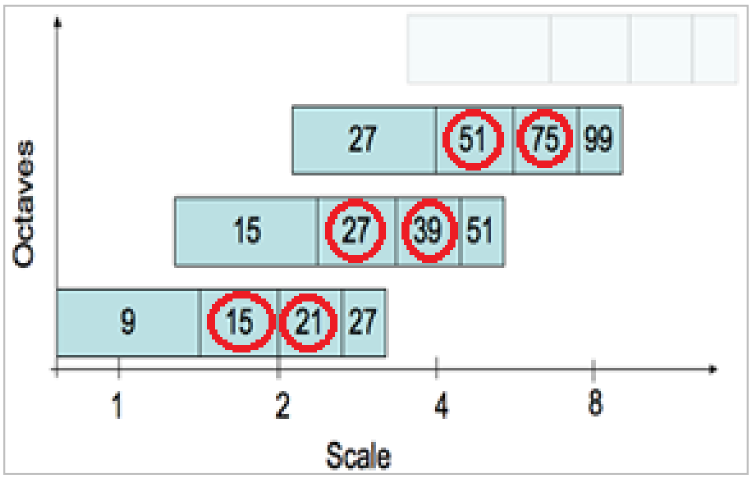


Figure 4-41 Scale-octave representation [46].

The algorithm searches the larger shapes at higher octaves and generates 4 different response layers for each octave. The first and last layers of each octave are only used for comparison. For example, at first octave the algorithm searches the interest points for the second and third response layers. The layers are shown in the circles in Figure 4-41. Every response layer corresponds to an approximation of a σ value. This σ value defines the variance of the approximated Gaussian filters. For instance, the first octave – 9 x 9 filter pair corresponds to **σ = 1.2.**

Scale information gives detail about the size of the detected SURF feature. The filter size, scale and the SURF feature size are directly proportional. If the scale of the SURF feature is around 4, this means that the SURF feature gives the highest response to the filter size 39 x 39. This scale property maintains the size of the SURF feature and prevents incorrect correspondences.

4.3.1 Interest Point Detection

Figure 4-42 shows the convolution of the 9 x 9 box filter for any center pixel (x, y). The result of the convolution is A-3B.

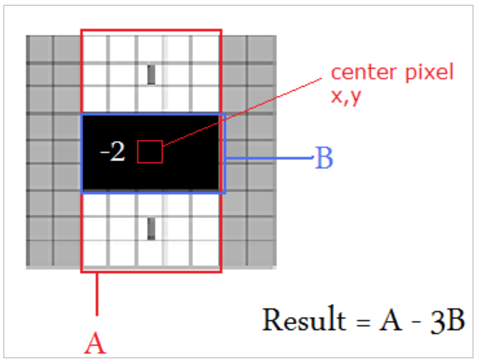


Figure 4-42 Sample convolution mask [46].

A and B define the sum of the intensities of the surrounding pixel. The A and B values can easily be found using integral image calculation. The details of the integral image are given in Appendix B. The convolution responses for each filter size and calculation of the response are found using equation 4.12. If determined value reaches the defined threshold the algorithm keeps it as a SURF feature candidate. For each SURF feature candidate the x, y, scale and laplacian values are retained. The x and y defines the location of feature in image, the scale defines the size of the feature and the laplacian define the blob type.

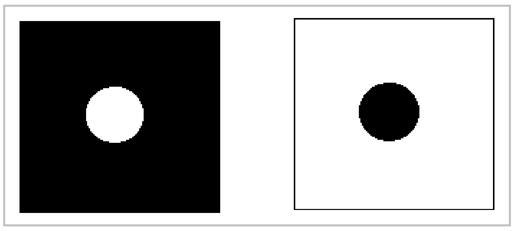


Figure 4-43 White and black shape.

For the SURF features, the determinant of the white and black shape is the same. The algorithm uses the sign of the laplacian *(trace of the Hessian matrix)* in order to avoid matching a white circle with a black one. The black shapes have positive values and the white shapes have negative values.

4.3.2 Non-maxima Suppression

The algorithm runs a non-maxima suppression process after finding the SURF interest point candidates. Excluding the weak interest points is carried out at two levels. In the first level, the algorithm applies threshold test to all the interest points within a layer. In this test the interest points above the threshold value are accepted. The higher the threshold value that is chosen, fewer but stronger interest points are chosen. The second level of filtering is called non-maximum suppression this is carried out across three layers with the center pixel in the center layer being compared with a total of 26 neighbors.

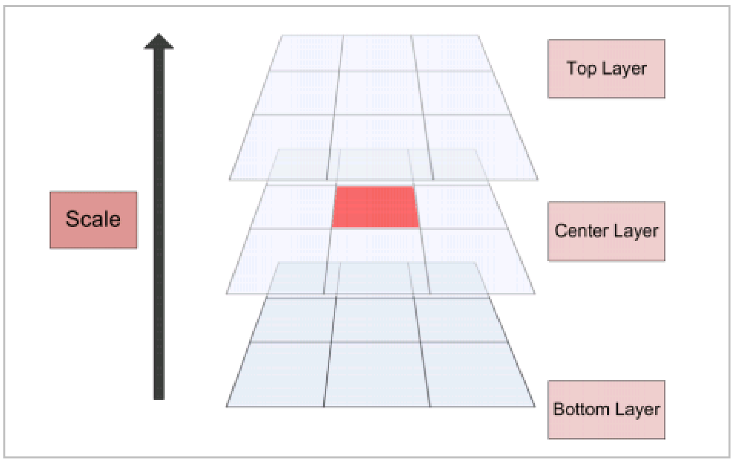


Figure 4-44 Representation of the non-maximum suppression [46].

The algorithm evaluates the center pixel as a SURF feature if it has the highest response value among the candidates. The algorithm eliminates the center pixel from the interest point candidates if the response value is lower than any of its neighbors.

4.3.3 SURF Feature Descriptor

The SURF feature detector finds the features on 2D depth image. Then in our applications, the algorithm finds the 3D location of these SURF features. We use camera calibration and focal length values for this computation. The final SURF feature vector contains the 3D location of found feature, scale and laplacian properties of the detected points.

The SURF feature descriptor was generated with 5 parameters and its vector representation is shown below.

$$Surf\ Point = [x \quad y \quad z \quad scale \quad laplacian]^{T} \tag{4.13}$$

The result of the interest point detection process on a sample image can be observed in Figure 4-45.

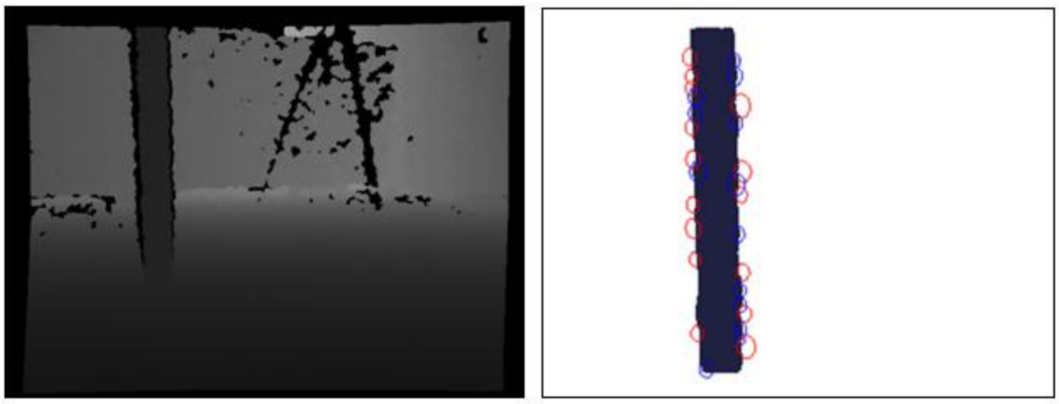


Figure 4-45 Depth Image and Extracted SURF Features *(MATLAB image).*

The SURF feature detection algorithm gives a high number of responses at the boundaries of the depth image. Thus, through the simulation, the algorithm has to handle high number of landmarks and simulation results which required a large amount of time. We expect that the high number of SURF features help the algorithm to decrease the motion error and increase the performance of SLAM.

Also, SURF feature points do not define objects, instead they give information about a small area of the image. The following Chapters 5 and 6 show the comparison of the final error rates. Comparing SURF and planar features gives information about the low number of compact features that define objects and the high number of localized features that gives information about that small portion of image.

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 Data Structure

In fastSLAM all the predictions concerning the pose of the robot are sampled with particles taken from the Gaussian distribution of the control data. Throughout the navigation the robot keeps in memory 100 different estimates *(particles)*. Each particle contains the state vector of the robot, particle weight and Gaussian estimation for each detected landmark.

The state vector of the robot is maintained in the form:

$$state\ vector = \begin{bmatrix} x \\ y \\ yaw \end{bmatrix} \tag{5.1}$$

In equation 5.1, x and y represent the estimation of the robot pose for one particle while the yaw represents the rotational information estimation around the z direction of the robot relative to the global coordinate system. In the implementation in this study z, roll and pitch were ignored because of the planar navigation environment. In fastSLAM, the particle weight in kept as a scalar value which defines reliability of the particle.

Also, the surrounding objects are defined with a Gaussian distribution. The robot keeps the landmarks with the mean vector and covariance matrix. Equations 5.2 and 5.3 show the mean vectors for each different landmark vector:

$$\mu_{\ landmark\ plane\ as\ point} = [x \quad y \quad z\ ]^T \tag{5.2}$$

$$\mu_{\ landmark\ plane\ as\ oriented\ point} = [x \quad y \quad z \ \ n_x \quad n_y \quad n_z]^T \tag{5.3}$$

$$\mu_{\, landmark\ plane\ as\ surface} = [x \quad y \quad z \quad n_x \quad n_y \quad n_z \quad area\,]^T \tag{5.4}$$

$$\mu_{\, landmark\ SURF\ point} = [x \quad y \quad z \quad scale \quad laplacian\,]^T \tag{5.5}$$

The covariance matrixes of the detected landmarks are kept in the form:

$$\Sigma_{\,plane\ as\ point} = \begin{bmatrix} \sigma_{x,x} & \sigma_{x,y} & \sigma_{x,z} \\ \sigma_{y,x} & \sigma_{y,y} & \sigma_{y,z} \\ \sigma_{z,x} & \sigma_{z,y} & \sigma_{z,z} \end{bmatrix} \tag{5.6}$$

$$\Sigma_{plane\ as\ oriented\ point} = \begin{bmatrix} \sigma_{x,x} & \sigma_{x,y} & \sigma_{x,z} & \sigma_{x,n_x} & \sigma_{x,n_y} & \sigma_{x,n_z} \\ \sigma_{y,x} & \sigma_{y,y} & \sigma_{y,z} & \sigma_{y,n_x} & \sigma_{y,n_x} & \sigma_{y,n_x} \\ \sigma_{z,x} & \sigma_{z,y} & \sigma_{z,z} & \sigma_{z,n_x} & \sigma_{z,n_x} & \sigma_{z,n_x} \\ \sigma_{n_x,x} & \sigma_{n_x,y} & \sigma_{n_x,z} & \sigma_{n_x,n_x} & \sigma_{n_x,n_y} & \sigma_{n_x,n_z} \\ \sigma_{n_y,x} & \sigma_{n_y,y} & \sigma_{n_y,z} & \sigma_{n_y,n_x} & \sigma_{n_y,n_y} & \sigma_{n_y,n_z} \\ \sigma_{n_z,x} & \sigma_{n_z,y} & \sigma_{n_z,z} & \sigma_{n_z,n_x} & \sigma_{n_z,n_y} & \sigma_{n_z,n_z} \end{bmatrix} \tag{5.7}$$

$$\Sigma_{plane\ as\ surface} = \begin{bmatrix} \sigma_{x,x} & \sigma_{x,y} & \sigma_{x,z} & \sigma_{x,n_x} & \sigma_{x,n_y} & \sigma_{x,n_z} & \sigma_{x,a} \\ \sigma_{y,x} & \sigma_{y,y} & \sigma_{y,z} & \sigma_{y,n_x} & \sigma_{y,n_x} & \sigma_{y,n_x} & \sigma_{y,a} \\ \sigma_{z,x} & \sigma_{z,y} & \sigma_{z,z} & \sigma_{z,n_x} & \sigma_{z,n_x} & \sigma_{z,n_x} & \sigma_{z,a} \\ \sigma_{n_x,x} & \sigma_{n_x,y} & \sigma_{n_x,z} & \sigma_{n_x,n_x} & \sigma_{n_x,n_y} & \sigma_{n_x,n_z} & \sigma_{n_x,a} \\ \sigma_{n_y,x} & \sigma_{n_y,y} & \sigma_{n_y,z} & \sigma_{n_y,n_x} & \sigma_{n_y,n_y} & \sigma_{n_y,n_z} & \sigma_{n_y,a} \\ \sigma_{n_z,x} & \sigma_{n_z,y} & \sigma_{n_z,z} & \sigma_{n_z,n_x} & \sigma_{n_z,n_y} & \sigma_{n_z,n_z} & \sigma_{n_z,a} \\ \sigma_{a,x} & \sigma_{a,y} & \sigma_{a,z} & \sigma_{a,n_x} & \sigma_{a,n_y} & \sigma_{a,n_z} & \sigma_{a,a} \end{bmatrix} \tag{5.8}$$

$$\Sigma_{SURF\ point} = \begin{bmatrix} \sigma_{x,x} & \sigma_{x,y} & \sigma_{x,z} & \sigma_{x,s} & \sigma_{x,l} \\ \sigma_{y,x} & \sigma_{y,y} & \sigma_{y,z} & \sigma_{y,s} & \sigma_{y,l} \\ \sigma_{z,x} & \sigma_{z,y} & \sigma_{z,z} & \sigma_{z,s} & \sigma_{z,l} \\ \sigma_{s,x} & \sigma_{s,y} & \sigma_{s,z} & \sigma_{s,s} & \sigma_{s,l} \\ \sigma_{l,x} & \sigma_{l,y} & \sigma_{l,z} & \sigma_{l,s} & \sigma_{l,l} \end{bmatrix} \tag{5.9}$$

In the matrices given above a defines the area of the plane, l and s defines the laplacian and the scale of the SURF feature.

The robot keeps the mean and covariance matrices for each detected landmark for every particle. In each step, the robot keeps **M.N** *(M:number of particles, N:Number of detected landmarks)* estimates for the surrounding objects.

5.2 Details of the Proposed Implementation

Figure 5-1 shows the steps of the fastSLAM algorithm used in this study.

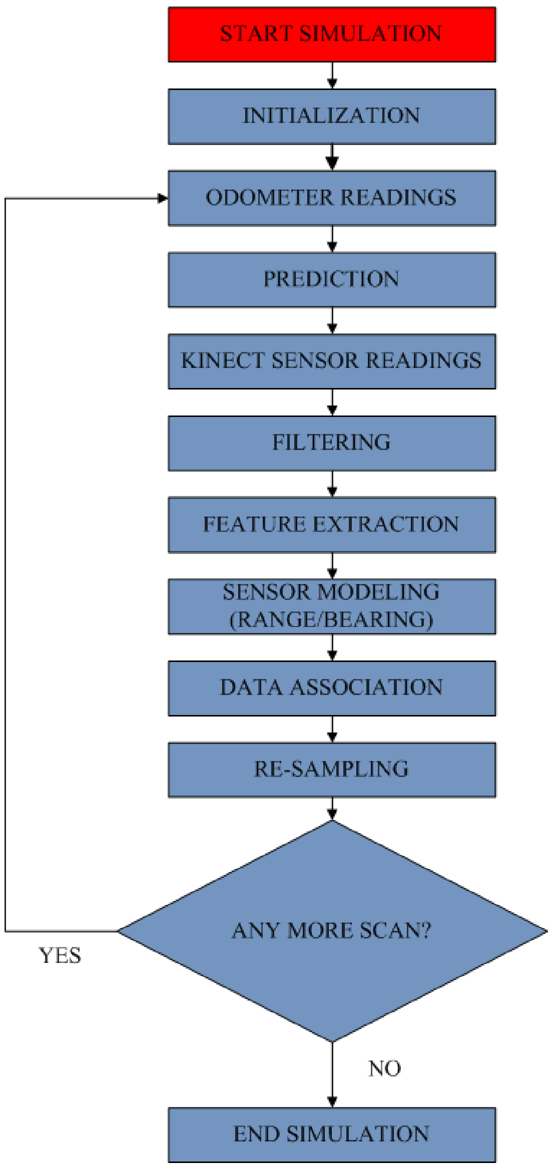


Figure 5-1 General fastSLAM algorithm.

First, the fastSLAM algorithm initializes the simulation parameters. Then the algorithm runs remaining steps iteratively until the end of the navigation.

All the main steps in the proposed algorithm are:

- Initialization
- Control Measurements
- Prediction
- Sensor Measurements
- Filtering Process
- Feature Extraction
- Modeling the Sensor Measurements
- Data Association
- Re-sampling

The code details concerning these steps are given in Appendix D. The simulator used in this study is the extended version of the code devised by Tim Bailey [47].

5.2.1 Initialization

In this step, the proposed algorithm undertakes the first initializations. The algorithm initializes the fastSLAM simulation parameters, vehicle model, motion model, sensor model, measurement model, animation setup and particles.

In this step, we define state vector and feature vector of robot in vehicle model initialization part as follows;

$$state\ vector = \begin{bmatrix} x \\ y \\ yaw \end{bmatrix} \tag{5.10}$$

The state vector only keeps the x, y and yaw values of the robot throughout the navigation. The navigation environment is planar and the z, roll and pitch values are not kept. These are meaningless for planar environments in the simulations in this study.

The vehicle is modeled as a triangle for easy visualization of simulations. The vehicle width and wheelbase are defined as 0.4 and 0.6 meters. These are the default values and user cannot change the vehicle model in the GUI. The start condition of the robot is defined as [0, 0, 0]. The ground truth data for the vehicle is shifted to [0, 0, 0].

The remaining initializations and code details are given in Appendix D.

5.2.2 Control Measurements

The motion model in this study is defined with an odometer motion model. The robot obtains the odometer sensor readings and updates the state vector. In fastSLAM, the robot updates all particles' state vectors. In the simulations, ground truth data is used to generate the odometer sensor readings. The odometer readings contain the $\Delta x, \Delta y, \Delta yaw$ values. The main simulator finds the true odometer values and gives them to the robot with the additional noise. The code details are given in Appendix D.

5.2.3 Prediction

The robot runs the prediction function after obtaining the odometer readings. The robot runs the prediction function for each landmark *(100 times)*, takes samples from the Gaussian distribution of odometer readings and adds these generated values to the particles' state vector.

$$Robot\ state\ vector^{k} = \begin{bmatrix} x + \Delta x \\ y + \Delta y \\ yaw\ + \Delta yaw \end{bmatrix} \tag{5.11}$$

k defines the particle number and the robot runs this prediction for each particle.

5.2.4 Sensor Measurements

In the implementation in this study, the robot moves, makes a prediction and then takes sensor measurements. In this study, taken from the Microsoft Kinect Sensor the sensor measurements contain the depth and RGB image of the relevant scan. These measurements are only a 3D scan of the navigation environment. Raw sensor readings do not give direct information about the landmarks. The robot runs feature the extraction process and models the scan data with a range bearing sensor model.

5.2.5 Filtering

The robot applies filtering to the raw 3D scan data before the feature extraction process. In the first step of filtering process, the robot generates the point cloud data of the obtained depth image with the help of the camera calibration data. Then the robot applies the filtering process to the generated point cloud data.

The robot filters the ground floor and background of the scan. Then robot runs the hierarchical clustering which is a type of unsupervised learning method. According to the result of the clustering process, the robot filters unnecessary and noisy point groups. The filtering process details are given in section 4.1 and Appendix D.

5.2.6 Feature Extraction

After the filtering process, the robot runs feature extraction algorithms. In the current investigation, SURF point features and plane features are used. The details of the feature extraction are given in section 4.2 and 4.3.

5.2.7 Modeling the Sensor Measurement

The algorithm converts the obtained feature vectors using the range-bearing sensor model this model contains 1 range information and 2 bearing information for 3D point landmark. All the sensor measurement models for each landmark vectors are given in the equations 5.12 to 5.15 below.

$$z_{plane\ as\ point} = [range \quad \alpha \quad \beta]^{T} \tag{5.12}$$

$$z_{plane\ as\ oriented\ point} = [range \quad \alpha \quad \beta \quad n_x \quad n_y \quad n_z]^{T} \tag{5.13}$$

$$z_{plane\ as\ surface} = [range \quad \alpha \quad \beta \quad n_x \quad n_y \quad n_z \quad area]^{T} \tag{5.14}$$

$$z_{plane\ as\ point} = [range \quad \alpha \quad \beta \quad scale \quad laplacian]^T \tag{5.15}$$

The sensor model differs according to the selected landmark vector. The sensor measurement for the **plane as point** landmark contains only the range and bearing information. All the other sensor measurement models are shown above. The code details are given in Appendix D.

5.2.8 Data Association

The robot uses the maximum likelihood estimation method for data association [6]. Figure 5-2 shows the one step of the SLAM process. The definitions are as follows:

z_{p1}: Defines the expected sensor readings from lm_1 *(landmark-1)*

z_{p2}: Defines the expected sensor readings from lm_2 *(landmark-2)*

z : Defines the sensor reading.

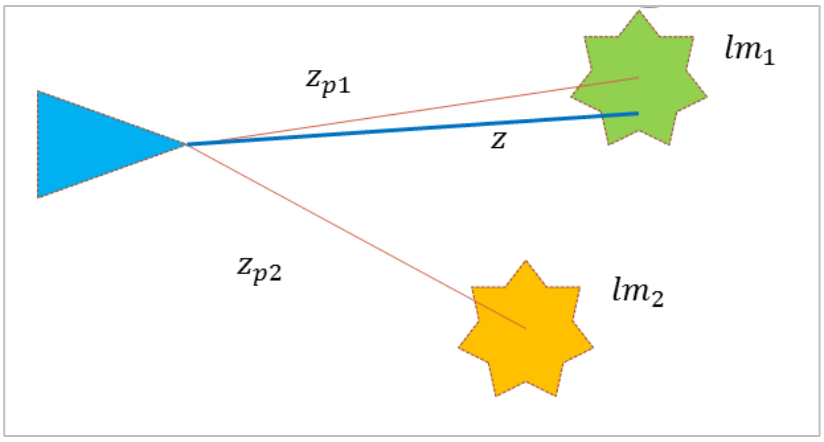


Figure 5-2 Data Association.

According to Figure 5-2, the obtained sensor measurement belongs to the first landmark. However, the robot does not have any prior knowledge about this and calculates the probability values according to the formula below for each earlier detected landmark.

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}}.e^{\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}} \tag{5.16}$$

The computations are;

$$p_1(z) = \det(2\pi\Sigma_1)^{-\frac{1}{2}}.\ e^{\left\{-\frac{1}{2}(z-z_{p1})^T\Sigma_1{}^{-1}(z-z_{p1})\right\}} \quad (5.17)$$

$$p_2(z) = \det(2\pi\Sigma_2)^{-\frac{1}{2}}.\ e^{\left\{-\frac{1}{2}(z-z_{p2})^T\Sigma_2{}^{-1}(z-z_{p2})\right\}} \quad (5.18)$$

If all the probability values are lower than the correspondence threshold, the algorithm evaluates the obtained sensor measurement as a new landmark. Otherwise it corresponds the measurement with the highest expected measurement.

The robot runs this computation for each landmark type. The only difference is the size of the sensor measurement and covariance matrix. All the sensor measurement vectors for different landmark vectors are listed below.

$$z_{plane\ as\ point} = [range \quad \alpha \quad \beta]^T \quad (5.19)$$

$$z_{plane\ as\ oriented\ point} = [range \quad \alpha \quad \beta \quad n_x \quad n_y \quad n_z]^T \quad (5.20)$$

$$z_{plane\ as\ surface} = [range \quad \alpha \quad \beta \quad n_x \quad n_y \quad n_z \quad area]^T \quad (5.21)$$

$$z_{plane\ as\ point} = [range \quad \alpha \quad \beta \quad scale \quad laplacian]^T \quad (5.22)$$

The data association process is the core of our study. In section 6 the effect of different landmark vectors on SLAM performance were demonstrated. The data association performance is the main cause of the performance differences between experiments.

Figure 5-3 shows the data association process.

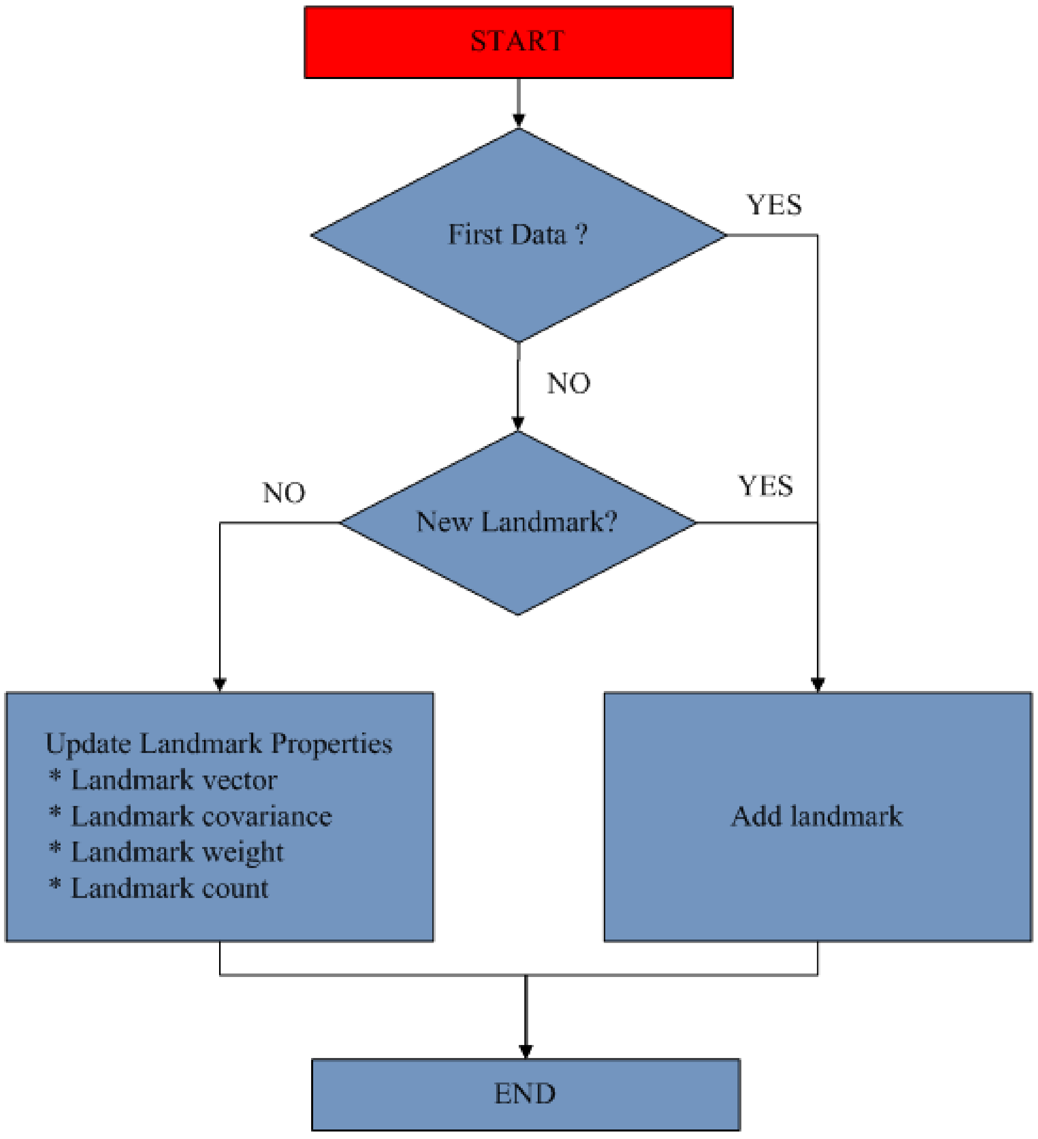


Figure 5-3 Data association process.

The code details about the association process are given in Appendix D.

5.2.9 Re-sampling Process

Re-sampling is the most important step in the fastSLAM algorithm. The robot keeps the landmark vectors independent from each other and at every step does not make any correction about unseen landmark vectors. However, this re-sampling step creates backward information passing by erasing the weak particles. The details of the re-sampling process are given in 2.7 and Appendix D.

5.2.10 Finalization Process

The proposed algorithm ends the simulation when the recorded dataset and the navigation path ends. At the end of the navigation, the proposed simulator calculates the error values according to the ground truth data. The simulator generates the error-algorithm step graph to show the performance of SLAM. The details about the GUI and final result visualization are given in Appendix C.

CHAPTER 6

EXPERIMENTS AND RESULTS

In this chapter, the effect of certain simulation parameters on the performance of the fastSLAM algorithm is evaluated using the 3D point cloud dataset and the pioneer robot. The final results are very impressive even though the simulated error is very high.

6.1 Experiment with the fre2_360 Map

In these experiments, the results are obtained using the **fre2_360** map. Different maps have been proposed for the robot SLAM algorithms; however, the **fre2_360** map is the simplest map for the chosen dataset.

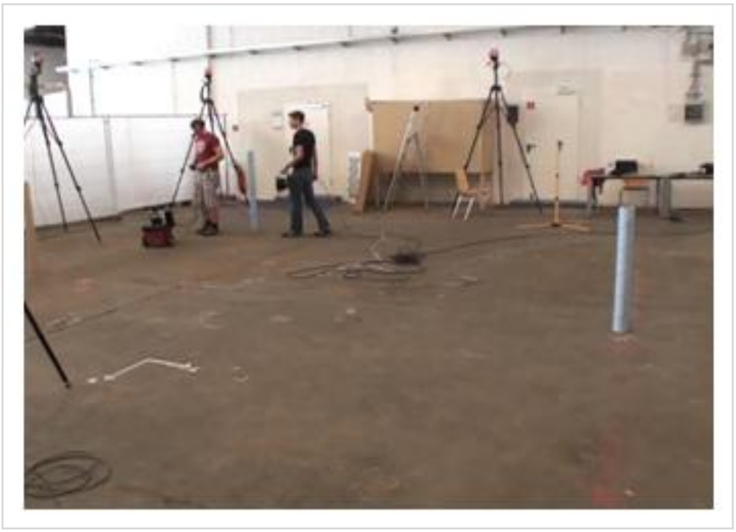


Figure 6-1 External view fre2_360 map [6].

The navigation environment is a closed area with columns and a chair. In the background, there are objects such as toys, a computer monitor and a plant, which are scanned by the Kinect sensor. In the simulations in the current study, the reliable range is less than 3.0 meters. Therefore, objects further than 3 meters *(toys, the monitor and the plant)* are neglected and they do not affect the simulations.

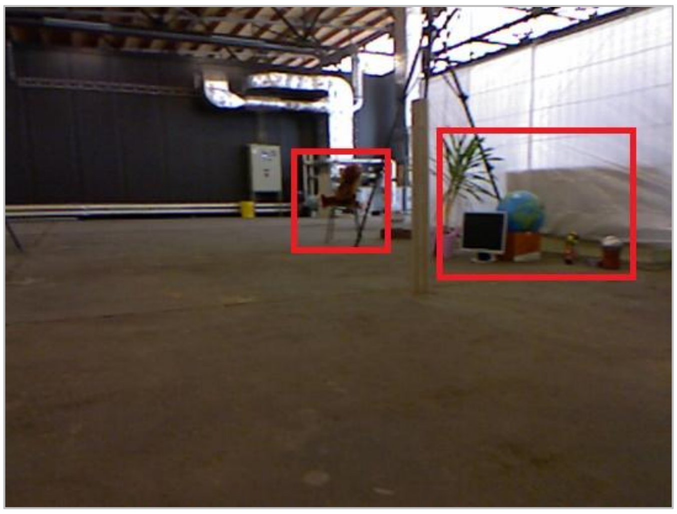


Figure 6-2 Neglected background objects [6].

In this environment, the robot detects 3 landmarks throughout the navigation. Figure 6-3 shows the detected landmarks and the navigation path.

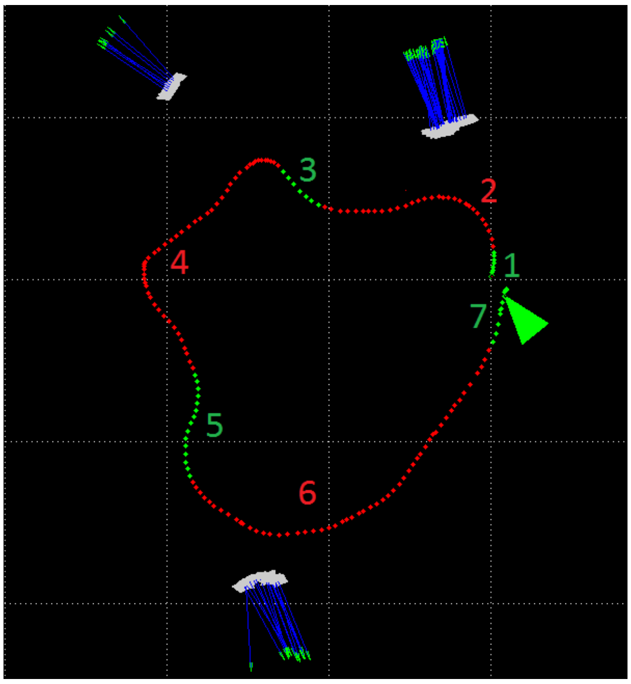


Figure 6-3 The ground truth of the vehicle and the detected planes. *(MATLAB Image).*

The explanation of the drawings in Figure 6-3 is given below:

- *Green dots: Scanning steps where the robot detects a landmark.*
- *Red dots: Scanning steps where the robot does not see any landmarks and moves only using the odometer data.*
- *White areas: Bird's-eye view of the plane.*
- *Blue Vectors: Normal vectors of the planes.*
- *Green triangle: The true pose of the vehicle*

The plane features are drawn according to the ground truth information. There is still a clear distribution as shown in Figure 6-4. This distribution is caused by errors in the sensor measurement, plane extraction algorithm and ground truth data.

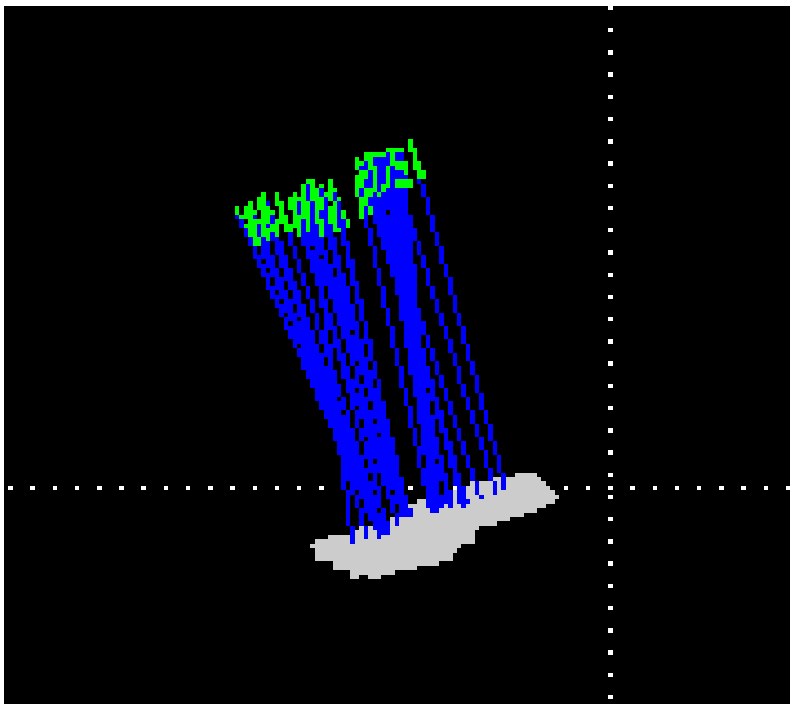


Figure 6-4 Distribution of the plane features. *(bird's-eye view) (MATLAB Image).*

Throughout the navigation, there are seven important phases as listed in Table 6-1. In phases 2, 4 and 6, the robot does not detect any landmarks. In these phases, the robot moves using only the predictions based on its control measurement. Since the navigation is performed without tracking any objects, there is a rapid increase in errors due to the miscalculation of the control measurement.

Table 6-1 Navigation summary of the Fre2_360 map

	Action	Running Algorithms	Result
1	Agent finds a new landmark.	Prediction-Correction	Slow increase in errors
2	No visible landmark	Prediction	Rapid increase in errors
3	Agent finds a new landmark.	Prediction-Correction	Slow increase in errors
4	No visible landmark	Prediction	Rapid increase in errors
5	Agent finds a new landmark.	Prediction-Correction	Slow increase in errors
6	No visible landmark	Prediction	Rapid increase in errors
7	Agent finds a landmark that was detected earlier.	Prediction-Correction	Important correction

Contrary to the other phases, in phases 1, 3, 5 and 7, the vehicle detects an object and runs the prediction and correction routines. In these phases of the navigation, errors increase slowly due to the re-sampling process. The algorithm calculates the weight of each particle, and performs re-sampling according to this information.

The correction in phase 7 is different from the others. In this phase, the robot closes the loop and calculates the particle weights according to the information that was gathered in phase 1. When the weight of the weak particles decreases, the algorithm removes the lighter particles through re-sampling. As a result of this process, the error correction rate improves remarkably.

Figure 6-5 presents the images of the environment throughout the navigation. As explained above, there are columns in the foreground and objects in the background. The sensor measurements further than 3.0 meters are not reliable for the Kinect sensor; therefore, the algorithm ignores these measurements and only detects the columns.

In the experiments using the **fre2_360** map, the robot navigates one turn in 100 algorithm steps. Steps **1-100** constitute the first loop, **101-200** define the second loop and **201-300** constitute the third loop. The robot detects three landmarks throughout the navigation. The first landmark is detected for the first time in step 1, and detected again in steps 88, 188 and 288. The robot detects the second landmark for the first time in step 25 and again in steps 125 and 225. The third landmark is detected first in step 53 and again in steps 153 and 253. Detecting the landmarks again, the robot can make successful corrections in the later steps.

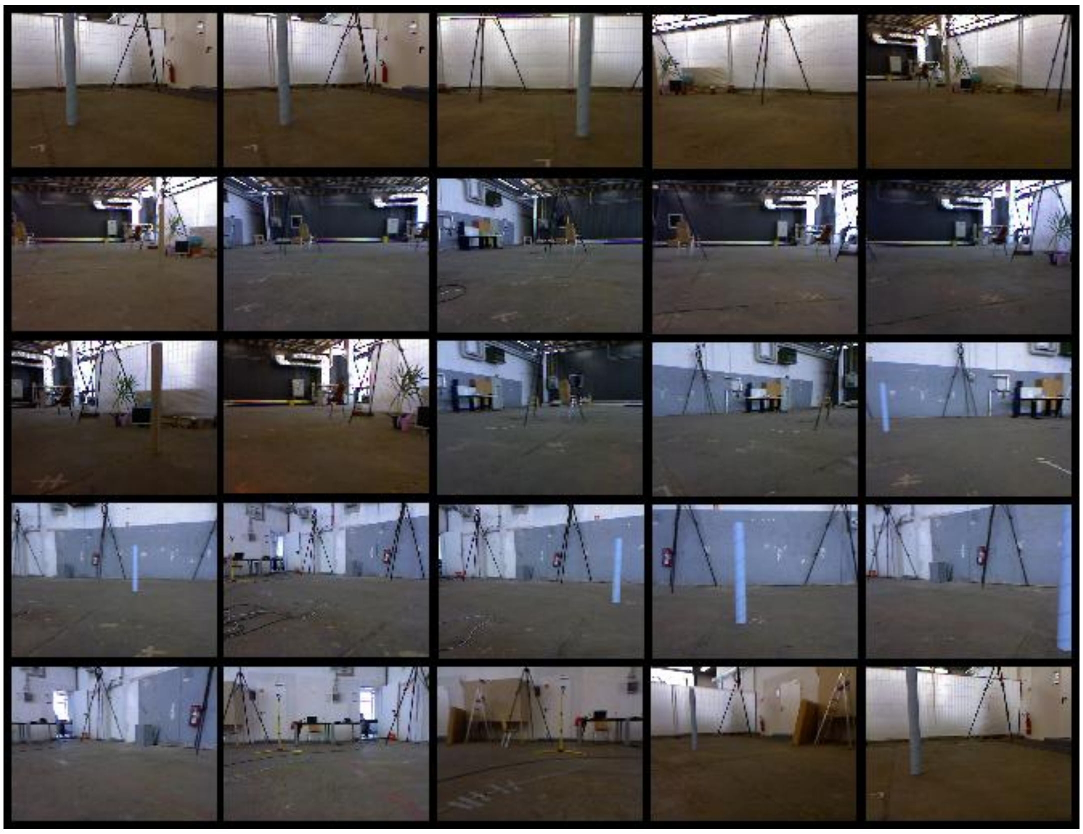


Figure 6-5 The map environment throughout the data collection process [6]. *(Edited with Paint)*

In this study, four different experiments were carried out using the **fre2_360** map. The only difference between the experiments concerns the landmark vector, which defines the features of objects in feature-based slam applications. The explanation of these landmark vectors is as follows;

Surf Point: Defines the location, scale and laplacian of the detected SURF feature.

$$SURF\ Point = [x \quad y \quad z \quad scale \quad laplacian\ ]^T \tag{6.1}$$

Plane as Point: Defines the center point of the detected plane.

$$Plane\ as\ point = [x \quad y \quad z\ ]^T \tag{6.2}$$

Plane as Oriented Point: Defines the center point and normal vector of the plane.

$$Plane\ as\ oriented\ point = [x \quad y \quad z \ \ n_x \quad n_y \quad n_z]^T \tag{6.3}$$

Plane as Surface: Defines the center point, normal vector and area of the plane.

$$Plane\ as\ Surface = [x \quad y \quad z \ \ n_x \quad n_y \quad n_z \quad area\ ]^T \tag{6.4}$$

All the simulation parameters used in the experiments are listed in Table 6-2.

Table 6-2 Simulation parameters for experiments using the Fre2_360 Map

	Exp. 1	**Exp. 2**	**Exp. 3**	**Exp. 4**
Landmark Vector	SURF Point	Plane as Point	Plane as Oriented Point	Plane as Surface
Number of Particles	100	100	100	100
Effective Particle Rate (%)	90%	90%	90%	90%
Correspondence Threshold	0.00001	0.00001	0.00001	0.00001
Scanning Step	10	10	10	10
Linear Velocity Errors (%)	10%	10%	10%	10%
Angular Velocity Errors (%)	10%	10%	10%	10%

As stated above, the only difference between the four experiments is the landmark vector. The fastSLAM algorithm is performed 150 times for each landmark vector. The final translational and rotational errors for each experiment are given in Table 6-3.

Table 6-3 The final results of errors *(Mean of 100 particles and 150 simulations)*

Landmark Vector	$\boldsymbol{error_{translational}(m)}$	$\boldsymbol{error_{rotational}(radian)}$
SURF Point	0.6831	0.3292
Plane as Point	1.6242	0.6792
Plane as Oriented Point	0.8939	0.4872
Plane as Surface	0.8421	0.4159

The errors indicate the mean of the error values of all particles and simulations and are calculated according to the following rule:

$$Error = \frac{TOTAL\ ERROR}{(\ ITERATION\ x\ PARTICLE\ NUMBER\ )} \tag{6.5}$$

A comparison of the final error values given in Table 6-3 is as follows;

$$\boldsymbol{err_{SURF\ Point} < err_{Plane\ as\ Surface} < err_{Plane\ as\ oriented\ point} < err_{plane\ as\ point}}$$

In experiment-1, the SURF points are used as the landmark vector for the SLAM algorithm. The SURF feature detector finds a high number of features at the edges of the objects, providing the best performance for the SLAM algorithm.

Other landmarks used in the remaining three experiments help define the planar features *(the center point, normal vector and area)* of the objects. In experiment-2, the algorithm runs using the **plane as point** landmark vector, which carries the center point of planar surfaces. In experiment-3, the **plane as oriented point** landmark vector is used and the center point and normal vector of planar surfaces are defined. In experiment-4, the algorithm uses the **plane as surface** landmark vector, which contains the information about the center point, orientation and area of the plane. According to the final results in Table 6-3, the size of the landmark vector and final errors is inversely proportional in all landmark vectors and the best result is obtained from the **plane as surface** landmark vector.

Time consumption is another important factor for SLAM applications. In Table 6-4, the number of the detected landmarks and the time taken for their detection are given in relation to each landmark vector.

Table 6-4 The results of the simulation *(The comparison of Landmark Vectors according to Time Consumption and Number of Landmarks)*

Landmark Vector	***Landmarks***	***Time(seconds)***
SURF Point	36	3935
Plane as Point	3	931
Plane as Oriented Point	3	925
Plane as Surface	3	915

The number of landmarks and time consumption are calculated according to the following rules;

$$Landmarks = \frac{TOTAL\ LANDMARKS}{(\ ITERATION\ x\ PARTICLE\ NUMBER\ )} \tag{6.6}$$

$$Time = \frac{TOTAL\ TIME}{ITERATION} \tag{6.7}$$

According to the results, the SURF landmark vector produces a lower error rate than the other landmark vectors. However, using the SURF landmark vectors results in a higher time consumption due to the high number of landmarks, which reached 36. One of the SLAM simulations using the SURF landmark vectors lasted 3935 seconds instead of 931 seconds, which means that defining the SURF features takes almost 4 times *(*3935 */ 931 = 4,22)* longer than in the use of other three vectors. On the other hand, there were small differences between the planar features in terms of time consumption due to the re-sampling process. The algorithm runs the re-sampling process more frequently if the estimation of the robot diverges from the ground truth. The **plane as surface** landmark vector quickly converges to the true pose with the help of the best correspondence performance obtained from the landmark vectors.

The final results in terms of error and time are directly proportional to those obtained from the planar landmark vectors.

$$\boldsymbol{time_{Plane\ as\ Surface}} < \boldsymbol{time_{Plane\ as\ oriented\ point}} < \boldsymbol{time_{plane\ as\ point}}$$

The time difference between the SURF point and other landmark vectors is due to the structure of the fastSLAM algorithm. As a result of running more data association functions due to the high number of particles involved in the **SURF point** landmarks, the robot keeps 36 landmarks for each particle, instead of 3. The feature extraction time is another cause for the difference in time consumption between the experiments. The comparison of the time taken for the feature extraction methods is given in Table 6-5.

Table 6-5 Comparison of the time taken for the feature extraction methods.

Feature Extraction Method	Time *(seconds)*
Plane Feature Extraction	3.3229
SURF Feature Extraction	7.9377

The SURF feature extraction method takes almost two times *(7.9377 / 3.3229)* longer than the plane feature extraction method. The SURF feature is not suitable for indoor real-time SLAM applications due to the high time consumption.

The performance of experiments is also analyzed throughout the navigation. Figures 6-6 and 6-7 show the graphical representation of the translational and rotational errors for the three-loop navigation throughout the motion path.

Before the loop-closure *(until step 88),* all the experiments have similar results. According to Figure 6-6 and Figure 6-7, the **plane as oriented point** and **plane as surface** landmark vectors cause higher errors in some areas compared with the other methods due to an algorithmic error that occurred in the plane detection process before step 88. However, the difference in the error rates is insignificant and thus, can be ignored. Consequently, the type of the landmark vector does not have a significant impact on the performance of the SLAM before the loop-closure.

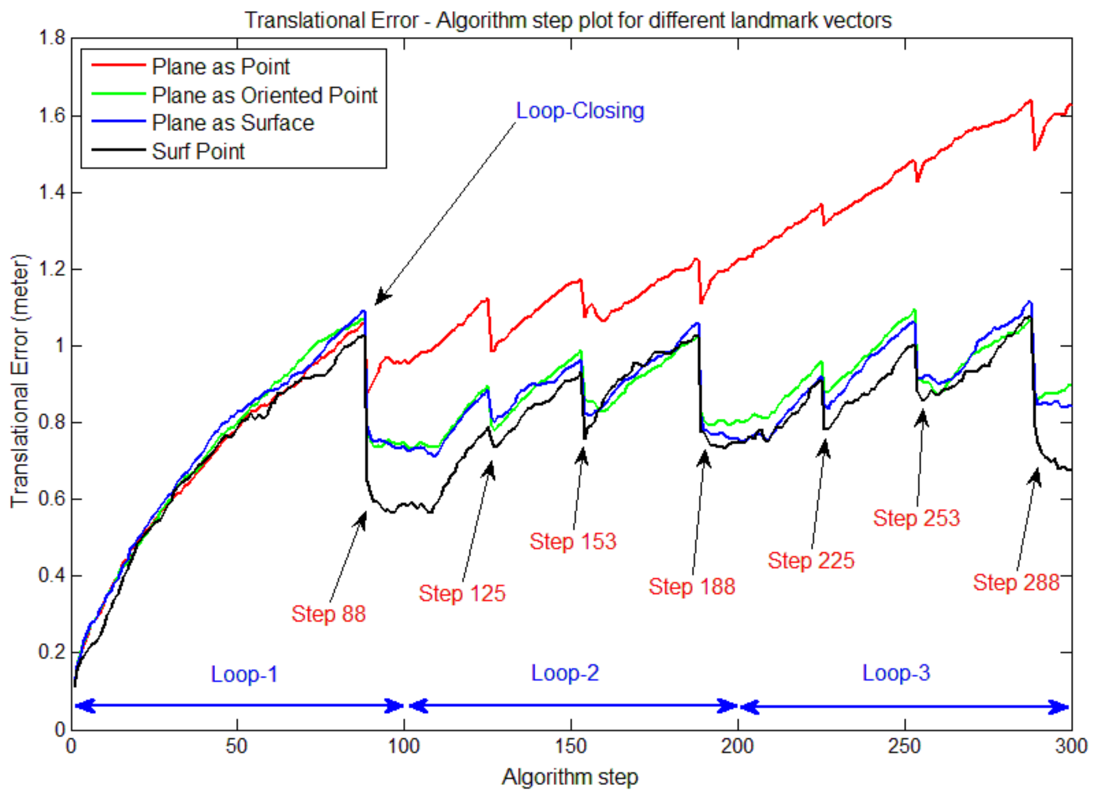


Figure 6-6 Translational error – the algorithm step plot for different landmark vectors. *(Values corresponding to the mean of 100 particles) (MATLAB Image).*

After the loop-closure, as a result of using a high number of landmarks, the **SURF point** landmark vector provides the best result and a better correction rate compared with the other planar landmarks. There are three landmarks on the fre2_360 map. The robot detects the first landmark for the first time in step 1 and again in steps 88, 188 and 288. The second landmark is detected for the first time in step 25, and again in steps 125 and 225. The third landmark is detected first in step 53 and again in steps 153 and 253. The most important step throughout the navigation is step 88, where the robot detects the first landmark for the second time and closes the navigation path. With this loop-closure, there is a remarkable improvement in the correction rate.

Figure 6-7 shows the graphical representation of rotational errors for the three-loop navigation throughout the motion path.

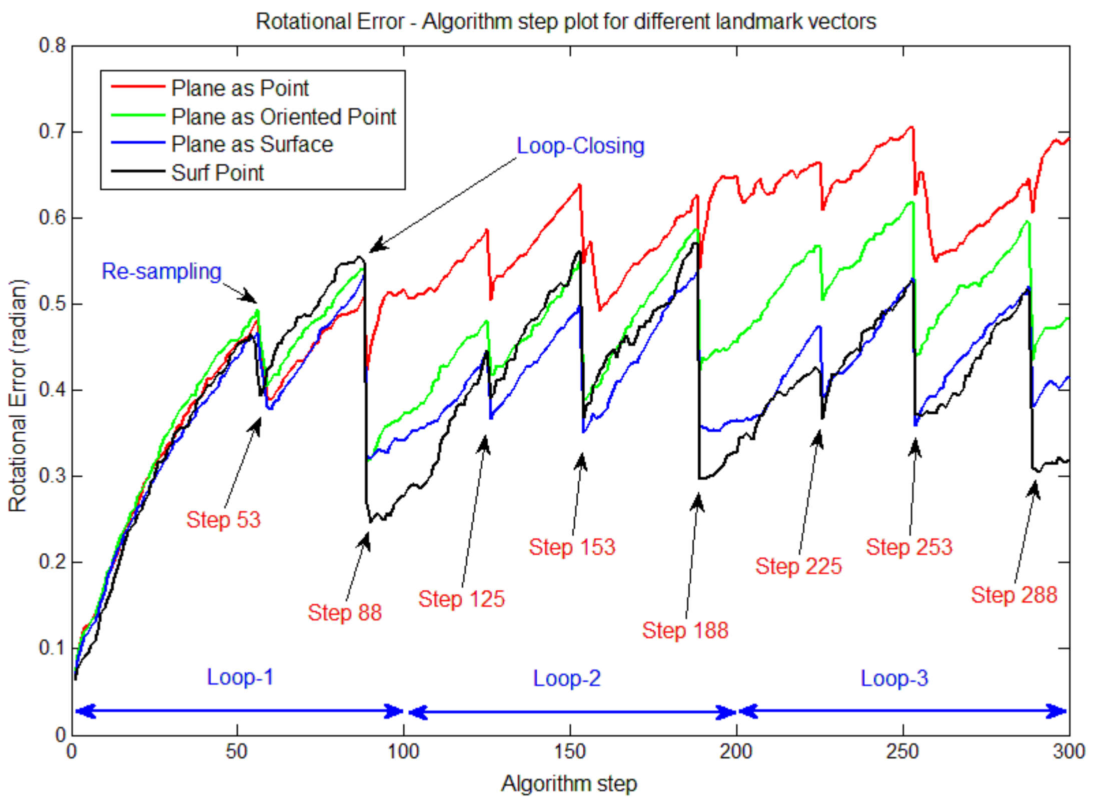


Figure 6-7 Rotational error – the iteration plot for different landmark vectors. *(Values corresponding to the mean of 100 particles) (MATLAB Image).*

The rotational error plots are almost the same as translational error plots. All the statements that are made for translational errors are also valid for rotational errors. Before the loop-closure *(until step 88)*, all the experiments give similar results in terms of errors. The landmark vector does not have any significant effect on the performance of the SLAM before the loop-closure. However, after the loop-closure, the **SURF point** landmark vector produces the best result due to the high number of landmarks, allowing the algorithm to define the surrounding objects with a higher number of properties and to provide a low rotational error for the planar landmarks. The root cause of this final result is the correspondence performance of different landmark vectors.

In step 88, the robot detects the first landmark for the second time and closes the navigation path. With the loop-closure, the correction rate improves. In steps 125, 153, 188, 225, 253 and 288, the robot detects the same landmarks again and decreases the errors. Also, in step 53, since the effective particle rate decreases below the predefined threshold (90%), the robot runs the re-sampling process. The effective particle rate plot according to the algorithm steps is given in Figure 6-8.

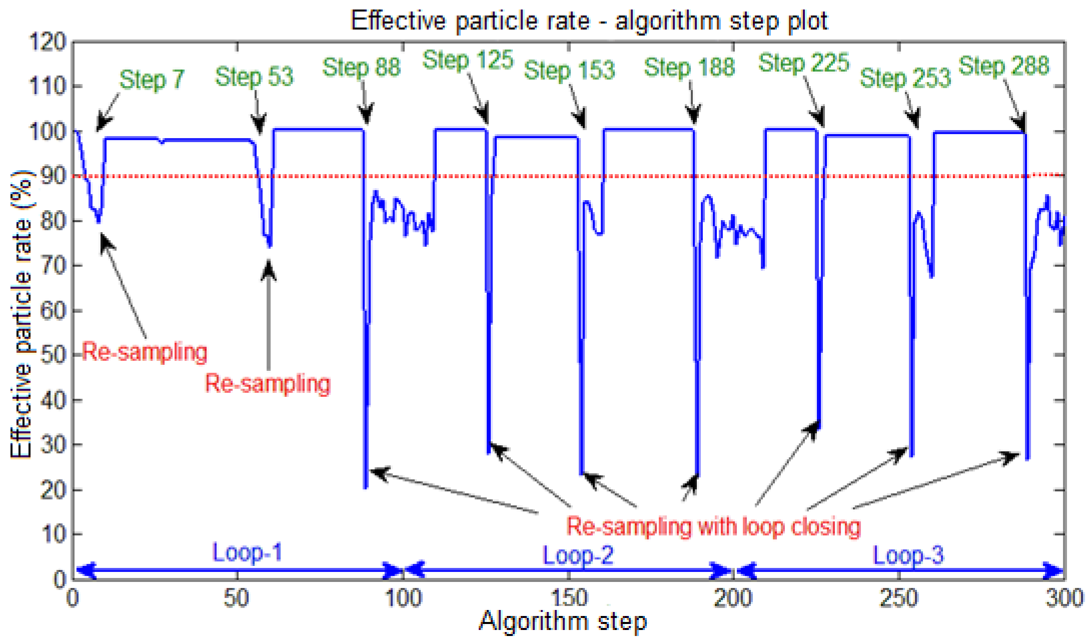


Figure 6-8 Effective particle rate plot for navigation on the fre2_360 map.

Table 6-6 presents the final error values of the **plane as point** and **plane as surface** landmark vectors and gives a comparison of their numerical performance.

Table 6-6 Comparison of the final performance of SLAM experiments.

Final Translational Errors				**Final Rotational Errors**			
E_{point}	$E_{surface}$	Δ	$\frac{\Delta}{E_{point}}$	E_{point}	$E_{surface}$	Δ	$\frac{\Delta}{E_{point}}$
1.6242	0.8421	0.7821	48%	0.6792	0.4159	0.2978	38%

The differences between the landmark vectors in terms of error and correction rates are calculated using the following formulas;

Δ : E_{point} - $E_{surface}$

$$\Delta > 0 \implies Plane\ as\ surface\ landmark\ vector\ decreased\ the\ errors$$

$$\Delta < 0 \implies Plane\ as\ surface\ landmark\ vector\ increased\ the\ errors$$

where;

$\boldsymbol{E_{point}}$ is the total number of errors obtained from the **plane as point** landmark vector,

$\boldsymbol{E_{surface}}$ is the total number of errors obtained from the **plane as surface** landmark vector, and

$\frac{\Delta}{\boldsymbol{E_{point}}}$ is the proportional evaluation of the decrease or increase in errors. This formula gives the rate of decrease in errors compared with the initial values *(obtained by using the point landmarks)*.

The **plane as surface** landmark vector, which carries more properties of the planar surfaces, provides a higher decrease in translational errors (48%) and rotational errors (38%) compared with the **plane as point** landmark vector, which only carries the center point.

As a result, the **SURF point** landmark vector provides better results than the other three vectors in terms of translational and rotational errors. However, the computational cost of the **SURF point** landmark vector is very high since the SURF interest point detector gives high responses on the edges of the depth image. Therefore, the use of **SURF point** landmark vectors for real-time indoor fastSLAM applications is not appropriate. The **SURF point** landmark vectors that are ignored in the remaining experiments and the contribution of planar features are investigated using the **plane as point**, **plane as oriented point** and **plane as surface** landmark vectors.

According to the results, the **plane as surface** landmark vector outperforms the other planar landmark vectors.

$$\boldsymbol{err_{Plane\ as\ Surface} < err_{Plane\ as\ oriented\ point} < err_{plane\ as\ point}}$$

The number of features contained by the **plane as surface, plane as oriented point, and plane as point** landmark vectors are 7, 6 and 3, respectively. This shows that a SLAM algorithm with more features provides a better SLAM performance.

6.2 The Effect of Parameters

6.2.1 The Effect of Linear Velocity Errors

This section demonstrates the effect of linear velocity errors on the performance of different landmark vectors. The error rates and other simulation parameters are listed in Table 6-7.

Table 6-7 Simulation Parameters.

	Experiment 1	**Experiment 2**	**Experiment 3**
Landmark Vector	Plane as Point	Plane as Oriented Point	Plane as Surface
Number of Particles	100	100	100
Effective Particle Rate (%)	90%	90%	90%
Correspondence Threshold	0.00001	0.00001	0.00001
Scanning Step	10	10	10
Linear Velocity Errors (%)	3%, 7%, 15%	3%, 7%, 15%	3%, 7%, 15%
Angular Velocity Errors (%)	10%	10%	10%

There are nine scenarios *(3x3)* for three different landmark vectors and three different linear velocity error rates. Figure 6-9 and Figure 6-10 show the effect of the linear velocity error rate on the SLAM performance before and after the loop-closure. According to these figures, the linear velocity error rate does not have any effect on the rotational errors before the loop-closure. So, the rotational errors are independent from the linear velocity errors before the loop-closure. All the experiments produce similar rotational error rates regardless of the rate of the injected linear errors. However, the linear velocity error rate has a significant effect on the translational error rate, which is directly proportional to the rate of the injected linear velocity errors. As explained above, this is valid for steps 1 to 88. In this interval, the planar features *(normal vector and area)* do not contribute to the performance of the SLAM simulation. In other words, the SLAM performance is independent from the landmark vectors before the loop-closure.

On the other hand, the principal effect of landmark vectors on the SLAM performance is prominent after the loop-closing process. Even though the error characteristics are very similar, there are some differences in the details.

Figure 6-9 shows the effect of the linear velocity error rate on the translational errors. In step 88, the robot detects the first landmark for the second time and decreases the translational errors through re-sampling. Therefore, steps 88, 125, 153, 188, 225, 253, and 288 are powerful steps due to the loop closure and the re-sampling process. A comparison of the translational error rates after the loop-closure is given below:

$$\boldsymbol{err}(\mathbf{3}\%)_{\boldsymbol{Plane\ as\ Surface}} \approx \boldsymbol{err}(\mathbf{3}\%)_{\boldsymbol{Plane\ as\ oriented\ point}} > \boldsymbol{err}(\mathbf{3}\%)_{\boldsymbol{plane\ as\ point}}$$

$$\boldsymbol{err}(\mathbf{7}\%)_{\boldsymbol{Plane\ as\ Surface}} \approx \boldsymbol{err}(\mathbf{7}\%)_{\boldsymbol{Plane\ as\ oriented\ point}} < \boldsymbol{err}(\mathbf{7}\%)_{\boldsymbol{plane\ as\ point}}$$

$$\boldsymbol{err}(\mathbf{15}\%)_{\boldsymbol{Plane\ as\ Surface}} \approx \boldsymbol{err}(\mathbf{15}\%)_{\boldsymbol{Plane\ as\ oriented\ point}} < \boldsymbol{err}(\mathbf{15}\%)_{\boldsymbol{plane\ as\ point}}$$

The **plane as a point** landmark vector provides better results (with an error rate of 3%) than the other landmark vectors. This is due to the algorithmic errors injected to the system by the **plane as oriented point** and **plane as surface** landmark vectors.

The **plane as oriented point** and **plane as surface** landmark vectors are successful in making corrections for the 7% and 15% error rates. The correction rate of the **plane as point** landmark vector is satisfactory for some intervals; however, the error rate is still higher than the rate obtained from the other landmark vectors. The correction rate of the **plane as point** landmark vector is particularly low for the 15% error rate. Since the robot corrects only a small percentage of the particles, and erases the corrected particles in the following steps, the rate of translational errors increases again.

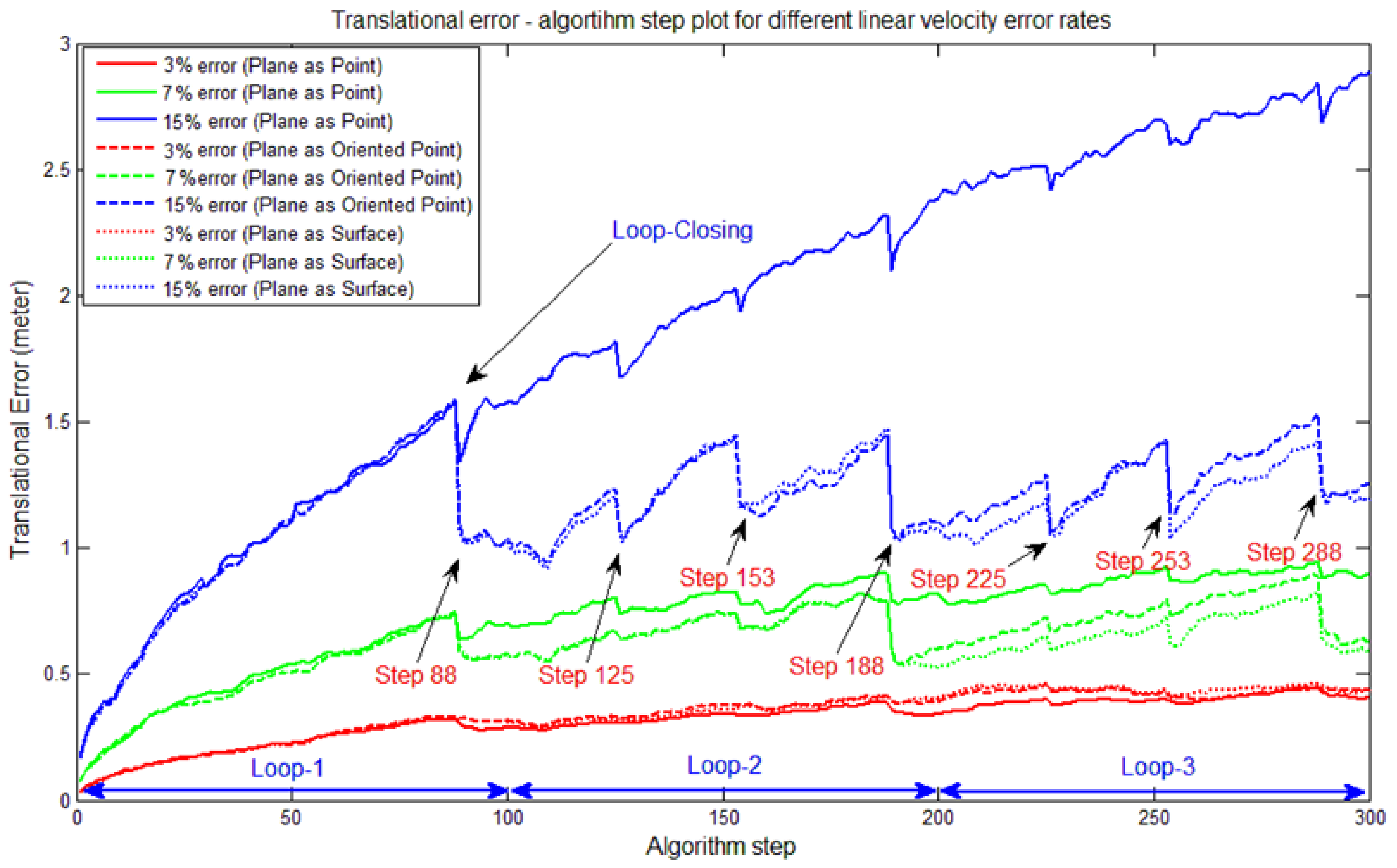


Figure 6-9 The effect of the linear velocity error rate on translational errors *(MATLAB Image)*.

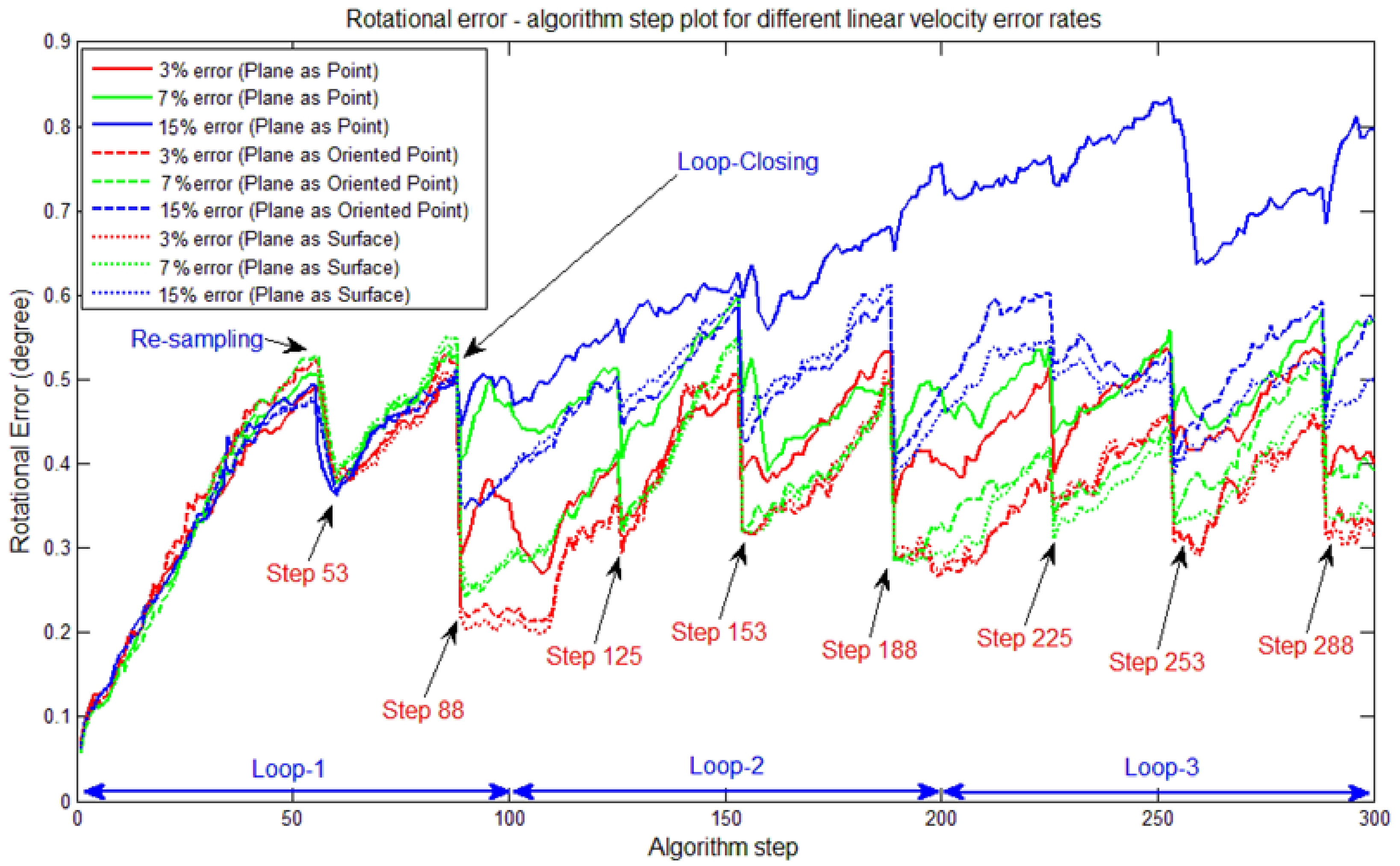


Figure 6-10 The effect of the linear velocity error rate on rotational errors *(MATLAB Image)*.

Figure 6-10 shows the effect of the linear velocity error rate on rotational errors. In step 53, the robot runs the re-sampling process correcting the errors in all experiments. In step 88, the first landmark is detected for the second time and the number of rotational errors increases as a result of the re-sampling process. Steps 88, 125, 153, 188, 225, 253, and 288 are important and effective steps since they involve the re-sampling process. A comparison of the rotational error rates after the loop-closure is given below:

$$err_{Plane\ as\ Surface} < err_{Plane\ as\ oriented\ point} < err_{plane\ as\ point}$$

This comparison is valid for all error rates after the loop-closure. However, the correction performance differs according to the rate of the injected errors. The **plane as oriented point** and **plane as surface** landmark vectors provide effective corrections even if the linear velocity error rate is 15%. The **plane as point** landmark vector provides corrections for the 7% error rate, but cannot produce satisfactory results when the error rate increases to 15%.

Table 6-8 presents the final error values and of the **plane as point** and **plane as surface** landmark vectors and gives a comparison of their numerical performance.

Table 6-8 A comparison of the final performance of SLAM experiments

		Final Translational Errors				**Final Rotational Errors**			
		E_{point}	$E_{surface}$	Δ	$\frac{\Delta}{E_{point}}$	E_{point}	$E_{surface}$	Δ	$\frac{\Delta}{E_{point}}$
Error Rate	**3%**	0.4085	0.4419	-0.0334	-8%	0.4007	0.3154	0.0853	21%
	7%	0.8920	0.5888	0.3032	34%	0.5722	0.3420	0.2302	40%
	15%	2.8880	1.1977	1.6903	59%	0.7958	0.4981	0.2978	38%

Both landmark vectors provide similar results for translational errors when the rate of the injected linear velocity errors is 3%. The robot follows the right path and the correspondence performance of the landmark vectors does not have any impact on the final result of the translational errors. However, the rate of the injected angular velocity errors is 10% for all the experiments. The **plane as point** landmark vector does not carry any rotational information. The **plane as surface** landmark vector carries rotational information and corrects 21% of errors even if the injected linear velocity error rate is 3%. The rotational correction increases to 40% when the injected linear velocity error is higher than 7%.

The difference between experiments arises when the injected linear velocity error rate increases to %15. For the %15 linear velocity error rate, the **plane as point** landmark vector cannot make correct correspondences; thus, the error correction rate using **plane as surface** landmark vector reaches 59% for translational errors and 40% for rotational errors.

6.2.2 The Effect of Angular Velocity Errors

This section demonstrates the performance of different landmark vectors for different rates of angular velocity errors. The error rates and other simulation parameters are listed in Table 6-9.

Table 6-9 Simulation Parameters

	Experiment 1	**Experiment 2**	**Experiment 3**
Landmark Vector	Plane as Point	Plane as Oriented Point	Plane as Surface
Number of Particles	100	100	100
Effective Particle Rate (%)	90	90	90
Correspondence Threshold	0.00001	0.00001	0.00001
Scanning Step	10	10	10
Linear Velocity Errors (%)	10%	10%	10%
Angular Velocity Errors (%)	3%, 7%, 15%	3%, 7%, 15%	3%, 7%, 15%

There are nine scenarios *(3x3)* for three different landmark vectors and three different rates of angular velocity errors. Figure 6-11 and Figure 6-12 show the effect of the angular velocity error rate on the SLAM performance before and after the loop-closure.

According to Figure 6-11 and Figure 6-12, the rate of angular velocity errors does not have any effect on the translational errors before the loop-closure. In other words, the translational errors are independent from the angular velocity error before the loop-closure. All the experiments produce similar results in term of translational errors regardless of the rate of the injected angular velocity errors. However, the angular velocity errors have a significant effect on the rotational error rate, which is directly proportional to the rate of the injected angular velocity errors. As stated before, this is valid for steps 1 to 88. In this interval, the planar features *(normal vector and area)* do not contribute to the performance of the SLAM simulation, so the SLAM performance is independent from the landmark vectors before the loop-closure.

On the other hand, the landmark vectors start to affect the SLAM performance after the loop-closure. Even though the error characteristics are very similar, there are some differences in the details.

Figure 6-11 and Figure 6-12 show the effect of the angular velocity error rate on the SLAM performance. In step 53, the robot runs the re-sampling process and corrects errors in all experiments. In step 88, the robot detects the first landmark for the second time and decreases the number of translational errors through re-sampling. Steps 88, 125, 153, 188, 225, 253, and 288 are important since they include the loop-closure and re-sampling processes. A comparison of the error rates after the loop-closure is as follows:

$$\boldsymbol{err_{Plane\ as\ Surface} < err_{Plane\ as\ oriented\ point} < err_{plane\ as\ point}}$$

This comparison is valid for all error rates after the loop-closing process. However, the performance differs according to the rate of the injected errors.

For the translational errors, the **plane as point** landmark vector provides corrections for the 3% and 7% angular velocity error rates. However, when the angular velocity error rate increases to 15%, the **plane as point** landmark vector is not able to make successful corrections and thus the number of errors increases with motion. On the other hand, the **plane as surface** and **plane as oriented point** landmark vectors provide satisfactory corrections for all error rates and the errors do not increase with motion.

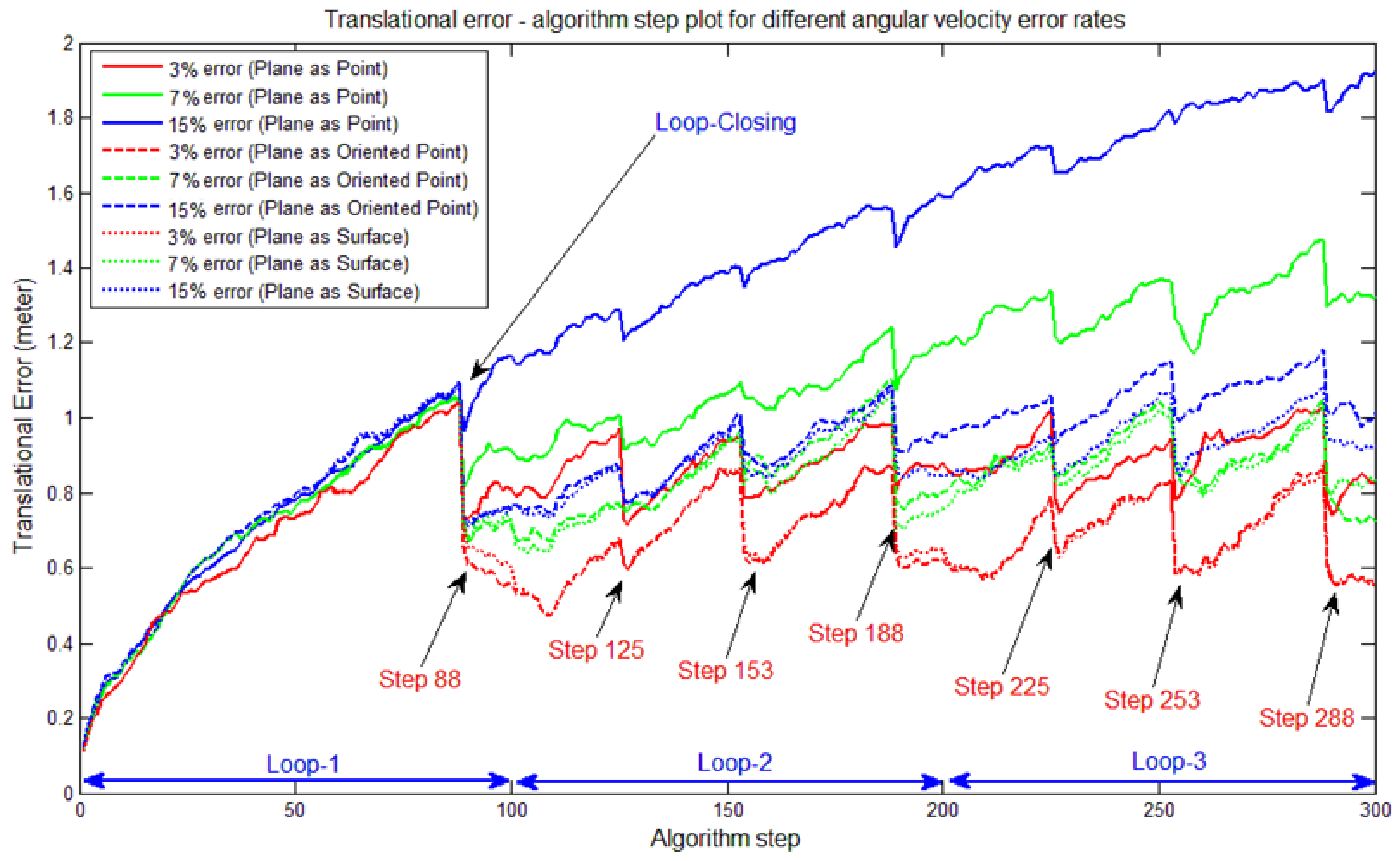


Figure 6-11 The effect of the angular velocity error rate on translational errors *(MATLAB Image)*

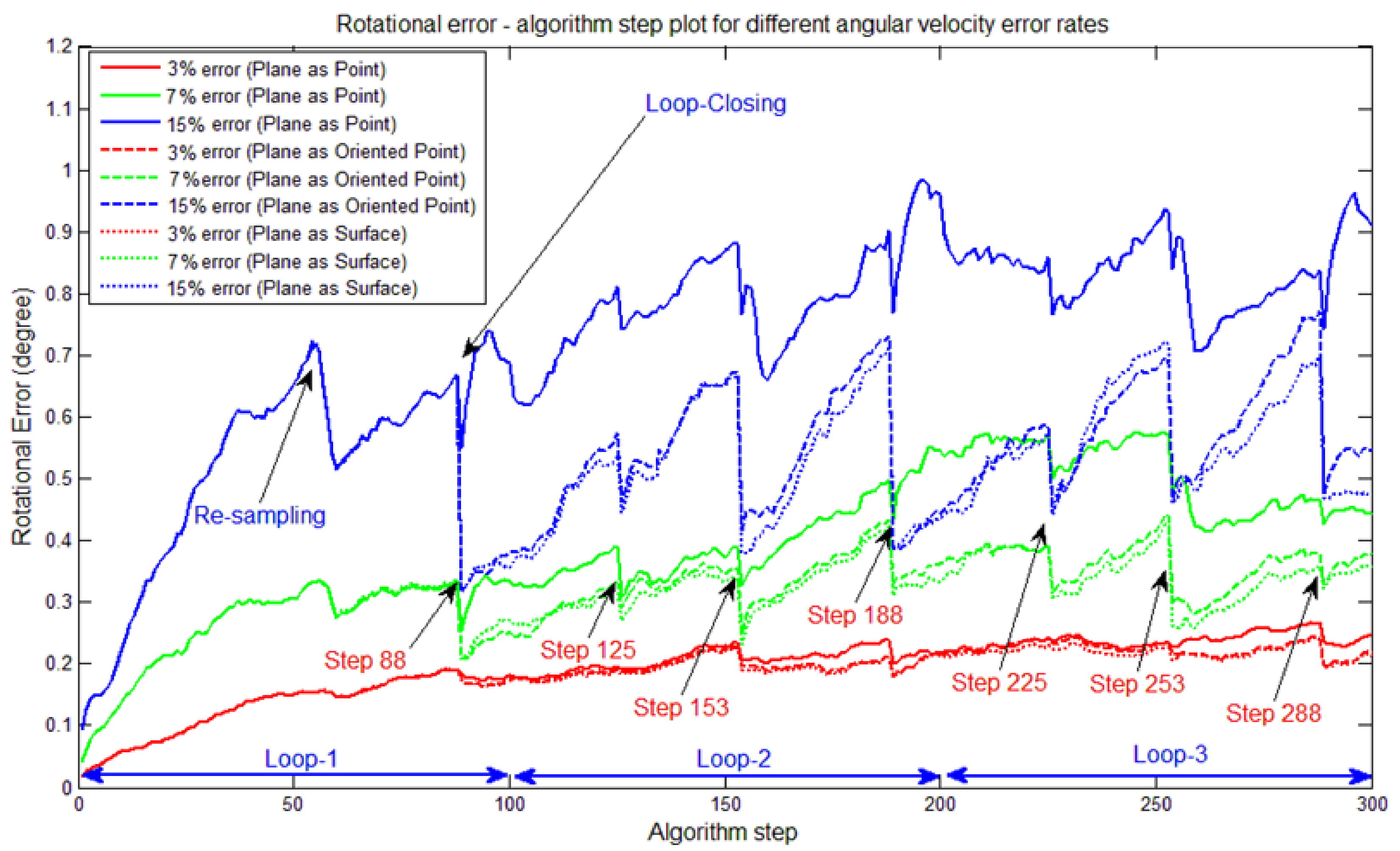


Figure 6-12 The effect of the angular velocity error rate on rotational errors *(MATLAB Image)*

For rotational errors, the **plane as a point** landmark vector with the 3% angular velocity error rate provides similar results to those obtained from the other landmark vectors due to the low rate of the injected angular velocity errors.

The **plane as oriented point** and **plane as surface** landmark vectors provide successful corrections for the 7% and 15% error rates. The **plane as point** landmark vector provides satisfactory results for some intervals, but the number of errors is still higher than those obtained from the other landmark vectors. The **plane as point** landmark vector provides the lowest correction for the 15% error rate since the robot corrects only a small percentage of the particles and erases the corrected particles in the following steps, which increases the number of errors again.

Table 6-10 presents the final error values of the **plane as point** and **plane as surface** landmark vectors and gives a comparison of their numerical performance.

Table 6-10 A comparison of the final performance of SLAM experiments

		Final Translational Errors				**Final Rotational Errors**			
		E_{point}	$E_{surface}$	Δ	$\frac{\Delta}{E_{point}}$	E_{point}	$E_{surface}$	Δ	$\frac{\Delta}{E_{point}}$
Error Rate	**3%**	0.8249	0.5541	0.2708	33%	0.2450	0.2127	0.0323	13%
	7%	1.3179	0.8285	0.4894	37%	0.4454	0.3588	0.0866	19%
	15%	1.9245	0.9277	0.9969	52%	0.9088	0.4734	0.4354	48%

Both landmark vectors give similar results in terms of rotational errors when the rate of the injected angular velocity errors is 3%. However, the injected linear velocity error rate is 10%. The plane as surface landmark vector corrects 33% of the errors even when the injected angular velocity error rate is 3%.

A difference between the experiments arises when the injected angular velocity error rate increases to %15. The **plane as point** landmark vector with a 15% angular velocity error rate cannot make the correct correspondences; however, the correction rate of the **plane as surface** landmark vector can reach 48% for rotational and 52% for translational errors.

6.2.3 The Effect of Particle Number

This section demonstrates the performance of different landmark vectors for different number of particles. The particle numbers and other simulation parameters are listed in Table 6-11.

Table 6-11 Simulation Parameters

	Experiment 1	**Experiment 2**	**Experiment 3**
Landmark Vector	Plane as Point	Plane as Oriented Point	Plane as Surface
Number of Particles	20, 100	20, 100	20, 100
Effective Particle Rate (%)	90%	90%	90%
Correspondence Threshold	0.00001	0.00001	0.00001
Scanning Step	10	10	10
Linear Velocity Errors (%)	10%	10%	10%
Angular Velocity Errors (%)	10%	10%	10%

There are six scenarios *(3x2)* for three different landmark vectors and two different particle numbers. Figure 6-13 and Figure 6-14 show the effect of each particle number on the SLAM performance before and after the loop closure.

The SLAM performance is independent from the particle number before the loop-closure. All the experiments give similar results in terms of translational and rotational errors for different number of particles between steps 1-88. In this interval, the **plane as surface** and **plane as oriented point** landmark vectors do not provide any extra contribution to the performance of the SLAM simulation. In step 53, the robot runs the re-sampling process and decreases the number of rotational errors in all experiments.

The effect of landmark vectors on the SLAM performance is prominent after the loop-closing process. The error comparison for translational errors and rotational errors are the same. The final error comparison after the loop-closure is as follows:

$$\boldsymbol{err_{Plane\ as\ Surface} < err_{Plane\ as\ oriented\ point} < err_{plane\ as\ point}}$$

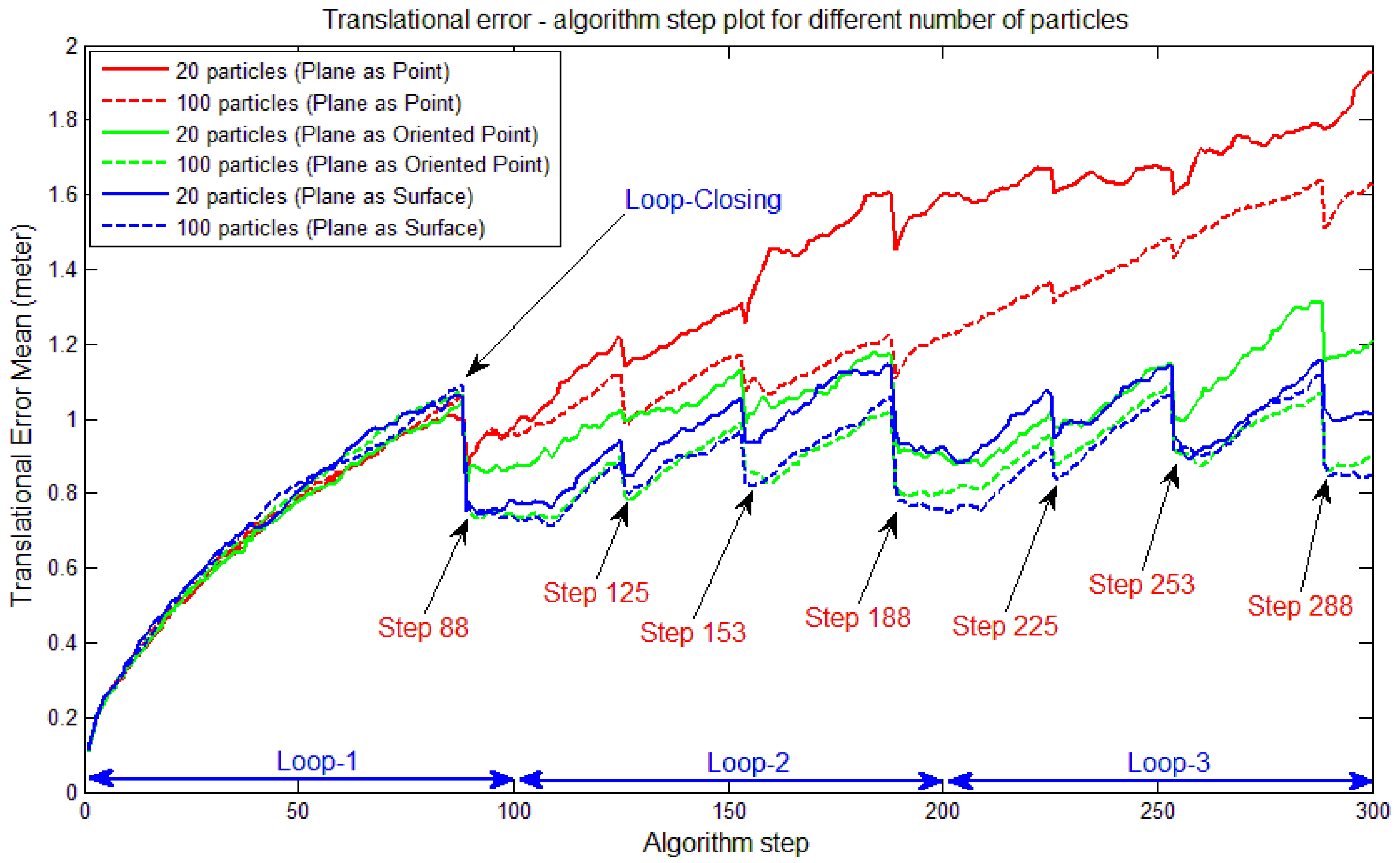


Figure 6-13 Effect of particle number on translational errors *(MATLAB Image)*

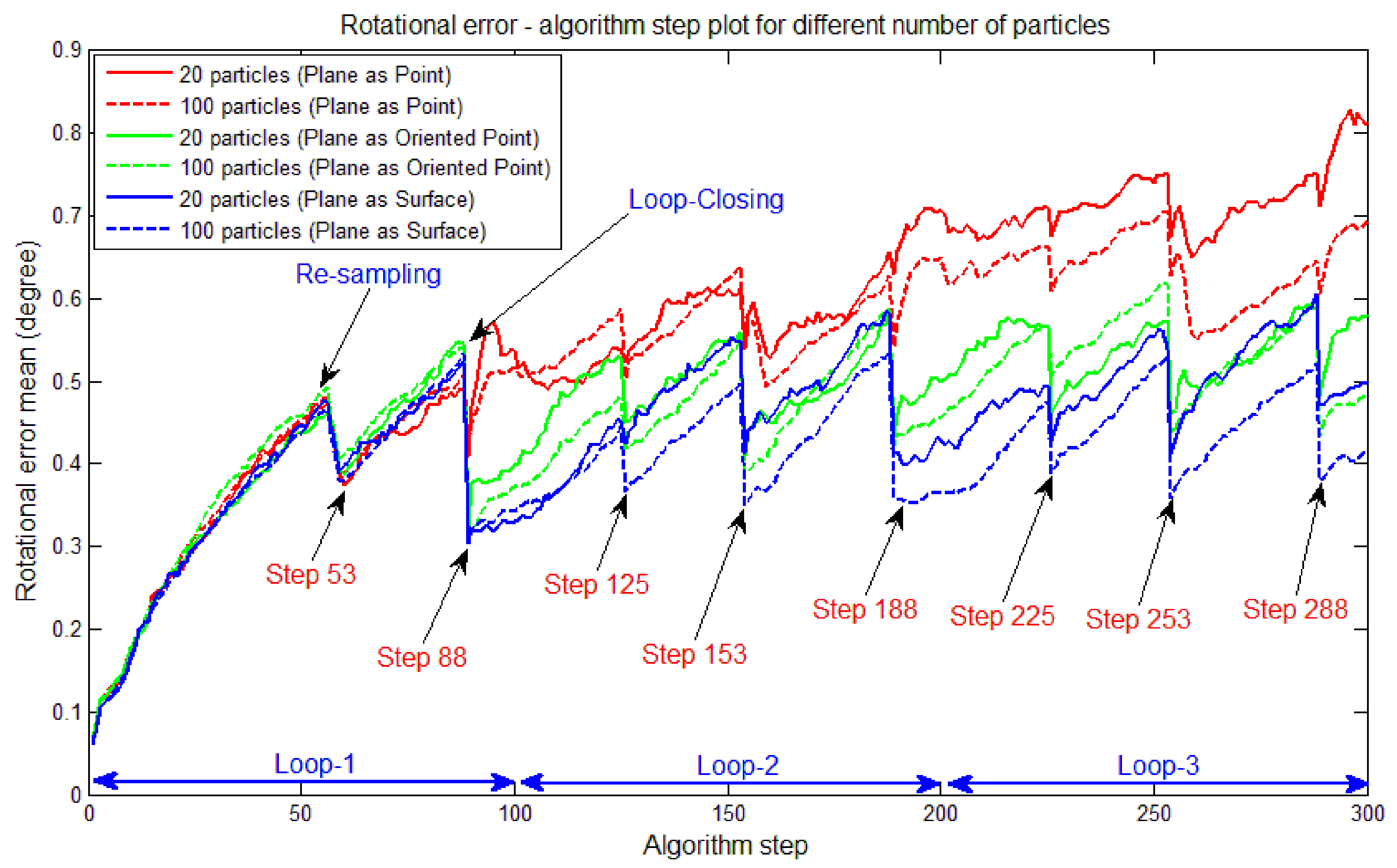


Figure 6-14 Effect of particle number on rotational errors *(MATLAB Image)*

This comparison is valid for both particle numbers *(20, 100)*. The **plane as point** landmark vector provides corrections to some extent but performs worse than the **plane as oriented point** and **plane as surface** landmark vectors for both particle numbers. For 20 and 100 particles, defining the plane with more parameters gives successful results after the loop-closure.

The **plane as point** landmark vector provides corrections for some of the particles, but the weights are not as high as expected. Therefore, these low weighted particles are eliminated after correction resulting in an increase in the errors. On the other hand, the **plane as surface** landmark vector provides successful corrections. Since many of the particles converge to the true pose, the robot can keep the pose error at a low level and thus there is no significant increase in the errors after the loop-closure.

The **plane as surface** and **plane as oriented point** landmark vectors with 20 particles provide similar successful results in the experiments using 100 particles. This result indicates the success of the particle filter approach. Even when the particle number is 20, the simulations converge to the true pose using the correct landmark vectors *(plane as oriented point or plane as surface).*

Table 6-12 presents the final error values of the **plane as point** and **plane as surface** landmark vectors and gives a comparison of their numerical performance.

Table 6-12 Comparison of the final performance of SLAM experiments

		Final Translational Errors				**Final Rotational Errors**			
		E_{point}	$E_{surface}$	Δ	$\frac{\Delta}{E_{point}}$	E_{point}	$E_{surface}$	Δ	$\frac{\Delta}{E_{point}}$
Particle	**20**	1.9322	1.0090	0.9232	48%	0.8105	0.4981	0.3124	38%
	100	1.6324	0.8472	0.7852	48%	0.6938	0.4155	0.2782	40%

According to Table 6-12, different particle numbers have a similar effect on the correction performance for both rotational and translational errors. So, the correction rate is independent from the particle number. The **plane as surface** landmark vector provides a 48% correction rate for translational errors and almost a 40% correction rate for rotational errors for both particle numbers. However, the number of corrections differs according to the number of particles. The number of particles and corrections are inversely proportional.

6.2.4 The Effect of Effective Particle Rate

This section demonstrates the performance of different landmark vectors for different effective particle rates. The effective particle rates and other simulation parameters are listed in Table 6-13.

Table 6-13 Simulation Parameters

	Experiment 1	**Experiment 2**	**Experiment 3**
Landmark Vector	Plane as Point	Plane as Oriented Point	Plane as Surface
Number of Particles	100	100	100
Effective Particle Rate (%)	20%, 90%	20%, 90%	20%, 90%
Correspondence Threshold	0.00001	0.00001	0.00001
Scanning Step	10	10	10
Linear Velocity Errors (%)	10%	10%	10%
Angular Velocity Errors (%)	10%	10%	10%

There are six scenarios *(3x2)* for three different landmark vectors and two different effective particle rates. Figure 6-15 and Figure 6-16 show the effect of the effective particle rate on the SLAM performance before and after the loop-closure.

The SLAM performance is independent from the effective particle rate before the loop-closure. All the experiments produce similar translational and rotational error rates for different effective particle rates between steps 1-88. In this interval, the **plane as surface** and **plane as oriented point** landmark vectors do not provide any extra contribution to the performance of the SLAM simulation. Also, in step 53, the robot runs the re-sampling process and decreases the number of errors in all experiments.

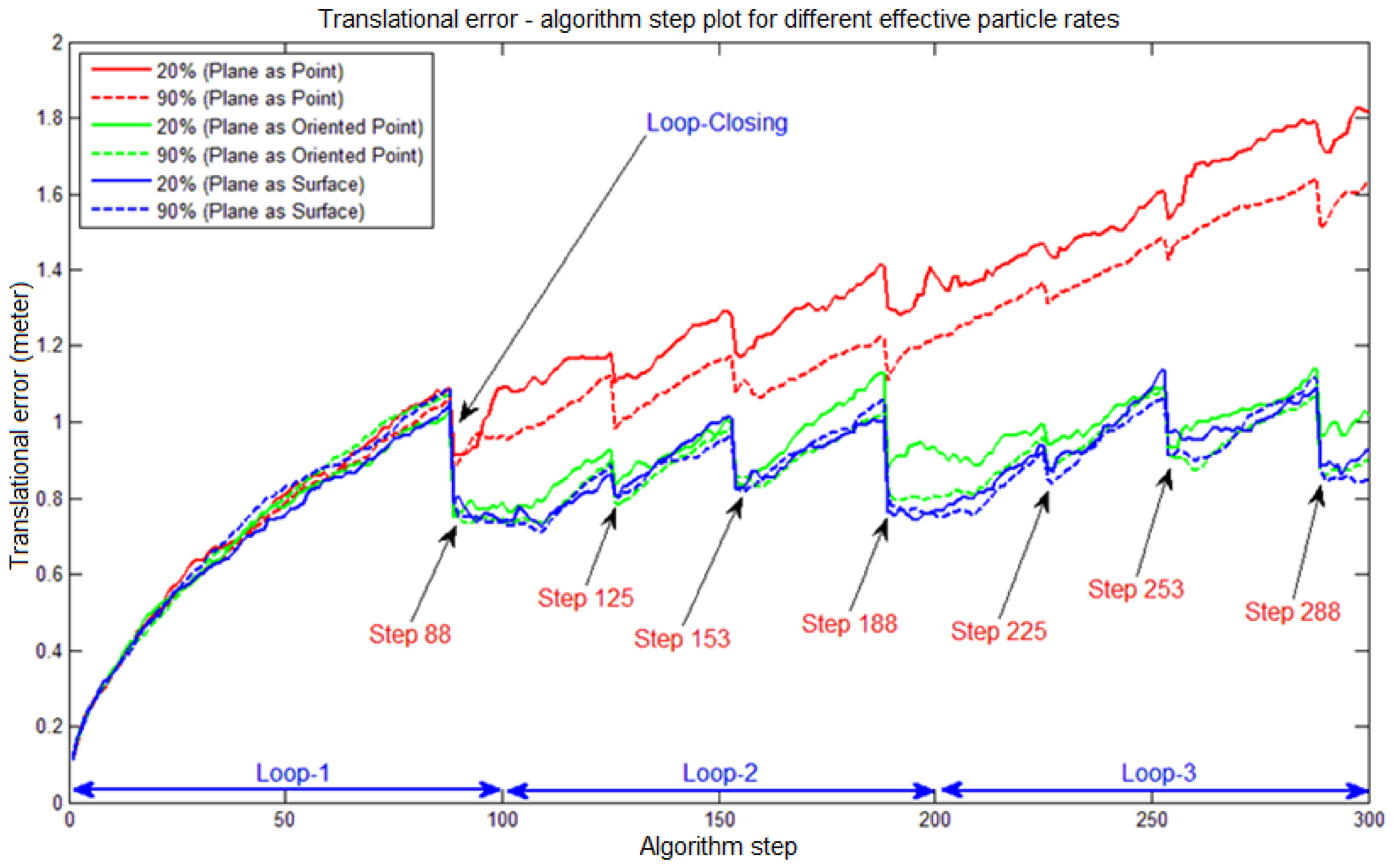


Figure 6-15 The effect of the effective particle rate on translational errors *(MATLAB Image)*.

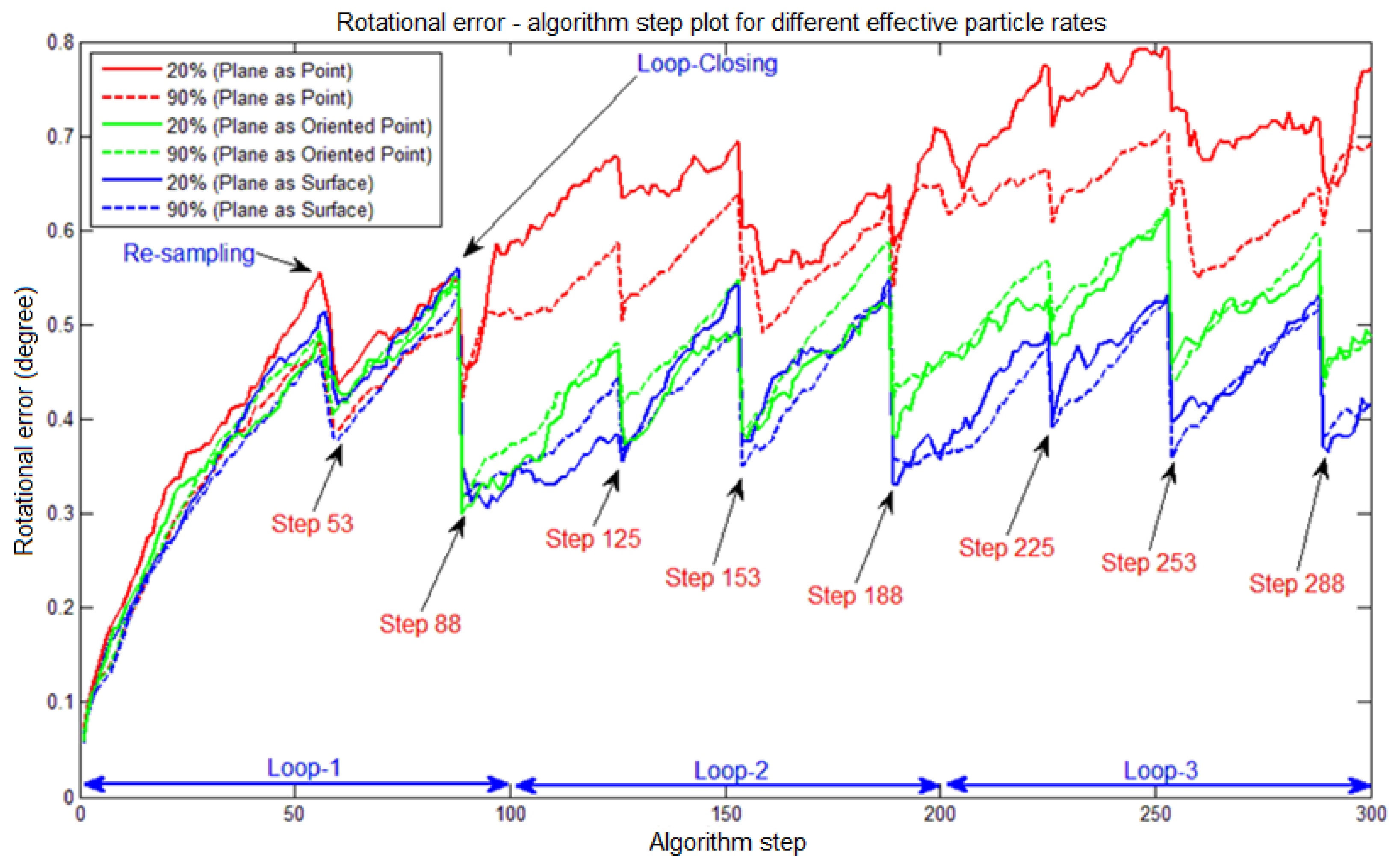


Figure 6-16 The effect of the effective particle rate on rotational errors *(MATLAB Image)*.

The effect of landmark vectors on the SLAM performance is prominent after the loop-closing process. The characteristics of translational and rotational errors are the similar for both effective particle rates. A comparison of the error rates after the loop-closure is given below.

$$\boldsymbol{err_{Plane\ as\ Surface} < err_{Plane\ as\ oriented\ point} < err_{plane\ as\ point}}$$

This comparison is valid for both effective particle rates *(20%, 90%)*. Even though the **plane as point** landmark vector provides corrections to some extent, it performs worse than the **plane as oriented point** and **plane as surface** landmark vectors for both effective particle rates. The **plane as surface** and **plane as oriented point** landmark vectors provide successful corrections for the 20% and 90% effective particle rates after the loop-closure. The robot makes significant number of corrections in steps 88, 125, 153, 188, 225, 253 and 288. According to Figure 6-8, in these steps the effective particle rate decreases to approximately 20% and therefore, the effective particle rate (20% or 90%) does not have any significant effect on the SLAM performance.

The **plane as point** landmark vector provides corrections for some of the particles but the weights are not as high as expected. Therefore, these low weighted particles are eliminated after correction, resulting in an increase in the number of errors again. On the other hand, the **plane as surface** and **plane as oriented point** landmark vectors provide successful corrections. As a result of many of the particles converging to the true pose, the algorithm is able to keep the error rate at a low level and there is no significant increase in errors after the loop-closure.

Table 6-14 presents the final error values of the **plane as point** and **plane as surface** landmark vectors and gives a comparison of their numerical performance.

Table 6-14 A comparison of the final performance of SLAM experiments

		Final Translational Errors				Final Rotational Errors			
		E_{point}	$E_{surface}$	Δ	$\frac{\Delta}{E_{point}}$	E_{point}	$E_{surface}$	Δ	$\frac{\Delta}{E_{point}}$
Eff. Particle Rate	**20%**	1.8177	0.9307	0.8870	48%	0.7729	0.4144	0.3585	46%
	90%	1.6324	0.8472	0.7852	48%	0.6938	0.4155	0.2782	40%

According to Table 6-14, the effective particle rate has a similar effect on the correction rates for both rotational and translational errors. The correction rate for translational errors is independent from the effective particle rate. For both effective particle rates, the **plane as surface** landmark vector corrects 48% of the translational errors and approximately 40% of the rotational errors. However, the number of corrections differs according to the effective particle rate. The effective particle rate and the number of corrections are inversely proportional.

6.2.5 The Effect of Re-sampling

This section demonstrates the effect of re-sampling on the performance of different landmark vectors.

Table 6-15 Simulation Parameters

	Experiment 1	**Experiment 2**	**Experiment 3**
Landmark Vector	Plane as Point	Plane as Oriented Point	Plane as Surface
Number of Particles	100	100	100
Effective Particle Rate (%)	90%	90%	90%
Correspondence Threshold	0.00001	0.00001	0.00001
Scanning Step	10	10	10
Linear Velocity Errors (%)	10%	10%	10%
Angular Velocity Errors (%)	10%	10%	10%

There are a total of six scenarios *(3x2)* for three different landmark vectors. Figure 6-17 and Figure 6-18 show the effect of re-sampling on the SLAM performance before and after the loop closure.

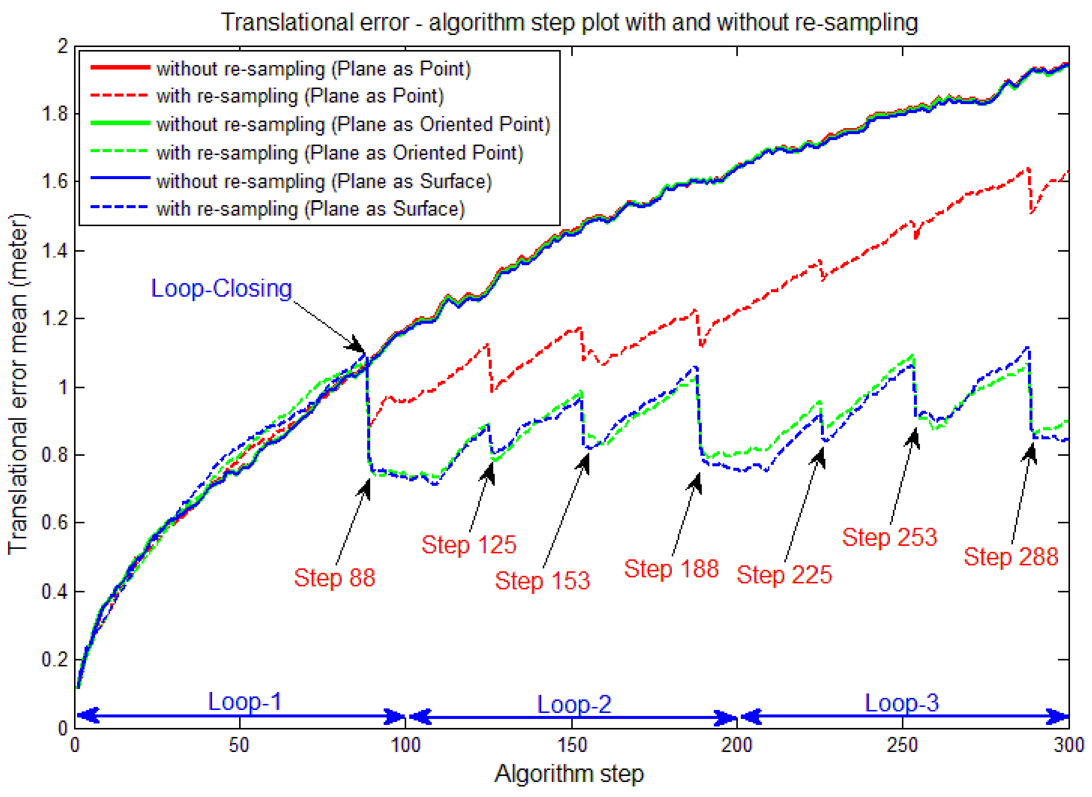


Figure 6-17 The effect of re-sampling on translational errors *(MATLAB Image).*

The **plane as point**, **plane as oriented point** and **plane as surface** landmark vectors do not make any corrections without re-sampling. The re-sampling process replaces the low weighted particles with copies of high weighted particles. Without the re-sampling process, the particle weights and their contribution to the SLAM problem is not meaningful, neither are the correction and data association steps; so the robot moves using only its predictions.

On the other hand, with the re-sampling process, all the landmark vectors provide corrections in steps 53, 88, 125, 153, 188, 225, 253, and 288.

Re-sampling is the most important part of fastSLAM algorithms. The re-sampling step replaces the light-weighted particles with the copies of stronger particles. The robot applies this re-sampling process according to the particle weights. These weights determine the extent to which the particles converge to the obtained measurement.

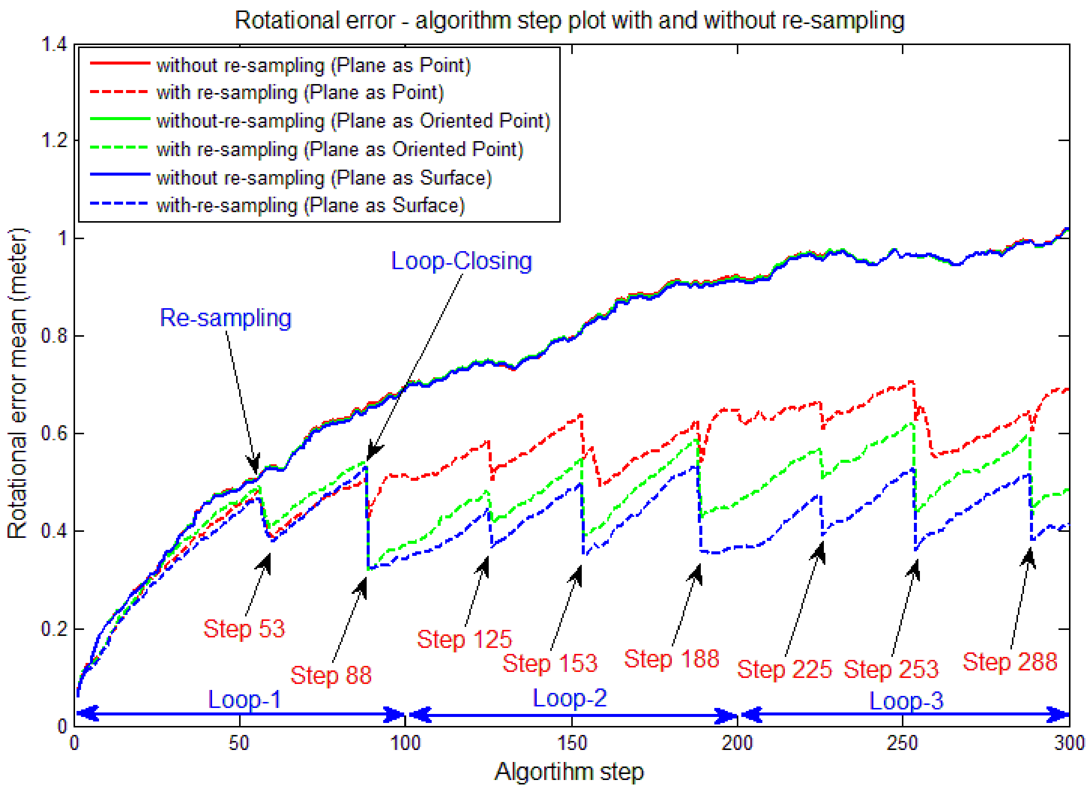


Figure 6-18 The effect of re-sampling on rotational errors *(MATLAB Image)*

Table 6-16 presents the final error values of the **plane as point** and **plane as surface** landmark vectors and gives a comparison of their numerical performance.

Table 6-16 The comparison of the final performance of SLAM experiments

	Final Translational Errors				**Final Rotational Errors**			
	E_{point}	$E_{surface}$	Δ	$\frac{\Delta}{E_{point}}$	E_{point}	$E_{surface}$	Δ	$\frac{\Delta}{E_{point}}$
Without Res.	1.9441	1.9436	0.0005	0.03%	1.0180	1.0192	-0.0012	-0.12%
With Res.	1.6324	0.8472	0.7852	48%	0.6938	0.4155	0.2782	40%

According to Table 6-16, the **plane as surface** and **plane as point** landmark vectors do not provide any corrections without re-sampling. The final errors are independent from the landmark vector. On the other hand, the **plane as surface** landmark vector corrects 48% of the translational errors and 40% of the rotational errors after the re-sampling process.

6.3 Experiment with the fre2_SLAM3 Map

The **fre2_SLAM3** map is more complex than the **fre2_360** map. Figure 6-19 shows the navigation environment for the **fre2_SLAM3** map. There are toys and other background objects in the navigation environment, but the robot filters these objects and detects only the columns and other planar surfaces.

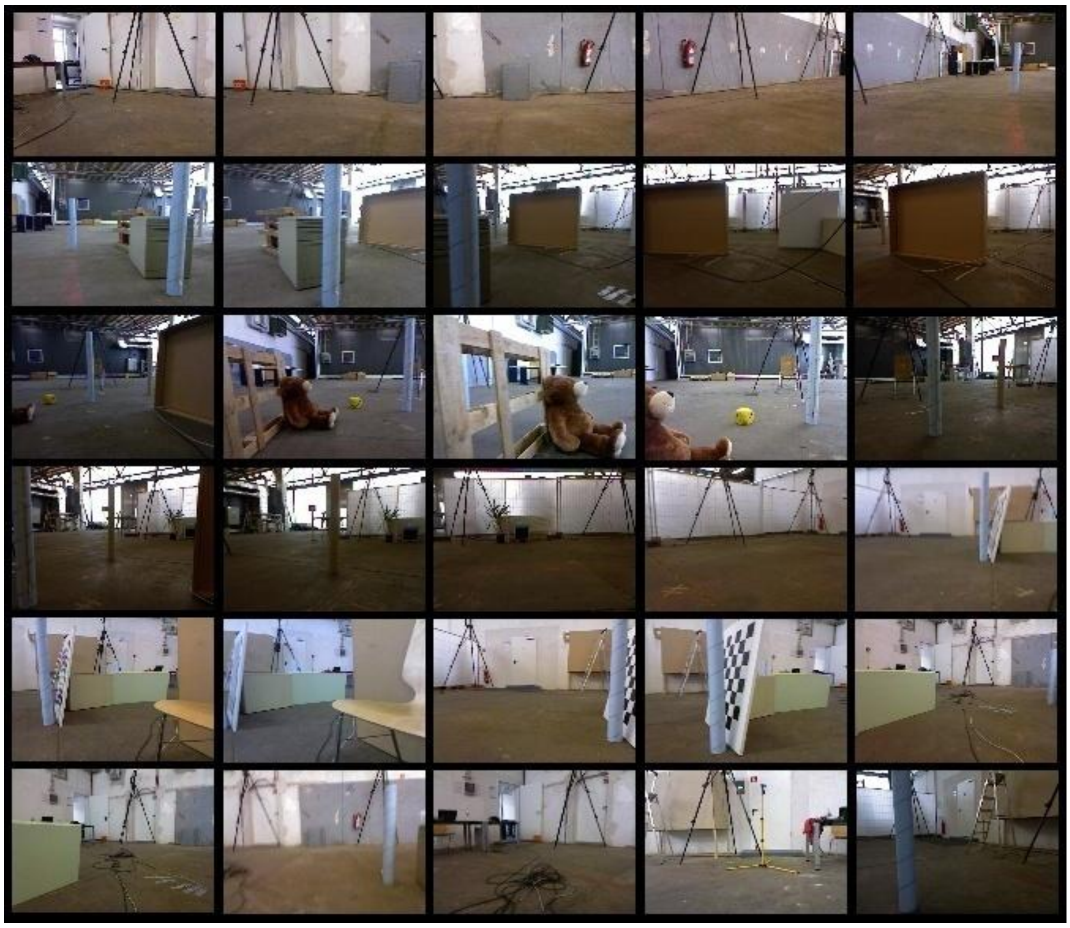


Figure 6-19 The environment for the FRE2_SLAM3 map [6].

The simulation parameters are listed in Table 6-17.

Table 6-17 Simulation parameters for fre2_SLAM3 map

	Experiment 1	**Experiment 2**	**Experiment 3**
Landmark Vector	Plane as Point	Plane as Oriented Point	Plane as Surface
Number of Particles	100	100	100
Effective Particle Rate (%)	90%	90%	90%
Correspondence Threshold	0.00001	0.00001	0.00001
Scanning Step	10	10	10
Linear Velocity Errors (%)	10%	10%	10%
Angular Velocity Errors (%)	10%	10%	10%

The final results of the three-loop navigation are given in Table 6-18.

Table 6-18 SLAM performance results

Landmark Vector	$\boldsymbol{error_{translational}(m)}$	$\boldsymbol{error_{rotational}(radian)}$
Plane as Point	2.4289	0.6129
Plane as Oriented Point	0.8686	0.3075
Plane as Surface	0.7488	0.2403

A comparison of the performance of landmark vectors for translational and rotational errors is given below.

$$\boldsymbol{err_{Plane\ as\ Surface} < err_{Plane\ as\ oriented\ point} < err_{plane\ as\ point}}$$

The results obtained from the **plane as surface** and **plane as oriented point** landmark vectors are similar and both lower than the result from the **plane as point** landmark vector. These results validate the results obtained from the earlier experiments in sections **6.1** and **6.2**. The high dimensional landmark vector, **plane as surface,** provides a better result than the other landmark vectors.

Figure 6-20 and Figure 6-21 present the algorithm steps for different landmark vectors in terms of translational and rotational errors, respectively.

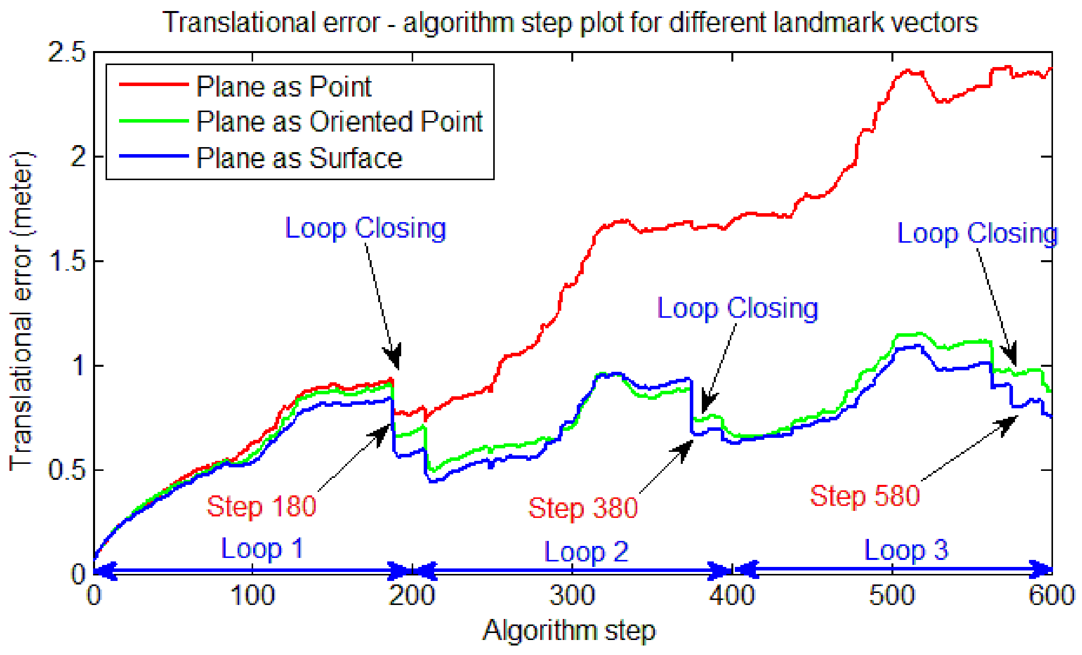


Figure 6-20 Translational error – algorithm step plot for different landmark vectors. *(MATLAB Image).*

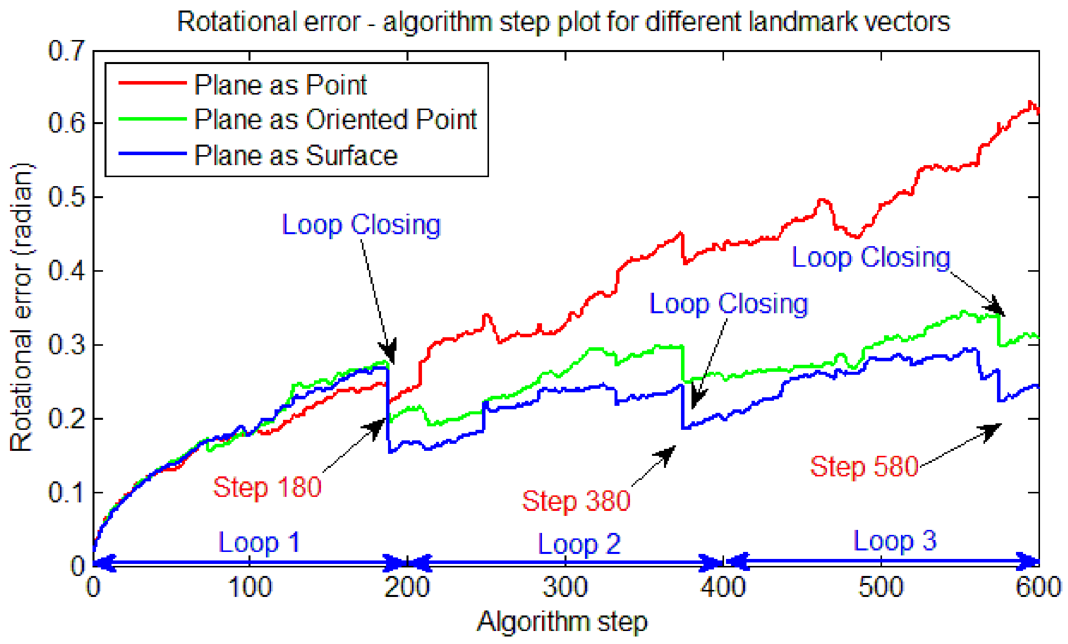


Figure 6-21 Rotational error – algorithm step plot for different landmark vectors. *(MATLAB Image).*

According to Figure 6-20 and Figure 6-21, in step 180, the robot closes the loop and decreases the number of translational and rotational errors for the **plane as surface** and **plane as oriented** point landmarks. The **plane as point** landmark vector provides corrections for translational and rotational errors at the first loop-closing *(in step 180)*. In step 380, corrections are still made for some of the rotational errors. However, the robot can no longer follow the right path after the second loop and thus in step 580, no correction is performed. On the other hand, the robot can follow the right path when using the **plane as oriented point** or **plane as surface** landmark vectors.

Table 6-19 shows the error results for the loop-closing steps.

Table 6-19 Error results for algorithm steps

	Plane as Point		**Plane as Surface**	
	err_{tr}	err_{rot}	err_{tr}	err_{rot}
Step 180	0.9152	0.2468	0.8246	0.2681
Step 380	1.6540	0.4131	0.6622	0.1930
Step 580	2.3818	0.5948	0.7947	0.2316
Step 600	2.4289	0.6129	0.7488	0.2403

Using the **plane as surface** landmark vector, translational errors are kept below 0.8 meters and rotational errors are maintained below 0.25 radian, after the steps 180, 380 and 580. On the other hand, the **plane as point** landmark vector does not provide any significant corrections in the loop closing steps. Both translational and rotational errors increase with motion. In the final step, translational and rotational errors reach 2.5 meters and 0.6 radian, respectively.

Table 6-20 shows a comparison of the performance of the **plane as surface** and **plane as point** landmark vectors.

Table 6-20 The comparison of the final performance of SLAM experiments

Final Translational Errors				Final Rotational Errors			
E_{point}	$E_{surface}$	Δ	$\frac{\Delta}{E_{point}}$	E_{point}	$E_{surface}$	Δ	$\frac{\Delta}{E_{point}}$
2.4289	0.7488	1.6801	69%	0.6129	0.2403	0.3726	60%

The **plane as surface** landmark vector corrects 69% more translational errors and 60% more rotational errors than the **plane as point** landmark vector. These correction rates clearly show the contribution of the **plane as surface** landmark vector to the SLAM performance.

6.4 Experiment with the Pioneer Robot

This section presents the comparison of different landmark vectors using a real-time application. In this part of the research, the **plane as point**, **plane as oriented point** and **plane as surface** landmark vectors are investigated. The navigation environment is the METU Computer Vision Laboratory and a pioneer 2 robot with a Kinect sensor is used.

Figure 6-22 shows the pioneer robot, navigation environment, and the marked ground, which provides the ground truth measurements at interval steps.

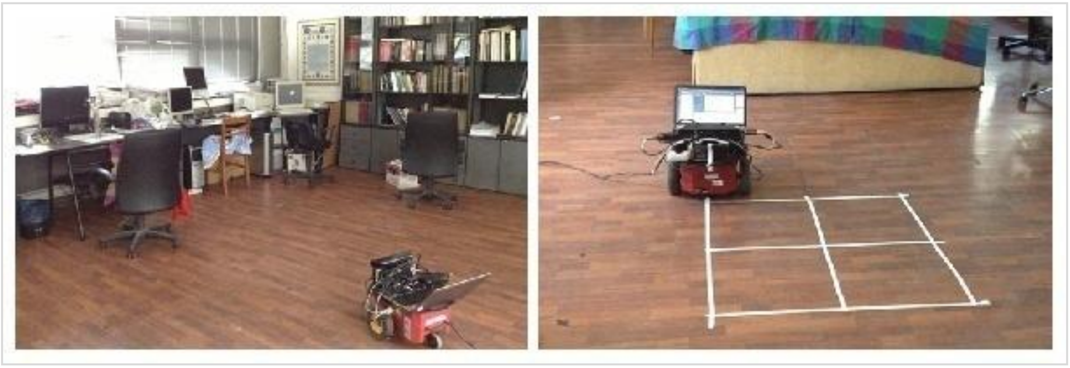


Figure 6-22 The map environment and the robot *(Images were taken in the METU Lab.)*

The navigation environment is noisy and crowded but the filtering process eliminates the unnecessary objects such as books and tables. After the filtering process, the only remaining points are the backs of the chairs. The navigation environment is shown in Figure 6-22 and Figure 6-23.

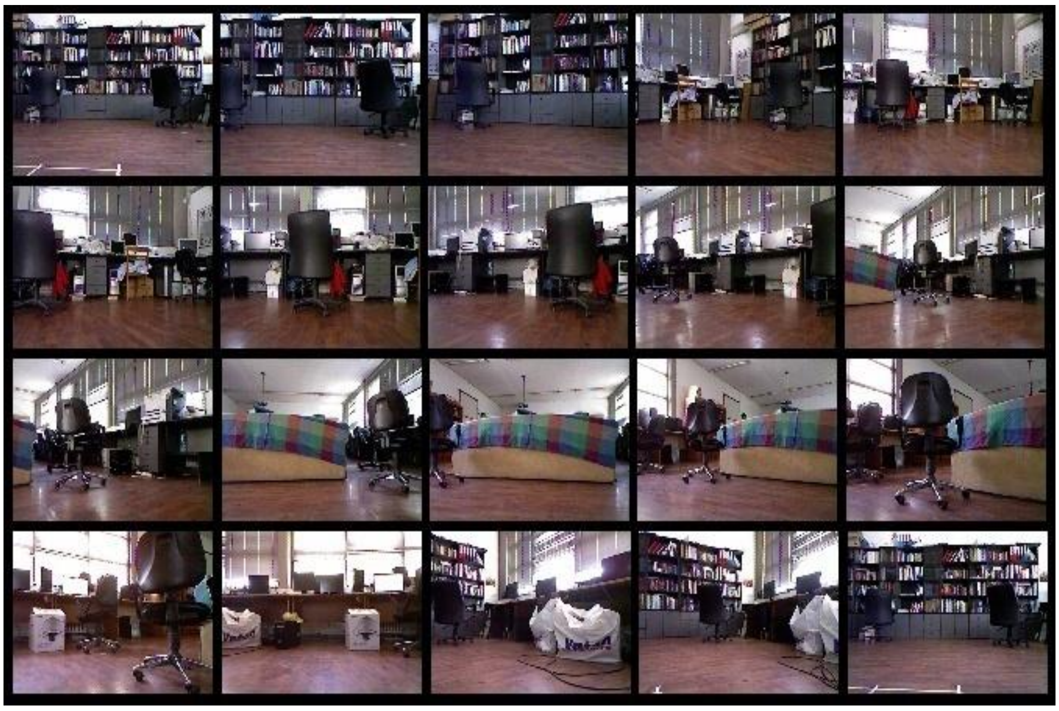


Figure 6-23 Screenshots from the RGB camera of the Kinect sensor during navigation *(Images were taken in the METU Lab.)*

The original depth and RGB image for the first scan are shown in Figure 6-24. The target objects are outlined in red.

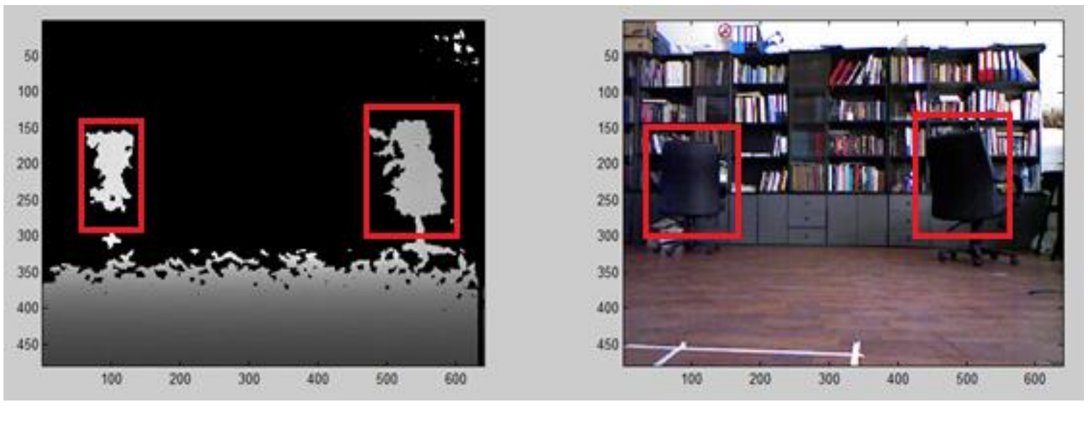


Figure 6-24 Sample depth and RGB image from a real-time application *(Taken in the METU Lab.)*

Figure 6-25 shows the extracted feature points after the filtering and clustering process *(back of the chair)* in green.

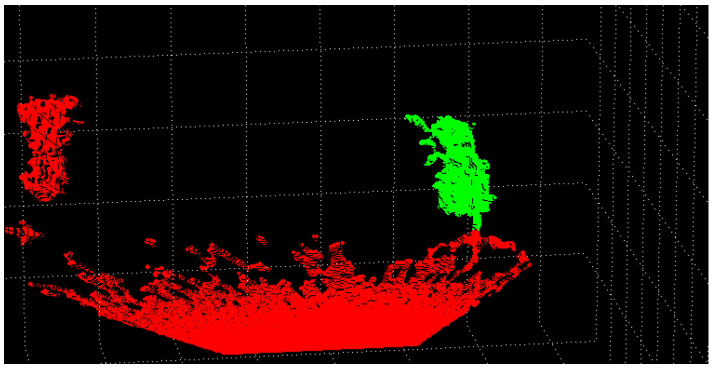


Figure 6-25 Sample 3D point cloud from a real time application *(MATLAB Image).*

The plane fit algorithms are applied to the green points, and SLAM algorithms are performed using these extracted features. Figure 6-26 shows the visual result of one-turn navigation.

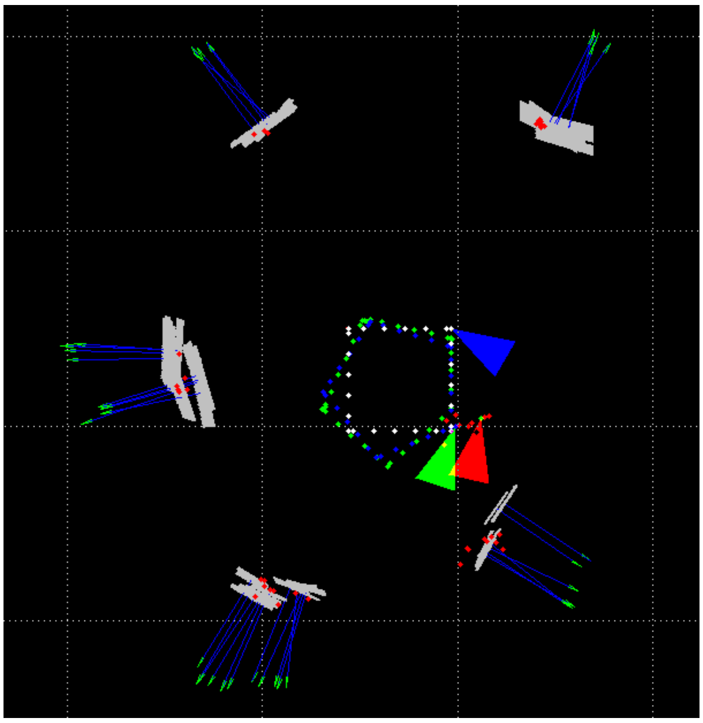


Figure 6-26 Final view of the vehicle and the detected landmarks *(MATLAB Image).*

The white dots indicate the determined path with the blue dots showing the ground truth and the green dots defining the found path *(mean of a hundred particles)*. The blue triangle shows the position after the commanded motion, the green triangle shows the real position of the vehicle after motion, and the red triangle indicates the final position of the particle with the maximum weight as found by the SLAM algorithm. A comparison of the results for translational and rotational errors is given in Figure 6-27 and Figure 6-28, respectively.

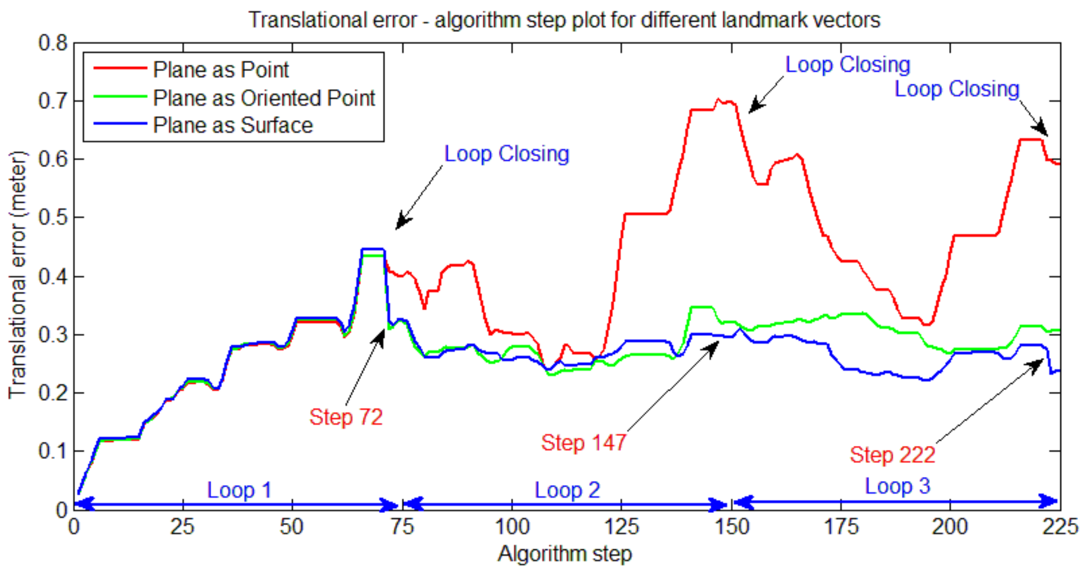


Figure 6-27 Comparison of the translational error rates in the recorded path of a real time application *(MATLAB Image).*

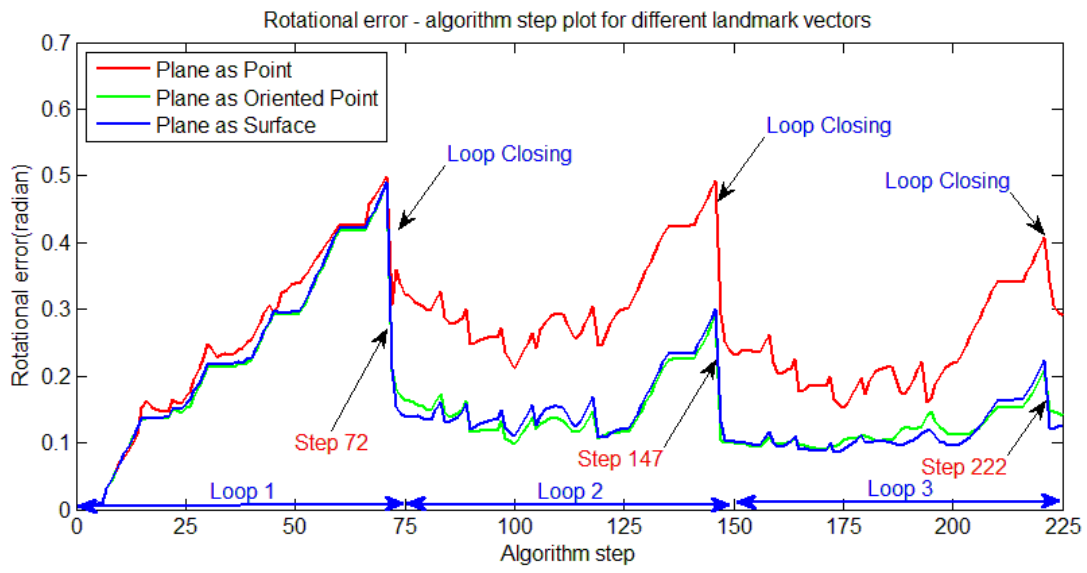


Figure 6-28 Comparison of the rational error rates in the recorded path of a real time application *(MATLAB Image).*

The robot completes one turn in 75 algorithm steps and detects 5 different landmarks throughout the navigation. The first landmark is detected in step 72 for the second time and the loop is closed. 147 and 222 are the other loop-closing steps. The robot also closes the loops 8 times in other interval steps *(such as 81, 90)* between 72 and 225. In these loop-closing steps, there is no significant error correction due to the low rates of translational and rotational errors. In the majority of the navigation, the robot runs the prediction and correction sequences together. There is not much space between the objects, and after the loop-closure, errors do not increase as a result of tracking the landmarks that were detected earlier. In the experiments using the **fre2_360** map there is much more space between the landmarks and therefore, the errors increase due to the lack of the correction sequence. As a result, a significant error correction was observed at the interval loop-closing steps *(125, 153, 225, and 253).*

In the real time application used in this study, the robot always finds the right path in all experiments involving different landmark vectors. All landmark vectors provide corrections in loop-closing steps, but the **plane as surface** landmark vector performs better due to its high correspondence performance.

Table 6-21 shows the final error results for the SLAM application performed in the computer vision laboratory.

Table 6-21 The final error results in the navigation of the Pioneer 2 robot.

Landmark Vector	$\boldsymbol{error_{translational}(m)}$	$\boldsymbol{error_{rotational}(radian)}$
Plane as Point	0.5899	0.2899
Plane as Oriented Point	0.3083	0.1410
Plane as Surface	0.2394	0.1250

A comparison of the performance of landmark vectors for translational and rotational errors is given below.

$$\boldsymbol{err_{Plane\ as\ Surface} < err_{Plane\ as\ oriented\ point} < err_{plane\ as\ point}}$$

The **plane as surface** and **plane as oriented point** landmark vectors provide similar results that were both lower than the result obtained from the **plane as point** landmark vector. These results validate the results obtained from the earlier experiments in sections **6.1**, **6.2** and **6.3**. The high dimensional landmark vector, **plane as surface,** performs better than the other landmark vectors.

Table 6-22 presents the final error values of the **plane as point** and **plane as surface** landmark vectors and gives a comparison of their performance in terms of translational and rotational errors.

Table 6-22 A comparison of the final performance of experiments.

Final Translational Errors				**Final Rotational Errors**			
$\boldsymbol{E_{point}}$	$\boldsymbol{E_{surface}}$	Δ	$\frac{\Delta}{\boldsymbol{E_{point}}}$	$\boldsymbol{E_{point}}$	$\boldsymbol{E_{surface}}$	Δ	$\frac{\Delta}{\boldsymbol{E_{point}}}$
0.5899	0.2394	0.3505	60%	0.2899	0.1250	0.1649	56%

According to the results, the real time application validates the thesis in this research. Using more compact landmark vectors results in a better correction performance compared with the other landmarks. The correction rate obtained from the **plane as surface** landmark vector is 60% higher for translational errors and 56% higher for rotational errors when compared with the results of the **plane as point** landmark vector. These correction rates clearly show the contribution of the **plane as surface** landmark vector to the SLAM performance. Defining the surrounding objects with surface properties provides a better correspondence and increases the SLAM performance.

CHAPTER 7

CONCLUSION

7.1 Summary and Conclusion

This thesis presents the contribution of planar features to the fastSLAM algorithm for indoor environments which were determined using two different feature detection methods *(SURF feature detection and plane feature detection)* and four different landmark vectors *(SURF points, plane as point, plane as oriented point and plane as surface).*

An accurate algorithm of SLAM is implemented by utilizing a particle filter. The performance of the SLAM is analyzed for different landmark vectors and different map environments.

The SURF feature detection algorithm gives a high number of responses at the edge of the depth images. A SLAM application with this high number of SURF features is more successful than the other applications. On the other hand, this high number of SURF features results in very high time consumption and makes SURF features useless for real time indoor applications.

Also, the map environment and navigation have an important effect on the performance of the SLAM algorithm. According to all the results, it was found that there are two different phases in simulations.

- Before the loop-closing
- After the loop-closing.

Before the loop closing, the approach proposed in this study does not guarantee a decrease in the number of errors due to the algorithmic errors injected to the system through the detection of the planar features. However, after the loop closing the proposed landmark vectors satisfy significant error correction and the error rate decreases compared with the number of errors that occur in the other method. Planar features may inject an error to the simulation because of the structure of plane detection algorithms, non-planar surfaces and sensor measurement errors. Planar features provide successful correction, despite these algorithmic errors.

Also, the proposed method tested under the effect of different simulation parameters and results supported our thesis. These simulation parameters are linear velocity error rate, angular velocity error rate, particle number and effective particle rate. These simulations were undertaken with the SLAM_360 map and comments on the results are given with their graphical representation.

Chapter 6 gives detailed information that shows that the proposed method is capable of better localization and mapping in shorter time as long as planar features are detected correctly.

7.2 Future Works

The important idea behind the proposed method is to define the 3D environment with compact landmark vectors. Therefore, new features, such as width and height information, can be added to the planar landmark vectors.

In the filtering step, the defined percentage of the data selected randomly, to speed up the algorithm. Randomly selection may result with the important data loss. The probabilistic SLAM approach can handle this situation generally however, an intelligent selection method may improve the performance of proposed method.

In the clustering step, the object size is defined according to the map environment and the Kinect camera resolution. For different map and sensor camera the simulations carried out in this study may fail. If the camera resolution is higher than the Kinect camera this may result in defining an object as a background. The algorithm should select the object size dynamically to handle this kind of problem and for generalized solution to every environment and camera type.

Throughout the research, the algorithm uses Kinect Camera data and because of the sensor lacking certain capabilities the proposed method is validated only for indoor environments. Therefore, the proposed ideas can be validated using laser scanner in outdoor environments.

Also, in the method proposed in this thesis planar surfaces were detected and the non-planar ones were neglected according to the standard deviation and the fitting performance criteria. The proposed method may be unsatisfactory if the environment is constituted by non-planar surfaces. Therefore, instead of defining surrounding objects according to their planar features, defining the curvature properties or objects themselves may give better results.

REFERENCES

[1] Sebastian Thrun, Wolfram Burgard, and Dieter Fox, *Probabilistic Robotics*. Cambridge, Massachusetts: The MIT Press, 2005.

[2] S., Beetz, M., Bennewitz, M., Burgard, W., Cremers, A. B., Dellaert, F., Fox, D., Haehnel, D., Rosenberg, C., Roy, N., Schulte, J., Schulz, D. Thrun, "Probabilistic algorithms and the interactive museum tour-guide robot Minerva," in *Journal of Robotics Research*, 2000.

[3] Sebastian Thrun et al., "Stanley: The Robot that Won the DARPA Grand Challenge," in *Journal of Field Robotics*, 2006.

[4] Andreas Nüchter, *3D Robotic Mapping The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom*.: Springer, 2009.

[5] N.J. Chen and J.S. Chen, "3D Scenes Registration using a 2D Laser Range Finder," *IEEE International Conference on Automation and Logistics*, pp. 457-463, August 2010.

[6] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A Benchmark for the Evaluation of RGB-D SLAM Systems," *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, October 2012.

[7] Paul J. Besl and Neil D. McKay, "A Method for Registration of 3-D Shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 239-256, February 1992.

[8] Peter Biber, "The Normal Distributions Transform: A New Approach to Laser Scan Matching ," in *Intl. Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, 2003, pp. 2743-2748.

[9] Cihan Ulaş and Hakan Temeltaş, "A 3D Scan Matching Method Based On Multi-Layered Normal Distribution Transform," in *International Federation of Automatic Control (IFAC)*, Milano, 2011, pp. 11602-11607.

[10] Renato F. Salas-Moreno, Richard A. Newcombe, Hauke Strasdat, Paul H. J.

Kelly, and Andrew J. Davison, "SLAM++: Simultaneous Localisation and Mapping at the Level of Objects," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[11] R Smith, M Self, and P Cheeseman, "Estimating uncertain spatial relationships in robotics," *Springer-Verlag*, pp. 167-193, 1990.

[12] J Guivant and E Nebot, "Optimization of the simultaneous localization and map-building algorithm for real time implementation," *IEEE Transactions on Robotics and Automation*, pp. 242-257, 2001.

[13] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit, "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem," *AAAI*, 2002.

[14] Masahiro Tanaka and Minoru Ito, "Experimental Results for Walking Navigation System Using FastSLAM," in *International Conference on Control, Automation and Systems*, Seoul, Korea, 2007.

[15] Georg Arbeiter, Jan Fischer, and Alexander Verl, "3D Environment Reconstruction for Mobile Robots using fast-SLAM and Feature Extraction," in *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)* , 2010.

[16] Yangming Li and Edwin B. Olson, "Extracting General Purpose Fetures from LIDAR data," *ICRA*, pp. 1388-1393, 2010.

[17] Jan Weingarten, *Feature Based 3D SLAM*. Lausanne, 2006.

[18] Tobias Hedlund, *Registration of multiple ToF camera point clouds*. Sweden, 2010.

[19] Michael Ying Yang and Wolfgang Förstner, "Plane Detection in Point Cloud Data," 2010.

[20] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz, "Towards 3D Point cloud based object maps for household environments," *Robotics and Autonomous Systems*, pp. 927-941, August 2008.

[21] Çağrı Turunç, An implementation of 3D SLAM with Planar Segments, 2012.

[22] Felix Endres, Jürgen Hess, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard, "3-D Mapping With an RGB-D Camera," *IEEE TRANSACTIONS ON ROBOTICS*, 2013.

[23] N Ganganath and H Leung, "Mobile Robot Localization using odometry and Kinect Sensor," *IEEE*, pp. 91-94, 2012.

[24] Jan Hartmann, Dariush Forouher, Marek Litza, Jan Helge Klüssendorff, and Erik Maehle, "Real-Time Visual SLAM Using FastSLAM and the Microsoft Kinect Camera," in *Proceedings of Robotik*, 2012.

[25] Yuichi Taguchi, Yong-Dian Jian, Srikumar Ramalingam, and Chen Feng, "SLAM Using Both Points and Planes for Hand-Held 3D Sensors," in *IEEE International Symposium on Mixed and Augmented Reality*, Atlanta, Georgia, 2012.

[26] Yuichi Taguchi, Yong-Dian Jian, Srikumar Ramalingam, and Chen Feng, "Point-Plane SLAM for Hand-Held 3D Sensors," in *International Conference on Robotics and Automation*, Karlsruhe,Germany, 2013.

[27] H. Cheong, S. Park, and S.- K. Park, "Topological Map Building and Exploration Based on Concave Nodes," *In Proc. of the Int. Conf. on Control, Automation and Systems*, pp. 1115-1120, 2008.

[28] A. Kalay, An Implementation of Mono And Stereo Slam System Utilizing Efficient Map Management Strategy, 2008, Ms. Thesis.

[29] Tim Bailey, Mobile Robot Localisation and Mapping in Extensive Outdoor Environments, 2002.

[30] A. Elfes, *Using Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. Pensylvania: Carnegie Mellon University, 1989.

[31] Greg Welch and Gary Bishop, *An Introduction to the Kalman Filter*., 2006.

[32] Sebastian Zug, Felix Penzlin, Andre Dietrich, Tran Tuan Nguyen, and Sven Albert, "Are laser scanners replaceable by Kinect sensors in robotic applications?," in *Robotic and Sensor Environments*, 2012.

[33] (2014, Oct.) Dallmeier Electronic GmbH & Co.KG. [Online]. http://www.youtube.com/watch?v=-DoTgQbALU0

[34] (2014, Oct.) RGB-D SLAM Dataset. [Online]. http://vision.in.tum.de/data/datasets/rgbd-dataset.

[35] (2014, Oct.) Raptor-E Digital RealTime System. [Online]. http://www.motionanalysis.com/html/industrial/raptore.html

[36] John Folkesson and Henrik Christensen, "Graphical SLAM - A Self Correcting Map," *ICRA*, pp. 383-390, 2004.

[37] Cihan Ulaş and Hakan Temeltaş, "A Fast and Robust Scan Matching Algorithm Based on ML-NDT and Feature Extraction," in *International Conference on mechatronics and Automation*, Beijing, 2011, pp. 1751-1756.

[38] Richard O. Duda, Peter O. Hart, and David G. Stork, *Pattern Classification*., 2000.

[39] Rafael C. Gonzales and Richard E. Woods, *Digital Image Processing*, 3rd ed.: Peorsan Prentice Hall, 2008.

[40] Matthieu Molinier, Tuomas Hame, and Heikki Ahola, "3D Connected Component Analysis for Traffic monitoring in Image Sequences Acquired from a Helicopter," in *SCIA*, Berlin, 2005, pp. 141-150.

[41] Tim Zaman. (2014, Oct.) Tim Zaman Project Reference. [Online]. www.timzaman.nl

[42] F. Tarsha Kurdi, T. Landes, and P. Grussenmeyer, "Hough-Transform and Extended Ransac Algorithm for Automatic Detection of 3D Building Roof Planes From Lidar Data," *IAPRS*, pp. 407-412, September 2007.

[43] Xiangfei Qian and Cang Ye, "NCC-RANSAC: A Fast Plane Extraction Method for 3-D Range Data Segmentation," in *IEEE Transactions on Cybernetics*, 2014.

[44] O. Gallo, R. Manduchi, and A. Rafii, ""CC-RRANSAC: Fitting planes in the presence of multiple surfaces in range data," in *ELSEVIER*, 2010, pp. 403-410.

[45] Herbert Bay, Andreas Ess, Tinne Tuytalears, and Luc Van Gool, "SURF: Speeded Up Robust Features," *Computer Vision and Image Understanding (CVIU)*, vol. 110, pp. 346-359, 2008.

[46] Duy Nguyen Ta, Wei Chao Chen, Natasha Genfald, and Kari Pulli, "SURFTrac: Efficient Tracking and Continuous Object Recognition using Local Feature Descriptors," *CVPR*, pp. 2937-3944, 2009.

[47] Tim Bailey. (2014, Oct.) Open SLAM. [Online]. https://openslam.org/bailey-slam.html

[48] Johnathan Mun, *Modeling Risk : Applying Monte Carlo Simulation, Real Options Analysis, Forecasting, and Optimization Techniques (2nd Edition)*. Hoboken, NJ, USA: Wiley, 2010.

APPENDIX A

MONTE CARLO ANALYSIS

The Monte Carlo simulation in its simplest form is a random number generator that is useful for estimation. The algorithm selects random values from user-predefined probability distribution and simulates the model [48].

Scientists, engineers, statisticians, business analysts, and others use computers to create the models of systems *(any system)* and to simulate reality by making predictions. These complex computations became possible with fast computers. These simulations account for randomness and future uncertainties through hundreds and even thousands of different scenarios *(simulations)*. The Monte Carlo Analysis is compiling all these simulations' results and making decision about the behavior of system [48].
In this thesis different SLAM methods are compared according to the Monte Carlo analysis.

The behaviors of algorithms are compared according to the changing values of linear velocity and angular velocity. In all simulations the linear velocity and angular velocity sampled from ground truth data.

In the current study the experiments are compared under different parameter conditions and the results are evaluated.

- Linear Velocity Error Rate *(3%, 7%, 15%)*
- Angular Velocity Error Rate *(3%, 7%, 15%)*
- Particle Number *(20, 100)*
- Effective Particle Rate *(20%, 90%)*
- Re-sampling *(With Re-sampling , Without Re-sampling)*

APPENDIX B

INTEGRAL IMAGE

The motivation behind creating an integral image is its simplicity in calculating the sum of all intensity values inside a rectangular region in the original image. This simplicity allows the fast computation of box filter convolutions.

The integral image I_{Σ} of an image I is defined below.

$$I_{\Sigma}(x, y) = \sum_{i=0}^{x} \sum_{j=0}^{y} I(i, j) \tag{B.1}$$

In other words, the intensity value at any location x, y in the integral image $I_{\Sigma}(x, y)$ is the sum of all intensity values of all pixels inside the rectangular region with the top left corner $(0,0)$ and bottom right corner (x, y) on the original image $I(x, y)$.

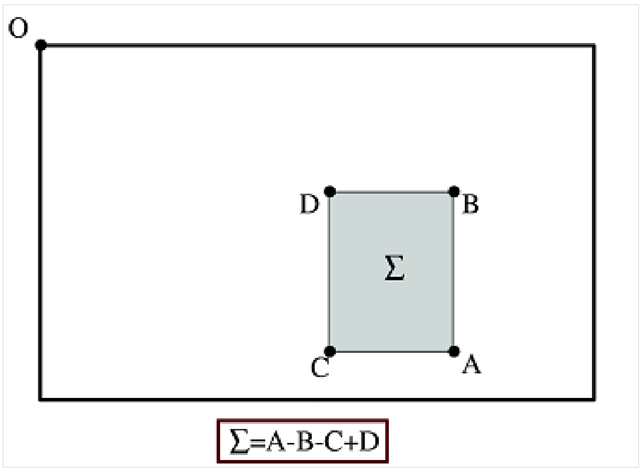


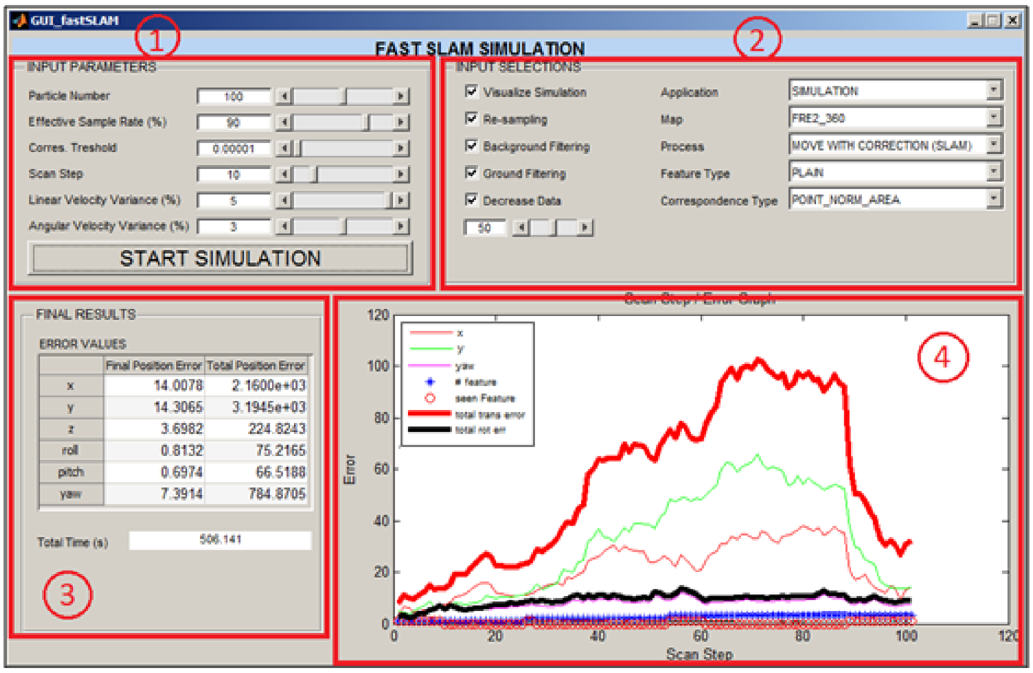
Figure B-1 Integral Image [45].

Calculating the sum of intensities of any rectangular area in original image takes only 3 additions.

APPENDIX C

SIMULATION INTERFACE (GUI)

A Graphical User Interface (GUI)was used to generate the SLAM simulation or control the robot in real time.



FAST SLAM SIMULATION

INPUT PARAMETERS

Particle Number	100
Effective Sample Rate (%)	90
Corres. Treshold	0.00001
Scan Step	10
Linear Velocity Variance (%)	5
Angular Velocity Variance (%)	3

INPUT SELECTIONS

✓	Visualize Simulation	Application	SIMULATION
✓	Re-sampling	Map	FRE2_360
✓	Background Filtering	Process	MOVE WITH CORRECTION (SLAM)
✓	Ground Filtering	Feature Type	PLAN
✓	Decrease Data	Correspondence Type	POINT_NORM_AREA
	50		

FINAL RESULTS

ERROR VALUES

	Final Position Error	Total Position Error
x	14.0078	2.1600e+03
y	14.3065	3.1945e+03
z	3.6982	224.8243
roll	0.8132	75.2165
pitch	0.6974	66.5188
yaw	7.3914	784.8705

Total Time (s) 506.141

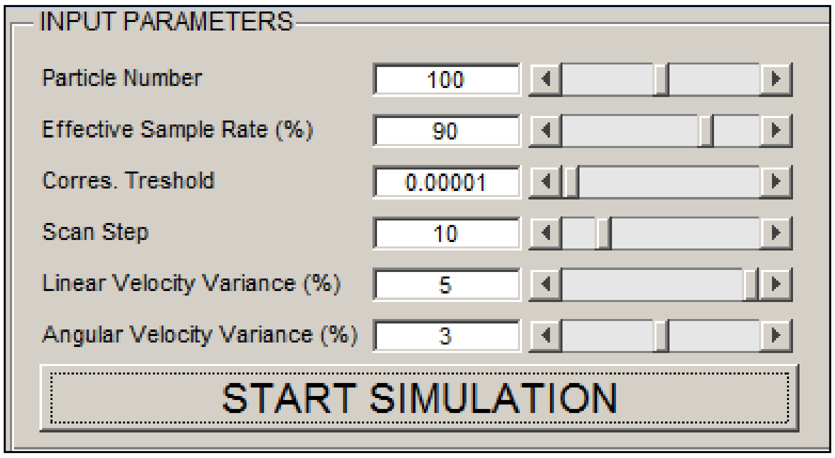
Figure C-1 GUI *(MATLAB Image).*

The aim of the GUI is to evaluate the performance of SLAM for one turn according to given input parameters. In Figure C-1, there are 4 main parts in the GUI. Parts 1 and 2 are concerned with input parameters and parts 3 and 4 are about the final results.

In part 1 there are simulation parameters which, together with their default values are listed below.

- FastSLAM sample *(particle)* number: ***Default is 100 particles.***
- Effective particle rate: ***Default is 90%.***

- Correspondence threshold: ***Default is 0.00001.***
- Scan step: ***Default is 10.***
- Linear velocity variance: ***Default is 5%.***
- Angular velocity variance: ***Default is 3%.***



INPUT PARAMETERS

Particle Number	100
Effective Sample Rate (%)	90
Corres. Treshold	0.00001
Scan Step	10
Linear Velocity Variance (%)	5
Angular Velocity Variance (%)	3

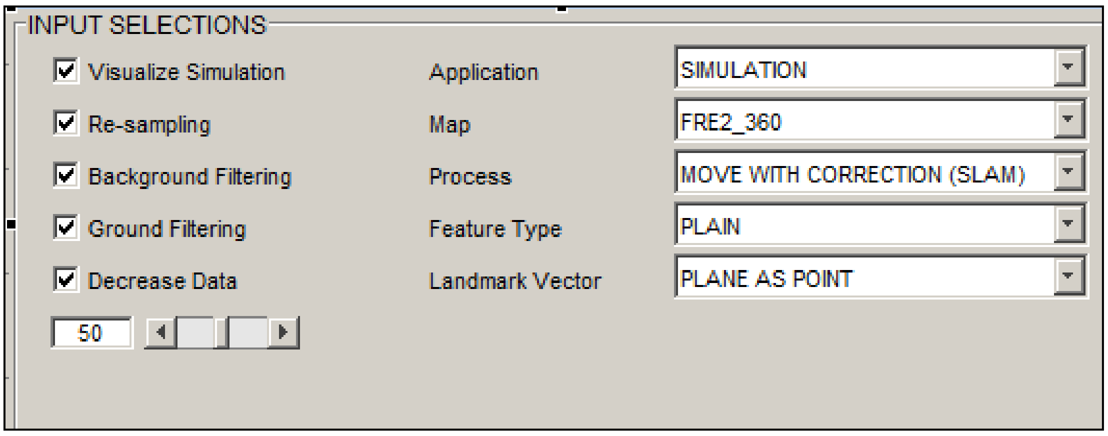
Figure C-2 Input parameters – Part 1 *(MATLAB Image).*

In part 2 there are selections regarding the simulation. Some are indispensable for the simulation. On the other hand, for debugging purposes a user could deselect or change the default values.

- Show Simulation Step by Step : ***Yes***
- Re-sampling : ***Yes***
- Background Filtering: ***Yes***
- Ground Filtering: ***Yes***
- Decrease Data: ***Yes (Default is 50%)***
- Application *(**Simulation**/Real Time)*
- Map : *(**FRE2_360** / FRE2_SLAM1/ FRE2_SLAM2 / FRE2_SLAM3)*
- Process: *(**Move with Correction (SLAM)** / Move with Prediction (Debug) / Move with Odometer(Debug))*
- Feature Type: *(**PLANE** / SURF)*

- Landmark Vector: (***Plane as Point*** */ Plane as Oriented Point / Plane as Surface / SURF Point)*

These parameters and selections are detailed in Appendix D.

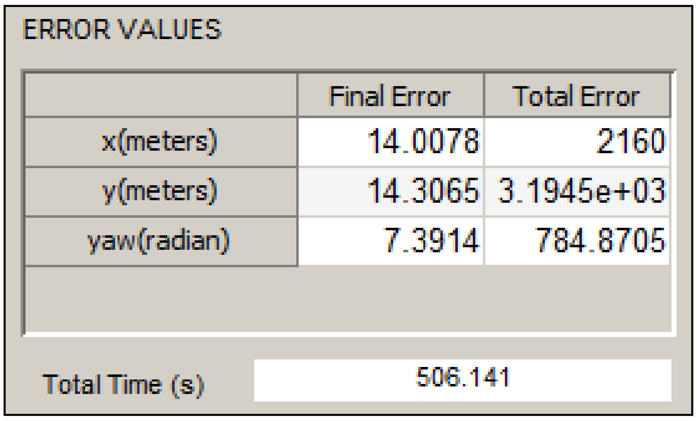

INPUT SELECTIONS

✓ Visualize Simulation	Application	SIMULATION
✓ Re-sampling	Map	FRE2_360
✓ Background Filtering	Process	MOVE WITH CORRECTION (SLAM)
✓ Ground Filtering	Feature Type	PLAIN
✓ Decrease Data	Landmark Vector	PLANE AS POINT
50		

Figure C-3 Input selection – Part 2 *(MATLAB Image).*

In part 3 there are simulation results. These results are concerned with the errors and time consumption.

- Final Position Error *(For x, y and yaw): Error at the end of the simulation.*
- Total Position Error *(For x, y and yaw): Cumulative error for every scan.*
- Total Time: Total time consumption for simulation.


ERROR VALUES

	Final Error	Total Error
x(meters)	14.0078	2160
y(meters)	14.3065	3.1945e+03
yaw(radian)	7.3914	784.8705

Total Time (s) 506.141

Figure C-4 Simulation error and time results – Part 3 *(MATLAB Image).*

In part 4 there a graphical view of simulation errors according to time *(scan step)is given*. In graph there are error values for x, y, yaw, total rotational and total translational according to time. The total number of features and current states *(seeing feature or not)* also can be seen in the graph for evaluation purposes.

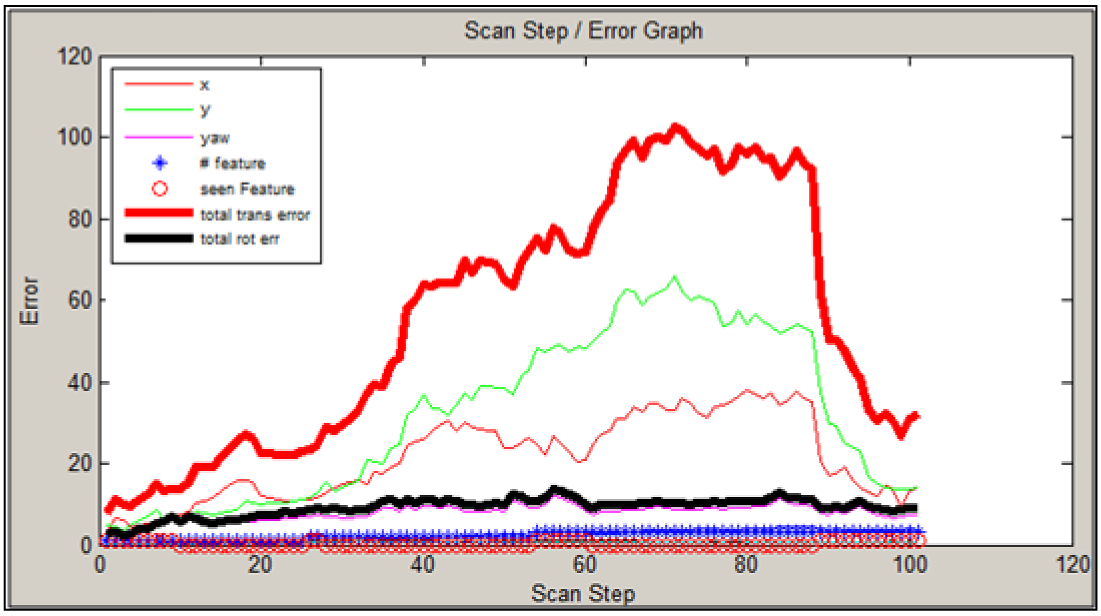


Figure C-5 Graphical View of Simulation Error with Time *(MATLAB Image).*

If the user selects the **"Show Simulation Step by Step" the** simulation steps are displayed in another screen. In figures C-6 and C-7, the found plane features, normal vectors, ground truth trajectory and predicted trajectory are shown.

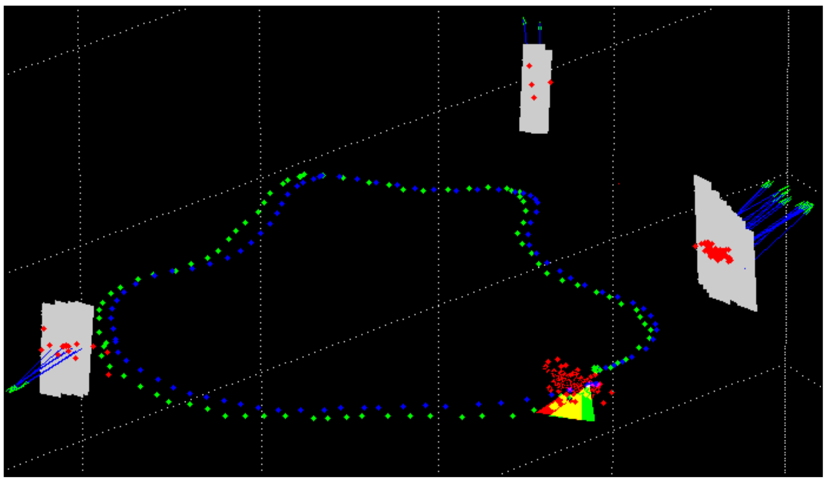


Figure C-6 Screenshot from the simulation environment *(MATLAB image).*

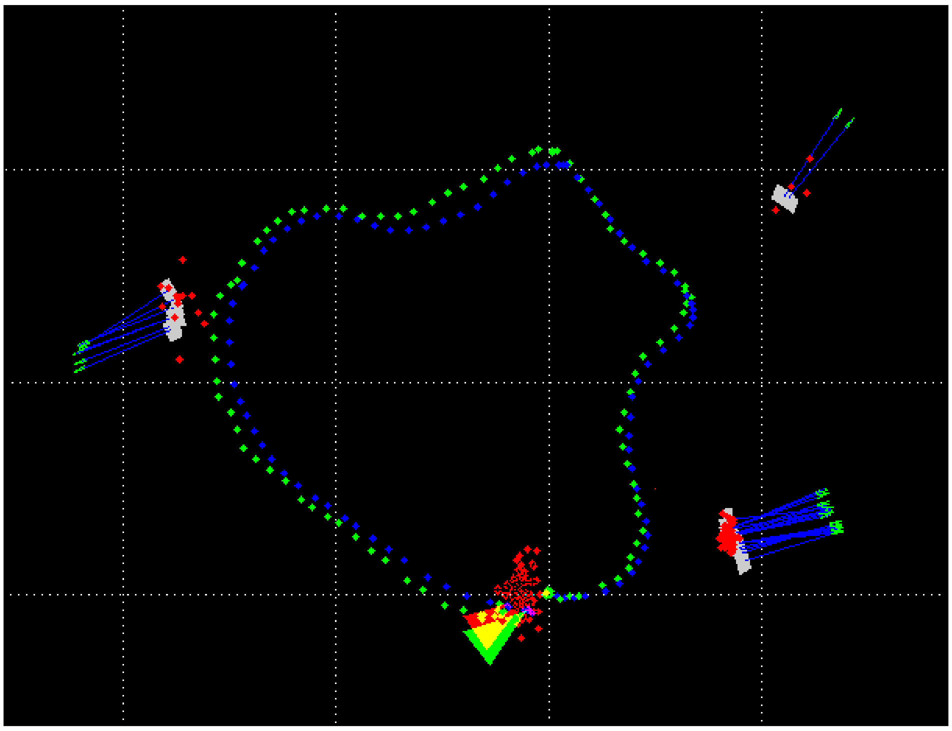


Figure C-7 Screenshot from the simulation environment *(MATLAB image)*.

The key properties of the visual environment are.

- Red dots : Particles
- Blue dots: Ground truth odometer data.
- Green dots: Noisy odometer data
- Green triangle: True pose of vehicle
- Red triangle: Maximum weighted pose for vehicle.
- White areas: Founded planes from 3D data
- Blue Vectors: Normal Vector for founded planes.

In the simulation environment there are 2 main choices; simulation with dataset and real time application.

For the first choice the algorithm uses the sensor measurements and the vehicle's ground truth pose data. These sensor measurements and the ground truth information are taken from the RGBD SLAM dataset. The algorithm simulates the odometer motion model, injects noise into the calculated odometer data and evaluates the performance of algorithm in relation to these injected noises.

The second choice is about controlling the robot in real time and evaluating the difference between determined path and calculated path using sensor measurements *(point, plane, surf)*. This time there is no ground truth data. The commanded motion data is erroneous because of motion errors and thus, the algorithm tries to decrease the error rate using sequential sensor measurements.

On the other hand for the real time application the robot does not have a path planning algorithm. It moves according to given path with some errors.

The required programs and installation process are given in Appendix E.

APPENDIX D

CODE DETAILS

Initialization

The program initializes the simulation and vehicle parameters at the beginning of the fastSLAM application. These parameters are as follows.

FastSLAM Simulation Parameters

CNF.NPARTICLES = inPr.nOfParticles;

The user can define the number of particles for simulation or real time application. The default value is 100.

CNF.NEFFECTIVE = (inPr.n_effect/100)*CNF.NPARTICLES;

The user can define the effective particle rate in terms of percentage. The default value is 90. The algorithm runs the re-sampling algorithm when the effective particle number is under the defined value.

CNF.SWITCH_RESAMPLE = inPr.sw_resample;

The user can select running re-sampling function. Re-sampling is an important and indispensable function for fast-SLAM applications. As a default, it is selected and user can uncheck this selection for debugging purposes.

CNF.STEP_SIZE = inPr.stepSize;

The user can select the simulation step size from GUI. The default value is 10 for the simulation and 1 for the real time application. For example; one of the datasets has 1209 RGBD data *(depth image and RGB image)* for 72 seconds of motion. Setting a step size 1 is the ideal case but this results in a high computational cost. In the real time application, the vehicle moves, stops, takes sensor data, evaluates it and then moves again. The value is meaningless for real time applications.

inPr.showSteps

If the user checks the selection box, the algorithm shows the navigation and map.

inPr.corrProp

The user can select different landmark vectors. The algorithm corresponds the sensor measurements according to the selected landmark vectors. There are 4 different landmark vectors.

- Plane as point
- Plane as oriented point
- Plane as surface
- Surf points

inPr.per_dec

The user can select the percentage of the raw data. The algorithm removes the selected percentage of data randomly. This reduction accelerates the simulation.

inPr.simID

The user can select the simulation with the RGBD SLAM Dataset or the real time application with the Pioneer robot and Kinect sensor.

inPr.mapID

The user selects the map for simulation. This selection is meaningless for the real time application.

inPr.ground_f

In the applications presented in this work, the filtering ground floor points are important but for debugging purposes the user can select an alternative case.

CNF.BACKGROUND_FILTER_THRESHOLD=3;

The algorithm filters the point group if the mean is further than the given threshold. Background filtering threshold is in meters.

```
CNF.GROUND_FILTER_THRESHOLD = 0.2;
```

The algorithm fits plane to the ground floor data and filters points closer than 0.2 meters to the plane.

```
CNF.NUMBER_OF_SAMPLE_GROUND_POINT = 2000;
```

2000 points is sufficient to define the planar surface for ground floor.

```
CNF.DATA_ASSOCIATION_TRESHOLD = inPr.tresh/inPr.stepSize;
```

The user defined threshold is valid if the user defined step size equals to 1, otherwise its value calculated with the formula above.

Vehicle Model and Motion Noise Parameters

```
CNF.VEHICLE_MODEL = [ 0.0    -0.6    -0.6  ;   % WHEELBASE
                      0.0     0.2    -0.2  ;   % WIDTH
                      0.0     0.0     0.0  ];
```

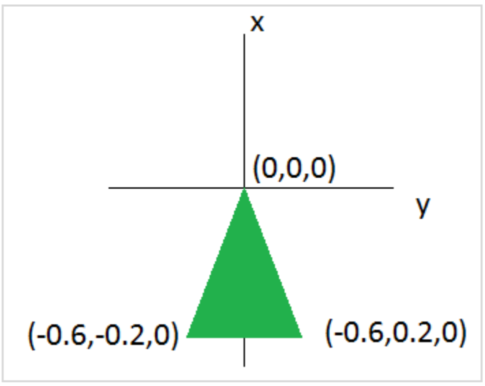


Figure D-1Vehicle Model and Start Condition

The vehicle is modeled as a triangle for the easy visualization of simulations. The vehicle width and wheelbase is defined as 0.4 and 0.6 meters. These are default values and user cannot change the vehicle model from the GUI. The start condition of the robot defined as [0, 0, 0].

```
MAX_TRANSFORMATION = 0.05*inPr.stepSize;         % meters
MAX_ROTATION = (3*inPr.stepSize)/180*pi;               % degree
```

The maximum motion at one step is defined as 0.05 meters for one step and it is scaled to step size for the application. It is the same for the rotational velocity. Its value is defined as 3 degree and scaled to step size.

```
perTr =inPr.perV;       % Error rate for linear velocity
perRot = inPr.perW;   % Error rate for angular velocity
```

Algorithm injects these error rates to the extracted data from ground truth.

```
sigmaTr = MAX_TRANSFORMATION * perTr/100;    %Trans. variance.
sigmaRot = MAX_ROTATION * perRot / 100;       %Rot. variance.

CNF.Q = [sigmaTr^2                 0         ;
                      0                  sigmaRot^2];
```

The variances of motion are defined with error rates and the allowed maximum velocities.

Sensor Model and Measurement Noise Parameters

```
σ_R = 0.2;                 % Variance value for range value – 0.2 meters
σ_B = 5*(pi/180);     % Variance value for bearing value – 5 degree
σ_NV = 0.1;              % Variance value for normal vector value – 0.1
σ_AREA = 0.5;        % Variance value for area of plane value – 0.5 m2
σ_SCALE = 0.1;      % Variance value for surf feature scale  - 0.1
```

In the application there is no any range bearing sensor but the found values are modeled as the range bearing sensor. The algorithm calculates the range of point, bearing values, normal vector, plane area or surf point scale. The found variance values are listed above.

The sensor measurement noise variance matrix differs according to selected feature type. Two are given below.

For plane as point:

```
CNF.R= [σ_R^2      0          0    ;
           0       σ_B^2      0    ;
           0         0      σ_B^2    ];
```

For plane as surface:

```
CNF.R= [σ_R^2    0        0       0        0         0         0;
          0     σ_B^2     0       0        0         0         0;
          0       0     σ_B^2     0        0         0         0;
          0       0       0     σ_NV^2     0         0         0;
          0       0       0       0      σ_NV^2      0         0;
          0       0       0       0        0       σ_NV^2      0;
          0       0       0       0        0         0      σ_AREA^2];
```

SLAM Animation Setup

setup_animations() function prepares all the animation environment. In this study animation environment generated as 20 x 20 x 20 cubical space. The visualization parameters about the vehicle and the landmarks are initialized.

Initialization of Particles

initialise_particles() function generates the start condition of all particles. At the start condition all particles contain the same data.

```
p(i).w= 1/np;                  %Particle weight
p(i).xv= [0;0;0;0;0;0];        %Predicted vehicle pose
p(i).xf= []; p(i).Pf= [];      %Founded landmarks and their covariance
p(i).count = [];               %Count number for every detected landmark
```

Reading Ground Truth and Scan List for Simulations

```
depth_scan_list = readDepthScanList(inPr.mapID);
```

This list keeps the name of the depth images sequentially. Using this list the algorithm obtains the proper depth image for selected scan.

```
ground_truth = readGroundTruth(inPr.mapID);
ground_truth = setStartZero(ground_truth);
```

The algorithm obtains the ground truth data for the vehicle pose and shifts this ground truth data to the start condition *(0, 0, 0)*.

Getting True value of Odometer Data

```
xtrue = calculateXtrue(depth_scan_list(scan_counter).id, ground_truth);
noisyOdometryData = calculateOdometryData(xtrue,xtrue_past,CNF.Q,1);
preFromOdometry = preFromOdometry+noisyOdometryData;
```

For example, for the first map there are 1209 scan steps. On the other hand the ground truth data has 21823 data lines. The scanning and ground truth frequencies are different and the proposed algorithm has to match the appropriate ground truth for relevant scan data. This correspondence is undertaken using the time stamps of the ground truth data and scan steps.

For example, for the depth scan image **“1311876800.398210.png” the** algorithm uses the ground truth data marked in red and the closest one is selected.

1311876800.3849 -1.8198 -0.7560 0.5685 0.1558 0.7219 -0.6603 -0.1361
1311876800.3883 -1.8198 -0.7560 0.5685 0.1559 0.7219 -0.6603 -0.1360
1311876800.3916 -1.8199 -0.7562 0.5686 0.1556 0.7215 -0.6609 -0.1358
1311876800.3950 -1.8199 -0.7562 0.5686 0.1555 0.7215 -0.6609 -0.1357
1311876800.3983 -1.8199 -0.7560 0.5686 0.1559 0.7218 -0.6604 -0.1360
1311876800.4017 -1.8199 -0.7560 0.5686 0.1559 0.7218 -0.6604 -0.1360

```
1311876800.4050 -1.8198 -0.7560 0.5686 0.1558 0.7219 -0.6604 -0.1360
1311876800.4083 -1.8199 -0.7562 0.5685 0.1555 0.7215 -0.6610 -0.1358
```

In fact, some small errors occurred in this process. For time stamp **1311876800.398210** the pose data is matched with **1311876800.3983.** This error is very small, and negligible so fastSLAM algorithm can handle this issue.

Prediction Step

```
for i=1:CNF.NPARTICLES
        particles(i)= predict(particles(i), noisyOdometryData, CNF.Q );
end
```

The algorithm runs this ***predict()*** function for each particle. The input parameters for predict function are noisy odometer data, odometer data noise variance and relevant particle.
In the predict function the algorithm takes samples from the Gaussian space of odometer data with ***normrnd()*** function.

```
odometryData(1,1) =  normrnd( odometryData(1,1),sqrt(Q(1,1)));
…
```

Filtering Process

```
[ last_data_filtered last_data_filtered_wback]= getFilteredScan(CNF
,depth_scan_list, scan_counter,inPr);
```

This function eliminates the unnecessary point groups and extracts the important part of data that can be defined as plane.

Image Data to 3D Point Cloud Data Conversion

```
data = convertTo3D(dImageFiltered,rgbImage);
```

This function generates the 3D point cloud data using camera focal length and camera calibration data. The calibration parameters given below are taken from the RGBD-SLAM dataset.

```
focalLengthX = 520.9;
focalLengthY = 521.0;
centerX = 325.1;
centerY = 249.7;
scalingFactor = 5000.0;
```

The default camera calibration data values are listed below.

```
 focalLengthX = 525.0;
 focalLengthY = 525.0;
 centerX = 319.5;
 centerY = 239.5;
 scalingFactor = 5000.0;
```

The algorithm undertakes this calculation for all points on the image data. There are nearly (640x480) 307200 points on the image.

```
Z = double( depth(v,u)) / scalingFactor;
X = (u - centerX) * Z / focalLengthX;
Y = (v - centerY) * Z / focalLengthY;
```

All these extracted 3D points are rotated to the vehicles coordinate frame.

```
R_toMap = [ 0  0  1;
           -1  0  0;
            0 -1  0];
data(1:3,:) = R_toMap* data(1:3,:);
```

Filtering Ground Points

```
[n_est ro_est] = findGroundNormal(data_filtered,…
```

This function finds the ground normal and with this normal value the algorithm fits plane to the ground data. This process is applicable for the simulations and run-time applications in the current study because the navigation area is planar.

```
data_filtered = removeGroundEffect(data,n_est,ro_est,CNF);
selectedPoints = find(distanceToPlane>t);
```

The algorithm finds the distance between the points and plane. According to the defined threshold "CNF.GROUND_FILTER_THRESHOLD" the algorithm removes the points if the distance is smaller than the specified threshold.

Filtering Background and Unnecessary Points

```
[data_filtered backgroundData] = removeBackgroundEffect(data,CNF);
```

The algorithm removes the background data and the other point groups that are not suitable for defining a plane. The algorithm selects 2000 points randomly from the point cloud data and classifies the points groups according the Euclidian distance to each other with hierarchical clustering *(unsupervised clustering method)*. The stopping criteria is important for hierarchical clustering and is defined as 0.205 meters.

```
CUTOFF =0.205;      %Stopping criteria for hierarchical clustering.
```

Then the algorithm removes the point groups if the group size is below 50 and over 1500.

```
noisyPoints = find(sizeList<50);
```

If group size is smaller than 50 the algorithm evaluates this point group as noise.

```
noisyPoints = find(sizeList>1500);
```

If this higher than 1500, this means that it covers the 75% of the scene and possibly the group constitutes a background.

```
noisyPoints= find(meanList>CNF.BACKGROUND_FILTER_THRESH);
```

On the other hand, if the mean of a group is far from the specified threshold the algorithm removes that group, because the data is unreliable and possibly part of the background.

```
noisyPoints = find( mAngleList>28 );
```

If the angle of any point in group exceeds 28 degrees the algorithm classifies this group as unusable. The algorithm decides that the point group is a part of the object and the fitting plane of this point group injects erroneous landmarks into the algorithm.

```
data_filtered(iL).Cov(1,1)>0.1 || data_filtered(iL).Cov(2,2)>0.1 || data_filtered(iL).Cov(3,3)>0.1
```

Now the algorithm has the point groups selected from 2000 points and to better define the plane the neighboring points *(ungrouped ones)* are added to this groups. The algorithm calculates the standard deviations of all the clusters and if they are higher than the specified threshold the algorithm removes relevant cluster.

Finally, the found clusters became suitable for plane feature extraction.

Feature Extraction

The feature extraction algorithm is selectable according to the given parameters from GUI.

Plane Feature Extraction

The plane extraction algorithm runs, if the selected landmark type is **plane as point**, **plane as oriented point** or **plane as surface.**

```
last_data_planes = extractPlanes(last_data_filtered,CNF);
```

extractPlanes() function extracts the plane feature parameters. The details of the plane feature extraction process is given in Section 4.2.

SURF Feature Extraction

The SURF feature extraction algorithm runs, if the selected landmark type is **SURF point**.

```
[iPoints  index]= surf_findSurfFeatures(image);
```

Surf_findSurfFeatures() function extracts the surf feature points. The details of the SURF feature extraction process is given in Section 4.3.

Range-Bearing of the Sensor Modelling

```
z = slam_get_features(last_data_planes);
```

slam_get_features() function extracts the sensor measurement from the plane features. The algorithm fills the z value for the selected landmark type.

```
z(1:3,i) = xyzToRangeBearing(z_3D(i));        % center point of plane
z(4:6,i)= z_3D(i).n;                           % normal vector of plane
z(7,i) = z_3D(i).A;                            % area of plane
```

The sensor measurement vector contains; range, bearing, normal vector and area information.

```
z = surf_findInterestPointMeas(iPoints,image);
```

On the other hand the ***surf_findInterestPointMeas()*** function finds the sensor measurement data which contains range, bearing, scale and laplacian.

```
z(1:3,counter) = xyzToRangeBearing(point);    % location of surf point
z(4,counter)= iPoints(i).scale;                % scale of surf point
z(5,counter)= iPoints(i).laplacian;            % laplacian
```

Data Association

```
[particles(i)] = slam_dataAssociateUnknown(…)
```

slam_dataAssociateUnkonwn() function makes the data association. The robot directly adds the new landmark if it is the first landmark vector.

```
particle = slam_addFeature(particle, z, R, corrProp);
```

Otherwise runs the association process for the detected landmark and the algorithm tries to understand whether the feature is new or old. If it is new the algorithm goes to the ***slam_addFeature()*** function and adds the detected landmark vector. If the sensor measurement is from an earlier detected landmark, the algorithm updates the landmark properties. These landmark properties are the landmark vector, landmark covariance, weight and count.

Visualization of Parameters

```
plotPlanes(last_data_planes,xtrue,z);
```

This ***plotPlanes()*** function handles all the steps that help to plot the plane.

```
do_plot(h, particles, xtrue, VEH);
```

do_plot() function plots particles, true state of vehicle and the prediction of vehicle state for the maximum weighted particle.

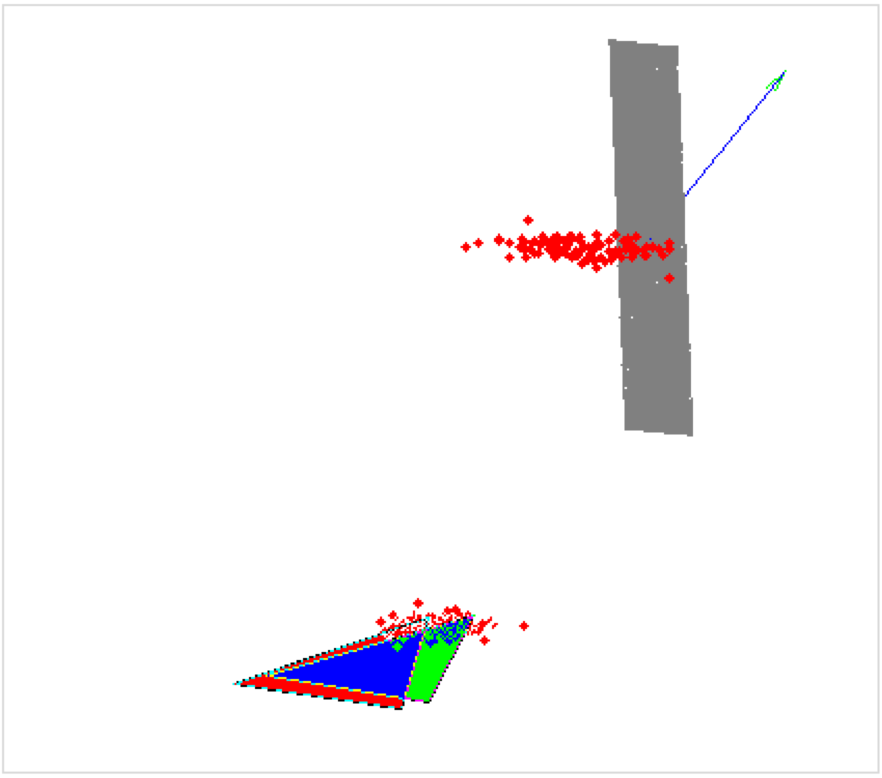


Figure D-2 Screenshot from simulation environment *(MATLAB image)*

In figure D-2 the gray area defines the plane, red points define the particles for landmark vector and the arrow defines the normal vector for the maximum weighted particle. Also green triangle is the true state of vehicle and red triangle is the prediction for maximum weighted particle.

Re-sampling Step

```
[particles]= slam_resampleParticles(…)
```

The re-sampling process is the most important part of the fastSLAM applications. If the effective number of particles decreases under the defined threshold, the algorithm runs re-sampling process.

```
ws= sum(w);         % Total of all weights
w= w/ws;             % Normalized weight vector.
```

Firstly, the algorithm normalizes the weight values and sends these values to the re-sampling function.

```
[keep, Neff] = stratifiedResample(w);
```

stratifiedResample() function finds the effective particle number and if it is under the specified threshold keeps the high weighted samples and removes the low weighted ones. So the high weighted samples are copied by the function. The stronger particles survive and weak ones disappear through the fastSLAM process.

Finalization of Simulation

If there is any other scan step, the algorithm computes the true value of the odometer data. Otherwise it ends the simulation. After finalization the algorithm computes the time consumption and error values. The it draws the error-time graph of simulation.

APPENDIX E

INSTALLATION

To run the simulation and real time applications the user should install the following programs.

1. Matlab 2012a for the simulation environment.
2. KinectSDK-v1.0-beta2-x64 _ 1.0.0.12 for the Kinect Camera.
3. MobileSim-0.5.0.exe and ARIA-2.7.5.2.exe to control the pioneer robot in real time.

In addition The dataset path in ***configParameters()*** function for each map should be defined as follows.

```
CNF.DEFAULT_DATA_PATH = 'D:\DATA\FRE2_360\';
```

The COM1 serial RS-232 port should be defined to communicate with the computer and the Pioneer 2 robot. Finally, the communication cable should be cross cable.