

**IMPROVED PERFORMANCE FOR SLAM
TECHNIQUES USING TRAP
CONFIGURED 2D LRFs**



M.Sc. THESIS

Osman ERVAN

Department of Control and Automation Engineering

Control and Automation Engineering Programme

DECEMBER 2016

**IMPROVED PERFORMANCE FOR SLAM
TECHNIQUES USING TRAP
CONFIGURED 2D LRFs**

M.Sc. THESIS

**Osman ERVAN
(504141125)**

Department of Control and Automation Engineering

Control and Automation Engineering Programme

Thesis Advisor: Prof. Dr. Hakan TEMELTAŞ

DECEMBER 2016

**TRAP KONFIGÜRASYONLU LAZER
SENSÖR İLE EZKH
TEKNİKLERİNİN İYİLEŞTİRİLMESİ**

YÜKSEK LİSANS TEZİ

**Osman ERVAN
(504141125)**

Kontrol ve Otomasyon Mühendisliği Anabilim Dalı

Kontrol ve Otomasyon Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Hakan TEMELTAŞ

ARALIK 2016

Osman ERVAN, a M.Sc. student of ITU Graduate School of Science Engineering and Technology 504141125 successfully defended the thesis entitled “IMPROVED PERFORMANCE FOR SLAM TECHNIQUES USING TRAP CONFIGURED 2D LRFs”, which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor : **Prof. Dr. Hakan TEMELTAŞ**
Istanbul Technical University

Jury Members : **Assist. Prof. Ali Fuat ERGENÇ**
Istanbul Technical University

Assist. Prof. Janset DAŞDEMİR
Yıldız Technical University

Date of Submission : **25 November 2016**

Date of Defense : **22 December 2016**





Don't dream your life, live your dreams...



FOREWORD

I would like to thank my family for their motivation and endless supports. For his inspiration, guidance and helps I would like to indicate my appreciation to my advisor Prof. Dr. Hakan TEMELTAŞ. For sharing his time, teaching me countless things and most importantly for his great friendship I also would like to thank to Onur ŞENCAN. I also want to indicate my appreciation to Assist. Prof. Tufan KUMBASAR for his motivation and his friendship. Additionally, I would like to express my gratitude for all my professors, lecturers, teachers from whole educational life. Finally, this research was partially granted under ITU Library Robotic System Project with the project number 38467 by ITU Coordinatorship of Scientific Research Projects.

December 2016

Osman ERVAN

TABLE OF CONTENTS

	<u>Page</u>
FOREWORD.....	ix
TABLE OF CONTENTS.....	xi
ABBREVIATIONS	xiii
SYMBOLS.....	xv
LIST OF TABLES	xvii
LIST OF FIGURES	xix
SUMMARY	xxi
ÖZET	xxv
1. INTRODUCTION	1
1.1 Motivation.....	1
1.2 Objectives	2
1.3 Contribution.....	2
1.4 Outline	2
2. RELATED WORKS AND LITERATURE SURVEY	5
3. THE UNCERTAINTY PROBLEM IN ROBOTICS	7
3.1 Estimation Under Uncertainty.....	7
3.1.1 Estimation with parametric filters	8
3.1.1.1 Kalman filters.....	9
3.1.1.2 Extended kalman filters	11
3.1.1.3 Unscented kalman filters.....	13
3.1.2 Estimation with non-parametric filters	17
3.1.2.1 Particle filter.....	17
3.2 Modeling Under Uncertainty.....	18
3.2.1 Robot motion models	19
3.2.1.1 Velocity motion model.....	20
4. ROBOT OPERATING SYSTEM (ROS).....	25
4.1 Basics of ROS.....	26
4.1.1 General definitions.....	27
4.1.1.1 ROS nodes	29
4.1.1.2 ROS topics	29
4.1.1.3 ROS messages.....	30
4.1.1.4 ROS services.....	30
4.1.1.5 ROS packages	31
4.1.2 Robots using ROS	31
4.1.3 Installation of ROS step by step	32
4.2 A Short Tutorial of ROS	35
4.2.1 For users who want to use ROS.....	35

4.2.1.1 Users unfamiliar to ubuntu and ROS	35
4.2.1.2 Users familiar to ubuntu but unfamiliar to ROS	36
4.2.1.3 Users familiar to ubuntu and ROS	36
4.2.2 For users who want to use something familiar instead of ROS	37
4.3 ROS Simulators	37
4.3.1 Gazebo	38
4.3.2 Rviz.....	40
4.4 A Real Time Implementation in ROS.....	42
5. LOCALIZATION AND MAPPING PROBLEM	47
5.1 Localization Problem.....	47
5.2 Mapping Problem	50
5.2.1 Grid map.....	50
5.2.2 Feature map	51
6. SLAM PROBLEM AND IMPLEMENTATION	53
6.1 SLAM Under Uncertainty	53
6.2 SLAM Techniques Available in ROS	55
6.2.1 Gmapping	55
6.2.2 Hector SLAM.....	56
7. EXPERIMENTAL WORKS.....	57
7.1 Experimental Setup	57
7.2 TRAP Configuration.....	60
7.3 Experiments	63
7.3.1 Comparison of the SLAM techniques for traditional configuration.....	65
7.3.1.1 Gmapping	65
7.3.1.2 Hector SLAM	66
7.3.1.3 Discussion.....	67
7.3.2 Comparison of the SLAM techniques for TRAP configuration.....	68
7.3.2.1 Gmapping	68
7.3.2.2 Hector SLAM	69
7.3.2.3 Discussion.....	70
7.3.3 Comparison of traditional and TRAP configuration	71
7.3.3.1 Gmapping	71
7.3.3.2 Hector SLAM	74
7.3.3.3 Discussion.....	75
8. CONCLUSIONS.....	77
REFERENCES.....	79
APPENDICES	83
APPENDIX A.1	84
APPENDIX A.2	86
CURRICULUM VITAE.....	89

ABBREVIATIONS

AGV	: Autonomous Guided Vehicle
AUV	: Autonomous Underwater Vehicle
CAD	: Computer Aided Design
CPU	: Central Processing Unit
EKF	: Extended Kalman Filter
GUI	: Graphical User Interface
IMU	: Inertia Measurement Unit
ITU	: Istanbul Technical University
LRF	: Laser Range Finder
LTS	: Long Term Service
NTP	: Network Time Protocol
OSRF	: Open Source Robotic Foundation
PR	: Personal Robot
RBPF	: Rao-Blackwellized Particle Filter
ROS	: Robot Operating System
SLAM	: Simultaneous Localization and Mapping
SAIL	: Stanford Artificial Intelligent Laboratory
STAIR	: Stanford Artificial Intelligent Robot
SDRAM	: Synchronous Dynamic Random Access Memory
TRAP	: Transversal Positioned
UAV	: Unmanned Aerial Vehicle
UKF	: Unscented Kalman Filter
USV	: Unmanned Surface Vessel



SYMBOLS

μ	: Mean
Σ	: Covariance
$\hat{\mu}_t$	: Predicted Mean
$\hat{\Sigma}_t$	: Predicted Covariance
t	: Time
X_t	: State
A_t	: State Matrix
B_t	: Input Matrix
C_t	: Output Matrix
Z_t	: Measurement
ε_t	: Error of the System
R_t	: Covariance of the Error of the System
δ_t	: Noise in the Measurement
K_t	: Kalman Gain
I	: Identity Matrix
SP	: Sigma Point
W_m	: Weight Defined for Every Sigma Point Related With Mean
W_c	: Weight Defined for Every Sigma Point Related With Covariance
U_t	: Control Input
$p(x)$	: Prior
θ	: Heading Angle
v_t	: Translational Velocity
ω_t	: Angular Velocity
r	: Radius



LIST OF TABLES

	<u>Page</u>
Table 3.1 : Similarities in kalman and extended kalman filter equations.....	12
Table 4.1 : Version history for ROS releases.....	33
Table 7.1 : Hardware equipments of the experimental setup.	58
Table 7.2 : Performance criterion of gmapping for traditional configurations.....	65
Table 7.3 : CPU load criterion of gmapping for traditional configurations	65
Table 7.4 : Algorithm runtime criterion of gmapping for traditional configurations.....	66
Table 7.5 : Performance criterion of hector SLAM for traditional configurations	66
Table 7.6 : CPU load criterion of hector SLAM for traditional configurations	66
Table 7.7 : Algorithm runtime criterion of hector SLAM for traditional configurations.....	67
Table 7.8 : Performance criterion of gmapping for TRAP configurations.....	68
Table 7.9 : CPU load criterion of gmapping for TRAP configurations.....	69
Table 7.10 : Algorithm runtime criterion of gmapping for TRAP configurations ..	69
Table 7.11 : Performance criterion of hector SLAM for TRAP configurations.....	69
Table 7.12 : CPU load criterion of hector SLAM for TRAP configurations	70
Table 7.13 : Algorithm runtime criterion of hector SLAM for TRAP configurations.....	70
Table 7.14 : Performance criterion of gmapping for traditional and TRAP configurations.....	71
Table 7.15 : CPU load criterion of gmapping for traditional and TRAP configurations.....	72
Table 7.16 : Algorithm runtime criterion of gmapping for traditional and TRAP configurations.....	73
Table 7.17 : Performance criterion of hector SLAM for traditional and TRAP configurations.....	74
Table 7.18 : CPU load criterion of gmapping for traditional and TRAP configurations.....	74
Table 7.19 : Algorithm runtime criterion of gmapping for traditional and TRAP configurations.....	75



LIST OF FIGURES

	<u>Page</u>
Figure 3.1 : Gauss probability distribution functions for different parameters.....	8
Figure 3.2 : The representation of a particle filter.....	17
Figure 4.1 : ROS node-topic-service-message relationship	27
Figure 4.2 : The ROS architecture	28
Figure 4.3 : The gazebo GUI start-up screen	39
Figure 4.4 : The rviz GUI start-up screen	41
Figure 4.5 : The USB hub	44
Figure 4.6 : The output of the view_robot.launch file.....	45
Figure 4.7 : The output after navigation package have been ticked.....	46
Figure 5.1 : Graphical representation of localization.....	48
Figure 5.2 : Comparison of localization in different aspects	48
Figure 6.1 : Representation of SLAM.....	54
Figure 7.1 : Experimental setup	58
Figure 7.2 : The CAD and technical drawing of Husky A200 [51] and [52].....	59
Figure 7.3 : Hokuyo UTM-30LX with technical drawing [53].	60
Figure 7.4 : Traditional 2D LRF configuration for 180° and 270°.	61
Figure 7.5 : TRAP configuration for 2D LRFs	62
Figure 7.6 : The experimental environment in real world and simulation.....	64
Figure 7.7 : Generated maps by gmapping and hector SLAM for trad. conf.	67
Figure 7.8 : Generated maps by gmapping and hector SLAM for TRAP conf. ...	71
Figure 7.9 : CPU load criterion of gmapping for traditional and TRAP conf.	73



IMPROVED PERFORMANCE FOR SLAM TECHNIQUES USING TRAP CONFIGURED 2D LRFs

SUMMARY

The issue covered in this thesis can be evaluated in two stages. In the first part of this work, some of the 2D laser range finder (LRF) based Simultaneous Localization and Mapping (SLAM) techniques available in Robot Operating System (ROS) have been discussed and compared with each other. These methods, which distinct differences from each other, are compared according to their advantages and disadvantages. These comparisons have been realized first in simulation environments, after then these comparisons have been done in real time experimental environment. There are two major reasons for working in a simulated environment. Among these, the battery consumption and charging problem which is the energy problem is the first reason. In real-time applications on the system, the battery drained within approximately one hour and recharged within five to six hours. This situation caused to a slowdown in the studies. It took at least six hours to wait for about an hour of work. In order to remove this problem, working in simulation environment has been preferred to solve the constraint in time. Another reason for the preference of the simulation environment is to avoid from possible damages of the equipments in any fault in the codes. In the simulation, crashing against the walls or obstacles do not damage the real system.

The second objection is a hardware innovation which was proposed and the effects of this innovation were discussed. The improvement offered is a structure related with 2D LRF position configuration. In a traditionally configuration, a single 2D laser sensor is positioned at the front of the vehicle centered. In this existing structure, the mobile robot does not scan every point around itself totally, and blind spots have been occurred around the robot. However, in the proposed new 2D LRF position configuration, it is possible to perceive every point around 360° without any blind spot. This structure have been named as Transversal Positioned and is represented by TRAP. Two 2D laser sensors have been applied in the TRAP configuration and the sensors are placed at angles of 45° on the diagonal line of the mobile robot and the backs of the LRFs are facing each other. In this way, it was provided to detect everything around the vehicle without a blind spot. In the same way, the traditional structure and the TRAP structure are compared in terms of the predetermined criteria, additionally their advantages and disadvantages have been examined.

ROS has been used in both real time environment and simulated environment. There are certain reasons why ROS has been chosen. One of the main reason for this selection is being an open source and freeware. Another factor is that being used by many researchers who work in academic, industrial and personal research area of robotics. Another reason for the preference of the ROS platform is the ease in which the simulation work is delivered in real time. Under normal circumstances it can be quite challenging to introduce a new hardware into the system and start using it. Considering the advantages provided by ROS, it is much easier to attach a new

equipment to a platform and start to use the data from the attached equipment. It is not necessary to write code snippets separately for each piece of hardware. The existing code snippets are shared with all users because the platform is open source. The users need only change the parameters in the code fragment in the direction of the request and quickly start to introduce the hardware to the system and start to receive data. This simplicity allows the users to spend their time on code algorithm development rather than hardware attachment. One of the biggest reasons why ROS is preferred is that the codes developed for the simulation environment can be applied directly to the real-time system with minor modifications.

The main focus of the thesis is the SLAM problem, which is often sought in the robotics literature. In order to be able to grasp this problem correctly, firstly, uncertainties in robotics were explained. These disclosures include filter structures used in this area, modeling mobile robot movements and sensor modeling. After the uncertainties in the field of robotics were expressed, explanations were given on localization and mapping issues. Explanations were made on what the localization and mapping problems are in the robotic research area, and what approaches are being sought. In order to understand the SLAM problem, it is necessary to understand the localization and mapping problems. The reason of that is because in SLAM these two problems have been followed simultaneously by each other. In the following, the definition of the SLAM problem is defined and information about what it is also given. ROS-compatible 2D laser sensor based SLAM methods were studied and investigated.

While experimenting with SLAM techniques available in ROS three environment scenarios have been created to be compatible with the studies. These scenarios vary depending on the complexity of the environment according to the number and type of obstacles. Depending on the complexity of the environment, it is possible to name these scenarios as easy, medium and difficult, respectively. These scenarios with different difficulty levels have been proposed in order to make the comparison of SLAM methods available in ROS as fair and detailed as possible. It has been analyzed how behaviors changed in different environments. The first scenario was designed as an empty corridor consisting of only walls without any obstacles in the environment. This scenario can also be called a simple scenario. In the second scenario some static obstacles were added and the environment was tried to be more complicated. This scenario can be called the average scenario. Finally, more stable obstacles have been added than the previous one, and with this, not only are the obstacles with moving motions added. The final scenario can also be called a difficult scenario. In order to be able to perform a fair evaluation, the scenarios are fixed for each different SLAM structure, ie no changes are applied and the situation is adapted to the moving obstacles. It has also been noted that in all cases the vehicle starts at the same starting point and determines the same ending point as the target itself. Experimental studies have been carried out in accordance with these criteria.

A similar comparison was made about the 2D laser sensors placed in the TRAP structure with the conventional 2D laser sensor construction. In this comparison, the same scenarios as the previous scenarios were used, and the starting points and the ending points were considered to be identical.

In the third scenario, including the dynamical obstacles, the difference between the TRAP and the traditional structure was clearly visible. In a conventional 2D laser sensor structure, when a moving obstacle has been detected in the environment, the

running algorithm can not directly understand whether it is a dynamical obstacle or not. Although in the algorithm in which the sensor has perceived an obstacle, this perceived structure may also be the result of a slip on the map. In situations like this the mobile robot needs additional data in order to understand what is perceived to be an obstacle or a slip on the map. In order to collect these additional data, the vehicle must rotate several times around its axis and the necessary data will be obtained as a result of these rotations. If the mobile robot encounters a moving obstacle again after it starts to move, it needs to rotate around its own axis again and get additional data. This requirement also extends the completion of the task. When the same experiment is repeated with the equipment for which the TRAP structure has been applied, the need to rotate the vehicle around its axis has been removed or reduced as far as possible. This has greatly reduced the completion time of the task. In addition, the faster completion of the algorithm also reduces the amount of battery used, thus providing a longer usage time.





TRAP KONFIGÜRASYONLU LAZER SENSÖR İLE EZKH TEKNİKLERİNİN İYİLEŞTİRİLMESİ

ÖZET

Bu tezde ele alınan konu iki aşamada değerlendirilebilir. Yapılan bu çalışmanın ilk aşamasında Robot İşletim Sistemi (Robot Operating System – ROS) uyumlu Eş Zamanlı Konumlama ve Haritalama (Simultaneous Localization and Mapping – SLAM) 2B lazer sensör tabanlı yöntemlerinden bazıları ele alınıp tanıtılarak birbirleriyle karşılaştırılmışlardır. Bu karşılaştırma esnasında birbirinden farklı olan bu yöntemler birbirlerine göre avantaj ve dezavantajlara göre kıyaslanmıştır. Yapılan bu kıyaslamalar öncelikle benzetim ortamında ardından da hazırlanan deney düzeneği üzerinde gerçek zamanlı olarak gerçekleştirilmiştir. Çalışmaların iki aşamalı olarak öncelikle benzetim ortamında gerçekleştirilmiş olması yapılan çalışmaları her ne kadar yavaşlatmış gibi gözükse de aslında sanılanın aksine çalışmaların daha hızlı bir şekilde ilerlemesine olanak sağlamıştır. Aslında benzetim ortamında çalışılmasının iki büyük nedeni vardır. Bunlardan ilki pil tüketimi ve şarj sorunu yani kısaca enerji sorunu. Sistem üzerinde yapılan gerçek zamanlı çalışmalarda yaklaşık olarak bir saat içerisinde boşalan pil beş – altı saat içerisinde tekrar dolmaktaydı. Bu durum çalışmaların hızlı bir şekilde ilerlemesine engel teşkil etmekteydi. Yaklaşık olarak bir saatlik bir çalışmanın sonunda en az altı saat beklemek gerekmekteydi. Bu kusuru ortadan kaldırmak adına, süre olarak herhangi bir kısıtlamaya neden olmayan benzetim ortamında çalışılması çözümü üretildi. Benzetim ortamının tercih edilmesinin bir diğer nedeni de kodlamada yapılan herhangi bir yanlışlıkta ekipmanların zarar görmesini ortadan kaldırmaktır. Simülasyonda duvara ve engellere çarpan robotun koştığı algoritma değiştirilerek gerçek sistemin aynı hataları yapıp hasar alması bu sayede önlenmiş oldu.

İkinci aşamada ise donanımsal olarak bir yenilik önerildi ve bu yeniliğin etkileri üzerine tartışıldı. Önerilen yenilik 2B lazer sensörleriyle alakalı bir yapıydı. Geleneksel yapıda tek bir tane 2B lazer sensör aracın ön tarafında ortalanmış bir şekilde konumlanmakta. Mevcut olan bu yapıda araç etrafındaki her noktayı kusursuz bir şekilde görememekte yani aracın etrafında kör noktalar oluşmakta. Oysa ki önerilen yeni 2B lazer sensör yerleşim konfigürasyonunda aracın 360° olarak etrafında olan biten her şeyi kör nokta olmadan algılaması mümkün kılınmıştır. Transversal Positioned (çapraz konumlanmış) olarak adlandırılan yapı TRAP olarak gösterilmektedir. TRAP yapısında iki adet 2B lazer sensör kullanılmakta ve sensörler aracın köşelerine sırtları birbirlerine dönük olacak şekilde köşegen üzerine 45°'lik bir açıyla yerleştirilmiştir. Bu sayede aracın etrafındaki bütün her şeyi kör nokta olmadan görmesi sağlanmıştır. Aynı şekilde geleneksel yapı ile TRAP yapısı belirlenen kriterler doğrultusunda kıyaslanmış, birbirlerine göre avantaj ve dezavantajları incelenmiştir.

Hem benzetim ortamında yapılan deneysel çalışmalarda hem de gerçek zamanlı ortamda yapılan deneysel çalışmalarda ROS kullanılmıştır. Bu tercihin yapılmasının en büyük nedenlerinden birisi yapının açık kaynak kodlu olması ve kullanımının tamamen ücretsiz olmasıdır. Yapılan bu tercihteki bir diğer etmen robotik alanında

akademik, endüstriyel ve kişisel olarak çalışma yapan araştırmacıların bu platformu kullanması popülerliğini arttırmakta ve tercih edilmesine neden olmaktadır. Ayrıca ROS platformunun tercih edilmesinin bir diğer nedeni de kullanımda sağladığı kolaylıklar ve benzetim çalışmalarının gerçek zamanlı ortama aktarılmasında sağlamış olduğu rahatlıklardır. Normal şartlarda yeni bir donanımın sisteme tanıtılıp kullanılmaya başlanması oldukça uğraştırıcı olabilir. Tanıtma işlemi sonrasında verilerin alınıp işlenmesi de bir o kadar zorlayıcı olabilir. Oysaki ROS'un sağladığı avantajlar göz önüne alındığında bu platformda yeni bir donanımın eklenmesi ve bu donanımdan veri alınıp kullanılmaya başlanması çok daha kolaydır. Her bir donanım için tekrardan ayrı ayrı kod parçacıklarını yazmak gerekmemektedir. Mevcut kod parçacıkları platformun açık kaynak kodlu olması sebebiyle bütün kullanıcılarla paylaşılmaktadır. Kullanıcı sadece kod parçacığında isteği doğrultusunda parametreleri değiştirerek hızlı bir şekilde donanımı sisteme tanıtıp veri almaya başlamaktadır. Bu da ROS platformunda donanım eklemeye harcanacak zamanı kod ve algoritma geliştirmeye aktarmaya olanak sağlamaktadır. Yazılımsal olarak benzetim ortamı için geliştirilen kodların küçük değişikliklerle doğrudan gerçek zamanlı sisteme uygulanabilir olması da ROS'un tercih edilmesi için en büyük nedenlerden biridir.

Tezde odaklanılan ana konu robotik literatüründe de sıklıkla çözümler aranılan SLAM problemidir. Bu problemin doğru bir şekilde kavranabilmesi için öncelikle robotikte belirsizlik konuları hakkında açıklamalar yapıldı. Bu açıklamalar, bu alanda kullanılan filtre yapılarını, mobil robot hareketlerinin modellenmesini ve sensör modellenmesini kapsamaktadır. Robotik alanındaki belirsizlikler ifade edildikten sonra konumlama ve haritalama konuları üzerine açıklamalar yapıldı. Robotik çalışmalarında geniş bir yer tutan konumlama ve haritalama problemlerinin ne olduğu ve hangi yaklaşımlarla çözümler arandığı üzerine açıklamalar yapıldı. SLAM probleminin anlaşılabilmesi için bahsi geçen konumlama ve haritalama problemlerinin anlaşılması gerekmektedir zira SLAM bu iki problemin eş zamanlı olarak birbirlerini takip etmesiyle gerçekleşmektedir. Devamında SLAM probleminin tanımı yapılarak ne olduğu hakkında bilgiler verildi. ROS uyumlu 2B lazer sensör tabanlı SLAM yöntemleri hakkında bilgi verilip inceleme yapıldı.

ROS uyumlu SLAM yöntemleriyle deneyse çalışmalar yapılırken çalışmalarla uyumlu olacak biçimde üç adet ortam senaryosu oluşturulmuştur. Bu senaryolar ortamın karmaşıklığına, engel yapıları ve sayılarına bağlı olarak farklılık göstermektedir. Ortamın karmaşıklığına bağlı olarak bu senaryoların sırasıyla kolay, orta ve zor olarak adlandırılması mümkündür. ROS uyumlu SLAM yöntemlerinin kıyaslamasının olabildiğince adilane ve ayrıntılı yapılabilmesi adına bu farklı zorluklardaki senaryolar önerilmiştir. Bu sayede farklı ortamlarda nasıl bir davranışın sergilendiği görülmüş oldu. Bu senaryolardan ilki içerisinde hiçbir engel bulunmayan sadece duvarlardan oluşan boş bir koridor olarak tasarlandı. Bu senaryo kolay senaryo olarak da adlandırılabilir. İkinci senaryoda ortama bazı durağan engeller eklenmiştir ve ortam biraz daha karmaşılaştırılmaya çalışılmıştır. Bu senaryo orta senaryo olarak adlandırılabilir. Son olarak bir önceki ortama göre daha fazla durağan engeller eklenmiştir ve bununla da yetinilmeyip ortama hareketli engeller eklenmiştir. Elde edilen son senaryo da zor senaryo olarak adlandırılabilir. Adaletli bir değerlendirmenin gerçekleştirilebilmesi adına her farklı SLAM yapısı için senaryolar sabit tutulmuştur yani herhangi bir değişiklik uygulamamıştır ve durum hareketli engellere de uyarlanmıştır. Ayrıca her durumda aracın aynı başlangıç noktasından başlayıp aynı

bitiş noktasını kendisine hedef olarak belirlemesine dikkat edilmiştir. Bu kriterler doğrultusunda deneysel çalışmalar yapılmıştır.

Benzer şekilde bir kıyaslama da geleneksel yapıdaki 2B lazer sensör yapısıyla TRAP yapısında yerleştirilen 2B lazer sensörler hakkında yapıldı. Bu kıyaslamada da bir önceki senaryolarla birebir aynı senaryolar kullanılmıştır ve başlangıç noktalarıyla bitiş noktaları da yine birebir aynı kabul edilmiştir.

Ortama hareketli engellerin de dahil edildiği üçüncü senaryoda TRAP ve geleneksel yapı arasındaki fark gözle görülür bir şekilde hissedilmiştir. Geleneksel 2B lazer sensör yapısında ortamda hareketli bir engel algılandığında koşturulan algoritma bunun hareketli bir engel olup olmadığını doğrudan anlayamamakta. Sensörün algılamış olduğu yapı her ne kadar bir engel olabilse de algılanan bu yapı haritada meydana gelen bir kaymanın da sonucu olabilir. Araç algılamış olduğu yapının engel mi yoksa haritada meydana gelen bir kayma mı olduğunu anlamak için ek verilere ihtiyaç duymaktadır. Bu ek verilerin toplanabilmesi için aracın kendi eksenini etrafında birkaç kere dönmesi gerekmektedir ve bu dönmelerin sonucunda ihtiyaç duyulan gerekli veri alınmış olacaktır. Araç harekete başladıktan sonra tekrar hareketli bir engelle karşılaşsa tekrar aynı şekilde kendi eksenini etrafında dönüp ek verileri alması gerekmektedir. Bu gereklilik de görevin tamamlanmasını uzatmaktadır. TRAP yapısı uygulanmış olan donanım ile aynı deney tekrarlandığında aracın kendi eksenini etrafında dönme gereksinimi ortadan kaldırılmıştır ya da olabildiğince düşürülmüştür. Bu durum görevin tamamlanma süresini büyük oranda azaltmıştır. Ayrıca algoritmanın daha hızlı tamamlanması harcanan pil miktarını da azaltarak daha uzun bir kullanım süresi sağlamaktadır. TRAP yapısının çıkış noktası her ne kadar aracın 360° etrafının kör nokta olmadan taranması olsa da aslında büyük rahatsızlık veren kendi eksenini etrafında dönüp veri toplama ihtiyacını da ortadan kaldırmıştır ya da en aza indirmiştir.



1. INTRODUCTION

There are two main goals for the thesis which are also correlated together. The first goal of the thesis is a comparison of Robot Operating System (ROS) based 2D Laser Range Finder (LRF) Simultaneous Localization and Mapping (SLAM) techniques. Additionally a new position configuration for LRFs has been developed called Transversal Position (TRAP) and benchmarking has been performed with the traditional configuration for 2D LRFs.

The comparisons of the SLAM techniques and position configuration of LRFs have been performed through simulations and real world experiments. The weaknesses and strengths of each SLAM algorithm and LRF position configurations have been explained detailed. These kind of benchmarks are essential for a decision over the SLAM techniques and position configurations for an application in mobile robotics.

In the introduction chapter the motivation, contribution, objectives and outline of the thesis have been given in the following sections.

1.1 Motivation

This thesis is interested in two main tasks. The first task is to make a comparison of available SLAM techniques in ROS for showing the positive and negative sides. This is important to make a decision over the SLAM techniques in implementation. Secondly this thesis is focused on improving a new technique for the LRF position to enhance the results of a SLAM techniques by changing only the hardware configuration of the experimental setup without altering the algorithm structure of codes.

The biggest motivation of this work is to making a connection between the theory and practice. The theories generated in the different areas of mobile robotics are crucial and promising for the future; however, if these theories and techniques do not implement in real systems these theories would be meaningless for the improvement. For this reason the SLAM techniques have been tried to implement in an open source software. ROS

is chosen as the software while it is an open source platform and is also one the most preferred software in academic and industrial applications.

The idea of improving the results of a SLAM algorithm by just modifying the hardware configuration without any change in the algorithm makes also a great impact on the motivation.

1.2 Objectives

This thesis has actually two main objectives. The first objective of this work is to make a presentation of a comparative evaluation of different SLAM techniques available for ROS. The tests have been performed simulations and real world experiments. The second objective is to offer a new position configuration for the LRFs in SLAM algorithms to improve the performance.

1.3 Contribution

The biggest contribution of this work is achieved by the TRAP configuration. Almost every 2D LRF based SLAM algorithm is applied with the traditional position configuration of LRF in which one LRF is positioned on the front of the mobile robot centered. However in TRAP configuration a pair of 2D LRFs with 270° scanning range are replaced with exterior corresponding angles to the front left/right and rear right/left of the vehicle and offers a 360° angle of field of view for the mobile robot without any blind region. The TRAP configuration ensures the AGV to identify anything inside the scanning area instantaneously. Furthermore, the experimental works in benchmarking of the SLAM techniques available in ROS provides a detailed review for each algorithm.

1.4 Outline

This thesis is organized in different sections, each referring to distinct phases of the work developed. The first chapter introduces the motivation, objectives, contributions, and the outline of this work.

Chapter 2 introduces related works on the literature. This chapter offers an introduction to the literature before starting to the technical chapters.

The technical chapter of the thesis starts in chapter 3 and ends in chapter 6. These technical chapters provide the fundamental technical information for the experimental works.

Chapter 3 is about the uncertainty problem in robotics. The uncertainty has been described in two parts; filters under uncertainty in section 3.1, in which the filters have been classified as parametric and nonparametric filters, and modeling under uncertainty in section 3.2, in which the modeling has been split into robot motion model and sensor model.

Chapter 4 is an extensive detailed chapter correlated with Robot Operating System (ROS). Basics of ROS and general definitions of ROS architecture have been given. Also this chapter contains a short tutorial about ROS and details about the list of codes have been given in Appendix A1. and A.2. At the end of the chapter a case study of ROS under simulation environment and a real time implementation of ROS have been performed.

Chapter 5 is about the localization and mapping problem in robotics; moreover, each problem has been defined individually in their own sections.

Chapter 6 is about the SLAM problem and the implementation of SLAM. Roughly this chapter has been constructed by two parts. In the first part of the chapter the SLAM problem under uncertainty have been defined. In the second part of the chapter some SLAM methods have been analyzed which are available in ROS. This chapter is also the final chapter of the technical chapters.

Chapter 7 is actually the main chapter of the thesis in which the experimental works have been achieved. In section 7.1 the experimental setup has been described. Section 7.2 is about the TRAP configuration which is a new Laser Range Finder (LRF) position configuration developed in this thesis. In section 7.3 experiments have been performed for a detailed comparison of SLAM techniques and the TRAP configuration.

Chapter 8 is the final part of the thesis in which the practical application of the study is discussed. Future works about this study have been defined and the thesis has been concluded.

2. RELATED WORKS AND LITERATURE SURVEY

Simultaneous localization and mapping (SLAM) is one of the most complex problems in robotics. In this problem the pose estimation of the mobile robot and the estimation of the environment have been executed simultaneously. In SLAM differently from the localization and mapping the predictions have been done without any initial information.

The first research over the SLAM problem has been performed by Smith and Cheesman [1] in 1987. One year later in 1988 Durrant-Whyte [2] offered the mathematical representation of the SLAM problem. During these years many researcher from all around the world have been worked with this problem. In the earlier times of the development of SLAM problem the relationship of mapping and localization on SLAM have not been known well and the researches have been performed in different areas from mapping and localization. In 1996 the SLAM acronym has been used first time by Durrant-Whyte, Rye and Nebot [3] in a journal printed in International Symposium on Robotics Research. In the meantime there have been other research groups focused on the localization and mapping problems together all around the world and one of the most knowns are Massachusetts Institute of Technology, Universidad de Zaragoza, Australian Centre for Field Robotics, etc. At the end of the 20th century a Kalman based filter have been tried to implement into a solution of SLAM problem first time by Thrun, Fox and Burgard [4]. A detailed survey for SLAM have been published by Durrant-Whyte and Bailey [5] and [6] in two parts.

During the last two decades numerous SLAM techniques, based on various algorithms and sensors, have been developed in all around the world. 2D laser range finder (LRF), 3D LRF, monocular camera, stereo camera and radar are on of a possible group of the sensors implemented in SLAM. Additionally the algorithms applied in SLAM vary, such as parametric filters, particle filters, optimization techniques, etc.

OpenSLAM [7] is a platform which provides the researchers interested in developing SLAM algorithms to publish their algorithms for information exchange. The authors

have the opportunity to explain their algorithms and published works related with the SLAM method. Bailey [8] generated a SLAM package which includes EKF-SLAM, UKF-SLAM, FastSLAM 1.0 and FastSLAM 2.0. Elizar and Parr [9] created DP-SLAM where this method has been described by Elizar and Parr [10]. Grisetti, Stachniss and Burgard [11] produced GMapping where this method has been described by Grisetti, Stachniss and Burgard [12]. Haehnel et. al. [13] generated GridSLAM where this method has been described by Haehnel et. al. [14]. Zhao, Huang, and Dissanayake [15] created Linear SLAM where this method has been described by Zhao, Huang, and Dissanayake [16]. Ball et. al. [17] produced ratSLAM where this method has been described by Ball et. al. [18]. Blanco, Madrigal, and Gonzalez [19] generated Bayesian range-only SLAM, RO-SLAM where this method has been described by Blanco, Madrigal, and Gonzalez [20]. Steux and Hamzaoui [21] created tinySLAM where this method has been described by Steux and Hamzaoui [22]. Mur-Artal, Montiel and Tardos [23] produced ORB-SLAM where this method has been described by Mur-Artal, Montiel and Tardos [23]. Civera and Montiel [24] generated EKFmonocularSLAM where this method has been described by Civera et. al. [25].

3. THE UNCERTAINTY PROBLEM IN ROBOTICS

The terms of uncertainty and robotics are separately far-reaching subjects; however, under this section and actually in this entire thesis it is focused on the uncertainty problem in robotics. To understand this problem it is necessary to give the explanations about uncertainty and robotics individually. Immediately afterwards an appropriate atmosphere will be generated for explaining the uncertainty problem in robotics.

In a most general sense robots can be defined as objects designed for make life easier by perform tasks in humans routine life and the science interested with these objects is robotics. This definitions is really very extensive. It is possible to make a classification about robots and robotics more detailed, but the focus point here is the uncertainty problem in robotics not directly robotics.

Uncertainty of a situation means briefly being not exact for any situation. The truth of any condition is expressed with statistical notation and it is possible to remark that there is a big relation between uncertainty and probability. Uncertain situations are mostly defined by probabilities.

Goal of this chapter is to combine the uncertainty problem with robotics together and make some useful explanations to enlighten the uncertainty problem in robotics. To achieve this objective the problem have been analyzed for filters under uncertainty and modeling under uncertainty.

3.1 Estimation Under Uncertainty

Filters described in this section are grouped in two families; Parametric filters and Non-parametric filters. Filters are great tools to transform the uncertainty situation or data into a meaningful one. The process can be defined actually very simple as the uncertainty will be filtered and modified into a data which is more sensible.

The most important thing in filtering is to select the most convenient filter type for the uncertainty. An incorrect implementation of a filter to an uncertainty most probably

will cause to a catastrophic result by converging the system into a more uncertain condition. That's why it is important to understand the filter families and filter types properly.

3.1.1 Estimation with parametric filters

Parametric filters have been accepted that the distribution of the beliefs are represented by multivariate normal distribution. The distribution can be defined by Equation (3.1) where the maximum value of the density over variable x is 1 and the minimum value is 0 under the probabilistic rules. Parametric filters are also called Parametric filters since it is obvious that the distribution is related with mean and covariance.

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu)\right] \quad (3.1)$$

The probabilistic density given in Equation (3.1) has two parameters; μ as mean and Σ as covariance. The properties of the given parameters could be defined as, the covariance is a positive semi-defined symmetric quadratic matrix and the mean is a vector related with the dimension of variable x . For an n -dimension x , the dimension of μ will be $n \times 1$ and Σ will have a dimension of $n \times n$.

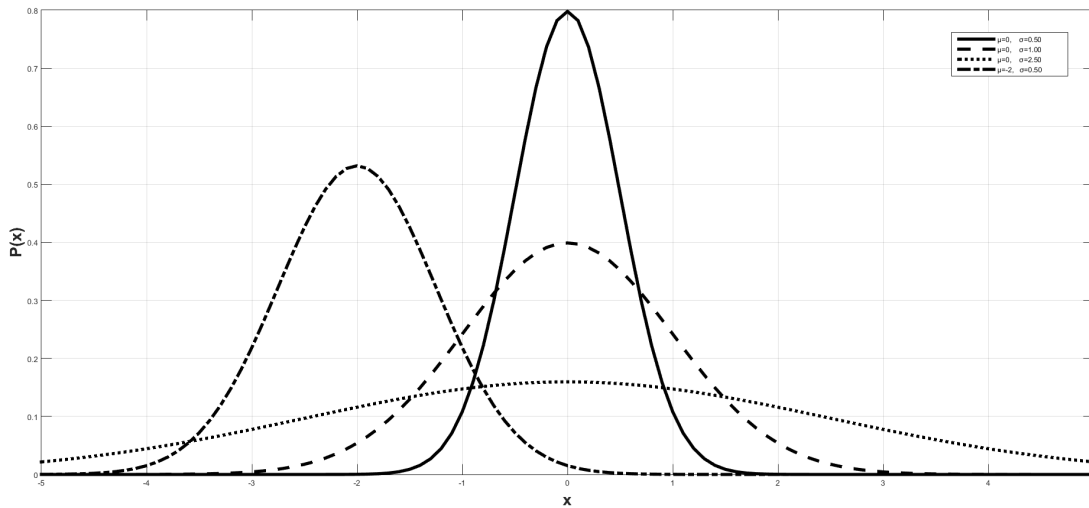


Figure 3.1 : Gauss probability distribution functions for different parameters

Parametric type filters are designed for distributions defined by Gaussian functions. A Gaussian function is in a bell curve shaped form. The length, width and shape of the bell show an alteration by the change in mean and covariance. Greater covariance

results as bell with a greater width and smaller length; however, smaller covariance results as the opposite. Figure (3.1) shows different Gauss functions with different mean and covariance values.

With a plainer expression the covariance is about the probability of the mean. Mean is always the value with the biggest probability. Covariance indicates the certainty of that value. The relationship of mean and covariance could be simply defined with a physical example. Let the value of the mean be the sensor measurement value. This measurement could be anything like distance, temperature, voltage, etc. When it is possible to declare that the measurement value is nearly to the exact value, covariance could be defined with small values, since the accuracy is very high. This case also indicates that the probability of being equal to mean is very high. In the opposite situation covariance must be defined by high numbers, while the error in measurement is much larger.

Parametric filters have been examined in three different groups as Kalman, Extended Kalman and Unscented Kalman filters. Even though these structure are named as filters they also are observers. A filter and observer mixture defines these type of filters as best.

3.1.1.1 Kalman filters

Two necessities are available to implement Kalman filter to a system. First of all the system must be linear and secondly system must also Gaussian. This means the states must also be linear with Gaussian noise as defined in Equation (3.2).

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t \quad (3.2)$$

x_t and x_{t-1} are the state vectors at time t with a size of $n \times 1$ and $t - 1$. u_t is used for the control vector at time t with a size of $m \times 1$. A_t is a square matrix with a size of $n \times n$ and B_t is another matrix with a size of $n \times m$. ε_t is a Gaussian vector with a size of $n \times 1$ which defines the error of the system. Gaussian errors are defined with 0 (zero) mean and its covariance is stated with R_t . All of these definitions are about the states and the probability of the states.

Measurement is also an important part of Kalman Filters, which is defined as z_t , and is shown in Equation (3.3). As it has been assumed that the number of states are n , the number of control inputs are m and the number of measurements are k , z_t is a vector with the size of $k \times 1$. C_t is a matrix and the dimensions are $k \times n$ and if $k = n$, C_t is a square matrix. The last character of Equation (3.3) δ_t represents the noise in measurement and is a vector with the same size of the measurement vector z_t . δ_t is similar to ε_t with 0 (zero) mean and a covariance of Q_t .

$$z_t = C_t x_t + \delta_t \quad (3.3)$$

The Kalman Filter algorithm is actually a two part algorithm. Prediction is the first part of Kalman Filter in which mean and covariance for next time step is predicted by previous mean and covariance values. These predictions are used in the second part of Kalman Filter, the measurement. Kalman gain, mean and covariance for the next step is generated in this second part.

$$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t \quad (3.4)$$

The parameters with bar represents that they are predicted values. In Equation (3.4) $\bar{\mu}_t$ is calculated by replacing μ_{t-1} with x_{t-1} in Equation (3.2).

$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t \quad (3.5)$$

The prediction of the covariance for next step is operated in Equation (3.5). These predictions are used in Kalman Filters second part in Equations (3.6)-(3.7).

$$K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1} \quad (3.6)$$

Kalman gain K_t is a parameter that is used for correcting the predicted mean and covariance values in Equations (3.7) and (3.8).

$$\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t) \quad (3.7)$$

In Equation (3.7), the predicted mean is compared with the measurement. If there is a difference between the measurement and prediction, the difference will be scaled with Kalman gain and the predicted mean $\bar{\mu}_t$ will be updated.

$$\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t \quad (3.8)$$

The corrected parameters μ_t and Σ_t are the outputs of the Kalman filter.

3.1.1.2 Extended kalman filters

Extended Kalman Filter (EKF) is indeed a modification of Kalman Filter to adapt it for nonlinear systems. The requirement to adapt Parametric Filters into nonlinear systems revealed EKF. The nonlinear form of state and measurement is given in Equation (3.9) and (3.10).

$$x_t = g(u_t, x_{t-1}) + \varepsilon_t \quad (3.9)$$

$$z_t = h(x_t) + \delta_t \quad (3.10)$$

The functions g and h are nonlinear functions of state transition probability and measurement probability. These functions are the upgrades in Kalman Filters. Extended Kalman Filter requires a linearisation step to implement the algorithm. The linearisation of each nonlinear function is achieved by taking the partial derivative.

Let G_t be the Jacobian Matrix of the nonlinear function $g(u_t, x_{t-1})$ which is obtained by taking the partial derivative given in Equation (3.11).

$$G_t = \frac{\partial g(\bar{x}_{t-1}, u_t)}{\partial x_{t-1}} = \begin{bmatrix} \frac{\partial g^1(\bar{x}_{t-1}, u_t)}{\partial x_{t-1}^1} & \frac{\partial g^1(\bar{x}_{t-1}, u_t)}{\partial x_{t-1}^2} & \dots & \frac{\partial g^1(\bar{x}_{t-1}, u_t)}{\partial x_{t-1}^n} \\ \frac{\partial g^2(\bar{x}_{t-1}, u_t)}{\partial x_{t-1}^1} & \frac{\partial g^2(\bar{x}_{t-1}, u_t)}{\partial x_{t-1}^2} & \dots & \frac{\partial g^2(\bar{x}_{t-1}, u_t)}{\partial x_{t-1}^n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g^n(\bar{x}_{t-1}, u_t)}{\partial x_{t-1}^1} & \frac{\partial g^n(\bar{x}_{t-1}, u_t)}{\partial x_{t-1}^2} & \dots & \frac{\partial g^n(\bar{x}_{t-1}, u_t)}{\partial x_{t-1}^n} \end{bmatrix} \quad (3.11)$$

The Jacobian Matrix of $h(x_t)$ will be found similarly to the G_t matrix. H_t is equal to the partial derivative of $h(x_t)$ with respect to x_{t-1} given in Equation (3.12).

$$H_t = \frac{\partial h(\bar{x}_t)}{\partial x_t} = \begin{bmatrix} \frac{\partial h^1(\bar{x}_t)}{\partial x_t^1} & \frac{\partial h^1(\bar{x}_t)}{\partial x_t^2} & \dots & \frac{\partial h^1(\bar{x}_t)}{\partial x_t^n} \\ \frac{\partial h^2(\bar{x}_t)}{\partial x_t^1} & \frac{\partial h^2(\bar{x}_t)}{\partial x_t^2} & \dots & \frac{\partial h^2(\bar{x}_t)}{\partial x_t^n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h^m(\bar{x}_t)}{\partial x_t^1} & \frac{\partial h^m(\bar{x}_t)}{\partial x_t^2} & \dots & \frac{\partial h^m(\bar{x}_t)}{\partial x_t^n} \end{bmatrix} \quad (3.12)$$

The dimensions of the Jacobian Matrices defined in Equations (3.11) and (3.12) are $n \times n$ and $m \times n$. The reason of this result is that the number of states are n and the number of measurements are m .

The necessity to start to the Extended Kalman Filter algorithm is to linearise the nonlinear functions $g(u_t, x_{t-1})$ and $h(x_t)$. The rest of the Extended Kalman Filter algorithm is very similar to the Kalman Filter algorithm. Table (3.1) shows the state and measurement equations for Kalman Filter and Extended Kalman Filter together.

Table 3.1 : Similarities in kalman and extended kalman filter equations

	Kalman Filter	Extended Kalman Filter
State (x_t)	$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t$	$x_t = g(u_t, x_{t-1}) + \varepsilon_t$
Measurement (z_t)	$z_t = C_t x_t + \delta_t$	$z_t = h(x_t) + \delta_t$

Similarly to the Kalman Filter algorithm, Extended Kalman Filter algorithm has also a prediction and a measurement part. After defining the Jacobian Matrix for linearisation, prediction is the first part of Extended Kalman Filter in which mean and covariance for next time step is predicted by previous mean and covariance values. These predictions are used in the second part of Extended Kalman Filter, the measurement. Kalman gain, mean and covariance for the next step is generated in this second part.

$$\bar{\mu}_t = g(u_t, \mu_{t-1}) \quad (3.13)$$

The parameters with bar represents that they are predicted values. In Equation (3.13) $\bar{\mu}_t$ is calculated by replacing μ_{t-1} with x_{t-1} in Equation (3.9).

$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t \quad (3.14)$$

The prediction of the covariance for next step is operated in Equation (3.14). These predictions are used in Extended Kalman Filters second part in Equations

(3.15)-(3.17). The only difference between Equation (3.5) and (3.14) is that A_t is replaced with G_t . In Kalman Filter the system is linear and there is no difficulty by generating A_t matrix; however, implementing G_t matrix into the algorithm is a little challenging, a Jacobian calculation is needed. The complexity of the Jacobian is increased by the number of states quadratically. This linearisation step is the biggest handicap of Extended Kalman Filter algorithm.

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1} \quad (3.15)$$

Kalman gain K_t is a parameter that is used for correcting the predicted mean and covariance values in Equations (3.16) and (3.17). The only difference between Equation (3.6) and (3.15) is that C_t is replaced with H_t , the Jacobian Matrix of $h(x_t)$.

$$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t)) \quad (3.16)$$

In Equation (3.16), the predicted mean is compared with the measurement. If there is a difference between the measurement and prediction, the difference will be scaled with Kalman gain and the predicted mean $\bar{\mu}_t$ will be updated.

$$\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t \quad (3.17)$$

The corrected parameters μ_t and Σ_t are the outputs of the Extended Kalman filter. Although being complex because of the linearisation step, fundamentally both filter algorithms Kalman and Extended Kalman are very similar. G_t is the equivalence of A_t , H_t is the equivalence of C_t and $h(\bar{\mu}_t)$, is the equivalence of $C_t \bar{\mu}_t$.

3.1.1.3 Unscented kalman filters

Unscented Kalman Filter is also generated for nonlinear Gaussian systems. Similarly to Extended Kalman Filter there is a linearisation step; however, this linearisation is quite different from the method used in Extended Kalman Filter. This linearisation is a stochastic linearisation which depends on weight some points of the nonlinear function by regression. In this weight process Unscented Kalman Filter assigns some points on the Gaussian which are called sigma point. The number of the sigma points depends to the dimension of the system. The sigma point positioned to the mean and for each

dimension two points symmetrically to a line along the mean are selected. According to this calculation it is possible to declare that for an n dimensional system there exists $2n + 1$ sigma points.

For a Gaussian with a dimension of n , mean of μ and covariance of Σ the sigma points $SP^{[i]}$ have been defined in Equations (3.18)-(3.20).

$$SP^{[0]} = \mu \quad (3.18)$$

In every Unscented Kalman Filter application the first sigma point always located at the mean μ defined in Equation (3.18).

$$SP^{[i]} = \mu + (\sqrt{\Sigma(n + \lambda)})_i \quad (3.19)$$

The sigma points in Equation (3.19) and (3.20) are symmetrically to each other. Equation (3.19) is defined for $i = 1, 2, \dots, n$ and Equation (3.20) is defined for $i = n + 1, n + 2, \dots, n + n$.

$$SP^{[i]} = \mu - (\sqrt{\Sigma(n + \lambda)})_{i-n} \quad (3.20)$$

λ is a parameter to define the locations of the sigma points with respect to mean. The formulation of λ has been given in equation (3.21) where n is the dimension, α and β are constant numbers.

$$\lambda = \alpha^2(n + \beta) - n \quad (3.21)$$

The λ parameter is also used for decide the weight for each sigma point. There are two weights defined for every sigma point related with mean $W_m^{[i]}$ and covariance $W_c^{[i]}$. The calculation of weight for $i = 0$ is given in Equation (3.22) and (3.23).

$$W_m^{[0]} = \frac{\lambda}{n + \lambda} \quad (3.22)$$

$$W_c^{[0]} = \left(\frac{\lambda}{n + \lambda}\right) + (1 - \alpha^2 + \eta) \quad (3.23)$$

η is a parameter to define the exactness of the Gaussian and is chosen $\eta = 2$ for an exact Gauss. The weights for $i \neq 0$ are equal for both mean and covariance and are given in Equation (3.24).

$$W_m^{[i]} = W_c^{[i]} = \frac{1}{2(n + \lambda)} \quad (3.24)$$

After the definition of sigma points their values in the nonlinear $g(u_t, x_{t-1})$ function must be find as defined in Equation (3.25).

$$\psi^{[i]} = g(SP^{[i]}) \quad (3.25)$$

Mean and covariance will be updated by weight the parameters. The update rule for the parameters are given in Equation (3.26) and (3.27).

$$\mu^* = \sum_{i=0}^{2n} W_m^{[i]} \psi^{[i]} \quad (3.26)$$

$$\Sigma^* = \sum_{i=0}^{2n} W_c^{[i]} (\psi^{[i]} - \mu^*)(\psi^{[i]} - \mu^*)^T \quad (3.27)$$

The main difference of Extended and Unscented Kalman Filters is the linearisation part which is not operated in Unscented Kalman Filter algorithm and is replaced by sigma points and weight operation. Equation (3.26) and (3.27) are the results of the weight operation by sigma point and these equations generates the prediction part of the Unscented Kalman Filter. In prediction part Equation (3.26) have been used directly to predict the mean. For the prediction of covariance Equation (3.27) have been updated by adding covariance of prediction noise R_t .

The measurement step in Unscented Kalman Filter is also similar to the previous Parametric filters and consists of three steps; however, there are some tiny differences given in Equations from (3.28) to (3.30).

$$K_t = \bar{\Sigma}_t S_t^{-1} \quad (3.28)$$

By calculation of the Kalman gain in Equation (3.28) a new parameter S_t is used which is the covariance of predicted observation $\hat{Z}_t$. This parameter is also implemented to the measurement of mean part in Equation (3.29).

$$\mu_t = \bar{\mu}_t + K_t(Z_t - \hat{Z}_t) \quad (3.29)$$

The update of the mean depends on the difference between the measurement Z_t and predicted observation of the measurement $\hat{Z}_t$.

$$\Sigma_t = \bar{\Sigma}_t^z - K_t S_t K_t^T \quad (3.30)$$

As a distinctness from Extended Kalman Filter the covariance have been updated by the predicted variance $\bar{\Sigma}_t^z$ and this variance with respect to the sigma point $SP^{[i]}$ and the observation of the sigma point $\bar{Z}_t^{[i]}$ is given in Equation (3.31).

$$\bar{\Sigma}_t^z = \sum_{i=0}^{2n} W_c^{[i]} (SP^{[i]} - \bar{\mu})(\bar{Z}_t^{[i]} - \hat{Z}_t)^T \quad (3.31)$$

The observation of the sigma point and the predicted observation are defined in Equation (3.32) and (3.33).

$$\bar{Z}_t = h(SP^{[i]}) \quad (3.32)$$

$$\hat{Z}_t = \sum_{i=0}^{2n} W_m^{[i]} \bar{Z}_t^{[i]} \quad (3.33)$$

To obtain the covariance of the predicted observation S_t Equation (3.32) and (3.33) must be calculated and these variables must be replaced in Equation (3.34).

$$S_t = \sum_{i=0}^{2n} W_c^{[i]} (\bar{Z}_t^{[i]} - \hat{Z}_t)(\bar{Z}_t^{[i]} - \hat{Z}_t)^T + Q_t \quad (3.34)$$

The corrected parameters μ_t and Σ_t are the outputs of the Unscented Kalman filter. Although the operations with the sigma points increase the complexity of the algorithm, the lack of necessity in calculating the Jacobian Matrices decrease the process load of the algorithm.

3.1.2 Estimation with non-parametric filters

As it has been mentioned in section 3.1.1 parametric filters are interested in systems defined by functions with known simple parameters. On the contrary of parametric filters, non-parametric filters have been generated for systems without any known constants. There is a possibility to state that non-parametric filters deal with numerous parameters which is the main difference with parametric filters.

There are many possible variations for the non-parametric filters created by methods different from each other; however, in this thesis one of the most popular non-parametric filter has been analyzed. The non-parameter filter have been developed by defining the posterior as samples and are called particle filter. Particle filters are one of the most popular non-parametric filter types preferred in robotics.

3.1.2.1 Particle filter

Particle filters approach to the posterior totality generated by particles with different size and weight. More clearly the posterior has been represented by randomly generated samples. Figure 3.2 represents the logic of a particle filter with a very simple approach.

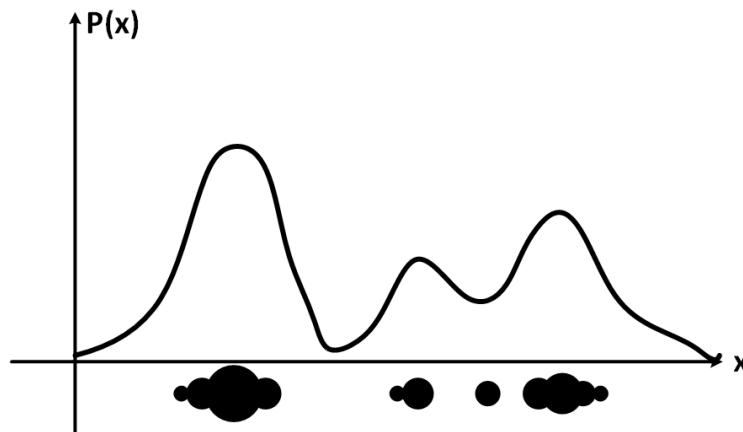


Figure 3.2 : The representation of a particle filter

The illustration in Figure 3.2 shows that a particle filter can deal a nonlinear function very easy without any complex mathematical calculations. Some samples have been assigned parallel to the magnitude of the function. The easiness in mathematical representation and dealing with nonlinear systems are one of the biggest advantages of particle filters.

The most important thing in particle filters is the sampling process in which the particles have been decided. The samples chosen as particles must be proportional to the posterior, since these particles represent the posterior. The density of the samples plays a role for generating the true posterior.

In a particle filter algorithm the inputs are the initial particle or the particle generated in the previous time interval X_{t-1} , the control input U_t and the sensor measurement Z_t . The output of the particle filter algorithm is as might be expected X_t , the particle at time interval t .

$$X_{0:t}^{[m]} = X_0^{[m]}, X_1^{[m]}, \dots, X_t^{[m]} \quad (3.35)$$

The particles from the beginning 0 until t have been shown in the equation 3.35 above. These particles have been generated from the posterior for a representation. By determining all the states the posterior have been shown in equation 3.36.

$$bel(X_{0:t}) = p(X_{0:t} | U_{1:t}, Z_{1:t}) \quad (3.36)$$

In equation 3.35 m represents the m -th particle for the system. Sampling the prior $p(x)$ by the initial condition x_0 results to obtain the first particle. This process must be continued until time interval $t - 1$ according to $bel(x_{0:t-1})$. For a more detailed information about particle filters reference [26] and reference [27] should be reviewed.

3.2 Modeling Under Uncertainty

To control a system it is important to generate a model of the system to define the physical structure with mathematical equations. To implement an algorithm to the system a model is necessary for the system and as the algorithms are probabilistic based methods, the models will be generated with the guidance of probabilistic rules. Modelling Under Uncertainty section has been consisted by two subsection Robot Motion Models and Sensor Models in which modelling under uncertainty information has been given for mobile robots and laser range finder (LRF) sensor. The information offered in this section is crucial to implement the probabilistic based models.

3.2.1 Robot motion models

Under this section the motion model of mobile robots have been analysed. Robot motion models have been dealt under this section have been classified into two classes, the velocity motion model and the nonholonomic motion model.

The mobile robot used for the experimental works is a differential drive with four wheels. The front and the rear wheels for each side are connected together and can not move separately like a differential wheel. Detailed information about the mobile platforms motion model applied in experimental works have been given in chapter 7. The reason for reviewing these two motion model methods is the accommodativeness with differential drives.

In the forthcoming motion models the structure has been given into two parts, kinematic equations and probabilistic equations. To understand these informations it is necessary to be familiar to the kinematic configurations and probabilistic kinematics.

The kinematics of a mobile robot is shown by the Cartesian coordinates (x, y, z) and the Euler angles (roll, pitch, yaw). Both configurations are 3 dimensional and the kinematic configuration is generally a 6 dimensional vector; however, mostly in this thesis the kinematic configuration will be shown in 3 dimensions, since the mobile robots mostly moves in a planar environment. Because of this reason the pose of the mobile robot will be a three dimensional vector shown in Equation (3.37).

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \quad (3.37)$$

The notation given in Equation (3.37) is also named as posture kinematics. Posture is a combination of position and orientation. The first two parameters x and y represents the location of the robot on the planar environment, θ represents the heading angle of the mobile robot.

Probabilistic kinematics is interested with the conditional density $p(x_t | u_t, x_{t-1})$. In this probability distribution x_t and x_{t-1} are the poses of the mobile robot and u_t is a control input which moves x_t to x_{t-1} .

3.2.1.1 Velocity motion model

In this motion model it has been assumed that the rotational (ω) and translational (v) velocities are the control inputs for the mobile robot. The control input u at time t is given in Equation (3.38).

$$u_t = \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} \quad (3.38)$$

The translational velocity at time t is represented with v_t and the rotational velocity at time t is represented with ω_t .

The aim is to calculate the probability $p(x_t|u_t, x_{t-1})$. $x_{t-1} = (x \ y \ \theta)^T$ represents the initial pose, $x_t = (x' \ y' \ \theta')^T$ is the pose that we are interested to find, and u_t given in Equation (3.38) is the control input which moves the robot from x_t to x_{t-1} .

In an ideal, noise free environment the motion of the robot could be defined by rotational moves and the radius of the movement is the ratio of the translational and angular velocities $r = v/\omega$. A linear movement is also a circular move with a radius with equal to infinite.

For assuming the velocities v and ω constant for Δt time, the center of the circle can be calculated with the following Equations (3.39) and (3.40):

$$x_c = x - \frac{v}{\omega} \sin \theta \quad (3.39)$$

$$y_c = y + \frac{v}{\omega} \cos \theta \quad (3.40)$$

From $t-1$ to t after Δt time, the robot will move from x_{t-1} to x_t . This change in pose is given in Equation (3.41):

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x_c + \frac{v}{\omega} \sin(\theta + \omega \Delta t) \\ y_c - \frac{v}{\omega} \cos(\theta + \omega \Delta t) \\ \theta + \omega \Delta t \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} -\frac{v}{\omega} \sin \theta + \frac{v}{\omega} \sin(\theta + \omega \Delta t) \\ \frac{v}{\omega} \cos \theta - \frac{v}{\omega} \cos(\theta + \omega \Delta t) \\ \omega \Delta t \end{bmatrix} \quad (3.41)$$

In real world there is no possibility to switch the velocity of the robot suddenly, and also keeping the velocity constant is also not probable. If the time interval Δt will

be chosen with small values, then the implementation of these assumptions could be possible.

Adding noise to the robot motion model will improve the model and the implementation will be more realistic. An error must be added to the commanded velocities to obtain the actual velocities. The actual velocities are shown with a hat and calculated by Equation (3.42):

$$\begin{bmatrix} \hat{v} \\ \hat{\omega} \end{bmatrix} = \begin{bmatrix} v \\ \omega \end{bmatrix} + \begin{bmatrix} \eta(\alpha_1|v| + \alpha_2|\omega|) \\ \eta(\alpha_3|v| + \alpha_4|\omega|) \end{bmatrix} \quad (3.42)$$

The noise η_b is a normal distributed or a triangular distributed white noise with a variance of b^2 . $\alpha_1 - \alpha_4$ are robot specific error parameters. These parameters depends on the accuracy of the robot.

There is no limitation to chose distribution model for the noise in velocities. Mostly a Gaussian distribution is chosen, but the triangular distribution is an alternative and is also preferred to define the noise. The model for the next pose after showing the effect of noise is given in Equation (3.43).

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} -\frac{\hat{v}}{\hat{\omega}} \sin \theta + \frac{\hat{v}}{\hat{\omega}} \sin(\theta + \hat{\omega}\Delta t) \\ \frac{\hat{v}}{\hat{\omega}} \cos \theta - \frac{\hat{v}}{\hat{\omega}} \cos(\theta + \hat{\omega}\Delta t) \\ \hat{\omega}\Delta t \end{bmatrix} \quad (3.43)$$

Equation (3.43) represents the final orientation of the robot by adding the noises in the velocities of the robot; all the same, there is a missing part to implement this model to real world. The pose of the robot is a three dimensional vector; notwithstanding, the noise η_b is defined only for the velocities. To imply the velocity motion model into a system in real world, the heading angle θ of the system must be also generalized. To achieve this goal it has been assumed that the robot generates an extra rotation as an error which is defined as $\hat{\phi}$. The new definition $\hat{\phi}$ is similar to the noises in velocities and is given in Equation (3.44):

$$\hat{\phi} = \eta(\alpha_5|v| + \alpha_6|\omega|) \quad (3.44)$$

In Equation (3.44) the parameters α_5 and α_6 are again parameters specific to the robot.

$$\theta' = \theta + \hat{\omega}\Delta t + \hat{\phi}\Delta t \quad (3.45)$$

The rearranged heading angle with noise is given in Equation (3.45). This form is more convenient to the real world applications and the reorganized velocity motion model is given in Equation (3.46):

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} -\frac{\hat{v}}{\hat{\omega}} \sin \theta + \frac{\hat{v}}{\hat{\omega}} \sin(\theta + \hat{\omega}\Delta t) \\ \frac{\hat{v}}{\hat{\omega}} \cos \theta - \frac{\hat{v}}{\hat{\omega}} \cos(\theta + \hat{\omega}\Delta t) \\ \hat{\omega}\Delta t + \hat{\phi}\Delta t \end{bmatrix} \quad (3.46)$$

The most general form of the velocity motion model is given in Equation (3.46). The calculation of the probability distribution $p(x_t|u_t, x_{t-1})$ in velocity motion model is simply given in Algorithm 1.

Algorithm 1 Velocity motion model

- 1: $\mu = 0.5 \times \frac{(x - x') \cos \theta + (y - y') \sin \theta}{(y - y') \cos \theta - (x - x') \sin \theta}$
 - 2: $x^c = \frac{x + x'}{2} + \mu(y - y')$
 - 3: $y^c = \frac{y + y'}{2} + \mu(x - x')$
 - 4: $r^c = \sqrt{(x - x^c)^2 + (y - y^c)^2}$
 - 5: $\Delta \theta = \text{atan2}((y' - y^c), (x' - x^c)) - \text{atan2}((y - y^c), (x - x^c))$
 - 6: $\hat{v} = \frac{\Delta \theta}{\Delta t} r^*$
 - 7: $\hat{\omega} = \frac{\Delta \theta}{\Delta t}$
 - 8: $\hat{\phi} = \frac{\theta' - \theta}{\Delta t} - \hat{\omega}$
 - 9: $p_1 = p((v - \hat{v}), (\alpha_1|v| + \alpha_2|\omega|))$
 - 10: $p_2 = p((\omega - \hat{\omega}), (\alpha_3|v| + \alpha_4|\omega|))$
 - 11: $p_3 = p((\hat{\phi}), (\alpha_5|v| + \alpha_6|\omega|))$
 - 12: $p(x_t|u_t, x_{t-1}) = p_1 p_2 p_3$
-

In Algorithm 1 to finalize the computation of $p(x_t|u_t, x_{t-1})$, three inputs are necessary. x_t , x_{t-1} and u_t are the inputs of the algorithm. For a constant velocity during a time period of Δt for a circular trajectory, the center of the circle $(x^c \ y^c)$ has been defined as given in Equation (3.47):

$$\begin{bmatrix} x^c \\ y^c \end{bmatrix} = \begin{bmatrix} \frac{x+x'}{2} + 0.5 \times \frac{(x-x') \cos \theta + (y-y') \sin \theta}{(y-y') \cos \theta - (x-x') \sin \theta} (y-y') \\ \frac{y+y'}{2} + 0.5 \times \frac{(x-x') \cos \theta + (y-y') \sin \theta}{(y-y') \cos \theta - (x-x') \sin \theta} (x-x') \end{bmatrix} \quad (3.47)$$

Equation (3.47) is the expanded form of the steps 2 and 3 in Algorithm 1. The radius of the circular trajectory is defined as an Euclidean distance in step 4. Also the change in heading angle $\Delta\theta$ is given in step 5. The translational distance of the robot from x_{t-1} to x_t can be found by multiplying the change in heading angle with the radius of the circle given in Equation (3.48):

$$\Delta\chi = r^c \Delta\theta \quad (3.48)$$

After the process of calculating $\Delta\chi$ and $\Delta\theta$ the real velocities and the real control input with noise can be defined easily as given in Equation (3.49):

$$\hat{u}_t = \begin{bmatrix} \hat{v} \\ \hat{\omega} \end{bmatrix} = \Delta t^{-1} \begin{bmatrix} \Delta\chi \\ \Delta\theta \end{bmatrix} \quad (3.49)$$

The error in the rotation of the robot $\hat{\phi}$ can be defined according to Equation (3.50):

$$\hat{\phi} = \Delta t^{-1} (\theta' - \theta) - \hat{\omega} \quad (3.50)$$

The difference between the real control input with error and commanded control input will give the errors defined in Equation (3.51):

$$e_v = v - \hat{v} \quad e_\omega = \omega - \hat{\omega} \quad e_\phi = \hat{\phi} \quad (3.51)$$

These error functions defined in Equation (3.51) will be used to define the probabilities of each variable in velocity motion model.

$$\eta = \begin{bmatrix} \eta(\alpha_1 |v| + \alpha_2 |\omega|)(e_v) \\ \eta(\alpha_3 |v| + \alpha_4 |\omega|)(e_\omega) \\ \eta(\alpha_5 |v| + \alpha_6 |\omega|)(e_\phi) \end{bmatrix} \quad (3.52)$$

Equation (3.52) represents the steps from 9 to 11 in Algorithm 1.

$$p(x_t|u_t, x_{t-1}) = (\eta(\alpha_1|v| + \alpha_2|\omega|)(e_v))(\eta(\alpha_3|v| + \alpha_4|\omega|)(e_\omega))(\eta(\alpha_5|v| + \alpha_6|\omega|)(e_\varphi)) \quad (3.53)$$

As a result o the calculations in velocity motion model the probability $p(x_t|u_t, x_{t-1})$ is achieved in Equation (3.53).



4. ROBOT OPERATING SYSTEM (ROS)

ROS is an open source program which is used for programming robotic systems by researchers in robotics community. Although being used by the researches of robotics community mostly, ROS is not the earliest program available for programming robots; however, it is one of the most used opensource program for this purpose. The necessity to a standardization in robot programming is the main reason for generating a platform like ROS. Stanford University is the pioneer for developing ROS by running two important programs STanford AI Robot (STAIR) and Personal Robots (PR). The period of development have been started in 2007 by Stanford Artificial Intelligence Laboratory (SAIL). The motivation of ROS was to encourage the Stanford AI Robot STAIR and Personal Robot projects. The aim of the project was to design a robot platform to integrate machine learning, navigation, planning, etc. techniques with artificial intelligent. [28]

The biggest motivation of ROS is to produce a framework available for almost every robot by little code modifications. This helpful property will provide the researchers in robotics area to move forward without repeating the boring and challenging previous steps. Less effort and more productivity is the key idea of ROS. After the first step for ROS in 2007 by SAIL, a robotics research institution Willow Garage continued to the ROS development one year later in 2008 with other institutions where the number is more than twenty. A lot of hardware configuration have been adopted to ROS by these institutions. After the improvements in ROS, robot manufacturers have been focused on products compatible with ROS. Not only robots, but also sensors have been produced parallel to robots available for ROS. [29]

ROS is not directly an operating system, it can be defined as an underdeveloped operating system or a meta operating system. ROS is depended to a real operating system. As being follow the Unix basic ROS is actually developed for Linux and especially works well with Ubuntu. Also ROS works with Windows and Mac OS X environments; however, it is just experimental and for an efficient utilization it is

necessary to work on Linux. It has been recommended that having Linux experience is crucial for becoming a ROS user.

There are some points about ROS which must be clarified to elucidate some mistakable topics. First of all, ROS is not a programming language like C++, Python or Matlab. ROS is a meta operating system which use programs mostly written in C++ and Python. Afterwards, it is possible to imagine ROS as a huge library; however, ROS provides some extra tools and it is not true to define ROS as a library. Also being terminal based and not having an advanced graphical user interface cause to a confusion in the definition of ROS. [30]

Being in a open source structure and offering the possibility to commercial use freely is one of the biggest opportunities which makes ROS so popular. Also the hardware available in ROS could be attached to any system very easy by just defining some necessary parameters. In other words ROS offers a great convenience in connection of devices available in this environment. The other facility provided by ROS is the simplicity by transferring the simulations into real time environment by minimal code alteration. There are much more opportunities provided by ROS but there is also one important budget saving facility which must be mentioned about. Rviz, which is a simulation tool in ROS, provides to process with data sets of any type of a sensor. Bag files are used to generate data sets as real. Under the favour of this property, without using a real sensor and being in the environment it is possible to manipulate the source for further research. All of these possibilities affect the motivation of using ROS positively.

4.1 Basics of ROS

This section contains a general definition for ROS which includes a detailed information about ROS architecture. This will be helpful to understand what is going on in ROS. Additionally under this section robots officially use ROS are introduced shortly. Finally basic notes about installation and the installation procedure have been given detailed in the final subsection of this section.

4.1.1 General definitions

As mentioned before, ROS is named as an operating system; however, ROS is not just like an operating system that is used in PC's like Mac OS X or Windows. ROS could be categorized as a middle-ware software or a meta operating system. Moreover, it is possible to define ROS as a bridge to reach to the robot by sending commands. Basically ROS works by publishing and subscribing messages to nodes, topics and services. Figure 4.1 shows visually the interaction of node, service and topics by publishing and subscribing messages.

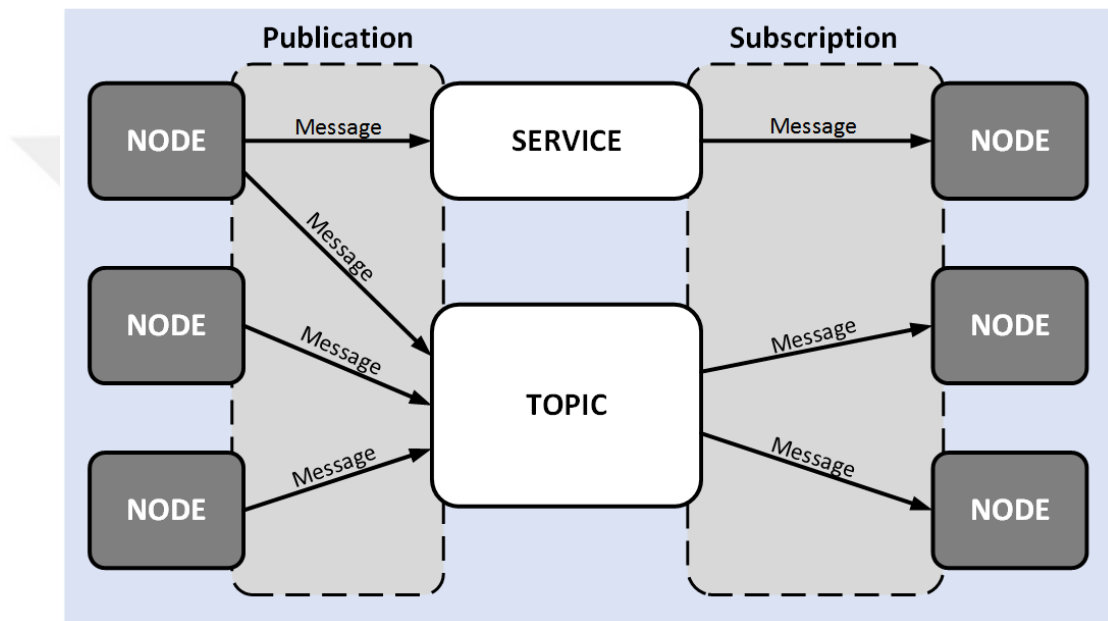


Figure 4.1 : ROS node-topic-service-message relationship

According to figure 4.1 nodes are able to publish more than one message to topics and services. The messages that nodes receive are called subscriptions, each node have the possibility to receive only one message. Services accept a single message, and also generate one message for subscription to a node. Distinctly from nodes, and services, topics have the ability to receive multiple messages, and additionally outputs more than one message.

In ROS architecture system there are many definitions that have to be explained to continue on ROS. The ROS architecture consists from Nodes, Topics, Messages, Services and Packages. The list could be extended; however, for a basic review on ROS these structures are adequate.

The ROS Universe can be simply likened to a library. Similarly it is possible to liken a ROS Repository to bookshelf in a library. Also there is a possibility to liken a Stack to a book and a Package to a chapter of a book.

The ROS Universe is a dynamic universe and is growing every day with the additions and updates. The ROS Universe is consisted from many numerous repositories with different variations of size. Repositories offer an improved file system for the ROS users and developers. Rather than locating all Packages in the same directory, organizing them into sub-directories (repositories) provides an easier navigation on the file system. Similarly to the ROS Universe, Repositories also have been consisted from many other particles called Stacks. The dependency on Stacks is for a better organization on Packages. This system is also helpful for sharing the codes, since a Stack will be include Packages interested together. Finally, the smallest part in ROS architecture Packages have been formed by nodes, messages, services, etc. A detailed definition for Packages will be given in forthcoming subsections.

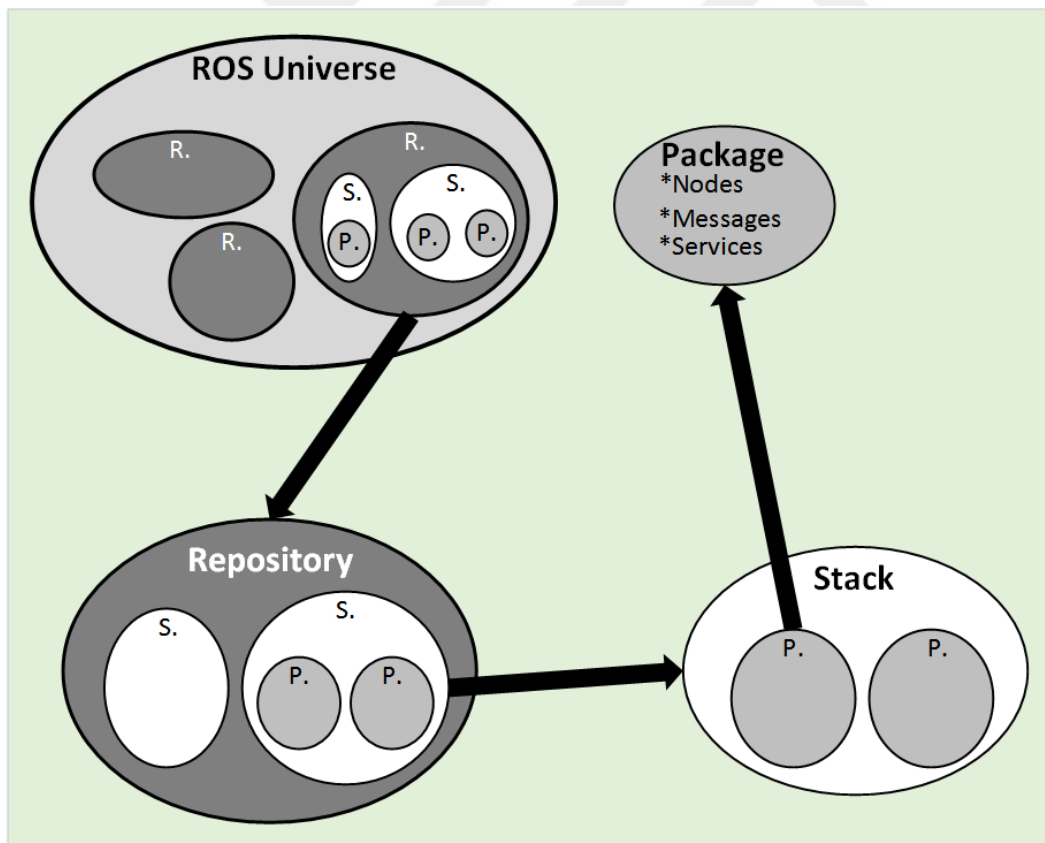


Figure 4.2 : The ROS architecture

Figure 4.2 represents the ROS architecture system visually. It could be clearly indicated that the ROS Universe is a collection of Repositories, a Repository is a

collection of Stacks, a Stack is a collection of Packages, and a Package contains nodes, messages and services.

In the forthcoming subsections ROS Nodes, ROS Topics, ROS Messages, ROS Services and ROS Packages have been explained detailed.

4.1.1.1 ROS nodes

ROS nodes are executable parts in ROS. These executables have been used for process various tasks. Topics and services are other ROS components in which nodes are communicated with messages. Every node must have a unique name, since it is necessary to prevent any communication error. Nodes send and receive messages by their names.

By creating a node there are programming languages available. C++ which uses the `roscpp` library and Python which uses the `rospy` library. Python is a much simpler language than C++; however, for a quick response C++ is required.

rostopic is one of the ROS tools which is used for getting informations about nodes and it also send commands to nodes. Detailed information about *rostopic* have been given in section 4.2.1.2.

One of the most powerful feature of ROS is giving the opportunity to change the parameters or name of a topic without recompiling the code and this provides a more fluent and faster process.

4.1.1.2 ROS topics

ROS topics are used for data transformation by nodes. There is no limitation in subscription for ROS topics. To subscribe any data from a topic it is necessary to have the same message type with it. There is no possibility to be a subscriber to a topic if the message types are not the same.

ROS topics transfer the data with two possible ways, TCPROS and UDPROS. TCPROS is a TCP-IP based transportation which uses TCP-IP connection. TCPROS is set default in data transfer. The second transportation way UDPROS is a UDP based transfer system.

Similarly to ROS nodes there is a powerful tool for ROS topics called *rostopic* which is used to show informations and give commands to ROS topics. Detailed information about *rostopic* have been given in section 4.2.1.2.

4.1.1.3 ROS messages

Messages are published by topics to transport the data from a node to another one. Messages mostly have a standard but there is also an opportunity to define a specific message type.

rosmmsg is a ROS tool to get information about messages. The usage of *rosmmsg* is as following:

```
rosmmsg show
```

This command is used to display the message.

```
rosmmsg list
```

This command lists all available messages.

```
rosmmsg package
```

This command used to show all the messages in the given package.

```
rosmmsg packages
```

This command gives all the packages which contain the message indicated.

```
rosmmsg users
```

This command used to list code files with the given type of message.

4.1.1.4 ROS services

Services are doing the same things; however, there is a difference between them. Services are also used in communication of nodes. To receive and reply data topics are not adequate and the need for services shows up.

There are two tools available for ROS services, *rossrv* and *rosservice*. *rossrv* is used to get information about services and is similar like *rosmmsg*. *rosservice* allows extra commands which are not included in *rossrv*. The usage of *rosservice* is as following:

```
rosservice call /service args
```

This command is used to list the services with the given arguments.

```
rosservice find msg-type
```

This command is used to find services with the given message type.

```
rosservice info /service
```

This command shows the informations about the service.

```
rosservice list
```

This command prints the active services.

```
rosservice type /service
```

This command is used to show the service type.

4.1.1.5 ROS packages

Package is the smallest part in ROS architecture. A package can be in different sizes by containing one ore more than one file inside.

In ROS universe there are countless ROS packages available to use. These packages could be defined as softwares of ROS. Mostly packages contain nodes which are executable codes and are used to do some tasks. *turtlesim* is a package which is used mostly in beginner tutorials of ROS. For example, *turtlesim_node* is an executable node inside the *turtlesim* package.

4.1.2 Robots using ROS

Over the past years the number of robotic platforms compatible with ROS have been increased significantly. More than 100 robot hardware platforms have the capability to run ROS [31]. This huge number of variety proves that ROS is one of the commonly used software for robotic systems and the significance for developers in robotic community is non-ignorable.

For a detailed categorisation the robot hardware platforms have been classified under the following ten classes: Mobile manipulators, mobile robots, manipulators, autonomous cars, social robots, humanoids, unmanned aerial vehicles (UAVs), autonomous underwater vehicles (AUVs), unmanned surface vessels (USVs) and others. The detailed list for the robots using ROS is available in the following reference [31].

4.1.3 Installation of ROS step by step

Under this section the first step has been explained for users who start to use ROS. The first step for any new software is to make the installation. Installation of ROS has been explained in this part, but before start to the installation it is important to give the crucial preliminary informations to start to the installation. There are some beta versions of experimental ROS on Windows; however, ROS is officially supported on Ubuntu. Anyone who has aimed to develop robotic applications for ROS must prefer Ubuntu as operating system on their pc. Ubuntu is an open source Linux operating system. According to Linux, Ubuntu is the most popular distribution used for Linux. [32]

Approximately every 6 months the developers of Ubuntu publish a new version for Ubuntu, and similarly the developers of ROS announce an improved edition for ROS more or less every year. Since beginning 24 releases have been done for Ubuntu. In a two year period in April a Long Term Support (LTS) version is released for Ubuntu and will be supported approximately for 5 years, which means LTS users will get the updates approximately for 5 years (... , 10.04, 12.04, 14.04, 16.04). The first number is the release year and the second number is the release month. In the other hand in October a standard version is released for Ubuntu (... , 14.10,15.04,15.10,16.10). In this versions the support is mostly 1 year. There are just a few improvements for the graphical user interface (GUI) between x.04 and x.10 versions. For a better experience it is logical to upgrade the version from x.04 to x.10; nonetheless, an upgrade for a LTS version cause to lose the support.

Likewise, there are 10 different ROS distributions available. It is not possible to install any ROS version to any Ubuntu version. Every ROS distribution is compatible with the convenient Ubuntu release. Table 4.1 shows the ROS distro's release dates, and coherent Ubuntu version and the bold Ubuntu versions represents the LTS versions [33].

Anyone who is ambitious to work with ROS needs to know the required Ubuntu releases for the ROS versions. Table 4.1 indicates this necessary informations and must be examined before start to any installation. The ROS version used in this thesis

Table 4.1 : Version history for ROS releases.

Distro	Release Date	Ubuntu Version
Kinetic	May 23, 2016	15.10, 16.04 , 16.10
Jade	May 23, 2015	14.04 , 14.10, 15.04
Indigo	July 22, 2014	13.10, 14.04
Hydro	September 4, 2013	12.04 , 12.10, 13.04
Groovy	December 31, 2012	11.10, 12.04 , 12.10
Fuerte	April 23, 2012	10.04 , 11.10, 12.04
Electric	August 30, 2011	10.04 , 10.10, 11.04, 11.10
Diamondback	March 2, 2011	10.04 , 10.10, 11.04
C Turtle	August 2, 2010	9.04, 9.10, 10.04 , 10.10
Box Turtle	March 2, 2010	8.04 , 9.04, 9.10, 10.04

is Indigo, and is installed on Ubuntu 14.04 Trusty Tahr LTS. The instructions given in this thesis are for the mentioned ROS and Ubuntu versions.

Ubuntu 13.10 or 14.04 must be installed before start to the installation of ROS Indigo. Ubuntu and also ROS are terminal based programs, so in installation it is necessary to use the terminal. ROS Indigo could be installed by following these steps [34].

First of all when installing something in Ubuntu it is necessary to set up your computer for the repository:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Next you have to set up your keys:

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net --recv-key 0xB01FA116
```

*Be sure that your pc is up to date. This code will just load the updates for your debian; however, if you write **upgrade** instead of **update** this will change your existing debian and may cause to problems.:*

```
sudo apt-get update
```

Make a desktop-full installation of ROS Indigo:

```
sudo apt-get install ros-indigo-desktop-full
```

*For installing system dependencies to use ROS it is necessary to initialize **rosdep**:*

```
sudo rosdep init
```

```
rosdep update
```

When launching a new shell it is necessary to inform the system about the ROS environment variables; however, it is possible to add the ROS environment variables automatically. This saves us to add the ROS environment variables to bash session in every launch of the shell:

```
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
source ~/.bashrc
source /opt/ros/indigo/setup.bash
```

To install **rosinstall** which is useful for download source trees for ROS packages:

```
sudo apt-get install python-rosinstall
```

For not getting any problems by using ROS packages:

```
printenv | grep ROS
source /opt/ros/indigo/setup.bash
```

Finally a workspace must be created for ROS. Every file will be generated into this workspace. This will help us to obtain a tidy environment for the file manager system:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
cd ..
catkin_make
```

The **catkin_init_workspace** command is necessary to build the workspace. After creating any package it is a necessity to run **catkin_init_workspace** under the path catkin_ws/src. To change your path run **cd ~/catkin_ws/src**. To finalize the preparation of the workspace for a new package **catkin_make** must be run under catkin_ws path.

After **catkin_make** new setup.*sh files have been created. This new change must be sourced:

```
source devel/setup.bash
```

To being sure that the workspace have been created correctly it must be checked out:

```
echo $ROS_PACKAGE_PATH
```

The output of the code above must be following if the workspace have been created correctly:

```
/home/<yourUserName>/catkin_ws/src:/opt/ros/indigo/share:/opt/ros/indigo/stacks
```

Of course *<yourUserName>* part will be the preset name that have been defined by the user at the beginning of installation of Ubuntu. If all of these steps have been completed faultless then the installation is done completely.

4.2 A Short Tutorial of ROS

This section is prepared to show what ROS is capable. Short tutorials and advises have been given for different type of users. The subsections of this section are for users who want to use ROS and for users who want to use something they are familiar instead of ROS. The aim of this thesis is not directly tutor readers about ROS; however, it has been preferred to inform with short tutorials to enlighten those who are interested in.

4.2.1 For users who want to use ROS

It is very normal that anyone who will use ROS at the first time will be confused. As ROS is Robot Operating System it is possibly that the user expect an operating system like Windows or Mac OS with a advanced graphical user interface; nevertheless, ROS is a terminal based without any graphical user interface operating system. As been terminal based and mostly interested with Ubuntu, Linux experience is necessary for a comfortable usage of ROS.

The user profile have been categorized to three different categories which are the users unfamiliar to both Ubuntu and ROS, the users familiar to Ubuntu but unfamiliar to ROS and the users familiar to both Ubuntu and ROS. In the subsections of this section tutorials for these three user profiles have been explained.

4.2.1.1 Users unfamiliar to ubuntu and ROS

Users who have no experience about Ubuntu should not directly start to learn ROS. It is very difficult to follow a way like this. In almost every tutorial on the web and books it has been supposed that the reader has some basic Ubuntu knowledge. Under this assumption not everything has been explained detailed, since it is basic Ubuntu knowledge. Anyone who have decided to learn ROS must ask himself or herself about

basic Ubuntu knowledge. If the answer is no the most wisely move is to learn the basics of ROS.

There are many tutorials available on the web and also books for different level of knowledge. These books [35], [36], [37], [38] and [39] in reference list could be preferred as source to learn Ubuntu.

As it is necessary to know the basic commands for Ubuntu to understand ROS, the basics of Ubuntu have been given shortly in Appendix A.1; besides, there are thousands of codes available in Ubuntu; however, these key commands are sufficient right now. As it has been mentioned at the beginning of the section it is important to have the basic knowledge of Ubuntu to continue learning ROS. It is very important to know the Ubuntu basics to move forward in ROS.

4.2.1.2 Users familiar to ubuntu but unfamiliar to ROS

The users who are familiar to Ubuntu are ready to learn ROS. There are some specific ROS commands, but mostly the code syntax of ROS is similarly to Ubuntu and understanding ROS would not a big deal for users use Ubuntu.

As it has been mentioned at the previous section that there are some basic Ubuntu commands similarly there are some basic commands for ROS and these commands will be used every time in ROS. These commands and definitions are given in Appendix A.2.

As a beginning to ROS, these giving commands are basically important and used mostly. This section is prepared as a cheat sheet for newbies in ROS.

4.2.1.3 Users familiar to ubuntu and ROS

Users in this category can easily adapt to ROS environment. After being familiar to the ROS environment anyone could start to work with the ROS tutorials. The best way is to start with the core ROS tutorials [40]. The core ROS tutorials are under two level for beginner and intermediate experience.

Anyone who starts to ROS must complete the beginner level tutorials. For achieving any success the beginner level tutorials must be completed at least five times and it would be very helpful to try alternative commands for the same output. This will help

the users to improve the tutorials and if there are any errors it will be possible to detect them.

After this point it is completely the users choice which way to follow. The best way to follow could be to work with other ROS libraries to improve the ROS knowledge. If there are any hardware like mobile robots and sensors available to use, writing packages or searching for working packages is a good way to follow. There is also a possibility to work with simulations. Gazebo is a powerful simulation tool for ROS users to generate real like environments with all physical laws.

4.2.2 For users who want to use something familiar instead of ROS

There is an alternative available for users who do not prefer to use ROS directly, which is Matlab Robotics System Toolbox. This toolbox provides algorithms for robotics applications. One of the biggest opportunities of Robotic System Toolbox is providing an interface between Matlab and ROS. This means that anyone who feels more comfortable in Matlab environment has the possibility to take advantage of ROS and ROS based environments and simulators like Gzebo and Rviz. This toolbox also supports C++ code generation which means ROS nodes can be generated by Matlab codes or Simulink block diagrams. Although all of these benefits the toolbox does not support to convert ROS nodes into mfiles. Anyone who is interested about Matlab Robotics System Toolbox should visit [41] for more detailed information about the toolbox.

4.3 ROS Simulators

Working with real robots is mostly more difficult than the estimated. There are numerous reasons to make an assumption like that. The most important reason is the economic burden for a complex and high technological system. There is no way for a researcher to work with a Mars robot if he/she is not an employee in NASA, since these type of a robot can not be afforded by personal funds. Similarly the requirement to a convenient research environment and storage of the robot is an issue which must be solved. Additionally the inability of battery and the recharge time of the battery is also an annoying problem. These challenging problems can cause to a drop in the

motivation level; however, all of these problems can be overcome under favour of simulations.

The simulations provide the researchers to change the environment as their wish, and by following this way any type of robot can be simulated in any type of environment. Very expensive sensors and actuators can be attached easily to the simulation and there is also a possibility to modify the dynamic models of these components. Moreover, datasets can be used as real sensor data in simulations for robotic researches.

There are many simulators, which provides low cost robotic platforms with a very simple interface, for the ROS universe. Among the different simulators in the forthcoming subsections Gazebo and Rviz have been analysed detailed.

4.3.1 Gazebo

Gazebo has been developed as a part of a project named "Player" as an open source 3D simulator. The Player project has been founded in University of Southern California by Brian Gerkey, Richard Vaughan and Andrew Howard. Nate Koenig and Andrew Howard developed Gazebo in 2002. After 9 years from the beginning, in 2011 the developer team continued to develop Gazebo independent from the Player project. Gazebo has been became a pioneer for the development of ROS and the PR2 robot by Willow Garage. The company Open Source Robotics Foundation (OSRF), which has been founded by Brian Gerkey and Nate Koenig, is recently the developer of Gazebo.

Gazebo is not only used for generating environments with real physics called worlds, but also it provides a platform to simulate real sensor data like depth camera or laser scanner. One of the best opportunities of Gazebo is the 3D environments with real like physics such as momentum, inertia, acceleration, and friction. Gazebo also offers to generate a prototype robot by generating it in the desires of the researchers or using directly the one-to-one model of an existing robot.

There are two different versions of Gazebo available. Gazebo 7.x is the newest version of Gazebo and must be installed separately from ROS. 7.x can not be installed in a full ROS installation default. However, Gazebo 2.x will be installed in full ROS installation default. If the installation of ROS was not performed as a full installation, then Gazebo must be also installed.

If the installation of ROS has been executed as a full installation Gazebo can be started by calling the Gazebo server by the following command:

```
gzserver
```

The Gazebo server will show some start-up messages about Gazebo. These messages cover the version information of Gazebo and also warnings or errors if exist. When everything works correctly the Gazebo client can be called for the GUI by the following command by running it in a new terminal.

```
gzclient
```

There is also a simpler way to start Gazebo by just typing "*gazebo*" into the terminal. This will also do the same thing with just one command. If there is no problem the Gazebo GUI will appear as given in Figure 4.3.

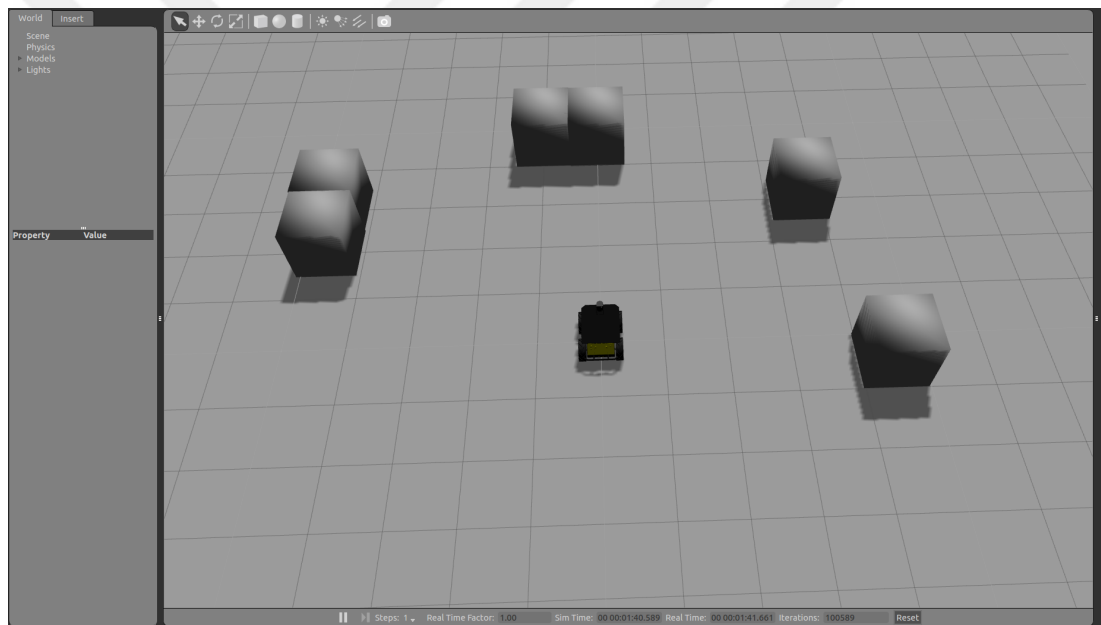


Figure 4.3 : The gazebo GUI start-up screen

In some cases the Gazebo GUI can crash and the window in Figure 4.3 will not appear. In a situation like that it is recommended to close all terminals and call the Gazebo server and client again. If the problem continuous, than an adjustment must be done to solve the problem. Most probably the reason of a problem like this is occurred by the GPU. The user must run some commands to solve the problem in Gazebo GUI.

The physical engine of Gazebo requires high performance in some situations and these peaks in GPU load is the reason of the crashes. If the hardware has the support it can be accelerated for cases like that; however, this process is not available for every GPU.

```
unset LIBGL_ALWAYS_SOFTWARE
```

The command above is used to set the acceleration process. All terminals must be terminated before run this command; therefore, the previous gzserver and gzclient must be terminated by "*ctr+c*" combination. After running the code above the Gazebo GUI must be called again by typing "*gazebo*".

```
export LIBGL_ALWAYS_SOFTWARE=1
```

To turn off the acceleration for decreasing the CPU load, the command above can be executed.

As it has been mentioned there are two versions available for Gazebo. The first version 2.x will be installed in a full ROS installation, on the other hand Gazebo 7.x, which is the newest version of Gazebo for November 2016, needs some extra command for an installation. First of all for installing a newer version of Gazebo, the older versions must be removed. These instructions are given for ROS Indigo.

```
sudo apt-get remove ros-indigo-gazebo*  
sudo apt-get remove ros-hydro-gazebo*  
sudo apt-get remove ros-groovy-gazebo*  
sudo apt-get remove ros-fuerte-gazebo*
```

All the Gazebo files for the following ROS distros have been removed after the commands given above. After removing the older Gazebo files the installation of Gazebo 7.x can be operated. There is a huge possibility for an update in the installation files and instructions, because of this reason the commands for installation are not given. Probably the best way for installation is following the steps from the official website of gazebo [42].

Gazebo provides to generate our own environments called world or load and simulate demo files or other worlds. For load and run any world the necessary packages must be installed into the workspace. The best way to understand Gazebo and its' integration with ROS is complete the official Gazebo tutorials.

4.3.2 Rviz

Rviz is another important simulator for ROS similarly like Gazebo; however, there are some characteristic differences between both simulations. Unlike Gazebo, the interface of Rviz is mostly related with the sensors and their outputs and interested to inform

the user about the sensor outputs. The best source for Rviz is the Rviz User's Guide documented by ROS officially [43].

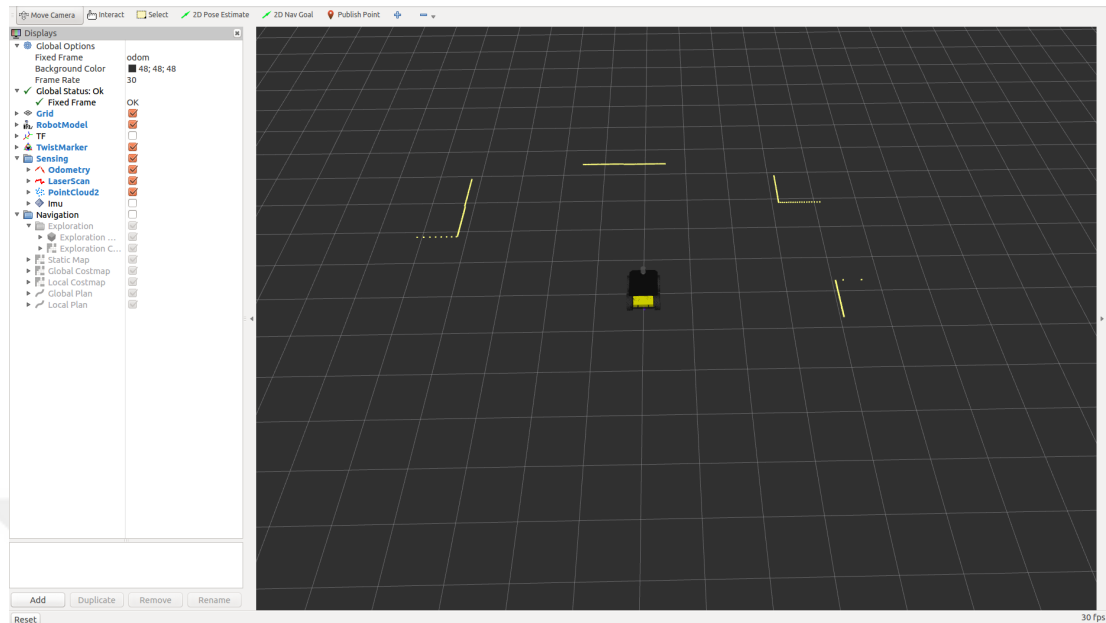


Figure 4.4 : The rviz GUI start-up screen

The Rviz screen at the start-up looks like given in Figure 4.4. The screen at the middle is similar to the GUI in Gazebo the motion of the robot and the data of the sensors like LRF and camera are visualized. In Gazebo a detailed model of the environment is available; nevertheless, in Rviz the picture is what the sensors reads and what the robot sees.

To use Rviz it must be installed on the computer. In a full installation of ROS, Rviz is mostly installed; however, the installation command can be necessary. To install Rviz separately:

```
rosdep install rviz
```

Roscore is necessary to start to Rviz, so the command "roscore" must be run before the beginning. After executing "roscore" Rviz can be started easily by typing "rviz" on a new terminal and run.

```
rviz
```

The Gazebo and Rviz outlines are for the same robotic system and environment system, given in Figure 4.3 and 4.4. On the left side of the Rviz screen there can be sensors

attached on the system and these sensors can be modified by turning on and turning off.

4.4 A Real Time Implementation in ROS

In this section a real time implementation in ROS with Clearpath Husky A200 have been performed. The codes have been run were generated for the ITU Library Robotics Project demonstrations with the name of ikuro. ikuro is the acronym of Itu KUtuphane RObotu which means ITU library robot. The experimental setup has been described in section 7.1; however, a short explanation can be helpful for understanding the real time implementation.

The mobile robot in the setup is Clearpath Husky A200. Hokuyo UTM-30LX has been implemented into the system as 2D LRF. There are two computer in the experimental setup; first one is on the mobile robot and the second one is on the ground station. The commands have been send to Husky A200 from the computer on the ground station.

To send any commands to the mobile robot the ground station must be connected to the AGV. The connection has been provided via wifi. A tutorial for the real time implementation is given in the forthcoming parts of the section.

- First of all a computer with Ubuntu 14.04 is needed and ROS Indigo must be installed on the computer.
- Next the Chrony application must be installed. For installation:

```
sudo apt-get install chrony
```

Chrony is necessary for the system clock synchronization with Network Time Protocol (NTP) servers.

- A catkin_ws must be created for the approaching steps. This step is for the users who have not created a catkin_ws yet.

```
source /opt/ros/indigo/setup.bash
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
cd ..
catkin_make
```

```
source devel/setup.bash
echo $ROS_PACKAGE_PATH
```

If the output of the last command is the same given below, then the workspace have been created successfully:

```
/home/<username>/catkin_ws/src:/opt/ros/indigo/share:/opt/ros/...
... indigo/stacks
```

- Similarly to the creating of catkin workspace, husky_desktop_ws must be created. Most probably husky_desktop_ws have not been created yet for all of the users. This is the same thing by creating catkin_ws, only catkin_ws must be altered with husky_desktop_ws.
- The next step after the creation of husky_desktop_ws is copying the iturobotics repository inside /husky_desktop_ws/src/:

```
cd husky_desktop_ws/src
git init
git remote add origin https://github.com/ituroboticslab/ikuro.git
git fetch
rm -f CMakeLists.txt
git checkout -t origin/master
```

Inside the /husky_desktop_ws/src/ location folders named "ikuro_control", "ikuro_description", "ikuro_msgs" and "ikuro_navigation" must be created. Also files named "CMakeLists.txt", "LICENCE" and "README.md" must be generated. If all of these have been done then this step has been passed successfully.

- The package named husky_viz is necessary for continue. The easiest way for installing any missing package is to use Synaptic Package Manager. If it is not installed in the computer it can be installed by the following command:

```
sudo apt-get install synaptic
```

- To finalize the necessities for the connection of the ground station to the mobile robot, the following lines must be pasted inside .bashrc file. To edit .bashrc file the following command must be entered:

```
gedit ~/.bashrc
```

The given lines must be pasted at the end of the .bashrc file.

```
source /opt/ros/indigo/setup.bash
source /home/<username>/catkin_ws/devel/setup.bash
source /home/<username>/husky_desktop_ws/devel/setup.bash
export EDITOR="gedit"
export ROS_HOSTNAME=$(hostname).local
#export ROS_MASTER_URI=http://$(ROS_HOSTNAME):11311
#export ROS_MASTER_URI=http://roboticslab.local:11311
export ROS_MASTER_URI=http://192.168.1.105:11311
export ROS_IP=192.168.1.104 #desktop ip
```

If all of these steps have been completed the ground station is ready for the connection with the mobile robot. At the beginning the ground station computer must be connected to the wifi network of the mobile robot. The name of the wifi network is "*dd_wrt*" and the password is "*roboticslab2016*".



Figure 4.5 : The USB hub

All of the switches of the USB hub given in Figure 4.5 must be in a closed position. To pass to the next step all the switches must be turned of in a order from left to right. In the next step the Desktop (computer on the ground station) must be connected to Husky (computer on the mobile robot). The following command cause to a pop-up which asks the password of Husky. The password is "*husky123123*".

```
ssh husky@roboticslab.local
```

After this step the terminal screen will be the Husky's terminal. The commands given on the terminal are now send to the mobile robot directly from the ground station. The following commands must be run on the Husky terminal:

```
source ~/husky_indigo_ws/devel/setup.bash
roscore &
roslaunch ikuro_navigation ikuro_navigation.launch
```

In normal conditions after giving the command **roscore**, a new terminal must be opened to give any further commands; however, when a new command opened on the ground station the ssh step is necessary to connect to Husky. "&" solves this problem. After the "roscore &" command pressing enter will let the user to give an extra command on the terminal where roscore runs.

There are many possibilities to run after the set of the mobile robot. Inside the husky_viz package there is a view_robot.launch file available to run. All of this instructions must be performed under the Desktop terminal which is following:

```
roslaunch husky_viz view_robot.launch
```

This command opens a predefined Gazebo file in which the mobile robot and 2D LRF informations are available in real time.

Figure 4.6 represents the output of the previous launch command. On the left side of the GUI sensors can be configured like the 2D LRFs and Microsoft Kinect sensor.

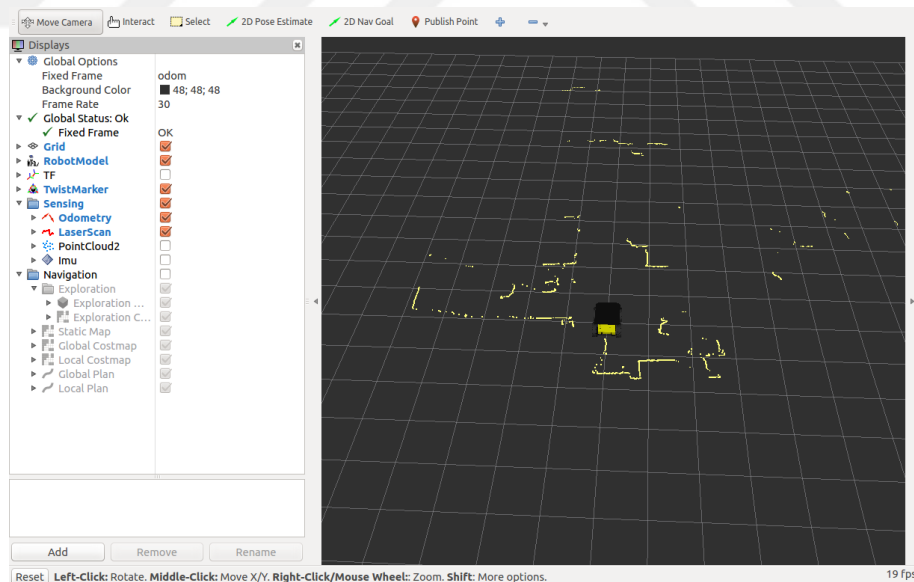


Figure 4.6 : The output of the view_robot.launch file

The yellow dots are the 2D LRF scans and represents the detected obstacles around the robot.

When the navigation package has been activated Figure 4.6 turns into Figure 4.7. In this Figure there are five regions available represented with five different colors. As

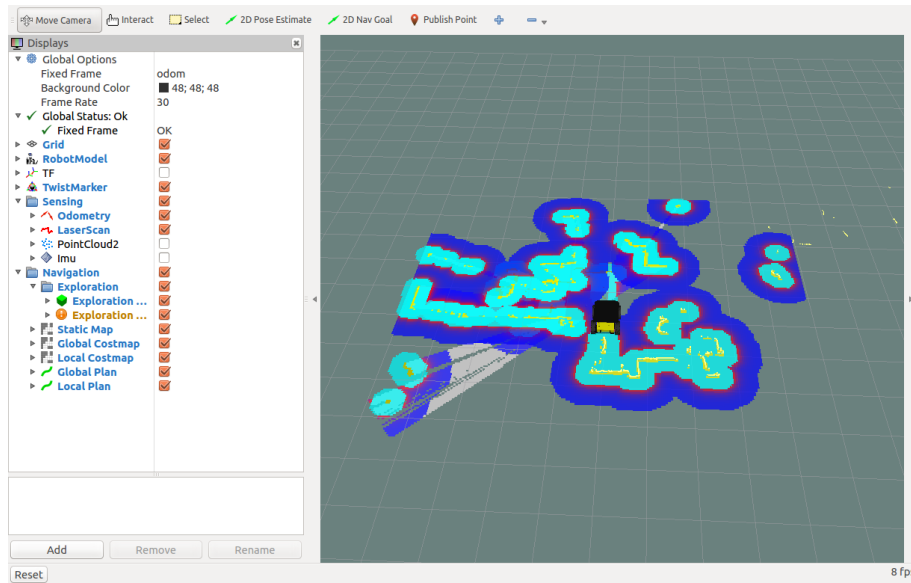


Figure 4.7 : The output after navigation package have been ticked

defined on Figure 4.6 similarly the yellow regions are pure obstacles and must be avoided. The turquoise region is relatively dangerous since this region is too near to the pure obstacles. The navy blue region is relatively secure because this region is far enough from the obstacles for generating a path and follow it. The purple region is a boundary between two relative regions. The grey region represents two things undiscovered region or regions without any obstacle.

5. LOCALIZATION AND MAPPING PROBLEM

This chapter covers the localization and mapping problems, which are one of the most focused problems in mobile robotics research area. For both problems probabilistic methods have been offered to find any solution.

It is necessary to know the basic approaches in localization and mapping problem to go further in Simultaneous Localization and Mapping (SLAM) algorithm in next step which is the main focus point of this thesis.

In the first part of this chapter the localization problem has been described and additionally some localization methods have been also defined. In the second part of the chapter a deeper overview over mapping problem, and in addition to some mapping presentations have been done. The solution for the problems depends on each other like a chicken and egg problem. To generate a map of the environment it is necessary to have the initial localization information and similarly for localization an initial map is demanded.

5.1 Localization Problem

In the most general sense localization is the estimation of robot position according to a given map. Localization is one of the basic problems in robotics and is demanded almost every mobile robotics applications.

The pose of the mobile robot is represented by position of the robot in xy coordinate frame and θ orientation angle. In detailed the pose of the mobile robot is shown as $X_t = (x \ y \ \theta)^T$. In localization problem maps generated in global coordinate system have been transformed into robot's local coordinate system. This transformation is necessary, since maps global coordinate system is independent from the robot's pose. The coordinate transformation of maps global coordinate system into robot's local coordinate system is not only necessary for the robot localization but also for the obstacles locations inside the environment.

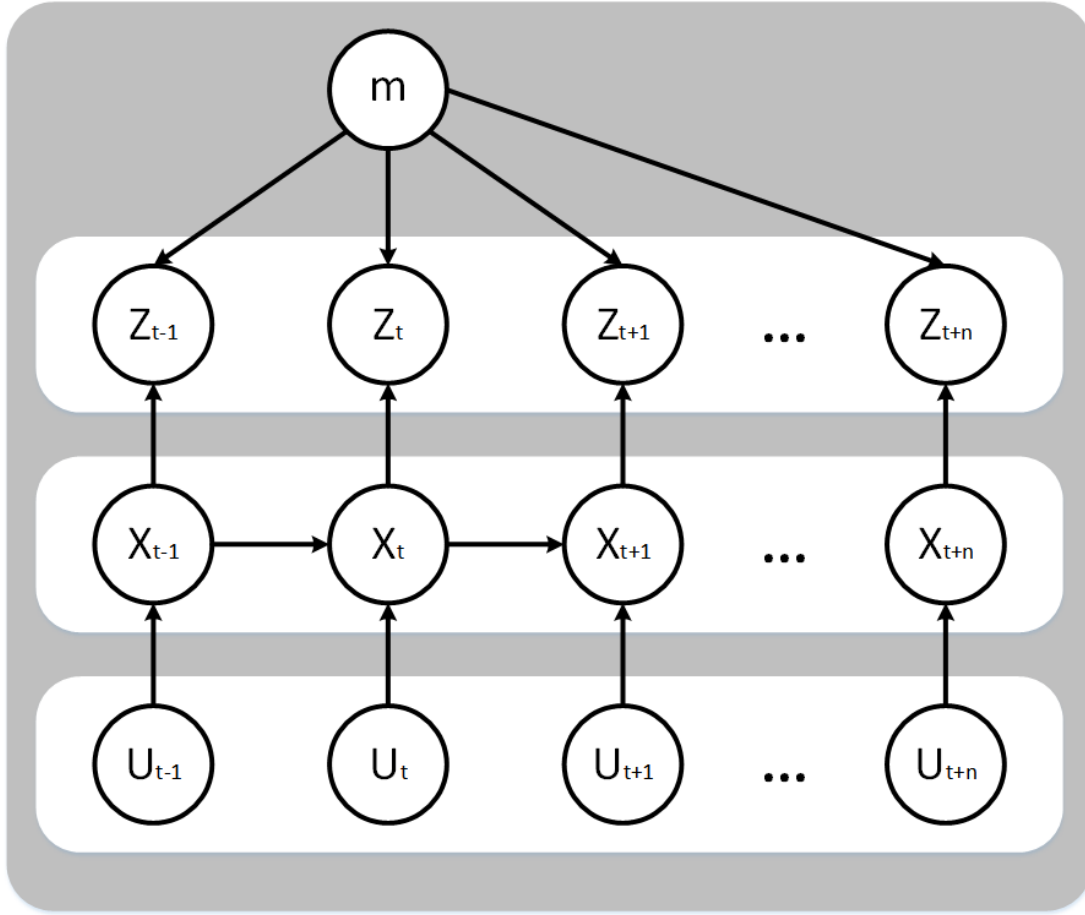


Figure 5.1 : Graphical representation of localization

A graphical representation of Localization is given in Figure 5.1. In this process a map of the environment m is given. Also the measurements Z and the control inputs are known. The only thing that is tried to be defined is the pose of the mobile robot $X_{t-m:t+n} = (x \ y \ \theta)^T$. Among numerous type of sensors the most preferred types are cameras, whit a possibility of monocular, stereo and RGB-D, and LRFs, as 2D and 3D.

The differences in the environment, knowledge about the initial robot pose and number of robots cause to a diversity in localization which has been shown in figure 5.2.

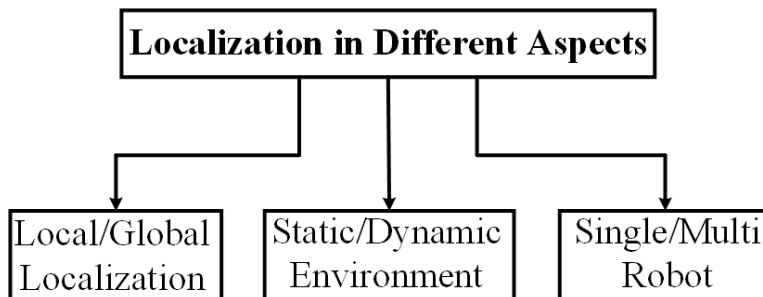


Figure 5.2 : Comparison of localization in different aspects

Localization problems have been occurred by the deficiency of the initial information. These problems have been reviewed under local/global localization. The most common problems are position tracking, global localization and kidnapped robot problem.

For a position tracking the initial robot pose must be known; however, noise and disturbance in the robot movement cause to an uncertainty. Mostly this uncertainty is assumed small; nevertheless, the uncertainty can result as a problem in position tracking. This problem is defined as a local localization problem, because the uncertainty is local.

It has been assumed that the robot pose is known; nonetheless, the known pose is the local pose not the global pose. The knowledge of the robots localization with respect to the global environment is uncertain. The global localization problem also comprises the position tracking problem.

The kidnapped robot problem is actually a different version of the global localization problem. In this problem the robot have been lost in the map. The robot location have been assumed as an incorrect pose and this situation will cause to a misunderstanding in robot localization. The main difference between global localization problem is that in kidnapped robot problem is sure about the pose estimation; however, in global localization problem can not make any estimation about the pose.

The problems encountered in localization can also be analyzed as the structure of the environment as static or dynamic. In a static environment the map does not change over the time. This means that all the obstacles are static and only the pose of the robot changes. Localization in static environments is much easier and efficient than dynamic environment. The reason of this statement is that the map of the environment changes over the time and the pose estimation of the robot with respect to a dynamic map is more difficult than a static map. Most probably the main reason of a dynamic environment is dynamic obstacles and the change over time in the map must be sensed by the sensor measurements.

The number of the mobile robots also effects the complexity of the localization. As might be expected single robot localization is more easier than a multi robot application. In a single robot localization all data has been received by one robot;

however, in a multi robot localization data has been received by different robots and this occurs to a necessity of communications of the robots together.

5.2 Mapping Problem

This section implies the mapping problem in Robotics. The problem is been defined and solutions of this problem have been discussed. Additionally most well known two mapping methods in robotics; grid mapping and feature map have been described.

To gain full autonomous properties into a mobile robot it is important to solve the mapping problem. Modeling the environment by detecting the obstacles and walls is the the main process in mapping. To achieve this goal different type of sensors must be implemented to the system. The most used sensors are cameras, laser, sonar and infrared sensors for range finding.

There is a deep relation between localization and mapping. Both localization and mapping results are necessary to each other. To achieve a map of the environment it is necessary to have the direct localization of the robot, likewise generating the localization information there is also a dependency on map. It is possible to state that localization and mapping problems depends on each other like a chicken and egg problem.

The mapping methods available in robotics literature are mostly implemented by the use of probabilistic techniques. The reason of this approach is that mostly the sensors and robots are modeled for probabilistic environments. There is a general assumption that most noises in sensors and other parts of the system are applied as Gaussian noise, because of this reason to express the uncertainty, probabilistic techniques have been implemented in robotics techniques and mapping is one of them. Probabilistic techniques are the best way to define uncertainties in a complex level.

5.2.1 Grid map

Grid maps are one of the simplest maps used for mobile robotics. The reason of the popularity is the simplicity of the process to generate the map by the sensor data. A grid map is divided into many grids and each of them represents a part of the map. If

the is any obstacle inside the grid, it is defined as an occupied grid; otherwise, the grid will be free. According to these two possibilities a grid map is been generated.

There is a possibility to generate a grid map in three dimensions; however, in ground vehicle applications if the mobile robots mostly makes planar motions, it is possible to indicate that the change in altitude can be neglected. This neglect will cause to a decrease in the dimensions of the map and the grids will be generated in the XY plane. In most unmanned ground vehicle mapping applications grid maps have been generated as two dimensional.

A grid map can be taken as a reference for a metric map which is one of the biggest advantages of the grid map. The resolution of the map is been decided by the size of the grids. A map generated by small grids will create a more detailed map. These type of maps are also classified as metric maps, since the data received by the sensors are metrical data.

Having a very simple process algorithm and being a metric map are the strongest characteristics of grid maps; on the other hand, these properties requires huge storage for these details and is the biggest disadvantage. For two dimensional grid maps the required data storage size is increased quadratic and for three dimensional maps the increment will be cubic. Because of this reason, grid mapping is more efficient for small mapping implementations. There is a possibility to split the environment into smaller parts to reduce the complexity of the map, and this type of a map can be classified as a hybrid map, a mixture of a metric and topological map.

5.2.2 Feature map

This type of maps focus mostly on detecting some special objects defined beforehand. These objects can be some specific things inside the environment like a door or a window for an indoor scenario or a tree, traffic signal or a building for an outdoor environment. These elements with a meaning are called feature or landmark and the maps generated for this purpose are named feature maps or landmark maps.

Differently from grid maps, feature maps do not contain detailed information of the environment in metrics, instead of this some predefined landmarks are used to define the map. The uncertainty of the robots localization rises until a landmark have been

detected, and after a landmark have been sensed by the sensor, the uncertainty have been reprocessed and a more accurate localization have been obtained.

Any type of a sensor is used to create a feature based map; nevertheless, image processing based sensors are mostly preferred for localization and SLAM algorithms.

There are three issues by selecting landmarks to generate a map with highest efficiency. First of all, the landmarks must be unique to each others, since they must be different from each other and specific for one coordinate inside the map. Secondly, it is important to assign a static object as a landmark, because any movement in the landmark will change the map and the results of the algorithms will be incorrect. For instance, a car could be assigned as a landmark; however, if it will change its' position this will change the map. Finally, for vision based algorithms, a landmark with a variable appearance is not an appropriate landmark. For example a tree is not a suitable landmark for a map which will be used four seasons, since the appearance of the tree will change over the time.

6. SLAM PROBLEM AND IMPLEMENTATION

This chapter covers the definition of SLAM problem and the SLAM techniques used in experimental works have been explained. Chapter 6 is the last technical chapter and contains the informations of the previous technical chapters 3, 4 and 5. The definition of the SLAM problem is given in section 6.1. Likewise, the SLAM techniques available in ROS have been explained in section 6.2. These techniques have been performed in chapter 7, the experimental works.

6.1 SLAM Under Uncertainty

SLAM covers the localization problem and the mapping problem in which both algorithms have been processed cyclical. This situation is mostly defined as a chicken and egg problem, because both algorithms depends on each other. As it has been explained in the previous chapter, the mobile robot needs an available map of the environment to generate the location information; similarly, the mobile robot needs the initial location data to generate the map of the environment. Being so complex and containing both mapping and localization problems makes the SLAM problem more difficult than the mapping and localization problems.

Under normal circumstances SLAM is not preferred to process if one of the map or localization information is available; however, in conditions where both the map and localization are not available SLAM techniques must be implemented into the system [44].

In most robotic applications, algorithms for SLAM problem have been tried to solved by particle filters and EKF filters which have been described in section 3.1 Estimation Under Uncertainty. There are also other solution methods applied for the SLAM problem; notwithstanding, the particle filter and EKF filter approaches are the most popular ones [7].

The definition of the SLAM problem is actually clear, the location of the robot x and the map of the environment m have been estimated according to the sensor measurements z . The algorithm of the SLAM technique depends on the filtering technique and the most popular approach for the SLAM problem is particle filter and extended kalman filter based methods [44].

SLAM is not just a simple algorithm, it is actually an idea and there are many different ways and approaches to solve the SLAM problem. SLAM is not just one step and these steps can be generated by different algorithms [45]. Some of the main steps of SLAM is given in figure 6.1.

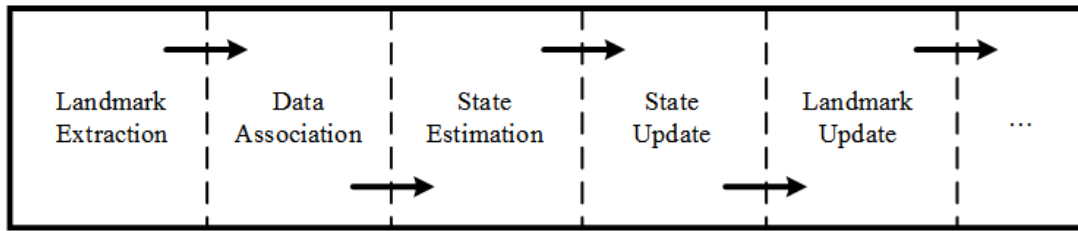


Figure 6.1 : Representation of SLAM

Mostly in robotics the position information have been obtained by the odometry results; however, odometry can not provide a faultless data. Therefore, there is a necessity to make some extra measurements to achieve logical position information. Usually LRFs or cameras have been used to get correct position information of the mobile robot. This process can also be called as landmark extraction which is the first step of SLAM. In landmark extraction there are many different methods available which are related to the landmarks and sensors used on the robot [45].

In data association the landmarks which are observed by the sensor measurements have been matched. This process can also be defined as a reobservation of the landmarks. For instance let us enter to a room and realize a special object which is different from the other ones inside the room like a plant. Now let us leave the room and later we will return to the room. If the same plant, which has been seen previously, have been realized the room will be realized by ourself. For a human it seems like a simple task; however, in data association it is not so easy. There are many important properties of landmarks which are crucial for a successfully process. First of all a landmark must be easy to realize. Additionally a landmark must be identical around the area. A traffic lamp seems like a bad landmark, since almost in every intersection there is one and this

will cause to a confusion to the mobile robot. Finally landmarks should be chosen from the static objects around, since if the landmark position will change in the map this will cause to a big confusion for the mobile robot. In data association the landmarks which have been seen multiple time will be passed to lower the database size.

For the estimation and update of the state and landmarks filter algorithms have been used, mostly extended kalman filter and particle filters. These filter algorithms have been consisted from two parts, the estimation and the update parts. These filter algorithms have been detailed described in chapter 3.

6.2 SLAM Techniques Available in ROS

The SLAM techniques available in the literature can be classified for two different classes. The first class is about the sensor dependency and the second class is about the software dependency. This thesis focuses on 2D LRF based SLAM techniques available in ROS. In openslam.org [7] and wiki.ros.org [46] the opensource SLAM techniques for different type of sensors and also available in ROS have been listed. Additionally the documentations assist the users for the implementation. Gmapping and Hector SLAM have been selected for the experimental works in chapter 7. These two SLAM methods have been chosen because of their high performance and being one of the most preferred SLAM algorithms in mobile robotic community.

6.2.1 Gmapping

Gmapping has been developed by Grisetti, Stachniss and Burgard in 2007. Gmapping is a Rao-Blackwellized particle filter (RBPF) based SLAM algorithm for 2D LRFs. In this research RBPFs have been used to solve the SLAM problem in a view of particle filter method. Adaptive techniques have been used for the reduction of the number of particles in RBPF algorithm for generating a grid map. The gmapping algorithm generates a 2D occupancy grid map which is like a buildings floorplan. To activate gmapping on a mobile robot an odometry and a LRF sensor must be equipped on the mobile robot [12].

RBPFs have been first time invented by Murphy [47] and Doucet [48] and adopted into the solution of SLAM problem. The biggest disadvantage of RBPF is the complexity of the required number of particles. Grisetti [12] applied some modifications into

the RBPF algorithm to solve the SLAM problem with a more effective way. In RBPF sensor observations and odometry readings have been used in the algorithm. The process have been constituted by four steps: Sampling, importance weighting, resampling and map estimation [12].

The gmapping algorithm has been not directly created for ROS; however, the gmapping package has been generated by Rabaud [49] first for ROS Electric and the last version has been updated for ROS Kinect.

6.2.2 Hector SLAM

The biggest difference of Hector SLAM from gmapping is the unnecessary to a odometry. This means the SLAM algorithm works without an odometry data. This opportunity makes it possible to implement the SLAM algorithm into unmanned aerial vehicles like quadrotors. As this SLAM algorithm is LRF based, a LRF is necessary to implement this method [50].

Hector SLAM has been actually developed for 3D motion estimation; notwithstanding, there is a possibility to implement this method for 2D SLAM applications. Furthermore, Hector SLAM has been developed also for ROS simultaneously. The motivation of this development was to produce a gmapping like SLAM algorithm available for mobile robots without the ability of odometry [50].

Similarly to gmapping, occupancy grid map has been used in hector SLAM for 2D SLAM applications. Nevertheless, the approach for SLAM is based on an Extended Kalman Filter (EKF) instead of RBPF [50].

The next chapter includes all the experimental works realized for the thesis. The main motivation is to show the effect of TRAP configuration for improving the results of a SLAM technique [55]. Additionally to the main motivation this thesis also focuses on comparing gmapping and Hector SLAM which are available in ROS.

7. EXPERIMENTAL WORKS

All the previous chapters in this thesis, especially chapter 4 and chapter 6 covers the basis of thesis and chapter 7. Experimental works for this thesis have been performed in this chapter. The comparison of ROS based SLAM techniques, and also a benchmarking of a sensor position configuration have been contained in the experimental works. The sensor configuration has been named as Transversal Position (TRAP) configuration and more detailed explanations have been given in section 7.2.

The experimental setup have been defined in the first section of this chapter. The hardware and software characteristics of the system have been given in this section additionally. The next section is about the TRAP configuration. A detailed definition have been done in this section. The forthcoming three sections are about the experiments. The experiments have been performed in three parts. The classification have been done for a better presentation. In the first section some ROS based SLAM techniques have been compared together. In the second part the configurations for the same ROS based SLAM techniques have been compared for TRAP configuration. The third part is generated for a comparison of the traditional sensor position configuration with TRAP configuration.

7.1 Experimental Setup

The experimental setup has been consisted basically by two main components, a mobile robot and a sensor. Of course the list of equipment is more than two; nevertheless, these two hardware are the main parts of the experimental setup. An Autonomous Guided Vehicle (AGV) have been applied as the mobile robot platform, since this thesis is focused on ground applications of mobile robots. The sensor attached on the mobile robot is a 2D Laser Range Finder (LRF). In addition to these hardwares the experimental setup has been constituted by large and small other equipments. The full list of the hardware components are given in Table 7.1.

Table 7.1 : Hardware equipments of the experimental setup.

Part	Type of Hardware	Model of Hardware	Quantity
A	Autonomous Guided Vehicle	Clearpath Husky A200	1
B	2D Laser Range Finder	Hokuyo UTM-30LX	2
C	RGB-D Camera	Microsoft Kinect	1
D	Stereo Camera	-	1
E	Inertia Measurement Unit	Xsense MTi-10	1
F	Router	Linksys	1
G	WiFi Antenna	-	1
H	Laptop	Dell Latitude E4200	1
I	On/Off Switch USB Hub	-	1
J	Power Supply	Xantrex	1
K	24V DC to 220V AC Inverter	-	1

A detailed view of the experimental setup is given in Figure 7.1; moreover, components on the hardware equipment list given in Table 7.1 have been shown over Figure 7.1.



Figure 7.1 : Experimental setup

All of the components except the inertia measurement unit (IMU) and DC to AC converter can be seen in Figure 7.1 which have been represented by the capitals in Table 7.1. The IMU is positioned inside the mobile robot approximately in the center inside the storage area of Husky A200. Also the DC to AC converter is attached inside the storage area which is used to provide power for the laptop.

Husky A200 is an Autonomous Guided Vehicle which has been produced by Clearpath Robotics, a Canadian mobile robotic manufacturer. The CAD and technical drawings of Husky A200 has been given in Figure 7.2. Husky A200 has a **length** of 990mm, a

width of 670mm and a **height** of 390mm out-to-out. There is also a user storage area in the center of the robot with a **length** of 296mm, a **width** of 411mm and a **height** of 155mm. The **wheel size** is 330mm [51] and [52]. This storage area has been used for placing all the cables of the hardware for a compact experiment setup.

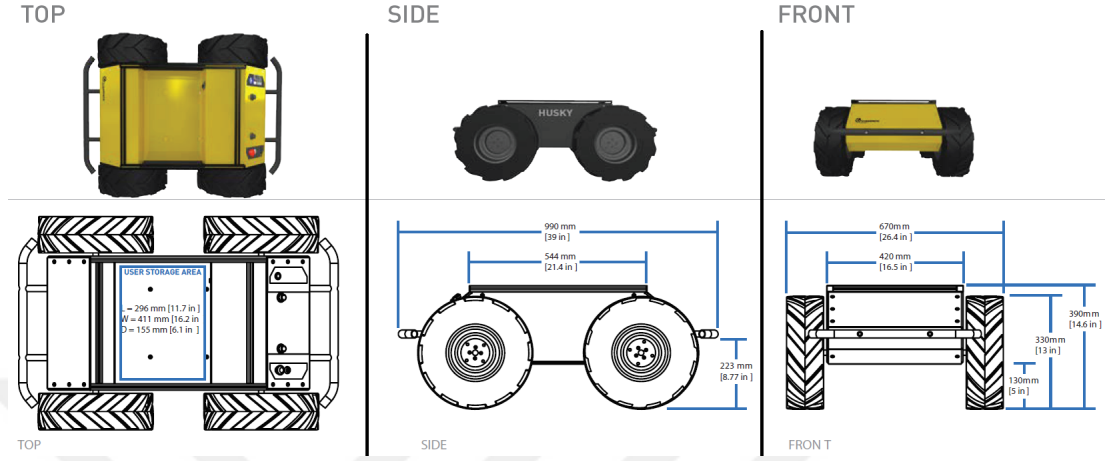


Figure 7.2 : The CAD and technical drawing of Husky A200 [51] and [52].

The rugged all terrain tires provides to use Husky A200 in various environments from indoor applications to harsh outdoor environments. Having the support of an open source platform ROS is also an advantage of Husky A200 to prefer by researchers in robotic community. Besides Husky A200 provides 5/12/24V outputs for attaching extra equipments, for instance sensors. The **weight** of Husky A200 is 50kg and offers a **maximum payload** of 75kg. The **maximum speed** is 1m/s (3.6km/h) and the **maximum climb grade** is 45°. The **battery** is a sealed lead acid type with 24V and 20Ah capacity. The **control modes** are direct voltage, wheel speed and kinematic velocity. The **feedback** options of the system are battery voltage, motor currents, wheel odometry and control system output [51] and [52].

The second main component of the experiment setup is the LRF sensor Hokuyo UTM-30LX. This sensor has been produced by a Japanese manufacturer Hokuyo Automatic Co., Ltd. UTM-30LX is one of the most preferred LRF models of Hokuyo among other LRFs produced by Hokuyo with different specifications for various applications. In Figure 7.3 the 2D LRF used on the experimental setup and the technical drawing of the 2D LRF [53] is given.

Hokuyo UTM-30LX is mostly preferred for mobile robots with a lower payload and small size because of the compact form and lightness of the sensor. The **weight** is

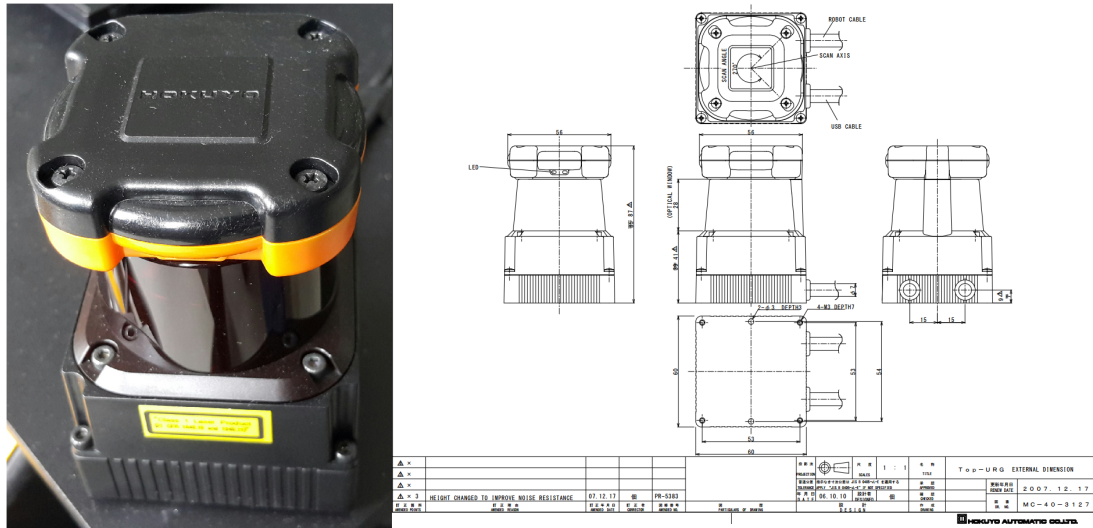


Figure 7.3 : Hokuyo UTM-30LX with technical drawing [53].

210g and the external dimensions are 60mm for **width**, 60mm for **depth** and 87mm for **height**. The **wavelength** of the laser source is 905nm. The **detection range** is between 0.1m to 30m with a **scan angle** of 270° and the **angular resolution** is 0.25°. The **accuracy** is ±30mm for 0.1m to 10m and ±50mm for 10m to 30m. The **scan period** for a cycle is 25ms. The **supply voltage** is 12VDC ± %10 and **supply current** is 0.7A(max.1A) [54].

7.2 TRAP Configuration

TRAP configuration is a new position configuration for 2D LRF. TRAP is the acronym of Transversal Positioned. In this configuration a pair of 2D LRFs with 270° scanning range are replaced with exterior corresponding angles to the front left/right and rear right/left of the vehicle and offers a 360° angle of field of view for the mobile robot without any blind region. The TRAP configuration ensures the AGV to identify anything inside the scanning area instantaneously [55].

The TRAP configuration is not only generated for 2D LRFs, but also generated for sensors in different variations such as monocular or stereo cameras, 2D or 3D LRFs, ultrasonic sensors, and radar sensors. However, the study in this thesis covers the TRAP configuration only for LRF sensors and the type of the LRF is 2D. Comparing 2D and 3D LRFs, in most applications of AGV's, 2D LRFs are preferred as sensors. The reason of this decision is the high price and the complexity in data fusion of a 3D LRF against 2D LRF. The same reasons motivated the experiments to 2D LRF [55].

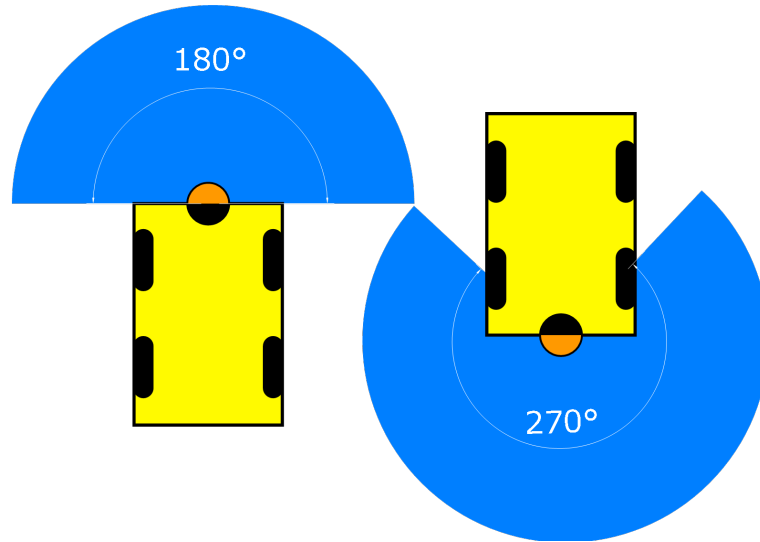


Figure 7.4 : Traditional 2D LRF configuration for 180° and 270° .

The most common configuration for LRF sensor applications is using one 2D LRF and positioning it in front of the mobile robot centered given in Figure 7.4; albeit, there are still some exceptions. The configuration of a 2D LRF positioned in front of the mobile robot offers a frontal and partially horizontal view for the AGV. Data received by 2D LRF is beneficial for modeling the environment to bring autonomous guidance to the mobile robot. Mapping, navigation, and path planning algorithms will be developed and operated on the mobile robot for the purpose of unmanned orientation.

A configuration of frontal centered position of a 2D LRF may cause some problems for using the maximum ranges of angular scanning. The scanning angle of LRFs preferred in academic and industrial researches are mostly 180° and 270° . Scanning area of a 180° LRF seems to be normal; nevertheless, the appearance of a 270° scan is most probably problematically. Some of the equipments on the mobile robot inside the left and right 45° parts will be detected by the LRF and must be ignored. The most practical solution is to limit the scanning angle with $\mp 90^\circ$; barely, this means to ignore totally a 90° scanning area. In other words this means to ignore nearly the 33.33% of the scanning area which cause to use the 2D laser sensor only with approximately 66.67% efficiency.

There is a possibility to make an assumption to overcome the problem in blind regions by positioning the 2D LRF in the traditional position having an angle with the floor. Although the different utilizations of 2D LRF positions, there is no way to acquire a view without any blind region. There is a requirement to attach a second 2D LRF for

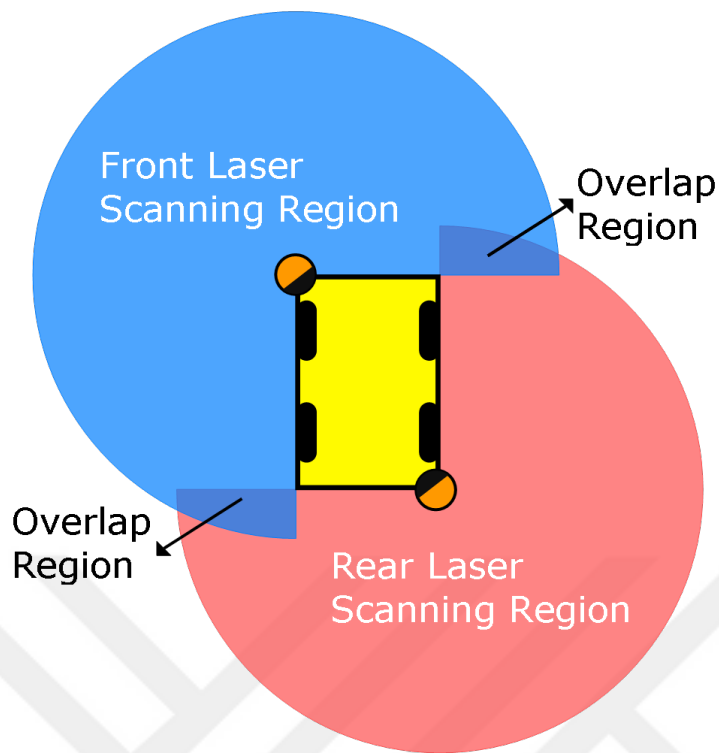


Figure 7.5 : TRAP configuration for 2D LRFs

a scan without any blind region. Also an appropriate location must be preferred for the 2D LRF. For instance placing one 2D LRF on front of the mobile robot and the other LRF on the rear of the mobile robot in a centered alignment symmetrically will not be a successful combination for a scan without any blind region. This mentioned configuration most probably cause to a narrow and long horizontal blind region.

The TRAP configuration has been generated as a result of the focus on obtain the optimal position for two 2D LRFs in unmanned applications. The LRFs have been located to the corner of the mobile robot in a transversal format to each other given in Figure 7.5. Moreover, both 2D LRFs have been positioned on the corner with a 45° angle with the mobile robots front and rear edge.

TRAP Configuration is appropriate for 2D LRFs with a scanning range of 270° like Hokuyo UTM-30LX. There are three crucial benefits for the TRAP Configuration. First of all the TRAP configuration offers to the 2D LRFs to scan a circular region and this means a 360° view for the mobile robot. Another advantage is that the upside of the mobile robot will be inside a blind region for both 2D LRFs and the equipments on the

mobile robot will not anymore act like disturbance for the 2D LRFs. The non-scanning 90° area have been turned to a fructuous feature. Finally the two 2D LRFs will generate overlap regions which will be useful for verification of both 2D laser sensors.

7.3 Experiments

In this section the experiments have been classified under three variations. The first part of the experiments are for SLAM techniques achieved in traditional 2D LRF configuration and have been compared with each other. In the next part of the experiments the same SLAM techniques have been achieved for TRAP configuration of 2D LRF configuration and again a benchmarking have been done for each SLAM method. Finally in the last part the SLAM techniques achieved for traditional configuration and TRAP configuration have been compared for a benchmark. The first two experiment is for comparing the different SLAM algorithms; however, in the last part of the experiments the difference between traditional configuration and TRAP configuration of LRF have been analyzed and compared.

To observe the efficiency of different SLAM algorithms and different sensor configurations exactly the same experimental environments have been used for a fair comparison. For achieve an extensive research three different scenarios have been designed. The first scenario is about an environment without any obstacles, a completely empty map, there exist only walls. Second scenario is just a little complex from the previous one. This scenario includes identical static obstacles. The obstacles are cubical formed with $1 \times 1 \times 1m$ dimensions and five of them have been used to generate a map with obstacles. The final scenario is the most compelling, it includes dynamic obstacles including static obstacles and the number of static obstacles have been increased. Fifteen obstacles, which are same with the previous scenario, have been implemented to the environment. Additionally to these obstacles three persons have been walked around randomly as dynamic obstacles. For these three scenarios a go to goal mission have been performed for different ROS based SLAM techniques in traditional and TRAP 2D LRF configurations. In every scenario the mobile robot have been started from the same point and the goal is also the same point. Only the model of the environment or similarly the map changes. A view of the experimental environment for the first scenario is given in Figure 7.6.



Figure 7.6 : The experimental environment in real world and simulation

The experiments have been achieved first of all in simulation environment. There are many reasons for this attitude. Working on a simulation makes it possible to make the experiments any time and any place. The Independence on the experimental setup solve the battery problem and this results to make long experiments without any concern of a battery die and battery charge problems which is really an annoying situation. Additionally simulation offers a protection to the real system from possible damages. In according these advantages a simulation environment has been designed in Gazebo for the real time environment shown in Figure 7.6. Through the agency of ROS by modifying a few lines of codes algorithms can be modified for the real time environment; therefor, the simulations do not increase the workload dramatically.

The experimental results have been examined by considering four basic criteria. These four criteria are: Performance, CPU load, algorithm runtime and power consumption. Under the performance criterion the analyze have been made for two benchmark. Benchmark I is about the accuracy for reaching the goal point and benchmark II is about traveled distance until reaching the goal (Δx). For the CPU load on the ground station has been evaluated as the CPU load criterion. The algorithm runtime criterion has been decided by the elapsed time from the beginning to the assignment until achieve the objectives. The last criterion power load is only meaningful if there is a change in the hardware, because differences in applied SLAM techniques do not cause to any change in power consumption. Therefore, the power consumption criterion have been considered in the comparison of traditional configuration and TRAP configuration of the LRFs.

7.3.1 Comparison of the SLAM techniques for traditional configuration

The first part of the experimental works are about comparing the SLAM techniques available in ROS for the traditional LRF configuration. The performed SLAM algorithms for the traditional LRF configuration are Gmapping and Hector SLAM.

7.3.1.1 Gmapping

The accuracy and traveled distance data have been calculated by the odometry and wheel encoders of the mobile robot. The accuracy have been decided by the final position of the mobile robot in the end of the algorithm. Similarly the traveled distance have been calculated by the change in the wheel encoders at the beginning and at the end of the algorithm. The data for performance criterion have been given in Figure 7.2.

Table 7.2 : Performance criterion of gmapping for traditional configurations

Scenario	Benchmark I	Benchmark II (Δx)
1	96.08%	6.9m
2	92.79%	39.2m
3	91.64%	169.1m

It is necessary to give the informations of the computers used on the experimental works before present the results in CPU load criterion. Two different laptops have been used for the experimental studies. One of this laptop is for the AGV, and the other one is for the ground station. The laptop on the AGV is less powerful than the laptop on the ground station. The reason of this situation is that the main programs have been run on the ground station laptop. The configurations for the laptop on the AGV are: Intel Core 2 Duo SU9600/1.6 GHz CPU and 4 GB 1333 MHz DDR3 SDRAM. On the other hand the laptop configurations for the ground station are: Intel Core i7 4710MQ/2.5 GHz CPU and 16 GB 2133 MHz. DDR4L SDRAM. Table 7.3 presents the CPU load values for three scenarios on the ground station laptop.

Table 7.3 : CPU load criterion of gmapping for traditional configurations

Scenario	$\bar{X}$	σ
1	6.32%	4.58%
2	7.11%	4.21%
3	9.07%	4.69%

In Table 7.3 $\bar{X}$ represents the mean value of the CPU load, and σ represents the standard deviation of the CPU load.

The algorithm runtime data have been collected with a high attention by keeping the beginning and goal points same in all scenarios and Table 7.4 has been generated to show the algorithm runtime of each scenario.

Table 7.4 : Algorithm runtime criterion of gmapping for traditional configurations

Scenario	Algorithm Runtime
1	7.2 s
2	49.7 s
3	396.4 s

There is no change in the hardware configuration of the experimental setup for the first part of the experiments; therefore, the power consumption criterion has been neglected.

7.3.1.2 Hector SLAM

The accuracy and traveled distance data have been calculated by the odometry and wheel encoders of the mobile robot. The accuracy have been decided by the final position of the mobile robot in the end of the algorithm. Similarly the traveled distance have been calculated by the change in the wheel encoders at the beginning and at the end of the algorithm. The data for performance criterion have been given in Figure 7.5.

Table 7.5 : Performance criterion of hector SLAM for traditional configurations

Scenario	Benchmark I	Benchmark II (Δx)
1	96.34%	7.0m
2	92.63%	38.8m
3	90.92%	169.0m

Table 7.6 presents the CPU load values for three scenarios on the ground station laptop.

Table 7.6 : CPU load criterion of hector SLAM for traditional configurations

Scenario	$\bar{X}$	σ
1	6.21%	4.13%
2	7.33%	4.32%
3	9.01%	4.75%

In Table 7.6 $\bar{X}$ represents the mean value of the CPU load, and σ represents the standard deviation of the CPU load.

The algorithm runtime data have been collected with a high attention by keeping the beginning and goal points same in all scenarios and Table 7.7 has been generated to show the algorithm runtime of each scenario.

Table 7.7 : Algorithm runtime criterion of Hector SLAM for traditional configurations

Scenario	Algorithm Runtime
1	7.6 s
2	49.0 s
3	397.3 s

There is no change in the hardware configuration of the experimental setup for the first part of the experiments; therefore, the power consumption criterion has been neglected.

7.3.1.3 Discussion

In this section Gmapping and Hector SLAM have been compared. There are no remarkable performance differences between these two SLAM methods; therefore, the comparison has been based on the generated maps given in figure 7.7.

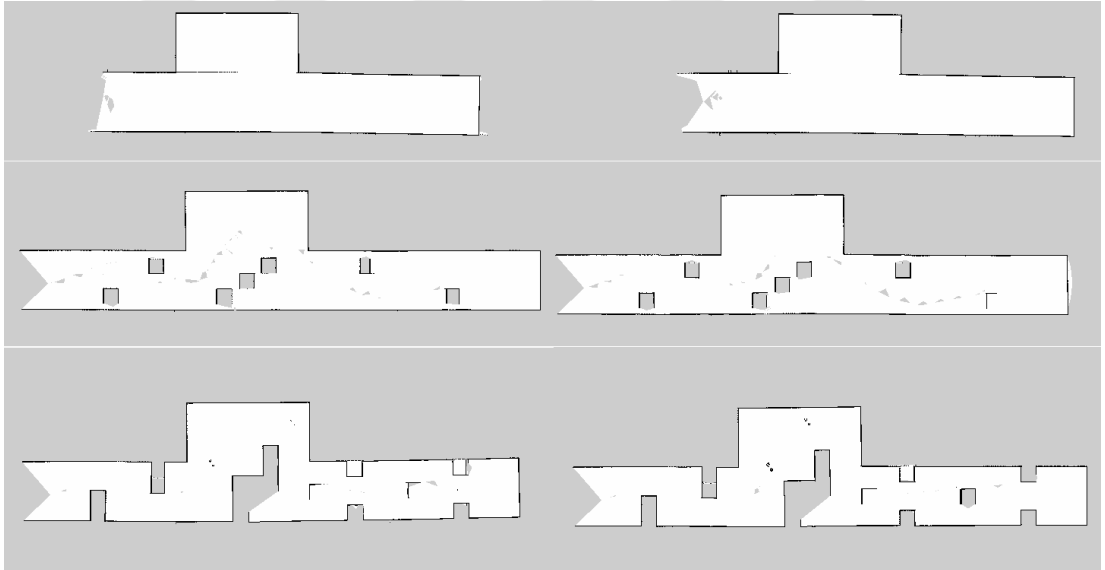


Figure 7.7 : Generated maps by gmapping and hector SLAM for trad. conf.

In figure 7.7 the maps on the left side have been generated by Gmapping and the other ones on the right side have been generated by Hector SLAM. All the three maps represent the three different scenarios generated for the experimental works. It is possible to declare that there are no significant differences for the maps created;

however, in scenario 2 the map of Gmapping and in scenario 3 the map of Hector SLAM are more detailed.

7.3.2 Comparison of the SLAM techniques for TRAP configuration

The second part of the experimental works are about comparing the SLAM techniques available in ROS for the TARP configuration in LRF sensors. The performed SLAM algorithms for the TRAP configuration are Gmapping and Hector SLAM.

7.3.2.1 Gmapping

The accuracy and traveled distance data have been calculated by the odometry and wheel encoders of the mobile robot. The accuracy have been decided by the final position of the mobile robot in the end of the algorithm. Similarly the traveled distance have been calculated by the change in the wheel encoders at the beginning and at the end of the algorithm. The data for performance criterion have been given in Figure 7.8.

Table 7.8 : Performance criterion of gmapping for TRAP configurations

Scenario	Benchmark I	Benchmark II (Δx)
1	96.21%	6.8m
2	95.40%	30.2m
3	95.01%	66.7m

It is necessary to give the informations of the computers used on the experimental works before present the results in CPU load criterion. Two different laptops have been used for the experimental studies. One of this laptop is for the AGV, and the other one is for the ground station. The laptop on the AGV is less powerful than the laptop on the ground station. The reason of this situation is that the main programs have been run on the ground station laptop. The configurations for the laptop on the AGV are: Intel Core 2 Duo SU9600/1.6 GHz CPU and 4 GB 1333 MHz DDR3 SDRAM. On the other hand the laptop configurations for the ground station are: Intel Core i7 4710MQ/2.5 GHz CPU and 16 GB 2133 MHz. DDR4L SDRAM. Table 7.9 presents the CPU load values for three scenarios on the ground station laptop.

In Table 7.9 $\bar{X}$ represents the mean value of the CPU load, and σ represents the standard deviation of the CPU load.

Table 7.9 : CPU load criterion of gmapping for TRAP configurations

Scenario	$\bar{X}$	σ
1	8.02%	4.10%
2	9.97%	4.41%
3	12.26%	4.33%

The algorithm runtime data have been collected with a high attention by keeping the beginning and goal points same in all scenarios and Table 7.10 has been generated to show the algorithm runtime of each scenario.

Table 7.10 : Algorithm runtime criterion of gmapping for TRAP configurations

Scenario	Algorithm Runtime
1	7.2 s
2	33.0 s
3	71.2 s

There is no change in the hardware configuration of the experimental setup for the second part of the experiments; therefore, the power consumption criterion has been neglected.

7.3.2.2 Hector SLAM

The accuracy and traveled distance data have been calculated by the odometry and wheel encoders of the mobile robot. The accuracy have been decided by the final position of the mobile robot in the end of the algorithm. Similarly the traveled distance have been calculated by the change in the wheel encoders at the beginning and at the end of the algorithm. The data for performance criterion have been given in Figure 7.11.

Table 7.11 : Performance criterion of hector SLAM for TRAP configurations

Scenario	Benchmark I	Benchmark II (Δx)
1	97.01%	7.0m
2	95.16%	29.6m
3	94.79%	67.3m

Table 7.12 presents the CPU load values for three scenarios on the ground station laptop.

Table 7.12 : CPU load criterion of hector SLAM for TRAP configurations

Scenario	$\bar{X}$	σ
1	7.63%	3.93%
2	10.12%	5.12%
3	11.59%	4.71%

In Table 7.12 $\bar{X}$ represents the mean value of the CPU load, and σ represents the standard deviation of the CPU load.

The algorithm runtime data have been collected with a high attention by keeping the beginning and goal points same in all scenarios and Table 7.13 has been generated to show the algorithm runtime of each scenario.

Table 7.13 : Algorithm runtime criterion of hector SLAM for TRAP configurations

Scenario	Algorithm Runtime
1	7.5 s
2	32.1 s
3	72.6 s

There is no change in the hardware configuration of the experimental setup for the second part of the experiments; therefore, the power consumption criterion has been neglected.

7.3.2.3 Discussion

In this section Gmapping and Hector SLAM have been compared. The results have been collected for the TRAP configuration. There are no remarkable performance differences between these two SLAM methods; therefore, the comparison has been based on the generated maps given in figure 7.8 similarly to the discussion for the traditional configuration comparison.

In figure 7.7 the maps on the left side have been generated by Gmapping and the other ones on the right side have been generated by Hector SLAM. All the three maps represent the three different scenarios generated for the experimental works. It is possible to declare that there are no significant differences for the maps created; however, in scenario 2 the map of Gmapping and in scenario 3 the map of Hector SLAM are more detailed similarly to the results of the traditional configuration maps. Moreover, the generated maps for traditional configuration given in figure 7.7 and

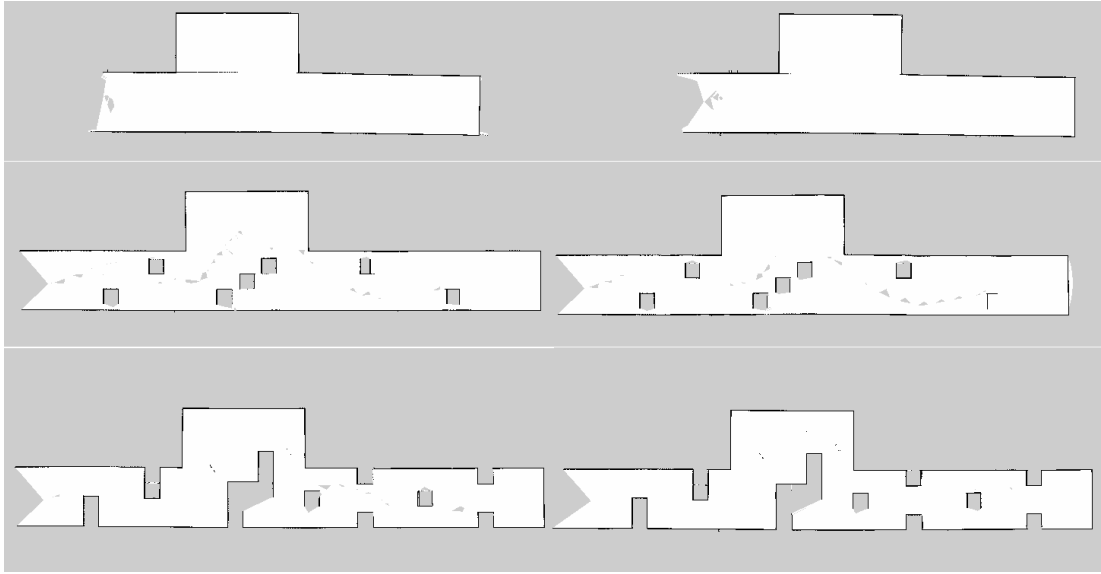


Figure 7.8 : Generated maps by gmapping and hector SLAM for TRAP conf.

TRAP configuration 7.8 looks very similar to each other; nonetheless, the detail level for scenario 3 has been improved for both methods.

7.3.3 Comparison of traditional and TRAP configuration

In this part of experimental works the TRAP configuration is compared with the traditional configuration in which the 2D LRF is positioned centered on the front and the rear of the mobile robot in ROS based SLAM techniques for Gmapping and Hector SLAM.

7.3.3.1 Gmapping

For the analyze of performance criterion, after the collection process it has been realized that the results for TRAP configuration are much better than traditional configuration. Table 7.14 have been created by using equation 7.1 for every scenario in each step.

Table 7.14 : Performance criterion of gmapping for traditional and TRAP configurations

Scenario	Benchmark I	Benchmark II (Δx)	Total
1	99.87	99.21	99.54
2	97.26	77.13	87.20
3	96.45	39.44	67.95

In table 7.14 being 100 means exactly the same value with both configurations. Lower than 100 means better results for TRAP configuration, and higher than 100 means better results for traditional configuration. The total consideration for each benchmark have been taken by giving the same weight 0.5 for both benchmarks; however, the weight can be changed as desired according to the importance of the benchmarks.

$$[Bennchmark]_i = \frac{[TRAP Configuration]_i}{[Traditional Configuration]_i} \times 100 \quad for \ i = I, II \quad (7.1)$$

The equation 7.1 is actually a percentage of the traditional configuration performance against the TRAP configuration performance.

It is necessary to give the informations of the computers used on the experimental works before present the results in CPU load criterion. Two different laptops have been used for the experimental studies. One of this laptop is for the AGV, and the other one is for the ground station. The laptop on the AGV is less powerful than the laptop on the ground station. The reason of this situation is that the main programs have been run on the ground station laptop. The configurations for the laptop on the AGV are: Intel Core 2 Duo SU9600/1.6 GHz CPU and 4 GB 1333 MHz DDR3 SDRAM. On the other hand the laptop configurations for the ground station are: Intel Core i7 4710MQ/2.5 GHz CPU and 16 GB 2133 MHz. DDR4L SDRAM. Table 7.15 presents the CPU load values for three scenarios on the ground station laptop.

Table 7.15 : CPU load criterion of gmapping for traditional and TRAP configurations

Scenario	Traditional Configuration		TRAP Configuration	
	$\bar{X}_1$	σ_1	$\bar{X}_2$	σ_2
1	6.32%	4.58%	8.02%	4.10%
2	7.11%	4.21%	9.97%	4.41%
3	9.07%	4.69%	12.26%	4.33%

The values with index 1 are collected for traditional configuration, and the values with index 2 are collected for TRAP configuration. $\bar{X}$ shows the mean value for the CPU load, and σ is used for standard deviation.

The CPU load for all three scenarios and two configurations have been also shown in Figure 7.9. The mean values and standard deviations in Table 7.15 have been calculated by using the CPU load chart in Figure 7.9. Each color represents a different scenario;

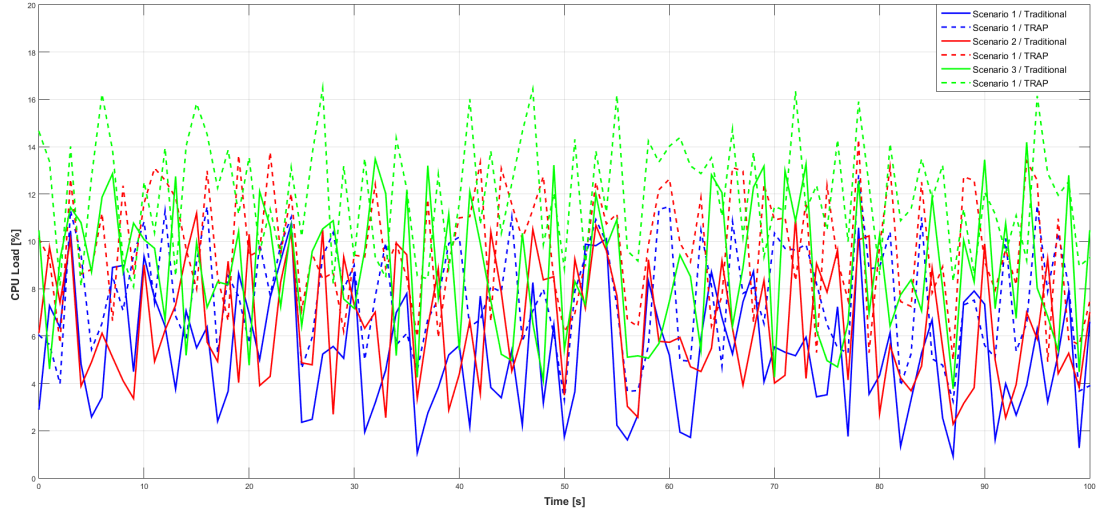


Figure 7.9 : CPU load criterion of gmapping for traditional and TRAP conf.

blue for first, red for second and green for third scenario. Continuous lines are for the traditional configuration and dashed lines are for the TRAP configuration.

In the algorithm runtime criterion for a fair assessment in all scenarios the environment have been kept identical for both configurations. Also it have been paid attention to keep the beginning and goal points same in all scenarios for two LRF positions. By following this way it is equitable to analyze the results.

Table 7.16 : Algorithm runtime criterion of gmapping for traditional and TRAP configurations

Scenario	Traditional Configuration	TRAP Configuration
1	7.2 s	7.2 s
2	49.7 s	33.0 s
3	396.4 s	71.2 s

Table 7.16 represents the time elapsed from beginning to the mission until reaching the goal point for all three scenarios in traditional configuration and TRAP configuration.

In the system setup for both configurations only the number of 2D LRFs changes. For the power consumption criterion this situation must be considered. As a reminder Hokuyo UTM-30LX have been used as 2D LRF. The sensor needs a 12V DC voltage and a maximum current of 1A with the mean of 0.7A. In the circumstances the power consumption of one Hokuyo UTM-30LX will be maximum 12W with a mean of 8.4W. For TRAP configuration the power consumption will be twofold of traditional configuration with a mean of 16.8W and maximum 24W.

7.3.3.2 Hector SLAM

For the analyze of performance criterion, after the collection process it has been realized that the results for TRAP configuration are much better than traditional configuration similarly to Gmapping. Table 7.17 have been created by using equation 7.1 for every scenario in each step.

Table 7.17 : Performance criterion of hector SLAM for traditional and TRAP configurations

Scenario	Benchmark I	Benchmark II (Δx)	Total
1	99.31	100.00	99.66
2	97.34	76.29	86.82
3	95.92	39.82	67.87

In table 7.17 being 100 means exactly the same value with both configurations. Lower than 100 means better results for TRAP configuration, and higher than 100 means better results for traditional configuration. The total consideration for each benchmark have been taken by giving the same weight 0.5 for both benchmarks; however, the weight can be changed as desired according to the importance of the benchmarks.

The equation 7.1 is actually a percentage of the traditional configuration performance against the TRAP configuration performance.

Table 7.18 presents the CPU load values for three scenarios on the ground station laptop.

Table 7.18 : CPU load criterion of gmapping for traditional and TRAP configurations

Scenario	Traditional Configuration		TRAP Configuration	
	$\bar{X}_1$	σ_1	$\bar{X}_2$	σ_2
1	6.21%	4.13%	7.63%	3.93%
2	7.33%	4.32%	10.12%	5.12%
3	9.01%	4.75%	11.59%	4.71%

The values with index 1 are collected for traditional configuration, and the values with index 2 are collected for TRAP configuration. $\bar{X}$ shows the mean value for the CPU load, and σ is used for standard deviation.

In the algorithm runtime criterion for a fair assessment in all scenarios the environment have been kept identical for both configurations. Also it have been paid attention to

keep the beginning and goal points same in all scenarios for two LRF positions. By following this way it is equitable to analyze the results.

Table 7.19 : Algorithm runtime criterion of gmapping for traditional and TRAP configurations

Scenario	Traditional Configuration	TRAP Configuration
1	7.6 s	7.5 s
2	49.0 s	32.1 s
3	397.3 s	72.6 s

Table 7.19 represents the time elapsed from beginning to the mission until reaching the goal point for all three scenarios in traditional configuration and TRAP configuration.

In the system setup for both configurations only the number of 2D LRFs changes. For the power consumption criterion this situation must be considered. As a reminder Hokuyo UTM-30LX have been used as 2D LRF. The sensor needs a 12V DC voltage and a maximum current of 1A with the mean of 0.7A. In the circumstances the power consumption of one Hokuyo UTM-30LX will be maximum 12W with a mean of 8.4W. For TRAP configuration the power consumption will be twofold of traditional configuration with a mean of 16.8W and maximum 24W.

7.3.3.3 Discussion

In this section the results of the experiments have been discussed deeply for all four criteria. As a reminder the criteria are; performance, CPU load, algorithm runtime, and power consumption.

For a diffusively look to the performance criteria under the guidance of Table 7.14, this criterion have been split into two benchmarks. Benchmark I is the accuracy, benchmark II is the traveled distance by reaching the goal. It has been seen unambiguously that TRAP configuration is much better than traditional configuration. There is not a big contradistinction in accuracy; notwithstanding, traveled distance makes the difference. Specially in scenario 2, the obstacles into the map cause to a drop in benchmark II results. Furthermore, the difference in Δx have been clearly seen in scenario 3, since the number of obstacles are triple time more than previous scenario, and dynamic obstacles have been integrated. It is possible to state that TRAP configuration shortens the traveled distance almost 2.5 times after a close look into table 7.14.

CPU load data are given in Table 7.15 and Figure 7.9 detailed. As it was expected the greatest difference is for 3th scenario; however, the change in the CPU load is not remarkable, just nearly 4%. This outcome could be misleading, since CPU load depends on the laptop configurations used as ground station. Even though TRAP configuration will not effect the CPU load remarkably, for mini computers like raspberry pi the effect will be more efficient. For strong pc configurations preferred as ground station in experimental setup there is no limitation in CPU load criterion to apply the TRAP configuration for 2D LRFs.

The results for algorithm runtime criterion are intrinsically related with performance criterion. The change in traveled distance effects directly the runtime for the algorithm. When Table 7.14 and Table 7.16 collocate the relationship with performance and runtime could be seen openly. As the traveled distance increase, similarly the algorithm runtime increase also. After a detailed view to Table 7.16, TRAP configuration reduce the algorithm runtime 39.64% for scenario 2 and 82.04% for scenario 3. It is apparent that TRAP configuration is a favorable solution for lower the traveled distance and algorithm runtime.

Power consumption criterion must be dealt only by the power consumptions of Hokuyo UTM-30LX, because all the other system components are identical for both traditional and TRAP configurations, after a view from this perspective to analyze the power consumption, as might be expected the power consumption will be doubled from 8.4W - 12W to 16.8W - 24W. These consumptions are insignificant for a system with a great battery as we installed to our system (20000mAh). For 2D LRFs requiring more power and systems with small batteries this criterion is more restricting.

For two different SLAM algorithm (Gmapping and Hector SLAM) it has been proved that the TRAP configuration performs more efficiently than the traditional configuration. The difference has been clearly understood in complex environments with dynamical obstacles.

8. CONCLUSIONS

The main objective of this thesis is to make a comparison of certain 2D LRF based SLAM techniques and improving the performance by using TRAP configuration. In chapter 7 the experiments have been performed for the comparison of those SLAM techniques and for the experience of the TRAP configuration. According to the results of the experimental works it can be clearly stated that for all 2D LRF based SLAM techniques TRAP configuration offers an admissible and a satisfying result against traditional configuration. The assertion was to develop a technique to enhance the performance of a SLAM algorithm, which is the TRAP configuration in this case, in this thesis. The results of the experimental works in chapter 7 proves that the TRAP configuration offers an improvement in the performance for SLAM algorithms tested in the experimental works.

This thesis focuses on improving the TRAP configuration by generating the mathematical representation and model of the system. Mainly the mathematical expression and model of the overlap regions are unfinished and the next step of this study is solving this uncertainty.



REFERENCES

- [1] **Smith, R. and Cheesman, P.** (1987). On the representation of spatial uncertainty, *International Journal of Robotics Research*, 5(4), 56–68.
- [2] **Durrant-Whyte, H.** (1988). Uncertain geometry in robotics, *IEEE Transaction on Robotics and Automation*, 4(1), 23–31.
- [3] **Durrant-Whyte, H., Rye, D. and Nebot, E.** (1996). Localisation of automatic guided vehicles, *International Symposium on Robotics Research*, 613–625.
- [4] **Thrun, S., Fox, D. and Burgard, W.** (1998). A probabilistic approach to concurrent mapping and localization for mobile robots, *International Journal of Robotics Research*, 31(1), 29–53.
- [5] **Durrant-Whyte, H. and Bailey, T.** (2006). Simultaneous Localization and Mapping: Part I, *IEEE Robotics & Automation Magazine*, 13(2), 99–110.
- [6] **Bailey, T. and Durrant-Whyte, H.** (2006). Simultaneous Localization and Mapping: Part II, *IEEE Robotics & Automation Magazine*, 13(3), 108–117.
- [7] <http://openslam.org/>, accessed: 24.11.2016.
- [8] <http://openslam.org/>, accessed: 24.11.2016.
- [9] <http://openslam.org/dpslam.html>, accessed: 24.11.2016.
- [10] **Elizar, A. and Parr, R.** (2003). DP-SLAM: Fast, Robust Simultaneous Localization and Mapping without Predetermined Landmarks, in *Proc. 18th Int. Joint Conf. on Artificial Intelligence (IJCAI-03)*.
- [11] <http://openslam.org/gmapping.html>, accessed: 24.11.2016.
- [12] **Grisetti, G., Stachniss, C. and Burgard, W.** (2007). Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters, *IEEE Robotics and Automation Society*, 23(1), 34–46.
- [13] <http://openslam.org/gridslam.html>, accessed: 24.11.2016.
- [14] **Haehnel, D. et al.** (2003). An efficient fastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements, *Intelligent Robots and Systems*, pp.206–211.
- [15] <http://openslam.org/linearslam.html>, accessed: 24.11.2016.

- [16] **Zhao, L., Huang, S. and Dissanayake, G.** (2013). Linear SLAM: A Linear Solution to the Feature-based and Pose Graph SLAM based on Submap Joining, *Intelligent Robots and Systems*.
- [17] <http://openslam.org/openrat slam.html>, accessed: 24.11.2016.
- [18] **Ball, D. et al.** (2013). OpenRatSLAM: an open source brain-based SLAM system, *Autonomous Robots*, 34(3), 149–176.
- [19] <http://openslam.org/ro-slam.html>, accessed: 24.11.2016.
- [20] **Blanco, J.L., Madrigal, J.A.F. and Gonzalez, J.** (2008). Efficient Probabilistic Range-Only SLAM, *International Conference on Intelligent Robots and Systems*, p.1017–1022.
- [21] <http://openslam.org/tinyslam.html>, accessed: 24.11.2016.
- [22] **Steux, B. and Hamzaoui, O.E.** (2010). International Conference on Control, Automation, Robotics and Vision, *International Conference on Intelligent Robots and Systems*, p.1975–1979.
- [23] <http://openslam.org/orbslam.html>, accessed: 24.11.2016.
- [24] <http://openslam.org/ekfmonoslam.html>, accessed: 24.11.2016.
- [25] **Civera, J. et al.** (2010). 1-Point RANSAC for EKF Filtering. Application to Real-Time Structure from Motion and Visual Odometry, *Journal of Field Robotics*, 27(5), 609–631.
- [26] **Thrun, S.** (2002). Particle Filters in Robotics, in *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*.
- [27] **Sanjeev, M. et al.** (2002). A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking, *IEEE Transactions On Signal Processing*, 50(2), 174–188.
- [28] **Quigley, M., Gerkey, B. and Smart, W.D.,** (2016). Programming Robots with ROS: A Practical Introduction to the Robot Operating System, O'Reilly Media, United States of America, 1. edition.
- [29] **Martinez, A. and Fernandez, E.,** (2013). Learning ROS for Robotics Programming: A practical, instructive, and comprehensive guide to introduce yourself to ROS, the top-notch, leading robotics framework, Packt Publishing, Mumbai, 1. edition.
- [30] **O’Kane, J.M.,** (2016). A Gentle Introduction to ROS, O’Kane, J. M., Columbia, 2.1.3. edition.
- [31] <http://wiki.ros.org/Robots>, accessed: 24.11.2016.
- [32] <http://www.linux.com/blog/>, accessed: 24.11.2016.
- [33] <http://wiki.ros.org/Distributions>, accessed: 24.11.2016.

- [34] <http://wiki.ros.org/indigo/Installation/Ubuntu>, accessed: 24.11.2016.
- [35] **TheUbuntuManualTeam**, (2016). Getting Started with Ubuntu 16.04.
- [36] **Helmke, M. et al.**, (2015). The Official Ubuntu Book, Prentice Hall, 8. edition.
- [37] **Raggi, E. et al.**, (2010). Beginning Ubuntu Linux, Apress, 5. edition.
- [38] **Grant, R. and Bull, P.**, (2010). Ubuntu for Non-Geeks a Pain-Free, Get-Things-Done Guide, no starch press, 4. edition.
- [39] **Peterson, R.**, (2008). Linux: The Complete Reference, McGraw-Hill Companies, 6. edition.
- [40] <http://wiki.ros.org/ROS/Tutorials>, accessed: 24.11.2016.
- [41] <https://www.mathworks.com/products/robotics/>, accessed: 24.11.2016.
- [42] <http://gazebosim.org/>, accessed: 24.11.2016.
- [43] http://docs.ros.org/indigo/api/rviz/html/user_guide/, accessed: 24.11.2016.
- [44] **Thrun, S., Burgard, W. and Fox, D.**, (2005). Probabilistic Robotics, The MIT Press, England, 1. edition.
- [45] **Riisgaard, S. and Morten, R.B.** SLAM for Dummies A Tutorial Approach to Simultaneous Localization and Mapping, The Dummies.
- [46] <http://wiki.ros.org/>, accessed: 24.11.2016.
- [47] **Murphy, K.** (1999). Bayesian map learning in dynamic environments, *Proc. Conf. Neural Inf. Process. Syst.*, pp.1015–1021.
- [48] **Doucet, A. et al.** (2000). Rao-Blackwellized particle filtering for dynamic Bayesian networks, *Proc. Conf. Uncertainty Artif. Intell.*, pp.176–183.
- [49] <http://wiki.ros.org/gmapping>, accessed: 24.11.2016.
- [50] **Kohlbrecher, S., Meyer, J., von Stryk, O. and Klingauf, U.** (2011). A Flexible and Scalable SLAM System with Full 3D Motion Estimation, *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, IEEE.
- [51] Clearpathh Robotics, (2015). Husky Overview.
- [52] Clearpathh Robotics, (2016). Husky Data Sheet.
- [53] http://www.hokuyo-aut.jp/02sensor/07scanner/download/pdf/UTM-30LX_ed_en.pdf, accessed: 24.11.2016.
- [54] Hokuyo Automatic Co., Ltd., (2012). Scanning Laser Range Finder UTM-30LX/LN Specification.

- [55] **Ervan, O., Şencan, O. and Temeltaş, H.** (2016). Transversal Positioned (TRAP) 2D Laser Range Finder Configuration, *2016 PRASA-RobMech International Conference*, Stellenbosh, South Africa.
- [56] **Foote, T.** (2013). tf: The transform library, *Technologies for Practical Robot Applications*, United States of America.



APPENDICES

APPENDIX A.1 : Basic Ubuntu Commands

APPENDIX A.2 : Basic ROS Commands



APPENDIX A.1

The **ls** *list* command:

```
ls
```

This command lists all the files in the current directory.

The **cd** *change directory* command:

```
cd directory_1 / directory_2 / ...
```

This command is used for changing directories.

The **pwd** *point working directory* command:

```
pwd
```

This command shows the directory located.

The **mkdir** *make directory* command:

```
mkdir directory_name
```

This command is used to create a directory.

The **rmdir** *remove directory* command:

```
mkdir directory_name
```

This command is used to delete an empty directory. It will not delete the directory if it is not empty.

The **rmdir** *remove directory* command:

```
mkdir directory_name
```

This command is used to delete an empty directory. It will not delete the directory if it is not empty.

The **rm** *remove* command:

```
rm file_name
```

This command is used to delete a file.

How to delete a **non-empty directory**:

```
rm -rf directory_name
```

This command is used to delete any directory.

The **mv** *move* command:

```
mv file_name folder_name_to_move_file
```

This command is used to move a file.

The **cp** *copy* command:

```
cp file_name folder_name_to_copy_file
```

This command is used to copy a file.

Running as administrator on Ubuntu:

```
sudo
```

The given command above must be attached to the beginning of any command to work as administrator. sudo means Super User DO.

The key combination to **copy** something on terminal:

```
ctrl + shift + c
```

The key combination to **paste** something to terminal:

```
ctrl + shift + v
```

The key combination to **suspend a process**:

```
ctrl + z
```

APPENDIX A.2

roscd: Is used for changing directory by using package name. The main difference between **cd** is that it is not necessary to give the full path of the package, just the package name is enough.

```
roscd package_name
```

rosls: Is used for listing the files and directories of a package. The main difference between **ls** is that it is not necessary to be at the point working directory where the package is.

```
rosls package_name
```

roscrcat-pkg: Is used for creating a ROS package.

```
roscrcat -pkg package_name
```

roswtf: Is used for checking any errors and warnings. It is possible to make a general check for everything running or check for a specific launch file.

```
roswtf  
roswtf launch_file_name
```

roscore: For running any package or launch file it is necessary to run a **roscore**. After running **roscore** the desired operation must be done in a new terminal; however, it is possible to write a command into the same terminal where **roscore** runs by adding **&** to the command.

```
roscore  
roscore &
```

rosmmsg show: Is used for showing ROS message type information in data size and data type.

```
rosmmsg show package_name
```

rossrv show: Is used for showing ROS service type information in data size and data type.

```
rossrv show package_name
```

roslaunch: Is used for running an executable in a package without being at the point working directory.

```
roslaunch package_name executable_name
```

rostopic: Is used for showing informations and giving commands to ROS nodes. **rostopic** is used with additional commands.

```
rostopic list           // lists the nodes which are active  
rostopic info node_name // gives informations about a node  
rostopic kill node_name // kills a running node  
rostopic ping node_name // shows the connection parameters  
rostopic machine machine_name // lists all running nodes on a machine
```

roslaunch: Is used for running a launch file.

```
roslaunch package_name launch_file_name
```

rostopic: Is similar to **roscall**, but the difference is shows informations and gives commands to ROS topics and used with additional commands.

```
rostopic list           //lists active topics
rostopic info topic_name //shows information about active topic
rostopic type topic_name //shows the type of a topic
rostopic find msg_type  //find topics by given type
rostopic echo topic_name //outputs messages published by topic
rostopic bw topic_name  //shows a topics bandwidth
rostopic hz topic_name  //shows publishing frequency of a topic
rostopic delay topic_name //shows the delay for a topic
rostopic pub topic_name topic_type data //publishes data to a topic
```

rosparam: Is used for setting and getting ROS parameters. **rosparam** is used with additional commands.

```
rosparam list           //lists all parameter names
rosparam load yaml_file //loads parameters from a yaml file
rosparam dump file_name.yaml //dumps parameters to a yaml file
rosparam delete parameter_name //deletes the given parameter
rosparam get parameter_name //get the give parameter value
rosparam set parameter_name data //sets a value to a parameter
```

rosviz: Is used for recording and playing topic data. Most generally these topics could be sensor values. The topic data are recorded as .bag file and **rosviz** will play .bag files.

```
rosviz record -a           //records all topics
rosviz record topic1 topic2 ... //records specific topics
rosviz play rosviz_file.bag //replays given files messages
```

tf: Is used for transformations between source and target frames, generated by Foote [56]. These are some examples for using the **tf** package.

Showing the frames being broadcast by **tf** over ROS:

```
rosviz tf view_frames
```

Downloading the frames being broadcast by **tf** over ROS as a pdf file:

```
evince <file_name>.pdf
```

This code will show the frame transformation offline.

Opening a runtime tool to show frames being broadcast by **tf** over ROS:

```
rosviz rqt_tf_tree rqt_tf_tree
```

This code will show the frame transformation online and the tree could be regenerated by pressing the refresh button.

To get the transformation information of any two frames broadcast over ROS:

```
rosviz tf tf_echo [reference_frame] [target_frame]
```



CURRICULUM VITAE



Name Surname: Osman ERVAN

Place and Date of Birth: Bremerhaven 20.07.1991

E-Mail: osmanervan@gmail.com

EDUCATION:

- **B.Sc.:** 2014, Istanbul Technical University, Faculty of Electrical and Electronics Engineering , Department of Control and Automation Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2015 - Present: Research assistant at Istanbul Technical University Department of Control and Automation Engineering.
- 3th scoring student of the educational year 2013 - 2014 in the department of Control and Automation Engineering.

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- Ay, E., Şencan, O., Ervan, O., Temeltaş, H., 2015. Gezgin Robotlarda ROS ile Açık Kaynaklı Uygulama Geliştirme. *Otomatik Kontrol Türk Milli Komitesi Konferansı - TOK 2015*, 2015 Denizli, Turkey.
- Ervan O., Şencan O., Temeltaş H., 2016. Diyagonal 2 Boyutlu Lazer Yerleşimi (d2Bly) Algoritması. *Otomatik Kontrol Türk Milli Komitesi Konferansı - TOK 2016*, 2016 Eskişehir, Turkey.
- Ervan O., Şencan O., Temeltaş H., 2016. Transversal Positioned (TRAP) 2D Laser Range Finder Configuration. *Symposium of the Pattern Recognition Association of South Africa (PRASA) and Conference of Robotics and Mechatronics - PRASA-RobMech 2016*, 2016 Stellenbosh, South Africa.