# A NOVEL MOBILE ROBOT NAVIGATION METHOD BASED ON COMBINED FEATURE BASED SCAN MATCHING AND FASTSLAM ALGORITHM

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

BY

AYHAN ÖZGÜR

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

SEPTEMBER 2010

Approval of the thesis:

# A NOVEL MOBILE ROBOT NAVIGATION METHOD BASED ON COMBINED FEATURE BASED SCAN MATCHING AND FASTSLAM ALGORITHM

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

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _______________

Prof. Dr. İsmet Erkmen
Head of Department, **Electrical and Electronics Engineering** _______________

Assist. Prof. Dr. Afşar Saranlı
Supervisor, **Electrical and Electronics Engineering, METU** _______________

Assist. Prof. Dr. İlhan Konukseven
Co-Supervisor, **Mechanical Engineering, METU** _______________

**Examining Committee Members:**

Prof. Dr. Aydan M. Erkmen
Electrical and Electronics Engineering, METU _______________

Assist. Prof. Dr. Afşar Saranlı
Electrical and Electronics Engineering, METU _______________

Prof. Dr. Kemal Leblebicioğlu
Electrical and Electronics Engineering, METU _______________

Assist. Prof. Dr. Yiğit Yazıcıoğlu
Mechanical Engineering, METU _______________

Bülent Bilgin, M.Sc.
Manager, ASELSAN _______________

**Date: 24.09.2010**

**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 : Ayhan ÖZGÜR

Signature :

# ABSTRACT

## A NOVEL MOBILE ROBOT NAVIGATION METHOD BASED ON COMBINED FEATURE BASED SCAN MATCHING AND FASTSLAM ALGORITHM

Özgür, Ayhan
M. Sc. Department of Electrical and Electronics Engineering
Supervisor: Assist. Prof. Dr. Afşar Saranlı
Co-Supervisor: Assist. Prof. Dr. İlhan Konukseven

September 2010, 162 pages

The main focus of the study is the implementation of a practical indoor localization and mapping algorithm for large scale, structured indoor environments. Building an incremental consistent map while also using it for localization is partially unsolved problem and of prime importance for mobile robot navigation. Within this framework, a combined method consisting of feature based scan matching and FastSLAM algorithm using LADAR and odometer sensor is presented. In this method, an improved data association and localization accuracy is achieved by feeding the SLAM module with better incremental pose information from scan matching instead of raw odometer output.

This thesis presents the following contributions for indoor localization and mapping. Firstly a method combining feature based scan matching and FastSLAM is achieved. Secondly, improved geometrical relations are used for scan matching and also a novel method based on vector transformation is used for the calculation of pose difference. These are carefully studied and tuned based on localization and mapping performance failures encountered in different realistic LADAR datasets. Thirdly, in addition to position, orientation information usage in line segment and corner oriented data association is presented as an extension in FastSLAM module.

The method is tested with LADAR and odometer data taken from real robot platforms operated in different indoor environments. In addition to using datasets from the literature, own datasets are collected on Pioneer 3AT experimental robot platform. As a result, a real time working localization algorithm which is pretty successive in large scale, structured environments is achieved.

**Keywords:** Scan matching, LADAR based feature extraction, Simultaneous localization and mapping.

# ÖZ

## ÖZNİTELİK TABANLI MESAFE EŞLEME VE FASTSLAM ALGORITMALARI BİRLEŞİMİ ÖZGÜN HARAKETLİ ROBOT NAVIGASYON YÖNTEMİ

Özgür, Ayhan
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Assist. Prof. Dr Afşar Saranlı
Ortak Tez Yöneticisi: Assist. Prof. Dr İlhan Konukseven

Eylül 2010, 162 sayfa

Bu çalışmadaki temel amaç, geniş boyutlu detaylı iç ortamlarda pratik iç ortam konumlama ve haritalama algoritması gerçeklenmesidir. Biriken tutarlı bir harita oluştururken bunun konumlama için kullanılması kısmen çözülmemiş bir problem olup hareketli robot navigasyonunda başlıca öneme sahip bir konudur. Bu çatı altında, LADAR ve odometre algılayıcılarını kullanılarak öznitelik tabanlı mesafe eşleme ve FastSLAM algoritmaları birleşimi bir yöntem sunulmaktadır. Bu yöntemde, SLAM modülüne odometre çıktısı yerine daha iyi bir konumlama bilgisi verilerek, iyileştirilmiş veri eşleme ve konumlama hassasiyeti sağlanmaktadır.

Bu tez iç ortamda konumlama ve haritalama ile ilgili şu katkıları sağlamaktadır: Birinci olarak, öznitelik tabanlı FastSLAM ve mesafe eşleme algoritmalarını birleştirmektedir. İkinci olarak mesafe eşleme için geliştirilmiş geometrik ilişkiler kullanılmıştır; ek olarak konum farkları için de vektör taşınmasına dayalı yeni bir metot önerilmiştir. Bu yöntemler üzerinde dikkatle çalışılıp farklı gerçekçi LADAR veri kümelerinde, konumlama ve haritalama hassasiyetinde oluşan hatalara göre düzenleme yapılmıştır. Üçüncü olarak, doğru parçası ve köşe tabanlı veri eşlemesi için konuma ek olarak, yön bilgisi kullanımı da FastSLAM modülüne bir eklenti olarak verilmiştir.

Algoritmalar farklı iç ortamlarda çalışan gerçek robot platformlarından alınan LADAR ve odometre verileri ile test edilmiştir. Literatürdeki bu veri kümelerine ek olarak, Pioneer 3AT deney robot platformlarıyla alınmış kendi veri kümelerimiz de kullanılmıştır. Sonuç olarak geniş boyutlu detaylı alanlarda çalışabilen gerçek zamanlı bir konumlama algoritması oluşturulmuştur.

**Anahtar kelimeler:** Mesafe eşleme, LADAR tabanlı öznitelik çıkarımı, Eş zamanlı konumlama ve haritalandırma

# ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor Assist. Prof. Dr Afşar Saranlı and co-supervisor Assist. Prof. Dr İlhan Konukseven for their guidance, advice, criticism, encouragements and insight throughout the research.

I would like to thank to ASELSAN Inc. for the facilities provided and my colleagues for their support during the course of the thesis.

I would also like to thank to Turkiye Bilimsel ve Teknolojik Arastirma Kurumu (TUBITAK) for their contribution to my Master of Science education with the scholarship.

I appreciate my family due to their great trust.

The last but not least acknowledgment is to my precious wife Öykü Bilgiç Özgür for her encouragement and continuous emotional support.

# TABLE OF CONTENTS

APPENDICES

# LIST OF SYMBOLS

| **Symbol** | **Description** |
|---|---|
| $\ldots curr$ | The structure belong to current scan, such as $\mathrm{C}_{\mathrm{curr}}{}^{[\mathrm{m}]}$, $\mathrm{C}_{\mathrm{curr}}{}^{[\mathrm{m}]}.ID$, $Cnr_{curr}{}^{[m]}$, $Angle\_P_{curr}{}^{[m]}\ldots$ |
| $\ldots prev$ | The structure belong to previous scan, such as $\mathrm{C}_{\mathrm{prev}}{}^{[\mathrm{m}]}$, $\mathrm{C}_{\mathrm{prev}}{}^{[\mathrm{m}]}.ID$, $Cnr_{prev}{}^{[m]}$, $Angle\_P_{prev}{}^{[m]}\ \ldots$ |
| $(x_p\ ,y_p,\theta_p)$ | Estimated pose of the robot in particle |
| $\varepsilon_{rot1}$ | First rotation, control input of vehicle according to odometer motion model |
| $\varepsilon_{trans}$ | Translation, control input of vehicle according to odometer motion model |
| $\varepsilon_{rot2}$ | Second rotation, control input of vehicle according to odometer motion model |
| $\theta^{[\mathrm{i}]}$ | Angle for ith member of range scan. In our case angle interval is 1 So $\theta^{[\mathrm{i}]}$=*i*. |
| $Angle\_P^{[m]}$ | Corner angle pattern, constructed from the slope difference of 6 consecutive lines, between the first one and other five |
| $\mathrm{C}^{[\mathrm{m}]}$ | Cluster ID |
| $\mathrm{C}^{[\mathrm{m}]}.ID$ | The real ID's of cluster points, which is the angle interval in the ran ge scan |
| $C^{[m]}.L$ | Fitting line belong to *m*th cluster |
| $C^{[m]}.L.dis^{[n]}$ | The distance nth point of *m*th cluster to with fitting line |
| $C^{[m]}.L.Length$ | The length of the line belong to *m*th cluster |
| $C^{[m]}.L.Lineclass$ | The class of fitting line (both, left, right side ended or open) |
| $C^{[m]}.L.Slope$ | The angular difference from x axis for the fitting line belong to *m*th cluster |
| $C^{[m]}.x,y$ | The cartesien coordinates of cluster points |

| | |
|---|---|
| $Cnr^{[m]}$ | Corner ID |
| $Cnr^{[m]}.Angle$ | The angle between the line pairs constructing the corner |
| $Cnr^{[m]}.Class$ | Information whether corner is real / virtual |
| $Cnr^{[m]}.Lpair$ | Line pair constructing $\mathrm{Cnr}^{[\mathrm{m}]}$ |
| $Cnr^{[m]}.x, y$ | The  cartesien coordinates of  corner |
| $d^{[i]}$ | Distance measurement for ith member of range scan |
| $edgeDist\_P^{[m]}$ | Corner distance pattern consist of the first line and the slope differences between the first line and five consecutive lines |
| *idf* | ID of the landmark observed |
| *Gu* | Jacobian matrix of  robot pose with respect to control input |
| *Gv* | Jacobian matrix of  robot pose with respect to vehicle states |
| *Gz* | Jacobian matrix of the landmark position with respect to the vehicle states |
| *Hv* | Jacobian matrix of predicted landmark observation with respect to vehicle states |
| *Hf* | Jacobian matrix of predicted landmark observation with respect to feature positions |
| $matched\_LP^{[m]}$ | The [*i, j*] ID's of the matched line pairs $\mathrm{C}_{\mathrm{prev}}^{[\mathrm{i}]}.L$ *and* $\mathrm{C}_{\mathrm{curr}}^{[\mathrm{j}]}.L$ |
| *MTx()* | Sub matching table of Criterion X |
| *MT(m,n)* | The voting given between the $\mathrm{C}_{\mathrm{prev}}^{[\mathrm{m}]}.L$ *and* $\mathrm{C}_{\mathrm{curr}}^{[n]}.L$ in matching table |
| *particle* | Particle class used in FastSLAM |
| $particle^{[\mathrm{n}]}.xv$ | Estimated pose of the robot for $particle^{[\mathrm{n}]}$ |
| $particle^{[\mathrm{n}]}.pv$ | Estimated pose covariance of the robot for $particle^{[\mathrm{n}]}$ |
| $particle^{[\mathrm{n}]}.xf$ | Estimated landmark position matrix for $particle^{[\mathrm{n}]}$ |
| $particle^{[\mathrm{n}]}.pf$ | Estimated covariance matrix of landmark positions for $particle^{[\mathrm{n}]}$ |
| *Q* | Measurement inputs noise matrix |
| *result_rotation* | The rotation found between consecutive LADAR range scans |
| *result_translation* | The translation found between consecutive LADAR range scans |
| *robot_pose* | Estimated pose of the robot |
| *R* | Control  inputs noise matrix |

*zf* Observed distance and bearing information, referenced to robot, of landmarks re-observed.

*zn* Distance and bearing information, referenced to robot, of landmarks observed first time.

*zp* Predicted distance and bearing information, referenced to robot pose in particle, of landmarks re-observed.

* Further information about the structure of the given symbols is given in Appendix.

# LIST OF ABBREVIATIONS

| **Symbol** | **Description** |
|---|---|
| INS | Inertial navigation system |
| GNSS | Global navigation satellite system |
| LADAR | Laser detection and ranging |
| EKF | Extended Kalman filter |
| SLAM | Simultaneous localization and mapping |
| CML | Concurrent mapping and localization |
| IMU | Inertial measurement unit |
| DGL | Deterministic global localization |

# CHAPTER 1

# INTRODUCTION

## 1.1 Motivation

One of the main issues in robotics is localization. It affects different main areas of robotics, such as mapping, navigation, path planning. Localization can be classified in two areas: Indoor and outdoor.

Outdoor localization can be achievable very easily by using a GNSS receiver [1]. GNSS is so important for outdoor localization, since it continuously gives global position. Nowadays there are many COTS GNSS products (that have the GNSS carrier signal process ability), that could provide centimeter level accuracy in position when a fully sky view is provided to the GNSS antenna. Also when two GNSS antennas are located on the robot with enough displacement, which is usually more than a meter, good heading accuracy is obtained. When a GNSS loss is occurred, this problem is mostly solved by using dead reckoning solutions, such as multi-sensorial solution based on odometer and inertial sensors (magnetometer, accelerometer and gyro). This kind of multi-sensorial solution is named as INS (Inertial Navigation Systems) [2]. By adding GNSS receiver to the INS system, a complete solution is provided for the localization. The accuracy of the given pose mostly depends on the quality of the sensors used in the solution.

Indoor localization is much harder than outdoor localization, since GNSS receiver does not work, which means a global reference could not provided for the localization. Again like the outdoor, INS or a solution based on inertial sensors and odometer can be provided. But this comes with a cost which is much more then the accuracy it gives. But there is one more point for indoor localization; it has a solution based on using some low cost sensors, which can not applicable for outdoor area, except the area is dense structured.

For indoor localization, measurement sensors are used. These measurement sensors are infrared sensors, ultrasonic sensors, radars and laser scanners. In our solution we have used laser scanner, which is also known as LADAR (Laser Detection and Ranging) [3]. The principle of the LADAR is that it sends multiple laser beams to the environment, and finds the distance from the incoming laser beam reflections. After this process, it gives an output to the user, known as a range scan, which is a list of points corresponding to the intersection points of a laser beam with objects in the robot's environment. The laser beam rotates in a horizontal plane and emanates from a sensor mounted on the robot. Thus a range scan describes 2D slice of the environment. Points of the range scan can be specified in a polar coordinate system whose origin is the location of the sensor, and the reference axis for the laser beam direction is the home orientation of the rotating range sensor. Each scan point is represented by the laser beam direction, and the range measurement along that direction. Suppose that the robot starts at pose Pref (which is our reference pose) and takes a scan (call it the reference scan Sref ). The robot then moves through a static environment to a new pose Pnew and takes another scan (call it the new scan Snew). The approximate difference of pose Pnew from pose Pref (i.e. the relative translation and rotation) is usually known from odometer information. However, this information is often imperfect due to wheel slippage. Our task is to determine the exact difference of pose Pnew with respect to pose Pref, by aligning the two scans. This application is known as scan matching.

Scan matching does not form the full solution. The localization obtained by scan matching is pretty good when it has compared with odometer pose. But this does not mean always a good localization will be obtained, by the increase in total distance passed the localization accuracy decreases. In fact during the pose estimation by the integration of pose differences, also the pose error integrates. As a result of this localization error, after a while the navigation of the robot and also mapping destroys. When a robot has come to a point that it has previously passed, the amount of slippage in pose of the robot is seen. To solve this problem, the area in the robot path is investigated and some unique and easily accessible features are registered in memory, to remember where you are when you have seen them again.  As a result of this, when a robot has came to previously seen area, again it observes these features. So it can relocate itself. As a result, the pose error of the robot always stays in bounds during its navigation making closed loops in a fixed structured indoor environment. Our task is also to keep the incremental pose error at limits during the navigation of the robot and create a consistent map, which is known as simultaneous localization and mapping (SLAM).

## 1.2 Scope of the Thesis

The scope of this algorithm is, to provide indoor pose estimation for MAGIC 2010 competition robots. Inputs are laser scanner readings, raw odometer *$(x,y,\theta)$* pose estimation . Outputs are global indoor 2D pose, suitable for replacing GPS inside a building raw local occupancy grid map.

This algorithm finds the robot pose by using feature based scan matching algorithm. Algorithm calculates the pose based on LADAR readings and uses odometer data for low structured environment that scan matching pose output could not take. Algorithm is improved with deterministic global feature matching techniques and also FastSLAM algorithm to increase pose accuracy in closed loop path cases. The algorithm provides the user, the robot pose and raw occupancy grid map. FastSLAM algorithm also gives post processed corrected path and mapping when it is used as full SLAM..

## 1.3 Primary Contributions

In this thesis, the following items are given as primary contributions:

- A combined localization method is presented based on feature based scan matching and FastSLAM algorithms with a joint feature set. Scan matching is given as a novel voting based algorithm while FastSLAM is given with a novel combined landmark observation and data association structure.
- The methodology used in scan matching gives the algorithm ability to cope with real LADAR data.
- Applied criterion based on matching the combination of multiple feature properties improves the correct matching ability of scan matching algorithm.
- A more accurate method is applied for the translation calculation between cognitive scans in scan matching.
- A real time working simple deterministic global localization algorithm is presented that improves scan matching localization in long open loop paths where FastSLAM does not applicable.
- Line and corner based data association with a validity gating on features is applied and orientation information of landmarks is added to state vector in EKF and data association together.

## 1.4 Outline of the Thesis

The structure of the thesis can be presented as follows:

In Chapter 2, a survey of work previously done in scan matching and line extraction algorithms is presented.

In Chapter 3, the theoretical background and the implementation of scan matching algorithm is presented. This part includes seven sub parts; these are the general structure of scan matching algorithm, sensor data information, line extraction algorithm, feature and feature properties extraction, main scan matching, global localization and occupancy grid mapping.

In Chapter 4, the theoretical background and the implementation of FastSLAM algorithm with the modifications in data association and measurement update parts according to needs in this thesis is presented.

In Chapter 5, the performance of the algorithms for sensor datasets taken from three different environments is presented.

In Chapter 6, the current work is summarized and the work done is discussed.

# CHAPTER 2

# LITERATURE SURVEY

In this section, the knowlcorner in the literature about the subjects in scope of the thesis is surveyed. These are in order, LADAR based feature extraction algorithms, scan matching methods and simultaneous localization and mapping methods (SLAM). Feature extraction methods are mostly based on line extraction algorithms, which form the infrastructure for scan matching and SLAM algorithms implemented in this thesis. The search in scan matching and SLAM algorithms is specialized on the algorithms based on LADAR data.

## 2.1 LADAR Based Feature Extraction

Feature extraction step is a vital part for the feature based scan matching algorithms. For the feature extraction line extraction and curve extraction methods are necessary components. Curve extraction and scan matching based on curve matching methods are computationally costly; therefore in most of the scan matching based algorithms on line extraction methods are preferred. This approach also gives a unique and dense feature set for scan matching, which consists of lines, corners, as well as structures formed from the combination of all of these. The comparison of line extraction algorithms is presented in [17]. The details given below are based on the results obtained from this paper. Different line extraction algorithms and their properties are summarized below followed by a comparison from the literature of their performances [17]:

**Split-and-merge** [17] is probably the most popular line extraction algorithm which has originated from computer vision. This algorithm is used for line extraction in our algorithm, so further information will be given in the next chapter.

Line regression algorithm [18] is proposed for map-based localization. It is based on the Hough Transform algorithm. The line extraction problem turns into a search problem in model space the Agglomerative Hierarchical Clustering (AHC) algorithm is applied to construct adjacent line segments. The main drawback of this approach is that it is theoretically quite complex and difficult to implement.

**RANSAC (Random Sample Consensus)** [19] is an algorithm mostly used in computer vision. It is a robust fitting algorithm in the presence of data outliers. The main advantages are that it is simple to implement and can be used with many types of features once we have the feature model. The main disadvantage of this approach is that it is iterative and stops when maximum number of iterations reached or too few points left, which makes is bad to use in real time implementations on robots.

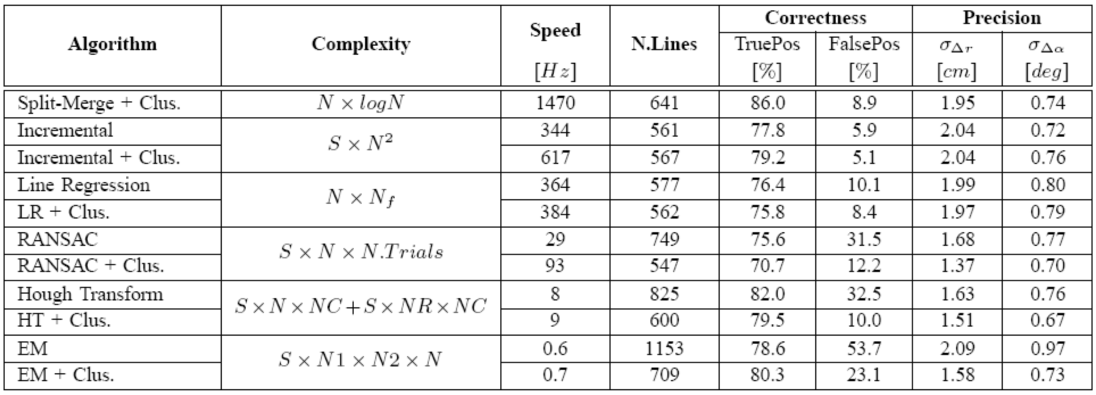
**Expectation-maximization** algorithm (EM) [20] is a probabilistic method and commonly used in missing variable problems. The main drawback is that it can be trapped in local minima and needs good initialization points.

**Hough transform** [21] is mostly used in image processing; it is applied to finding line in intensity images. To extract line segments from range scan data, it is brought to robotics. It has two main drawbacks: The first one is that while estimating the line parameters it does not take noise and uncertainty into account. The other one is that choosing an appropriate grid size is too difficult.

### 2.1.1 Comparison of Line Extraction Algorithms

According to the results given in Table 2.1 (the data is borrowed from [17]), Split and Merge algorithm has minimum complexity and maximum speed according to given table above. Also we can see that considerably good true and false matching rates. So for line extraction part implementation of the feature based scan matching algorithm given in next chapter, Split and Merge Algorithm is used.

**Table 2.1: The comparison of line extraction algorithms [17]. The algorithms are compared based on complexity, speed, correctness and precision criteria.**



| Algorithm | Complexity | Speed | N.Lines | Correctness | | Precision | |
|---|---|---|---|---|---|---|---|
| | | | | TruePos | FalsePos | $\sigma_{\Delta r}$ | $\sigma_{\Delta\alpha}$ |
| | | $[Hz]$ | | [%] | [%] | $[cm]$ | $[deg]$ |
| Split-Merge + Clus. | $N \times logN$ | 1470 | 641 | 86.0 | 8.9 | 1.95 | 0.74 |
| Incremental | $S \times N^2$ | 344 | 561 | 77.8 | 5.9 | 2.04 | 0.72 |
| Incremental + Clus. | | 617 | 567 | 79.2 | 5.1 | 2.04 | 0.76 |
| Line Regression | $N \times N_f$ | 364 | 577 | 76.4 | 10.1 | 1.99 | 0.80 |
| LR + Clus. | | 384 | 562 | 75.8 | 8.4 | 1.97 | 0.79 |
| RANSAC | $S \times N \times N.Trials$ | 29 | 749 | 75.6 | 31.5 | 1.68 | 0.77 |
| RANSAC + Clus. | | 93 | 547 | 70.7 | 12.2 | 1.37 | 0.70 |
| Hough Transform | $S \times N \times NC + S \times NR \times NC$ | 8 | 825 | 82.0 | 32.5 | 1.63 | 0.76 |
| HT + Clus. | | 9 | 600 | 79.5 | 10.0 | 1.51 | 0.67 |
| EM | $S \times N1 \times N2 \times N$ | 0.6 | 1153 | 78.6 | 53.7 | 2.09 | 0.97 |
| EM + Clus. | | 0.7 | 709 | 80.3 | 23.1 | 1.58 | 0.73 |

## 2.2 Scan Matching

Scan matching is a popular way of recovering a mobile robot's motion. Given two scans recorded at two different positions, the translation and the rotation that aligns these scans give an estimate of the robot's path between these scans. In fact, the first scan serves as reference scan and has previously been taken or is stored in a-priori map of scans. The second scan (called current scan) is matched against the reference scan and its scan position is determined relatively to the position of the reference scan. The result of this match is a position correction which can be applied to the current robot position. Such an estimate can be used as an enhanced odometer, can help to close loops or can be the perceptual basis of relation-based SLAM in the Lu and Milios style [4]. Many ways to calculate such estimates have been proposed but all algorithms share on common attribute: They match pairs of scans.

Scan Matching algorithms based on LADAR data can be divided into two main categories. These are **feature based scan matching algorithms** and **raw data based scan matching algorithms**.

### 2.2.1 Raw Data Based Scan Matching Algorithms

Raw data based scan matching techniques are based on directly matching of raw LADAR data. The other name of this type of matching is point to point matching. These methods can be categorized as iterative scan matching methods, such as iterative matching range point (IMPR)[5], iterative dual correspondence (IDC)[5], point-wise scan matching[6], MbICP[7], histogram based scan matching [8]. Point wise scan matching [6] weights the contribution of each scan point to the overall matching error according to its uncertainty. Metric-based ICP (MbICP) [7] presents a metric-based matching algorithm by point-wise matching dense range scans. It uses geometric distance that takes into account the translation and rotation of a robot simultaneously. Histogram based approaches [8], use a special representation of the scanned data for matching of two consecutive scans. Mostly these methods first align the LADAR data by using the position difference coming from the odometer outputs. Then iteratively they search for the optimum pose alignment to decrease the distance between the points of the two following scans. The main drawback of these methods is they iteratively converge. Minimizing the error is a typical optimization problem with well-known problems. Generally, gradient descent methods are implemented, transforming the second scan such that the distance is reduced. Iteration leads to a local minimum of the error function. Also getting optimum alignment between scans can cause too much time consumption. To solve this drawback maximum time cycle can be added. In fact, adding time cycle can highly reduce the performance of the algorithm. Due to the iterative calculations required to obtain optimal scan matching in these algorithms, computational complexity are high, and in order of $O(n^k)$, in which $k > 1$ and $n$ is the number of scan points [9]. In point-wise approaches, $n$ is approximately two orders of magnitude more than that of feature-based methods. So these algorithms are mostly preferred for post processing of LADAR data.

### 2.2.2 Feature Based Scan Matching Algorithms

Feature based scan matching algorithms are based on feature extraction from raw LADAR data and use them for matching. The main advantage of feature based scan matching algorithms is they have bounded time cycle. Maximum time consumption for one cycle can be calculated for any type of range scans given. Since feature based methods uses high level objects for scan matching, which makes them with less computational complexity then point to point methods [9]. The dependence on the odometer data is much less in feature based methods than point to point ones. Mostly point to point methods needs odometer for initialize their scan pairs (to decrease pose difference between the scans). But there are some feature based methods, represented without odometer usage [10] [11] [12] [13]. These are mostly defined as pure scan matching methods. Main drawback in pure scan matching methods is when the environment is not structured enough, pose calculation could not obtained. So algorithm fails in these points. In [10], a comprehensive geometric model for robot mapping based on shape information is presented. Polygonal lines, called polylines, serve as the basic representation of shape as a structure of boundaries. Matching two shapes means matching two ordered sets of polylines against each other according to their similarity. In [11], a method based on geometrical relations on lines and corners is represented. The method proposed in [12] uses a panorama laser range finder and identifies line segments representing linear structures in the environment. This is accomplished by pattern matching and pattern recognition on line segment sets through a dynamic programming algorithm. The scan matching method proposed in [13] matches two scans by using geometric features based on line segments, also called Complete Line Segment (CLS) relationships. The method singles out complete line segments that represent complete linear structures in the environment and uses them to match between the local and global maps. The feature based methods that also include odometer information are also presented in [14] [15] [16]. In [14], a promising algorithm is proposed for high speed robots. This method is fast, but it mostly fails from the lack of robustness property of feature extraction. In [15], both scans are replaced by stochastic representations (histograms) and matching is solved by finding the maximum of a cross correlation function. The main drawback of the algorithm is especially suited for a perpendicular environment but it is not difficult to modify the algorithm to work in non-perpendicular environments. In [16], a method known as anchor point relation matching (APR) method is presented. APR is a pattern matching algorithm designed for the real-time search of best matching laser scans in a set of given reference scans. The algorithm's output is a number of weighted hypotheses which makes APR especially attractive for probabilistic techniques aiming at global localization capabilities. The algorithm is successful in sampled areas in the map. However a complete localization on all over the map is not presented.

## 2.3 Simultaneous Localization and Mapping

SLAM (Simultaneous localization and mapping) problem, also known as Concurrent Mapping and Localization (CML) problem), searches for whether it is possible for a mobile to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map. The main way used for the solution of the SLAM problem can be given simply with following steps. First, during the motion of the robot through its path, it continuously predicts its pose. This can be done by using a localization algorithm based on the sensors available. In fact, inertial sensors can be used for this purpose, such as encoders, gyro, compass, IMU. As a simple solution example, the integration of encoder outputs in the direction given by compass can be given. Also as a more complex solution, integration of pose differences obtained by stereo vision or scan matching can be used. Second, the global localization of the robot is needed to be achieved. This is done by landmark extraction from the constructed map, matching of these landmarks when they are seen more than once (which is known as data association problem) and the correction in the pose of the robot based on the position and orientation difference between the matching landmarks. In this step, when the landmark is seen again, the incremental pose error of the robot or the orientation error of landmark is reduced to the error level which one is smaller from them. By this, error correction in pose of the robot and also in the landmarks of the map is achieved, simultaneously. Third, during all these steps a consistent mapping of the enviroment is achieved. If the error correction is for all path of the robot, this is known as full SLAM. On the other hand if this correction affects only the current pose than it is known as online SLAM.

Achieving better localization and mapping, in different environments for different robot and sensor levels constructs the diversity in the solution of the SLAM problem, which consists of different solutions in global SLAM method and its sublevels, such as state estimation, landmark extraction, data association, state update and landmark update. As it is mentioned, in Section 2.2, the existence of uncertainty in the robot pose estimation and in the observation data almost obligates the implementation of stochastic algorithms for the estimation of the map and the vehicle localization. Initial works were presented [24, [25], [26]. In [25], it is showed that while the robot moves through an unknown environment by taking relative observations of landmarks, the estimates of these are all correlated with each other because of common error in estimated vehicle location. This a joint state composed of the vehicle pose and every landmark position, to be updated by each landmark observation is required for a consistent full solution to the SLAM problem. In [26], Extended Kalman Filter solution (EKF) is given which involves a recursive update procedure that comprises prediction, observation, and update steps. One of the main issues in SLAM is data association problem. Because of pose and observation data error, the landmarks extracted can be mismatched with the previously extracted ones. This causes wrong pose estimates, which diverges the Kalman filter. To reduce the effect of feature detection errors, in [27] and [28], noval feature management techniques are presented. Also in [28], the idea of provisional feature lists in SLAM is developed.

A key limitation of the EKF solution to the SLAM problem lies in the quadratic nature of the covariance matrix (O($n^2$), where n is number of landmarks). For this purpose EKF SLAM algorithms that gain remarkable scalability through decomposing the map into submaps, in which covariances are held separately are presented [28] [29]. Despite these are efficient, they do not propagate the information through the matrix of local maps. A numbers of researchers are developed hybrid SLAM techniques, in these maps are decomposes maps into local occupancy grid maps and combining these by using expectation maximization algorithm [30],[31]. The particle filtering method [32] is another important alternative to the Extended Kalman Filter. In particle filtering, continuous distributions are approximated by discrete random measures, which are composed of weighted particles, where the particles are samples of the unknown states from the state space, and the particle weights are "probability masses" computed by using Bayes theory. Particle filters are used for localization in [33] representing Monte Carlo Localization for mobile robots.

One of the pioneer methods that combine particle filters with EKF SLAM localization is FastSLAM [34]. FastSLAM uses particle filters for estimating the robot path. Hence mapping problem is factored to many separate problems, one for each feature in the map. It estimates these map feature locations using separate low dimensional EKF for each individual frame. This gives lower computational complexity ( O($n_l * n_p$)), where $n_l$ is the number of landmarks and $n_p$ is the number of particles). FastSLAM can cope with non linear robot motion models, which is not applicable to previous SLAM methods. Other key advantage of FastSLAM is data association decisions can be made on a per-particle basis. As a result, it maintains posteriors over multiple data associations, not just the most likely one like in EKF SLAM. Another advantage comes from using particle filters which can cope with non linear robot motion models.

The main drawback of FastSLAM is that its performance is degraded when high uncertainty is presented in vehicle estimate (or a high number of particles are necessary which increases the computational complexity of the algorithm). For this purpose, there are hybrid approaches that combine some other SLAM approach with scan matching. In [34], a hybrid algorithm that combines FastSLAM with scan matching to minimize odometer error is presented (Thereby reducing the number of particles needed to build large-scale maps). In this paper, the implemented scan matching algorithm is ICP (iterative closest point) and standard FastSLAM 2.0 algorithm is merged. On the other hand, use scan matching to improve the data association robustness of an EKF variant called the geometric projection filter (GPF) [35].

One of the main areas of SLAM is Active SLAM. This combines the three problems: exploration, localization and mapping. The robot plans to find actions that reduce uncertainty both in localization and mapping. Balancing exploration and localization, the paper [46] describe exploration trajectories that optimize the mean uncertainty in order to build more accurate maps. In paper [47], active sensing with a particle filter, clustering the particles into groups and calculating the total expected entropy for the particle filter by a weighted average of the expected entropy for each group is done. In [47], active SLAM including active loop closure is presented. The SLAM algorithms applied in Active SLAM papers are investigated. The information of exploration parts are discarded.

## 2.4 Scan Matching and SLAM Applied in Map Merging Algorithms:

Map merging constitutes building a consistent map of the environment from the data obtained from different robots. If the initial locations of the robots are known it is easier, which is an extension of single robot mapping [43] [44]. When the robot does not know its position, the problem becomes more difficult. Since this time the robot's traces are need to be connected. This is one of the hardest problems in mapping, which is known as loop closure. For this purpose different SLAM methods are applied. According to [40], these methods are scan matching SLAM [35], grid based FastSLAM [45], Rao-Blackwellized mapping [45]. These methods are overlaps with the research area of us about SLAM techniques. Since our main intend to improve localization is totally connected with mapping. Since an improvement in localization is resulted with an improvement in mapping.

In some papers some solutions combining SLAM with scan matching is given. In [39], a method called as polar scan matching is presented. This method combines scan matching and EKF-SLAM. A raw LADAR data based scan matching is implemented and it is used just for matching of landmarks. Polar Scan Matching is applied for map merging in [41][42]. In [40], the method given in [34] is implemented for the SLAM process of each robot before map merging. In [35], a scan matching method based on ICP algorithm is merged with FastSLAM algorithm. In this paper, the decrease in computational complexity, data association capability in larger scale environments is successfully presented. However it has some deficiencies. It uses an iterative scan matching algorithm, which has chance to converge to local minima. Also it does not have a fixed convergence time. In FastSLAM part, the algorithm is applied in standard way; no special improvement is added to FastSLAM. In spite of these, it gives a great localization and mapping capability, which promotes people to walk through the scope of this paper.

**Summary:** In this chapter, literature survey about feature extraction, scan matching and SLAM algorithms is presented. This survey is based on the literature about methods and algorithms mainly developed on laser scanner information. In feature extraction part, literature about the line extraction algorithms is presented with a comparison table. In scan matching part, survey about feature based and raw data based scan matching techniques is presented. Finally in SLAM algorithms part, all the researches in simultaneous localization and mapping area are presented. In next chapter; theoretical and practical information about the implemented feature based scan matching algorithm in this thesis will be given.

# CHAPTER 3

# SCAN MATCHING

According to the framework given in literature survey, it is seen that feature based scan matching algorithms has lower computational complexity and has advance in real time working applications than point based ones because of their non-iterative approaches. Also, in this thesis, features extracted in scan matching also constitute the input set of intended deterministic global localization (DGL) and FastSLAM algorithms. For this purpose feature based scan matching method is chosen.

In this thesis, the validity of extracted features has a great importance. Since they have been used as an input for all following algorithms given, they affect the accuracy of the whole method. Therefore the comparison between line extraction algorithms is carefully examined, Split and Merge algorithm, which gives the most optimistic results, is implemented.

In main scan matching part, an algorithm calculating the pose difference from the geometrical relations between line segment and corner features is implemented. The pose of the robot is calculated from the incoming pose differences of main scan matching part. The incremental error in pose is eliminated with deterministic global localization (DGL) approach by using the re-observation of landmark features in closed loop cases.

## 3.1 Structure of the Scan Matching Algorithm Implemented

In this part, the structure of the scan matching algorithm will be described. In Figure 3.1, the input output structure is given, in Figure 3.2 the subparts of the general algorithm is given and in Figure 3.3, the detailed structure of algorithm and its subparts is given.

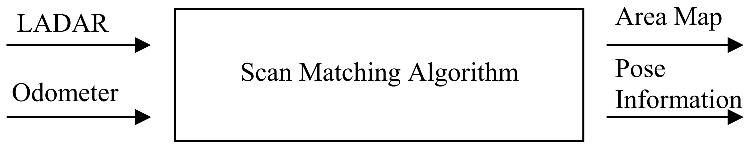


**Figure 3.1:General Input / Output Structure of Scan Matching Algorithm**

The scan matching algorithm accepts laser scanner data (180 range readings incoming from LADAR with 0.5 / 1 degree scan intervals) and odometer data (synchronized estimated pose information of the robot). As an output, it gives the constructed map of the area (as occupancy grid map) and the scan matching based estimated pose information of the robot. Scan matching algorithm consists of four main parts. These are;

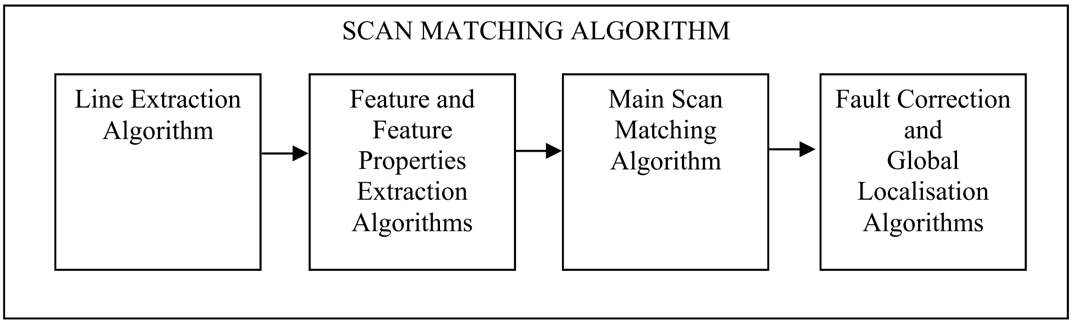

SCAN MATCHING ALGORITHM
Line Extraction Algorithm
Feature and Feature Properties Extraction Algorithms
Main Scan Matching Algorithm
Fault Correction and Global Localisation Algorithms

**Figure 3.2: The main parts of the scan matching algorithm implemented**

In line extraction part, the raw LADAR range scan is clustered and line segments are fitted. In feature and feature properties extraction part, features, such as corners, and feature properties, such as line length, distance between parallel lines, distance between the corners, line end information …etc. are found. These outputs are configured according to the needs of the main scan matching part. In main scan matching part, a matching table is constructed, the line pairs from the previous and current scan are voted, according the overlapping of their feature properties. In fault correction and global localization part, the detected errors occur in the main scan matching part are eliminated. By using global matching techniques in closed loop cases, the error accumulated in the pose is removed. The detailed structure of the scan matching algorithm is given as:

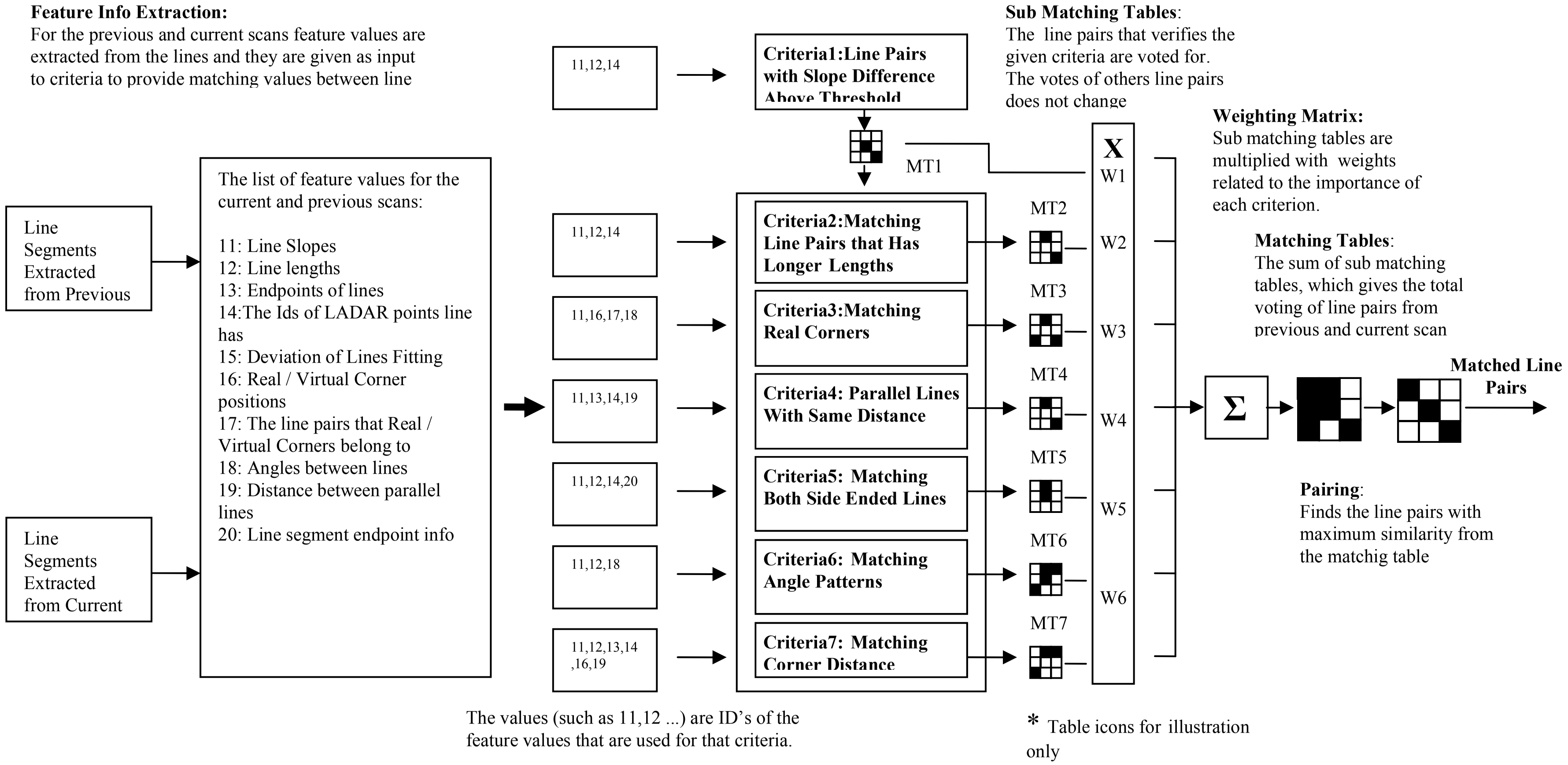

Feature Info Extraction:
For the previous and current scans feature values are extracted from the lines and they are given as input to criteria to provide matching values between line
Line Segments Extracted from Previous
Line Segments Extracted from Current
The list of feature values for the current and previous scans:
11: Line Slopes
12: Line lengths
13: Endpoints of lines
14:The Ids of LADAR points line has
15: Deviation of Lines Fitting
16: Real / Virtual Corner positions
17: The line pairs that Real / Virtual Corners belong to
18: Angles between lines
19: Distance between parallel lines
20: Line segment endpoint info
11,12,14
11,12,14
11,16,17,18
11,13,14,19
11,12,14,20
11,12,18
11,12,13,14 ,16,19
The values (such as 11,12 ...) are ID's of the feature values that are used for that criteria.
Criteria1:Line Pairs with Slope Difference Above Threshold
MT1
Criteria2:Matching Line Pairs that Has Longer Lengths
Criteria3:Matching Real Corners
Criteria4: Parallel Lines With Same Distance
Criteria5: Matching Both Side Ended Lines
Criteria6: Matching Angle Patterns
Criteria7: Matching Corner Distance
MT2
MT3
MT4
MT5
MT6
MT7
Sub Matching Tables:
The line pairs that verifies the given criteria are voted for. The votes of others line pairs does not change
X
W1
W2
W3
W4
W5
W6
* Table icons for illustration only
Weighting Matrix:
Sub matching tables are multiplied with weights related to the importance of each criterion.
Matching Tables:
The sum of sub matching tables, which gives the total voting of line pairs from previous and current scan
Σ
Matched Line Pairs
Pairing:
Finds the line pairs with maximum similarity from the matchig table

**Figure 3.3: The detailed structure of scan matching algorithm.**

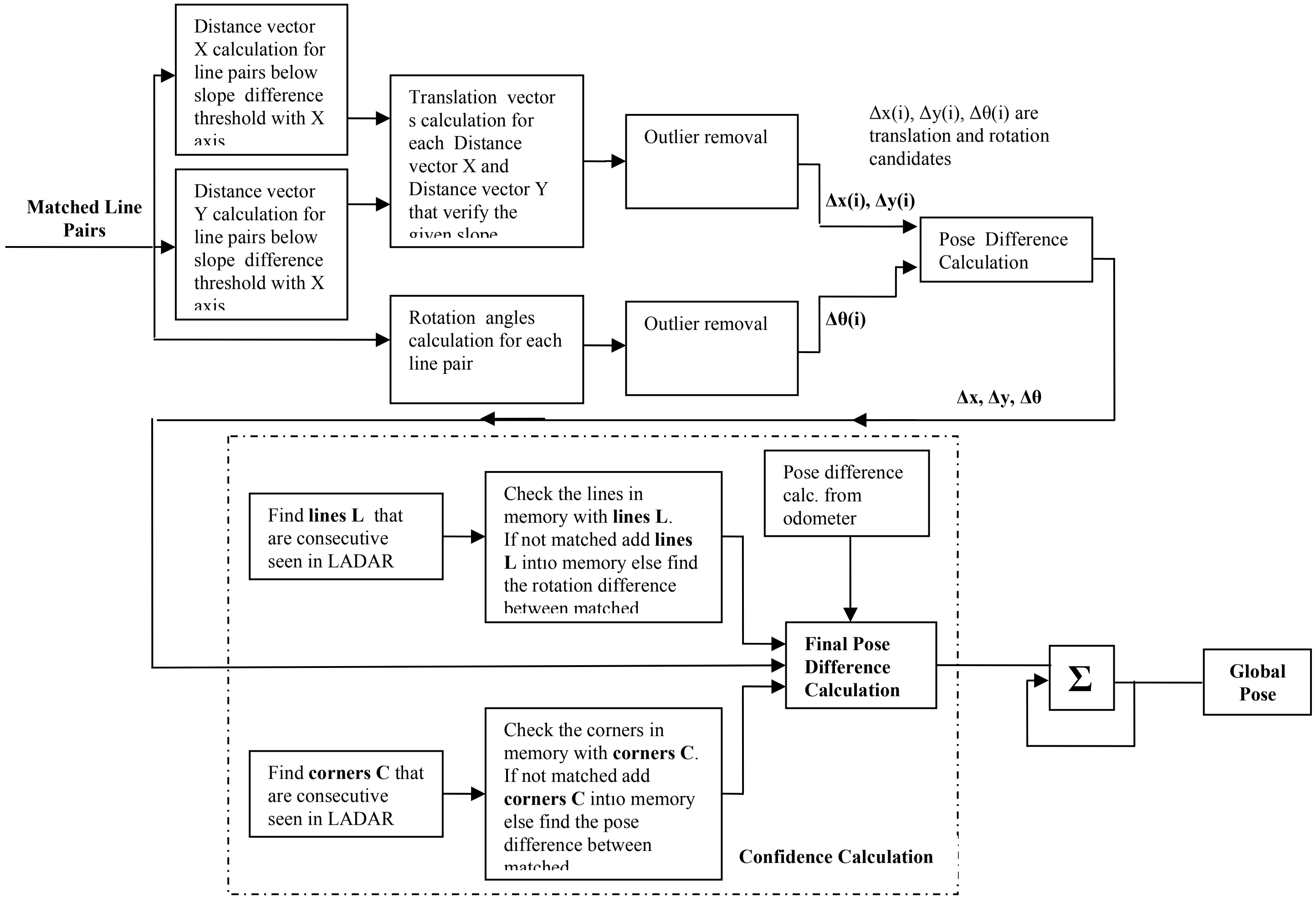

Matched Line Pairs
Distance vector X calculation for line pairs below slope difference threshold with X axis
Distance vector Y calculation for line pairs below slope difference threshold with X axis
Translation vector s calculation for each Distance vector X and Distance vector Y that verify the given slope
Rotation angles calculation for each line pair
Outlier removal
Outlier removal
Δx(i), Δy(i), Δθ(i) are translation and rotation candidates
Δx(i), Δy(i)
Δθ(i)
Pose Difference Calculation
Δx, Δy, Δθ
Find lines L that are consecutive seen in LADAR
Check the lines in memory with lines L. If not matched add lines L intıo memory else find the rotation difference between matched
Pose difference calc. from odometer
Final Pose Difference Calculation
Find corners C that are consecutive seen in LADAR
Check the corners in memory with corners C. If not matched add corners C intıo memory else find the pose difference between matched
Confidence Calculation
Σ
Global Pose

**Figure 3.3: The detailed structure of scan matching algorithm (continued)**

## 3.2 LADAR Data

Laser scanner is measurement sensor based on the time of flight principle by sending a laser pulse in a narrow beam toward the object and measuring the time taken by the pulse to be reflected off the target and returned back to the sender. The types of LADAR's used in our tests are given in Figure 3.4.

**Figure 3.4: The SICK LMS200 and OEM1000 laser scanners that the range scans are obtained.**

In our configuration, LADAR scans a 180 degree area (semi circle) with 1 degree intervals. So we obtain 181 range scans with 5 Hz update rate. We can define these scans as $d^{[0]}$, $d^{[1]}$, $d^{[2]}$, …, $d^{[n]}$, $d^{[n+1]}$,… , $d^{[180]}$, where $d^{[n]}$ means the distance obtained between LADAR and reflecting object and n means the angle of the laser mirror while that measurement has obtained. Related illustration is given in Figure 3.5.

**Implementation Notes:**

- The distance obtained from LADAR is in +/- 38 mm accuracy interval.
- Maximum range that can obtain from LADAR is 80 meters. But distance between two consecutive LADAR beam increases with distance, that is given as,

$$l = 2d\tan(\theta) \qquad \textbf{(3-1)}$$

Where we have *l* as the target distance between measurement points at distance *d,* and *θ* is the angular separation between individual beams.

According to Equation 4.1, for a perpendicular object to laser beam, if *θ*=1 degree, for *d*=10 meters, l= 0.35 m, for *d*=20 meters, l= 0.7 m.

- The small changes in roll and pitch of the robot affect the level that LADAR beam reflects in vertical axis (height). So for long ranges these can be reflected from the ground level or from the ceiling. For a LADAR set in 50 centimeters height from ground level, 1 degree change in roll and pitch causes the LADAR beam to reflect from the ground from 28 meters, 2 degree change causes from 14 meters.

Also there is an increase in size of the beam by the distance. In our case, LADAR is set to about 50 centimeters height and maximum +/- 1.5 degree roll and pitch change is observed. By considering all the notes given above, the maximum range for LADAR is set to 12 meters.

- The LADAR output rate is set to 5 Hz. (Maximum available rate for 115200 baud RS232 message read). This update rate is fast enough for us. Since the maximum speed robot could have is in translational 50 cm/s and in rotation 60 degree/s (The scan matching algorithm is tested with diluted LADAR output (1 Hz) and it is seen that no performance loss is seen about 100 cm translational or 30 degree rotational motions.).

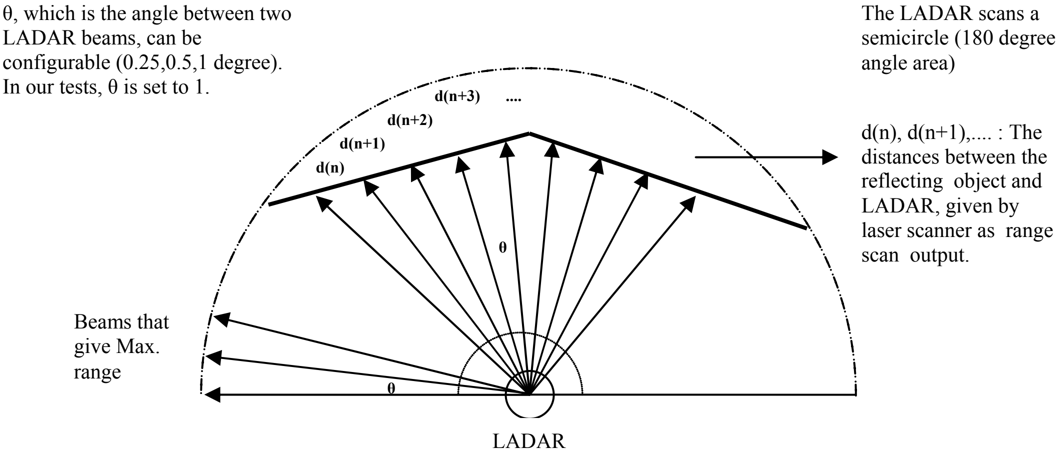

θ, which is the angle between two LADAR beams, can be configurable (0.25,0.5,1 degree). In our tests, θ is set to 1.
The LADAR scans a semicircle (180 degree angle area)
d(n+3)
....
d(n+2)
d(n+1)
d(n)
d(n), d(n+1),.... : The distances between the reflecting object and LADAR, given by laser scanner as range scan output.
θ
Beams that give Max. range
θ
LADAR

**Figure 3.5: LADAR working principles. LADAR has uniformly separated beams spanning a certain angular range (180 degree for our case).**

**The Effect of Robot Motion to LADAR Scans:** The LADAR mirror turn rate is set to maximum, to prevent distortions of the objects seen while robot rotates. This is 20 Hz for Sick OEM 1000 LADAR, which is used for the tests in ASELSAN indoor area and the scanning frequency is 75 Hz for Sick LMS 200 LADAR which is used in test area Ancona and test area Intel Research Laboratory. It can be said as for 20 Hz, the time passed for a 180 degree scan is about 1/40 second. Also for 75 Hz one, this is about 1/150 second. For all these cases, the maximum rotation speed of the robot is set as maximum 60 degree/second. For the other test areas, the datasets are investigated and a turn rate more than 60 degree/second could not be found. So it can be said for LMS 200, the error in orientation of the lines from start of the scan to end is about 0.4 degree, for OEM 1000 1.5 degree. In fact for the continuous turns, the matched line pairs in consecutive scans are about the same rotation angles of the LADAR, so same error is on the matched line pairs. For example the ones close end of the scan is matched with the ones close to the end of the scan. Same is valid for the ones close to start point. As a result, same distortion is seen and this does not effect scan matching, except the turns including acceleration. However for these ones the effect is still much smaller with the respect to one between startpoint of scan and endpoint.

## 3.3 Line Extraction: Split and Merge Algorithm

Line extraction is a vital part for the feature based scan matching algorithms. In fact, it gives raw line segments for the construction of high level features in scan matching. Split and merge algorithm is one of the most preferred line extraction algorithms. It is chosen because of its high performance between line extraction algorithms given in comparison (in Table 2.1).

### 3.3.1 The Structure of Split and Merge Algorithm

As the line extraction algorithm, split and merge technique is used. The illustration giving the structure of Split and Merge algorithm is given in Figure 3.6.

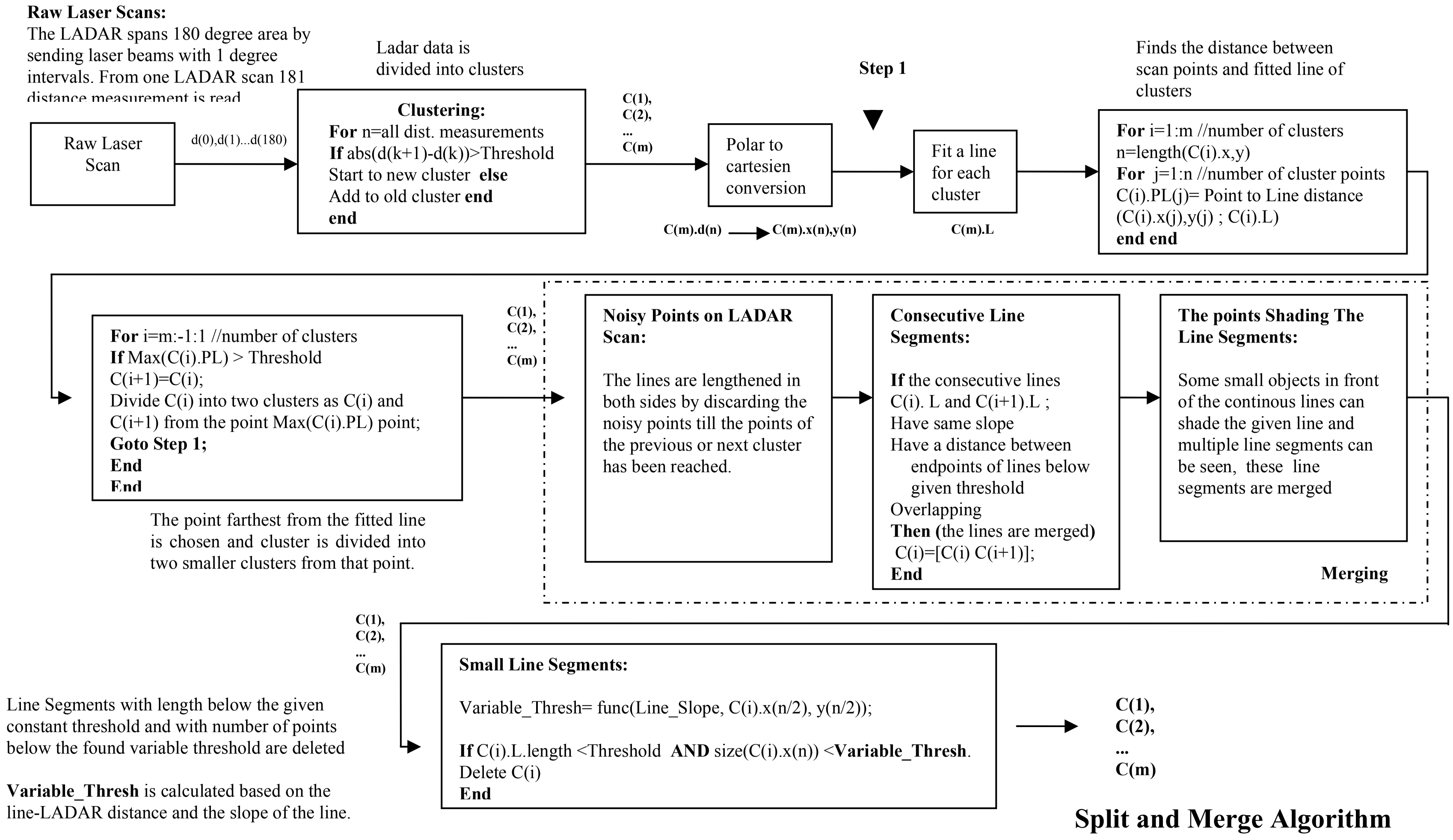

Raw Laser Scans:
The LADAR spans 180 degree area by sending laser beams with 1 degree intervals. From one LADAR scan 181 distance measurement is read
Raw Laser Scan
d(0),d(1)...d(180)
Ladar data is divided into clusters
Clustering:
For n=all dist. measurements
If abs(d(k+1)-d(k))>Threshold
Start to new cluster else
Add to old cluster end
end
C(1),
C(2),
...
C(m)
Polar to cartesien conversion
C(m).d(n) → C(m).x(n),y(n)
Step 1
Fit a line for each cluster
C(m).L
Finds the distance between scan points and fitted line of clusters
For i=1:m //number of clusters
n=length(C(i).x,y)
For j=1:n //number of cluster points
C(i).PL(j)= Point to Line distance (C(i).x(j),y(j) ; C(i).L)
end end
For i=m:-1:1 //number of clusters
If Max(C(i).PL) > Threshold
C(i+1)=C(i);
Divide C(i) into two clusters as C(i) and C(i+1) from the point Max(C(i).PL) point;
Goto Step 1;
End
End
The point farthest from the fitted line is chosen and cluster is divided into two smaller clusters from that point.
C(1),
C(2),
...
C(m)
Noisy Points on LADAR Scan:
The lines are lengthened in both sides by discarding the noisy points till the points of the previous or next cluster has been reached.
Consecutive Line Segments:
If the consecutive lines
C(i). L and C(i+1).L ;
Have same slope
Have a distance between endpoints of lines below given threshold
Overlapping
Then (the lines are merged)
C(i)=[C(i) C(i+1)];
End
The points Shading The Line Segments:
Some small objects in front of the continous lines can shade the given line and multiple line segments can be seen, these line segments are merged
Merging
C(1),
C(2),
...
C(m)
Line Segments with length below the given constant threshold and with number of points below the found variable threshold are deleted
Variable_Thresh is calculated based on the line-LADAR distance and the slope of the line.
Small Line Segments:
Variable_Thresh= func(Line_Slope, C(i).x(n/2), y(n/2));
If C(i).L.length <Threshold AND size(C(i).x(n)) <Variable_Thresh.
Delete C(i)
End
C(1),
C(2),
...
C(m)
Split and Merge Algorithm

Figure 3.6: General structure of Split and Merge

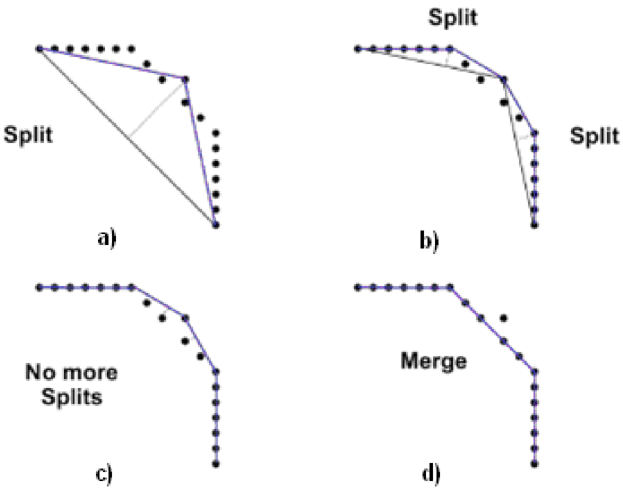


**Figure 3.7: The figure illustrating the steps of Split and Merge Algorithm [17].**

In Figure 3.7, four images give the steps of Split and Merge Algorithm.

- In *image a*, a line is fitted to the cluster and the point that is with maximum distance to fitting line is found, which is above the threshold parameter, and the cluster is divided into two from that point.
- In *image b*, again the points that are with maximum distance to fitting lines are found and the clusters are divided into two from these points.
- In *image c*, no more splits are found, since all the points belong to the cluster are below the threshold parameter.
- In *image d*, the consecutive clusters that confirm the given conditions (given in Step 4) are merged.

A range scan describes a 2D slice of the environment. Points of a range scan are specified in polar coordinate system (Figure 3.5) whose origin is the location of the sensor. It is common in literature to assume that the noise on range measurement follows a Gaussian distribution with zero mean, a range variance and negligible angular uncertainty.

A line in polar coordinates is expressed as:

$$\begin{aligned} &x_i \cos(a) + y_i \cos(a) = r \\ &(x_i, y_i) \in L \end{aligned} \quad \textbf{(3-2)}$$

where $-\pi < a <= \pi$ is the angle between the $x$ axis and the normal of the line, r >= 0 is the perpendicular distance of the line to the origin; *(x, y)* are the Cartesian coordinates of a point lying on the line. *L* is the point set that constructs the line.

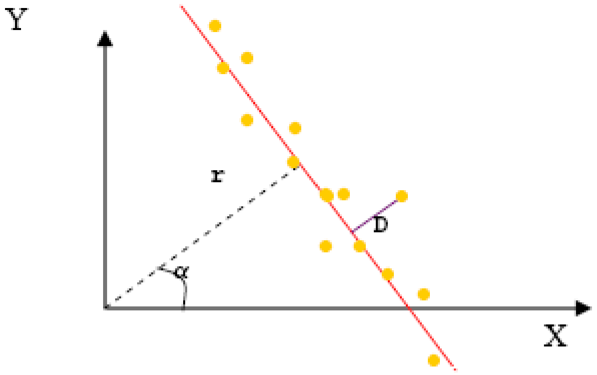


**Figure 3.8: Fitting line parameters:** ***D*** **is the fitting error we aim to minimize expressing a line with polar parameters (*r,α*).**

The outline for the implemented algorithm based on LADAR data is given below:

**Step 1:** Create clusters from the given LADAR data according to given cluster threshold parameter. In this step, the LADAR scans from *d(0), ... d(180)* are controlled whether

$$(d^{[n+1]} - d^{[n]}) > Threshold \qquad \textbf{(3-3)}$$

Clusters smaller than minimum cluster points are removed from the cluster set.

**Step 2:** For each cluster; a line is fitted according to formula given below:

$$D^2 = \sum_{i=1}^{m}(r - C_x(i)\cos(a) - C_y(i)\sin(a)) \qquad \textbf{(3-4)}$$

Where *m* is the total number points in the cluster, $C_x(i)$ is the x coordinates of the cluster elements, $C_y(i)$ is the y coordinates of the cluster elements. *α* is the line angle and *r* is the line radius, which is the distance between fitting line and origin, given in Figure 3.8.

The solution of *(r, α)* can be found imposing the derivatives of equation by *(r, α)*:

$$\frac{\partial D^2}{\partial r} = 0 \qquad \frac{\partial D^2}{\partial a} = 0 \tag{3-5}$$

That is:

$$x_{mean} = \frac{1}{m}\sum_{i=1}^{m} C_x(i) \tag{3-6}$$

$$y_{mean} = \frac{1}{m}\sum_{i=1}^{m} C_y(i) \tag{3-7}$$

$$a = 0.5\tan^{-1}\left( \frac{-2\sum_{i=1}^{m}((C_x(i) - x_{mean})(C_y(i) - y_{mean}))}{\sum_{i=1}^{m}(C_y(i) - y_{mean})^2 - \sum_{i=1}^{m}(C_x(i) - x_{mean})^2} \right) \tag{3-8}$$

$$r = \cos(a)x_{mean} + \sin(a)y_{mean} \tag{3-9}$$

**Step 3:** If there are points with a distance to fitting line above the threshold, the point farthest from the fitting line is chosen and cluster is divided into two smaller clusters from that point, then pass to Step 2, else pass Step 4.

**Step 4:** Lines are merged according to given rules below:

- For each line, points continued from the end of the line are controlled whether they fit the line parameters. If there are some points confirm the specified properties, they are added to line set, and the line parameters of the line set are calculated again. Two main factors are controlled for this process.

i. If there are points further from the laser scanner with respect to fitting line, this means these points belong to a different structure. Therefore the line ends without adding these points.

ii. If there are points nearer to the laser scanner with respect to fitting line, these lines prevent us to see additional points of fitting line. If number of preventing points is below a threshold, the fitting line is lengthened.

- If there are two following lines with same slope these lines are controlled whether they can be belong to same structure. If these lines are decided to be in the same structure they are merged.

- If there are lines that have length and number of points less than the defined thresholds, these lines are deleted. Two parameters of the threshold are defined according to three main factors given below. These are:

    i. If a line fitting is far away from the laser scanner it contains less points, however the distance between these points increases and the length of the line increases.

    ii. If slope of the line is close to perpendicular to the laser scanner rays that it intersects, then the points of the fitting line become much more further from each other. Therefore while number of points decreases the length increases.

    iii. If the fitting line is so close to the laser scanner, number of points increases but length of the line decreases.

- The lines between two following parallel lines are specially considered. If they contain small number of points, then the slope of these can easily affected from the noisy character of the LADAR data. These lines are important to find real corners. So any change in their slope affects the position of the corner. Therefore they are corrected as they have best slope estimate.

## 3.4 Feature and Feature Properties Extraction Algorithms

In this part, from the line segments coming from the line extraction part, features are extracted. These features are basically line segments and corners, but also combination of these are also used, such as parallel line segments, patterns formed from consecutive line series, patterns formed from consecutive corner series... Also the properties are extracted from this feature set, such as slopes, line lengths, corner distances.... All these are used as inputs for the matching process in main scan matching part and global localization part.

### 3.4.1 Line Information Extraction

Following feature properties are extracted from each line segment found by using Split and Merge algorithm:

1) The slope of line is found. This slope is given as the angular difference of line direction with start point which is between -180 and +180 in class element $C^{[m]}.L.Slope$.

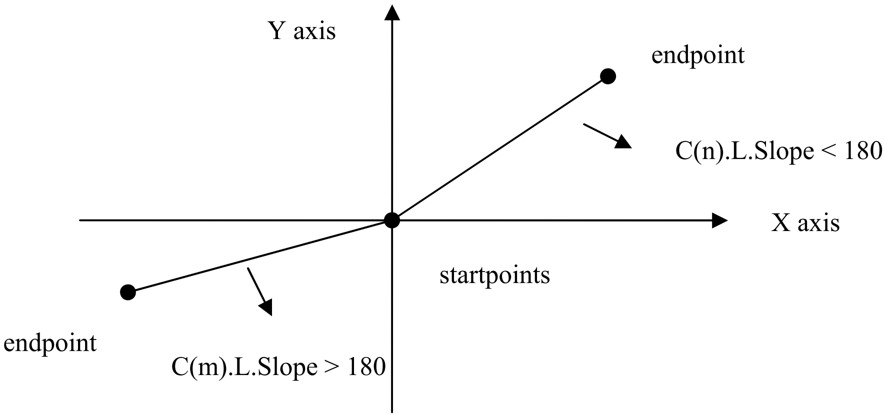


**Figure 3.9: The figure illustrating the slope of a line.** $C.L.Slope$

2) The Cartesian coordinates of fitting line endpoints are calculated ( $C^{[m]}.x, y$ ).

3) The length of line is found which the distance between the endpoints is ( $C^{[m]}.L.Length$ ).

4) ID's of points in the cluster are found which is defined as $\mathrm{C}^{[m]}.ID^{[1:n]}$.They are $\theta^{[i]}$, the angle of the point in polar coordinates given in range scan raw output $(\mathrm{d}^{[i]}, \theta^{[i]})$.

5) If the points following an endpoint of the fitting line are further away from the laser scanner than the line endpoint

$$\begin{aligned} &\mathrm{d}^{[i]} > \mathrm{d}^{[j]} \\ &j = \mathrm{C}^{[m]}.ID^{[endpoint]} \\ &i = j+1 \; or \; j-1 \; and \; j \notin \mathrm{C}^{[m]}.ID \end{aligned} \qquad \textbf{(3-10)}$$

or the points following are close to the endpoint,

$$\left|d^{[i]} - d^{[j]}\right| < Threshold$$
$$j = C^{[m]}.ID^{[endpo\,int]} \qquad \textbf{(3-11)}$$
$$i = j + 1 \; or \; j - 1 \; and \; j \notin C^{[m]}.ID$$

than this line is ended in that endpoint. The lines are classified as, open, right side ended, left side ended, both side ended.

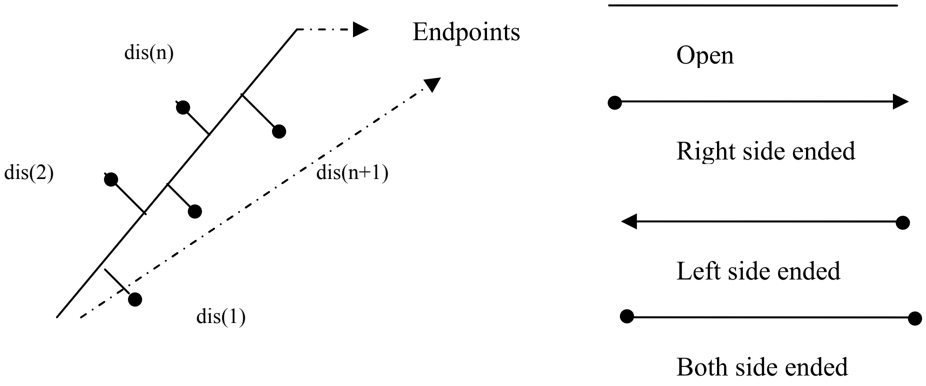


**Figure 3.10: The illustration about the definitions given in line information extraction.**

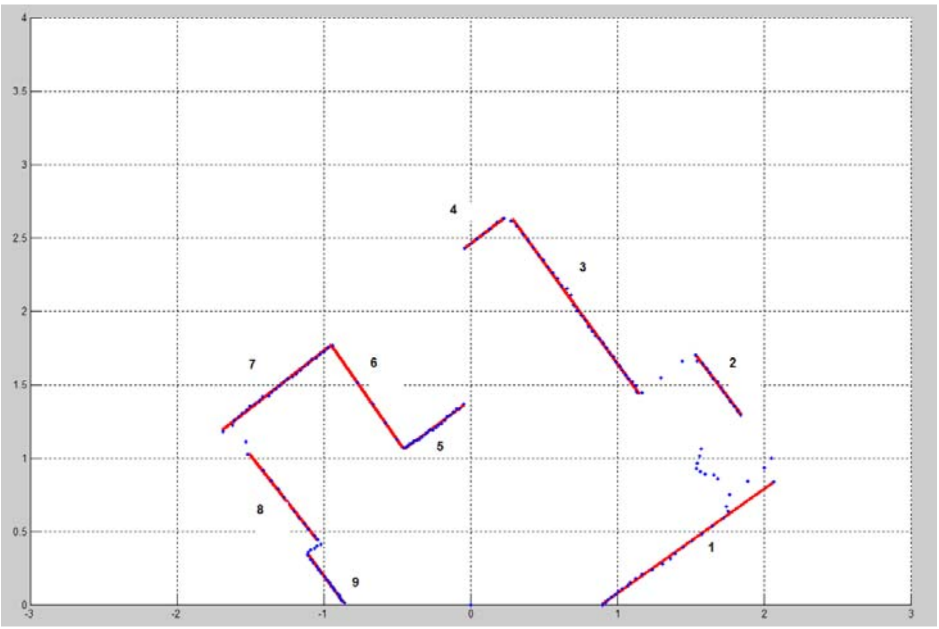


**Figure 3.11: The lines extracted from the LADAR data.**

When we look at Figure 3.11:

a) We can define Line1 as open. We do not know right side of the line, since it is at the end of our scan range, it can be open or ended. At the left side of the line, after the endpoint there are points closer to LADAR, which means this line can proceed behind these points so this endpoint is also open.

b) We can define Line 2 as left side ended. There are points closer to LADAR than the right endpoint of the line. At left side, there are points closer to LADAR than the left endpoint, but they start from a position so close to line end. Therefore the line could not proceed behind these points.

c) We can define Line 3 as both side ended. Since the points in right side are further away from LADAR then the right endpoint, and the ones in right side is in position so close to the left line end.

..................

The variable, appointed for line classification information, is given as $C^{[m]}.L.LineClass$.

6) Deviation of the points for the fitting line is calculated. Since the mean is the position of the fitting line so $\sigma(C^{[m]})$ becomes square root of point to fitting line distances.

$$\sigma(C^{[m]}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\mathrm{C}^{[\mathrm{m}]}.L.dis^{[i]})^2} \qquad \textbf{(3-12)}$$

Where $\mathrm{C}^{[\mathrm{m}]}.L.dis^{[n]}$ the distance between the *i*th cluster point and fitting line is, $C^{[m]}$ is the cluster and *m* is the ID of the cluster.

### 3.4.2 Corner Information Extraction

1) The intersection points of the lines are found. These intersection points are defined as real corners and virtual corners.

- **Real corner** is defined as the intersection point of two following lines with too close endpoints.
- **Virtual corner** defined as the intersection point of the extension of two lines.

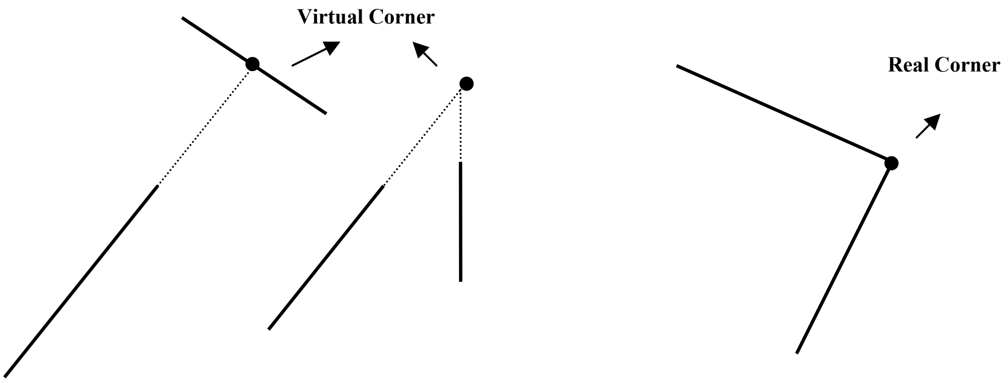


**Figure 3.12: The illustration that shows how Virtual /Real Corners are defined.**

The corners are defined by $Cnr^{[m]}$ class. $Cnr^{[m]}.Class$ corresponds the information about whether the corner is real or virtual.

2) The angles between the lines constructing the corner are found. *Cnr(m).Angle* corresponds the angle between constructing lines. Since the slope of lines ($C^{[m]}.L.Slope$) constructing the corner are defined between -180 and 180, than the angle of corner becomes between -180 and 180, which includes information whether the corner is concave or convex inside.

3) The positions of corners in three different categories are found. These are;

- Real/Virtual corners belong to long constructing lines
- All real corners
- All real/virtual corners

Categorization of corners is important since they have different importance in the scan matching algorithm part.. $Cnr^{[m]}.x$ and $Cnr^{[m]}.y$ corresponds the position of the corner.

4) The line pairs that construct the corner are found. $Cnr^{[m]}.Lpair$ corresponds the line pair constructing the corner.

### 3.4.3 Corner Angle and Distance Pattern Extraction

Corner angle patterns are constructed from the slope difference of six consecutive lines, between the first one and other five. Related illustration is given in Figure 3.13.

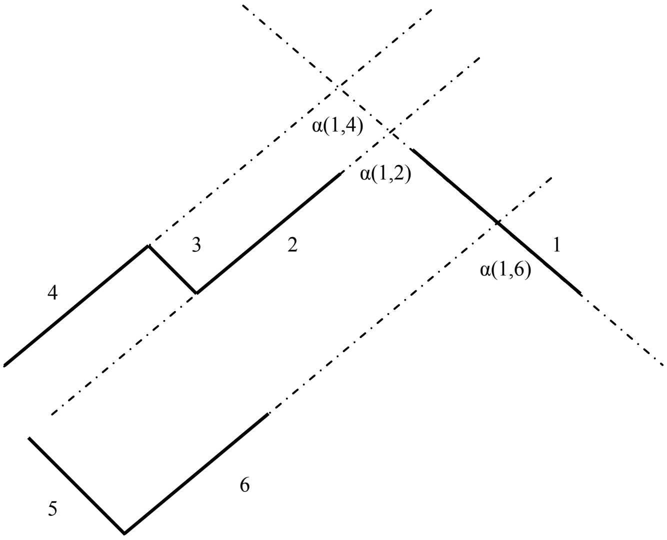


**Figure 3.13: The schema illustrating the angle pattern for the first line**

As you can see from Figure 3.13, the angle pattern consists of the first line and the slope differences between the first line and five consecutive lines. So we can define the pattern *as*;

**Table 3.1: The pseudo code of the algorithm for finding the angle pattern. The pattern consists of the slope differences between the first and consequent five lines.**

```
Algorithm_ Angle Pattern( C)
for each j=1,...,5
    if $C^{[i]} = \phi$ or $C^{[i+j]} = \phi$ then
        $Angle\_P^{[i,j]}=0;$
    else
        $Angle\_P^{[i,j]} = C^{[i]}.L.slope - C^{[i+j]}.L.slope$ //the slope difference
    endif
endfor
$Angle\_P^{[i]} = \left[Angle\_P^{[i,1]},...,Angle\_P^{[i,5]}\right]$
return $Angle\_P^{[i]}$
```

Corner distance patterns are constructed from the intersection point distance of 6 consecutive lines, between the first one and other five. Related illustration is given in Figure 3.14.

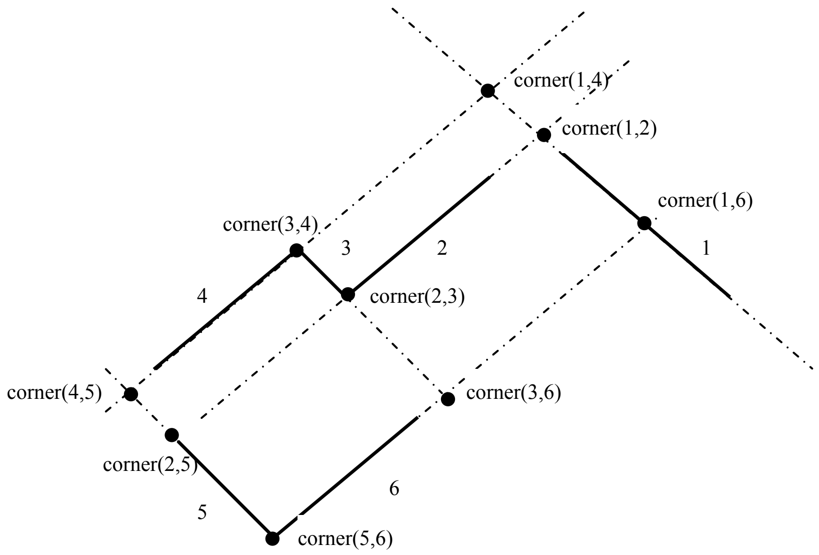


**Figure 3.14: The schema illustrating the corner pattern.**

As you can see from Figure 3.14, the corner distance pattern consists of the first corner and the differences between the first corner and five consecutive corners. So we can define the pattern $cornerDist^{[i]}$ as;

**Table 3.2: The pseudo code defining the corner distance pattern. The pattern consists of the corner distance differences between the first and consequent five lines.**

```
Algorithm_Corner_Distance_Pattern( C)
function corner(L1,L2) // calculates the intersection of line L1 and L2
for each j=1,...,5
    if $C^{[i]} = \phi$ or $C^{[i+j]} = \phi$ then
        $cornerDist\_P^{[i,j]} = 0;$
    else
        $cornerDist\_P^{[i,j]} = (\textbf{corner}(C^{[i]}.L, C^{[i+1]}.L) - \textbf{corner}(C^{[i]}.L, C^{[i+j]}.L))$
    endif
endfor
$cornerDist\_P^{[i]} = \left[cornerDist\_P^{[i,1]},...,cornerDist\_P^{[i,5]}\right]$
return $cornerDist\_P^{[i]}$
```

## 3.5 Main Scan Matching Part

In scan matching part, the features and feature properties extracted from the previous and current scans are compared according to the seven criteria given in next sections. All the feature information extracted in Section 3.4 is coming from the line segments and their clusters from Section 3.3: Line Extraction Part. Therefore any overlap between feature properties shows the overlap between the features they have been belonging. In fact, these features again shows the overlap between the line segments they have been constructed. As a result, each comparison made by given criteria in Sections 3.5.2 and 3.5.8 turns as a vote for the related line segments given in matching table in Section 3.5.1. From the votes in the matching table predicted line segment matches are found and the resulting pose difference in translation and rotation between the consecutive scans is found.

### 3.5.1 Matching Table

While doing feature based scan matching, a matching table has constructed. The columns of this table consist of the line segments of the previous scan and the rows of this table consist of the line segments of current scan. Below a matching table sample is given in Table 3.4 during the matching of two scans taken from Radish data [22].

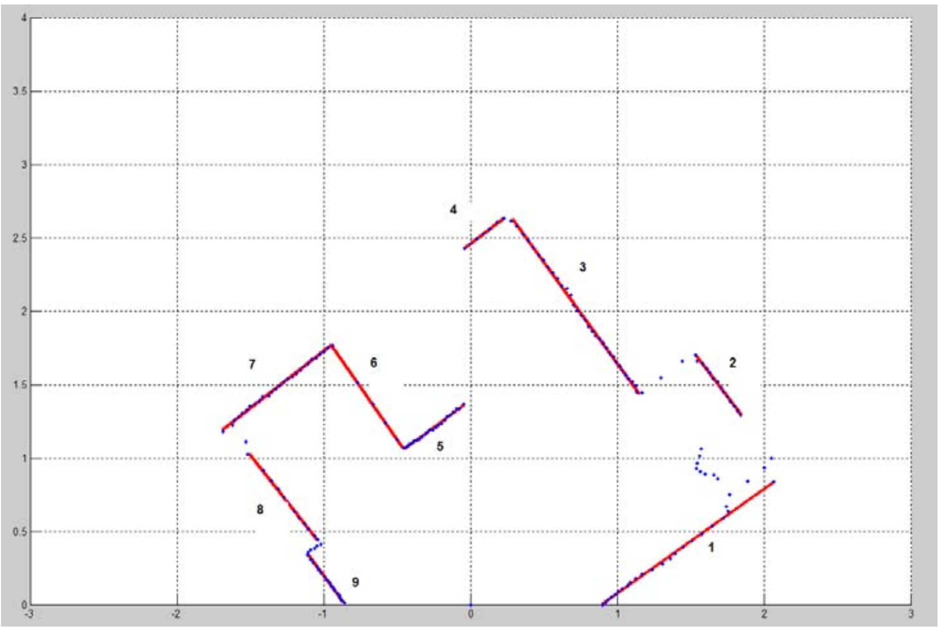


**Figure 3.15: Previous LADAR scan and fitting lines on the LADAR data.**

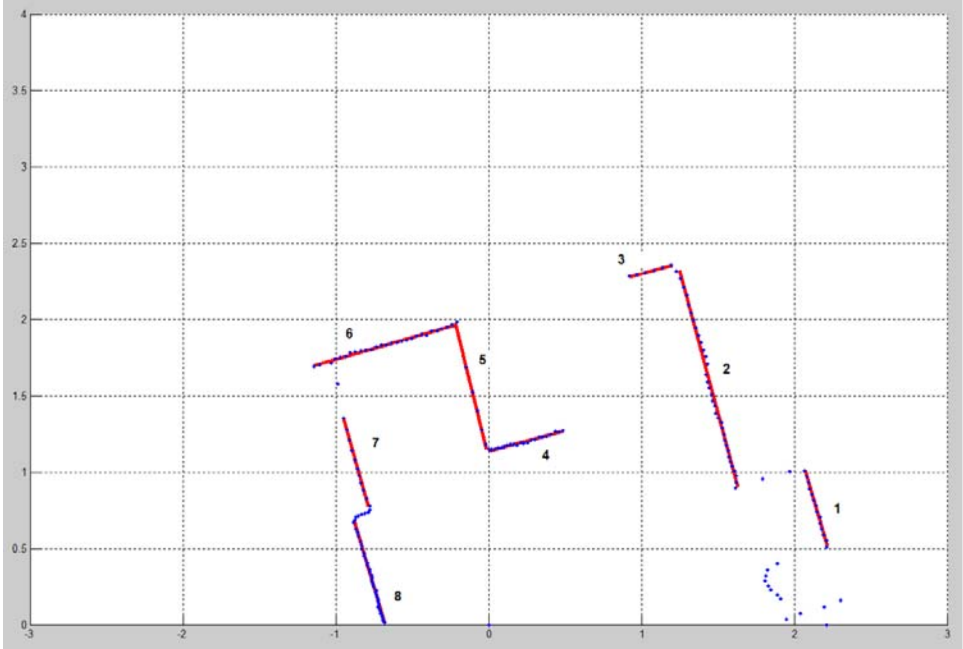


**Figure 3.16: Current LADAR scan and fitting lines on the LADAR data.**

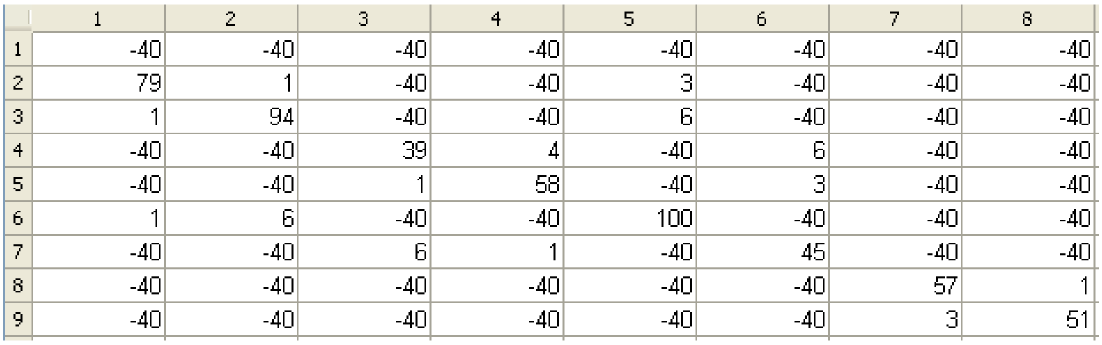
**Table 3.3: Matching table. This table consists from the rows that are line segments of previous scan and columns that are line segments of current scan.**

$$MT = \begin{bmatrix} lp(1,1) & lp(1,2) & lp(1,3) & \dots & \dots & lp(1,n) \\ lp(2,1) & lp(2,2) & lp(2,3) & \dots & \dots & lp(2,n) \\ lp(3,1) & lp(3,2) & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & lp(m-1,n-1) & lp(m-1,n) \\ lp(m,1) & lp(m,2) & \dots & \dots & lp(m,n-1) & lp(m,n) \end{bmatrix}$$

where *lp(m,n)* is the matching score between line pair $C_{prev}{}^{[m]}.L$ and $C_{curr}{}^{[n]}.L$. *m* and *n* are the ID of the cluster extracted from the previous and current LADAR scans.

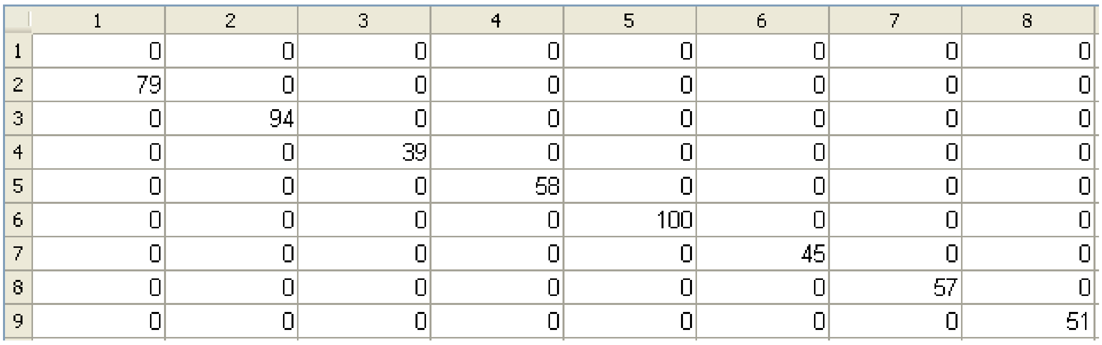
As you can see from Table 3.3; matching table shows the voting's of the line pairs. The columns show extracted lines of current scan and the rows show extracted lines of previous scan. So the number set in *MT(m,n)*, corresponds the voting (matching score) for the line of cluster *m* from the previous scan and line of cluster *n* from current scan.

**Table 3.4: A sample matching table and the votes between the fitting lines of previous scan and current scan.**



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | -40 | -40 | -40 | -40 | -40 | -40 | -40 | -40 |
| 2 | 79 | 1 | -40 | -40 | 3 | -40 | -40 | -40 |
| 3 | 1 | 94 | -40 | -40 | 6 | -40 | -40 | -40 |
| 4 | -40 | -40 | 39 | 4 | -40 | 6 | -40 | -40 |
| 5 | -40 | -40 | 1 | 58 | -40 | 3 | -40 | -40 |
| 6 | 1 | 6 | -40 | -40 | 100 | -40 | -40 | -40 |
| 7 | -40 | -40 | 6 | 1 | -40 | 45 | -40 | -40 |
| 8 | -40 | -40 | -40 | -40 | -40 | -40 | 57 | 1 |
| 9 | -40 | -40 | -40 | -40 | -40 | -40 | 3 | 51 |

**Table 3.5: Resulting matching table after the line matches with highest rankings are chosen and the others are set to zero.**



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 79 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 94 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 39 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 58 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 100 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 45 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 57 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 51 |

In Table 3.5, the resulting matching table is presented after the line matches with highest rankings are chosen and the others are set to zero. Line 1 from current scan ($C_{curr}^{[1]}.L$ ) is matched with Line 2 from previous scan ($C_{prev}^{[2]}.L$ ), Line 2 from current scan is matched with Line 3 from previous scan and it goes alike. As we can see from the figures given above, from the two following scans the fitting lines are found and they are voted according to the voting criteria given in next part. The votes coming from voting criteria are weighted according to their importance and summed to construct the matching table. The maximum voted line pairs in matching table are found and these line pairs will be used for the calculation of pose difference. Voting criteria's are given below:

### 3.5.2 Criterion 1: Elimination of Line Pairs with Slope Difference above Threshold

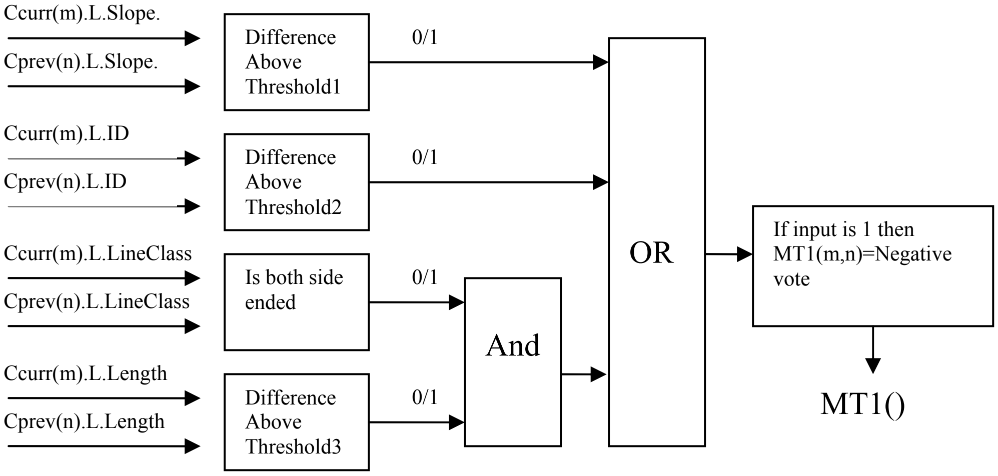


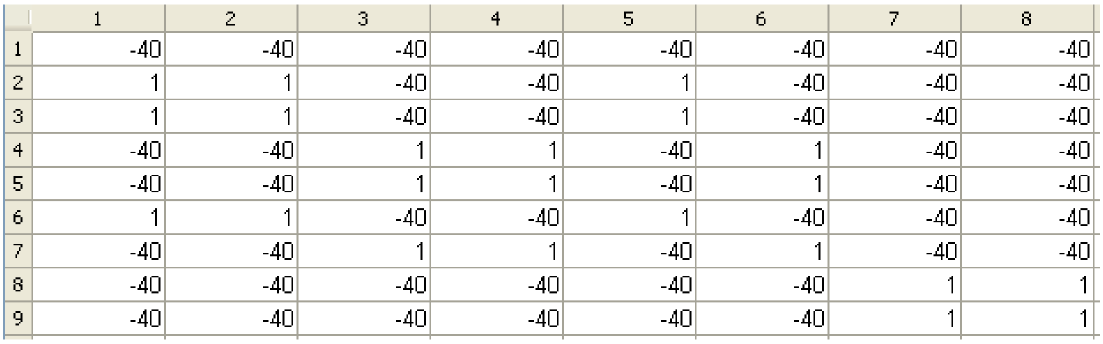
**Figure 3.17: The figure illustrating the scheme of Criterion 1 implemented.**

At the beginning of scan matching, the line pairs from the following scans are checked whether their slope difference are below the threshold, which is given as parametric and set in configuration file. Deciding this threshold is important, since it means that whether the robot turns in *q* degrees between the following scans. If the slope of line in previous scan is *Q* degrees, so the matched line in next scan has a slope between *Q-q* and *Q+q*. This means all the line pairs that do not ensure this interval can be considered as mismatching pairs.

This threshold is decided according to turn limits of the robot between two LADAR scans. From the Radish data [22] taken it is seem that robot has a turn rate less than 60 degree per LADAR sample. Therefore it is set to 60 degrees. For any test, it needs to be set according to LADAR scanning frequency and the robot turning rate.

The negatively voted line pairs feed the other criteria. Therefore any process in other criteria for a line pair that has no chance to match is prevented, which results with a considerable decrease in the computational complexity of the algorithm.

**Table 3.6: Sub matching table (*MT1*) of Criterion 1. The line pairs that do not assure the given criteria are negatively voted with weighting.**



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | -40 | -40 | -40 | -40 | -40 | -40 | -40 | -40 |
| 2 | 1 | 1 | -40 | -40 | 1 | -40 | -40 | -40 |
| 3 | 1 | 1 | -40 | -40 | 1 | -40 | -40 | -40 |
| 4 | -40 | -40 | 1 | 1 | -40 | 1 | -40 | -40 |
| 5 | -40 | -40 | 1 | 1 | -40 | 1 | -40 | -40 |
| 6 | 1 | 1 | -40 | -40 | 1 | -40 | -40 | -40 |
| 7 | -40 | -40 | 1 | 1 | -40 | 1 | -40 | -40 |
| 8 | -40 | -40 | -40 | -40 | -40 | -40 | 1 | 1 |
| 9 | -40 | -40 | -40 | -40 | -40 | -40 | 1 | 1 |

In Table 3.6, the resulting sub matching table is shown when the Criterion 1 is applied onto the LADAR range scan given in Figure 3.15 and Figure 3.16. Negatively voted line pairs are not fed to Criterion 2, 3, 4, 5, 6, 7. This prevents the algorithm not to work on with the non-matching line pairs, which decreases computational complexity.

### 3.5.3 Criterion 2: For Line Pairs That Has Longer Lengths

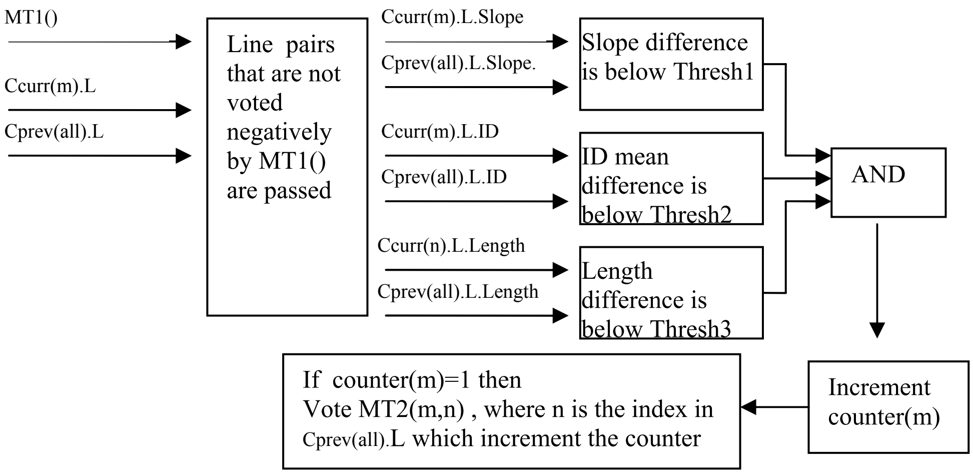

MT1()
Ccurr(m).L
Cprev(all).L
Line pairs that are not voted negatively by MT1() are passed
Ccurr(m).L.Slope
Cprev(all).L.Slope.
Ccurr(m).L.ID
Cprev(all).L.ID
Ccurr(n).L.Length
Cprev(all).L.Length
Slope difference is below Thresh1
ID mean difference is below Thresh2
Length difference is below Thresh3
AND
Increment counter(m)
If counter(m)=1 then
Vote MT2(m,n) , where n is the index in Cprev(all).L which increment the counter

**Figure 3.18: Sub matching table (*MT2*) of Criterion 2. The figure illustrating the scheme of Criterion 2 implemented.**

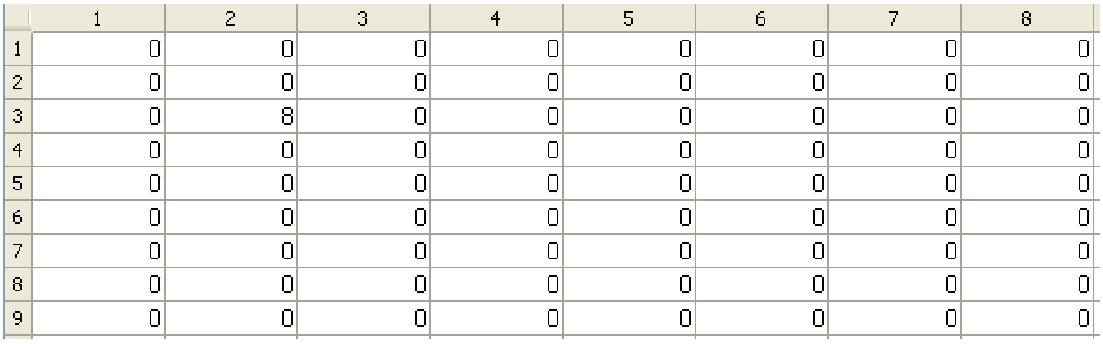
This criterion adds extra voting for line pairs that has longer line lengths. In fact, it helps matching in case of the area has large line segments and the robot has sharp turns.

Especially, this case is seen while robot is going on long corridors and makes sharp turns. In fact, these areas are low structured, instead of complex criteria that are based on matching multiple features or feature properties, simple criteria can be advantageous. Since low numbered line cannot be applicable to a complex criteria. Also checking a few simple but decisive properties can be enough, as like the given criterion.

In a scan pair, if there is a longer line than the others and the same lines are seems to be seen in the next scan, this criterion highly votes them, but if these lines are more than one and so close to each other, this criterion does not vote the line pairs to prevent mismatches. The sub criteria for this are:

- Line lengths,
- Number of longer lines,
- The slope and  midpoint change between the longer lines in the same scan

**Table 3.7: Sub matching table (*MT2*) of Criterion 2. The line pairs that supply the Criterion 2 are voted with weighting.**



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

In Table 3.7, the resulting sub matching table is shown when the Criterion 2 is applied onto the LADAR range scan given in Figure 3.15 and Figure 3.16. $C_{prev}^{[3]}.L$ and $C_{curr}^{[2]}.L$ has considerable longer line length compared with other lines with same slope in their neighborhood. Therefore this line pair is seems to be unique and highly voted in *MT2*.

### 3.5.4 Criterion 3: For the Lines that Have Real Corners

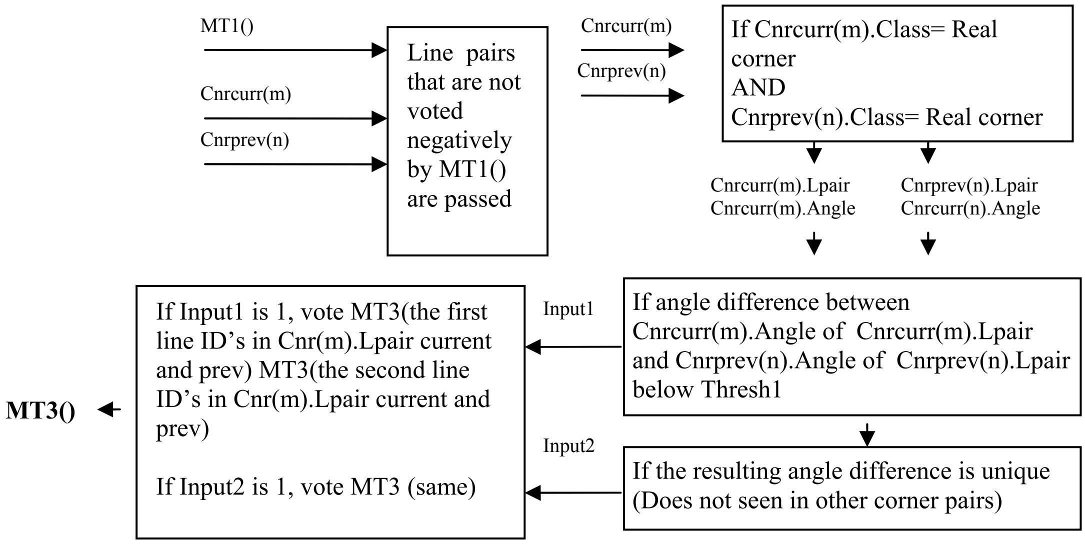

MT1()
Cnrcurr(m)
Cnrprev(n)
Line pairs that are not voted negatively by MT1() are passed
Cnrcurr(m)
Cnrprev(n)
If Cnrcurr(m).Class= Real corner AND Cnrprev(n).Class= Real corner
Cnrcurr(m).Lpair
Cnrcurr(m).Angle
Cnrprev(n).Lpair
Cnrcurr(n).Angle
If angle difference between Cnrcurr(m).Angle of Cnrcurr(m).Lpair and Cnrprev(n).Angle of Cnrprev(n).Lpair below Thresh1
If the resulting angle difference is unique (Does not seen in other corner pairs)
Input1
Input2
If Input1 is 1, vote MT3(the first line ID's in Cnr(m).Lpair current and prev) MT3(the second line ID's in Cnr(m).Lpair current and prev)
If Input2 is 1, vote MT3 (same)
MT3()

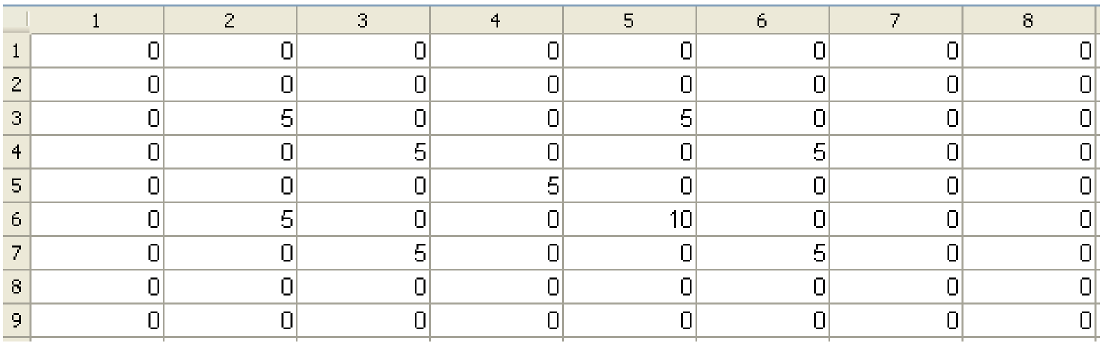
**Figure 3.19: The figure illustrating the scheme of Criterion 3 implemented.**

This criterion adds voting to matching table whether the real corners of following scans match. The rules are;

- If the real corners has same angle (with a given tolerance set in the configfile) between the lines constructing the corner in both following scans.
- If the real corners in following scans can have chance to be same if the robot makes a turn in given limits.
- Extra Vote: If the real corner has a unique angle between constructing lines.

Then the line pairs constructing these corners are voted.

**Table 3.8: Sub matching table (*MT3*) of Criterion 3. The line pairs that supply the Criterion 3 are voted with weighting.**



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 5 | 0 | 0 | 5 | 0 | 0 | 0 |
| 4 | 0 | 0 | 5 | 0 | 0 | 5 | 0 | 0 |
| 5 | 0 | 0 | 0 | 5 | 0 | 0 | 0 | 0 |
| 6 | 0 | 5 | 0 | 0 | 10 | 0 | 0 | 0 |
| 7 | 0 | 0 | 5 | 0 | 0 | 5 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

In Table 3.8, the resulting sub matching table is shown when the Criterion 3 is applied onto the LADAR range scan given in Figure 3.15 and Figure 3.16. In the table, the line pairs that verify the given criterion are voted. The weight of these voting is set low, except that a unique match is not seen. In fact, in the angles of real corners ($Cnr^{[i]}.Angle$), multiplicative of 90 degrees are commonly seen, because of the perpendicular construction of features, such as corridors, rooms, furniture… However the power of this criterion comes for low structured environments or environments that has real corner angle that are not multiplicative of 90 degrees. In this type of environments mostly these corner angles are unique, so they are highly voted.

### 3.5.5 Criterion 4: For Parallel Line Distance between the Lines

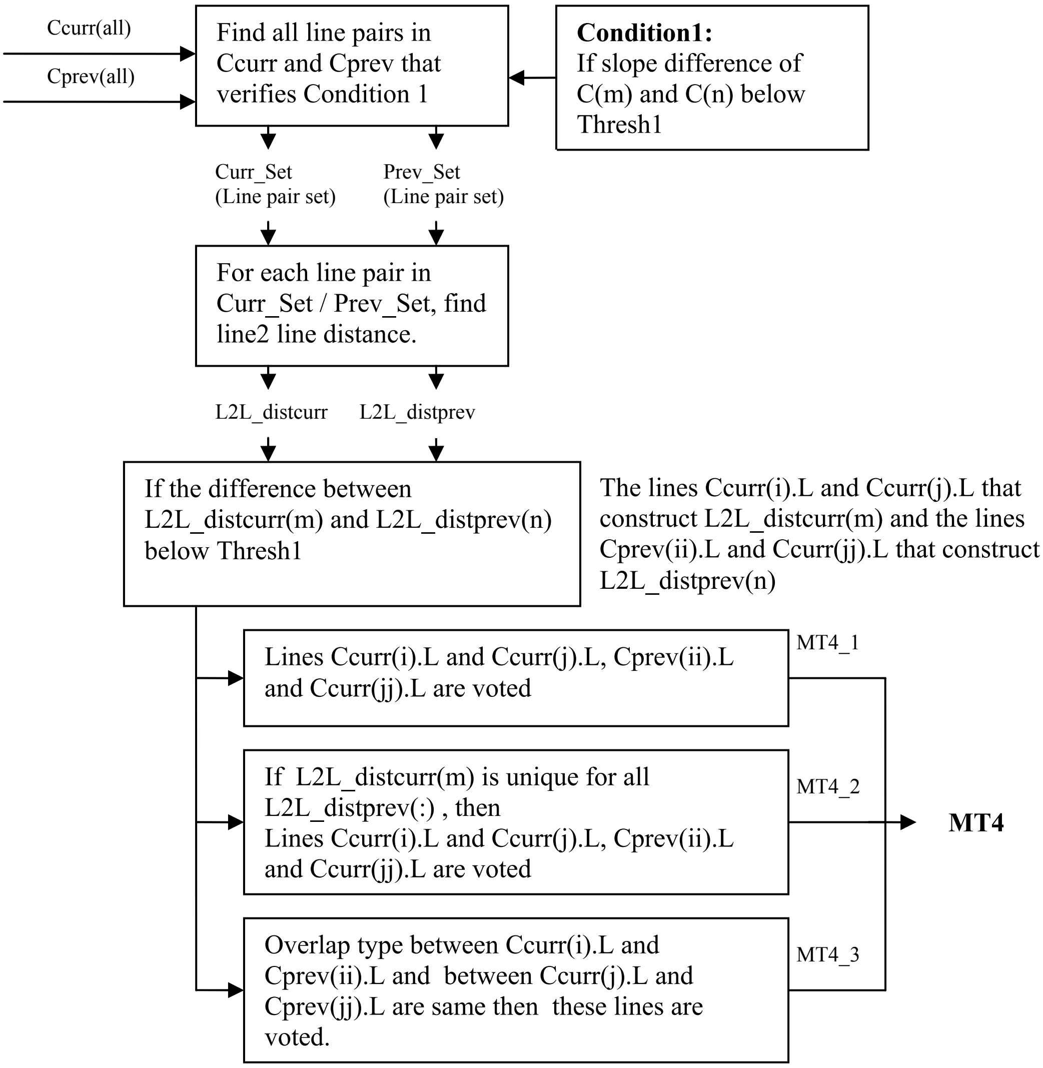

Ccurr(all)
Cprev(all)
Find all line pairs in Ccurr and Cprev that verifies Condition 1
Condition1:
If slope difference of C(m) and C(n) below Thresh1
Curr_Set (Line pair set)
Prev_Set (Line pair set)
For each line pair in Curr_Set / Prev_Set, find line2 line distance.
L2L_distcurr
L2L_distprev
If the difference between L2L_distcurr(m) and L2L_distprev(n) below Thresh1
The lines Ccurr(i).L and Ccurr(j).L that construct L2L_distcurr(m) and the lines Cprev(ii).L and Ccurr(jj).L that construct L2L_distprev(n)
Lines Ccurr(i).L and Ccurr(j).L, Cprev(ii).L and Ccurr(jj).L are voted
MT4_1
If L2L_distcurr(m) is unique for all L2L_distprev(:) , then Lines Ccurr(i).L and Ccurr(j).L, Cprev(ii).L and Ccurr(jj).L are voted
MT4_2
Overlap type between Ccurr(i).L and Cprev(ii).L and between Ccurr(j).L and Cprev(jj).L are same then these lines are voted.
MT4_3
MT4

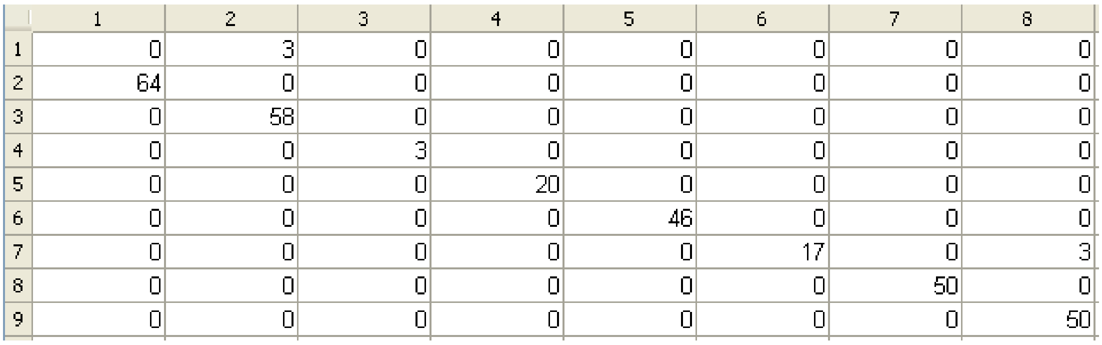
**Figure 3.20: The figure illustrating the scheme of Criterion 4 implemented.**

Distance between parallel lines is an important criterion for scan matching. Since distance between parallel lines does not depend on the pose of the robot. Depending on noise rate of distance measurement sensor (in our case this is laser scanner which has very low noise rate which is below 4 centimeters) and also robustness of the feature extraction algorithm (Since the line fit averages the measurements , error in position of lines will be the mean of the error coming from the measurement sensor readings. Also a good line extraction method needs to find the slope of the line with minimum error), the tolerances for the distances between the lines decrease which make the line distances much more specific and distinctive. The criterion consists of:

- If the distance between the line pairs in following scans are same (with a given tolerance set in the configfile).
- If the slope between lines in the pairs in following scans are about same (with a given tolerance set in the configfile).
- If the both line pairs in following scans ensures the conditions given in **Criterion I.**
- Extra Vote: If the distance between the lines pair in following scans is so distinct that none of other distances is close to it (uniqueness) (with a given tolerance set in the configfile) than extra vote is given to the line pair.
- Extra Vote: If the lines are same overlapping state extra vote is given to the line pairs.

In Figure 3.20, the general schema that describes Criterion 4 is given.

**Table 3.9: Sub matching table (*MT4*) of Criterion 4. The votes of the lines according to the given Criterion 4 with weighting.**



|   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 64 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 58 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 20 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 46 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 17 | 0 | 3 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 50 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 50 |

In Table 3.9, the resulting sub matching table is shown when the Criterion 4 is applied onto the LADAR range scan given in Figure 3.15 and Figure 3.16. As we can see from Table 3.9, some of the line pairs are highly voted. In fact, this comes mostly from extra votes. Since most of the line to line distances for the given range scans in these figures are unique. Also for the cases where multiple line pairs are seen with same line to line distance, overlap type concept is added. This prevents matching of the line pairs with different relative neighbourhood to each other, such as the ones in a corridor divided by another.

**Overlap Type Calculation**

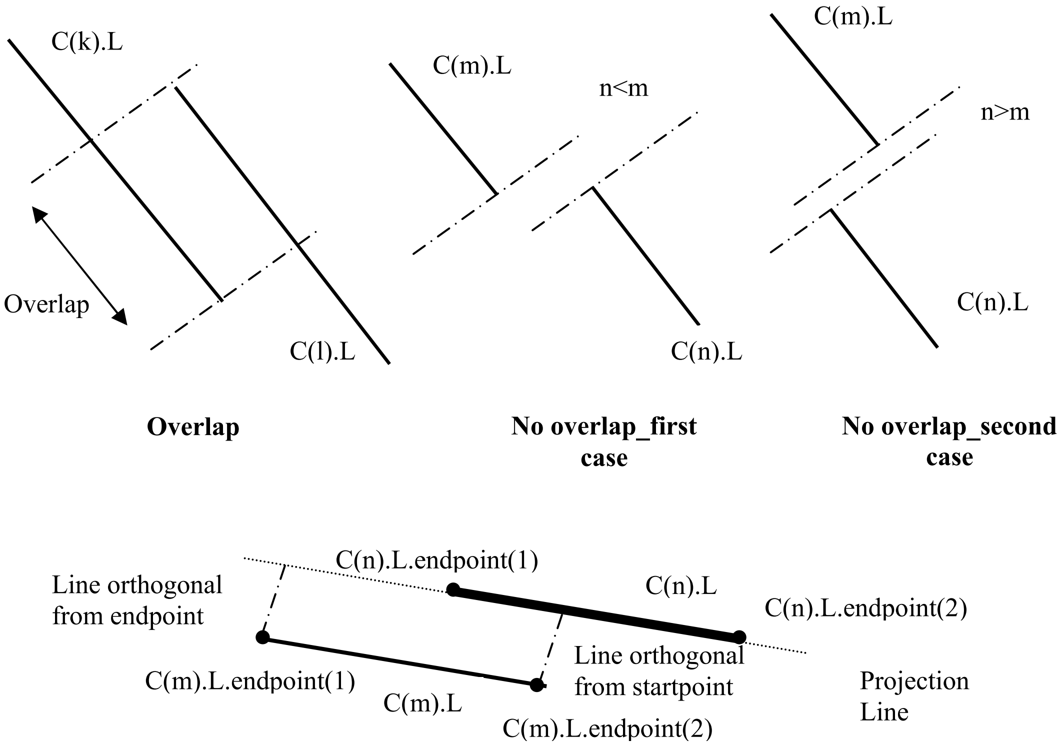


**Figure 3.21: The scheme illustrating overlap types defined.**

Overlap type is the value that defines whether the projections of two parallel lines on a parallel line with same slope overlaps. It can take three values: Overlap, no overlap first case, no overlap second case, which is given in Figure 3.21. Overlap corresponds that when these lines are projected on a parallel axis, their intervals in that coordinate frame overlaps. No overlap first case, in this axis the interval of first seen line by laser scanner is higher than the second one. No overlap second case, in this axis the interval of first seen line by laser scanner is lower than the second one. The pseudo code for this process is given below:

**Table 3.10: The algorithm that calculates the overlap types.**

```
Algorithm Overlap_Type_Calculation(C[m].L , C[n].L )
function orthogonal_line(Line1, Point1)    \\ computes the line passing through Point1 and orthogonal to Line1
function compdist2line(Line1, Point1) \\ computes the distance between Line 1 and Point1
```

**Table 3.10 Continued**

```
$C^{[m]}.Ort\_Line^{[1]}$ = **orthogonal_line(** $C^{[m]}.L$ **,** $C^{[m]}.L.endpoint^{[1]}$ **)**
$C^{[m]}.Ort\_Line^{[2]}$ = **orthogonal_line(** $C^{[m]}.L$ **,** $C^{[m]}.L.endpoint^{[2]}$ **)**
**if compdist2line** ( $C^{[m]}.Ort\_Line^{[1]}$ , $C^{[n]}.L.endpoint^{[2]}$ ) >0 and ...
  **compdist2line** ( $C^{[m]}.Ort\_Line^{[2]}$ , $C^{[n]}.L.endpoint^{[2]}$ ) >0 and...
  **compdist2line** ( $C^{[m]}.Ort\_Line^{[1]}$ , $C^{[n]}.L.endpoint^{[1]}$ ) >0 and...
  **compdist2line** ( $C^{[m]}.Ort\_Line^{[2]}$ , $C^{[n]}.L.endpoint^{[1]}$ ) >0 **then**
        result=No_Overlap_First
**elseif compdist2line** ( $C^{[m]}.Ort\_Line^{[1]}$ , $C^{[n]}.L.endpoint^{[2]}$ ) <0 and...
  **compdist2line** ( $C^{[m]}.Ort\_Line^{[2]}$ , $C^{[n]}.L.endpoint^{[2]}$ ) <0 and ...
  **compdist2line** ( $C^{[m]}.Ort\_Line^{[1]}$ , $C^{[n]}.L.endpoint^{[1]}$ ) <0 and...
  **compdist2line** ( $C^{[m]}.Ort\_Line^{[2]}$ , $C^{[n]}.L.endpoint^{[1]}$ ) <0 **then**
        result= No_Overlap_Second
**else**
        result= Overlap
**endif**
**return** result
```

### 3.5.6 Criterion 5: For Lines Ended in Both Sides

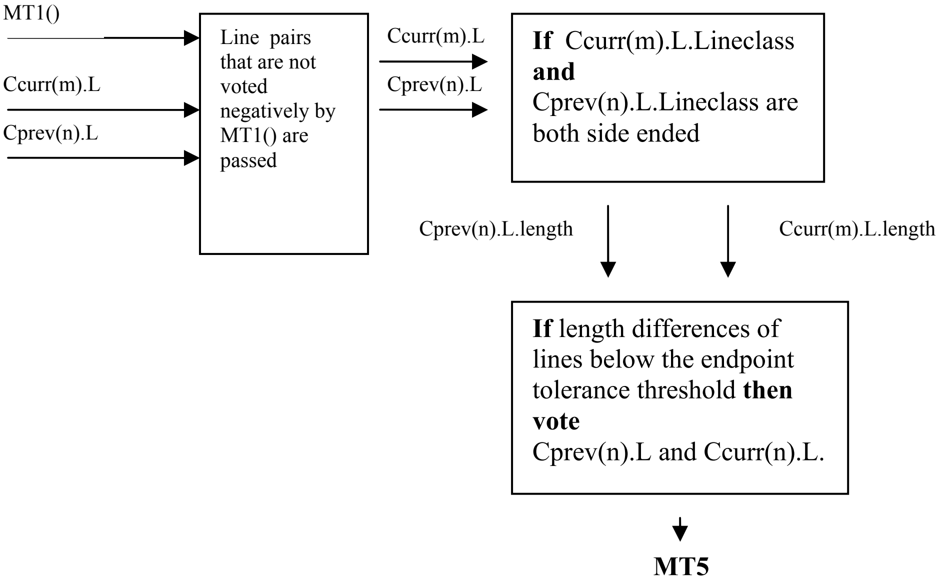

MT1()
Ccurr(m).L
Cprev(n).L
Line pairs that are not voted negatively by MT1() are passed
Ccurr(m).L
Cprev(n).L
**If** Ccurr(m).L.Lineclass **and** Cprev(n).L.Lineclass are both side ended
Cprev(n).L.length
Ccurr(m).L.length
**If** length differences of lines below the endpoint tolerance threshold **then vote** Cprev(n).L and Ccurr(n).L.
**MT5**

**Figure 3.22: The figure illustrating the scheme of Criterion 5 implemented.**

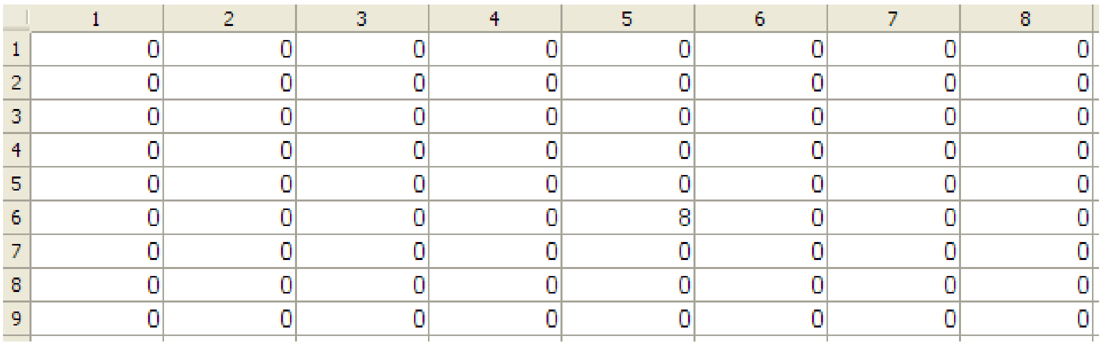
Some of the lines in the scan are ended in both sides. The information about how a line is accepted as ended is given in Section 3.4.1. The matching criteria are:

- If the both lines in the following scans are ended in both sides
- If the both lines have the same line length (with a given endpoint tolerance threshold calculated for each line).

The negative voting criterion is:

- If the both lines in the following scans are ended in both sides but line length difference is above the threshold (which is set in the config file)

**Table 3.11: Sub matching table (*MT5*) of Criterion 5. The votes of the lines according to the given Criterion 5 with weighting.**



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 8 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

In Table 3.11, the resulting sub matching table is shown when the Criterion 5 is applied onto the LADAR range scan given in Figure 3.15 and Figure 3.16

### Endpoint Tolerance Threshold Calculation

Endpoint tolerance threshold is the given tolerance value while matching both side ended lines. The length of a real line segment could change significantly between two consecutive scans based on the position and slope of the real line segment. Since there is always an error at the endpoints (we do not know how much more line segment goes after the endpoint found from LADAR data) based on the divergence between two consecutive range scan beam at that distance, which is given with Equation 4.11. The illustration of this situation is given in Figure 3.23:

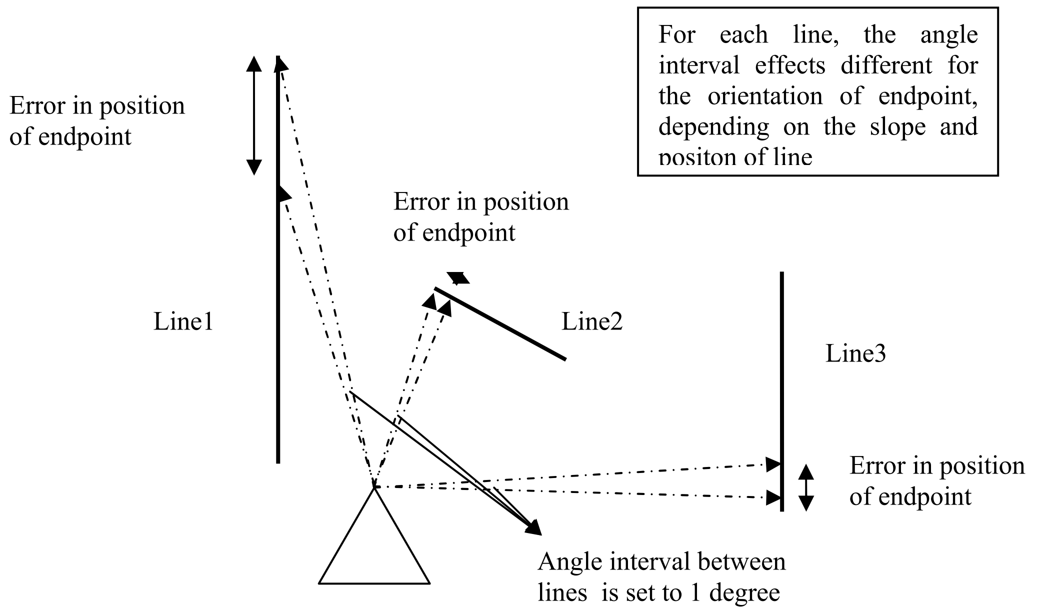

For each line, the angle interval effects different for the orientation of endpoint, depending on the slope and positon of line
Error in position of endpoint
Error in position of endpoint
Line1
Line2
Line3
Error in position of endpoint
Angle interval between lines is set to 1 degree

**Figure 3.23: The illustration that is showing the affect of slope and orientation of the line to the position of an endpoint.**

For this purpose a variable threshold is defined for real corner matching criterion. The formula for this threshold is:

$$Threshold^{[k,l]} = 2 * \tan(\pi/180) * 2 * \max\left( \left\| \frac{1}{m} \sum_{i=1}^{m} C_{curr}{}^{[k]}.xy^{[i]} \right\|, \left\| \frac{1}{n} \sum_{i=1}^{n} C_{prev}{}^{[l]}.xy^{[i]} \right\| \right) \quad \textbf{(3-13)}$$

Where $C_{curr}{}^{[k]}.xy^{[i]}$ ith x, y Cartesian coordinates of kth cluster of current scan is, $C_{prev}{}^{[l]}.xy^{[i]}$ is *i*th x, y Cartesian coordinates of *l*th cluster of previous scan. *m, n* are the total number of scan points . The formula assumes the endpoint error based on the maximum distance between LADAR and center of line in current and previous scans. The result is multiplied according to the projection calculation onto the line.

This criteria is simple, but it is good to find and vote the both side ended lines in an environment. On the other hand, as it is given in endpoint tolerance threshold calculation part, the length of a both side ended line can still be changed according to the aspect of the LADAR into this line and also the distance between the line and LADAR. So correctly calculating this threshold, such as given above, solves the non-matching problems of same both side ended lines in previous and current data.

### 3.5.7 Criterion 6: Searching Angle Pattern between the Following Lines

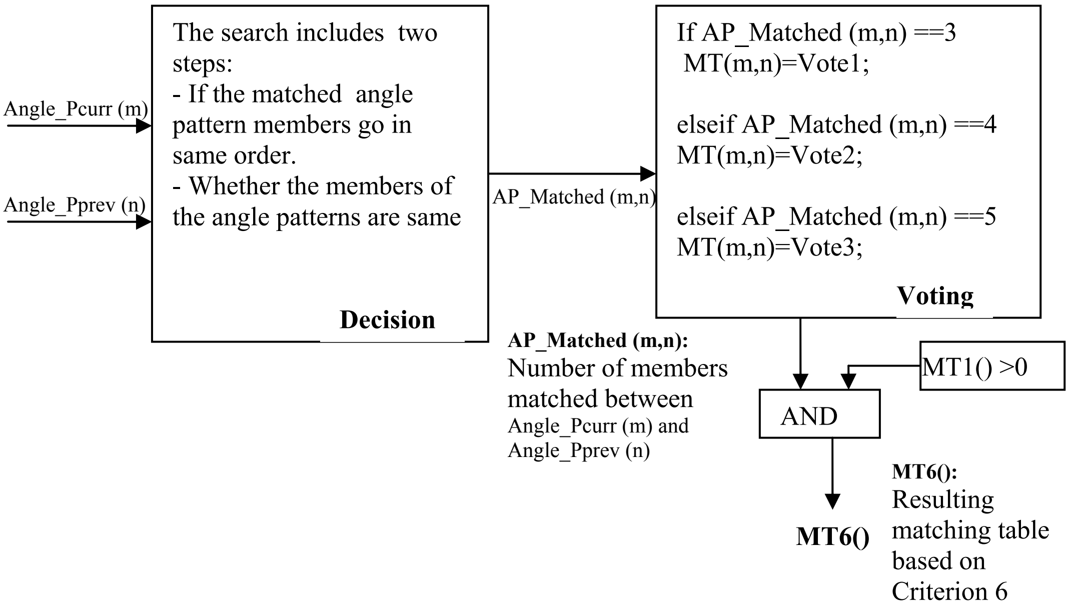

Angle_Pcurr (m)
Angle_Pprev (n)
The search includes two steps:
- If the matched angle pattern members go in same order.
- Whether the members of the angle patterns are same
Decision
AP_Matched (m,n)
If AP_Matched (m,n) ==3
MT(m,n)=Vote1;
elseif AP_Matched (m,n) ==4
MT(m,n)=Vote2;
elseif AP_Matched (m,n) ==5
MT(m,n)=Vote3;
Voting
AP_Matched (m,n): Number of members matched between Angle_Pcurr (m) and Angle_Pprev (n)
MT1() >0
AND
MT6()
MT6(): Resulting matching table based on Criterion 6

**Figure 3.24: The figure illustrating the scheme of Criterion 6 implemented.**

Criterion 6 is based on, by using the angle differences creating an angle pattern between the following lines in a scan and searching the same pattern in the following scan. This pattern search is made as:

1) First the slope differences of the lines are found. This is a matrix as given in Table 3.12:

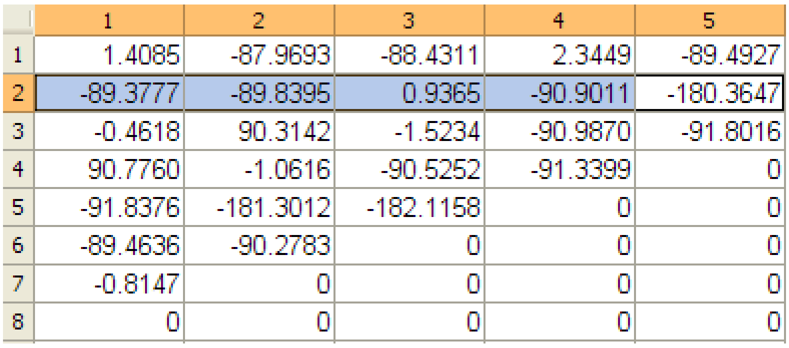
**Table 3.12: The structure of slope differences matrix of the lines.**

| | | | | |
|---|---|---|---|---|
| $m_1 - m_2$ | $m_1 - m_3$ | $m_1 - m_4$ | $m_1 - m_5$ | $m_1 - m_6$ |
| $m_2 - m_3$ | $m_2 - m_4$ | $m_2 - m_5$ | $m_2 - m_6$ | $m_2 - m_7$ |
| $m_3 - m_4$ | $m_3 - m_5$ | $m_3 - m_6$ | $m_3 - m_7$ | $m_3 - m_8$ |
| $m_4 - \ldots$ | …. | …. | …. | …. |
| …. | …. | …. | …. | …. |

where $m_i$ corresponds to $C^{[i]}.L.slope$, the slope of fitting line to cluster *i*. These slope difference values are found for both of the following scans. Each row of this matrix corresponds to $Angle\_P^{[i]}$, which is found with given pseudo code in Table 3.1 .

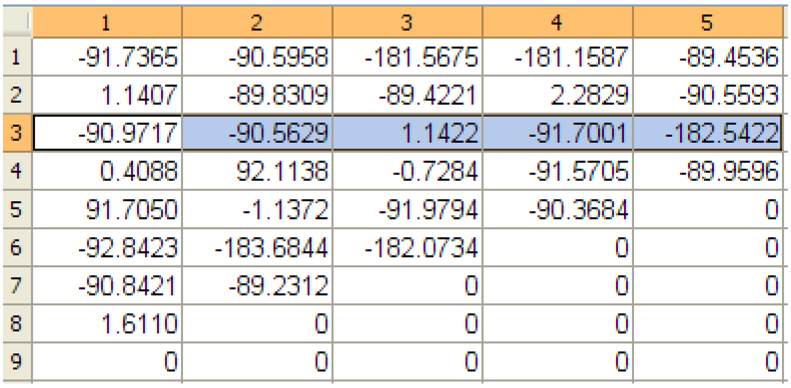
2) These matrices are voted according to their level they overlap. Each $Angle\_P_{curr}^{[i]}$ of the current scan is compared with $Angle\_P_{prev}^{[j]}$.

**Table 3.13: The slope differences matrix of the previous scan.**



| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 1.4085 | -87.9693 | -88.4311 | 2.3449 | -89.4927 |
| 2 | -89.3777 | -89.8395 | 0.9365 | -90.9011 | -180.3647 |
| 3 | -0.4618 | 90.3142 | -1.5234 | -90.9870 | -91.8016 |
| 4 | 90.7760 | -1.0616 | -90.5252 | -91.3399 | 0 |
| 5 | -91.8376 | -181.3012 | -182.1158 | 0 | 0 |
| 6 | -89.4636 | -90.2783 | 0 | 0 | 0 |
| 7 | -0.8147 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 |

**Table 3.14: The slope differences matrix of the current scan.**



| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | -91.7365 | -90.5958 | -181.5675 | -181.1587 | -89.4536 |
| 2 | 1.1407 | -89.8309 | -89.4221 | 2.2829 | -90.5593 |
| 3 | -90.9717 | -90.5629 | 1.1422 | -91.7001 | -182.5422 |
| 4 | 0.4088 | 92.1138 | -0.7284 | -91.5705 | -89.9596 |
| 5 | 91.7050 | -1.1372 | -91.9794 | -90.3684 | 0 |
| 6 | -92.8423 | -183.6844 | -182.0734 | 0 | 0 |
| 7 | -90.8421 | -89.2312 | 0 | 0 | 0 |
| 8 | 1.6110 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 |

As we can see from the Table 3.13 and Table 3.14, these scans overlap in the dark colored 1x5 rectangles. If the pattern is fully overlaps with the lines that construct than the pattern takes the maximum voting in the overlap table. If the overlap decreases in the pattern, such as 4-3 slope differences overlap, then the voting for this pattern decreases. The structure of the overlapping table is given in Table 3.15:

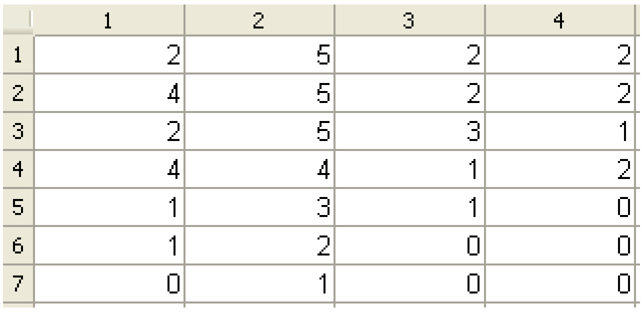
**Table 3.15: The structure of overlapping table. For a chosen row and column, the first number gives the row used from Table 3.13 and the second number gives the row used from Table 3.14.**

| ***The row in slope difference matrix in previous scan*** | ***(The row in slope difference matrix in previous scan ,The row in slope difference matrix in current scan)*** | | | |
|---|---|---|---|---|
| ***1*** | *(1,1)* | *(1,2)* | *(1,3)* | *(1,4)* |
| ***2*** | *(2,2)* | *(2,3)* | *(2,4)* | *(2,5)* |
| **…** | … | … | … | … |
| ***N*** | *(N,N)* | *(N,N+1)* | *(N,N+2)* | *(N,N+3)* |

In Table 3.15, for a chosen cell, the first number gives the row used in slope difference matrix in previous scan and the second number gives the row used in slope difference matrix in current scan. So the overlapping score of these rows is calculated and put into the cell.

For the example LADAR scans given in Figure 3.15 and Figure 3.16 the resulting overlap table is given in Table 3.16:

**Table 3.16: The resulting overlap table in the structure given with Table 3.15 for Figure 3.15 and Figure 3.16.**



| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 2 | 5 | 2 | 2 |
| 2 | 4 | 5 | 2 | 2 |
| 3 | 2 | 5 | 3 | 1 |
| 4 | 4 | 4 | 1 | 2 |
| 5 | 1 | 3 | 1 | 0 |
| 6 | 1 | 2 | 0 | 0 |
| 7 | 0 | 1 | 0 | 0 |

When we have looked at this overlap table, the maximum points are seen at the second column on the rows. This means that if we have looked at Nth slope difference matrix in the previous scan, it overlaps with N+1th slope difference matrix in the current scan. We can see the overlapping in Table 3.13 and Table 3.14 at the 2nd row and the 2nd column (which means 3rd slope difference matrix row) of the overlap table in Table 3.16.

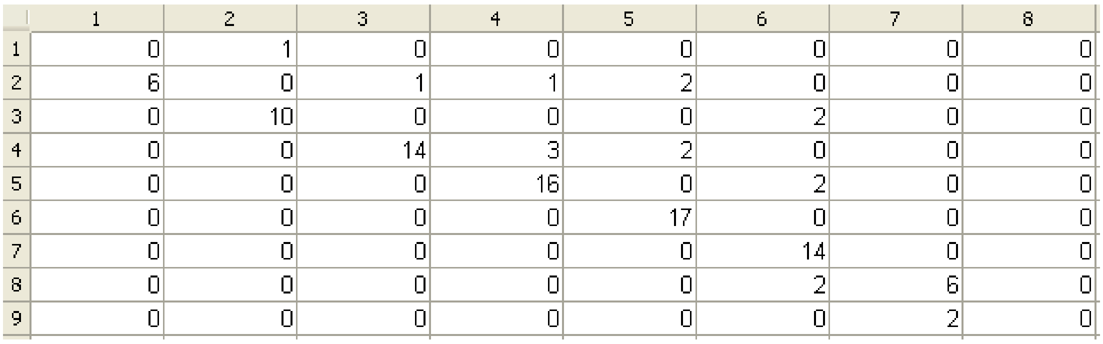
3) The maximum points for each row in the overlap table are found and the lines that construct this pattern are voted by considering their value in overlap table. In fact, when the maximum point in the overlap table decreases, this means the lines constructing that pattern

does not fully overlap, for example some of the lines do not seen or detected in one of the scans. So reliability of that pattern decreases.

4) This criterion is applied to the scan pair twice**.** One controls the line angle patterns with the next line angle patterns in the following scan and the other controls the line angle patterns with the previous line angle patterns in the following scan. This is made twice since whether pose angle changes left or right, this is not known yet while this criterion is applied.

The resulting addition from this criterion to the matching table is given below:

**Table 3.17: Sub matching table (*MT6*) of Criterion 6. The votes of the lines according to the given Criterion 6 with weighting.**



|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 6 | 0 | 1 | 1 | 2 | 0 | 0 | 0 |
| 3 | 0 | 10 | 0 | 0 | 0 | 2 | 0 | 0 |
| 4 | 0 | 0 | 14 | 3 | 2 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 16 | 0 | 2 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 17 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 14 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 2 | 6 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |

In Table 3.17, the resulting sub matching table *(MT6)* is shown when the Criterion 6 is applied onto the LADAR range scan given in Figure 3.15 and Figure 3.16. In this criterion, if the overlapping is high, it means we are pretty sure that all these line pairs are matching, and voting to these line pairs increases. As mentioned in Section 3.5.4, because of the perpendicular construction of features, the angle patterns consisting of slope differences that are multiplicative of 90 degrees are commonly seen. So a wrong overlap between irrelevant patterns can be seen. As a result, if overlap is less, it means we are not sure about the matching. Therefore voting is not given to them.

### 3.5.8 Criterion 7: Searching an Corner Distance Pattern between the Following Corners

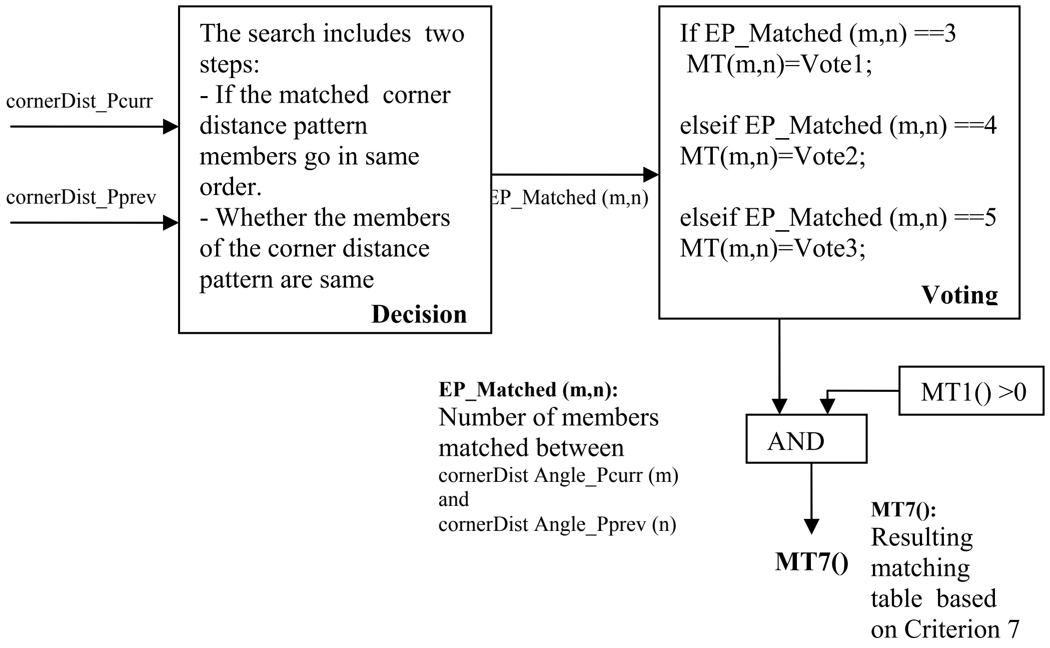

cornerDist_Pcurr
cornerDist_Pprev
The search includes two steps:
- If the matched corner distance pattern members go in same order.
- Whether the members of the corner distance pattern are same
Decision
EP_Matched (m,n)
If EP_Matched (m,n) ==3
MT(m,n)=Vote1;
elseif EP_Matched (m,n) ==4
MT(m,n)=Vote2;
elseif EP_Matched (m,n) ==5
MT(m,n)=Vote3;
Voting
MT1() >0
AND
EP_Matched (m,n):
Number of members matched between
cornerDist Angle_Pcurr (m)
and
cornerDist Angle_Pprev (n)
MT7()
MT7():
Resulting matching table based on Criterion 7

**Figure 3.25: The figure illustrating the scheme of Criterion 7 implemented.**

This criterion is based on searching the distances between all types of following corners. The steps of the criterion are given below:

1) The intersection points of consecutive lines are found. These intersection points can be the real corners or virtual corners where the extensions of the line pairs intersect. For each following line pair which assures the line angle difference above a given threshold, the corner point is found.

**Implementation Notes:** The angle difference threshold for the line pairs is important, since if this difference is smaller, the corner point orientation became much more inaccurate. Since a small error in the slope of the lines makes the corner point move on the direction of the resultant vector generated by the lines constructing the corner (Also given in Figure 3.29).

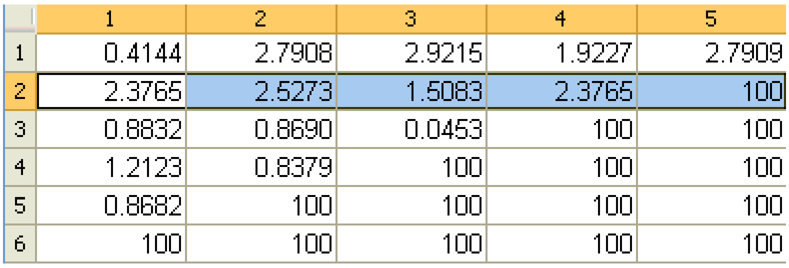
2) The distance between the following corner points are found. According to the found distances, corner distances table is created for the two following scans. This is a matrix given in Table 3.18:

**Table 3.18: The structure of corner distances table.**

| | | | | |
|---|---|---|---|---|
| $\left\|e_{1,2}-e_{1,3}\right\|$ | $\left\|e_{1,2}-e_{1,4}\right\|$ | $\left\|e_{1,2}-e_{1,5}\right\|$ | $\left\|e_{1,2}-e_{1,6}\right\|$ | $\left\|e_{1,2}-e_{1,7}\right\|$ |
| $\left\|e_{2,3}-e_{2,4}\right\|$ | $\left\|e_{2,3}-e_{2,5}\right\|$ | $\left\|e_{2,3}-e_{2,6}\right\|$ | $\left\|e_{2,3}-e_{2,7}\right\|$ | $\left\|e_{2,3}-e_{2,8}\right\|$ |
| $\left\|e_{3,4}-e_{3,5}\right\|$ | $\left\|e_{3,4}-e_{3,6}\right\|$ | $\left\|e_{3,4}-e_{3,7}\right\|$ | $\left\|e_{3,4}-e_{3,8}\right\|$ | $\left\|e_{3,4}-e_{3,9}\right\|$ |
| $\left\|e_{4,5}-\ldots\right\|$ | .... | .... | .... | .... |
| .... | .... | .... | .... | .... |

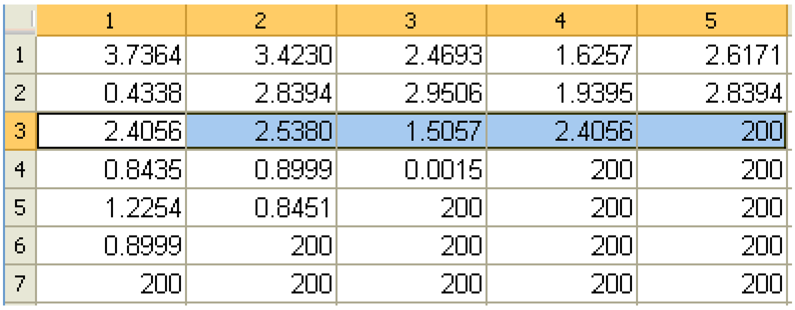
Where $e_{i,j}$ corresponds the intersection point of line $\mathrm{C}^{[i]}.L$ and $\mathrm{C}^{[i+j]}.L$. Therefore $\left\|e_{i,i+1}-e_{i,i+j}\right\|$ corresponds to $cornerDist\_P^{[i,j]}$. Each row of this matrix corresponds to $cornerDist\_P^{[i]}$, which is found with given pseudo code in Table 3.2. The slope difference values are found for both of the following scans. If the slopes of these line pairs are below the given threshold, then an corner point could not be defined. So the distances related with this corner are given as blank in the corner distances table (given as 100 and 200 in Table 3.19 and Table 3.20).

**Table 3.19: The corner distances matrix of the previous scan.**



| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0.4144 | 2.7908 | 2.9215 | 1.9227 | 2.7909 |
| 2 | 2.3765 | 2.5273 | 1.5083 | 2.3765 | 100 |
| 3 | 0.8832 | 0.8690 | 0.0453 | 100 | 100 |
| 4 | 1.2123 | 0.8379 | 100 | 100 | 100 |
| 5 | 0.8682 | 100 | 100 | 100 | 100 |
| 6 | 100 | 100 | 100 | 100 | 100 |

**Table 3.20: The corner distances matrix of the current scan.**



| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 3.7364 | 3.4230 | 2.4693 | 1.6257 | 2.6171 |
| 2 | 0.4338 | 2.8394 | 2.9506 | 1.9395 | 2.8394 |
| 3 | 2.4056 | 2.5380 | 1.5057 | 2.4056 | 200 |
| 4 | 0.8435 | 0.8999 | 0.0015 | 200 | 200 |
| 5 | 1.2254 | 0.8451 | 200 | 200 | 200 |
| 6 | 0.8999 | 200 | 200 | 200 | 200 |
| 7 | 200 | 200 | 200 | 200 | 200 |

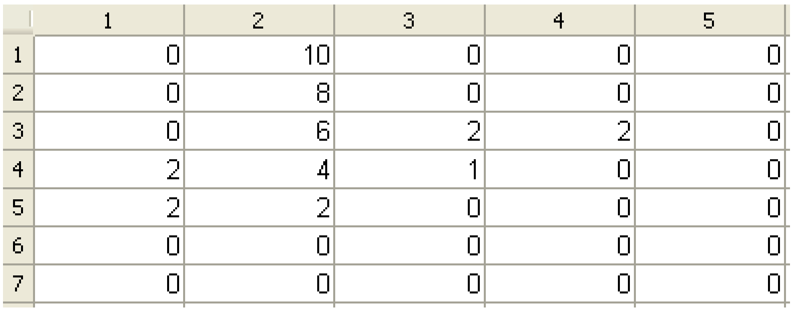
3) According to the corner distance matrices of following scans, an overlap table is created. The structure of the overlap table is given in Table 3.21:

**Table 3.21: Corner distances overlap matrix. For a chosen row and column, the first number gives the row used from Table 3.19 and the second number gives the row used from Table 3.20.**

| ***The row in corner distances matrix in previous scan*** | ***(The row in corner distances matrix in previous scan, The row in corner distances matrix in current scan)*** | | | |
|---|---|---|---|---|
| ***1*** | *(1,1)* | *(1,2)* | *(1,3)* | *(1,4)* |
| ***2*** | *(2,2)* | *(2,3)* | *(2,4)* | *(2,5)* |
| **…** | … | … | … | … |
| ***N*** | *(N,N)* | *(N,N+1)* | *(N,N+2)* | *(N,N+3)* |

In Table 3.21, for a chosen cell, the first number gives the row used in corner distances matrix in previous scan and the second number gives the row used in corner distances matrix in current scan. So the overlapping score of these rows is calculated and put into the cell. For example, LADAR scans given in Figure 3.15 and Figure 3.16, the resulting overlap table is given below as:

**Table 3.22: The resulting corner distances overlap table in the structure given with Table 3.21 for Figure 3.15 and Figure 3.16.**



| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 10 | 0 | 0 | 0 |
| 2 | 0 | 8 | 0 | 0 | 0 |
| 3 | 0 | 6 | 2 | 2 | 0 |
| 4 | 2 | 4 | 1 | 0 | 0 |
| 5 | 2 | 2 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 |

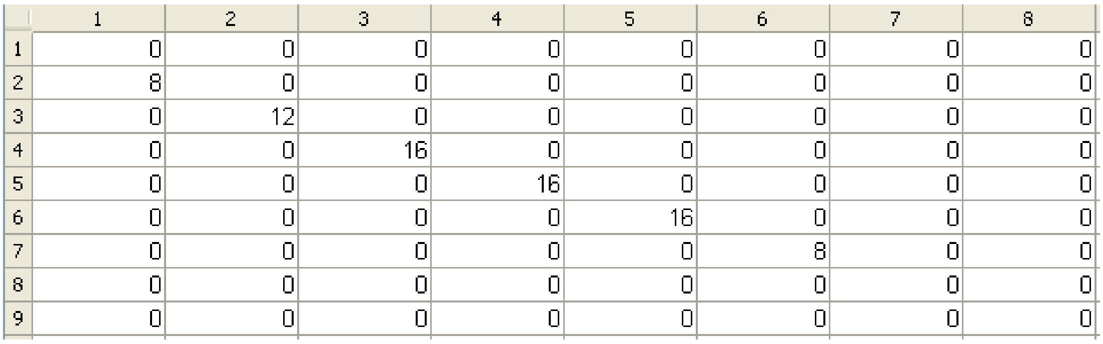
When we have looked at this overlap table, two maximum points are seen at the second column on the rows. This means that if we have looked at 2nd row in corner distance matrix of the previous scan, it overlaps with 3rd row in corner distance matrix of the current scan or 4th row in corner distance matrix of the previous scan; it overlaps with 5th corner distance matrix of the current scan. We can see the overlapping in Table 3.19 and Table 3.20 at the

2nd row and the 2nd column (which means 3rd corner distance matrix row) of the overlap table in Table 3.22.

The maximum points for each row in the overlap table are found and the lines that construct this pattern are voted by considering their value in overlap table. In fact, when the maximum point in the overlap table decreases, this means the lines constructing that pattern does not fully overlap. So reliability of that pattern decreases. As a result the voting of the lines they construct also decreases.

This criterion is applied twice from current to next and current to previous, again because of the same reasons given in Criterion 6. The resulting addition from this criterion to the matching table is given below:

**Table 3.23: Sub matching table (*MT7*) of Criterion 7. The votes of the lines according to the given Criterion 7 with weighting.**



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 12 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 16 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 16 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 16 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

In Table 3.23, the resulting sub matching table is shown when the Criterion 6 is applied onto the LADAR range scan given in Figure 3.15 and Figure 3.16.

**Implementation Notes:** There is an important difference between Criterion 6 and Criterion 7; Criterion 6 looks for the angles between the lines. A pattern consisting of angles is more easily seen then a pattern consisting of corner distances. Since the intersecting lines mostly create angles with degrees 0, 90,180,270 in an environment. In fact most of the indoor areas are in rectangular shape and contains rectangular furniture. But to see same distances between the corners is hard and to see same distance pattern is much harder. Therefore the voting values of Criterion 7 need to be higher than Criterion 6.

### 3.5.9 Sub Matching Tables Addition with Weighting

In this part, sub matching tables are multiplied with weightings and summed and final matching table (*MT()*) is formed. As you can see from the sub matching tables given, the affect of these sub matching tables to the final matching table is different. If we evaluate each criterion by considering its importance with respect to others:

**Criterion1:** This criterion gives the line pairs that have no chance to match because of the dynamic limitations of the robot. To remove the effect of these line pairs, they are highly negatively voted.

**Criterion 2:** The weighting to this criterion is low, since in spite of the sub criteria which make mismatching harder, there is still a chance to make a mismatch. But in uniqueness case, this weight is multiplied with inner weight, which highly increases its voting.

**Criterion 3:** This criterion is a simple criterion, and open to mismatches, so it is weighted low. But in low structured environments, number of real corners became so low such as one or two…Therefore in these cases, since the high inner weight of uniqueness case, the line pairs are highly voted, which helps the matching table in these environments.

**Criterion 4:** The weighting of this criterion is neither high nor low. Most of the time, the voting output depends on the inner weighting. Parallel line pairs can be seen easily, but also these pairs with equivalent distance can be seen sometimes, which can cause mismatches. Therefore inner weight of this is taken low. If the line to line distance is different from others (uniqueness case), the parallel line pair is highly weighted. Also when the overlapping for the sub criteria increases, inner weighting highly increases. As a result we can see high voting at Table 3.9, because of high inner weights coming from high degree overlap and uniqueness case.

**Criterion 5:** The weighting of this criterion is low. Since length of both sides ended lines can easily be affected from the line slope seen from LADAR position which is given in Endpoint Tolerance Threshold Calculation part. Also they are affected from the errors occurred in the LADAR data, such as multiple line fit to same corridor. Therefore unless the inner weight is high because of uniqueness, the output is low.

**Criterion 6:** The weighting of this criterion is normal. The decision is totally given to inner weighting. Inner weighting is low, if the overlap between the angle patterns is low, it is zero for one or two angle overlap, but if it increases the weight also increases.

**Criterion 7:** The weighting of this criterion is higher than normal. In fact it has same pattern structure with Criterion 6, but the mismatch becoming chance is much lower. Therefore it has higher weighting than the one of Criterion 6. The inner weighting is same as Criterion 6.

### 3.5.10 Important Notes about the Criteria

The main aim in the criteria given between 3.5.2 and 3.5.8 is decreasing the positive votes for mismatches, also increase the voting of correct matches. For this purpose some methods in literature are eliminated and some new methods are added. These are;

1) Unique line pairs in a criterion compared are highly voted. This means if a relation between line pairs does not seen between any different line pairs, these are highly promoted. For example, this is mostly seen for parallel line distances, most of the distances between parallel lines are so specific that they could not be equal to each other, because of the low position error of LADAR data and the fitting line from the least square method.

2) Any criterion directly giving a decision as the given line pairs do not match is not used. In fact, this kind of rules makes the matching line pairs, which are voted positively from all other criteria, invalid. Therefore, except the Criterion 1, which makes a direct decision for just obvious ones by taking into account the vehicle dynamics, no criteria that gives such a decision is used. All criteria are based on promotion of matching ones.

In the real LADAR data, sometimes information about the given line segment changes. This could be in different properties of features, such as a change in line length, line endpoint positions, or line type, also in corner position or existence of a line or corner. In fact, the reason can be that any slope change in LADAR referenced to the ground plane, that navigation in a dynamic environment or that the objects reflects the LADAR beams with changing intensities. An example situation taken from real LADAR data is presented in Figure 3.26.

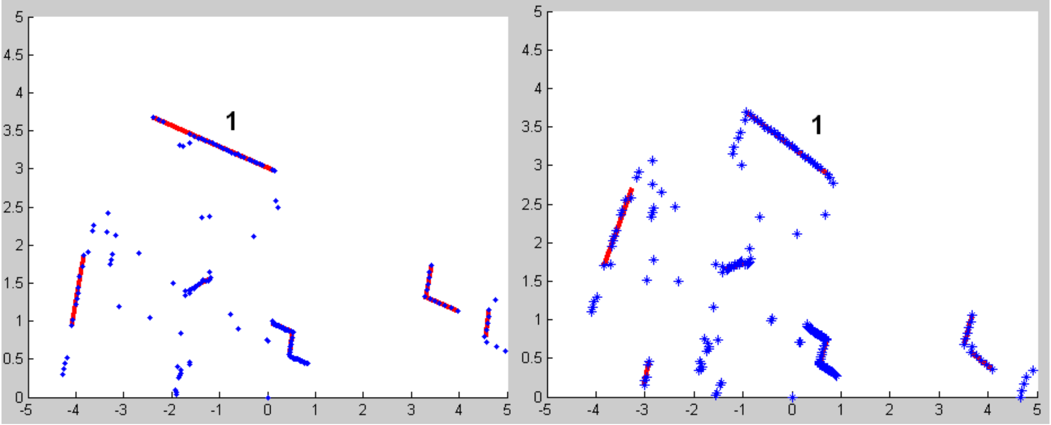


**Figure 3.26: The figure illustrating the fitting lines to LADAR data. Left one is from previous scan and the right one is from current scan.**

As it is seen from Figure 3.26, a change in LADAR data between following scans occurs for the environment around the fitting line numbered 1 (points on its left disappeared). Therefore the length and the type of the line change.

3) The relations about the type of line ends, such as open or left side ended, right side ended are not used as individual criterion, also for a decision as the given line pairs do not match. Since these are easily affected from the wrong fitting lines to the clusters corresponding same feature in consecutive scans in real LADAR data. The real LADAR data is exposed to some errors. Therefore the types of endpoints are easily changed. As a result, a definite decision is not appropriate.

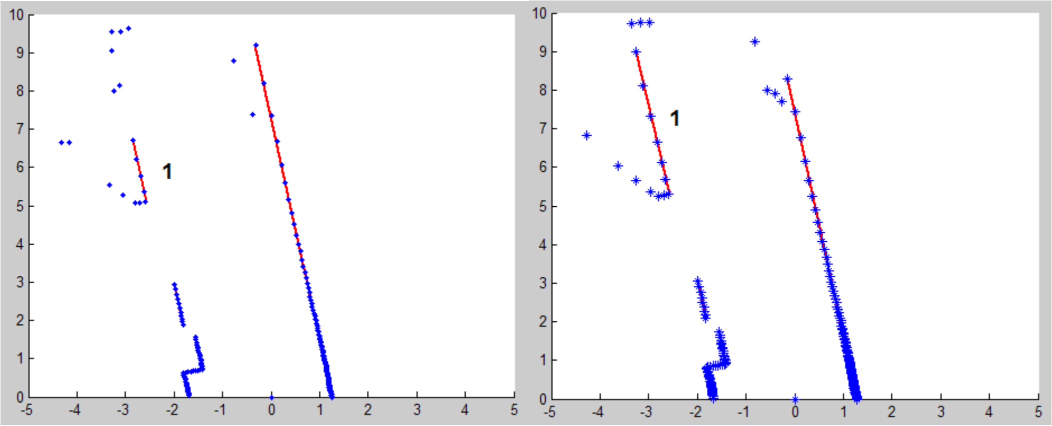


**Figure 3.27: The figure illustrating the fitting lines to LADAR data. Left one is from previous scan and the right one is from current scan.**

In Figure 3.27, the line numbered with 1 belongs to same fitting line in current and previous scans. These lines could be defined as both side ended lines. In fact, it is seen that the endpoint of lines close to (0,0) point is ended with points goes perpendicular to the line, When we have look at their endpoint further from origin, the line in left figure is ended, since the consecutive point is too far from the endpoint, which is above the expected divergence between consecutive scans explained in Figure 3.23. For the line in right figure, again there are points showing that the line is ended, which are close and right side of the line. Therefore any comparison could decide as they are not matching. Also with a small error one of them could be decided as left side ended.

4) Angle pattern and corner distance pattern is specially used. Since going from an array of feature properties, such as angle difference series of consecutive lines, decides the matching of line pairs much accurate than the comparison of just one feature property. In fact, the angle differences of lines are multipliers of 90 degree in most indoor environments, which could cause mismatches when one by one comparison is made.

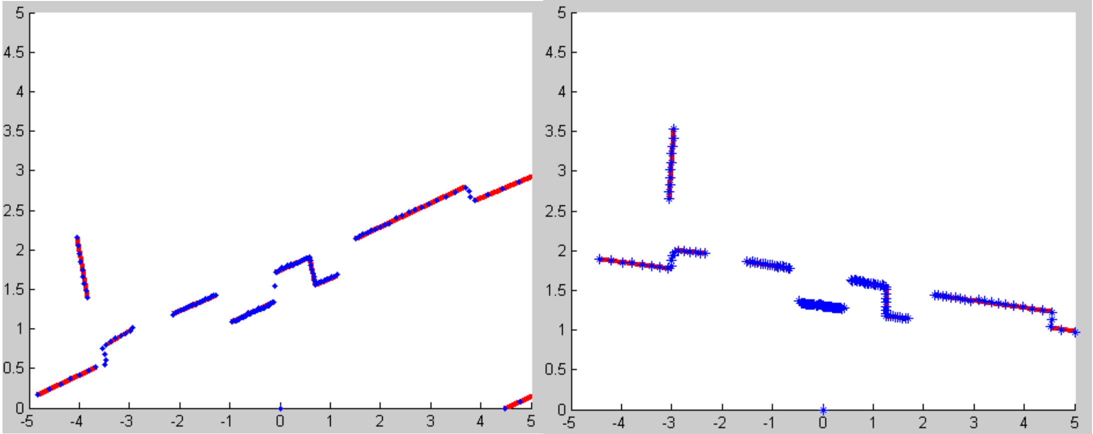


**Figure 3.28: The figure illustrating the fitting lines to LADAR data. Left one is from previous scan and the right one is from current scan.**

In some environments the lines constitute mind mixing situations that scan matching should handle. As you can see from Figure 3.28, there are multiple line pairs with same parallel line distance and there are multiple angles between lines that are equal and multiple corner distances that are equal. For this purpose, using criterion that compares the properties of multiple line pairs gives more accurate results.

### 3.5.11 Application Areas of Criteria and Their Benefits:

Each criterion has an importance in the implementation of scan matching. Criterion 1 prevents algorithm from redundant calculation, since it demotes the line pairs which does not have any chance to match because of the dynamics of the robot (especially maximum rotation rate of the robot). If the limitations of the criterion are held in dynamics of the robot, it does not have a chance to make wrong judgment.

Criterion 2 is added to consider low structured environments. This kind of situations is mostly seen for the LADAR data of real environments, such as long corridors, dead-end ways. Also these are seen especially when the robot turns in such an area. Since the area is low structured, the observed features are not enough to apply standard criteria. For this purpose, the lines that are considerably longer than others in same slope are matched. This algorithm is simple; gives advantage in low structured environments, and usually could not find matching in dense structured environments. Also its mismatch chance is highly degraded by addition of uniqueness.

Criterion 3 is a standard simple matching criterion. It is specially implemented for any case that has corner intersection angles different from standard ones (90,270). It gives important advance in these situations. By uniqueness search, its wrong matching capability highly degraded. Also this criterion is powerful in low structured environments.

Criterion 4 is a standard matching criterion. But it has improved by addition of extra voting to uniqueness and overlap calculation. In most of the real environments parallel lines are seen. Mostly these line pair distances are unique. Furthermore, to separate the line pairs with same line to line distance, overlap information is added. Also to prevent some mismatches, extra vote to uniqueness is added.

Criterion 5 is a standard matching criterion. But it has improved by addition of endpoint tolerance concept, which means an uncertainty is added to the length of both side ended line. Addition of this concept improves the robustness of the criterion; it prevents some of the potential mismatches happen because of the view angle of LADAR to these lines. On the other side, by addition of uniqueness, it gives a better correct decision capability.

Criterion 6 and 7 are novel techniques. They are not so powerful in low structured environments. On the other side, they have an important advance in an environment with adequate numbered features. Because of the pattern, comparison between multiple features is applied, which results with votings to multiple line segments. Also this results with an important decrease in mismatching chance.

Each criterion has very low computational complexity with respect to all scan matching algorithm. Also each criterion has a special benefit in the appropriate environments mentioned above. Also some of them are general and constructing the base of the LADAR feature based scan matching. Extracting one of them does not give a benefit to algorithm computationally. Conversely it degrades the robustness of the algorithm. Accordingly a comparison does not made between the criteria.

### 3.5.12 Evaluation of Matching Table

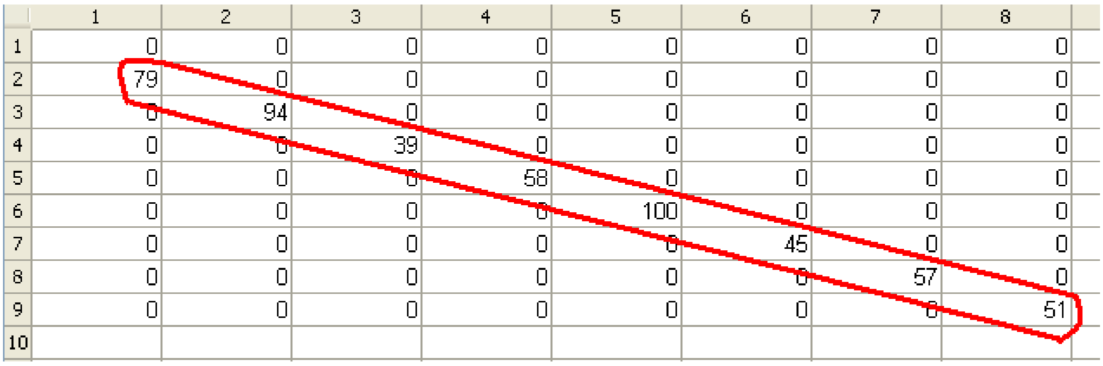
After the matching table is constructed, it is need to be evaluated correctly to find pose difference precisely. The following subsections gives the procedure implemented for the pose difference calculation.

#### 3.5.12.1 Threshold the Matching Table

The steps for threshold the matching table are:

1) First the values that are below a threshold (which is given in the config file) in the table are discarded.
2) Then for the rows and columns that contain the maximum points are found.
3) Zeroizing is started from the maximum point in the table by resetting its rows and columns. It goes to the smaller maximum points in other rows and columns.
4) In this process if column id in finding maximum of the rows and the row id in finding maximum of the columns do not match, this means for that row there is a conflict. So the one, which is close to the line passing from maximum points is taken. The resulting matching table is given in Table 3.24:

**Table 3.24: Final resulting matching table (*MT()*) . The corresponding row and column of the cells with values more than zero shows the matched line pairs.**



|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 79 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 94 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 39 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 58 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 100 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 45 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 57 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 51 |
| 10 |  |  |  |  |  |  |  |  |

**Notes:** As an example, assume the maximum point in column 3 will be 39 in row 4, and if maximum point in row 3 will be 42 in column 5, then the one closer to line connecting the maximum points will be chosen.

The line pairs which contain positive number in corresponding row and column of the final resulting table, are defined as *matched_LP* (matching line pairs). We can define them as:

**Table 3.25: The pseudo code giving the calculation of *matched_LP* (matching line pairs).**

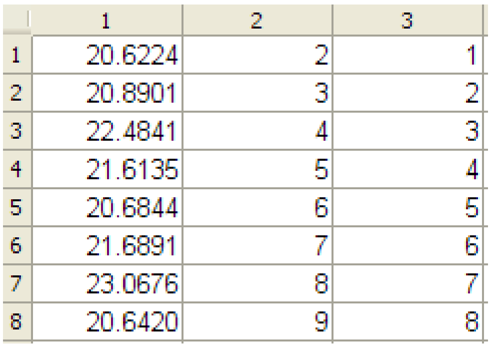
```
Algorithm_matched_Lines ($C_{curr}$ , $C_{prev}$ , MT)
count=1;
while count< ( number of MT()>0)
    if $\sum_{i=1}^{n} MT(count,i) > 0$ where n is the number of columns in MT()
        find the Index of MT(count,:)>0
        store $\langle LineID_{prev}^{[count]} = count, LineID_{curr}^{[count]} = Index \rangle$ to $matched\_LP^{[count]}$
        count=count+1;
    endif
endwhile
return matched_LP
```

where $matched\_LP$ consists of the line pairs' cluster ID from the previous and current scan matched according to *MT()*.

### 3.5.12.2 Finding the Rotation in the Pose of the Robot

In this part, the rotation in the pose of the robot is calculated. For this purpose; angle differences of the line pairs (one from previous scan and one from the current scan) are calculated.

**Table 3.26: The angle difference between the lines of previous and current scan.**



|   | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 20.6224 | 2 | 1 |
| 2 | 20.8901 | 3 | 2 |
| 3 | 22.4841 | 4 | 3 |
| 4 | 21.6135 | 5 | 4 |
| 5 | 20.6844 | 6 | 5 |
| 6 | 21.6891 | 7 | 6 |
| 7 | 23.0676 | 8 | 7 |
| 8 | 20.6420 | 9 | 8 |

In Table 3.26, first column consists from the angle difference in degree, second column the line number in current scan, third column the line number in the previous scan. As you can see from the angle difference in column 1 given in figure above, the robot has a rotation about 20 degrees. To find the result much precise a method eliminating the outliers is used. The steps for this method are given in Table 3.27:

**Table 3.27: The pseudo code of the algorithm that finds the rotation in robot pose from the slope differences between the matched line pairs by using outlier rejection methods.**

```
Algorithm_Rotation_Finding($C_{curr}$, $C_{prev}$, matched_LP, Threshold)
function slope_difference (slope1,slope2) // finds the slope difference
retrieve $\langle LineID_{prev}, LineID_{curr} \rangle$ from matched_LP
for i=1 to number of $LineID_{prev}$
        $SlopeDiff^{[i]}$ = slope_difference($C_{prev}^{[LineID_{prev}^{[i]}]}$.L.slope , $C_{curr}^{[LineID_{curr}^{[i]}]}$.L.slope )
endfor
deviation= $\infty$ ;
count= size of translations
while (deviation > Threshold1 and count> Threshold2 )
        count= number of members in $LineID_{prev}$
```

**Table 3.27continued**

```
    $j = LineID_{prev}^{[i]}$
    $jj = LineID_{curr}^{[i]}$
    $res^{[i]} = \sqrt{C_{prev}^{[j]}.L.length * C_{curr}^{[jj]}.L.length}$
    $result\_rotation = \frac{\sum_{i=1}^{count}(SlopeDiff^{[i]} * res^{[i]})}{\sum_{i=1}^{count} res^{[i]})}$
    C=max(| SlopeDiff - result_rotation|) ;
    find Index of C in max(| SlopeDiff - result_rotation|);
    store...
    $\left\langle \begin{array}{l} SlopeDiff^{[1:Index-1]}, SlopeDiff^{[Index+1:count]}, LineID_{prev}^{[1:Index-1]}, LineID_{prev}^{[Index+1:count]}, \\ LineID_{curr}^{[1:Index-1]}, LineID_{curr}^{[Index+1:count]} \end{array} \right\rangle$
    to matched_LP
    count=count-1
    deviation=max(| SlopeDiff - result_rotation|)
endwhile
return result_rotation
```

As you can see from the given method; the rotation in the pose is calculated recursively, in which for each step, the outlier angle which causes the biggest deviation (from the mean of the slope differences) above the defined threshold is eliminated. Also the rotation is calculated as a weighted sum of the slope differences by considering the length of the lines which composes them.

**Implementation Notes**: Considering the length of the lines is so important. Since, while the length of the line increases, the accuracy of its slope also increases. This is coming from the error model of LADAR and the structure of Split and Merge Algorithm.

### 3.5.12.3 Finding the Translation in the Pose of the Robot

In this part, the translation in the pose of the robot is calculated. For this purpose, the resulting matching table is controlled whether there are any line pairs which have an angle difference above the given tolerance from the found rotation result in the pose. The ones above the given tolerance are set to zero in the matching table. After this step, the line pairs are divided into two groups to calculate translational difference in X and Y axis.

The lines with a slope difference below a threshold to the X axis line are found for the current scan. These lines are grouped two by two with the corresponding ones from the previous scan as they provide the slope difference between them below the predefined threshold. A vector is constructed by using the line to line distance and direction information between these line pairs. Constructed vector dataset is defined as *vector_x*. Same procedure is applied to for the lines with a slope difference below a threshold to the Y axis line. Constructed vector dataset is defined as *vector_y.* Vector summation of vector pairs, one taken from *vector_x* dataset and one taken from the *vector_y* dataset gives a good estimate for the translation vector of the robot.

This method uses a similar calculation with finding translation from the distance between matched corners from previous and current scan. However, this method eliminates the error coming from the composed error in position of corners. Since the distance is taken from the center of matched lines and also the vectors have a direction difference above a level between each other. Further information is given in The Details of Proposed Method for Translation part. The algorithm which finds the translational difference for the line pairs below slope difference threshold with X axis is given in Table 3.28:

**Table 3.28: The algorithm that finds the translation vector from the line pairs below slope difference threshold with X axis.**

```
Algorithm_Translational_Difference_VectorX ($C_{curr}$, $C_{prev}$, matched_LP, result_rotation, Threshold)

function trans_acc2rot(point1, rotation1) // transfers the cartesien coordinates of point1 according to given rotation in cylindirical domain
function comp_dist2line(line1,point1) // calculates the point1 to line1 distance
retrieve $\langle LineID_{prev}, LineID_{curr} \rangle$ from matched_LP
rotation=result_rotation
count=1;
for j= 1 to to size of $LineID_{prev}$
        $k = LineID^{j}_{prev}$
        $m = LineID^{j}_{curr}$
        if $| C^{[k]}_{prev}.L.slope - C^{[m]}_{curr}.L.slope + rotation | < Thresh1$ and ...
        $((C^{[m]}_{curr}.L.slope < 60)$ or $(C^{[m]}_{curr}.L.slope > 300)$ or $(120 < C^{[m]}_{curr}.L.slope < 240))$
                $LineIDx^{[count]}_{prev} = LineID^{j}_{prev}$
                $LineIDx^{[count]}_{curr} = LineID^{j}_{curr}$
```

**Table 3.28 continued**

```
12:            mid_point=trans_acc2rot ( mean of $C_{prev}^{[k]}.L.endpo\text{int}$, rotation)
13:            $dist\_linx^{[count]}$ = comp_dist2line ($C_{curr}^{[m]}.L$ ,mid_point)
14:            Store $\langle dist\_linx^{[count]}, LineIDx_{prev}^{[count]}, LineIDx_{curr}^{[count]} \rangle$ to $vector\_x^{[count]}$
15:            Store $\langle slope\_x^{[count]} = C_{curr}^{[m]}.L.slope + 90 \rangle$ to $vector\_x^{[count]}$
16:            count=count+1;
17:        endif
18: endfor
19: return vector_x
```

Same algorithm is used for the line pairs below slope difference threshold with Y axis. So the if statement inside the pseudo code became as:

**Table 3.29: The algorithm that finds the translation vector from the line pairs below slope difference threshold with Y axis.**

```
Algorithm_Translational_Difference_VectorY ($C_{curr}$, $C_{prev}$ ,matched_LP,
result_rotation, Threshold)
.................
11:    if $| C_{prev}^{[k]}.L.slope - C_{curr}^{[m]}.L.slope + rotation | < Thresh1$ and ...
       (($C_{curr}^{[m]}.L.slope$ < 210) ... or ($C_{curr}^{[m]}.L.slope$ >300) || (30< $C_{curr}^{[m]}.L.slope$ < 150))
.................
14:            Store $\langle dist\_liny^{[count]}, LineIDy_{prev}^{[count]}, LineIDy_{curr}^{[count]} \rangle$ to $vector\_y^{[count]}$
15:            Store $\langle slope\_y^{[count]} = C_{curr}^{[m]}.L.slope + 90 \rangle$ to $vector\_y^{[count]}$
.................
19:return vector_y
```

After *vector_x* and *vector_y* dataset has been found, the translation vector candidates are calculated by taking one from *vector_x* dataset and one from *vector_y* dataset. For the calculation first the vector pairs that have a direction difference below a given threshold are eliminated. This is important for the accuracy of translation calculation. From the succeeded ones, translation vector candidates are calculated with the direction and magnitude information. The algorithm which finds these is given in Table 3.30:

**Table 3.30: Algorithm calculating the translational motion of the robot from the translation calculated translation vectors found in Table 3.28 and Table 3.29.**

```
Algorithm_Translational_Change(vector_x, vector_y, $C_{curr}$, $C_{prev}$)
function trans_acc2rot(point1, rotation1) // transfers the cartesien coordinates of point1 according to given rotation in cylindirical domain
function comp_dist2line(line1,point1) // calculates the point1 to line1 distance
retrieve $\langle dist\_linx, slope\_x, LineIDx_{prev}, LineIDx_{curr} \rangle$ from vector_x
retrieve $\langle dist\_liny, slope\_y, LineIDy_{prev}, LineIDy_{curr} \rangle$ from vector_y
for i= 1 to size of $slope\_x$
    for j= 1 to size of $slope\_y$
        diff= $slope\_x^{[i]} - slope\_y^{[j]}$
        if $|diff| > Thresh1$
            $\theta = arctan\left( \frac{dist\_liny^{[j]} - dist\_linx^{[i]} * \cos(diff)}{dist\_linx^{[i]} * \sin(diff)} \right)$
            $const = \left| \frac{dist\_linx^{[i]}}{cos(diff)} \right|$
            $translation = \left[ const * cos\left(slope\_x^{[i]} + \theta - 90\right) \quad const * sin\left(slope\_x^{[i]} + \theta - 90\right) \right]$
            IDx=[ $LineIDx_{prev}^{[i]}$   $LineIDx_{curr}^{[i]}$ ]
            IDy=[ $LineIDy_{prev}^{[i]}$   $LineIDy_{curr}^{[i]}$ ]
            L1=trans_acc2rot ( mean($C_{prev}^{[IDx(1)]}$.L.endpoint),...
            result_rotation)+translation
            L2=trans_acc2rot (mean($C_{prev}^{[IDy(1)]}$.L.endpoint), ...
            result_rotation)+translation
            if comp_dist2line (L1, $C_{curr}^{[IDx(2)]}$.L ) < Threshold1 and...
            comp_dist2line(L2, $C_{curr}^{[IDy(2)]}$.L ) < Threshold1
                $\text{trans}^{[\text{count}]} = \text{translation}, \text{ID1}^{[\text{count}]} = \text{IDx}, \text{ID2}^{[\text{count}]} = IDy$
                store $\langle \text{trans}^{[\text{count}]}, \text{ID1}^{[\text{count}]}, \text{ID2}^{[\text{count}]} \rangle$ to $translations^{[count]}$
                count=count+1
            endif
        endif
    endfor
endfor
return translations
```

After the translation vector candidates are found, the final translation is calculated like a method given in Table 3.27. The pseudo-code of the algorithm is:

**Table 3.31: Algorithm calculating the final translation of the robot from the translation vectors by using outlier rejection methods.**

```
Algorithm_TranslationofTheRobot ($C_{curr}$, $C_{prev}$, translations, Threshold)
deviation=$\infty$;
retrieve $\langle \text{trans}, \text{ID1}, \text{ID2} \rangle$ from translations
count= number of members in ID1
while (deviation > Threshold1 and count> Threshold2 )
      count= number of members in ID1
      $j = ID1_{prev}^{[i]}$
      $jj = ID1_{curr}^{[i]}$
      $res^{[i]} = \sqrt{C_{prev}^{[j]}.L.length * C_{curr}^{[jj]}.L.length}$
      $result\_translation = \frac{\sum_{i=1}^{count}(translation^{[i]} * res^{[i]})}{\sum_{i=1}^{count} res^{[i]})}$
      C=max(|translation - result_translation|)
      find Index of C in max(|translation - result_translation|)
      store...
      $\left\langle \begin{matrix} translation^{[1:Index-1]}, translation^{[Index+1:count]}, ID1^{[1:Index-1]}, ID1^{[Index+1:count]}, \\ ID2^{[1:Index-1]}, ID2^{[Index+1:count]} \end{matrix} \right\rangle$
      to translations
       count=count-1
      deviation=max(|translation - result_translation|);
endwhile
return result_translation
```

As you can see from the given method in Table 3.31; the translation in the pose is calculated recursively, in which for each step, the outlier translation which causes the biggest deviation above the defined threshold is eliminated. Also the translation value is calculated by considering the length of the lines which composes these translations.

### The Details of Proposed Method for Translation

To find the translation in the pose between following scans the translation based on line distances between matched lines is used. It is seen that using corner points of the matched lines are easily affected from the small deviations in the slopes of the lines. This affect is illustrated in Figure 3.29:

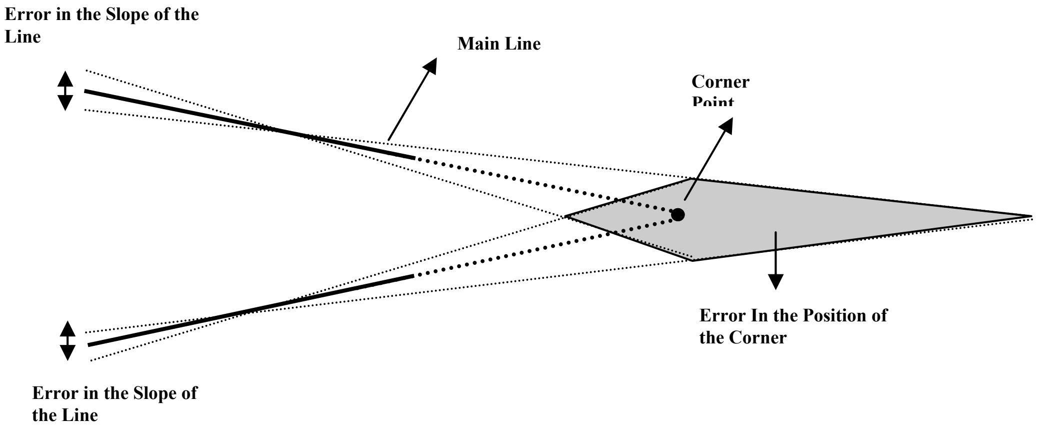

Error in the Slope of the Line
Main Line
Corner Point
Error In the Position of the Corner
Error in the Slope of the Line

**Figure 3.29: The effect of a small error in the slopes of lines constructing the corner point.**

As you can see from the figure; small changes in slope of lines highly affect the point they intersects. This error increases:

- When the length of the lines decreases
- When the angle between lines increases
- When extension of the lines increases (If the corner point is at the extensions of these lines)

As a result, the orientation change calculation based on corner points is influenced to a high noise. The steps of proposed method to solve this orientation problem are given below:
with a slope difference below a threshold to the Y axis line

1) Group the lines into two classes as the ones with a slope difference below a threshold to the X axis line (defined as close to X axis). and Y axis line (defined as close to Y axis).
2) Find the distance vector (*vector_x*) between the lines close to x-axis in previous and current scan
3) Find the distance vector (*vector_y*) between the lines close to y-axis in previous and current scan
4) Find the translation vector candidates from the distance vectors in x and y axis (*vector_x* and *vector_y*)

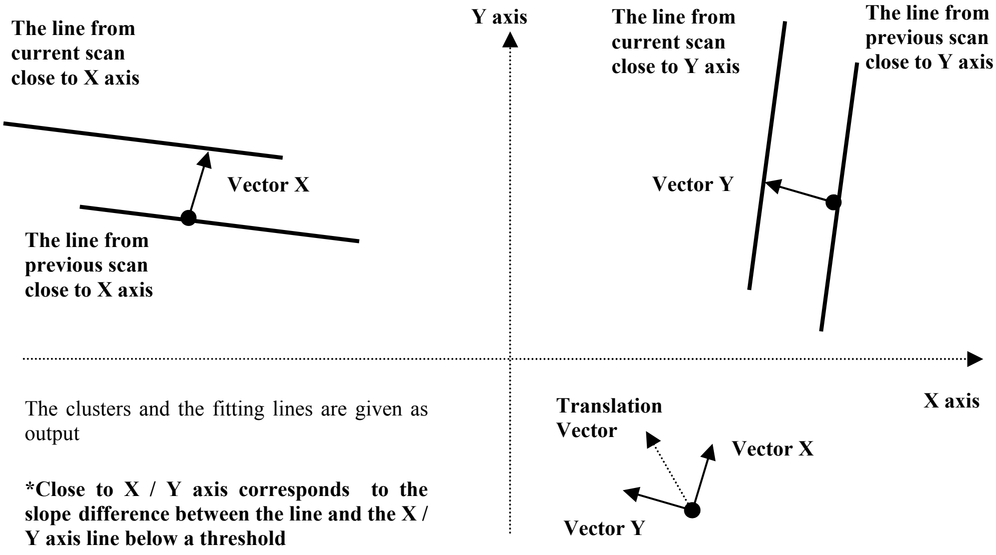

The line from current scan close to X axis
Vector X
The line from previous scan close to X axis
Y axis
The line from current scan close to Y axis
The line from previous scan close to Y axis
Vector Y
X axis
The clusters and the fitting lines are given as output
*Close to X / Y axis corresponds to the slope difference between the line and the X / Y axis line below a threshold
Translation Vector
Vector X
Vector Y

**Figure 3.30: The calculation of translation vector.**

5) Find the resulting translation, by eliminating the outliers in the translation vectors matrix, till the deviation is below a threshold value.

### 3.5.13 The Cases that Pose Difference Could Not Calculate

While the robot is going through an unstructured environment, such as the places where lines and corners could not found, the places where short and scattered lines are observed or long corridors…, rotation and translation difference could not be calculated. For these cases two different solutions are applied:

1) If the rotation and translation in the robot pose could not be found. The pose difference is taken from the odometer data. The pseudo code that finds this translation is given below:

**Table 3.32: The algorithm that finds the robot pose update based on the odometer data.**

```
Algorithm Odometer_Pose_Update (odometer,robot_pose)
translation_odo= $odometer_{curr}$ - $odometer_{prev}$  (x,y information)
rotation_odo= $odometer_{curr}$ - $odometer_{prev}$  ($\theta$ information)
translation= translation_acc2rot ( || translation_odo||, rotation_odo)
rotation= rotation_odo;  return translation,rotation
```

As it is seen from the pseudo code, the change in the ododata between following scans are calculated. The angle difference between the angle of the robot coming from scan matching in previous step and the angle of the robot coming from odometer in current step is found. The change in the ododata is transformed according to found angle difference.

**Implementation Notes:** Sometimes when the robot starts scan matching, the odometer data of the robot is not being initialized. So the coordinate axes of odometer and the scan matching does not overlap. To solve this problem, coordinate transformation is need to be made, which is also added in the code above.

2) If the robot goes through an environment that consists of lines in the same direction, such as long corridors, which contain long parallel lines, the rotation in pose of the robot could easily be calculated. Since all the lines belong to the cluster close to one axis (X-axis or Y-axis) and there is no line belong to the cluster close to other axis, translation is could not be calculated.

For this situation the rotation is calculated and translation is calculated from the odometer by using calculations given in Step 1**.**

### 3.5.14 Discussion about the Probabilistic Aspects of Scan Matching

Scan matching applied in this thesis is propagated in a deterministic way. However it has probabilistic aspects. In fact the matching table, which constructs the baseline of the algorithm, consist of votings based on the importance of applied criteria and overlap level between the feature sets in given criteria. Each criterion includes multiple votings inside and these are multiplied according to the importance of the criterion. Instead of continuous interval for votings as in probabilistic case, these are discredited in some levels. Also for the selection of matched line pairs, a gating is applied according to the general distribution of matched line pairs. The calculation of translation and rotation includes the lengths of line pairs used. Their effect to the final pose is engaged to the length of their line pairs. This can be explained in similar way, as a line length based distribution addition to pose calculation in probabilistic aspects. Furthermore the final pose is calculated recursively by the elimination of some candidates according to their difference from the mean. Also this elimination is stopped if the variance is below a given value. This can be interpreted as a probabilistic gating.

## 3.6 Fault Correction and Deterministic Global Localization Algorithm

In this part, some simple methods to solve probable pose update errors encountered during the navigation of the robot is presented in fault correction part. Also a simple method, which is tried as an initiate for the SLAM implementation, is given in next chapter. This method is named as deterministic global localization algorithm. The main aim in this algorithm is to remove the accumulated error in pose and obtain more consistent mapping.

### 3.6.1 Fault Correction:

Scan matching algorithms could encounter mismatches. This could be because of a change in pitch and roll of LADAR orientation, a turn faster than the algorithm could handle, unstructured environment or non reflective materials observed… In these cases scan matching algorithm can fails, which is resulted with a wrong pose difference update. In these cases, after the wrong pose obtained all the localization output and mapping goes on with the affect of this wrong update. For this purpose, the output of different sensors could be used as checkpoint for the main scan matching output. In our datasets, odometer output is available. Therefore, it is used to correct the pose difference if a jump in pose has occurred in main scan matching algorithm while it is not seen in synchronized odometer data. However the error tolerance between sensors and scan matching, the situations that correction is applied are needed to investigate carefully. For example, if a yaw rate gyro is used, whether an error is occurred in rotation can be checked. For skid steer robots, translation difference in scan matching can be checked with odometer while the robot is not turning. In fact, a correction is applied to pose difference according to the accuracy of used sensor. In these datasets, the fault check and correction based on odometer data is implemented as an option but not used. The reason is to prevent algorithms to be dependent to odometer data and also to make improvement process unaffected from odometer data.

### 3.6.2 Deterministic Global Localization (DGL):

As we know Scan Matching is an algorithm based on the matching of following scans. As like odometer, the error is accumulated. As the distance the robot has gone increases, the accumulated error also increases. To solve this, a low level SLAM initiate tried to be added. If we have able to track some reliable features in these following scans, we can detect the mismatches in scan matching when it has occurred. For this purpose an algorithm is formed, which tracks stable features and makes a correction when a jumpy drift has occurred in the scan matching. The algorithm consists of two parts, the correction based on real corners and correction based n both side ended lines.

#### 3.6.2.1 Correction Based on Real Corners

Real corners are good features for tracking and making global corrections on the pose of the robot. The advantages of these features are:

- The position difference between the matched corners could directly be used for finding the correction in translation error of the robot.
- Since the real corners consist of constructing line pairs, the difference between the slopes of matched line pairs could directly be used for finding the correction in rotation error of the robot.
- Since real corners consist of directly intersection of the line pairs, not from their extensions, the error in position of the corner is much less.

For this purpose an algorithm is constructed consisting of two parts: Finding stable features and make correction based on these features. Finding stable features is an important part since factitious lines could be extracted from line extraction algorithm and these factitious lines could construct factitious corners. Before sending the real corner to the correction part, we wait for to be seen multiple times. The pseudo code for finding stable corner features is given below:

**Table 3.33: The pseudo code that eliminates cognitive real corners.**

```
Algorithm_Stable_Corners( $Cnr_{curr}$ , robot_pose, Threshold)
function transform_global( point1, pose)  // trans fer  the point1 in local coordinates according to the given pose information
retrieve $\langle Cnr\_Real\_Memory, Cnr\_Count \rangle$ from memory
$Cnr\_Real$ = the members of $Cnr_{curr}$.Class which equals "Real Corner"
for i=1 to number of members in $Cnr\_Real$
    $Cnr\_Real^{[i]}.xy$ =transform_global( $Cnr\_Real^{[i]}.xy$ , robot_pose)
    for j=1 to number of members in $Cnr\_Real\_Memory$
        if distance between ( $Cnr\_Real^{[i]}.xy$, $Cnr\_Real\_Memory^{[j]}.xy$ ) <...
        Threshold1
            if slope difference between ( $Cnr\_Real^{[i]}.slope$ ,...
            $Cnr\_Real\_Memor.^{[j]}.slope$ ) < Threshold2
                $Cnr\_Count^{[i]}$ = $Cnr\_Count^{[i]}$ +1
            else
                Add $Cnr\_Real^{[i]}$ to $Cnr\_Real\_Memory$
            endif
        endif
    endfor
endfor
for i=1 to number of members in $Cnr\_Real\_Memory$
    if ( $Cnr\_Count^{[i]}_{prev} = Cnr\_Count^{[i]}_{curr}$ )
        Delete ( $Cnr\_Real\_Memory^{[i]}$ )
    endif
endfor
store $\langle Cnr\_Real\_Memory, Cnr\_Count \rangle$ to memory
stable_Cnr= members of $Cnr\_Real\_Memory$ that verifies
  $Cnr\_Count^{[i]} > Threshold3$
return stable_Cnr
```

The steps of pseudo code given in Table 3.33 are:

1) Compare real corners (corners) with corners in memory. If they are not seen in memory, add them to memory. If they are seen in memory, increment their count.

2) Give the corners as output that have counts above the threshold and do not be decided to delete in that step.

3) If the corners in memory are not seen in real corners for that step, delete the unseen ones from memory.

The pseudo code to make correction based on these corner features is given in Table 3.34:

**Table 3.34: The algorithm that calculates global correction for the robot pose if available.**

```
Algorithm_Corrrection (stable_Cnr,Threshold)
retrieve $\langle dist\_traveled, Cnr\_Stable\_Memory \rangle$ from memory
for i=1 to number of stable_Cnr
    for j=1 to number of $Cnr\_Stable\_Memory$
        if distance between ($stable\_Cnr^{[i]}.xy$, $Cnr\_Stable\_Memory^{[j]}.xy$) <...
        Threshold1
            if slope difference between ($stable\_Cnr^{[i]}.slope$, ...
            $Cnr\_Stable\_Memory^{[j]}.slope$) < Threshold2
                if $dist\_traveled^{[j]}$ >Threshold3
                    $found\_Cnr^{[count]}$ =[$stable\_Cnr^{[i]}$ $Cnr\_Stable\_Memory^{[j]}$]
                    reset $dist\_traveled^{[j]}$
                    count=count+1
                else
                    $found\_flag^{[i]}$ =1
                endif
            endif
        endif
    endfor
endfor
for i=1 to number of stable_Cnr
    if $found\_flag^{[i]}$ ==0
        Add $stable\_Cnr^{[i]}$ to $Cnr\_Stable\_Memory$
    endif
endfor
$dist\_traveled = dist\_traveled + || scan\_mat\_pose_{curr} - scan\_mat\_pose_{prev} ||$
for i=1 to number of found_Cnr
    $correction^{[i]}$ = distance between ($found\_Cnr^{[i,1]}$ , $found\_Cnr^{[i,2]}$)
endfor
result_correction=$\frac{1}{m}\sum_{i=1}^{m} correction^{[i]}$
return result_correction
```

In the correction part, given in pseudo code in Table 3.34,

1) The incoming stable corners (*stable_Cnr* ) from finding stable features part are controlled with the corners in the memory ( $Cnr\_Stable\_Memory$ ) based on

- The distance between the corners.
- The slope difference between the line pairs constructing the corners.
- The slope difference between the lines in same side of the corners.
- The distance travelled (*dist_traveled*) when the corner in the memory is seen again.

2) If the input corner matches any of the corners in the memory according to the rules given in Step 1, the position and slope difference is taken and added to the *correction* matrix. If not, the corner is added to the corners in memory.

3) If multiple correction values are found the mean of these values is calculated and pose correction is given as *result_correction.*

**Implementation Notes:** Adding the distance travelled (*dist_traveled*) is important. By this property after a stable corner is first seen it could make correction after the robot goes a distance above given threshold (*Threshold3).* Since the corner is seen multiple times in consecutive scans this property prevents continuous corrections after each time the corner is seen. The aim is to make a correction after the loop in the robot path is closed. Continuous corrections based on same corner makes the robot localization oscillatory. In fact, in continuous correction case the robot pose will be dependent on the correction coming from an corner instead of total scan matching solution given in 3.5.12.2 and 3.5.12.3.

### 3.6.2.2 Correction Based on Both Side Ended Lines

Both side ended lines are good features for tracking and making global corrections on the rotation of the robot. They have a unique length that is not affected from the position of the robot. Therefore after a long run it can be still easily detected because of its length. For this purpose, first both side ended lines are tracked in cognitive scans. This is a kind of gating and it protects the rotation calculation part from the phantom features, which decreases the wrong matching chance and also the workload on the rotation calculation part. In rotation calculation part, stable lines incoming are checked with the ones in memory according to predefined tolerances and in case of matching, corrections in pose is made.

**Table 3.35: The pseudo code that eliminates cognitive both side ended lines.**

```
Algorithm_Stable_Lines($C_{curr}$ ,robot_pose,Threshold)
retrieve $\langle L\_BSE\_Memory, L\_Count \rangle$ from memory
$L\_BSE$ = the members of $C_{curr}$.L.Lineclass which equals “Both Side ended”
for i= 1 to number of $L\_BSE$
    $L\_Pos^{[i]}$ =transform_global( mean( $L\_BSE^{[i]}.L.endpo\text{int}$ ), robot_pose)
    for j=1 to number of $L\_BSE\_Memory$
        if distance between ( $L\_Pos^{[i]}$ and ...
        $mean(L\_BSE\_Memory^{[j]}.L.endpo\text{int})$ )< Threshold1
            if slope difference between ( $L\_BSE^{[i]}$.L.slope and...
            $L\_BSE\_Memory^{[j]}$.L.slope ) < Threshold2
                $L\_Count^{[i]} = L\_Count^{[i]} + 1$
            else
                Add $L\_BSE^{[i]}$ to $L\_BSE\_Memory$
            endif
        endif
    endfor
endfor
for i= 1 to number of $L\_BSE\_Memory$
    if ( $L\_Count^{[i]}_{prev} = L\_Count^{[i]}_{curr}$ )
        Delete ( $L\_BSE\_Memory^{[i]}$ )
    endif
endfor
store $\langle L\_BSE\_Memory, L\_Count \rangle$ to memory
stable_Lines= members of $L\_BSE\_Memory$ that verifies ...
  $L\_Count^{[i]} > Threshold3$
return stable_Lines
```

The steps of pseudo code are:

1) Compare both side ended lines with lines in memory.If they are not seen in memory add to memory.If they are seen in memory increment their count.
2) Give the lines that have count above a threshold and do not be decided to delete.
3) If the lines in memory do not see in both side ended lines, delete the unseen ones from memory.

The pseudo code to make rotation correction based on these corner features is given below:

**Table 3.36: The algorithm that calculates global rotational correction for the robot pose.**

```
Algorithm_Rotation_Corrrection (stable_Lines,Threshold)
retrieve $\langle dist\_traveled, L\_Stable\_Memory \rangle$ from memory
for i= 1 to number of stable_Lines
    for j=1 to number of $L\_Stable\_Memory$
        if distance between (mean($stable\_Lines^{[i]}.endpoint$), ...
        mean($L\_Stable\_Memory^{[j]}.endpoint$)) < Threshold1
            if slope difference between ($stable\_Lines^{[i]}$.slope,...
            $L\_Stable\_Memory^{[j]}$.slope) < Threshold2
                if $dist\_traveled^{[j]}$ >Threshold3
                    $found\_L^{[count]}$=[$stable\_Lines^{[i]}$ $L\_Stable\_Memory^{[j]}$]
                    reset $dist\_traveled^{[j]}$
                    count=count+1
                endif
                $found\_flag^{[i]}$=1
            endif
        endif
    endfor
    if $found\_flag^{[i]}$=0
        Add $stable\_Lines^{[i]}$ to $L\_Stable\_Memory$
    endif
endfor
$dist\_traveled = dist\_traveled + \| scan\_mat\_pose_{curr} - scan\_mat\_pose_{prev} \|$
for i=1 to number of found_L
    rotation_correction$^{[i]}$ = slope difference between ($found\_Cnr^{[i,1]}$ and ...
    $found\_Cnr^{[i,2]}$)
endfor
result_rotation_correction=$\frac{1}{m}\sum_{i=1}^{m} rotation\_correction^{[i]}$
return result_rotation_correction
```

In the correction part, given in pseudo code above,

1) The incoming stable lines (*stable_Lines* ) from finding stable features part are controlled with the lines in the memory ( $Line\_Stable\_Memory$ ) based on

- The distance between the center point of lines
- The slope difference between the lines

- The distance traveled (*dist_traveled*) when the line corner in the memory is seen again.

2) If the input line matches any of the lines in the memory according to the rules given in Step 1, the slope difference is taken and added to the *rotation_correction* matrix. If not the corner is added to the lines in memory.

3) If multiple correction values are found the mean these values are calculated and rotation correction in the pose is given as *result_rotation_correction.*

## 3.7 Occupancy Grid Mapping

For the demonstration of scan matching results, occupancy grid mapping is used. Brasenham Line Algorithm [23] is used for the calculation of the empty grid cells before the LADAR scan hit point. For occupancy grid mapping the hit point of LADAR range scan with a triangular density function in 2D plane is applied. But it is seen that to use such a function increase the thickness of the border lines, which makes the analysis of performance of the algorithm harder. In performance tests, thin border lines are needed, since if any error occurs it can be observed by the change in position or slope of the border line. Therefore this mapping process downgraded to the deterministic level. Also the probability of an occupied cell to be unoccupied is highly degraded.

## 3.8 Chapter Summary

In this chapter, scan matching algorithm is presented. Scan matching algorithm takes LADAR and odometer data as an input and gives area map and pose information as an output. At the beginning, LADAR data is processed by a line extraction algorithm, which is known as Split and Merge algorithm. Resulting line segments constructs the base for feature and feature properties set for main scan matching part. These features are line segments and corners and patterns consist of their multiple sets. As the feature properties, type of line ends, type of corners, distance between parallel lines, unique lines with longer length, patterns consisting of consecutive corner distances… etc. are used. These feature set and properties taken from consecutive LADAR scans are evaluated according to their overlap level by different criteria. This evaluation is resulted as votings between the initial line pairs constructing these features from previous and current scan in matching table. By the evaluation of votes in matching table, matched line pairs are obtained. Between these line pairs transformation vectors are defined. By using vector addition the translation and rotation change between consecutive scans is calculated. Summation of these differences gives the pose of the robot.

During the incremental pose estimation, also the error in pose is accumulated. For this purpose a DGL algorithm is added. This algorithm uses both side ended lines and real corners as landmarks. All these landmark candidates are passed through a gating to prevent comprising of phantom landmarks.  When a landmark is observed again, such as in closed loop cases, the pose of the robot is updated according the accumulated error between the initial and last positions of landmarks. This gives an error correction in a considerable level for the pose of the robot.

During the implementation of global localization part, it is seen that the localization accuracy utmost descends to the pose error level of landmark that is initially seen. Besides, no correction is given to estimated position and orientation of landmarks in memory. It results as the permanence of pose error during all the path of the robot. After an accumulated error level with the instant one coming from scan matching, it precludes the association of landmarks and observed landmarks are detected as new landmarks.  As a result of evaluation of these problems, FastSLAM algorithm is chosen to be implemented to solve these.

# CHAPTER 4

# SIMULTANEOUS LOCALIZATION AND MAPPING

In this thesis, simultaneous localization and mapping (SLAM) algorithm is also implemented to yield global localization. From SLAM algorithms, FastSLAM 2.0 is chosen since it solves the problems coming from DGL implementation mentioned in summary part of Chapter 3. Furthermore, its advantages to EKF SLAM algorithms are mentioned in Section 2.3 of Literature Survey part. In this chapter, FastSLAM is added at the output of main scan matching algorithm. Addition of FastSLAM gives the ability to eliminate the accumulated pose error coming from scan matching algorithm, on the other hand scan matching algorithm gives a better pose information (than odometer) which makes the data association problem easier, rate of wrong associations decreases and as a result, the performance of FastSLAM increases.

In this part, standard FastSLAM 2.0 algorithm and a modified version are implemented. For the prediction step, odometer motion model is used. Landmarks are extracted from the feature set used in scan matching part. In data association, particle weight calculation and feature update parts, multiple positions of landmark is used to add orientation information of landmark into calculations. In modified version of FastSLAM algorithm, single position of landmark is used, but also orientation information of landmark is added as a parameter into the EKF filter.

## 4.1 Structure of the FastSLAM Algorithm Implemented

In this part, the structure of the FastSLAM algorithm implemented will be described. The appropriate illustration describing the general structure of the algorithm is given in Figure 4.1 and pseudo code giving its implementation is given in Table 4.1.

FastSLAM algorithm is a combination of particle filter and Extended Kalman filter (EKF). All the information about the localization of the robot and map of the area is kept in particles. In fact, for each particle EKF estimates the pose of the robot and landmarks with their covariance. So the structure of particle consist of the current and previous poses of the robot, the pose error covariance matrix, the orientation of landmarks and their covariance matrix and the weight of the given particles in pose estimation of the robot. The structure that particle contains can be given as:

$$particle^{[m]}.xv = \lfloor x_p{}^{[m]}, y_p{}^{[m]}, \theta_p{}^{[m]} \rfloor \tag{4-1}$$

Where $x_p{}^{[m]}, y_p{}^{[m]}, \theta_p{}^{[m]}$ are the pose estimation of the robot for particle *m,* they can be defined as vehicle states;

$$particle^{[m]}.xf = \begin{bmatrix} lm_x{}^{[m,1]} & lm_x{}^{[m,2]} & lm_x{}^{[m,3]} & ..... & lm_x{}^{[m,n]} \\ lm_y{}^{[m,1]} & lm_y{}^{[m,2]} & lm_y{}^{[m,3]} & ..... & lm_y{}^{[m,n]} \end{bmatrix} \tag{4-2}$$

Where $lm_x{}^{[m,n]}$ are the pose estimation of landmark *n* of the particle *m.* $particle^{[m]}.pv$ is the pose 3 x 3 covariance matrix of $particle^{[m]}.xv$ and $particle^{[m]}.pf^{[n]}$ is the 2 x 2 covariance matrix of for each landmark, that is $particle^{[m]}.xf^{[n]} = \begin{bmatrix} lm_x{}^{[m,n]} \\ lm_y{}^{[m,n]} \end{bmatrix}$.

The assumed noise in each step for $particle^{[m]}.xv$, which is known as process noise, is given as:

$$R = \begin{bmatrix} \varepsilon_{rot1} & 0 & 0 \\ 0 & \varepsilon_{trans} & 0 \\ 0 & 0 & \varepsilon_{rot2} \end{bmatrix} \tag{4-3}$$

Where $\varepsilon_{rot1}$, $\varepsilon_{trans}$ and $\varepsilon_{rot2}$ are the zero order Gaussian noises, in order, for the first rotation $\delta_{rot1}$, the translation $\delta_{trans}$ and second rotation $\delta_{rot2}$ for the odometer motion model given in Section 4.2 . $\delta_{rot1}$, $\delta_{trans}$, $\delta_{rot2}$ are the control inputs. The assumed noise for each step for $particle^{[m]}.pv$, which is known as measurement noise, is given as:

$$Q = \begin{bmatrix} \varepsilon_d & 0 \\ 0 & \varepsilon_\theta \end{bmatrix} \tag{4-4}$$

Where $\varepsilon_d$ and $\varepsilon_\theta$ are the zero order Gaussian noises, in order, for distance, and angular orientation of the landmark referenced to pose of the robot.

In the algorithm, first control inputs ($\delta_{rot1}$, $\delta_{trans}$, $\delta_{rot2}$) according to odometer motion model is calculated from the incoming pose difference from scan matching (Δx, Δy, Δθ) **.** Pose of the robot in each particle ( $particle^{[m]}.xv$) is updated according to control inputs and assumed pose noise (*R*). This is known as **prediction step** of EKF. When a new landmark is observed, the position of landmark is added ($particle^{[m]}.xf^{[n]}$) with its covariance ($particle^{[m]}.pf^{[n]}$). When a landmark is observed again, the pose of the robot (state update) and landmark positions (landmark update) are corrected, which is known as **measurement (correction) update** step of EKF.

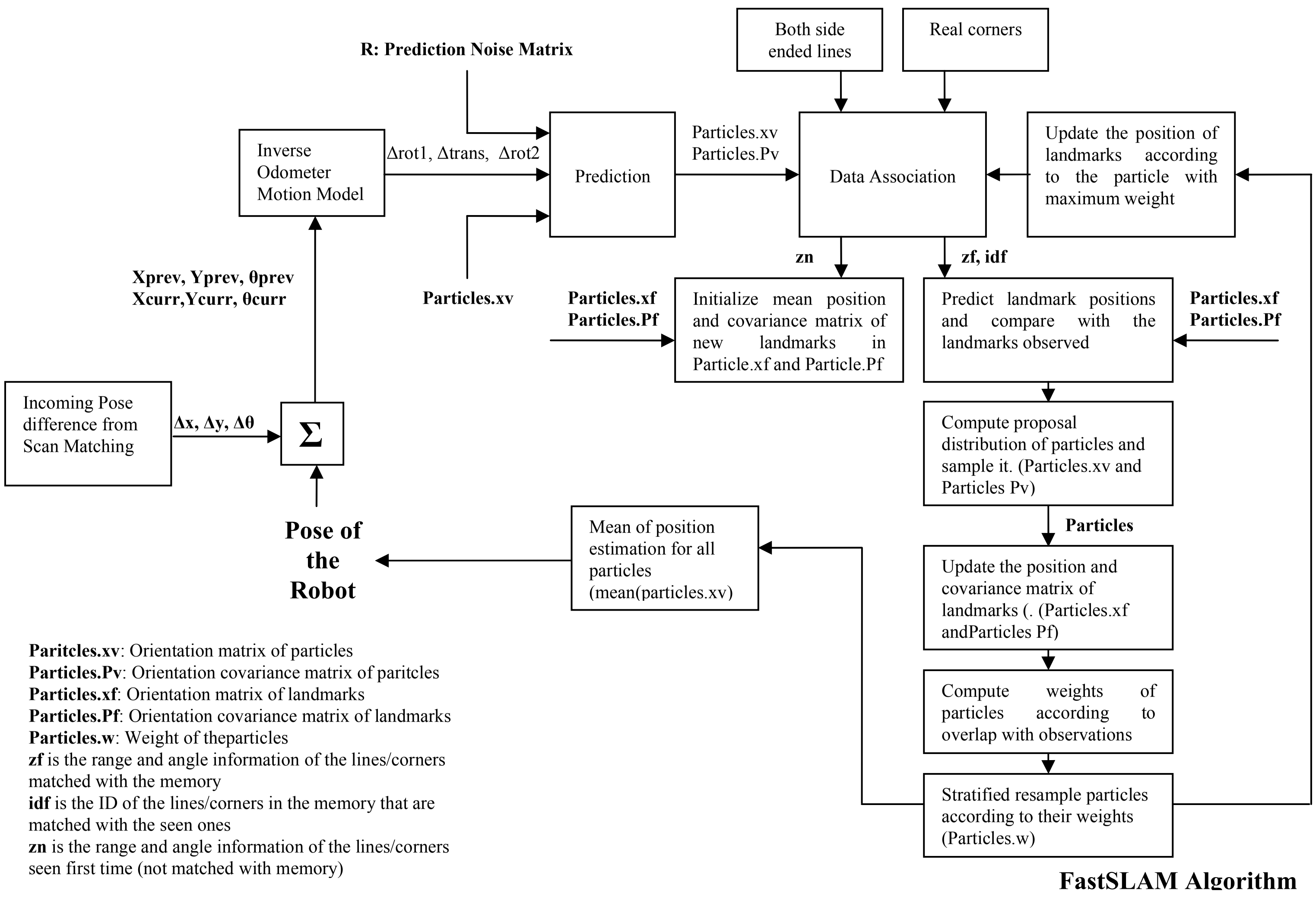
In FastSLAM 2.0 algorithm, in the measurement update, first the proposal distribution in particles is sampled according to the observed landmark, which gives better particle diversity. Then the weights of the particles are recalculated. This is done according to the overlap between predicted landmark position and observed landmark. The weight shows the accuracy of estimated robot path and landmark positions calculated for that particle. The surviving particles are resampled in proportion to their weights as they give predefined number of particles. This gives better particle diversity. After this step the landmark orientations and their covariance are updated same as the measurement update part of EKF. In Table 4.1, the steps of the FastSLAM algorithm are given as a pseudo code and in Figure 4.1, the general structure of FastSLAM algorithm is given as a schema:

**Table 4.1: The pseudo code giving the implementation of FastSLAM algorithm.**

```
Algorithm_FastSLAM (particle, $robot\_pose_{prev}$, $robot\_pose_{curr}$, $C_{curr}.L$, $Cnr_{curr}$)

take FastSLAM paremeters,threshold and noise values (Q,R) from configuration file
[$\delta_{rot1}$, $\delta_{trans}$, $\delta_{rot2}$]=inverse_odometer_motion($robot\_pose_{prev}$, $robot\_pose_{curr}$);
particle =FastSLAM_Prediction (particle, $\delta_{rot1}$, $\delta_{trans}$, $\delta_{rot2}$ ,Q);
stable_Lines=Stable_Lines($C_{curr}.L$, $robot\_pose_{curr}$ ,Threshold);
[znL ,zfL ,idfL]=FastSLAM_DataAssociation_Lines (stable_Lines,Threshold,...
    $robot\_pose_{curr}$)
stable_Corners=Stable_Corners($Cnr_{curr}$, $robot\_pose_{curr}$ ,Threshold);
[znC, zfC ,idfC]=FastSLAM_DataAssociation_Corners (stable_Corners, Threshold,
    $robot\_pose_{curr}$);
zn=[ znL  znC]; zf=[ zfL  zfC]; idf=[ idfL  idfC];
if zn is not empty
    particle =Landmark_Initialization (particle, zn, R)
endif
if  zf is not empty
    particle= Sample_Proposal (particle, Hv, Hf, Sf);
    particle= Feature_Update (particle, Hv, Hf, Sf,Q);
    particle= ComputeWeight (particle, zf, idf, R);
    particle=Particle_Resampling (particle);
endif
[$robot\_pose_{curr}$]= average of robot pose estimated from the most weighted
particles
```

As it is given in Table 4.1, by using **inverse odometer motion** function control states are found and the pose is updated according to control states by ***FastSLAM_Prediction*** functions.***Stable_Lines,FastSLAM_DataAssociation_Lines,Stable_Corners,FastSLAM_DataAssociation_Corners*** functions form the data association part. For new observed landmark ***Landmark_Initialization*** function is applied. For re-observed landmarks, ***Sample_Proposal, Feature_Update, ComputeWeight, Particle_Resampling*** functions are applied, which forms the measurement update part.


R: Prediction Noise Matrix
Both side ended lines
Real corners
Inverse Odometer Motion Model
Δrot1, Δtrans, Δrot2
Prediction
Particles.xv Particles.Pv
Data Association
Update the position of landmarks according to the particle with maximum weight
Xprev, Yprev, θprev Xcurr,Ycurr, θcurr
Particles.xv
zn
zf, idf
Particles.xf Particles.Pf
Initialize mean position and covariance matrix of new landmarks in Particle.xf and Particle.Pf
Predict landmark positions and compare with the landmarks observed
Particles.xf Particles.Pf
Incoming Pose difference from Scan Matching
Δx, Δy, Δθ
Σ
Compute proposal distribution of particles and sample it. (Particles.xv and Particles Pv)
Particles
Pose of the Robot
Mean of position estimation for all particles (mean(particles.xv)
Update the position and covariance matrix of landmarks (. (Particles.xf andParticles Pf)
Compute weights of particles according to overlap with observations
Stratified resample particles according to their weights (Particles.w)
FastSLAM Algorithm

**Paritcles.xv**: Orientation matrix of particles
**Particles.Pv**: Orientation covariance matrix of paritcles
**Particles.xf**: Orientation matrix of landmarks
**Particles.Pf**: Orientation covariance matrix of landmarks
**Particles.w**: Weight of theparticles
**zf** is the range and angle information of the lines/corners matched with the memory
**idf** is the ID of the lines/corners in the memory that are matched with the seen ones
**zn** is the range and angle information of the lines/corners seen first time (not matched with memory)

**Figure 4.1: The figure illustrating the general structure of FastSLAM algorithm.**

## 4.2 Inverse Odometer Motion Model

Inverse odometer motion model takes current $(x_{prev}, y_{prev}, \theta_{prev}) = robot\_pose_{prev}$ and previous $(x_{curr}, y_{curr}, \theta_{curr}) = robot\_pose_{curr}$ pose of the robot (known as vehicle states) and calculates the rotation and translation parameters. (Previous pose is set in memory for each step and the current pose is calculated from the previous pose and $(\Delta x, \Delta y, \Delta \theta)$ values coming from the main scan matching part). The motion of the robot is assumed as a rotation, a straight line motion and another rotation. Decomposition of the parameters is the first rotation $\delta_{rot1}$, the translation $\delta_{trans}$ and second rotation $\delta_{rot2}$. These parameters are enough to describe the motion of the robot between two steps.

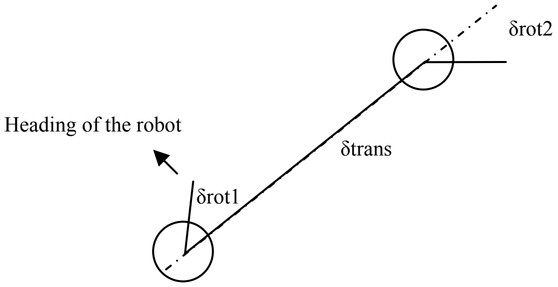


**Figure 4.2: Odometer model. The robot motion in the time interval (t-1,t) is approximated by the first rotation $\delta_{rot1}$, the translation $\delta_{trans}$ and second rotation $\delta_{rot2}$.**

The calculation of given parameters (known as control inputs) can be given with the equations as:

$$\delta_{rot1} = \arctan(\frac{y_{curr} - y_{prev}}{x_{curr} - x_{prev}}) - \theta_{prev} \quad \textbf{(4-5)}$$

$$\delta_{trans} = \sqrt{(y_{curr} - y_{prev})^2 + (x_{curr} - x_{prev})^2} \quad \textbf{(4-6)}$$

$$\delta_{rot2} = \theta_{curr} - \theta_{prev} - \delta_{rot1} \quad \textbf{(4-7)}$$

## 4.3 Prediction Step

In prediction step, the pose of the robot is estimated for each particle according to $[x_p \ , y_p, \theta_p]$ previous pose values registered in *particles.xv* and motion parameters $[\delta_{rot1}$, $\delta_{trans}$, $\delta_{rot2}]$. Mean of the pose is predicted according to the odometer motion model. The equations about the model can be given as:

$$x_p = x_p + \delta_{trans} \cos(\delta_{rot1} + \theta_p) \quad \textbf{(4-8)}$$

$$y_p = y_p + \delta_{trans} \sin(\delta_{rot1} + \theta_p) \quad \textbf{(4-9)}$$

$$\theta_p = \theta_p + \delta_{rot1} + \delta_{rot2} \quad \textbf{(4-10)}$$

**Table 4.2: The prediction step of FastSLAM algorithm.**

```
Algorithm_FastSLAM_Prediction (particle, $\delta_{rot1}$, $\delta_{trans}$, $\delta_{rot2}$, Q)
for i= 1: all particles
    $[x, y, \theta] = particle^{[i]}.xv$; $pv = particle^{[i]}.pv$;
    $\mathrm{xv} = \begin{bmatrix} x + \delta_{trans}\cos(\delta_{rot1} + \theta) \\ y + \delta_{trans}\sin(\delta_{rot1} + \theta) \\ \delta_{rot1} + \delta_{rot2} + \theta \end{bmatrix}$
    $\mathrm{Gv} = \begin{bmatrix} 1 & 0 & -\delta_{trans}\sin(\delta_{rot1} + \theta) \\ 0 & 1 & \delta_{trans}\cos(\delta_{rot1} + \theta) \\ 0 & 0 & 1 \end{bmatrix}$
    $\mathrm{Gu} = \begin{bmatrix} -\delta_{trans}\sin(\delta_{rot1} + \theta) & \cos(\delta_{rot1} + \theta) & 0 \\ \delta_{trans}\cos(\delta_{rot1} + \theta) & \sin(\delta_{rot1} + \theta) & 0 \\ 1 & 1 & 1 \end{bmatrix}$
    $pv = Gv\, pv\, Gv^T + Gu\, pv\, Gu^T$;
    $particle^{[i]}.xv = xv$; $particle^{[i]}.pv = pv$;
endfor
return particle.xv, particle.pv
```

The pseudo code for prediction calculation is given in Table 4.2. In Table 4.2, the Jacobian matrix *(Gv)*, the pose of the robot estimated in particle with respect to vehicle states can be calculated as:

$$Gv = \begin{bmatrix} \frac{\partial x_p}{\partial x_p} & \frac{\partial x_p}{\partial y_p} & \frac{\partial x_p}{\partial \theta_p} \\ \frac{\partial y_p}{\partial x_p} & \frac{\partial y_p}{\partial y_p} & \frac{\partial y_p}{\partial \theta_p} \\ \frac{\partial \theta_p}{\partial x_p} & \frac{\partial \theta_p}{\partial y_p} & \frac{\partial \theta_p}{\partial \theta_p} \end{bmatrix} \quad \textbf{(4-11)}$$

And the Jacobian matrix *(Gu)*, the pose of the robot estimated in particle with respect to control inputs can be calculated as:

$$Gu = \begin{bmatrix} \frac{\partial x_p}{\partial \delta_{rot1}} & \frac{\partial x_p}{\partial \delta_{trans}} & \frac{\partial x_p}{\partial \delta_{rot2}} \\ \frac{\partial y_p}{\partial \delta_{rot1}} & \frac{\partial y_p}{\partial \delta_{trans}} & \frac{\partial y_p}{\partial \delta_{rot2}} \\ \frac{\partial \theta_p}{\partial \delta_{rot1}} & \frac{\partial \theta_p}{\partial \delta_{trans}} & \frac{\partial \theta_p}{\partial \delta_{rot2}} \end{bmatrix} \quad \textbf{(4-12)}$$

## 4.4 Landmark Extraction and Data Association:

Landmark extraction and data association is one of the hardest steps in SLAM algorithms. From the feature set created, the landmarks are needed to be carefully defined. In fact, they should be so unique features that they are not wrongly associated to a previously seen landmark. Wrong association can be devastating as it means the robot will think it is somewhere different from where it actually is. As a result, the FastSLAM algorithm diverges.

In landmark extraction part, same features (both side ended lines and real corners) are used as landmarks. As a result, same extraction algorithms given for global localization part are used (given in Table 3.33 and Table 3.35); the one in Table 3.33 is for eliminating cognitive real corners and the one in Table 3.35 is for eliminating cognitive both side ended lines. These two steps extract the landmarks that will be used as reference in SLAM. In these steps also, the gating rule: A feature is not actually considered as a landmark worthwhile to be used in SLAM unless it is seen N times, is implemented. This rule eliminates feeding the algorithm with bad landmarks.

As it is seen from the algorithms, the landmarks include position information and orientation information. The position information is the center point of the line and the corner positions. The orientation information is the slope of the line and the slopes of the lines constructing the corner. In SLAM algorithm implementations based on LADAR data mostly uses just the position information of landmark, which is associated with the one in memory by using methods like maximum likelihood , nearest-neighbor approach [37] etc… In our implementation each landmark is fed to the landmark position matrix of particles (*particles.pf*) with two positions. This adds the orientation information of a landmark to FastSLAM algorithm and improves its performance.

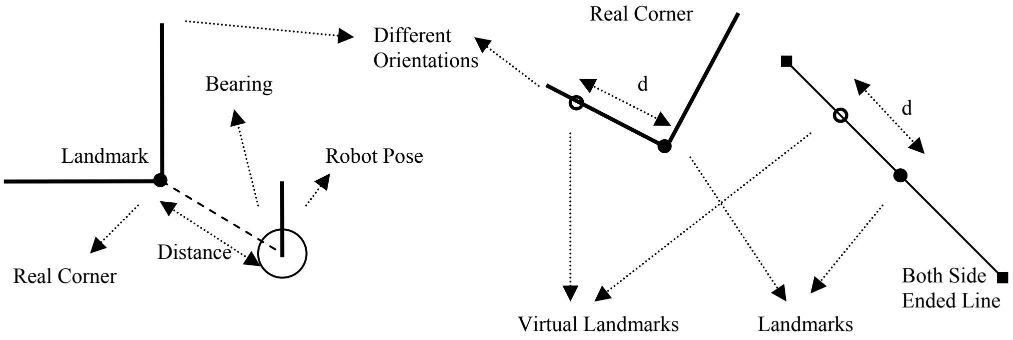


**Figure 4.3: Illustration showing the landmarks and their properties used in FastSLAM algorithm.**

When a landmark is first seen, the pose of the robot in particles ( *particle.xv)* is sampled from their proposal distribution ( *particle.pv)* and than the landmark is registered in particle given in equation

$$particle^{[i]}.xf^{[j]} = \begin{bmatrix} x_p^{[i]} + r\cos(\theta_p^{[i]} + b) \\ y_p^{[i]} + r\sin(\theta_p^{[i]} + b) \end{bmatrix} \quad \textbf{(4-13)}$$

where $[x_p\ , y_p, \theta_p]$ pose values registered in *j*th feature of *i*th particle, *r* is the landmark range information and *b* is the bearing information referenced to the robot pose. As a result the landmark information registered in particle consists of just the position of the landmark. Assume the case the robot sees the landmark first and its position is initialized at particles about the same (low variance in pose of the robot). When the particles see the observed landmark again, they can be weighted equally while they see it with different poses then which pose in the particle is close to real robot pose? The situation is illustrated in Figure 4.4:

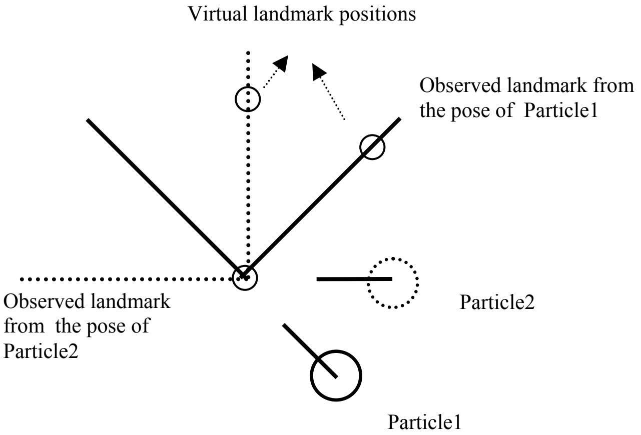

Virtual landmark positions
Observed landmark from the pose of Particle1
Observed landmark from the pose of Particle2
Particle2
Particle1

**Figure 4.4: The robot sees the observed landmark again and for Particle1 and Particle2 the position of the landmark overlaps with the position predicted. Therefore particles take same weight, which results in with wrong robot pose prediction.**

To prevent the situation given in Figure 4.4, two landmark positions (landmark and virtual one) are used, which force the algorithm to calculate the weights for both, which also promote the particles with the heading information close to real robot pose.

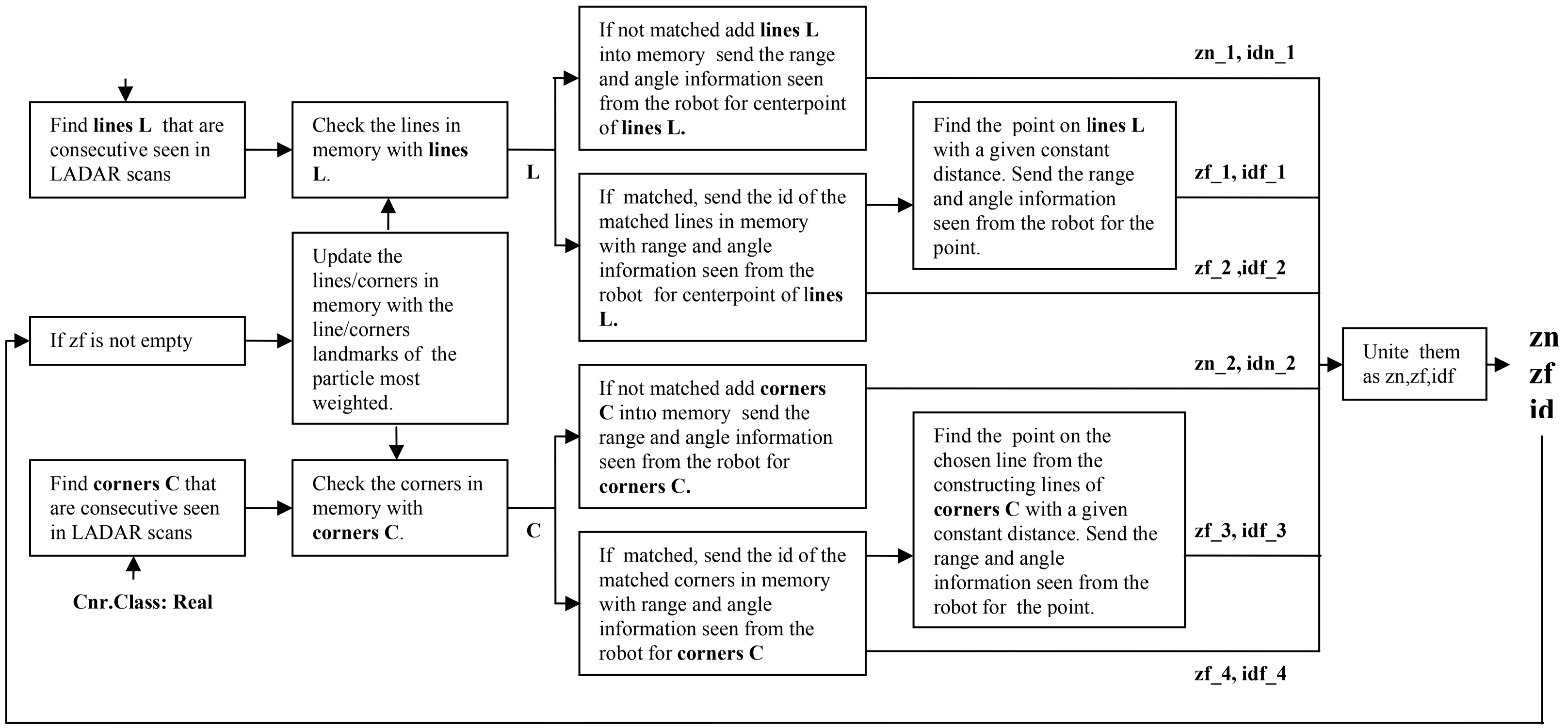
Slope of the lines constructing the landmark are also used in data association to separate them (which makes the data association robust to failures), but these slopes must also need to be updated with the position of landmarks to make correct data associations. An updated position of a landmark does not include any information about the slope of the lines constructing. By adding a virtual landmark position to FastSLAM module, correction in slope of the lines is also achieved.

The method using two landmark positions for each landmark gives an improvement when the environment does not have many landmarks. For the situations when different landmarks are seen with small steps, the particles with correct heading information are again promoted. However, for a SLAM algorithm making an association using just the landmark positions degrades the performance of the data association. In fact, in such a case algorithm needs landmarks separated from each other with longer distances, else the chance to make a mismatch between different ones increases.

In Figure 4.5, the general structure of data association part is given and in Table 4.3: The pseudo code for the implementation of this part is given. As we can see from Figure 4.5, first the landmark candidates are extracted from the feature set. These are the real corners (*Cnr.Class: Real)* and both side ended lines (*C.L.Lineclass: BothSide Ended Lines)*. These landmark candidates are controlled whether they are seen multiple times (to prevent burden on the algorithm with fake landmarks). Also it is controlled whether inside the area the landmark found; another landmark candidate with about the same properties is seen. If this happens, both landmark candidates are not accepted. This decreases the chance to make wrong data association between landmarks with close properties and distances.

The new observed landmarks are compared with previously seen ones, whether they could be the same feature. For the corners, this is controlled between incoming stable corners (*stable_Cnr*) from finding stable features part and the corners in the memory ($Cnr\_Stable\_Memory$) based on:

- The distance between the corners.
- The slope difference between the line pairs constructing the corners.
- The slope difference between the lines in same side of the corners.
- The distance travelled (*dist_traveled*) when the corner in the memory is seen again.


Find lines L that are consecutive seen in LADAR scans
Check the lines in memory with lines L.
L
If not matched add lines L into memory send the range and angle information seen from the robot for centerpoint of lines L.
If matched, send the id of the matched lines in memory with range and angle information seen from the robot for centerpoint of lines L.
Find the point on lines L with a given constant distance. Send the range and angle information seen from the robot for the point.
zn_1, idn_1
zf_1, idf_1
zf_2 ,idf_2
If zf is not empty
Update the lines/corners in memory with the line/corners landmarks of the particle most weighted.
Find corners C that are consecutive seen in LADAR scans
Cnr.Class: Real
Check the corners in memory with corners C.
C
If not matched add corners C intıo memory send the range and angle information seen from the robot for corners C.
If matched, send the id of the matched corners in memory with range and angle information seen from the robot for corners C
Find the point on the chosen line from the constructing lines of corners C with a given constant distance. Send the range and angle information seen from the robot for the point.
zn_2, idn_2
zf_3, idf_3
zf_4, idf_4
Unite them as zn,zf,idf
zn
zf
id

**zf** is the range and angle information of the lines/corners matched with the memory
**idf** is the ID of the lines/corners in the memory that are matched with the seen ones
**zn** is the range and angle information of the lines/corners seen first time (not matched with memory)

**Figure 4.5: The figure illustrating landmark extraction and data association part of FastSLAM algorithm**

For the lines, this is controlled between incoming stable lines (*stable_Lines* ) from finding stable features part and the lines in the memory ( *Line _ Stable _ Memory )* based on :

- The distance between the center point of lines.
- The slope difference between the lines.
- The distance traveled (*dist_traveled*) when the line corner in the memory is seen again.

If the landmarks are matched than the distance and bearing information ( *zf =[d b])* between the landmark in memory ( *L _ Stable _ Memory , Cnr _ Stable _ Memory )* and observed one (*stable_Lines, stable_Corners* ) is calculated. This information is given with the ID of landmark in memory (*idf*). If a new landmark is observed its position is set into memory, an ID is assigned and again calculated distance and bearing values ( *zn =[d b])* referenced to pose of the robot are given as output. When landmark update information is come to data association algorithm, all the landmark positions in memory are updated according to the landmark position estimates of most weighted particle.

**Table 4.3: The pseudo code giving the implementation of data association for both side ended lines.**

```
Algorithm_FastSLAM_DataAssociation_Lines (stable_Lines, Threshold, robot_pose, landmark_update_flag, particles)

$[x_r, y_r, \theta_r]$=robot_pose:
retrieve $\langle dist\_traveled, L\_Stable\_Memory \rangle$ from memory
if landmark_update_flag equals to 1
    Update $L\_Stable\_Memory$ according to particles.xf;
endif
for i= 1 to number of stable_Lines
    for j=1 to number of $L\_Stable\_Memory$
        if distance between (mean($stable\_Lines^{[i]}$.endpoint), ...
        mean($L\_Stable\_Memory^{[j]}$.endpoint)) < Threshold1
            if slope difference between ($stable\_Lines^{[i]}$.slope, ...
            $L\_Stable\_Memory^{[j]}$.slope) < Threshold2
                if $dist\_traveled^{[j]}$ >Threshold3
                    $found\_L^{[count]}$ =[$stable\_Lines^{[i]}$]
                    $found\_ID^{[count]}$ = j
                    reset $dist\_traveled^{[j]}$
```

**Table 4.3 continued**

```
                    count=count+1
                endif
                $found\_flag^{[i]}$=1
            endif
        endif
    endfor

// the step that gives landmark observation to landmarks if a new one is found
    if $found\_flag^{[i]}$=0
        Add $stable\_Lines^{[i]}$ to $L\_Stable\_Memory$
        d= mean($stable\_Lines^{[i]}.endpoint$) - $x_r$
        b= arctan (mean($stable\_Lines^{[i]}.endpoint - x_r$) - $stable\_Lines^{[i]}.slope$;
        $zn^{[2i-1]}$=[ d b]
        Offset= [cos($stable\_Lines^{[i]}.slope$) sin($stable\_Lines^{[i]}.slope$)]
        Extension_Point=d+ $x_r$ +Offset
        d2= Extension_Point - $x_r$
        $zn^{[2i]}$=[ d2 b]
    endif
endfor

// the step that gives landmark observation to landmarks if a one is reobserved
for i=1 to number of $found\_L$
    d= mean($found\_L^{[i]}.endpoint$) - $x_r$
    b= arctan (mean($found\_L^{[i]}.endpoint - x_r$) - $found\_L^{[i]}.slope$
    $zf^{[2i-1]}$=[d b]; $idf^{[2i-1]} = found\_ID^{[i]}$
    Offset= [cos($found\_L^{[i]}.slope$) sin($found\_L^{[i]}.slope$)]
    Extension_Point=d+ $x_r$ +Offset;
    d2= Extension_Point - $x_r$
    $zf^{[2i]}$=[ d2 b]; $idf^{[2i]} = found\_ID^{[i]} + 1$
endfor
$dist\_traveled = dist\_traveled + \| robot\_pose_{curr} - robot\_pose_{prev} \|$;
return zn,zf,idf
```

*FastSLAM_DataAssociation_Lines* algorithm given in Table 4.3 is also used for data association of real corners. Because of the high similarity between the algorithms the pseudo code for real corners is not given. In *FastSLAM_DataAssociation_Corners* algorithm *stable_Lines* are replaced by *stable_Corners*, $L\_Stable\_Memory$ *is* replaced by

*Cnr _ Stable _ Memory*, *found_L* is replaced by *found_Cnr.* In same manner, the observed landmarks are given in *zf* and *idf*, and new landmarks are given in *zn* as the output.

## 4.5 Landmark Observation (Augmentation):

When a landmark is observed for the first time, it is initialized in particles. The pseudo code of the initialization is given as:

**Table 4.4: The pseudo code giving the implementation of landmark initialization.**

```
Algorithm_FastSLAM_Landmark_Initialization (particle, zn, R)
lenz=number of landmarks set in particle.xf
for i=1: all particles
    $[x, y, \theta] = particle^{[i]}.xv$ ;
    for j= 1 to number of landmarks in zn
        [r,b]=zn    // r is the range and b is the bearing information
        $Gz = \begin{bmatrix} \cos(\theta+b) & -r\sin(\theta+b) \\ \sin(\theta+b) & r\cos(\theta+b) \end{bmatrix}$;
        $particle^{[i]}.xf^{[j+lenz]} = \begin{bmatrix} x + r\cos(\theta+b) \\ y + r\sin(\theta+b) \end{bmatrix}$ ;
        $particle^{[i]}.pf^{[j+lenz]} = Gz\, R\, Gz^T$ ;
    endfor
endfor
```

As it is given in Table 4.4; from the distance and bearing information *zn,* incoming from the data association part, which is distance and bearing information relative to robot pose estimated in particle, the landmark position in global coordinates is estimated and registered in *particle.xf* of that particle. Also covariance of landmark position is calculated as

$$particle.pf = Gz\, R\, Gz^T \quad \textbf{(4-14)}$$

Where *Gz* corresponds to the Jacobian of the landmark position with respect to the vehicle states and *R* corresponds to measurement noise, *Gz* can be defined as:

$$Gz=\begin{bmatrix} \frac{\partial d}{\partial x_p} & \frac{\partial d}{\partial y_p} & \frac{\partial d}{\partial \theta_p} \\ \frac{\partial b}{\partial x_p} & \frac{\partial b}{\partial y_p} & \frac{\partial b}{\partial \theta_p} \end{bmatrix} \quad \textbf{(4-15)}$$

where Gz shows how much the range and bearing change as $[x_p, y_p, \theta_p]$ change.

## 4.6 Landmark Re-Observation:

In landmark re-observation part, the predicted landmark observations, the Jacobians of these observation positions with respect to vehicle states and also with respect to feature states are need to be taken to be used in further parts. For all observed landmarks, the distance and bearing information is predicted from estimated positions in particles, given as:

$$zp \ = [d_{lm}, b_{lm}] = \begin{bmatrix} \sqrt{(lm_x - x_p)^2 + (lm_y - y_p)^2} & \arctan(\frac{lm_y - y_p}{lm_x - x_p}) - \theta_p \end{bmatrix} \quad \textbf{(4-16)}$$

Where $[d_{lm}, b_{lm}]$ corresponds the distance and bearing information of the predicted landmark observation, $[x_p, y_p, \theta_p]$ corresponds to the pose estimation in given particle and $[lm_x, lm_y]$ corresponds to the estimated landmark positions in particles. Jacobian of predicted landmark observation positions with respect to vehicle states is taken as:

$$Hv = \begin{bmatrix} \frac{\partial d_{lm}}{\partial x_p} & \frac{\partial d_{lm}}{\partial y_p} & \frac{\partial d_{lm}}{\partial \theta_p} \\ \frac{\partial b_{lm}}{\partial x_p} & \frac{\partial b_{lm}}{\partial y_p} & \frac{\partial b_{lm}}{\partial \theta_p} \end{bmatrix} \quad \textbf{(4-17)}$$

Jacobian with respect to feature states is taken as:

$$Hf = \begin{bmatrix} \dfrac{\partial d_{lm}}{\partial lm_x} & \dfrac{\partial d_{lm}}{\partial lm_y} \\ \dfrac{\partial b_{lm}}{\partial lm_x} & \dfrac{\partial b_{lm}}{\partial lm_y} \end{bmatrix} \tag{4-18}$$

Innovation covariance, feature observation given the vehicle uncertainty is calculated as

$$S_f = Hf \; Pf \; Hf + R \tag{4-19}$$

Where, $Pf$ corresponds to the pose covariance matrix of given particle. The pseudo code giving the implementation of these Jacobians is given in Table 4.5:

**Table 4.5: The pseudo code representing the initial calculations for FastSLAM algorithm.**

```
Algorithm_Compute_Jacobians ( particle$^{[i]}$, zf, idf, R)

$[x, y, \theta] = particle^{[i]}.xv$ ;
$[lm_x^{[idf]} \; lm_y^{[idf]}] = particle^{[i]}.xf^{[idf]}$ ;
$Pf = particle^{[i]}.pf^{[idf]}$ ;
for j=1:all idf
      $d^{[j]} = \sqrt{(lm_x^{[j]} - x)^2 + (lm_y^{[j]} - y)^2}$
//predicted observation
      $zp^{[j]} = \begin{bmatrix} d^{[j]} & \arctan(\frac{lm_y^{[j]} - y}{lm_x^{[j]} - x}) - \theta \end{bmatrix}$ ;
//Jacobian with respect to vehicle states
      $Hv^{[j]} = \begin{bmatrix} \frac{x - lm_x^{[j]}}{d^{[j]}} & \frac{y - lm_y^{[j]}}{d^{[j]}} & 0 \\ \frac{lm_y^{[j]} - y}{(d^{[j]})^2} & \frac{x - lm_x^{[j]}}{(d^{[j]})^2} & -1 \end{bmatrix}$ ;
//Jacobian with respect to feature states
      $Hf^{[j]} = \begin{bmatrix} \frac{lm_x^{[j]} - x}{d^{[j]}} & \frac{lm_y^{[j]} - y}{d^{[j]}} \\ \frac{y - lm_y^{[j]}}{(d^{[j]})^2} & \frac{lm_x^{[j]} - x}{(d^{[j]})^2} \end{bmatrix}$ ;
```

**Table 4.5 continued**

```
// innovation covariance
    $Sf^{[j]} = Hf^{[j]} Pf^{[j]} (Hf^{[j]})^T + R$ ;
endfor
$zp = \begin{bmatrix} zp^{[1]} \\ zp^{[2]} \\ .... \\ zp^{[j]} \end{bmatrix}; Hv = \begin{bmatrix} Hv^{[1]} \\ Hv^{[2]} \\ .... \\ Hv^{[j]} \end{bmatrix}; \; Sf = \begin{bmatrix} Sf^{[1]} & 0 & 0 & 0 \\ 0 & Sf^{[2]} & 0 & 0 \\ 0 & 0 & ... & 0 \\ 0 & 0 & 0 & Sf^{[j]} \end{bmatrix}$ for all idf
return zp, Hv, Hf, Sf;
```

After the prediction step, the estimated pose of the robot and covariance is needed to be updated according to the observed landmark. This part forms the difference between the FastSLAM 1.0 and FastSLAM 2.0 algorithms. In FastSLAM 2.0, before the measurement update, sampling poses are updated based on the landmark measurement and control states which gives better diversity of particles for the weight calculation. The pseudo code for the implementation is given in Table 4.6:

**Table 4.6: The pseudo code for proposal sampling part:**

```
Algorithm_FastSLAM_Sample_Proposal (particle, Hv, Hf, Sf)

N=number of particles
for i= 1to N
    [zp, Hv, Hf, Sf]=Compute_Jacobians( $particle^{[i]}$, zf, idf, R);
    for j= all idf
        $v = zf^{[j]} - zp^{[j]}$;
        $particle^{[i]}.pv = \left((Hv^{[j]})^T (Sf^{[j]})^{-1} Hv^{[j]} + (particle^{[i]}.pv)^{-1}\right)^{-1}$;
        $particle^{[i]}.xv = particle^{[i]}.xv + \left(particle^{[i]}.pv (Hv^{[j]})^T (Sf^{[j]})^{-1} v\right)$;
    endfor
endfor
sample $particle^{[i]}.xv$ from $particle^{[i]}.xv$ mean and $particle^{[i]}.pv$ covariance
return particle
```

The positions of landmarks and their covariance in sampled particles are updated which is known as measurement update of EKF. From the measurement covariance and its observation Jacobian with respect to measurement states, the Kalman gain (*K*) is calculated. Kalman gain is computed to find out how much the observed landmarks will be trusted and as such how much to be wanted to gain from the new knowledge they provide. The positions of landmarks and their covariance is updated according to Kalman gain. The pseudo code for the implementation is given in Table 4.7:

**Table 4.7: The pseudo code representing the feature update part of FastSLAM.**

```
Algorithm_FastSLAM_Feature_Update (particle, Hv, Hf, Sf,Q)
N=number of particles
for i= 1to N
[zp, Hv, Hf, Sf]=Compute_Jacobians( $particle^{[i]}$, zf, idf, R);
    for j= all idf
        $v = zf^{[j]} - zp^{[j]}$;
        $K^{[i,j]} = particle^{[i]}.xf^{[j]} (Hf^{[j]})^T (Hf^{[j]} particle^{[i]}.xf^{[j]} (Hf^{[j]})^T + Q)^{-1}$;
        $particle^{[i]}.xf^{[j]} = particle^{[i]}.xf^{[j]} + K^{[i,j]} v$;
        $particle^{[i]}.pf^{[j]} = (I - K^{[i,j]} Hf^{[j]}) particle^{[i]}.pf^{[j]}$;
    endfor
endfor
return particle
```

When a landmark is observed again, the matching probability of the predicted orientation for the landmark in particle and the observed orientation of the landmark with same ID is checked. This gives us the information about the validity of pose and landmark orientation given in that particle. The pseudo code for the calculation of particle weight can be given as:

**Table 4.8: The pseudo code giving the calculation of particle weights.**

```
Algorithm_FastSLAM_ComputeWeight (particle, zf, idf, R, Hv, Hf, Sf)
for i=1: all particles
    [zp, Hv, Hf, Sf]=Compute_Jacobians( $particle^{[i]}$, zf, idf, R);
    v=zf-zp;
    N=number of elements in v;
    V=convert v to Nx1 matrix;
    $S = Hv \ \ Pv \ (Hv)^T + Sf$ ;
    $den = \sqrt{(2\pi)^N \ | S |}$; $num = -0.5 V^T S^{-1} V$;
    $particle^{[i]}.w = \frac{den}{e^{num}}$ ;
endfor
return $particle^{[i]}.w$
```

From innovation covariance (S) including pose and feature uncertainty and the difference between observed and predicted landmark distance and bearings ($zf$ - $zp$) , the consistency of that particle is calculated, which is known as importance weight ( $particle^{[i]}.w$ ).

After the weights are calculated for all particles given in Table 4.8, the particles are resampled. Resampling before computing proposal permits better particle diversity. In this step the ones with less weight are eliminated and the ones with more weight are promoted by expressing them multiple particles, which is known as stratified sampling in statistics. The resampling step is given as a pseudo code in Table 4.9:

**Table 4.9: The pseudo code representing the particle resampling part.**

```
Algorithm_FastSLAM_Particle_Resampling (particle)
N=number of particles
Cumulative_sum=0;
for i= 1 to N
    $particle^{[i]}.w = \frac{particle^{[i]}.w}{\sum_{j=1}^{N} particle^{[j]}.w} + Cumulative\_sum;$
    Cumulative_sum= Cumulative_sum+ $particle^{[i]}.w;$
endfor
$di = array\ from \frac{1}{2N}\ to \left(1 - \frac{1}{2N}\right) with\ \frac{1}{N}\ steps$
rand= random N numbers ;
for i=1 to N ;    $s^{[i]} = di^{[i]} + \frac{1}{N} rand^{[i]} - \frac{1}{2N}$    ;    endfor;
keep=null matrix (1 to number of particles)
count=1;
for i=1:N
    while count<=len and $s^{[count]} < particle^{[i]}.w;$
        keep(count)=i;
        count=count+1;
    end
end
particles= particles(keep);
$particle^{[i]}.w = 1/N;$
return particle
```

As it is seen from Table 4.9, according to the given weight, the particles are resampled. In the resampling process, the particles are copied proportional to their weights in total sum of weights, as they obtain fixed number of particles.

## 4.7 FastSLAM Algorithm with Landmark Orientation Update:

In this part, FastSLAM algorithm is modified to add the slope of the lines as a parameter to the algorithm. This part includes the same application with the two landmark position addition to handle the update for the orientation of landmark inside, given in Section 4.4 and in Figure 4.4. This time a landmark is not defined with two positions, instead the slope of its line is directly added to the filter. All the steps about the operation of the FastSLAM algorithm are same; some parameters related with landmarks in the algorithm are changed. When a landmark is seen first time, its parameters are defined as:

$$\left[r, b, \phi\right] = zn \tag{4-20}$$

Where $r$ is the range information, $b$ is the bearing information of the landmark referenced to robot and $\phi$ is the orientation of landmark inside (the difference between the slope of the line constructing the landmark in global coordinate frame and heading of the robot registered in particle). So its orientation (landmark state) is expressed in particle as:

$$particle.xf = \begin{bmatrix} lm_x \\ lm_y \\ lm_\gamma \end{bmatrix} = \begin{bmatrix} x_p + r\cos(\theta_p + b) \\ y_p + r\sin(\theta_p + b) \\ \theta_p + \phi \end{bmatrix} \tag{4-21}$$

where $lm_\gamma$ is the estimated orientation information of landmark in global frame according to supposed robot pose in particle. By addition of orientation information, the jacobian of landmark observation with respect to robot pose estimated in particle becomes:

$$Gz= \begin{bmatrix} \frac{\partial r}{\partial x_p} & \frac{\partial r}{\partial y_p} & \frac{\partial r}{\partial \theta_p} \\ \frac{\partial b}{\partial x_p} & \frac{\partial b}{\partial y_p} & \frac{\partial b}{\partial \theta_p} \\ \frac{\partial \phi}{\partial x_p} & \frac{\partial \phi}{\partial y_p} & \frac{\partial \phi}{\partial \theta_p} \end{bmatrix} = \begin{bmatrix} \cos(\theta_p + b) & -r\sin(\theta_p + b) & 0 \\ \sin(\theta_p + b) & r\cos(\theta_p + b) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4-22}$$

For landmark re-observation case, the predicted landmark position (expected measurement) becomes;

$$zp = \begin{bmatrix} r_{lm} \\ b_{lm} \\ \phi_{lm} \end{bmatrix} = \begin{bmatrix} \sqrt{(lm_x - x_p)^2 + (lm_y - y_p)^2} \\ \arctan(\frac{lm_y - y_p}{lm_x - x_p}) - \theta_p \\ lm_\gamma - \theta_p \end{bmatrix} \tag{4-23}$$

where $\phi_{lm}$ corresponds to relative predicted landmark orientation degree based on robot heading and landmark orientation information in global frame registered in the particle. Jacobian of predicted landmark observation with respect to vehicle states is taken as:

$$Hv = \begin{bmatrix} \frac{\partial r_{lm}}{\partial x_p} & \frac{\partial r_{lm}}{\partial y_p} & \frac{\partial r_{lm}}{\partial \theta_p} \\ \frac{\partial b_{lm}}{\partial x_p} & \frac{\partial b_{lm}}{\partial y_p} & \frac{\partial b_{lm}}{\partial \theta_p} \\ \frac{\partial \phi_{lm}}{\partial x_p} & \frac{\partial \phi_{lm}}{\partial y_p} & \frac{\partial \phi_{lm}}{\partial \theta_p} \end{bmatrix} = \begin{bmatrix} \frac{x - lm_x^{[j]}}{d^{[j]}} & \frac{y - lm_y^{[j]}}{d^{[j]}} & 0 \\ \frac{lm_y^{[j]} - y}{(d^{[j]})^2} & \frac{x - lm_x^{[j]}}{(d^{[j]})^2} & -1 \\ 0 & 0 & -1 \end{bmatrix} \tag{4-24}$$

Jacobian of predicted landmark observation (expected measurement) with respect to landmark states is taken as:

$$Hf = \begin{bmatrix} \frac{\partial r_{lm}}{\partial lm_x} & \frac{\partial r_{lm}}{\partial lm_y} & \frac{\partial r_{lm}}{\partial lm_\gamma} \\ \frac{\partial b_{lm}}{\partial lm_x} & \frac{\partial b_{lm}}{\partial lm_y} & \frac{\partial b_{lm}}{\partial lm_\gamma} \\ \frac{\partial \phi_{x}}{\partial lm_x} & \frac{\partial \phi_{lm}}{\partial lm_y} & \frac{\partial \phi_{lm}}{\partial lm_\gamma} \end{bmatrix} = \begin{bmatrix} \frac{lm_x^{[j]} - x}{d^{[j]}} & \frac{lm_y^{[j]} - y}{d^{[j]}} & 0 \\ \frac{y - lm_y^{[j]}}{(d^{[j]})^2} & \frac{lm_x^{[j]} - x}{(d^{[j]})^2} & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{4-25}$$

The corresponding formulas in FastSLAM algorithm are updated according to the equations calculated between Equation 4-20 and 4-25. As a result, with same structure a modified FastSLAM algorithm that handles the orientation information of landmarks is implemented.

## 4.8 Chapter Summary:

In this chapter, the implementation of FastSLAM algorithm is presented. This algorithm takes pose difference from main scan matching part and also uses the feature and feature properties set extracted in scan matching part. The pose of the robot is relocated by particles. From the incoming pose difference from scan matching algorithm, by using inverse odometer model, the control input is calculated. In prediction update step, the pose estimation in particles is updated according to the control input. Landmark extraction is made from the feature set extracted in scan matching part. Both side ended lines and real corners are used as landmark candidates. Confident candidates are sent to data association part and registered in memory with an ID. When a new landmark is observed, its position information and covariance matrix is set. When a landmark is re-observed, the likelihood of observed landmark information with the estimated one's information is calculated. The mean and covariance of pose in particle is updated according to this information, and sampled. The landmark positions and covariance in resulting particles are updated according to calculated Kalman gain. According to overlap between predicted and observed landmark information, weights are calculated. By using the weights the particle set is resampled. In this step, the ones that landmark prediction does not match with observed one are eliminated and the rate of surviving ones in particle set is increased in proportion to weights.

# CHAPTER 5

# EXPERIMENTAL RESULTS

The performances of main scan matching, scan matching with deterministic global localization, scan matching with FastSLAM algorithms are tested with different real LADAR and odometer datasets that are taken from Radish Data Repository and from our tests with our robots and courses. In the figures below, the performance of the algorithms with given datasets are presented. In our experiments the performance of the algorithms is analyzed in two steps:

**Maximum Local Distortion:** In this step, maps created by scan matching with DGL and with FastSLAM algorithms are compared inside. We look at areas where the robot passed multiple times. This gives an insight about the consistency of the algorithm. The distortion occurred on the features shows the error accumulated on scan matching during all the path till it comes to the given area. The procedure for the calculations is:

1) Some areas in algorithm map are numbered as check areas.
2) Distortion on the features in position and slope in these areas is checked. In fact, the accumulated error on the algorithm is seen as a slippage on the features in these areas. Also instant errors on scan matching can be observed.
3) The distortion as pixel position difference is multiplied with grid size and projected onto x and y axis.

**Maximum Global Distortion:** In this step, the maps created by algorithms are checked with the original map. We take same features from both maps and look at the position difference between these points. This measure gives us an insight about the accuracy of our algorithm related to the real map. The procedure for the calculations:

1) The both maps are set into the same grid size, which is 0.1 meters in our mapping process. Algorithm map is rotated as the features at the starting point will have the same slope with the ones at original map.
2) Some areas in maps are numbered as check areas (These areas same as the ones in maximum local distortion). The area at the startpoint is numbered as reference area (which is given as R).
3) In these areas, the pixel positions of the same corners in the original map and scan matching result are compared referenced to the same corner in reference area (given with R).
4) The difference between the pixel positions of these corners in both maps multiplied with grid size (0.1 meters) , which gives us the distortion in the algorithm map referenced to the original map in that position.
5) The slopes of the same lines in same numbered area in both maps are found.
6) The slope difference gives us the distortion in the slope of to robot referenced to the original map.

In Figure 5.11, Figure 5.21, Figure 5.34 the numbered places are checking areas.

**Mean Position Error** and **Mean Orientation Error**: In maximum global localization part, worst position and orientation errors for each area are calculated. From these errors, the mean position and orientation error is calculated according to the given formulas:

$$Mean\,Position\,Error = \frac{1}{n}\sum_{i=1}^{n}\sqrt{x_i^{\,2} + y_i^{\,2}} \qquad \textbf{(5-1)}$$

$$Mean\;Orientation\,Error = \frac{1}{n}\sum_{i=1}^{n}\theta_i \qquad \textbf{(5-2)}$$

where $x_i$ and $y_i$ corresponds to worst position error in *Area i* and $\theta_i$ corresponds to worst orientation error in *Area i. n* is the number of areas that the given test area is divided.

## 5.1 Test 1: ASELSAN Indoor Area

The tests are made with

- Pioneer 3AT Robot (Differential Drive)
- SICK OEM1000 Laser Scanner
- 180 degree LADAR scan with 1 degree interval

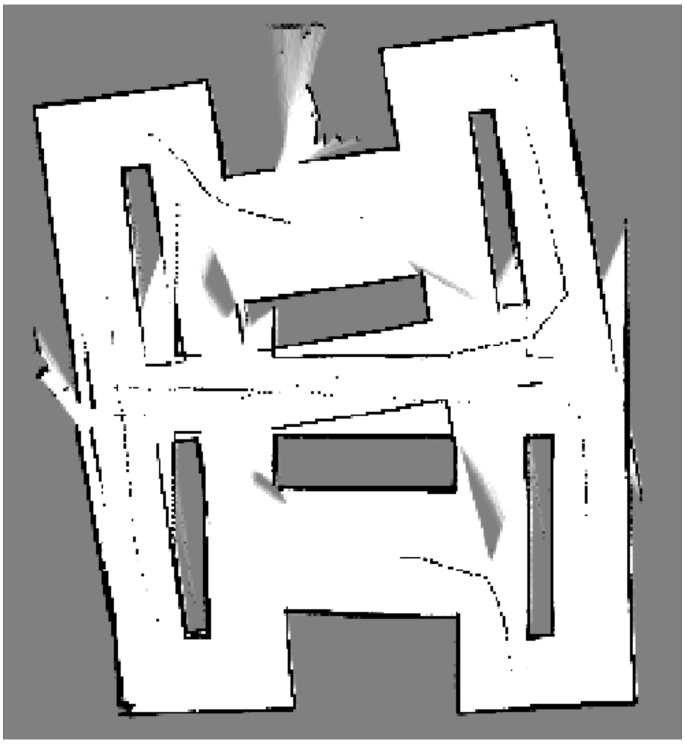


**Figure 5.1: The occupancy grid map of ASELSAN Indoor Area by using main scan matching.**

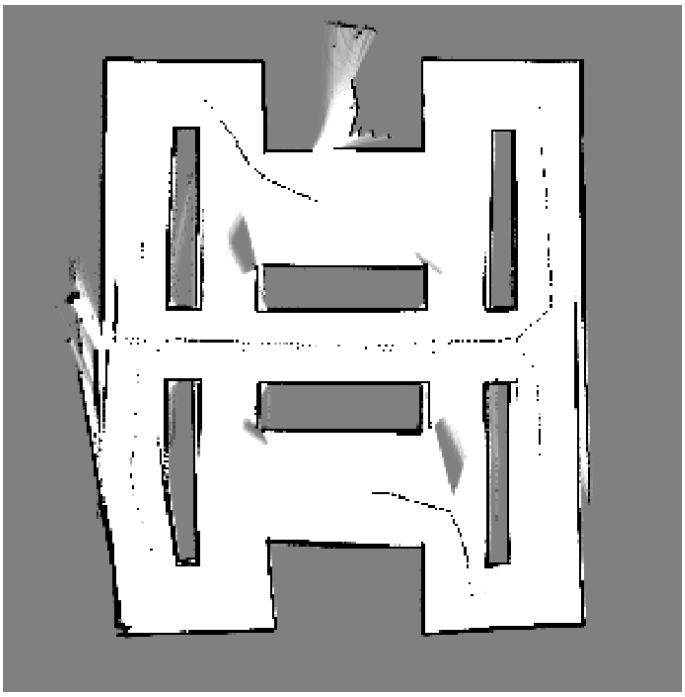


**Figure 5.2: The occupancy grid map of ASELSAN Indoor Area by using scan matching with DGL algorithm.**

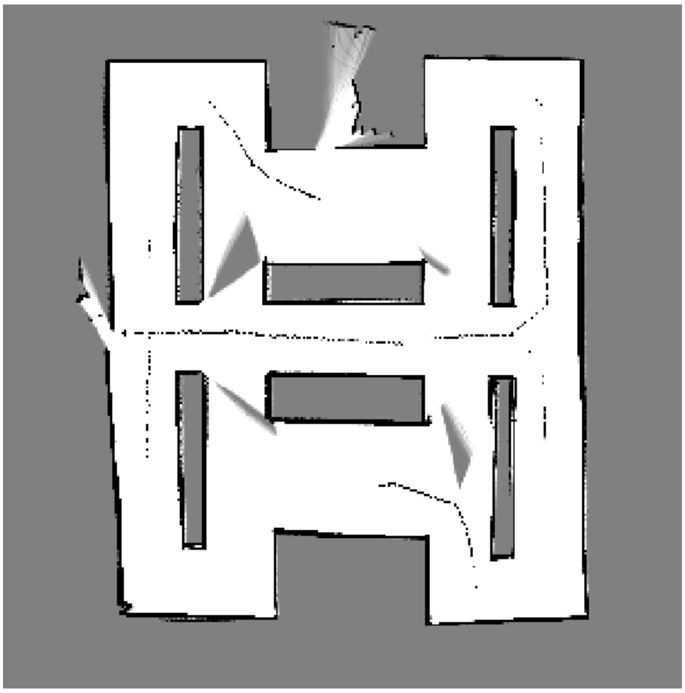


**Figure 5.3: The occupancy grid map of ASELSAN Indoor Area by using scan matching with full FastSLAM algorithm.**

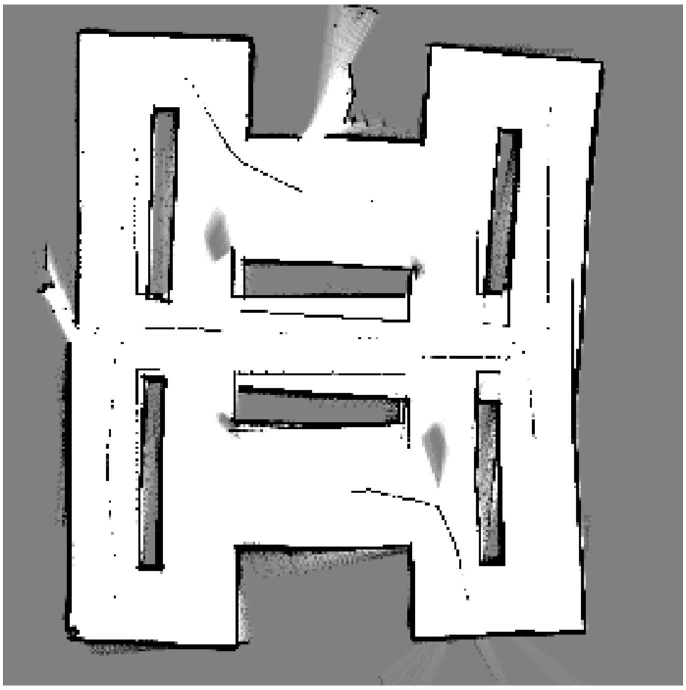


**Figure 5.4: The occupancy grid map of ASELSAN Indoor Area by using the raw odometer data.**

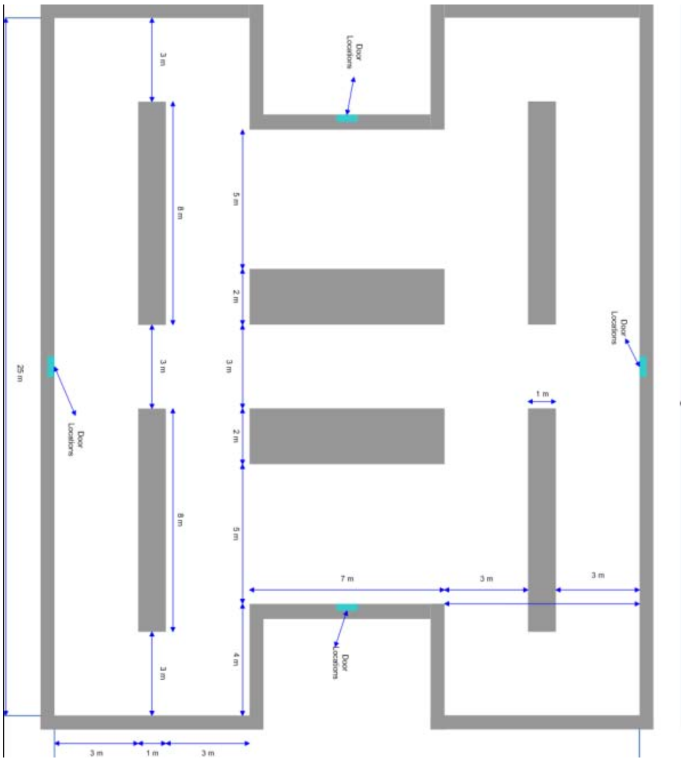


**Figure 5.5: The real map of ASELSAN indoor area.**

In Figure 5.1, the scan matching result is presented that is obtained from the raw LADAR and odometer data. In Figure 5.2, scan matching result with DGL algorithm is presented. In Figure 5.3, scan matching result with full FastSLAM algorithm is presented. Figure 5.4, mapping by using raw odometer output is presented. In Figure 5.5, the real map is presented from ASELSAN Indoor Area.

In Figure 5.1, a rotational error is observed in mapping, this error is coming from the incremental error till the area given with number 5 in Figure 5.11 plus the mismatches occurred in this area. This pose error is corrected by DGL algorithm in Figure 5.2 and FastSLAM algorithm in Figure 5.3.

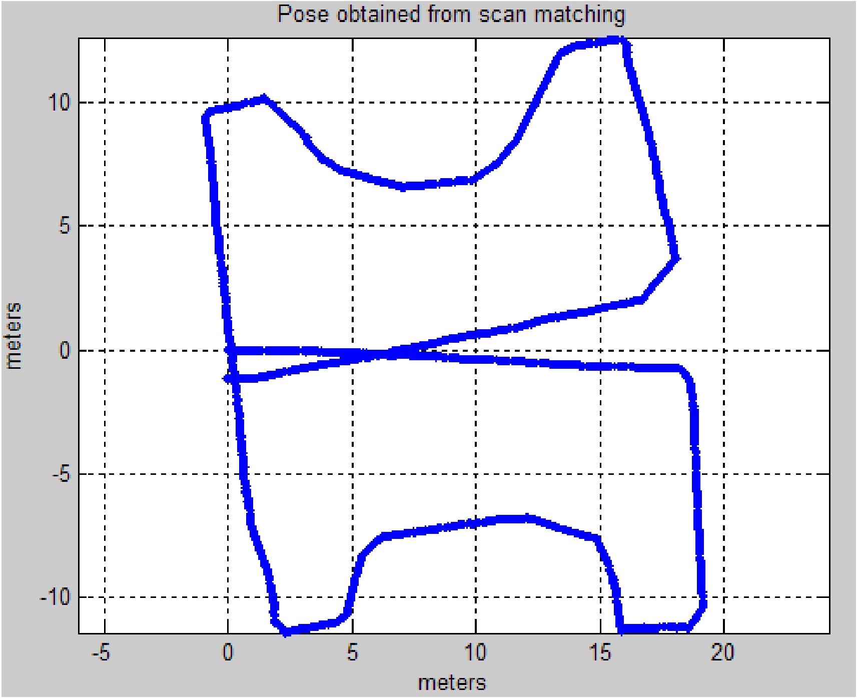


**Figure 5.6: The pose information obtained from main scan matching algorithm.**

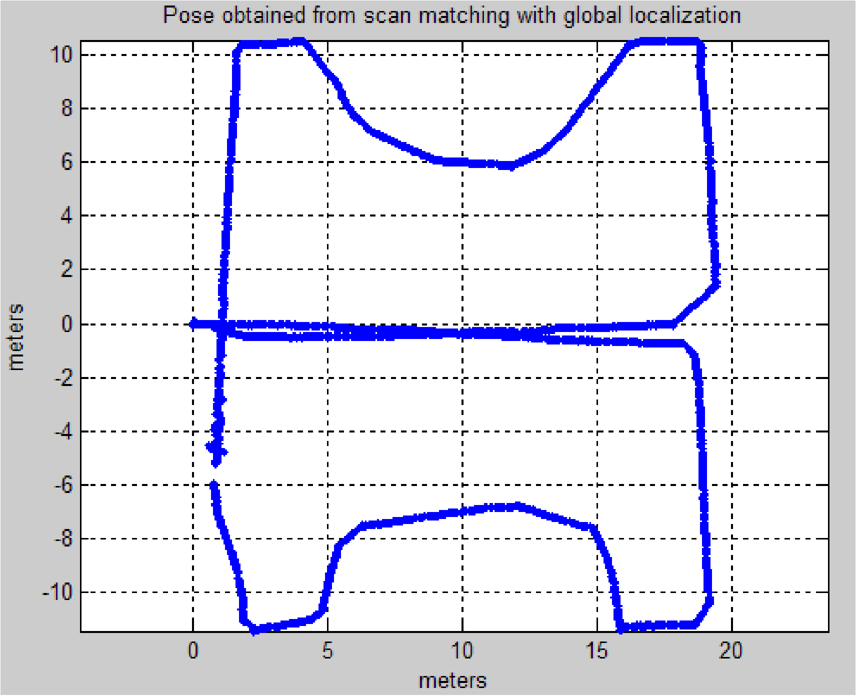


**Figure 5.7: The pose information obtained from scan matching with DGL algorithm.**

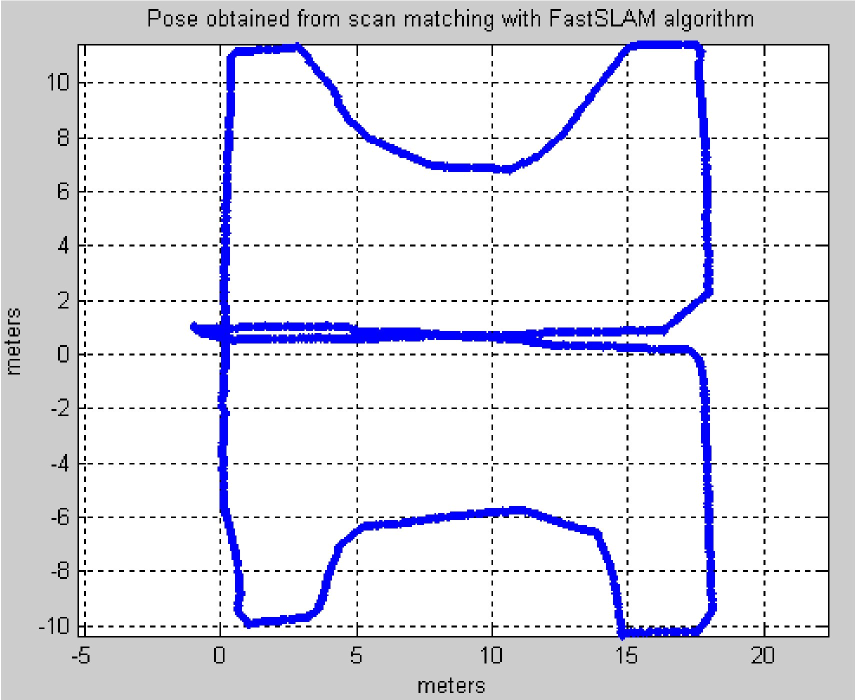


**Figure 5.8: The pose information obtained from scan matching with full FastSLAM algorithm.**

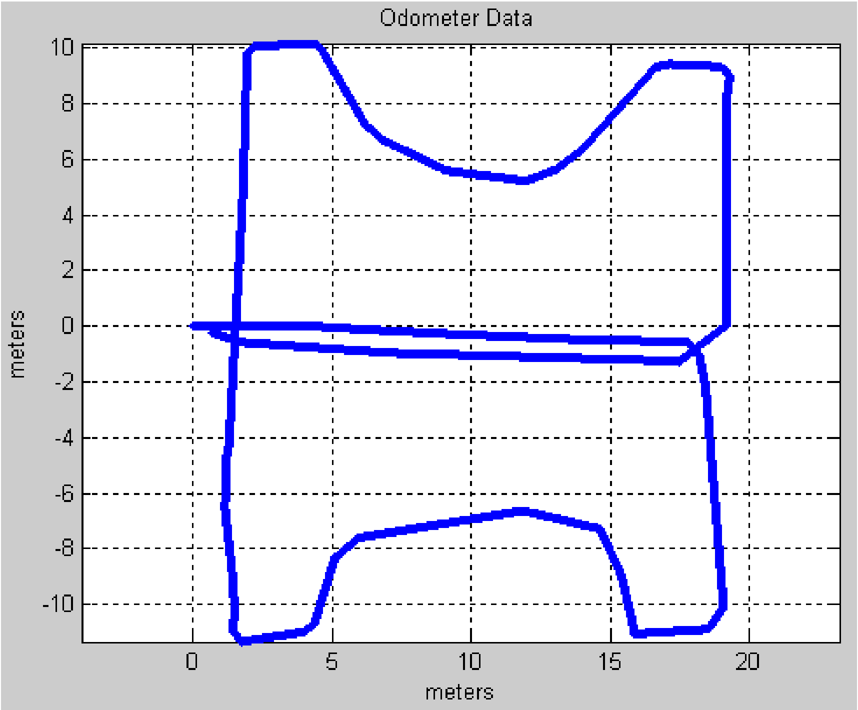


**Figure 5.9: The raw odometer data taken from the robot.**

In Figure 5.6, the pose information obtained from scan matching algorithm, in Figure 5.7, the pose information obtained from scan matching algorithm with DGL algorithm is presented. In Figure 5.8, the pose information obtained from scan matching with FastSLAM algorithm is presented. In Figure 5.9, the raw odometer data that is taken from the dataset is presented. As it is seen from Figure 5.6, because of mismatch and accumulated error a rotational error is occurred. In Figure 5.7, DGL algorithm corrects this error, which is seen as a jump in (-1,6) coordinates. In Figure 5.8, this correction is spread on previous poses and a smooth path without a jump is seen

The robot starts from the (0,0) point and goes through the x axis, then it turns right and maps all the right side area, which could be defined as y<0 in odometer data. Then it maps the left side area, which could be defined as y>0 in odometer data. Then it turns back to (0, 0) point from the center of total area through X axis in odometer plot.

For considering the error in the map we can compare the slippage on the wall positions which located on the start and end path of the robot.

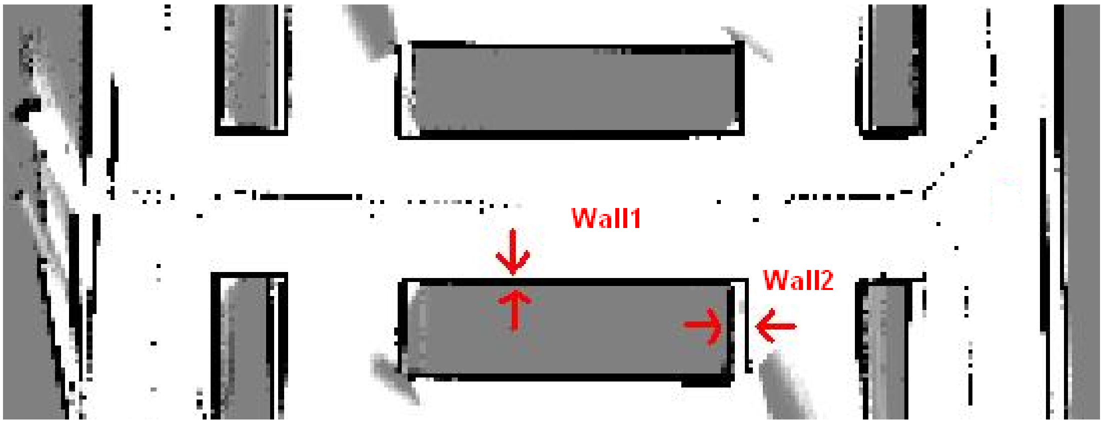


**Figure 5.10: Zoomed area at the middle of scan matching result given in Figure 5.2.**

To find the total error, the slippage on the wall position given with Wall1 and Wall2 is looked. The slippage in Wall1 is below 10 centimeters, because no change in the grids of the wall is seen. The slippage in Wall2 is 4 grid cell sizes, which is about 40 centimeters. Also so less error is seen in the rotation of the robot, which is found by subtracting the slopes of wall given with Number 1 at the start and end of the path, which is about 0.3 degrees. So we could not see the error on the wall with 0.1 meters grid size.

As a result, if we assume Wall Number 1 as X axis and Wall Number 2 as Y axis, we can say that we have obtained 0.4 meters in X axis, <0.1 meters in Y axis translational error and 0.3 degree rotational error in a distance about 121.5 meters traveled in a path consist of closed loops with combined method consisting of scan matching and DGL algorithm.

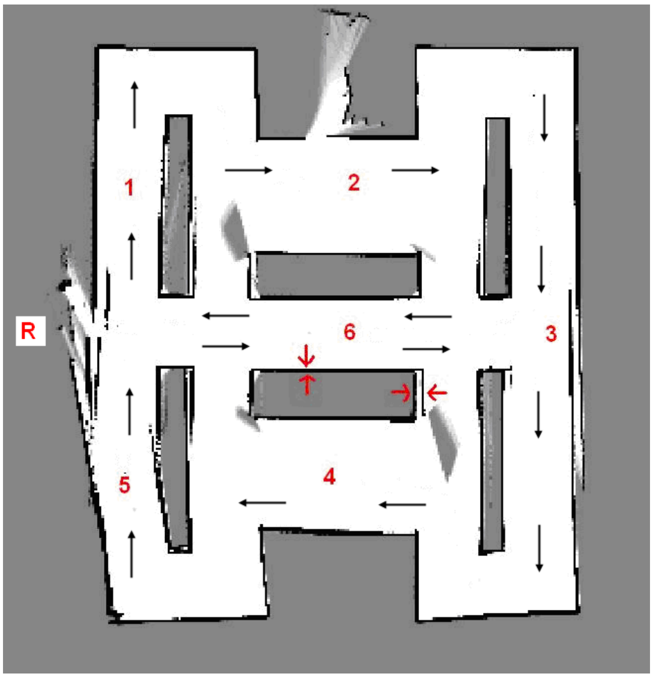


**Figure 5.11: The figure showing the check areas that maximum global/local distortions are calculated between localization algorithms and original map. It also shows the path odf the robot by arrows.**

The accuracy of the x, y values is 0.1 meters, which is the grid size; also accuracy of the θ values is less then 0.3 degree, because of the ratio between the grid size and longest line observed. The results are given as absolute values. As it is seen from Figure 5.9, all the area except Area 6 in Figure 5.11, are passed once, so these areas seems to be inappropriate for local distortion check. However, as it is seen from the distortions in these areas that are in neighborhood of Area 6, same walls are seen more than once, one is while passing through area 1, 2, 3, 4, 5 and one is while passing through area 6 as a consequence of high range capability of LADAR. Consequently, local distortion is calculated for all these areas.

**Table 5.1: This table gives the local distortions obtained inside the scan matching result based on the numbering given in Figure 5.11.**

| Maximum Local Distortions for Scan Matching with DGL Algorithm | | | |
|---|---|---|---|
| | Local Distortions | | |
| Area ID | x (meters) | y (meters) | θ (degree) |
| 1 | 0.1 | <0.1 | 0.53 |
| 2 | 0.3 | <0.1 | 0.85 |
| 3 | 0.4 | <0.1 | 1.2 |
| 4 | 0.2 | 0.1 | 0.48 |
| 5 | 0.8 | 0.2 | 9.4 |
| 6 | 0.4 | <0.1 | 0.51 |

The results given in Table 5.1 shows that the map we have created by scan matching is consistent. In Area 5, we observe 9.4 degree rotational error; also it can be seen in Figure 5.11. This error is corrected while the robot is passing from Area 5 to Area 6, since tracked features by global localization algorithms are observed again. Therefore in Area 6, localization accuracy increases. Also while passing from Area 3 to Area 6, the accumulated rotational error is corrected, which is seen in Table 5.1

**Table 5.2: This table gives the local distortions obtained inside the scan matching result based on the numbering given in Figure 5.11.**

| Maximum Local Distortions for Scan Matching with Full FastSLAM Algorithm | | | |
|---|---|---|---|
| | Local Distortions | | |
| Area ID | x (meters) | y (meters) | θ (degree) |
| 1 | <0.1 | <0.1 | 0.30 |
| 2 | 0.1 | <0.1 | 0.27 |
| 3 | 0.2 | 0.1 | 0.52 |
| 4 | 0.2 | 0.1 | 0.32 |
| 5 | 0.3 | 0.1 | 0.87 |
| 6 | 0.1 | 0.2 | 0.27 |

As we can see from the results given in Table 5.2, there is an increase in consistency of the map when it has compared with scan matching with DGL algorithm. Because of particle structure of FastSLAM algorithm, for each landmark re-observation the path is optimized according to the weights that the particles has taken, also the landmark positions are updated. As a result better translation and rotation accuracy is seen.

**Table 5.3: This table gives the global distortions obtained between the scan matching result and original map in sampled areas.**

| Maximum Global Distortions for Scan Matching with DGL Algorithm | | | |
|---|---|---|---|
| | Global Distortions | | |
| Area ID | x (meters) | y (meters) | θ (degree) |
| 1 | 0.3 | 0.1 | 0.55 |
| 2 | 0.5 | 0.1 | 0.85 |
| 3 | 0.6 | 0.1 | 1.2 |
| 4 | 0.2 | 0.2 | 0.48 |
| 5 | 0.6 | 0.2 | 9.4 |
| 6 | 0.4 | <0.1 | <0.3 |

The results given in Table 5.3 shows that during the entire path, scan matching algorithm gives pose output with less than 0.6 meters error in translation. The correction information of rotational error in Area 5 is given in Maximum Local Distortions part.

**Table 5.4: This table gives the local distortions obtained inside the scan matching result based on the numbering given in Figure 5.11.**

| Maximum Local Distortions for Scan Matching with Full FastSLAM Algorithm | | | |
|---|---|---|---|
| | Global Distortions | | |
| Area ID | x (meters) | y (meters) | θ (degree) |
| 1 | <0.1 | 0.2 | 0.30 |
| 2 | 0.2 | <0.1 | 0.27 |
| 3 | 0.1 | 0.1 | 0.52 |
| 4 | 0.2 | 0.3 | 0.32 |
| 5 | 0.3 | 0.1 | 0.87 |
| 6 | 0.1 | 0.1 | 0.27 |

As we can see from the results given in Table 5.4 and Table 5.2, there is an increase in consistency of the map when it has compared with scan matching with DGL algorithm. In full FastSLAM algorithm, the paths of the particles with maximum weight appointed are survived. This resulted with back propagated correction in previous poses of the robot. As a consequence of this, the separation in the wall at Area 3 and the inclination in the inner wall in Area 5 are not observed in Figure 5.3.

**Table 5.5: This table gives the mean position error and orientation error for Test Area 1 according to the equations 5.1 and 5.2.**

| | Mean Position Error | Mean Orientation Error |
|---|---|---|
| Scan Matching | 1.83 | 5.8 |
| Scan Matching +DGL | 0.46 | 2.13 |
| Scan Matching +FastSLAM | 0.23 | 0.42 |

## Evaluation of Algorithms for Test 1:

In given Area 1 three algorithms are tested. These are scan matching, scan matching with deterministic global localization (DGL) and scan matching with FastSLAM algorithm.

When they are compared, in order with increase in pose estimation accuracy and map consistency, main scan matching, scan matching with DGL algorithm, scan matching with FastSLAM algorithm. As it is seen in Figure 5.1, the scan matching algorithm came across with an error in Area 6. This comes from the addition of accumulated incremental heading error till that point and the wrong heading calculation occurred at the given area in scan matching algorithm. Since any global correction does not occur in scan matching, the heading error is not corrected in rest of the path. So a great distortion in map has occurred. The effect of this distortion can be seen in Figure 5.6, the upper half and down half of the path are consistent inside but the heading error occurred locates them with angular difference.

Scan matching with DGL algorithm gives better estimate for the path. But since there is no correction for the previous poses when a landmark is reobserved, it gives worse results than the scan matching with FastSLAM algorithm. In fact their difference specially can be seen in Area 6 (the slope of the lines) of Figure 5.2 and Figure 5.3. The rotational error occurred in scan matching is corrected in both ones. But while the previous mapping and pose error stays for the one with DGL implementation, this is corrected to a certain extent for the one with FastSLAM algorithm. This is also seen in joint wall passing through Area 3. In Area 3, right side of the wall seen first and left side of the wall is seen before the largest cycle in map is closed. As a result, because of accumulated error, gradient of the wall changes and a fracture occurs between thsese sides for DGL implementation in Figure 5.2. This is corrected for the next mappings when landmarks in right side of Area 3 are seen. However same line seems to be about straight for FastSLAM algorithm in Figure 5.3.

In Figure 5.4, the mapping for raw odometer is given. This is given to show that a small error in pose results with a great distortion in mapping. Actually the odometer data given in Figure 5.9 seems to be correct (This accuracy comes from the driving style, center of gravity distribution and unweared platform pieces.), but when a mapping is placed on it, the distortion has come off.

## 5.2 Test 2: University of Ancona - LADAR and Odometer Data Taken from RADISH Data Repository

The test is made with;

- Pioneer 3DX Robot (Differential Drive)
- SICK LMS 200 Laser Scanner
- 180 degree LADAR scan with 1 degree interval

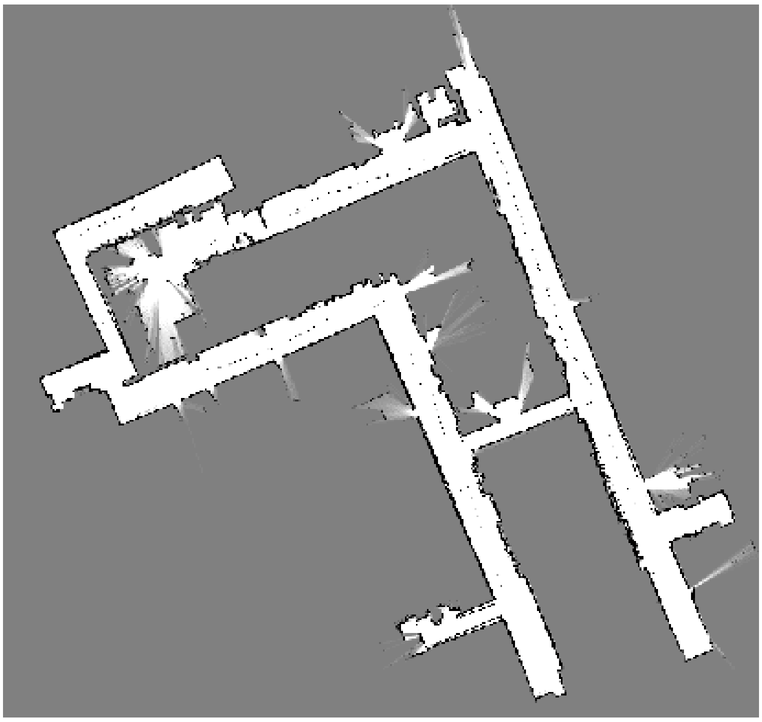


**Figure 5.12: The occupancy grid map obtained from main scan matching.**

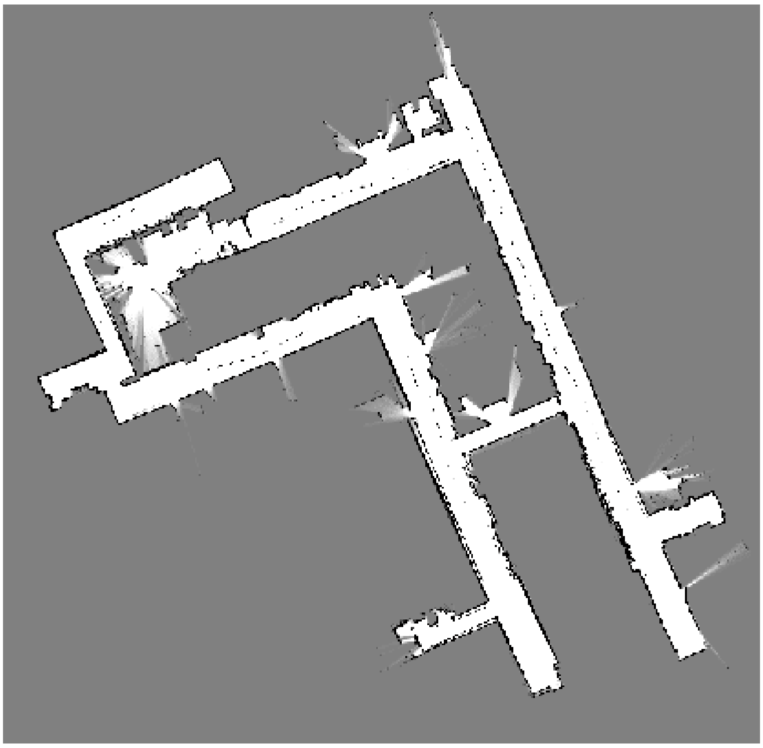


**Figure 5.13: The occupancy grid map obtained from scan matching with DGL algorithm.**

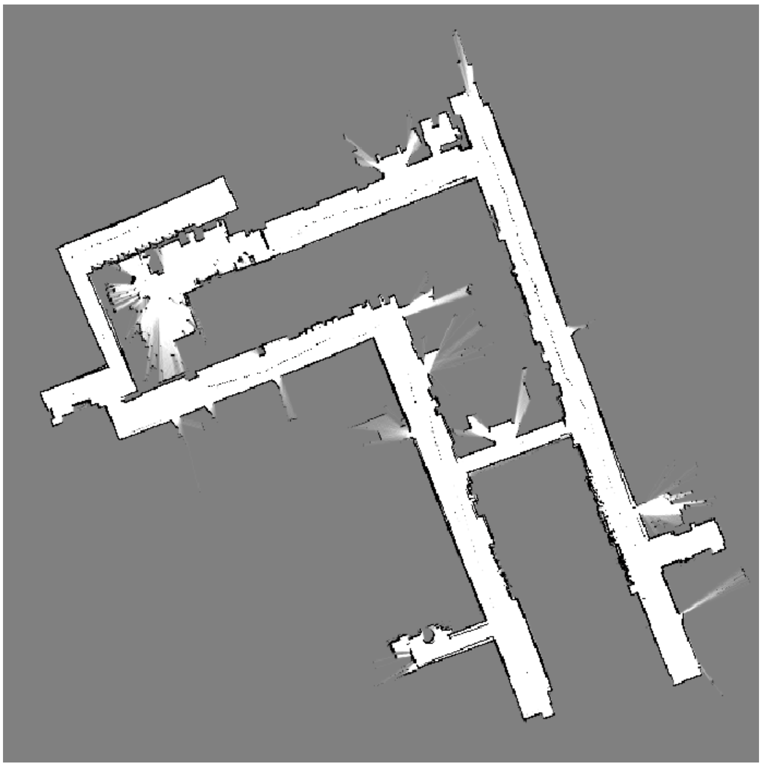


**Figure 5.14: The occupancy grid map obtained from scan matching with full FastSLAM algorithm.**

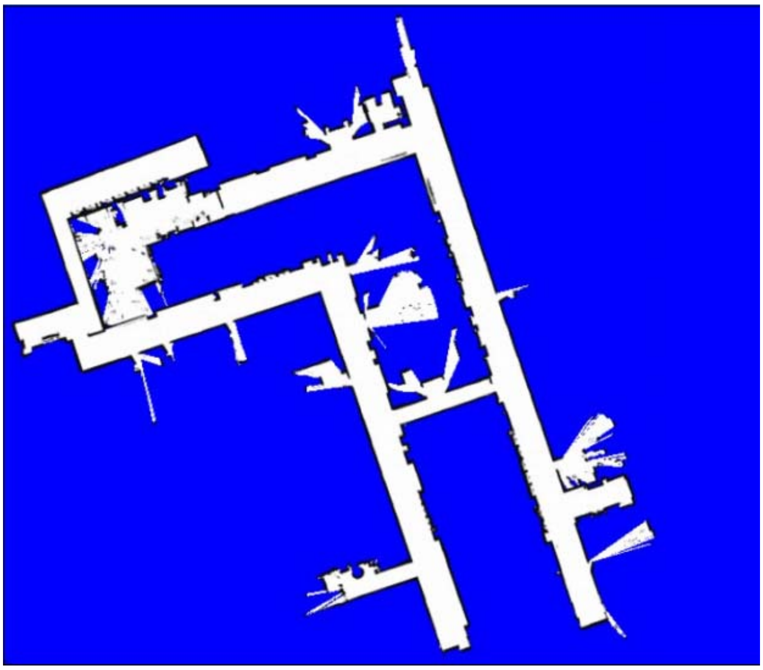


**Figure 5.15: The real map given by University of Ancona.**

In Figure 5.12 the main scan matching result is presented that we have obtained from LADAR and odometer data. In Figure 5.13, the scan matching with DGL result and in Figure 5.14, the scan matching with full FastSLAM algorithm is presented. In Figure 5.15, the real map is presented given by University of Ancone, which is the source of dataset taken from the Radish Data Repository.

If Figure 5.12 and Figure 5.13 are compared with Figure 5.15, scan matching outputs are so successful. Both maps are consistent inside and match with real map. When Figure 5.12 is compared with Figure 5.13, an incremental error in pose is seen. This error is small but it can be seen as multiple lines at the areas where closed loop case in path occurs. In fact resulting error at the end is easily seen on the walls between the startpoint and endpoint of the wall (Start and endpoints of the path and closed loop areas are given in Figure 5.21, R as the area at startpoint and 7 as the area at endpoint, 2- 4 – 5 are closed loop areas.)

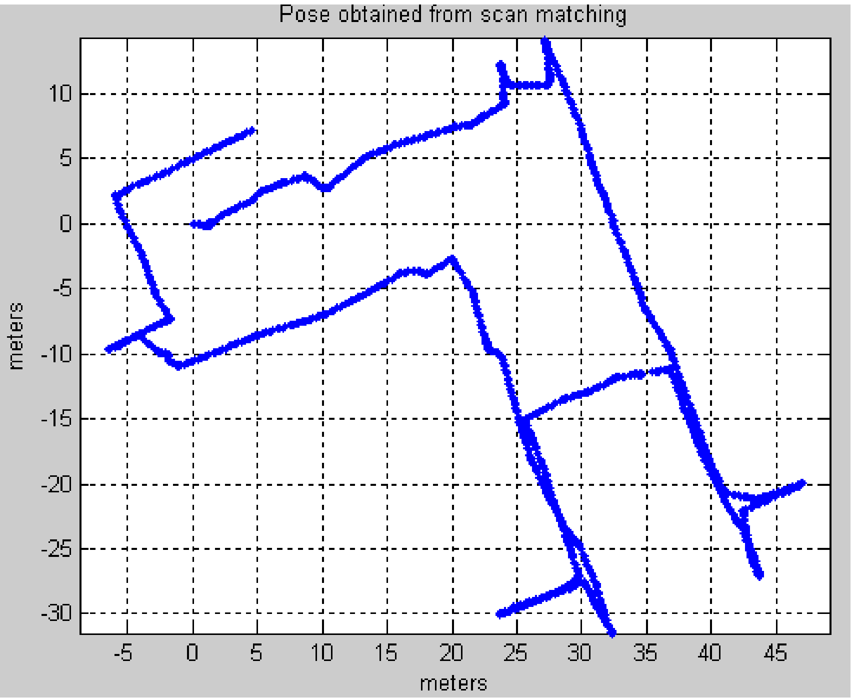


**Figure 5.16: The pose information obtained from scan matching algorithm.**

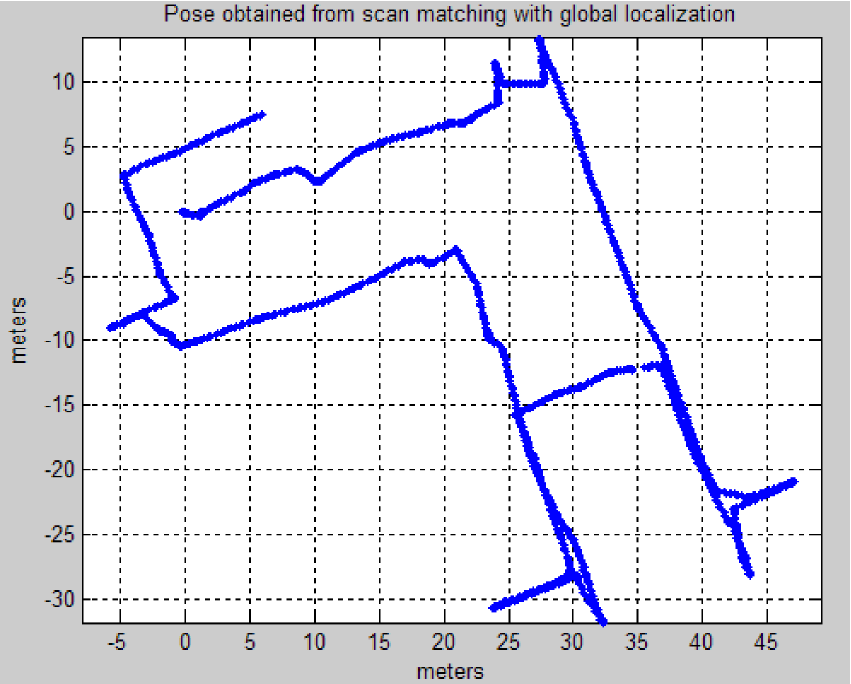


**Figure 5.17: The pose information obtained from scan matching with DGL algorithm.**

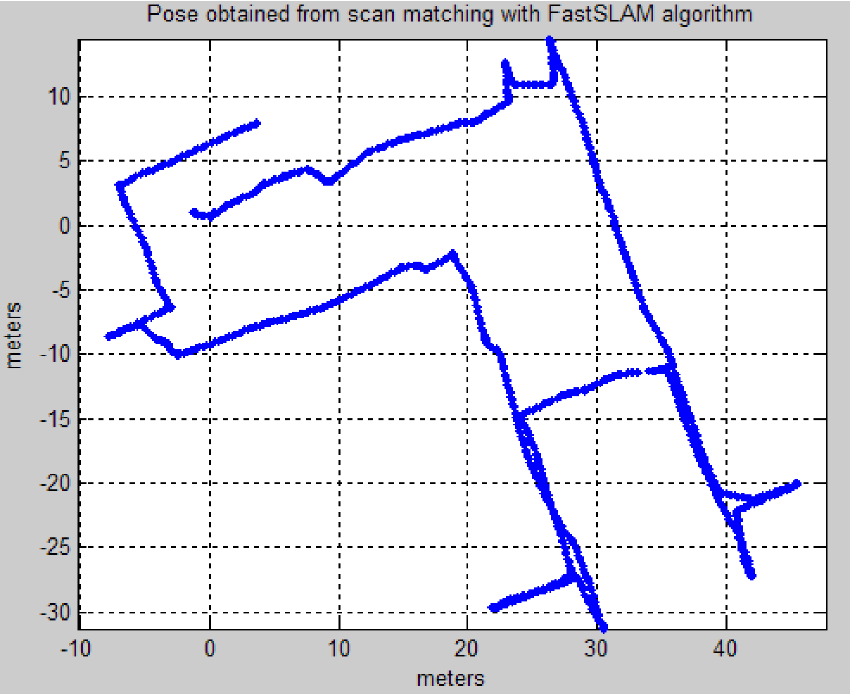


**Figure 5.18: The pose information obtained from scan matching with full FastSLAM algorithm.**

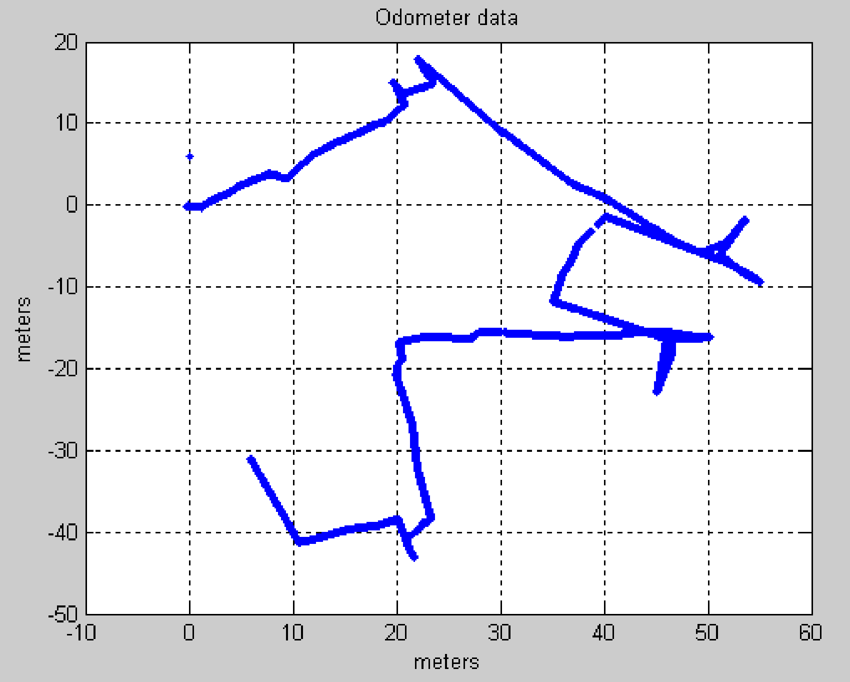


**Figure 5.19: The odometer data given while the robot is moving on the map given in Figure 5.15.**

In Figure 5.16, the pose information obtained from scan matching algorithm, in Figure 5.17, the pose information obtained from scan matching with DGL algorithm and in Figure 5.18 the pose information obtained from scan matching with FastSLAM algorithm is presented. In Figure 5.19, the raw odometer data that is taken from the dataset is presented.

As you can see from Figure 5.19, the raw odometer data diverges after a movement about 25 meters from the start point. When we have come to the endpoint (settled about (5,-30) coordinates), there is big error more than 30 meters in translational and about 90 degrees in rotational. However the odometer data obtained from scan matching in Figure 5.16 and Figure 5.17 and Figure 5.18 overlap with the map given in Figure 5.15. It is assumed that the map given by University of Ancona as real map and make comparison between the real map and the map we have obtained from algorithms. During this test, total distance that is traveled is about 236.8 meters.

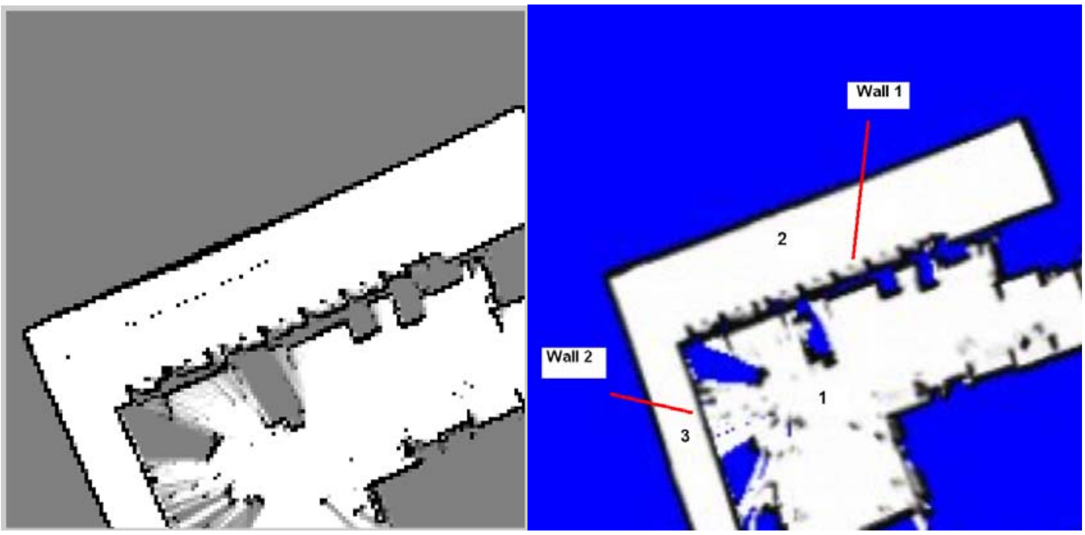


**Figure 5.20: The two figures given are the zoom in of the maps presented in Figure 1 and 2. The left image is from Figure 1 and the right one is from Figure 2.**

In Figure 5.20, the zoom in area shows both the area the navigation has started and the area that navigation has ended. The area identified with Number 1 in right image is the room that the robot has started and the area identified with Number 2 in right image is the corridor that the robot has ended its navigation. So the Wall 1 between the corridor and the room gives us idea about the error occurred in axis perpendicular to the wall during this navigation. In our scan matching result the wall thickness is found about 5 grids (since our grid size is 10x 10 cm), which is 0.5 meters error if no thickness is assumed for the wall. Wall 2 between the corridor and the room gives us idea about the error occurred in axis perpendicular to the wall (this axis is parallel to Wall 1) during this navigation. Wall thickness is found about 3 grids, which is 0.3 meters error if no thickness is assumed for the wall.

We have look at the slope of the line fitting, looking to the area Number 1 and at the slope of the line fitting, looking to the area Number 2 to find the error in rotation. The resulted line slope difference is about 0.8 degrees, which is our rotational error.

As a result, if we assume Wall 1 as y axis and Wall2 as x axis, we can say that we have obtained 0.5 meters in y axis, 0.3 meters in x axis translational error and 0.8 degree rotational error in a distance about 236.8 meters traveled in a path consist of small closed loops with scan matching with DGL algorithm.

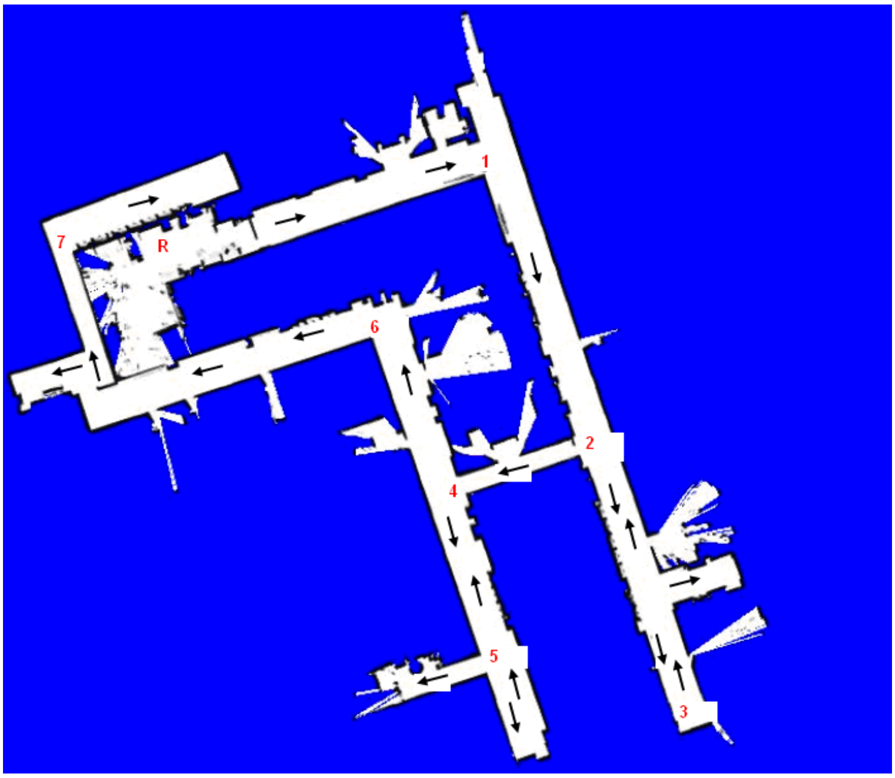


**Figure 5.21: The figure showing the check areas that maximum global distortions are calculated between scan matching result and original map.**

The accuracy of the x, y values is 0.1 meters, which is the grid size; also accuracy of the θ values is less then 0.3 degree, because of the ratio of longest line to the grid size. The results are given as absolute values.

**Table 5.6: This table gives the local distortions obtained inside the scan matching with DGL algorithm result based on the numbering given in Figure 5.21**

| Maximum Local Distortions for Scan Matching with DGL Algorithm | | | |
|---|---|---|---|
| | Local Distortions | | |
| Area ID | x (meters) | y (meters) | θ (degree) |
| 1 | NA | NA | NA |
| 2 | 0.1 | <0.1 | 0 |
| 3 | <0.1 | <0.1 | 0 |
| 4 | 0.2 | 0.2 | 0.17 |
| 5 | 0.4 | 0.1 | 1.2 |
| 6 | NA | NA | NA |
| 7 | 0.5 | 0.3 | 0.8 |

In Table 5.6, the method is not applicable to the areas given with ID 1 and ID 6, since the robot does not pass in these areas more than once. For ID 7, again the robot passes from this area once, but this area is neighbor with the robot reference area. So a comparison is available. The results given in Table 5.6, shows that the map created by DGL algorithm, is consistent.

**Table 5.7: This table gives the local distortions obtained inside the scan matching with FastSLAM result based on the numbering given in Figure 5.21**

| Maximum Local Distortions for Scan Matching with full FastSLAM Algorithm | | | |
|---|---|---|---|
| | Local Distortions | | |
| Area ID | x (meters) | y (meters) | θ (degree) |
| 1 | NA | NA | NA |
| 2 | 0.1 | <0.1 | 0 |
| 3 | <0.1 | <0.1 | 0 |
| 4 | 0.2 | <0.1 | 0.22 |
| 5 | 0.3 | <0.1 | 0.30 |
| 6 | NA | NA | NA |
| 7 | 0.7 | 0.4 | 0.8 |

According to given results in Table 5.7, the mapping is consistent. Again same reasons are valid for the area tagged with NA. As it is seen from the results a consistent map is constructed.

**Table 5.8: This table gives the global distortions obtained between the DGL algorithm result and original map in sampled areas.**

| Maximum Global Distortions for Scan Matching with DGL Algorithm | | | |
|---|---|---|---|
| | Global Distortions | | |
| Area ID | x (meters) | y (meters) | θ (degree) |
| 1 | 0.3 | 0.4 | 0.96 |
| 2 | 0.6 | 0.3 | 1.62 |
| 3 | 0.6 | 0.2 | 0.58 |
| 4 | 0.3 | 0.4 | 1.57 |
| 5 | 0.6 | 0.5 | 1.33 |
| 6 | 0.5 | 0.1 | 0.59 |
| 7 | 0.3 | 0.1 | 0.20 |

As we can see from Table 5.8, while the robot goes through the path, pose error occurs with the original map and this error about 1.6 meters in translational and about 3.6 degree in rotational. Since the path contains small closed loops and goes totally different place from the start point, we can see the error accumulation in scan matching. Also the long corridor between Area 1 and 2 reduces the scan matching performance, since the area is unstructured scan matching uses odometer data, which also adds the error incoming from odometer. However the accumulated rotational error in Area 5 is corrected while passing back Area 4, and resulted correction is seen in Area 6 .

**Table 5.9: This table gives the global distortions obtained between the full FastSLAM result and original map in sampled areas.**

| Maximum Global Distortions for Scan Matching with FastSLAM Algorithm | | | |
|---|---|---|---|
| | Global Distortions | | |
| Area ID | x (meters) | y (meters) | θ (degree) |
| 1 | 0.5 | 0.1 | 1.68 |
| 2 | 0.1 | 0.6 | 2.44 |
| 3 | 0.3 | 0.7 | 1.12 |
| 4 | 0.6 | 0.2 | 2.15 |
| 5 | 0.4 | 0.6 | 1.03 |
| 6 | 0.4 | 0.5 | 0.08 |
| 7 | 0.6 | 0.2 | 0.12 |

**Table 5.10: This table gives the mean position error and orientation error for Test Area 2 according to the equations 5.1 and 5.2.**

| | Mean Position Error | Mean Orientation Error |
|---|---|---|
| Scan Matching | 0.69 | 1.39 |
| Scan Matching +DGL | 0.54 | 0.97 |
| Scan Matching +FastSLAM | 0.64 | 1.26 |

### Evaluation of Algorithms for Test 2:

In Test Area 2, three algorithms are tested. These are scan matching, scan matching with deterministic global localization (DGL) and scan matching with FastSLAM algorithm.

If they are compared:, in order with increase in pose estimation accuracy and map consistency: main scan matching, scan matching with FastSLAM algorithm, scan matching with DGL algorithm. In this area, scan matching gives very good results, and it calculates the pose of the robot and makes mapping considerably well. Scan matching with DGL algorithm, makes corrections in Area 2, 4 and 5. When the loop is closed, DGL algorithm eliminates the incremental error coming from these areas. Therefore a better accuracy in pose is obtained than the main scan matching algorithm.

Scan matching with FastSLAM algorithm gives worse results than with DGL algorithm. This comes from the divergence of the FastSLAM algorithm if any closed loop does not occur. When the areas in Figure 5.21 are examined a closed loop is seen when the robot goes from Area 2 to Area 3 and goes back to Area 2. However, since the robot goes a long distance from the startpoint to Area 2, the particles are scattered in a wide area. When a landmark is observed its position is set into the particle according to the estimated robot pose in particle. When this landmark is seen again all the particles are weighted, this will survive the ones that correct the error between the times when the landmark is observed first and observed again. However the particles are still scattered because of the error till the landmark is initially observed. As a result of this, with elimination of some paths by weighting, the distribution of paths is corrupted and a bias in pose of the robot occurs. This affects the following pose estimations in particles. As a result, while a correction in closed loops occurs, because of high diversity in pose of the particles, the path of the robot and mapping is deteriorated. The result of this situation in mapping is given as:

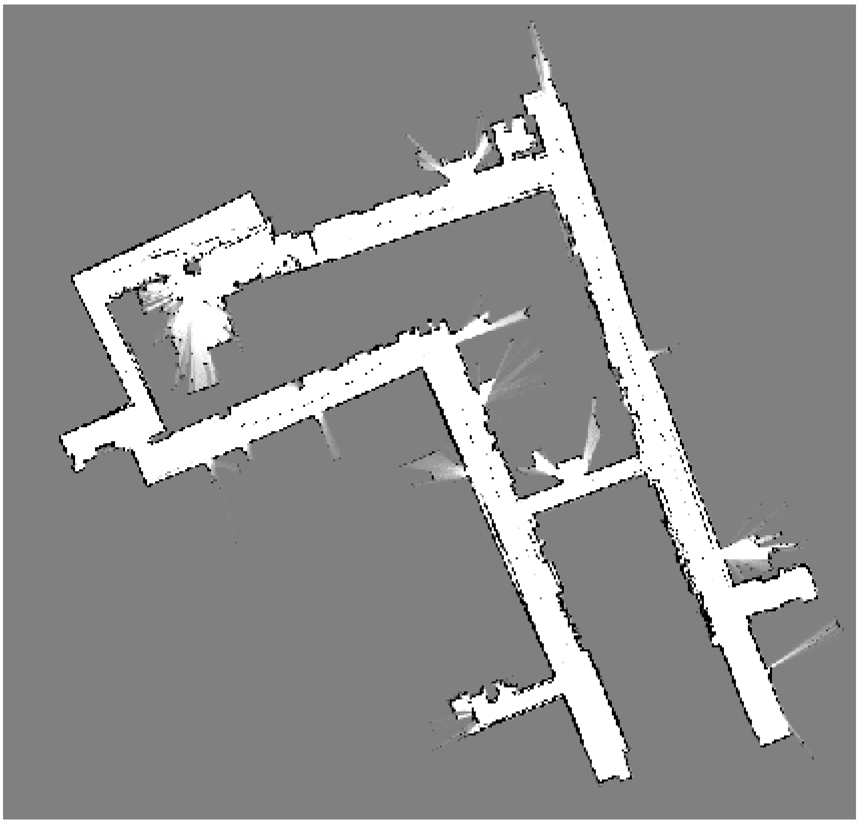


**Figure 5.22: The occupancy grid map obtained from scan matching with full FastSLAM algorithm (with high Gaussian noises in poses of particles)**

When the values of Gaussian noise in pose of the robot are decreased, the robot path and mapping quality increase. In fact when the robot comes to Area 2, the diversity of the particles becomes much smaller. But the important point is whether the increase in diversity of particles after landmark is initially observed is enough to correct the accumulated error till the landmark is reobserved. As a result of this the effect of corrections in closed loop cases decreases. Therefore the result converges to the main scan matching result. Consequently an accuracy and mapping come into existence between main scan matching and scan matching with DGL algorithm results, which is given in Figure 5.14.

## 5.3 Test 3: Part of the Intel Jones Farms Campus, Oregon - LADAR and Odometer Data Taken from RADISH Data Repository

The test is made with;

- Pioneer 2DX Robot (Differential Drive)
- SICK LMS 200 Laser Scanner
- 180 degree LADAR scan with 1 degree interval

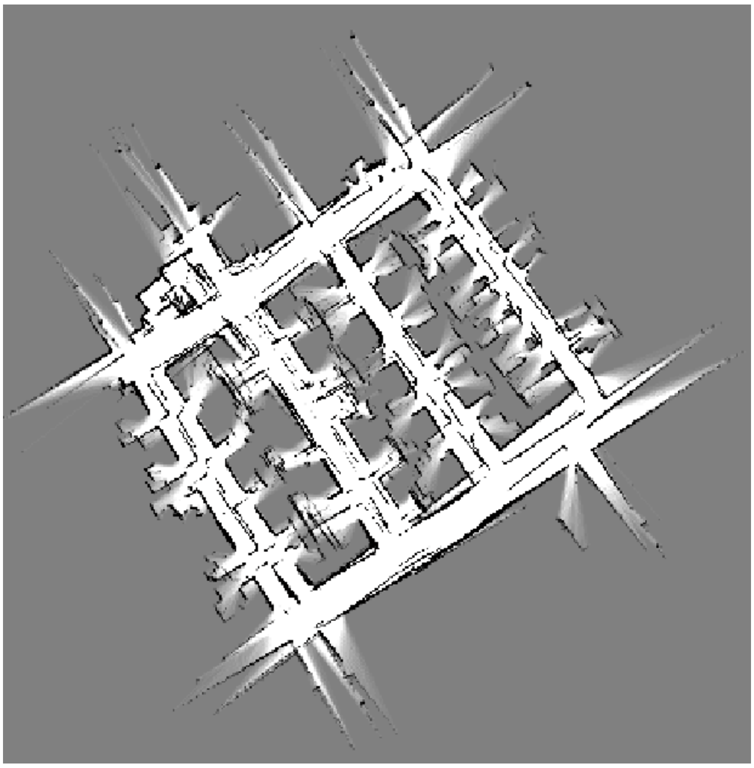


**Figure 5.23: The occupancy grid map obtained from main scan matching.**

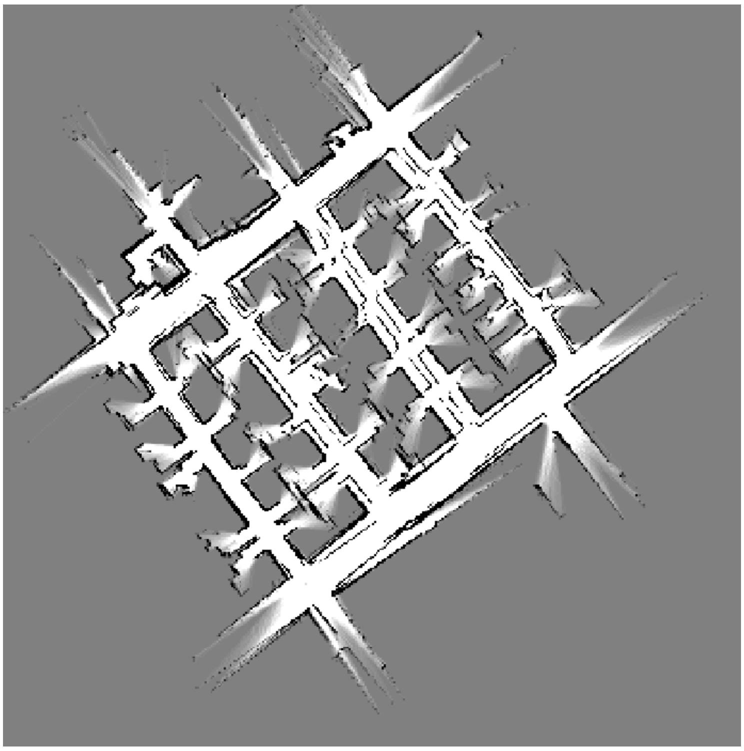


**Figure 5.24: The occupancy grid map obtained from scan matching with DGL algorithm.**

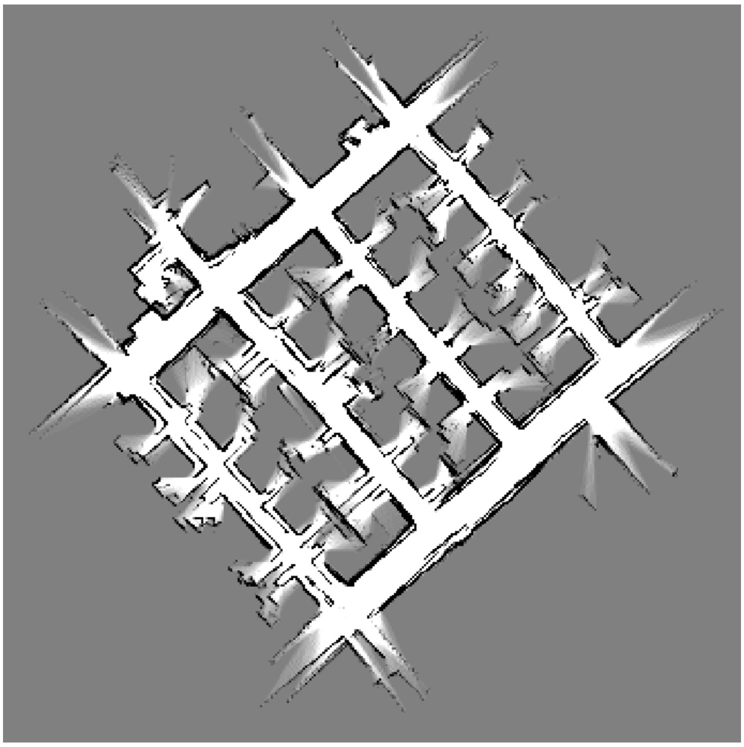


**Figure 5.25: The occupancy grid map obtained from scan matching with full FastSLAM output.**

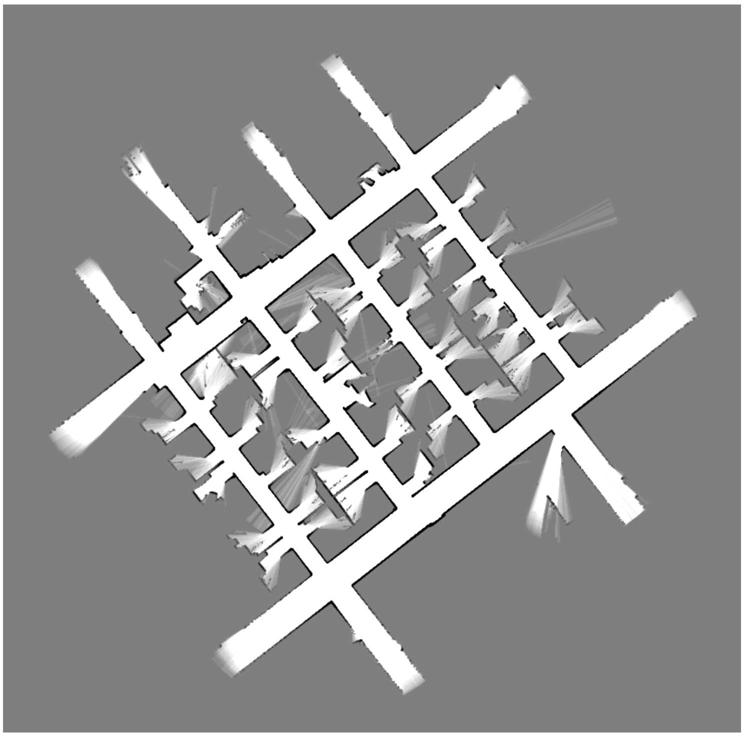


**Figure 5.26: The real map given by Intel Lab.**

In Figure 5.23 the scan matching result is presented that we have obtained from the raw LADAR and odometer data. In Figure 5.24, the scan matching with DGL algorithm result is presented. In Figure 5.25, the scan matching with FastSLAM algorithm result is presented. In Figure 5.26, the real map is presented given by Part of the Intel Jones Farms Campus, Oregon, which is the source of dataset taken from the Radish Data Repository [22].

If Figure 5.23 and Figure 5.26 are compared, scan matching seems to suffer from the incremental error accumulated in pose. This is seen when the robot goes on multiple closed loops in this situation. This error is about 2 meters in translation and 3 degrees in rotation (can be seen from Figure 5.28) in 239.6 meters distance traveled, but it is enough to demolish the map when multiple passes are occurred. But it is still so successful when it is compared with a map created from raw odometer data. The main scan matching output without any multiple passes can be seen in Figure 5.27. This figure gives an idea about the effect of incremental error between one loop and multiple loop cases.

If Figure 5.24 and Figure 5.26 are compared, as a result of DGL algorithm, the errors in closed loop paths are corrected and a map very close to the real map is constructed. The pose error between these can be seen in Table 5.13.

If Figure 5.25and Figure 5.26 are compared, FastSLAM algorithm gives pretty good results, a better localization and mapping is achieved than DGL algorithm. Most of the doorways are mapped with less than 0.5 degree rotational error and 0.3 meters translational error, that is given in Table 5.14. The given errors are occurred when the observed landmarks could not be matched with the previously seen ones, so they are added as new ones.

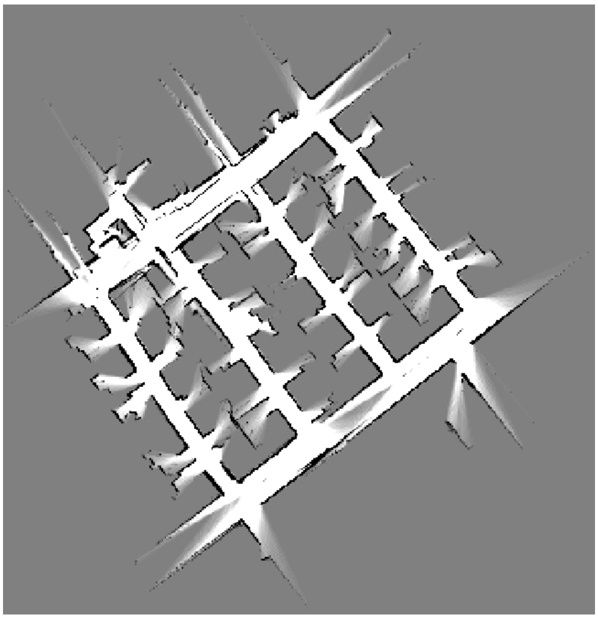


**Figure 5.27: The scan matching output without multiple loops.**

In Figure 5.27, the scan matching result when the robot goes and comes to the startpoint without making any loop (sample data taken from the total one) is given. The total error accumulated in pose is less than 0.4 meters in translation and 1.2 degree in rotation for 88.6 meters traveled.

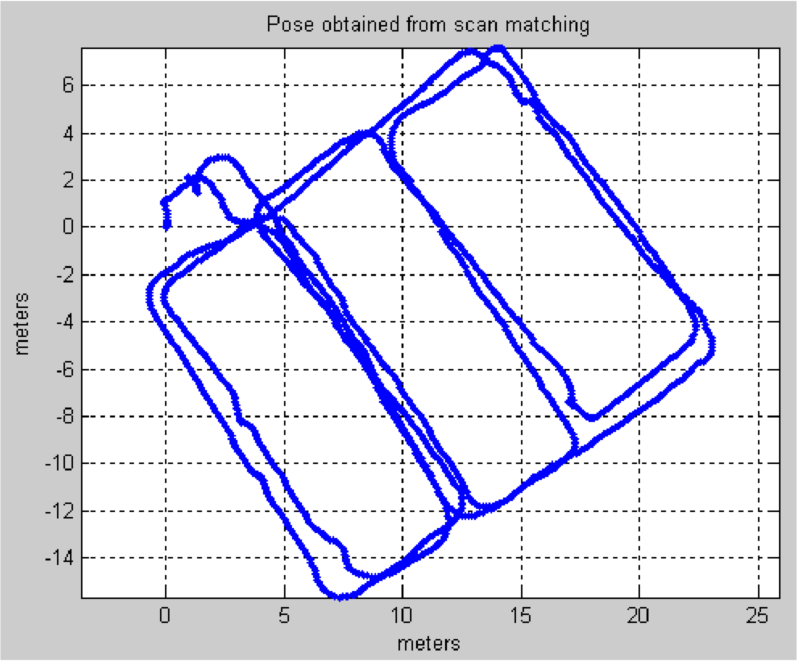


**Figure 5.28: The pose information obtained from scan matching algorithm.**

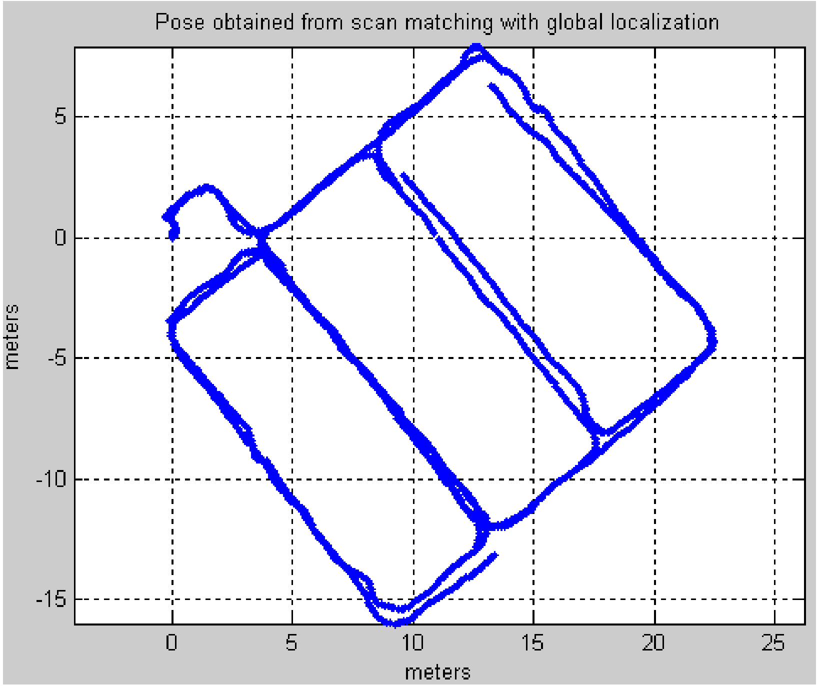


**Figure 5.29: The pose information obtained from scan matching with DGL algorithm.**

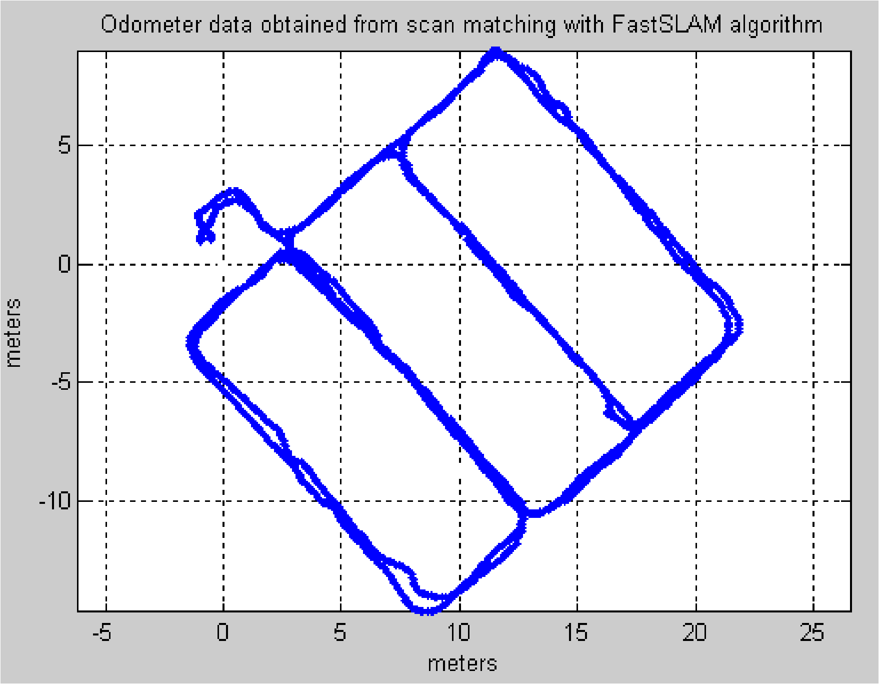


**Figure 5.30: The pose information obtained from scan matching with full FastSLAM algorithm.**

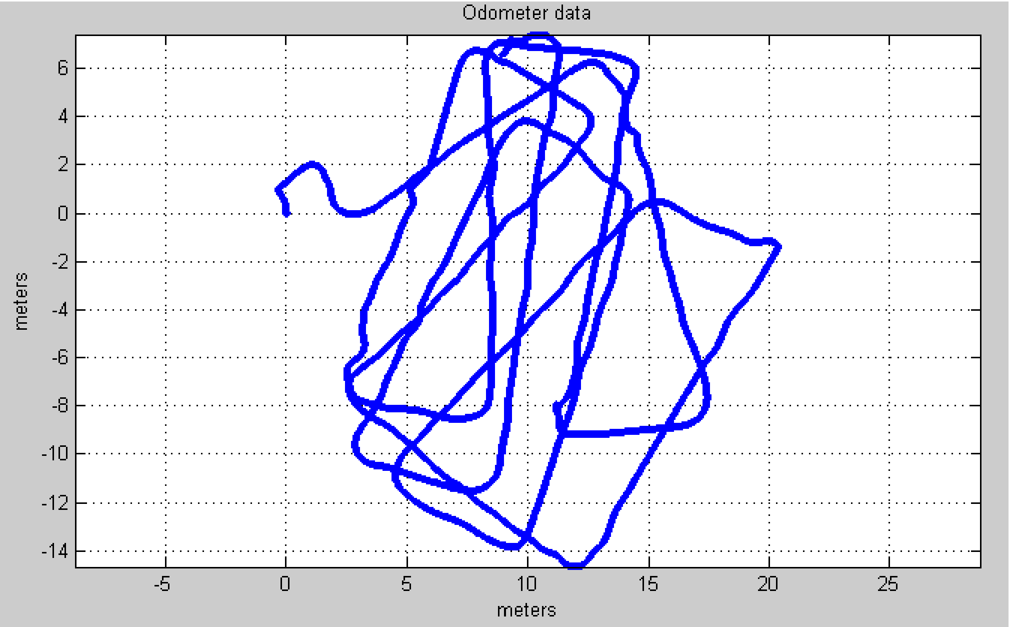


**Figure 5.31: The odometer data given while the robot is moving on the map given in Figure 5.26.**

In Figure 5.28, the pose information obtained from scan matching algorithm, in Figure 5.29, the pose information obtained from scan matching algorithm with deterministic global localization (DGL) is presented. In Figure 5.30, the pose information obtained from scan matching algorithm with FastSLAM algorithm is presented. In Figure 5.31, the raw odometer data that is taken from the dataset is presented.

In Figure 5.29, the jumps in the pose data obtained from the scan matching algorithm are coming from the corrections in global localization part. These are the corrections based on the features observed more than once.

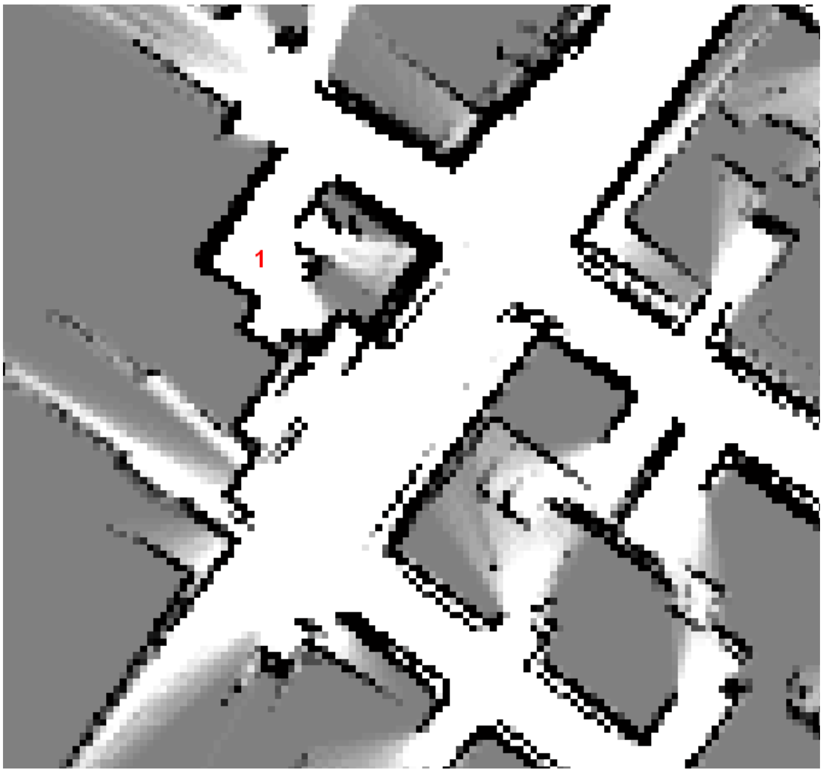


**Figure 5.32: The startpoint and also endpoint of the robot for this dataset is shown with Number 1, which is the zoom in of left part of scan matching obtained given in Figure 5.24.**

As you can see from Figure 5.32, the start point and endpoint of the robot is in same place. The robot has traveled totally about 239.6 meters distance. When we have looked at the thickness of the walls around the point shown with number 1, it is in 1 grid size thickness, which means there is no translational and rotational error seen when the robot has come to the endpoint. But since there are multiple close loops (it can be seen on Figure 5.31), while the robot passing through the places that it has seen, it corrects its translation and rotation. So when it has come to the start point all the error has gone, because of global matching ability of the algorithm.

For this purpose we can look at the places where the slippage in the orientation of the walls increases to find the maximum local error during the path.

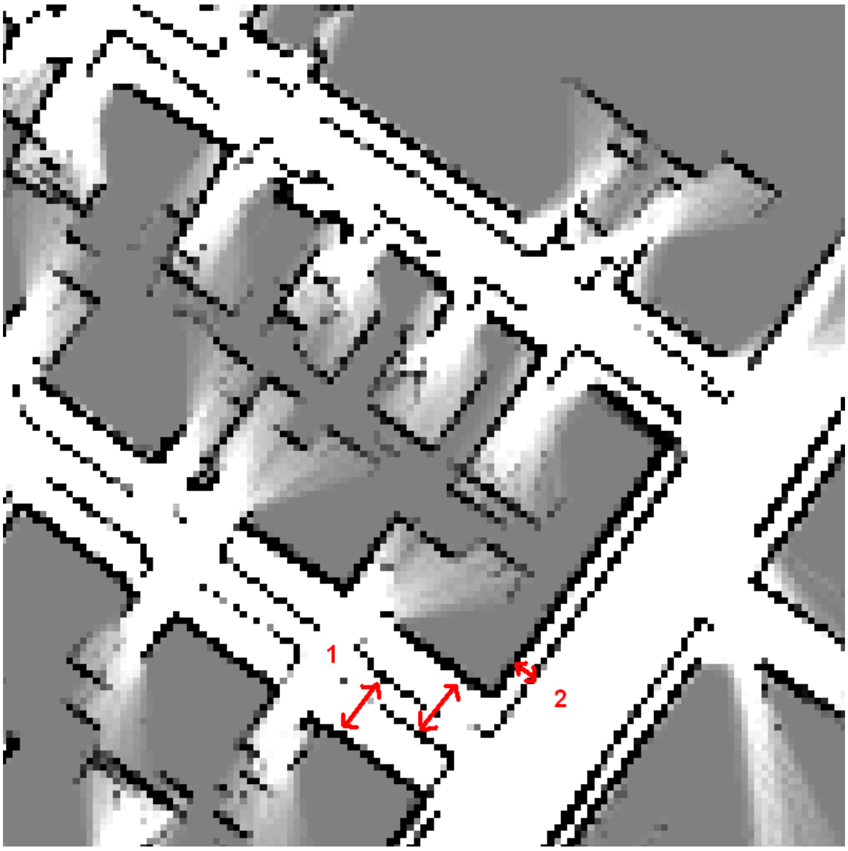


**Figure 5.33: The area that maximum slippage is occurred.**

When we look at the Figure 5.33, taken from the top part of Figure 5.24, which is second corridor from the top, maximum slippage is seen. The two arrows close to Number 1 are the footprints of the corridor before and after the slippage have occurred. The translational error occurred in axis perpendicular to the wall during this navigation is about 6 grids, which is 0.6 meters error The local translational error occurred perpendicular to the wall in the area given with Number 2 is about 4 grids, which is about 0.4 meters. The maximum resulting local rotational error seen is about 0.7 degree in this area.

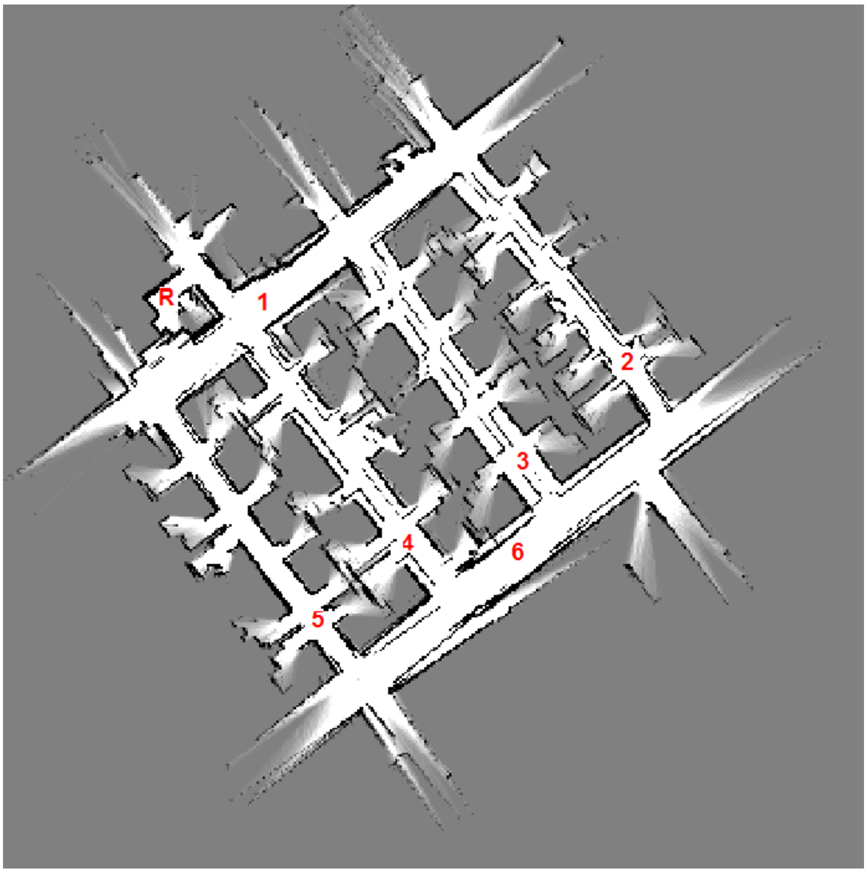


**Figure 5.34: The figure showing the check areas that maximum global/local distortions are calculated between scan matching result and original map.**

The accuracy of the x, y values is 0.1 meters, which is the grid size; also accuracy of the θ values is less then 0.3 degree, because of the grid size. The results are given as absolute values.

**Table 5.11: This table gives the local distortions obtained inside the DGL algorithm result based on the numbering given in Figure 5.34.**

| Maximum Local Distortions for Scan Matching with DGL Algorithm | | | |
|---|---|---|---|
| | Local Distortions | | |
| Area ID | x (meters) | y (meters) | θ (degree) |
| 1 | 0.3 | 0.4 | 0.27 |
| 2 | 0.3 | 0.4 | 0.57 |
| 3 | 0.4 | 0.5 | 0.34 |
| 4 | 0.3 | 0.3 | 0.72 |
| 5 | 0.1 | 0.1 | 0.47 |
| 6 | 0.3 | 0.3 | -2.87 |

The results given in Table 5.11 shows that the map we have created by DGL algorithm is consistent. It seems an error occurs in rotation θ in Area 6, you can also see this rotational error in the map, but this error corrected by the fault correction and global localization part of the algorithm. So when the robot goes to the other areas this θ value again comes below one degree.

**Table 5.12: This table gives the local distortions obtained inside the full FastSLAM result based on the numbering given in Figure 5.34.**

| Maximum Local Distortions for Scan Matching with Full FastSLAM Algorithm | | | |
|---|---|---|---|
| | Local Distortions | | |
| Area ID | x (meters) | y (meters) | θ (degree) |
| 1 | 0.1 | 0.1 | 0.24 |
| 2 | 0.1 | 0.1 | 0.25 |
| 3 | 0.1 | 0.1 | 0.28 |
| 4 | 0.3 | 0.2 | 0.34 |
| 5 | 0.2 | 0.2 | 0.29 |
| 6 | 0.3 | 0.1 | 0.62 |

As we can see from the table, the distortion always stays below one meter in X and Y axis and also θ usually stays below one degree error.

**Table 5.13: This table gives the global distortions obtained between the DGL algorithm result and original map in sampled areas.**

| Maximum Global Distortions for Scan Matching with DGL Algorithm | | | |
|---|---|---|---|
| | Global Distortions | | |
| Area ID | x (meters) | y (meters) | θ (degree) |
| 1 | 0.1 | 0.1 | 0.52 |
| 2 | 0.2 | 0.4 | 0.43 |
| 3 | 0.4 | 0.2 | 0.28 |
| 4 | 0.2 | 0.2 | 0.33 |
| 5 | 0.6 | 0.1 | 0.47 |
| 6 | 0.1 | 0.3 | 2.95 |

In Table 5.13, the global worst distortions are given relative to the original map given. As we can see from the table, the distortion always stays below one meter in X and Y axis. Also we can say that the θ usually stays below one degree error. It seems an error occurs in rotation θ in Area 6, you can also see this rotational error in the map, but this error corrected by the fault correction and global localization part of the algorithm. So when the robot goes to the other areas this θ value again comes below one degree.

**Table 5.14: This table gives the global distortions obtained between the full FastSLAM result and original map in sampled areas.**

| Maximum Global Distortions for Scan Matching with Full FastSLAM Algorithm | | | |
|---|---|---|---|
| | Global Distortions | | |
| Area ID | x (meters) | y (meters) | θ (degree) |
| 1 | 0.1 | 0.2 | 0.27 |
| 2 | 0.1 | 0.1 | 0.30 |
| 3 | 0.2 | 0.1 | 0.17 |
| 4 | 0.1 | 0.3 | 0.26 |
| 5 | 0.2 | 0.3 | 0.28 |
| 6 | 0.2 | 0.2 | 0.21 |

**Table 5.15: This table gives the mean position error and orientation error for Test Area 3 according to the equations 5.1 and 5.2.**

| | Mean Position Error | Mean Orientation Error |
|---|---|---|
| Scan Matching | 0.83 | 3.72 |
| Scan Matching +DGL | 0.39 | 0.81 |
| Scan Matching +FastSLAM | 0.26 | 0.25 |

## Evaluation of Algorithms for Test 3:

In given Test Area 3, three algorithms are tested. These are scan matching, scan matching with deterministic global localization (DGL) and scan matching with FastSLAM algorithm.

If they are compared, it is seen that scan matching gives a great improvement in pose estimated when it is compared with raw odometer output. The drawback of scan matching algorithm is the pose error accumulates through the distance traveled increases. In fact, in multiple loop cases it can be seen as the distortions on the map. In single loop cases, accumulated error can be seen when the map created is compared with original map by using positions of chosen features in both map.

Scan matching with DGL algorithm gives better results than the main scan matching algorithm. As it is seen the pose error accumulated in scan matching is corrected with re-observation of landmarks. The problems about this algorithm are that it corrects itself to the error level when it has first seen the landmark observed, there is no any correction in positions of the landmarks observed or the previous path of robot. This causes corruption in pose when it has re-observed a landmark which has previously seen with bad pose estimate. However this error never passes the error level that the pose error of the scan matching in place landmark is observed. Also the pose corrections result as jumps in path of the robot, which can be seen in Figure 5.29. In fact, these jumps could cause problems in navigation of the robot (such as for path planning or obstacle avoidance).

Scan matching with FastSLAM algorithm gives better accuracy than with DGL algorithm. The main advantage of FastSLAM algorithm comes from using particle filter and extended kalman filter all together. By using multiple particles different robot paths can be determined, and the optimal path is estimated according to the landmark observations. Also by using EKF, the pose and landmark positions are updated. As a result, in full SLAM implementation a map with higher consistency is obtained, which shows better pose estimation is obtained, given in Figure 5.25 and Figure 5.30.

## 5.4 Online SLAM and Full SLAM Comparison:

Online SLAM problem involves estimating the posterior over the momentary pose along with map. On the other hand full SLAM calculates a posterior over the entire path along with the map instead of just current pose. Thus full SLAM, needs the entire path estimate from the startpoint in each particle and also when an update comes, the entire map is need to be updated according to all updated path. For this purpose, online SLAM is applicable as real time working, while full SLAM is mostly applicable as post processing because of its high process load. In the thesis, the FastSLAM results are demonstrated as full SLAM results.

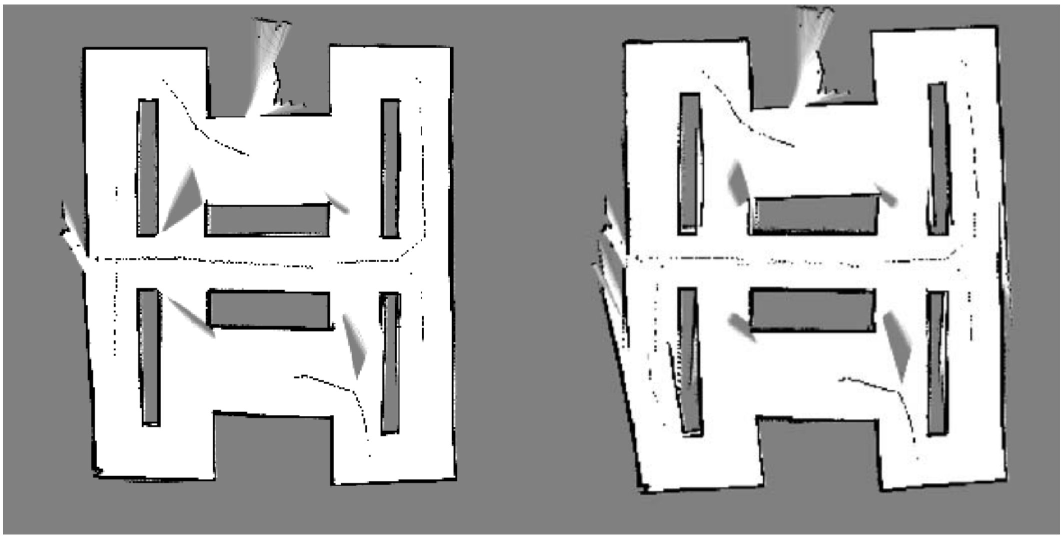


**Figure 5.35:Aselsan Indoor Area: The figure in the left belongs to the full FastSLAM result while the figure in the right belongs to the online FastSLAM result.**

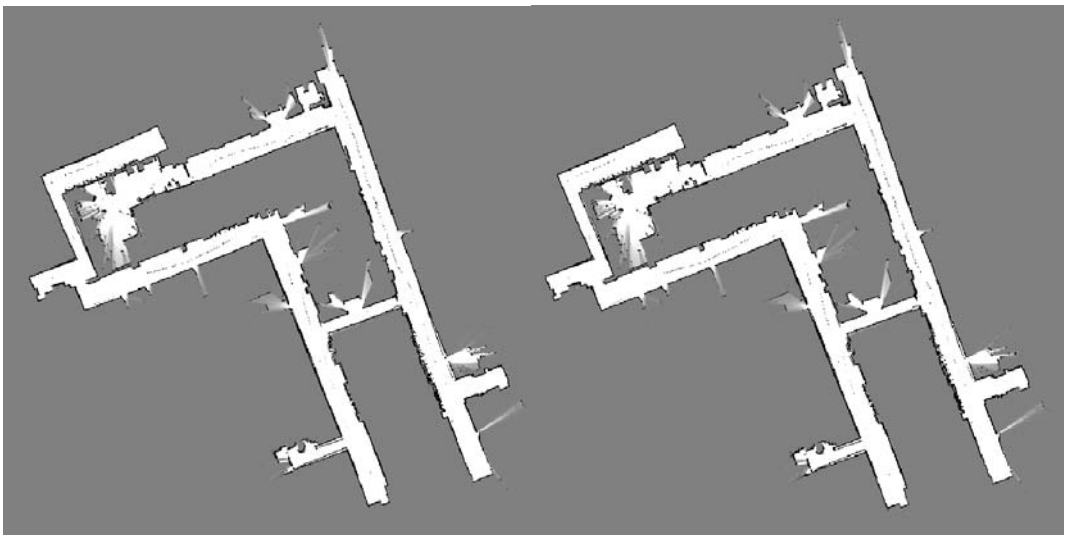


**Figure 5.36: Ancona Univercity: The figure in the left belongs to the full FastSLAM result while the figure in the right belongs to the online FastSLAM result.**

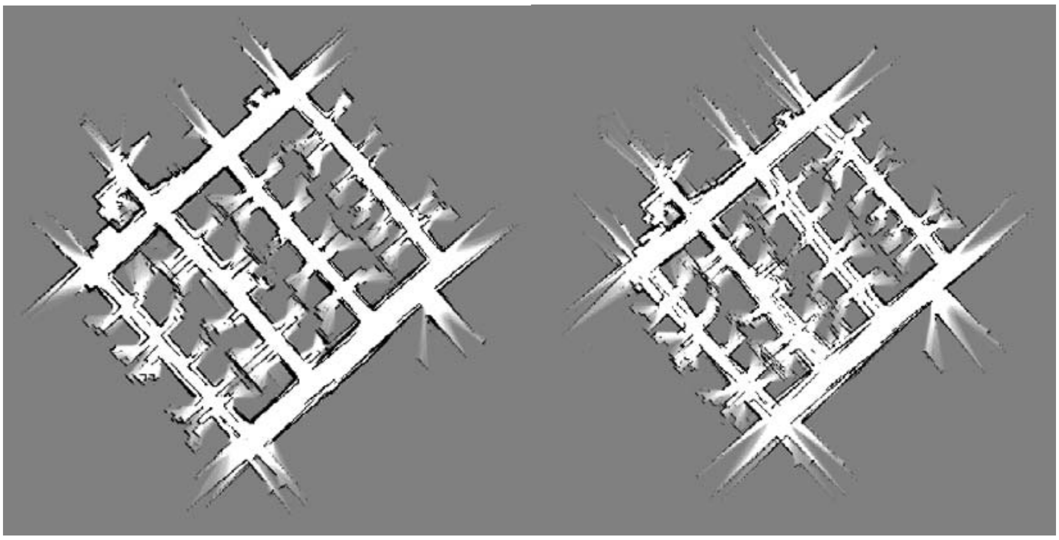


**Figure 5.37: Intel Research Lab: The figure in the left belongs to the full FastSLAM result while the figure in the right belongs to the online FastSLAM result.**

In Figure 5.35 and Figure 5.37, an illustration giving the online FastSLAM and full FastSLAM is presented. As it is seen, in full SLAM, previous poses of robot and all constructed map till that timestep is updated according to the information registered in the most weighted particle in loop closing cases. Therefore there is back propagation in correction of pose error in full SLAM case. However the accumulated error till the landmark reobservation stays on the map in online SLAM. As a result full SLAM gives better pose estimate and more consistent map, in the other hand online SLAM gives real time working ability. In this thesis mapping in full FastSLAM is presented as post processing since occupancy grid mapping is used and in occupancy grid map, each time about 180 LADAR scans need to be considered. However, mapping in full FastSLAM can also be used as real time working when the map consists of high level features like line segments or landmarks instead of raw laser scans.

## 5.5 Computational Complexities of Algorithms:

The algorithms are evaluated whether they can be used in real time applications with a robot. This test is done on the data of Intel Research Area, which is given in Section 5.3. Since this data is most appropriate one for all three algorithms given. In fact, it has many features to be used as landmarks and contains many closed loops for FastSLAM and DGL algorithms (especially to see the process time, in weight calculation and feature update steps). Also it is well structured, so main scan matching needs to work more feature extraction and matching operations. The LADAR data in our tests is taken maximal with 5 Hz frequency. Therefore 0.2 second timestep intervals are enough for the real time application.  During our tests, Intel Core 2 Due T7200 2.00 GHz Mobile Processor is used. The results obtained are presented:

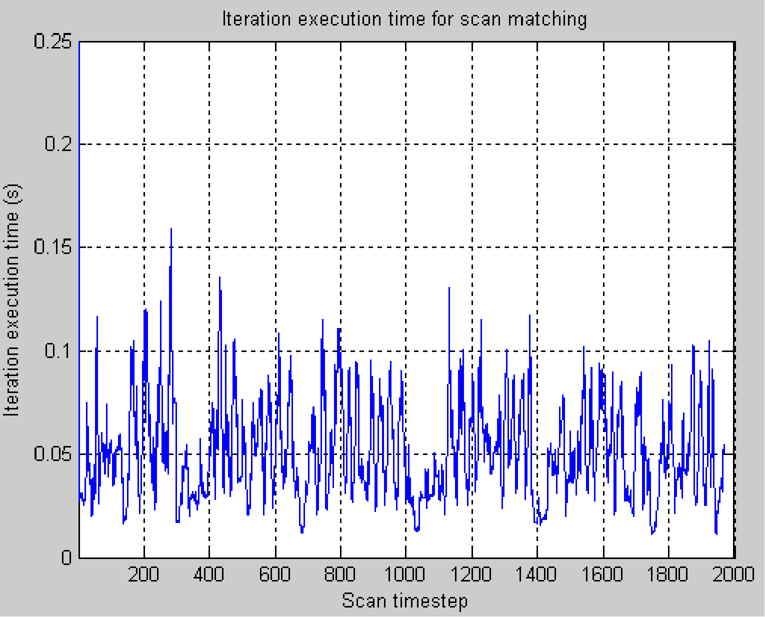


**Figure 5.38: The illustration showing the process time for main scan matching (averaged from 10 trials).**

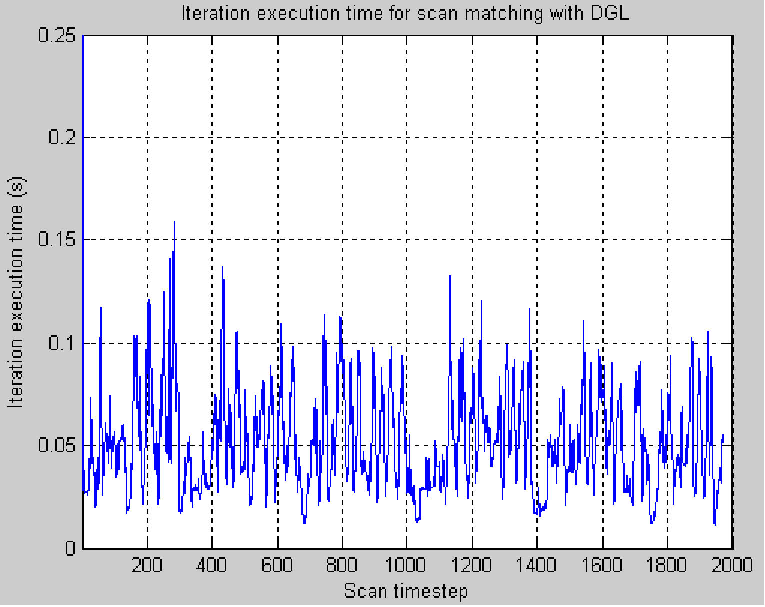


**Figure 5.39: The illustration showing the process time for scan matching with DGL algorithm (averaged from 10 trials)..**

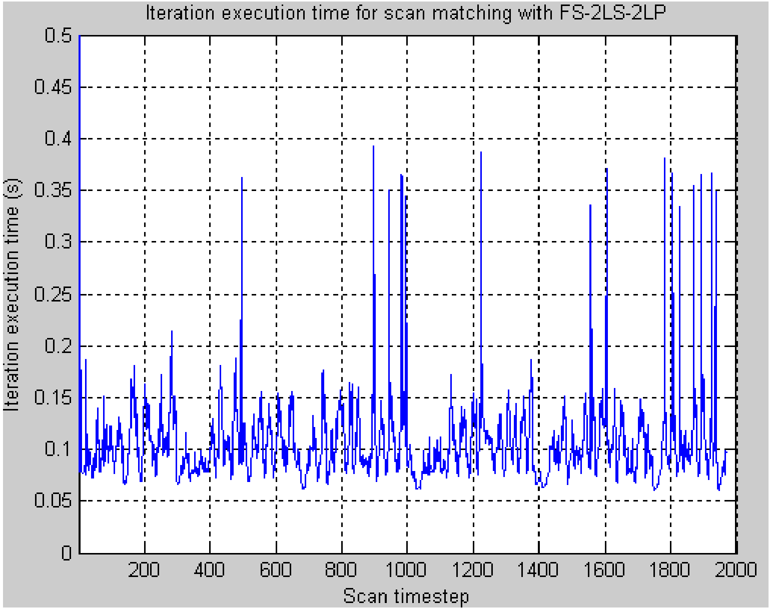


**Figure 5.40: The illustration showing the process time for scan matching with FastSLAM algorithm (with 200 particles) (averaged from 10 trials).**

**Table 5.16: The table showing the mean and maximum timesteps obtained during the execution of algorithms in Intel Research Area (averaged from 10 trials).**

| | Mean Iteration Exec. Time (s) | Max Iteration Exec. Time (s) |
|---|---|---|
| Scan Matching | 0.0497 | 0.1563 |
| Scan Matching with DGL | 0.0518 | 0.1586 |
| Scan Matching with FastSLAM (200 particles) | 0.1043 | 0.3924 |

As it is seen from the figures and table, for all the process time in mean, is below 0.2 seconds. For main scan matching and the one with DGL algorithm, the maximum process time is still below the 0.2 second. However, the one with FastSLAM algorithm (with 200 particles), the instantaneous process time can be about 0.4 second given in Figure 5.38. But this corresponds to 2 laser scans. In fact, because of low rotational and translational speed of the robot, the algorithms are still working with 4 laser scan steps in the given datasets. Therefore we can say that all these algorithms can work real time in given applications. For the localization performance scan matching with DGL algorithm needs to be chosen. It contains a low increase in computational complexity referenced to main scan matching, given in Figure 5.39 and Figure 5.40, and an important improvement in localization. Also for a better performance, especially for large scale environments with multiple loops, scan matching with FastSLAM algorithm needs to be used in spite of its doubled computational complexity referenced to scan matching with DGL algorithm.

## 5.6 The Comparison between the FS_2LS_2LP and FS_3LS_1LP Algorithms:

In this part, scan matching with two different implementation of FastSLAM algorithm is compared. We can define FastSLAM algorithm with two landmark positions registered in particles to define a landmark with its orientation as FS_2LS_2LP algorithm. FS_2LS_2LP means FastSLAM with two landmark states and two landmark positions. Also we can define FastSLAM algorithm with orientation information addition to position registered in particles to define a landmark as FS_3LS_1LP algorithm. FS_3LS_1LP means FastSLAM with three landmark states and one landmark position. To compare their performances, they are tested with the given test areas:

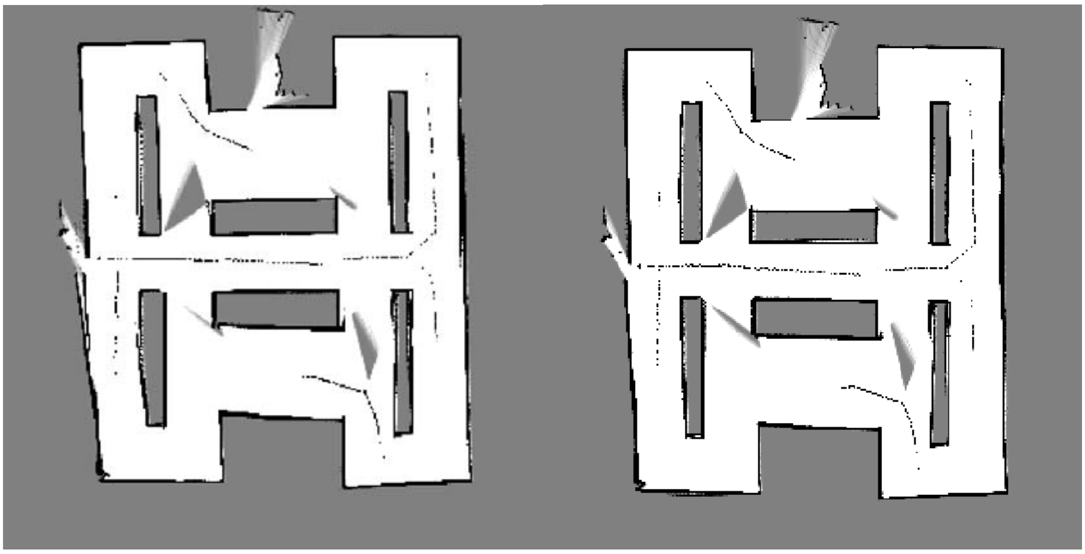


**Figure 5.41: ASELSAN Indoor Area: The figure in the left belongs to the FS_3LS_1LP result while the figure in the right belongs to the FS_2LS_2LP result.**

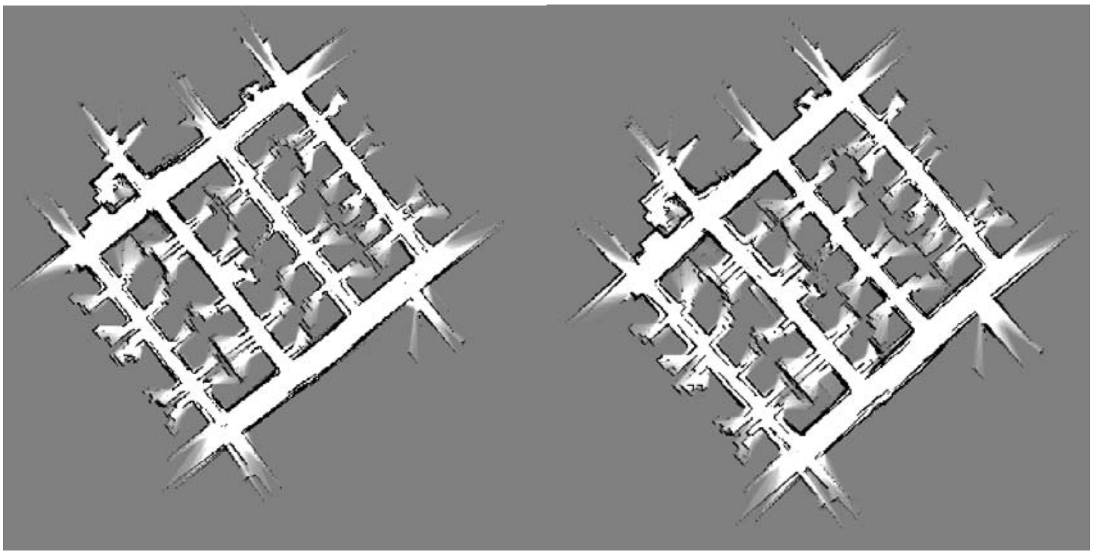


**Figure 5.42: Intel Research Lab: The figure in the left belongs to the FS_3LS_1LP result while the figure in the right belongs to the FS_2LS_2LP result.**

As it is seen from Figure 5.41 - Figure 5.42, the distortions on the mapping are about same in both algorithms. It can be said as scan matching with FS_2LS_2LP and FS_3LS_1LP algorithms have similar outputs. Also this shows a modified version of FastSLAM, which is given as FS_3LS_1LP algorithm is successfully implemented. The resulting computational complexities of these two algorithms can be given as:

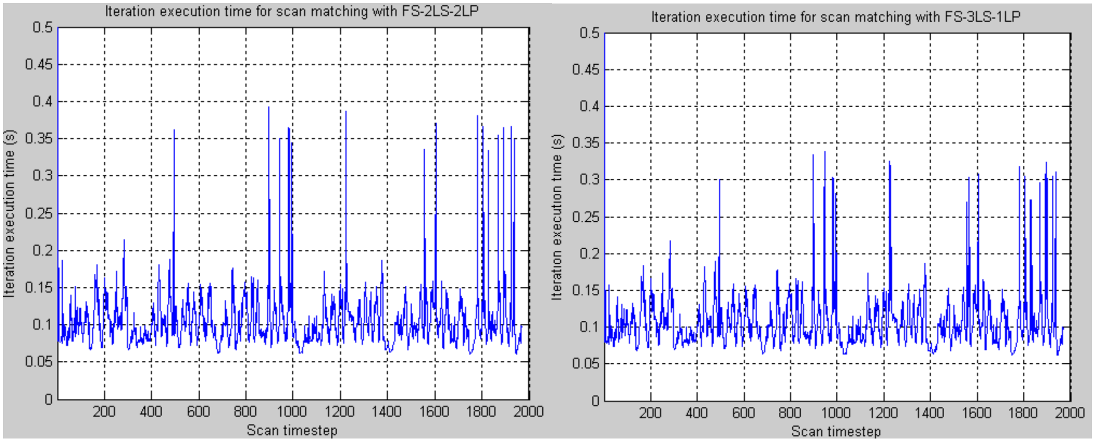


**Figure 5.43: The illustration showing the iteration execution time for scan matching with FS_2LS_2LP and FS_3LS_1LP algorithm (with 200 particles) (averaged from 10 trials).**

**Table 5.17: The table showing the mean and maximum iteration execution time obtained during the execution of algorithms in Intel Research Area.**

| | Mean Iteration Exec. Time (s) | Max Iteration Exec. Time (s) |
|---|---|---|
| Scan Matching with FS_2LS_2LP | 0.1043 | 0.3924 |
| Scan Matching with FS_3LS_1LP | 0.1069 | 0.3379 |
| Scan Matching with FS_2LS_2LP Excluded Landmark Re-observation Execution Time | 0.1015 | NA |
| Scan Matching with FS_3LS_1LP Excluded Landmark Re-observation Execution Time | 0.1027 | NA |

According to the results given in Figure 5.43 and in Figure 5.14, FS_2LS_2LP algorithm has a little bit better iteration execution time. However FS_3LS_1LP algorithm has better maximum iteration execution time. The execution time of these two algorithms is taken, except the places where landmark re-observation is occurred. It is seen that their mean execution time difference decreases. This overlaps with the theoretical operation of FastSLAM algorithm, since in these steps only prediction update occurs, no weight calculation or measurement update occurs, which means that this part is independent from the landmarks and landmark related calculations and need to have same mean execution time.

Maximum iteration execution time is seen in landmark re-observation cases. The difference between maximum iteration execution times comes from the number of calculations occurred in the algorithms. In FS_2LS_2LP algorithm, number of landmark positions is doubled, so the time in all the relevant parts with landmark re-observation, which are proposal sampling, the measurement update in EKF algorithm and the weight calculation part is doubled. However in FS_3LS_1LP algorithm, this time is increased since all the matrix productions, jacobians, inverse matrix calculations are made with higher dimension matrices (since the landmark is stated with three parameters).

A definite decision could not be given between the computational complexities of the algorithms. Since the Matlab is optimized for matrix based calculations, which could give some more benefit in matrix calculations instead of loops. In the other hand, Cholesky factorization is used for inverse calculations, which gives an enhancement, but search tree is not used for data association, which gives an aggravation. As it is seen from the pseudo codes of the algorithm, the increase in number of landmarks affects the computational complexities and total number of parameters linearly, on the other side the increase in parameters of landmarks affects the computational complexities and total number of parameters quadratic.

## 5.7 The Sensitivity Analysis of the Algorithm:

In this part, some most important parameters of the algorithm will be changed in an interval and their effects to the performance of the algorithm are analyzed:

### 5.7.1 The Change in Number of Particles for FastSLAM Algorithm:

Number of particles is one of the most important parameter for FastSLAM algorithm, It decides with how many particles that the given covariance in pose of the robot will be expressed. This parameter changes for different environments mapped. Since size of the loop closings, the error incoming from the scan matching algorithm (for standard FastSLAM the odometer error), the sequence of closed loops in total path… effect the number of particles needed. For this purpose, Intel Research Lab (Test Area 3 given in Figure 5.34), which is the most complex environment in datasets for FastSLAM algorithm, is analyzed. The performance is calculated as the mean and variance of the localization error seen in six areas given (in Figure 5.34). The mapping of this environment with different number of particles is presented below for scan matching with full FastSLAM algorithm in Figure 5.44 and Figure 5.45:

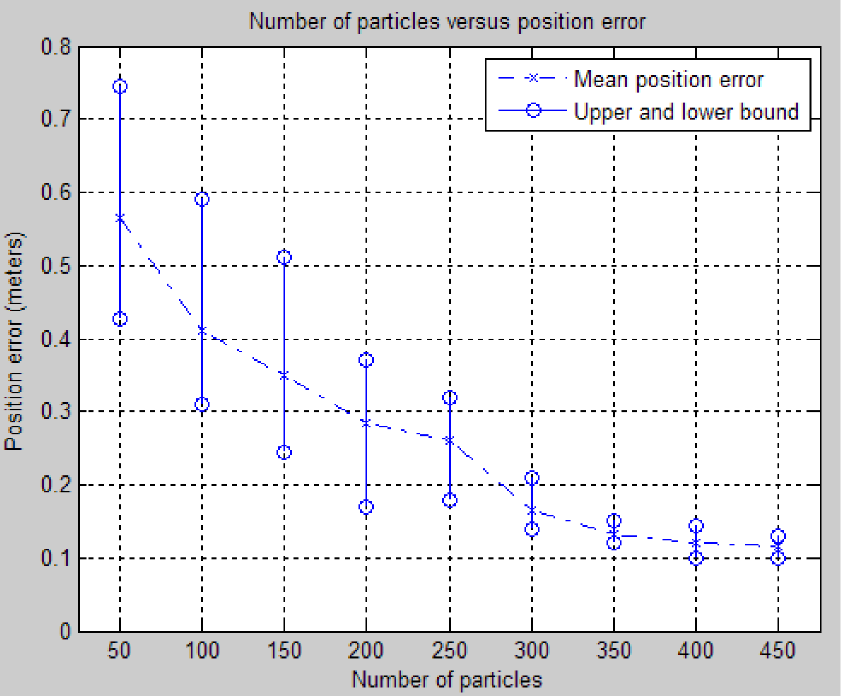


**Figure 5.44: In figure, the change in mean position error for the change in number of particles is presented (averaged from 10 trials).**

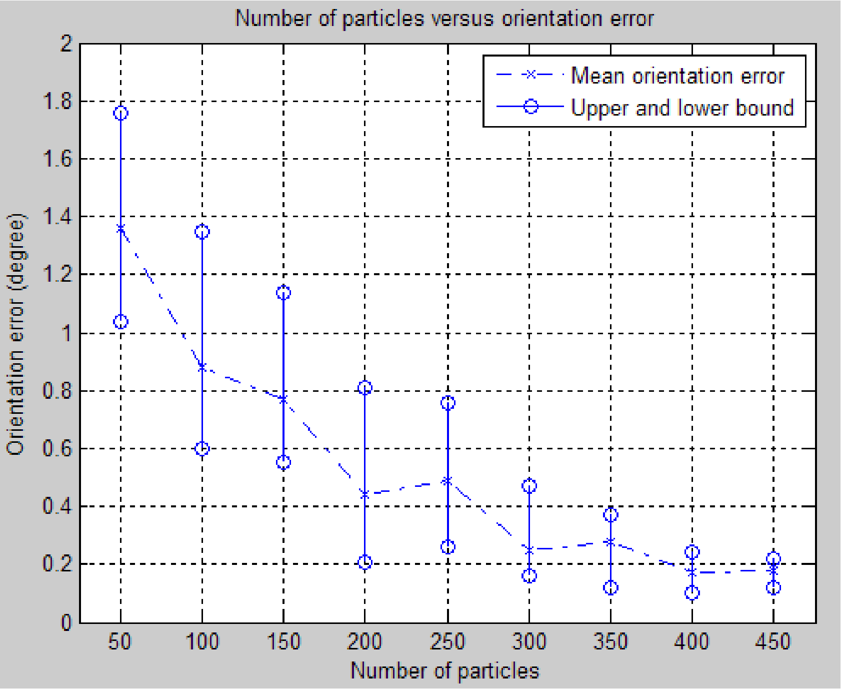


**Figure 5.45: In figure, the change in mean orientation error for the change in number of particles is presented (averaged from 10 trials).**

In Figure 5.44, the change in mean positioning error with respect to number of particles is given. In Figure 5.45, the change in mean orientation error with respect to number of particles is given. Also for the trials, the worst and best results are given. As it is seen from the results, the performance of the algorithm increases with increment in number of particles. Also the the interval between the bounds decreases. In other hand, while the number of particles decrease, the results obtained from FastSLAM algorithm comes closer to the main scan matching results, but still for 50 particles FastSLAM has much better results (it is seen for main scan matching given in Figure 5.23, the mean position error is about 1.1 meters and mean orientation error is about 2.1 degree. ). After 300 particles, considerable improvement does not occur in mean position and orientation error; also the results are so close the zero localization error. However by the increase in particles, mean and maximum iteration execution time also increases, which has given in Table 5.18. After 300 particles, more than 0.6 maximum iteration execution time is seen, which corresponds to loss of three LADAR scans.

**Table 5.18: The table showing the mean and maximum iteration execution time obtained during the execution of FS_2LS_2LP in Intel Research Area.**

| Number of Particles | Mean Iteration Exec. Time (s) | Max Iteration Exec. Time (s) |
|---|---|---|
| 50 | 0.0911 | 0.2084 |
| 100 | 0.0934 | 0.2670 |
| 150 | 0.0974 | 0.3324 |
| 200 | 0.1015 | 0.3924 |
| 250 | 0.1039 | 0.4519 |
| 300 | 0.1082 | 0.5115 |
| 350 | 0.1117 | 0.5796 |
| 400 | 0.1150 | 0.6433 |
| 450 | 0.1189 | 0.7235 |

Mapping results obtained during the trials with maximum and minimum number of particles is given below in Figure 5.46:

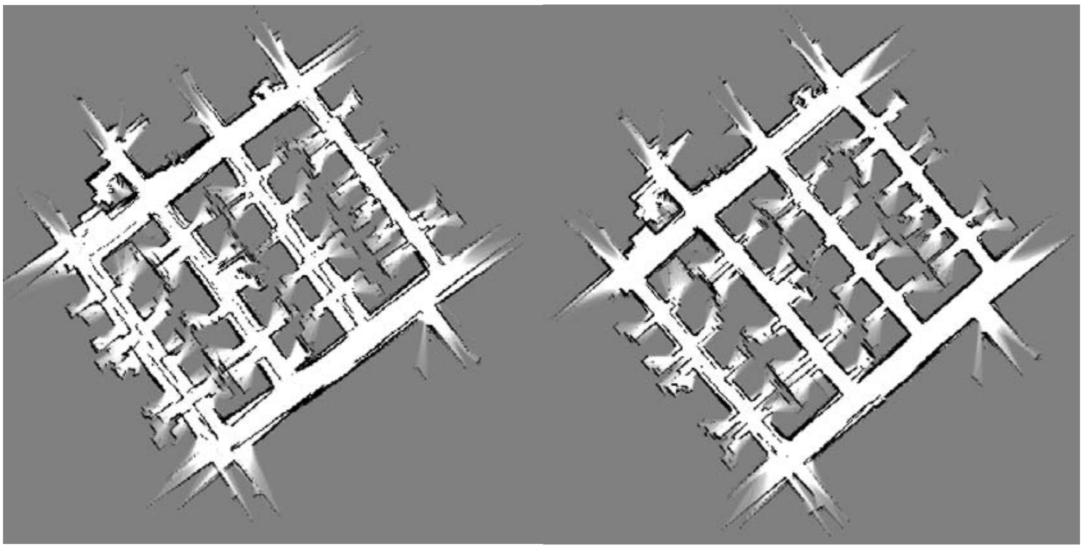


**Figure 5.46: The figure illustrating the mapping performance of FS_2LS_2LP algorithm for 50 and 450 particles. The one on the left for 50 particles, the one on the right is for 450 particles.**

### 5.7.2 The Change in Scan Interval of LADAR:

In this part, the affect of change in scan interval of LADAR to the performance of localization is tested. For this purpose the scan interval has decreased with 20 degree intervals, which has resulted with datasets consist of LADAR scan with 160,140, 120,… degree intervals. These datasets are tested with main scan matching algorithm:

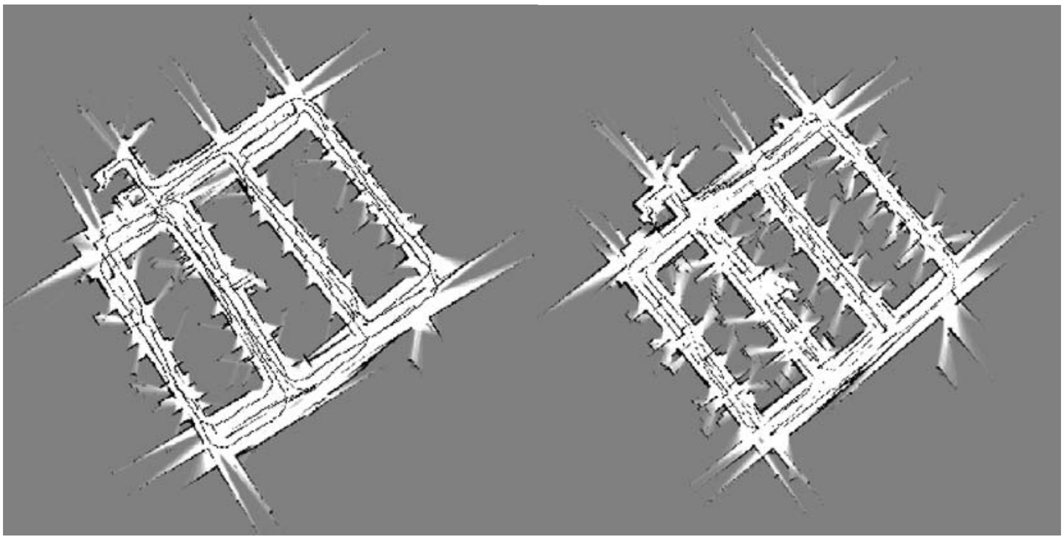


**Figure 5.47: Scan matching results with 100 (left) and 120 degree (right) intervals for LADAR scan.**

As it is seen from Figure 5.47, the performance of scan matching algorithm degrades with the decrease in number of particles. About 120 degree interval, about 1.2 meters and 0.7 degree mean localization error is seen. When it comes to 100 degree the localization becomes much worse. Because of the global localization ability of FastSLAM algorithm, a better performance in spite of the decrement in LADAR scan interval is seen. This is presented with the following results given in

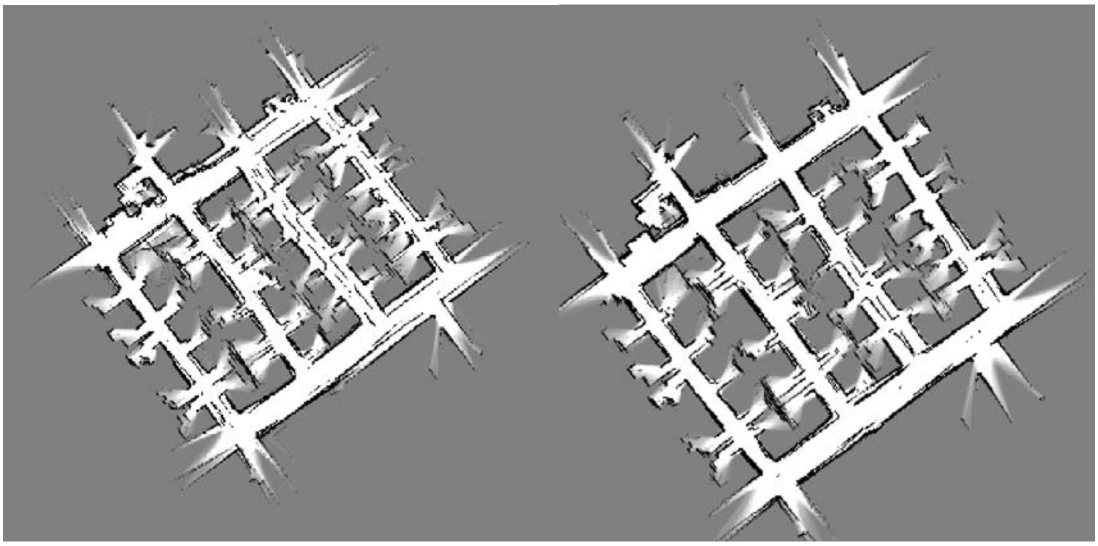


**Figure 5.48: Scan matching results with FastSLAM algorithm result 90 (left) and 100 degree (right) intervals for LADAR scan (450 particles)( The results are presented as 180 degree interval LADAR mapping, to visualize the distortions better).**

As it is seen from Figure 5.48, scan matching with FastSLAM algorithm still gives acceptable results when it comes to 90 degree LADAR scan interval. After this interval, it the algorithm also completely fails. The obtained localization performance results with respect to change in interval of LADAR scan is given in Figure 5.49 and Figure 5.50:

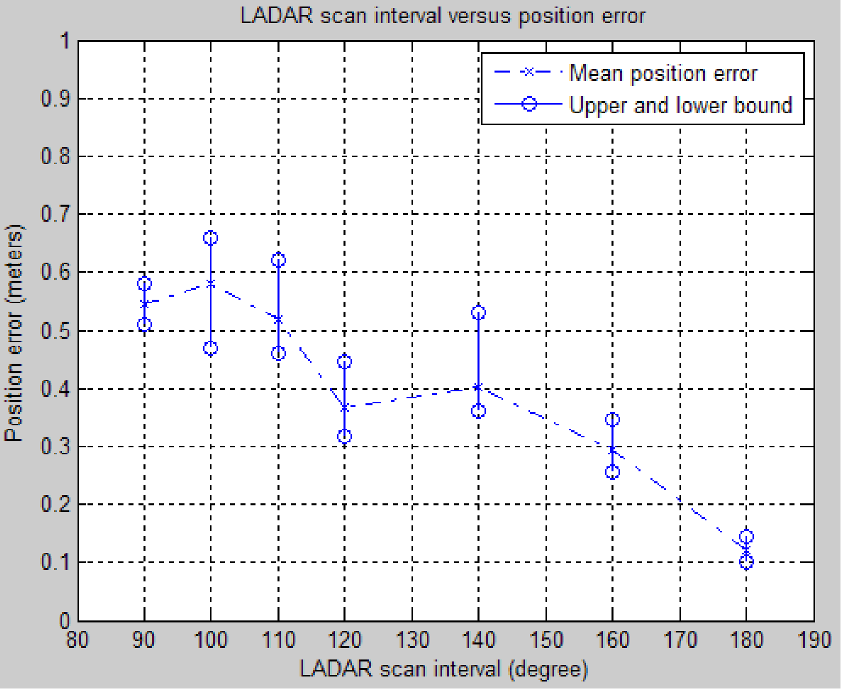


**Figure 5.49: In figure, the change in mean position error for the change in LADAR scan interval is presented (averaged from 5 trials).**

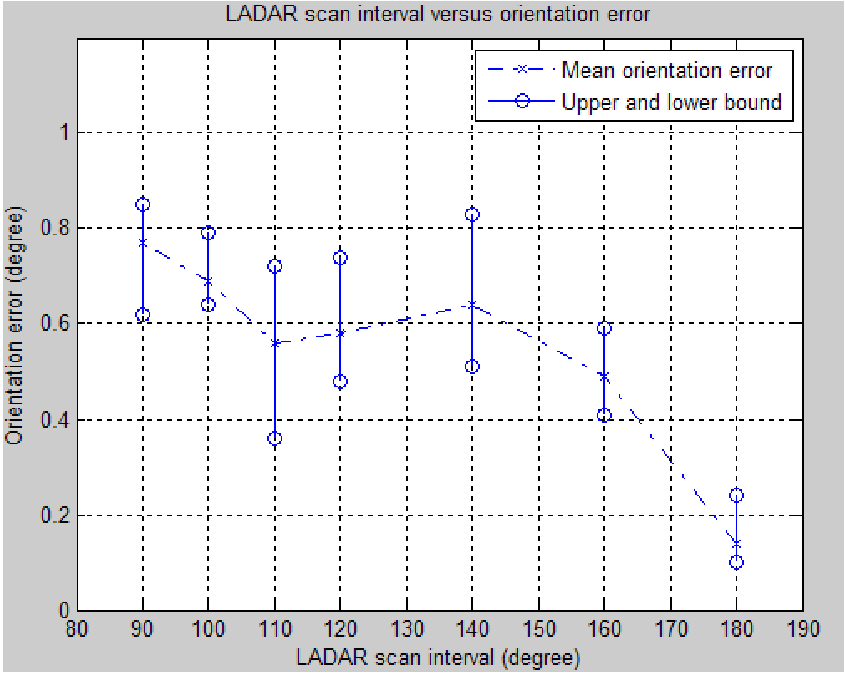


**Figure 5.50: In figure, the change in mean orientation error for the change in LADAR scan interval is presented (averaged from 5 trials).**

### 5.7.3 The Performance Comparison for All Implemented Algorithms:

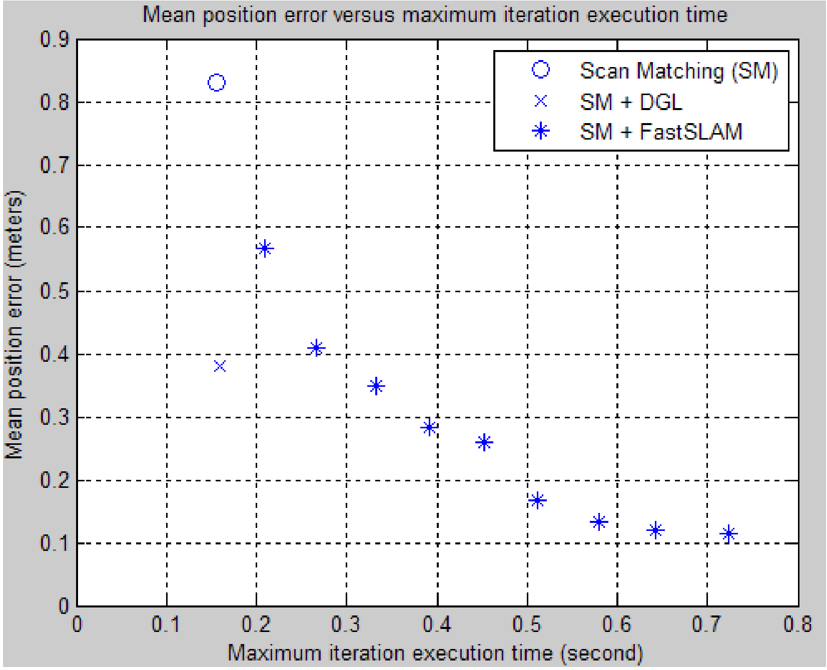


**Figure 5.51: Mean position error versus maximum iteration execution time. * corresponds SM+FastSLAM and increase 50 to 450 from left to right by 50 particles intervals.**

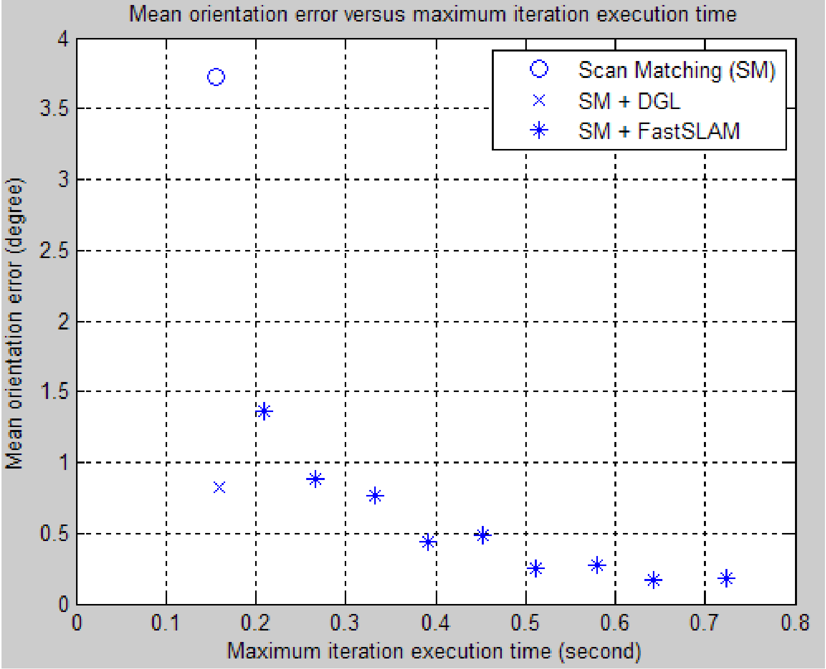


**Figure 5.52: Mean orientation error versus maximum iteration execution time. * corresponds SM+FastSLAM and increase 50 to 450 from left to right by 50 particles intervals.**

As it is seen from the Figure 5.51 and Figure 5.52, main scan matching algorithm gives worst results. When the DGL algorithm is added, the computational complexity a little bit increases, however the performance in localization and mapping increases in a considerable level. When the FastSLAM algorithm is added to the main scan matching algorithm, it catches the performance of DGL algorithm after it passes 100 particles. After number of particles pass about 300 particles, the performance in position and orientation flattens. This means that increasing number of particles after this level does not bring any contribution to the performance.

For 200 particles, we can say that FastSLAM algorithm has better performance from all others and also it has still real time working capability (assumption is, robot has rotation rate less than 60 degree per second and LADAR works in 5 Hz output rate). All the algorithms are executed as m file in MATLAB command window. When these algorithms are converted into .exe or .mex file, the computational complexity is expected to decrease half of the given in this thesis, according to the results of personal implementations and the analysis given in [38].

# CHAPTER 6

# CONCLUSION

In this thesis, a solution to indoor localization problem for autonomous robot navigation is presented. The aim is a practical localization algorithm implementation for extensive indoor environments, which is real time working, robust to the problematic effects coming with the real LADAR data and also has an acceptable success in different structured environments. During the research period, the algorithms in the literature are surveyed; the methods supporting our goals are selected. The start point of this thesis is the scan matching algorithm presented in the thesis given in [17]. During the implementation period, the algorithm is tested with a broad training dataset containing real LADAR and odometer data of different environment and scenarios. In fact, for each time, after the performance outputs obtained for different test beds, the algorithm is optimized with appended, omitted and revised methods.

Scan matching algorithm given in this thesis consists of four main parts. These are line extraction, feature and feature properties extraction, main scan matching, fault correction and global localization part. The line extraction algorithm implementation is based on the literature survey. The optimum line extraction algorithm is implemented based the correctness, precision and speed criteria according to given comparisons in papers. The implemented line extraction algorithm is optimized for potential encountered line fitting problems in real LADAR data. In feature and feature properties extraction part, conventional features and properties given in the literature are extracted. On top of them, also a pattern concept based on the multiple feature set is added. In main scan matching part, the overlap in these inputs is used for the voting procedure of matching tables constructed from the line pairs extracted from cognitive scans. Criteria in this part are carefully considered and the ones that can be adapted to real LADAR data are tried to be considered. In fact, in addition to the standard comparing techniques as criteria, simple pattern comparing techniques are used, promotion to unique features is given, and also some special cases are considered, such as the ones that adopt the algorithm to low structured environments. As a result an improvement in the performance of main scan matching part is seen. For pose calculation, novel precise methods are used. Main aims in main scan matching are to choose these criteria flexible for different environments and to implement the algorithm tolerable to mistakes in voting. In fault correction and global localization part, the scan matching outputs are compared with odometer outputs and also a pose update from odometer is given for the cases scan matching could not find. Global localization is achieved by tracking of appropriate features extracted (real corners and both side ended lines as landmark features), and making correction based on these for the closed loop cases, which is a kind of deterministic global localization implementation.

In SLAM part, the main scan matching algorithm is combined with FastSLAM algorithm instead of deterministic global localization methods. In this part, pose differences obtained from scan matching are used as improved odometer input and passed through the odometer motion model for this algorithm. Also since scan matching is feature based, landmarks of FastSLAM algorithm are taken from the feature set extracted. In data association part, as a contribution, for each landmark observation two landmark positions are added to landmark set of particles. This gives the algorithm ability to update and resample its particles also considering the orientation of landmarks in both data association and weight calculation parts. In fact, in environments with few number of landmarks are available, this property makes the particles scatter around the correct heading. Furthermore, the feature properties, dependent and independent from the pose, are applied for comparing the landmarks which have decreased the wrong association chance for the objects that stay in same likelihood.

The mapping is presented in occupancy grid map. The performance of the algorithms are compared inside by considering the corruptions on the occupancy map and compared with the maps given with Radish data [22], if available. As we can see from the experimental results for long distance traveled and also for different environments given, just by using real LADAR data and raw odometer (which is used just for the case pose difference could not be found), the scan matching algorithm still gives promising results in where the odometer data totally is corrupted. Furthermore in closed loop cases, addition of deterministic global localization method improves the pose estimation of scan matching, while FastSLAM improves the pose estimation of global localization.

In this thesis it is showed that combining the scan matching and SLAM algorithms results as improvement in pose estimation and mapping. In fact, scan matching gives better pose estimation than raw odometer output. However it suffers from the error accumulated for long distances. In other side, SLAM algorithms have pose error correction ability in closed loop cases. However, because of using raw odometer as pose input, they suffer from the data association problem, sometimes which ends with divergence of the filter. When these algorithms are combined, scan matching gives the SLAM algorithm better pose input. As a result of this, noise levels in SLAM algorithm decreases. This simplifies the data association problem and also increases the size of the loop that SLAM algorithm works without divergence. On the other hand, SLAM algorithm reduces the accumulated pose error for scan matching in closed loop cases. Moreover in full SLAM usage, it gives better estimated path with previously corrected poses and a map with higher consistency. In specific, Split and Merge Algorithm is applied in line extraction for its advance referenced to other algorithms. Feature based scan matching is used because of its better real time working capability and lower computational complexity referenced to raw LADAR data based algorithms. Also it decreases the workload of SLAM algorithm, because of joint feature set. FastSLAM algorithm is used because of its computational advantage and robustness to the data association problems referenced to EKF-SLAM methods. These algorithms are supported with improvements in accuracy and matching capability of scan matching and data association capability of SLAM algorithm.

As a future work, complex line and corner based patterns as landmarks can be used for data association in SLAM algorithms, which will increase the size of the loop, also will be a good solution for kidnapped robot problem. Also scan matching part can be improved by adding matching of curved objects, such as the ones that can be defined as circle or the ones that does not have a specific shape but arises from the continuity of data points in an order. Furthermore the given methods in scan matching can be converted into probabilistic methods. Different sensor outputs can be integrated into algorithm. In fact, according to the accuracy of the sensors used, the outputs of the given sensors can be applied such as, a bound in matching of line pairs, correction or direct pose estimate in prediction part of SLAM algorithm. On the other hand, the algorithm is applied with only LADAR and minimum level odometer usage, to see the success of algorithmic implementation.

# REFERENCES

**[1]** Cranenbroeck, J., Ashcroft, N., Awasthi, N., GNSS Positioning Infrastructure and Operations, Trimble Co, 2005.

**[2]** Hinueber, E., Introduction into Inertial Measurement Technology, iMAR GmbH, 2005.

**[3]** LADAR Information, http://en.wikipedia.org/wiki/LIDAR, August 2010.

**[4]** Lu, F., Milios, E., Globally Consistent Range Scan Alignment for Environment Mapping, Autonomous Robots, vol. 4, pp. 333–349, 1997.

**[5]** Lu, F., Milios, E., Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans, J. of Intelligent and Robotic Systems, vol. 20, pp. 249–275, 1997.

**[6]** Pfister, S., Kriechbaum, K., Roumeliotis, S., Burdick, J., Weighted Range Sensor Matching Algorithms for Mobile Robot Displacement Estimation, IEEE International Conference on Robotics and Automation (ICRA'02), pp. 667–1674, 2002.

**[7]** Minguez, J., Lamiraux, F., Montesano, L., Metric-Based Scan Matching Algorithms for Mobile Robot Displacement Estimation, IEEE International Conference on Robotics and Automation (ICRA), 2005.

**[8]** Moon, J. H., You, B. J., Kim, H., Oh, S. R., Fast Markov Localization Using Angle Histogram, 11th IEEE International Conference on Advanced Robotics (ICAR'03), pp. 411–416, 2003.

**[9]** Aghamohammadi, A. A., Taghirad, H. D., Tamjidi, A. H., Mihankhah, E., Feature-Based Range Scan Matching For Accurate and High Speed Mobile Robot Localization, ECMR, 2007.

**[10]** Latecki, L. J. , Lakaemper, R., Sun, X., Wolter, D. , Building Polygonal Maps from Laser Range Data. In Proc. Int. Cognitive Robotics Workshop, pp. 56–62, 2004.

**[11]** Yakın, İ., Line Segment Based Range Scan Matching without Pose Information for Indoor Environments, Msc Thesis, Bilkent Uni., 2008.

**[12]** Einsele., T. , Real-Time Self-Localization in Unknown Indoor Environments Using a Panorama Laser Range Finder, Intelligent Robots and Systems (IROS '97), 1997.

**[13]** Zhiyu, X., Zezhong, X., Jilin, L., Scan Matching Based on CLS Relationships, IEEE International Conference on Robotics, Intelligent Systems and Signal Processing,Vol. 1, pp. 99– 104, 2003.

**[14]** Lingemann, K., Surmann, H., Nuchter, A., Hertzberg, J., High-Speed Laser Localization for Mobile Robots, Journal of Robotics and Autonomous Systems, vol. 51, no. 4, pp. 275–296, 2005

**[15]** Weiss, G., Puttkamer, E., A Map Based on Laserscans without Geometric Interpretation, Intelligent Autonomous Systems, pp. 403–407, IOS Press, 1995.

**[16]** Weber, J., Jörg, K. W., Puttkamer, E., APR - Global Scan Matching Using Anchor Point Relationships, 6th International Conference on Intelligent Autonomous Systems (IAS-6), 2000

**[17]** Nguyen, V., Martinelli, A., Tomatis, N., Siegwart, R., A Comparison of Line Extraction Algorithms using 2D Laser Rangefinder for Indoor Mobile Robotics, Intenational Conference on Intelligent Robots and Systems (IROS'05), 2005.

**[18]** K. O. Arras and R. Siegwart. Feature Extraction and Scene Interpretation for Map-Based Navigation and Map Building, Symposium on Intelligent Systems and Advanced Manufacturing, 1997.

**[19]** Fischler, M., Bolles, R., Random Sample Consensus: A Paradigm for Model Fitting with Application to Image Analysis and Automated Cartography, Communications of the ACM, vol 24, no 6, pp. 381–395, 1981.

**[20]** Pfister, S. T. , Roumeliotis, S. I., Burdick, J. W., Weighted Line Fitting Algorithms for Mobile Robot Map Building and Efficient Data Representation, IEEE International Conference on Robotics and Automation, 2003.

**[21]** Forsyth, D. A., Ponce, J. , Computer Vision: A Modern Approach, Prentice Hall, 2003.

**[22]** RADISH Data Repository, www.openslam.org, August 2010, 17th.

**[23]** Bresenham Line Alg., http://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm, August 2010, 6th.

**[24]** Ayache, N., Faugeras, O.D., Maintaining a Representation of the Environment of a Mobile Robot. IEEE Transaction on Robotics and Automation, vol 5, pp. 804-819, 1989.

**[25]** Smith, R., Self, M., Cheeseman, P., Estimating Uncertain Spatial Relationships in Robotics, Autonomous Robot Vehicles, no 1, pp 167:193, 1990.

**[26]** Leonard, J. J., Durrant-Whyte, H. F. , Simultaneous Map Building and Localization for an Autonomous Mobile Robot, IEEE Conference on Intelligent Robots and Systems, pp 1442-1447, 1991.

**[27]** Bailey, T. Mobile Robot Localization and Mapping in Extensive Outdoor Environments, Phd thesis, University of Sydney, Australian Centre for Field Robotics, 2002.

**[28]** Williams, S., Local Map Representation: Analysis of the Relative Landmarks Cross-Covariance Factors, ACFR, 2000.

**[29]** Leonard, J.J., Feder, H.J.S. A Computational Efficient Method for Large-Scale Concurrent Mapping and Localization, Robotics Research, The 9th International Symposium (ISRR'99), pp 169-176., 2000.

**[30]** Tomatis, N., Nourbakhsh, I., Siegwart, R., Hybrid Simultaneous Localization and Map Building: Closing the loop with multi-hypotheses tracking. International Conference on Robotics and Automation, 2002

**[31]** Thrun, S., Burgard, W., Fox, D., A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots, Machine Learning, vol.31, pp.29-53, 1998.

**[32]** Fox, D., Thrun, S., Dellaert, F., Burgard, W., Particle Filters for Mobile Robot Localization, In Sequential Monte Carlo Methods in Practicle, 2000.

**[33]** Fox, D., Thrun, S., Dellaert, F., Burgard, W., Robust Monte Carlo Localization for Mobile Robots, Artificial Intelligence, vol.128, pp.99-141, 2001.

**[34]** Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B., FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem, Artificial Intelligence, 2002.

**[35]** Hahnel, D., Burgard, W., Fox, D., Thrun, S., An Efficient FastSLAM Algorithm for Generating Maps of Large-Scale Cyclic Environments from Raw Laser Range Measurements, IEEE Conferance Intelligent Robots and Systems, pp. 206–211, 2003.

**[36]** Pradalier, C., Sekhavat, S., Concurrent Matching, Localization and Map Building Using Invariant Features, IEEE Conference Intelligent Robots and Systems, pp. 514– 520, 2002.

**[37]** Riisgard, S., Blas, M.R., SLAM for Dummies, A Tutorial Approach to Simultaneous Localization and Mapping.

**[38]** Converting M-Files to Stand-Alone Applications, http://technologyinterface.nmsu.edu/5_1/5_1f/5_1f.html, September 2010, 22th.

**[39]** Tungadi, F., Kleeman, L., Multiple Laser Polar Scan Matching with Application to SLAM, in Australasian Conference on Robotics and Automation, 2007.

**[40]** Le´on, A., Barea, R., Bergasa, L.M., L´opez, E., Oca˜na, M., Schleicher, D., SLAM and Map Merging, Journal of Physical Agents, vol. 3, pp 13-23, 2009,

**[41]** Tungadi, F., Lui, W.L:D., Kleeman, L., Jarvis, R., Robust Online Map Merging System using Laser Scan Matching and Omnidirectional Vision, IROS 2010.

**[42]** Tungadi, F., Kleeman, L., Loop Exploration for SLAM with Fusion of Advanced Sonar Features and Laser Polar Scan Matching, IROS 2009

**[43]** Thrun, S., Burgard, W., Fox, D., A Real-time Algorithm for Mobile Robot Mapping with Applications to Multi-robot and 3D Mapping", ICRA 2000, 2000.

**[44]** Williams, S. B., Dissanayake, G., Durrant-Whyte, H., Towards Multi-vehicle Simultaneous Localisation and Mapping", ICRA 2002.

**[45]** Thrun, S., Burgard, W., Fox, D., Probabilistic Robotics, Cambridge, MA: MIT Press, 2005

**[46]** Sim, R., Roy, N., Global a-optimal Robot Exploration in SLAM, ICRA, 2005.

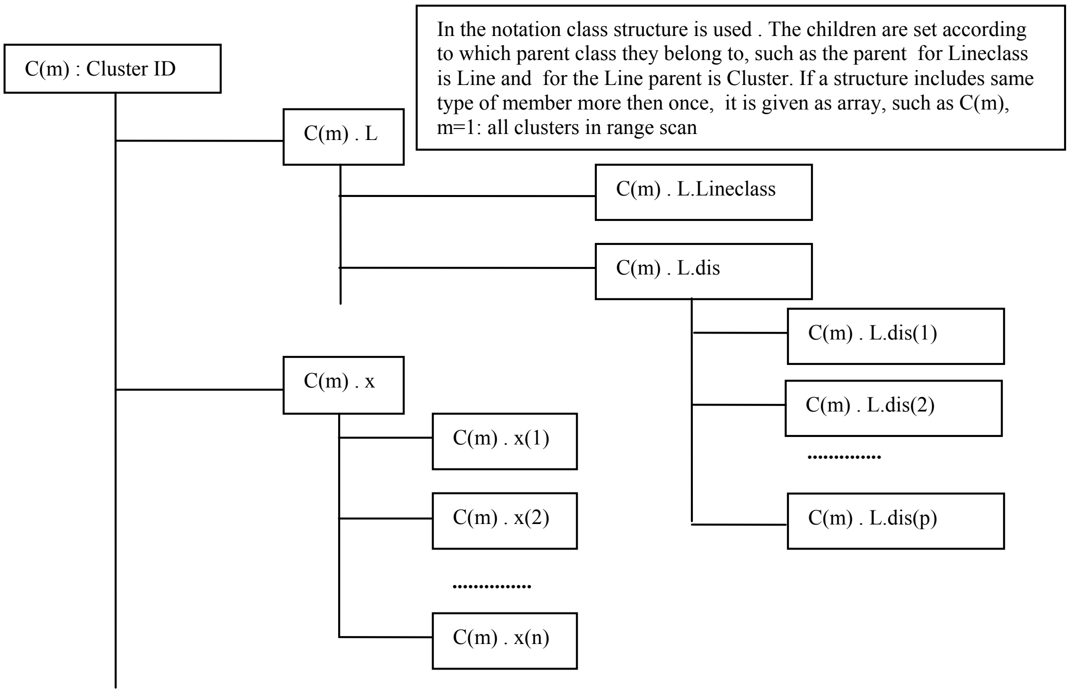
**[47]** Kummerle,R., Triebel, R.,Pfaff, P., Burgard, W., Active Monte Carlo Localization in Outdoor Terrains Using Multi-level Surface Maps, Field and Service Robotics, 2007.

**[48]** Stachniss, C., Exploration and Mapping with Mobile Robots. PhD thesis, University of Freiburg, 2006.

# APPENDIX A

# ADDITIONAL INFORMATION

## A.1 Data Structures Used to Implement Algorithms:



In the notation class structure is used . The children are set according to which parent class they belong to, such as the parent for Lineclass is Line and for the Line parent is Cluster. If a structure includes same type of member more then once, it is given as array, such as C(m), m=1: all clusters in range scan

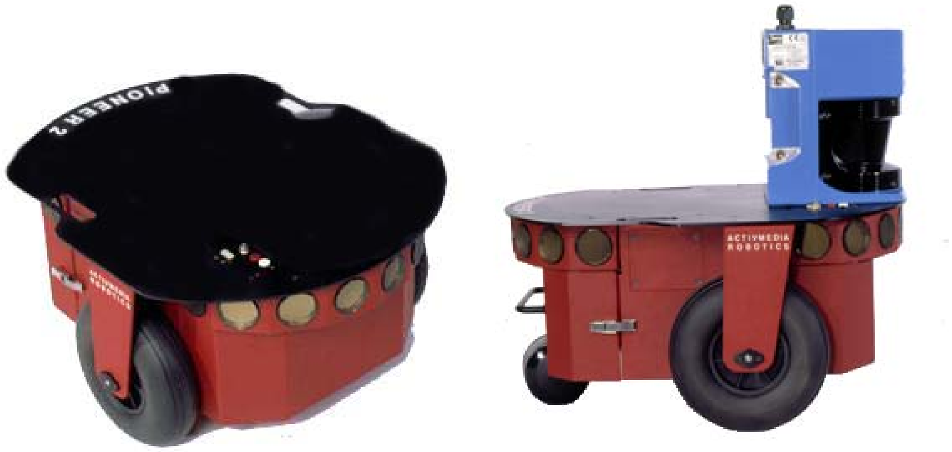
**Figure A.1: The figure illustrating the class structure defined for scan matching**

The class structure created for all variables given in this thesis is:

Class structure is used for the notational definitions that are given in list of symbols. An illustration is given for this class structure in Figure A.1. In the definitions the main set the property is belong to is given first, while the definition goes left, the structure is specialized for the given property. If we examine an example structure $\mathrm{C}^{[\mathrm{m}]}.L.LineClass$ :

$\mathrm{C}^{[\mathrm{m}]}$ corresponds to the *m*th ID cluster created from raw LADAR range scan. $\mathrm{C}^{[\mathrm{m}]}.L$ corresponds to the fitting line to the *m*th ID cluster created from raw LADAR range scan. $\mathrm{C}^{[\mathrm{m}]}.L.LineClass$ corresponds to the class of the line fitting (such as both, left, right side ended or open) to the *m*th ID cluster created from raw LADAR range scan.

## A.2 Testbed Platforms:



**FigureA.2: Pioneer 2DX differential drive testbed robot with SICK LMS 200 Laserscanner**

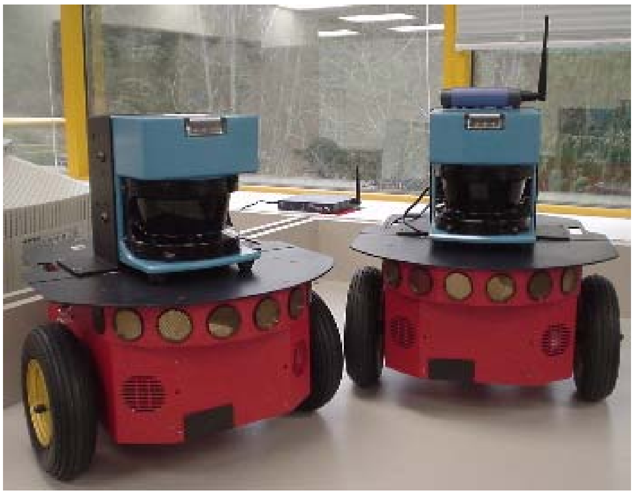


**Figure A.3: Pioneer 3DX differential drive testbed robot with SICK LMS 200 Laserscanner**

In FigureA.2 and Figure A.3, the testbed robot platforms that raw sensor data is acquired are presented. These platforms are commercial robots dedicated for robotic applications. They are three wheeled (one is dummy wheel) differential drive platforms. They provide directly estimated odometer output as $[x, y, \theta]$ referenced to the coordinate frame at their startpoint, which is assumed as [0, 0, 0].