T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ARAMA KURTARMA AMAÇLI DİFERANSİYEL SÜRÜŞLÜ TEKERLEKLİ ROBOT TASARIMI VE GERÇEKLENMESİ

FURKAN ÇAKMAK

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI

DANIŞMAN
YRD. DOÇ. DR. SIRMA ÇEKİRDEK YAVUZ

İSTANBUL, 2014

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

ARAMA KURTARMA AMAÇLI DİFERANSİYAL SÜRÜŞLÜ TEKERLEKLİ ROBOT TASARIMI VE GERÇEKLENMESİ

Furkan ÇAKMAK tarafından hazırlanan tez çalışması 17.06.2014 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Yrd. Doç. Dr. Sırma ÇEKİRDEK YAVUZ

Yıldız Teknik Üniversitesi

Jüri Üyeleri

Yrd. Doç. Dr. Sırma ÇEKİRDEK YAVUZ

Yıldız Teknik Üniversitesi _____

Yrd. Doç. Dr. Mehmet Fatih AMASYALI

Yıldız Teknik Üniversitesi _____

Yrd. Doç. Dr. Umut Engin AYTEN

Yıldız Teknik Üniversitesi _____

Bu çalışma, Türkiye Bilimsel ve Teknolojik Araştırma Kurumu TÜBİTAK' ın EEEAG-113E212 numaralı projesi ile desteklenmiştir.

Bu çalışma, Yıldız Teknik Üniversitesi Bilimsel Araştırma Projeleri Koordinatörlüğü' nün 2013-04-01-KAP02 numaralı projesi ile desteklenmiştir.

ÖNSÖZ

Çalışmamı, yorucu yüksek lisans sürecini tamamlamamda bana her türlü desteği veren, çalışmalarımda beni cesaretlendiren ve her zaman yanımda olan aileme ithaf ediyorum.

Çalışmalarımı yönlendiren, araştırmalarımın her aşamasında bilgi, öneri ve yardımlarını esirgemeyen danışman hocam Sayın Yrd. Doç. Dr. Sırma Çekirdek Yavuz'a sonsuz teşekkürlerimi sunuyorum.

Çalışmalarım sırasında önemli katkılarda bulunan ve yönlendiren, yoğunluğu çok olmasına rağmen takıldığım yerleri sabırla dinleyerek çözüm önerileri getiren Sayın Arş. Gör. Dr. Erkan Uslu'ya derin duygularımla teşekkür ederim.

Başta Olasılıksal Robotik Grubu üyeleri olmak üzere Yıldız Teknik Üniversitesi, Bilgisayar Mühendisliği Bölümü akademik personeline ve çalışanlarına desteklerinden dolayı teşekkürlerimi sunuyorum.

Ayrıca her zaman yayımda olduklarını hissettiğim Yiğitler Oymağı izcilerine, izcilerime teşekkürü borç bilirim.

Yüksek Lisans çalışmamın gerçekleşmesinde proje desteği sağlayan Yıldız Teknik Üniversitesi Bilimsel Araştırma Projeleri Koordinatörlüğü'ne teşekkür ederim.

Son olarak Yüksek Lisans sürecimi 1001 - Bilimsel ve Teknolojik Araştırma Projelerini Destekleme Programı kapsamında destekleyen TÜBİTAK'a teşekkür ederim.

Haziran, 2014

Furkan ÇAKMAK

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ

λ	Dalga boyu
a	Verilen komut kümesi
m	Haritası
n_i^T	Doğru normalinin transpozu
p	Robotun anlık pozisyonu
p_i	Laser noktaları kümesi
o	Okunan duyarga bilgileri
S^{ref}	Referans yüzeyi alanı
u	Odometri ölçümleri
w	Parçacık ağırlıkları
X	Parçacıkların anlık inanç durumu
x	Robotun gezinleri
z	Ortama dair gözlemler

KISALTMA LİSTESİ

AHRS	Attitude and Heading Reference System
AMCL	Augmented Monte Carlo Localization
BSD	Berkeley Software Distribution
DC	Direct Current
DDR3	Double Data Rate 3
EKF	Extended Kalman Filter
GPS	Global Position System
HD	High Definition
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
LiPo	Lithium Polymer Battery
LSM	Laser Scan Matcher
LSM-A	LSM with IMU Sensor
LSM-AK	LSM with IMU Sensor and Kinematics Model
LSM-K	LSM with Kinematics Model
MCL	Monte Carlo Localization
NIST	The American National Institute of Science and Technology
P3AT	Pioneer3-AT
PLICP	Point-to-Line Iterative Closest Point
PR2	Personal Robotics 2
PSO	Particle Swarm Optimization
RAM	Random Access Memory
ROS	The Robot Operating System
RPM	Revolutions Per Minute
RRL	Rescue Robot League
RGB-D	Red Green Blue – Depth
RPE	Robot Pose EKF
RSL	Rescue Simulation League
SLAM	Simultaneous Localization and Mapping
SSH	Secure SHell
STAIR	STanford Artificial Intelligence Robot
XML	Extensible Markup Language

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

ÖZET

ARAMA KURTARMA AMAÇLI DİFERANSİYAL SÜRÜŞLÜ TEKERLEKLİ ROBOT TASARIMI VE GERÇEKLENMESİ

Furkan ÇAKMAK

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Tez Danışmanı: Yrd. Doç. Dr. Sırma ÇEKİRDEK YAVUZ

Birçok farklı alandaki teknolojik yeniliklerle birlikte robot teknolojisi de hızla gelişim kazanmıştır. Gerektiğinde insanların yüklerini azaltmak için geliştirilen bu sistemlerin bir kısmı, aynı zamanda eğlence amaçlı da tasarlanmaktadır. Günümüz itibariyle askeri ve biyomedikal sistemler, küçük ev aletleri, insan benzeri humanoid robotlar vb. olmak üzere birçok farklı alanda robot teknolojisi kullanılmaktadır.

Bu alanda kendini gösteren bir diğer branş da arama kurtarma robot teknolojileridir. Doğal afetlerde veya kazalarda insan müdahalesinin zor ve yetersiz olduğu durumlarda arama kurtarma ekiplerine bilgi sağlamak için tasarlanan robot platformları bu başlık altında incelenmektedir. Deprem, sel, radyoaktif kazalar ve çeşitli felaketlerde kullanılmak üzere geliştirilen, bir kısmı alanında özelleşmiş, bir kısmı genel kullanım amacıyla üretilmiş çeşitli robot teknolojileri arama kurtarma alanında hizmet vermektedir.

Her bir kaza ve enkaz branşında hizmet vermesi için geliştirilen farklı robotların ortak gereksinimleri vardır. Mesela her bir robot, farklı modeller kullansa da hareket etmek zorundadır. Aynı şekilde her bir robot farklı miktarlarda da olsa, ortama dair bilgi edinme ve gezindiği ortamı tanıma ihtiyacı hissetmektedir.

Bu çalışma kapsamında, bahsedilen ortak gereksinimler tespit edilmiş ve arama kurtarma amaçlı diferansiyel sürüşlü tekerlekli robot platformu tasarımı ve

gerçeklenmesi yapılmıştır. Bu bağlamda ROS (The Robot Operating System) platformu Hydro sürümü üzerinde ROS gereksinimleri dikkate alınarak ve kütüphaneleri kullanılarak arama kurtarma robotlarında olan ihtiyaçların giderilmesiyle ilgili tasarımlar uygulamalı olarak gerçekleştirilmiş ve deneysel sonuçları elde edilmiştir.

Bu gereksinimler, robotun sürüşü, konum ve duruş kestirimi, ortamın haritasının çıkarılması ve otonom olarak gezinimi başlıkları altında incelenebilir. Bu çalışma için geliştirilen robot platformu, 4 teker çekişli diferansiyel sürüşlü bir hareket modeline sahiptir. Her bir tekerlek farklı hız komutları alabilmekte, böylece istenilen dönüşler sağlanabilmektedir. Konum ve duruş kestirimi için lazer mesafe ölçüm duyargası kullanarak çalışan temelde tarama eşlemeye (scan matching) dayanan laser_scan_matcher yöntemi kullanılmış ve deneysel sonuçlar toplanmıştır. Haritalama problemi için genel olarak gMapping üzerinde durulmuş ve bu yöntemle ilgili deneyler yapılarak sonuçlar elde edilmiştir. Çalışma kapsamında incelenen bir diğer konu robotun navigasyonudur. Bunun için ROS üzerinde gerçekleştirilen “navigation stack” paketi incelenmiş çalıştırılmış ve sonuçlar alınmıştır. Bu paket, temelde robotun lokalizasyon bilgisine ihtiyaç duyduğu için Monte Carlo lokalizasyonunun (MCL) adaptif versiyonu olan Augmented MCL (AMCL) kullanılmış ve sonuçları alınmıştır.

Yapılan deneylerde, gerçek robot platformu kullanılmış ve sonuçlar farklı parametrelerle test edilerek eniyilenmiştir.

Anahtar Kelimeler: Arama kurtarma robotları, haritalama algoritmaları, gMapping, lokalizasyon, Augmented Monte Carlo lokalizasyonu, ROS (The Robot Operating System) Hydro, laser scan matcher, robot platformu

ABSTRACT

DIFFERENTIAL DRIVE WHEELED ROBOT DESIGN AND IMPLEMENTATION FOR SEARCH AND RESCUE

Furkan ÇAKMAK

Department of Computer Engineering

MSc. Thesis

Adviser: Assist. Prof. Dr. Sırma ÇEKİRDEK YAVUZ

Robot technology has developed rapidly within many different areas of technological innovation. Some of these systems not only developed to reduce the daily load of the life but also designed for entertainment. Nowadays, robot technology is used in many different areas including military applications, biomedical systems, humanoid robots and small appliances.

The search and rescue robot technology is another branch, attracting a lot of interest among researchers all over the world. It is obvious that using robots in disaster environments can provide information to the rescue teams in natural disasters or accidents where human intervention is difficult and inefficient. Various search and rescue situations where robots may serve include earthquakes, floods, accidents and radioactive disasters. Some of these robots specialized in a specific area while others are produced for general use.

Different robots, designed to serve in different type of accidents and debris, have common needs. For example, each robot has to navigate through the environment, even though it uses different models. In the same way, each of them needs to obtain information about the environment and to recognize the environment in different levels.

In this study, a differential drive wheeled robot platform is designed and implemented to respond common requirements identified for search and rescue operations. Hydro version of the ROS framework and existing libraries are used in this study to implement some algorithms essential for search and rescue robots. Drivers provided within ROS are taken into consideration during the design of the robot platform.

Main requirements examined can be classified as robot navigation, pose estimation, simultaneous localization and mapping. The robot platform, developed for this study has a four-wheeled differential drive motion model. Each wheel may take a different speed commands. In this way, desired rotation can be provided. Laser_Scan_Matcher (LSM) method was used for location estimation. The method uses laser distance measurements to match consecutive laser scans. After that, the experimental results were collected. For mapping problem, the study focuses on gMapping method. Related experiments and the results obtained by this method are discussed. Another issue that was analyzed in the study is the robot's navigation. For that purpose, "navigation stack" package provided for ROS has been run and the results were examined. This package requires localization information of the robot. To obtain the localization information the Augmented Monte Carlo Localization (AMCL) method, which is an adaptive version of MCL, was used.

In the experiments, the real robot platform is used and the results are tested with different parameters to optimize.

Key words: Search and rescue robots, mapping algorithms, gMapping, localization, Augmented Monte Carlo Localization, ROS (The Robot Operating System) Hydro, laser scan matching, robot platforms

BÖLÜM 1

GİRİŞ

1.1 Literatür Özeti

Arama kurtarma robotlarının barındırması gereken çeşitli özellikler; konum ve duruş kestirimi, haritalama, navigasyon, keşif ve kurban tespiti gibi başlıklar altında incelenebilir.

Robotun konumunun kestirimi için farklı yöntemler kullanılmaktadır. Bunlardan en temeli robotun tekerleklerine bağlanan bir encoder aracılığıyla tekerlek odometrisi bilgisinin üretilmesidir [1]. Bu sistemlerde tekerleğin döndüğü kadar robotun gittiği kabul edilir. Bu varsayım her zaman gerçekleşmeyeceği için gürültülü bir odometri verisi üretilmektedir. Bu sebeple literatürde çokça kullanılan bir yöntem değildir. Bir diğer konum kestirim yöntemi lazer taramalarının birbirleri ile eşlenmesi ile gerçekleştirilir [2], [3], [4], [5]. Bu yöntemlerde bir lazer mesafe ölçüm duyargası yardımı ile ardışık zaman aralıklarında alınan taramalar birbirleri ile eşleştirmeye çalışılarak robotun translasyonu ve rotasyonu ile ilgili bilgi çıkarımı gerçekleştirilmektedir. Odometri çıkarımında sıkça kullanılan diğer bir yöntem ise eş zamanlı konum belirleme ve haritalama (SLAM) algoritmaları ile yapılmaktadır [6], [7], [8], [9]. Bu yöntemlerde ortamın haritasının çıkarımıyla birlikte, konum tahmini de yapılmaktadır. Hesap karmaşıklığı diğer yöntemlere göre fazla olsa da özellikle hataların global olarak giderilmesinden dolayı, bu yöntemlerle odometri çıkarımı oldukça efektiftir. Bunların dışında literatürde görsel verilerden odometri çıkarımı gerçekleştirilmesiyle ilgili çalışmalar da vardır.[10], [11], [12]. Bu yöntemlerde, Kinect gibi 3 boyutlu ortam görüntüsü elde edebilen kameralar yardımı ile ardışık zamanlarda

ortamın görüntüsü alınarak birbirleri ile eşleştirilmeye çalışılır. Bu eşleştirme sonucunda robotun pozundaki değişim ile ilgili bir hipotez ortaya atılır ve odometri çıkarımı gerçekleştirilmiş olur.

Bu konuda ele alınması gereken bir diğer başlık robotun lokalizasyonudur (konum ve duruş kestirimidir). Literatürde sıkça robotun pozisyon takibi ile ilgili problemler üzerinde durulmuştur [13], [14], [15]. Bu problemde robotun başlangıç konumu bilinmektedir ve gezinimi esnasında odometri üzerinde oluşacak artımsal hataların kompanse edilmesi sağlanmaya çalışılmaktadır. Özellikle global lokalizasyon yöntemleri yaygınlaştıktan sonra bu yöntemlerin kullanımı azalmıştır. Global lokalizasyon yöntemlerinde ise robotun başlangıç konumu bilinmeksizin konum ve duruş kestiriminin gerçekleştirilmesi yapılmaktadır [16], [17], [18]. Bu yöntemler robotun pozisyonunun bilindiği yöntemlere göre daha karmaşıktır. Bu başlık altında robotun kaçırılması problemini de çözmeye çalışan yöntemler incelenebilir [19].

Haritalama problemine gelindiğinde ise literatürde eş zamanlı konum belirleme ve haritalama yöntemleri (Simultaneous Localization and Mapping – SLAM) olarak incelendiği görülmektedir [6], [8], [20], [21], [22]. Bu yöntemlerde, mobil robot ortamı gezerken, eş zamanlı olarak (gerçek zamanlı) ortamın haritasını çıkartması amaçlanmıştır. Eş zamanlı olması metotların donanım bağımlı olmasını da beraberinde getirmekle birlikte diğer yandan efektif ve performansı daha iyi yöntemlerin de ortaya çıkmasına vesile olmuştur. gridMapping [6], [7] ve hectorMapping [8] bu yöntemlerden sıkça kullanılanı ve bu çalışma kapsamında üzerinde çalışılan, bir framework olan, ROS (The Robot Operating System) ortamında gerçekleştirilmiş olanlarıdır. Bunların yanı sıra FastSLAM [23] ve Extended Kalman Filter (EKF) [24], [25] gibi yöntemler de bu başlık altında incelenebilmektedir.

Mobil robotlarda navigasyon problemi, üzerinde çokça durulan problemlerden birisidir. Temelde robotun verilen hedefe otonom bir şekilde gitmesini amaçlayan navigasyonun gerçekleştirilmesi için literatürde farklı yaklaşımlar bulunmaktadır. Efektif landmark seçimi [26] ile navigasyonun gerçekleştiği metotların daha hızlı ve güvenilir sonuçlar verdiği tespit edilmiştir. Bunun yanı sıra bir dönem grid tabanlı haritalar yerine topolojik haritalar üzerinde çalışılarak navigasyon gerçekleştirilmeye çalışılmıştır [27].

Yine, ortamın görsellerinin birbirleri ile eşleştirilmesi ile zemin tespiti yapılarak navigasyonun gerçekleştirilmeye çalışıldığı yöntemler de literatürde mevcuttur [28].

Eksplorasyon (Exploration) kısmına gelindiğinde ise mobil robotun bilinmeyen bir ortama gidişi ile ilgili yol planlaması beklenmektedir. Ayrıca eksplorasyon, aynı zamanda kısmen navigasyon problemini de içerisine almaktadır. Bu hedefi gerçekleştirebilmek için literatürde çok farklı yöntemler kullanılmıştır. En temelde bilinmeyen bir ortamda hedef seçilmesi ve bu hedefe diferansiyel uzaklığın belirlenerek eksplorasyonun gerçekleştirildiği yöntemler [29] yatmaktadır. Bu yöntemlerin bir kısmında bilinmeyen ortamlardaki hedef tespiti için ayrık zamanlı Markov işlemleri kullanılmaktadır [30]. Bazı yöntemlerde ise karınca ve arı kolonileri (Particle Swarm Optimization - PSO) gibi biyolojik olaylardan esinlenerek çok sayıda robotun, birbirleri ile haberleşerek belirli bir alanı gezmeleri incelenmektedir [31].

Arama kurtarma amacının gerçekleşebilmesi için bu tarz robotların barındırması gereken en önemli özellik ise kurban/kazazede tespiti için kullanılan yöntemlerdir. Bu yöntemler birkaç farklı başlık altında incelenmektedir. İnsan anatomisini inceleyerek, insan vücudunun parçalara ayrıldığı ve her bir parçanın özelliklerinin çıkarılarak insan vücudunun tespit edildiği yöntemler [32], [33] literatürde kullanılmıştır. Bu yöntemlerin içerisinde hareket tespiti algoritmalarının da kullanıldığı görülebilmektedir. Bununla beraber lazer mesafe ölçüm duyargası kullanılarak yine insan vücudunun belirli bölgelerinin özellikleri üzerinden tespitinin gerçekleştirildiği yöntemler [34], [35] ve termal kameralar yardımı ile insan vücudunun tespit edildiği yöntemler [36] literatürde rastlanılan diğer kurban tespiti yöntemlerindendir.

Bu çalışma kapsamında robot platformlarında kullanılmak üzere ROS (The Robot Operating System) framework'ü üzerinde gerçeklenmiş metotlar ve yöntemler kullanılmıştır. ROS, bilinen manada bir işletim sistemi değildir. Heterojen hesaplama kümeleri üzerinden iyi yapılandırılmış haberleşme katmanı sağlayan bir çatıdır [37], [38]. ROS kullanılmadığı durumlarda soket programlama gibi haberleşme alt yapısıyla ilgili gereksinimleri kullanıcının düzenlemesi gerekirken, ROS framework'ü bu tarz gereksinimleri kendi içerisinde karşılayarak kullanımı daha kolay bir sistem sunmaktadır.

ROS platformuna uyumlu olarak geliştirilen ve hazır platform olarak elde edilebilen robot sistemleri de vardır. Bunlara PR2 [39], TurtleBot [40] ve P3AT [41] örnek verilebilir. PR2, Willow Garage gurubunun geliştirdiği diğerleri arasında en gelişmiş robot platformudur denilebilir. Kameranın konumlandırıldığı insan başına benzer hareketli bir platforma ve hareketli kollara da sahiptir. Diğer robot türlerine göre serbestlik derecesi daha fazladır. TurtleBot platformu 3 tekerlekli, üzerinde Kinect kamera barındıran farklı özellikler eklenerek gelişmeler yapılabilecek bir platformdur. P3AT ise diferansiyel sürüşlü 4 tekerlekli, üzerinde farklı grupların birçok araştırma yaptığı bir robot platformudur.

Daha önce de bahsedildiği gibi bu tez çalışması kapsamında ROS platformu üzerinde gerçekleştirilmiş yöntemler üzerinde durularak bir sistem oluşturulmuştur.

1.2 **Tezin Amacı**

Arama kurtarma robotları doğal afetlerde veya kazalarda enkaz alanlarına müdahalede kullanılmak için geliştirilen robot sistemleridir. Farklı alanlarda kullanılmak için özelleşmiş modelleri bulunsa da genel olarak her birinde bulunması gereken çeşitli özellikler mevcuttur. Bunlar, robotun hareket modeli, konum ve duruş kestirimi, haritalama ve otonom gezinim problemleridir. Bu problemin çözümü için ortaya atılan algoritmalar lazer mesafe ölçüm duyargası, atalet duyargası ve görüntüleme sistemleri gibi çeşitli algılayıcılara ihtiyaç duymaktadırlar. Ortama dair bilgilerin edinilmesinde kullanılan bu duyargalar farklı modellerde ve özelliklerde olabilmektedir. Ancak, robot platformlarının geliştirilmesinde, bu özelliklerden bağımsız olarak çalışabilen yöntemler tasarlanmaktadır. Bunun için üretilen çözümler genelleştirilmiş ve farklı platformlarda da çalışabilmek üzere geliştirilmiştir.

Yapılan bu çalışmada, Yıldız Teknik Üniversitesi Bilgisayar Mühendisliği Bölümü Olasılıksal Robotik Grubu'nun geliştirmekte olduğu arama kurtarma robot sistemi için bir platform tasarımı gerçekleştirilmiştir. ROS (The Robot Operating System) Hydro sürümü üzerinde gerçeklenen platformda, haritalama, konum kestirimi ve navigasyon gibi problemler için çeşitli yöntemlerin denenmesi ve deneysel sonuçların elde edilmesi amaçlanmıştır.

Tez çalışmasında Bölüm 2 ile bir parçası olunan Olasılıksal Robotik Grubu'nun her yıl katıldığı RoboCup yarışmaları tanıtılmıştır. Bölüm 3 ile tasarlanan robot platformunun ve kullanılan duyargaların özelliklerine ve şekillerine yer verilmiştir. Bölüm 4'te platformun üzerinde çalıştırılan bir framework olan ROS ortamına, tarihçesine ve özelliklerine değinilmiştir. Bölüm 5 ile konum kestirimi için kullanılan tarama eşleme tabanlı Laser_Scan_Matcher (LSM) yöntemi ele alınmış ve deneysel sonuçları elde edilmiştir. Bölüm 6'da farklı kaynaklardan elde edilen odometri bilgilerinin birleştirilmesi ile daha iyi odometri bilgisi üretmek için kullanılan Robot_Pose_EKF yöntemi ve deneysel sonuçları ele alınmıştır. Bölüm 7'de eş zamanlı konum kestirim ve haritalama (SLAM) yöntemlerinden olan gridMapping (gMapping) algoritmasına ve deneysel sonuçlarına değinilmiştir. Bölüm 8'de gezinge ve konum kestirim yöntemlerinde mesafe ve atalet duyargalarının kullanımına, Bölüm 9'da ise navigasyon için gerekli olan lokalizasyon (konum ve duruş) bilgisinin üretimi için kullanılan Monte Carlo lokalizasyonunun (MCL) adaptif bir versiyonu olan Augmented MCL yöntemine yer verilmiştir. Bölüm 10 ile robotun kısmi otonom gezinimi için kullanılan ROS üzerinde gerçeklenen "navigation" paketinin ayrıntıları ve deneysel sonuçları incelenmiştir. Bölüm 11'de ise tüm yöntemlerde elde edilen sonuçlar özetlenmiştir.

1.3 **Hipotez**

Tez çalışması kapsamında arama kurtarma amaçlı bir robot platformu tasarlanarak, bunun üzerinde konum kestirimi, haritalama ve navigasyon gibi problemler için çeşitli algoritmalar denenmiştir. Deneylerde kullanılan yöntemler farklı parametreler için denenerek farklı ortamlar için farklı sonuçlar elde edildiği gözlemlenmiş ve bu sebeple ortama bağımlı olarak farklı parametrelerin kullanılması önerilmiştir. Gezinge kestirim yöntemlerinin kıyaslanmasında bir model ortaya atılmış ve kıyaslamalar bunun üzerinden yapılmıştır. ROS framework'ü robot platformları için efektif sonuçlar üreten uygun bir iskelet yapısıdır. Bu bağlamda, üzerinde gerçekleştirilen paketlerin analizinin ne şekilde olduğu test edilmiş ve değerlendirmelerde bulunulmuştur.

Tez çalışmasını yönlendiren temel hipotez şu şekilde verilebilir: "Arama kurtarma amaçlı geliştirilen bir robot sistemi, hareket modeli, konum ve duruş kestirimi, haritalama ve navigasyon gibi özellikler barındırmalıdır."

BÖLÜM 2

ROBOCUP YARIŞMALARI TANITIMI

RoboCup, 1997'den beri her yıl düzenlenen uluslararası robotik yarışmalarıdır. Yarışmanın amacı halka açık, cazip, ancak zorlu bir ortam oluşturarak yapay zekâ ve robotik araştırmalarını desteklemektir. RoboCup ismi "Robot Soccer World Cup" yarışmasının ismi kısaltılarak oluşturulmuştur. Ancak yarışma kapsamında futbol oynayan robotların yanı sıra birçok farklı ligde yarışan robot platformları da vardır. Alt dalları olmakla beraber yarışmalar genel olarak 4 ana başlık altında yapılmaktadır [42]

- RoboCup Soccer: Farklı boyutlarda ve özelliklerde futbol oynayan robotların yarıştıkları liglerin ana başlığıdır.
- RoboCup Rescue: Gerek simülasyon, gerekse gerçek robot branşlarında arama kurtarma robotlarının yarıştığı liglerin ana başlığıdır (bu tez çalışmasına da konu olan robot platformu, bu başlık altında gerçek robot liginde yarışmaktadır).
- RoboCup@Home: İnsan gibi sosyal olabilmek yolunda ilerleyen ve insanların yaşadığı ortamlarda, onların başa çıktığı sorunlarla başa çıkmaya çalışan robotların yarıştığı ligin ana başlığıdır.
- RoboCupJunior: Diğer 3 branşta yapılan yarışmaların belirli bir yaş grubu öğrenciler için yapıldığı liglerinin ana başlığıdır.

Yukarıda da bahsedildiği gibi bu teze de konu olan robot platformu, RoboCup Rescue liginde yarışmak için tasarlanmaktadır. Bu bağlamda, robotun geliştirilmesi sırasında yarışmalarda uygulanan çeşitli kısıtlar ve uygulamalar esas alınmıştır. Mesela yarışma ortamında eğimli yüzeyler 15° ve 30° olmak üzere sadece 2 farklı açıda olmaktadır. Aynı

şekilde koridorların genişliği ve yüksekliği 120 cm'dir ve koridor duvarları suntadan yapılmıştır. Geliştirme sırasında robotun üzerinde bulunan duyargalar da bu kriterler dikkate alınarak temin edilmiştir. Tezin ilerleyen kısımlarında bahsedileceği gibi bu çalışmada tekerlek odometrisi ve görsel odometri robotun gezinimi için kullanılmamıştır. Tezin anlatımı esnasında RoboCup Rescue ligine referans gösterilerek kısıtlamalara ilgili başlık altında değinilecektir. Genel olarak RoboCup yarışmalarını simgeleyen futbol oynayan bir robot görseline Şekil 2.1'de yer verilmiştir.

Şekil 2.1 RoboCup yarışmalarını simgeleyen futbol oynayan bir robot

Bu yarışmalar her yıl farklı bir ülkede gerçekleştirilmektedir. Yarışmaların yanı sıra, bunlara hazırlık için takımların, birbirlerinin durumlarını görebilecekleri hazırlık ligleri de yapılmaktadır. Yine bunlar da farklı ülkelerin üstlenmesi ile gerçekleştirilmektedir. 2011 yılında gerçekleştirilen RoboCup yarışmalarına, Boğaziçi Üniversitesi aracılığıyla Türkiye ev sahipliği yapmıştır. Yarışmanın kuralları, the American National Institute of Science and Technology (NIST) tarafından hazırlanmakta ve sürekli geliştirilmeye çalışılmaktadır [43]. Yarışmaların zorluk dereceleri her yıl arttırılmaktadır. Bu sayede takımların üzerinde çalıştıkları robotlara ekstra özellikler eklemeleri ve bu sayede araştırma geliştirme faaliyetlerine katkıda bulunmaları amaçlanmaktadır.

2.1 **RoboCup Rescue League (RoboCup Arama-Kurtama Ligi)**

RoboCup Arama-Kurtarma Ligi, deprem gibi afet ortamlarının simüle edildiği yerlerde kazazedeleri bulmak için robot platformlarının yarıştığı uluslararası bir ligdir. Rescue Simulation League (RSL) ve Rescue Robot League (RRL) olmak üzere 2 farklı dalda

gerçekleştirilmektedir. RSL, tamamen simülasyon ortamında gerçekleştirilirken, RRL'de gerçek robotlar ve test alanları kullanılmaktadır. 2009 yılından beri her iki lig de RoboCup yarışmalarının bir parçası olmuştur.

Bu yarışmada yarışan robotlar, gezdikleri ortamlarda buldukları kazazedeleri simgeleyen cisimlerden, gezdikleri ortama ait çıkarttıkları haritalardan ve kazazedeleri harita üzerine işaretleyebilmelerinden puan almaktadırlar. Yarışmanın belirli aşamalarında robotların otonom gezmeleri, belirli aşamalarında ise bir operatör yardımı ile gezmeleri istenmektedir. Aynı şekilde yarışma kapsamında hazırlanan ortamlar birkaç zorluk derecesinden oluşmaktadır. Zorluk dereceleri daha çok ortamın engellerinin fazlalığı ve düzensiz olmasıyla sağlanmaktadır. Yarışma sırasında her bir takıma standart olarak 20 dakika süre verilmekte ve bu 20 dakika sürede hazırlanan ortamı gezmeleri ve kazazedeleri tespit etmeleri istenmektedir. Yarışmalarda kullanılan ortamın bir örneğine Şekil 2.2'de yer verilmiştir.

Şekil 2.2 RoboCup Rescue liginde 2008 yılında kullanılan bir ortam

Ortam ve yarışma alanı özellikleri, robot platformunda olması gereken özellikler ve yarışmanın değerlendirme kriterleri detaylı olarak [44]'de bulunabilir. Arama kurtarma liginde yarışan bazı robotların görsellerine Şekil 2.3'te yer verilmiştir.

Şekil 2.3 RoboCup Rescue liginde yarışan robotlardan görseller

2.2 **Başarılarımız**

RoboCup arama kurtarma ligleri kapsamında Yıldız Teknik Üniversitesi Bilgisayar Mühendisliği Bölümü Olasılıksal Robotik Grubu olarak aldığımız çeşitli dereceler aşağıdaki gibidir:

- 2012 İran RoboCup Open (RSL)- ***İkincilik***
- 2012 Meksika RoboCup (RSL) - ***İkincilik***
- 2013 İran RoboCup Open (RSL) - ***Birincilik***
- 2013 Hollanda RoboCup (RSL) – ***İkincilik***

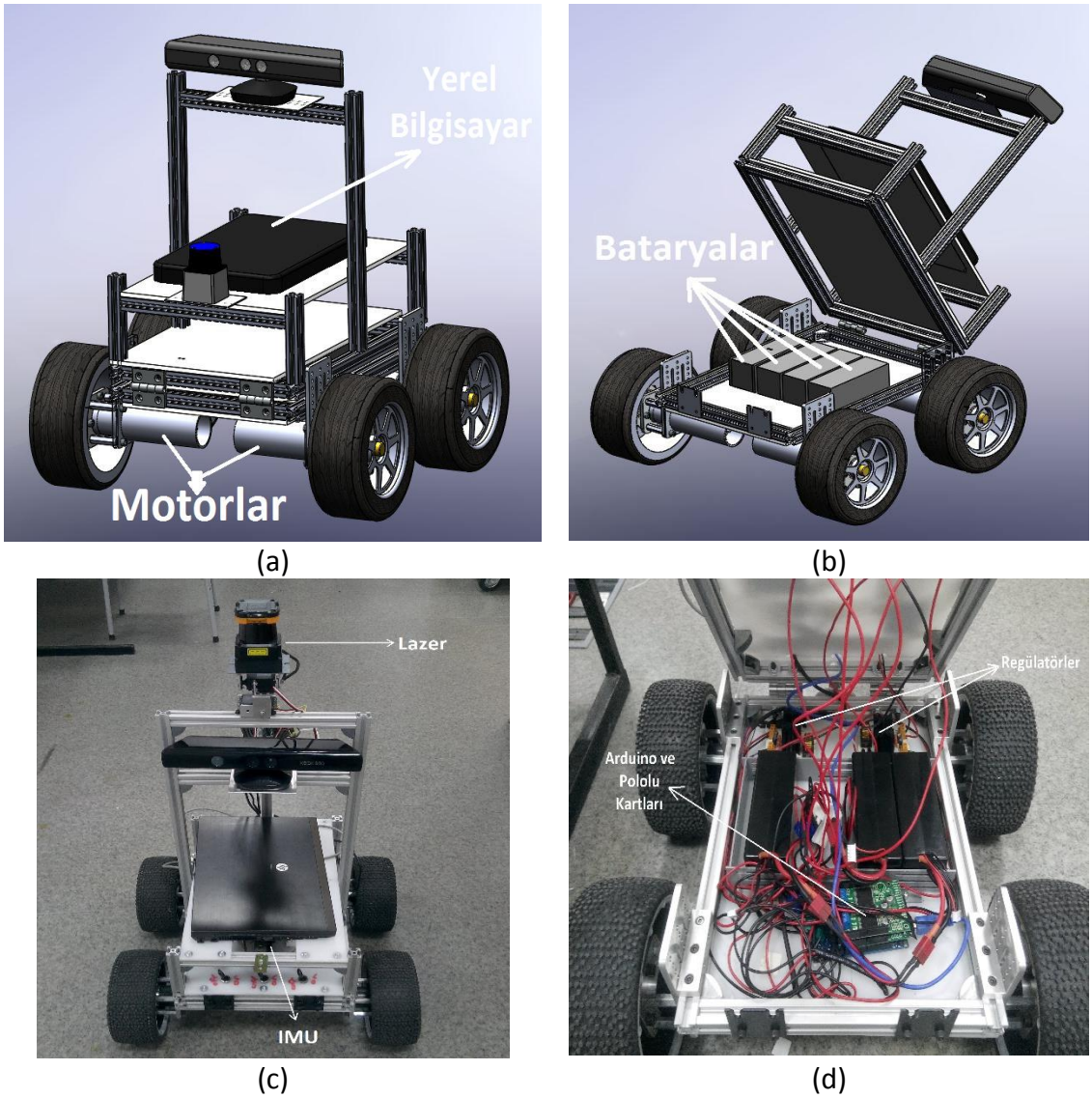
BÖLÜM 3

ROBOT PLATFORMU

Bu tez ile arama kurtarma amacıyla kullanılmak üzere 4 tekerlekli, diferansiyel sürüşlü bir robot platformu geliştirilmiştir. Robotun tekerlerine bağlı olan motorlar birbirinden bağımsız çalışabilecek şekilde tasarlanmıştır ve 12V'luk bir gerilimle beslenmektedir. Motorların ucuna 14 cm çapında lastik tekerlekler takılmıştır. Robot, birbirine bağlantısı kolay olan profiller yardımı ile oluşturulmuştur. Tekerleklerin bağlı olduğu gövdenin orta kısmında açılır kapaklı bir alan yapılmış, bu alanın, robotun hareketi ve kontrolü için kullanılan entegreleri, kartları, regülatörleri ve bataryaları barındırması amaçlanmıştır. Bu alanın hemen üstünde, atalet duyargasının (IMU) ve USB Hub'ın yerleştirilebileceği bir alan oluşturulmuş ve onun üstünde de robot sistemini kontrol edecek yerel bilgisayarın yerleştirilebileceği bir alan yapılmıştır. Lazer mesafe ölçüm duyargası ise robotun en üst noktasına yerleştirilerek 270°'lik açılarla tarama yapabilmesi sağlanmıştır. Hazırlanan robot platformunun görselleri Şekil 3.1'de verilmiştir.

Robot platformunu oluşturan parçaların bağlantılarını gösteren bir şemaya Şekil 3.2'de yer verilmiştir. Buradan da görülebileceği gibi birisi yerel, birisi uzak bilgisayar olmak üzere 2 farklı bilgisayar kullanılmıştır. Yerel bilgisayarda robotun çalışmasını sağlayacak düğümler çalıştırılmakta, uzak bilgisayarda ise robotun çıkarttığı haritanın ve izlediği gezinimin görüntülenebileceği "rviz" aracı çalıştırılmaktadır. Aynı zamanda "rviz" aracı üzerinden robota gerçekleştirmesi için çeşitli komutlar da verilebilmektedir. Robotun kontrolü için kullanılan bir diğer araç ise bir joystick'tir. Otonom gezmediği durumlarda

robotun gezinimi bu kol yardımı ile yapılmaktadır. Kol üzerinden farklı hız bilgileri ile robota komutlar gönderilerek, robotun hızının değişmesi sağlanabilmektedir.



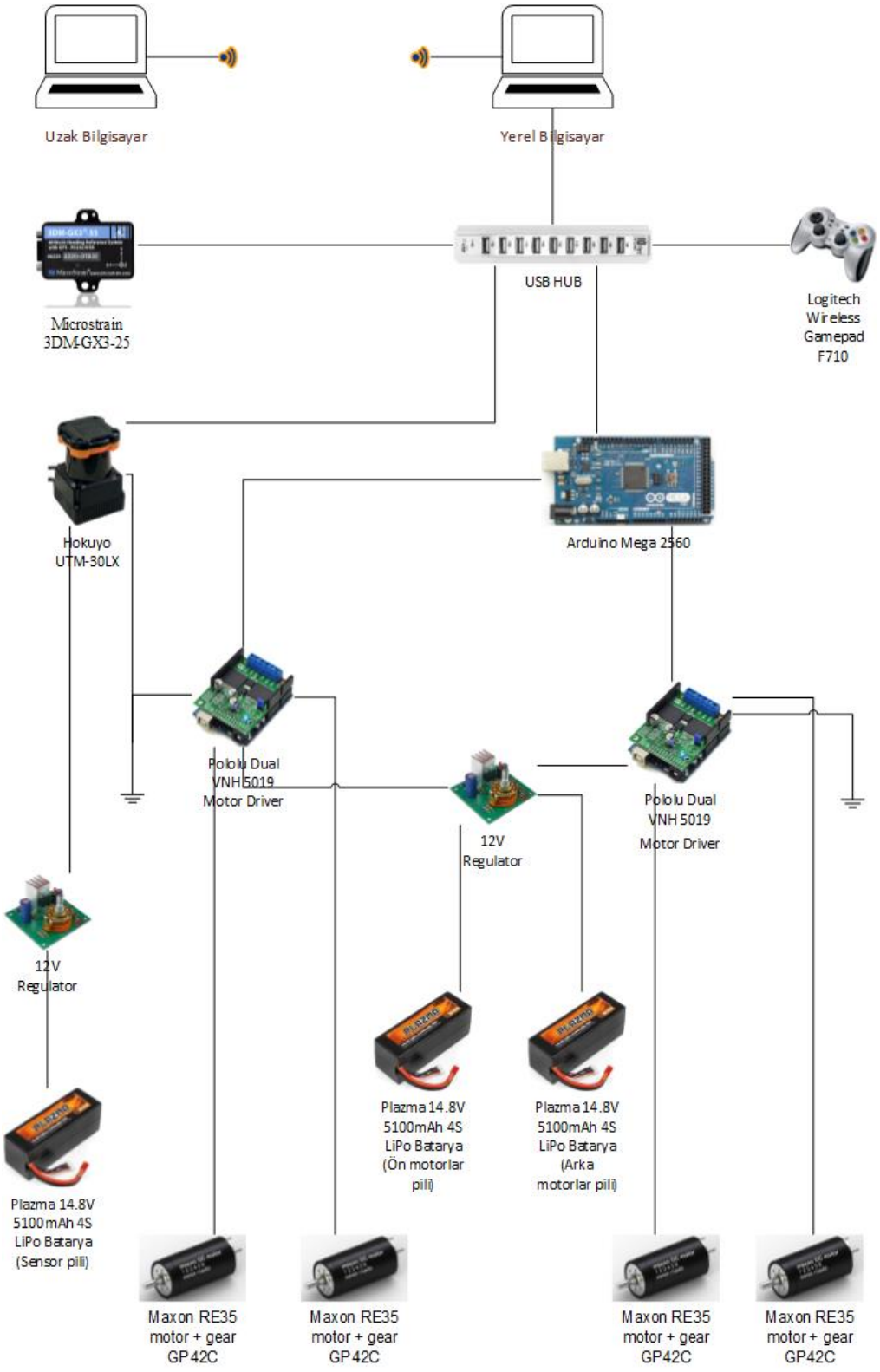
(a) (b)

(c) (d)

Şekil 3.1 Robot platformları görselleri

Platform üzerinde 2 adet Pololu shield kullanılmıştır. Bunların birisi 2 tane ön tekerleği sürmekte ve 12V'luk bir gerilimle beslenmekte, diğeri de 2 tane arka tekerleği sürmekte ve aynı şekilde farklı bir bataryadan 12V'luk gerilimle beslenmektedir. Seri port haberleşmesi Arduino kartı üzerinden gerçekleştirilmektedir. Lazer mesafe ölçüm duyargası ise, motorları besleyen bataryalardan hariç, farklı bir batarya ile 12V'luk bir gerilimle beslenmekte ve doğrudan USB Hub yardımı ile yerel bilgisayara bağlanmaktadır. Atalet duyargası ise doğrudan USB Hub üzerinden beslenmekte ve ek gerilime ihtiyaç duymamaktadır. Daha önce bahsedildiği gibi robot, ROS platformunda

geliştirilmiştir. ROS, Linux tabanlı bir çatıdır. Bu bağlamda bilgisayarlar üzerine Ubuntu 10.04 LTS işletim sistemi kurularak ROS ortamı hazırlanmıştır.



Şekil 3.2 Robot platformu bağlantılarının şematize edilmiş hali

Bu aşamadan sonra robot platformu üzerinde kullanılan, Şekil 3.2'de de gösterilen, donanımların özelliklerine detaylı olarak değinilmiştir.

3.1 **Yerel ve Uzak Bilgisayarlar (Toshiba Portege R830-137)**

Çalışma kapsamında biri yerelde diğeri uzakta kullanılmak üzere 2 adet Toshiba dizüstü bilgisayar kullanılmıştır. Bu bilgisayarların özellikleri Çizelge 3.1'de, görüntüsü ise Şekil 3.3'te verilmiştir.

Çizelge 3.1 Yerelde ve uzakta kullanılan bilgisayarın özellikleri

Toshiba Portege R830-137 Özellikleri	
Boyutu	16:9 en-boy oranı
İşlemci türü	Intel® Core™ i7-2620M İşlemci
RAM özellikleri	4 GB - 1333 MHz – DDR3
Grafik kartı	Intel® HD Graphics 3000
Disk özellikleri	256 GB
Ağırlığı	1,48 kg

Şekil 3.3 Yerelde ve uzakta kullanılan bilgisayarın görüntüsü

Yukarıdaki özellikleri verilen bilgisayar gerek işlem yükünü karşılayabilmesi, gerekse hafızasının miktarı bakımında tez çalışması kapsamında kullanılabilecek yeterli düzeyde donanımı bir bilgisayardır. Tezde kullanılan yöntemlerin gerçek zamanlı çalışabilmesi için sisteme ne kadar bağımlı olduğuna yöntemler anlatılırken değinilecektir.

3.2 **Lazer Mesafe Ölçüm Duyargası (Hokuyo UTM-30LX)**

Tez çalışması kapsamında 30 metreye kadar 270°'lik açıyla ölçüm yapabilen Hokuyo UTM-30LX lazer mesafe ölçüm duyargası kullanılmıştır. Duyarganın özelliklerine Çizelge 3.2'de, görseline ise Resim 3.4'te yer verilmiştir.

Çizelge 3.2 Lazer mesafe ölçüm duyargasının özellikleri

Hokuyo UTM-30LX Özellikleri	
Güç kaynağı	12V DC ± %10 (0.7A)
Işık kaynağı	Yarı iletken lazer diyot (λ=905nm)
Tarama mesafesi ve açısı	0.1-30 metre, 270°
Tarama doğruluğu	0.1 ila 10m:±30mm, 10 ila 30m:±50mm
Açısal çözünürlüğü	0.25°
Tarama frekansı	25 ms/tarama
Gürültü seviyesi	25 dB'dan daha az
Ara yüz	USB 2.0
Çalışma koşulları	-10 ila 50 derece, maksimum %85 nem
Ağırlığı	370 g.

Şekil 3.4 Lazer mesafe ölçüm duyargası görseli

Tasarlanan bu robot, RoboCup Rescue liginde yarışmak için geliştirilmektedir. Yarışma ortamının özellikleri gereği 30 metreye kadar 270°'lik açıyla tarama yapabilen lazer mesafe ölçüm duyargası yeterli özelliklere sahiptir.

3.3 **Atalet Duyargası (IMU - Microstrain 3DM-GX3-25)**

Robotun üzerinde dijital bir pusula olarak nitelendirilebilecek bir adet atalet duyargası (Inortia Measurement Unit - IMU) bulunmaktadır. IMU, açısal momentum prensibine dayanarak robotun oryantasyonunu ölçen bir cihaz olan gyroscope'u, hızlanmanın miktarını ölçen bir ivmeölçeri (accelerometer) ve manyetik alanın yönünü tespit eden bir magnetometer'ı barındırmaktadır. Bu özellikleri bir arada barındıran sistemlere attitude and heading reference system (AHRS) sistemleri denmektedir. IMU da küçük bir AHRS sistemidir. 3 eksende ivme miktarını, oryantasyon değişim miktarını ve manyetik açısal değişimin miktarını verebilmektedir. İçerisindeki farklı duyargalardan okuduğu bilgileri füzyon ederek quaternion tipinde bir poz bilgisi üretmektedir. IMU'nın özelliklerine Çizelge 3.3'te, görseline ise Şekil 3.5'te yer verilmiştir.

Çizelge 3.3 Atalet duyargasının özellikleri

IMU - Microstrain 3DM-GX3-25 Özellikleri	
Kafa açısı mesafesi	3 eksende 360°
İvmeölçer mesafesi	±5
Gyroscope mesafesi	±300°/sn.
Çözünürlük	<0.1°
Veri üretim frekansı	1 kHz'e kadar
Ara yüz	USB 2.0
Güç kaynağı	+3.2 ila +16 Volt DC
Çalışma koşulları	-40° C to +70° C
Ağırlığı	18 g.

Şekil 3.5 Atalet duyargası görseli

3.4 **Joystick (Logitech Wireless Gamepad F710)**

Robotun otonom gezmediği durumlarda uzaktan kontrolünü gerçekleştirebilmek için bir joystick kullanılmıştır. 2 adet AA pil ile çalışan bu uzaktan yönetme kolu, açık alanda 100 metreye kadar kablosuz olarak iletişim kurabilmektedir. Yerel bilgisayara kablosuz bir USB ucuyla bağlanmaktadır. Hem analog hem de dijital değerler üretebilen butonlar barındıran bu joystick için ROS üzerinde sürücü yazılmıştır. Bu bakımdan ROS platformu üzerinde rahatça kullanılabilmektedir. Joystick'in bir görseline Şekil 3.6'de yer verilmiştir.

Şekil 3.6 Joystick görseli

Tuşlar kullanıldığında, ROS üzerinde ne şekilde değerler üretildiğini gösteren bir çizelge EK A'da verilmiştir.

3.5 **Kontrol Kartı (Arduino Mega 2560)**

Arduino, bir mikro denetleyicidir. Kolay bir şekilde çevresiyle etkileşime girebilen sistemler tasarlanabilecek açık kaynaklı bir geliştirme platformudur. Analog ve dijital girişleri sayesinde analog ve dijital verileri işlenebilmektedir. Duyargalardan gelen verileri kullanılarak, dış dünyaya çıktı (ses, ışık, hareket vs.) üretilebilmektedir [45]. Arduino ile bir çok farklı spesifik uygulama yapılabilmekle beraber bu çalışma kapsamında, robot süren Pololu shield'ları kontrol etmek amacıyla kullanılmaktadır. Pin bağlantıları Pololu shield anlatılırken gösterilecektir. Şekil 3.7'de kullanılan Arduino Mega 2560'a ait bir görsele yer verilmiştir.

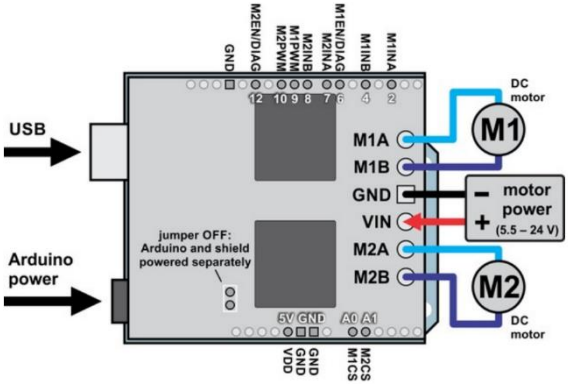
Şekil 3.7 Arduino Mega 2560 görseli

3.6 **Motor Sürücü (Pololu Dual VNH5019 Motor Driver)**

Motorların kontrolünü sağlamak amacıyla Pololu shield'lar kullanılmıştır. Her bir shield 2 motor sürebilmektedir. Robotun 4 motoru olduğu için 2 tane Pololu kart (İlki ön motorları sürmek için, ikincisi ise arka motorları sürmek için) kullanılmıştır. Pololu kartın Arduino üzerine oturtulmuş şekliyle bir görseline Şekil 3.8'de yer verilmiştir.

Şekil 3.8 Pololu Dual VNH5019 Motor Driver görseli

Pololu kartın motor ve gerilim bağlantılarını şematize eden bir görsele Şekil 3.9'da yer verilmiştir. Şekilden de görülebileceği gibi her bir kart 2 motor sürebilmektedir. Pin bağlantılarına ise Çizelge 3.4'te yer verilmiştir.



Şekil 3.9 Pololu Dual VNH5019 Motor sürücünün Arduino bağlantılarının görseli

Çizelge 3.4 Pololu Dual VNH5019 Motor sürücünün Arduino pin bağlantıları

Arduino Pin	**VNH5019 Driver Pin**	**Basic Function**
Digital 2	M1INA	Motor 1'den A girişine bağlantı
Digital 4	M1INB	Motor 1'den B girişine bağlantı
Digital 6	M1EN/DIAG	Motor 1 enable girişi/hata çıkışı
Digital 7	M2INA	Motor 2'den A girişine bağlantı
Digital 8	M2INB	Motor 2'den B girişine bağlantı
Digital 9	M1PWM	Motor 1 hız girişi
Digital 10	M2PWM	Motor 2 hız girişi
Digital 12	M2EN/DIAG	Motor 2 enable girişi/hata çıkışı
Analog 0	M1CS	Motor 1 akım algı çıkışı
Analog 1	M2CS	Motor 2 akım algı çıkışı

3.7 **Motorlar (Maxon RE35 motor + gear GP42C)**

Daha önceden de bahsedildiği gibi robot üzerinde 4 adet DC motor kullanılmıştır. Her bir tekerleği farklı motor kontrol etmektedir. Kullanılan motorun modeli "Maxon RE 35 Ø35 mm, Graphite Brushes, 90 Watt" tır. Çizelge 3.5'te motorun özelliklerine, Şekil 3.10'da motorun bir görseline yer verilmiştir.

Şekil 3.10 Maxon RE35 motor görseli

Çizelge 3.5 Maxon RE35 motor özellikleri

Maxon RE35 motor Özellikleri	
Çalışma gerilimi	12V
Nominal hız	6500 rpm (≈ 300 cm/sn)
Maksimum hız	12000 rpm (≈ 500 cm/sn)
Nominal tork	73.1 mNm
Nominal akım	4A
Çalışma koşulları	-30 ila +100 °C arası
Maks. bobin sıcaklığı	+155 °C
Genişliği	3.5 cm
Ağırlığı	340 g.
Dönüş yönü	Saat yönü

Özellikle motorun hız bilgileri incelendiğinde 300 cm/sn hıza kadar nominal hız değeri olduğu görülmektedir. Ancak motorların taşıdığı aracın ağırlığı arttıkça bu değer oldukça azalmaktadır. Robotun gezdiği ortamın işlenebilir lazer taramasını alabilmesi için tespit edilen maksimum hız değeri 50 cm/sn olmuş ve robot sistemi tasarlanırken bu şekilde maksimum hız bilgisi sınırlandırılmıştır.

3.8 **Regulator (12V)**

Robot üzerinde kullanılan bataryalar 14.8V'a kadar gerilim üretmektedir. Donanımlar ise 12V'luk gerilimde çalışmakta, maksimum %5'lik bir hata payını tolere edebilmektedir. Bu sebeple bataryalardan alınan gerilimler bir regülatör kullanılarak 12V'a düşürülmektedir. Her bir regülatör, 2 girişe ve 2 çıkışa sahip olduğu için ve sistemin tasarımında 3 adet 12V'a ihtiyaç olduğu için (2 tane Polulu shield için, 1 tane de lazer mesafe ölçüm duyargası için) toplamda 2 tane regülatör kullanılmıştır. Bu regülatörler, 2A'ya kadar destekleyen direnç ayarlıdırlar. Bu sebeple kullanılan donanımlarda uyumlu olarak çalışabilmektedirler. Bir regülatör görseline Şekil 3.11'de yer verilmiştir.

Şekil 3.11 12V'luk regülatör görseli

3.9 **Batarya (Plazma 14.8V 5100mAh 4S LiPo)**

Robot platformunu beslemesi için 3 adet 14.8V-5100 mAh'lık 4 hücreli lityum polimer batarya kullanılmıştır. Beslenmesi gereken donanımlar 12V'luk gerilime ihtiyaç duydukları için bu bataryalar yeterli düzeyde gerilim üretebilmektedirler. Yeniden doldurulabilme özelliğine sahip bu bataryaya ait bir görsele Şekil 3.12'de yer verilmiştir.

Şekil 3.12 Plazma 14.8V 5100mAh 4S LiPo görseli

BÖLÜM 4

ROS (THE ROBOT OPERATING SYSTEM)

ROS (The Robot Operating System), robotlar üzerinde çalışan yazılımlar geliştirmek için hazırlanmış esnek yapılı bir framework'tür. ROS, yazılımları gerçekleştirmek için kullanılabilecek çeşitli araçların, kütüphanelerin ve düzenlemelerin bir araya gelmesiyle karmaşık görevlerin basitleştirilerek güçlü robot davranışlarının birçok farklı robot platformu üzerinde başarıyla kurulabilmesi amacıyla geliştirilmektedir.

Farklı platformlar üzerinde çalışabilecek genel amaçlı robot yazılımları yapmak zordur. Özellikle insanlara önemsiz gibi görünen sorunlar, robotun perspektifinden bakıldığında çok farklı zorluklar içermektedir. Bu zorlukları gidermek için uğraşan bir bireyin, laboratuvarın veya kurumun tek başına bunların üstesinden gelmesi oldukça güçtür. Sonuç olarak ROS, robotik yazılımlarını, işbirliği yaparak geliştirmeyi teşvik etmek için sıfırdan inşa edilmiş bir framework'tür ve geliştirilmesi işbirlikçiler sayesinde sürekli devam etmektedir. Örneğin bir robot, iç mekânlarda haritalama yapmak konusunda iyiyse ve bu robotu gerçekleştiren grup, haritalama ile ilgili tecrübelerini ROS framework'ü ile paylaşırsa, diğer gruplar da tekerleği yeniden icat etmek yerine bu haritalamayı kullanarak ve/veya geliştirerek hem zaman hem de iş gücü tasarrufu sağlamış olacaklardır. Aynı şekilde, konum belirleme, bilgisayarla görme, navigasyon ve birçok farklı robot platformları üzerinde gerçekleştirilen yazılımlarda ROS üzerinde uygulanmaktadır [46].

Kısaca ROS'un tarihine değinmek gerekirse, 2000li yılların ortalarında Stanford Üniversitesi robotik grubunun (STAIR), robot çalışmalarında dinamik yazılım ortamı geliştirmeyi amaçlamasıyla ilk çalışmalar ortaya çıkmaya başlamıştır. 2007 yılında robot

platformları geliştirmek için bir kuluçka merkezi sayılabilecek Willow Garage robotik laboratuvarı, sağladığı bilimsel kaynakların kapsamlarını genişleterek iyi tasarlanmış ve test edilmiş robotik yazılımları meydana getirmeye başlamıştır. Bu çaba bu tarihten itibaren, birçok robotik araştırmacısı tarafından, hem ROS çekirdeğini oluşturacak fikirler oluşması, hem de bazı modüllerin gerçekleştirilmesi bakımından desteklenmeye başlanmıştır. Genel olarak çekirdek çalışmanın tamamlanmasının ardından BSD açık kaynak lisansı iznini kullanarak ortaya çıkan framework açık kaynak bir yazılım haline getirildi ve bu aşamadan sonra robotik yazılımlarında geniş çaplı kullanılan bir araç haline geldi. ROS framework'ünü kullanan robotik grupları, ROS üzerinde kendi kod depolarını (repository) açmaya ve katkıda bulunmaya başladılar.

Ardından ROS framework'ünün genel olarak şekillendiği "ROS 0.4" sürümü Ocak 2009'da yayınlandı. Willow Garage ekibi PR2 robotları üzerinde oldukça gelişmeler elde etti ve Ocak 2010'da "ROS 1.0" ile ROS framework'ü yeni eklentiler kazanmakla birlikte dokümante edilmeye de başladı. 2010 yılı içerisinde ROS'a, kod depoları olan farklı grupların kodları için hataların tespit edildiği, önerilerin sunulduğu ve sorunların giderildiği bir soru-cevap kısmı eklendi. Mart 2011'de, ROS'un 3. sürümü olan "ROS Diamondback" yayınlandı. Aynı yıl hazır platformuyla satılan TurtleBot robotu için "launcher" yayınlandı. Yine aynı yılın Ağustos'unda 4. sürüm "ROS Electric" yayınlandı. 2012'ye gelindiğinde ise çok daha fazla algoritmanın ROS ortamında gerçeklenmesiyle beraber 5. sürüm "ROS Fuerte" yayınlandı. Bir robotik olimpiyatları olan RoboCup'ta artık birçok takım ROS framework'ü üzerinde geliştirmeler yapmaya başladı. Bunun üzerine 2012 Ağustos'ta Avustralya'da "ROS Yaz Okulu" yapıldı. Bu dönemde ROS kullanıcıları için bir kaynak olması amacıyla "ROS By Example" kitabı Kasım 2012'de yayımlandı. 2012'nin sonuna gelindiğinde ise 6. sürüm olan "ROS Groovy" yayınlandı. Groovy'yle beraber ROS paketleri "catkin" paketlerine dönüştürüldü. Yine bir kaynak olması bakımından 2013 Eylül'de "Learning ROS for Robotics Programming" kitabı ve aynı ay içerisinde ROS'un 7. Sürümü olan "ROS Hydro" yayımlandı [46]. Daha önce de bahsedildiği üzere bu tez kapsamında ROS framework'ünün, tezin yazımı aşamasında son sürümü olan, "ROS Hydro" sürümü kullanılmıştır.

4.1 **ROS Düğümleri (Nodes)**

ROS düğümleri, çeşitli işlemler yapan process'ler gibi düşünülebilir. Düğümler, çeşitli servislerin, mesajların ve parametrelerin bir araya gelmesiyle oluşurlar. Kendileri de bir bütünün parçası olduklarında anlam ifade etmeye başlarlar. Çünkü bir robot sistemi birçok farklı düğümün birlikte çalışmasıyla oluşmaktadır. Örnek vermek gerekirse, bir düğüm, robot üzerinde bulunan lazer mesafe ölçüm duyargasının kontrolünü sağlarken, diğer düğümler robotun tekerleklerinin sürülmesi, lokalizasyonun hesaplanması veya ortamın haritasının çıkartılması gibi farklı işleri yaparlar.

Bunun içindir ki; ROS ortamında düğümlerin kullanılmasının tüm bir sistem düşünüldüğünde oldukça yararları vardır. Diğer bir yandan, her bir düğüm kendi içerisinde izole edildiği için, herhangi bir düğümde meydana gelen hatalar sadece o düğümü ilgilendirecek, diğer düğümler çalışmaya devam edecektir. Bu izole sistem, hatalara karşı toleransları arttırmak için yapılmıştır. Monolotik sistemlerle kıyaslandığında kod karmaşıklığı azalmaktadır. Uygulama (Implementation) detayları gizli tutulabildiği için, kullanıcıya sadece düğümlerin nasıl kullanıldığını öğrenmek düşmektedir. Bu bir soyutlaştırma da sağlamaktadır. Yani farklı programlama dilleri ile de uyumluluk içerisinde olacak şekilde tasarlanmışlardır.

Sistem içerisinde her çalışan düğüm için eşsiz (unique) bir id tanımlanır. Aynı zamanda her bir düğümün bir tipi vardır. Mesela lazer mesafe ölçüm duyargası ile iletişime geçip ondan okuduğu değerleri bir mesaj olarak bastıran bir düğümümüz olduğunu düşünürsek; bu düğümün adı lazer_dugumu, tipi ise hokuyo_node olabilmektedir. Buradaki isimlendirme ve tiplendirme kavramlarını nesneye dayalı kavramlardan örnek vererek açıklamak gerekirse; düğüm bir metoda karşılık gelirken düğümün tipi bir sınıfa karşılık gelmektedir. Yani düğümün içerisinde bulunduğu ROS paketi o düğümün tipi olmaktadır. ROS'da düğümler overwrite edilememektedir. Bu sebeple aynı isimde farklı parametreler alan iki düğüm yazıldığında, ROS ilgili paket içerisindeki ilk düğümü seçecek diğerini hiçbir zaman kullanmayacaktır. "rosnode" komutuna ilgili düğüm ismi gelecek şekilde terminalde sorgu yapıldığında, o düğümün bilgilerine (hangi paket tarafından yayınlandığına, hangi sıklıkla yayınlandığına, içerisinde hangi veriler olduğuna, frame numarasının ne olduğunu, vb.) erişilebilir [46].

4.2 **ROS Mesajları (Topics)**

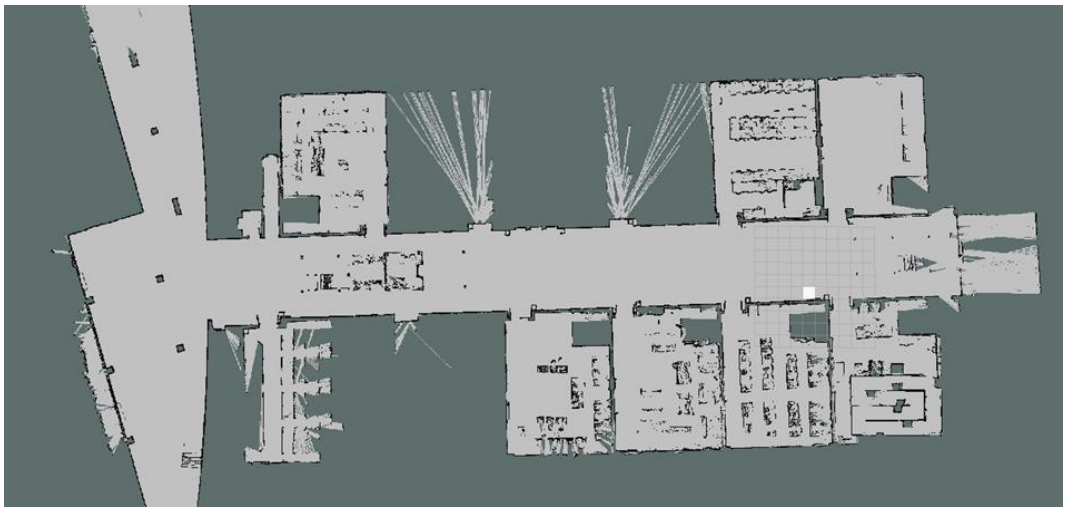
ROS'da düğümler (nodes) birbirleri ile mesajlar (topics) üzerinden haberleşmektedirler. Haberleşme şu şekilde gerçekleşmektedir: Bir ROS düğümü, kendine tanımlanmış işlemleri yaptıktan sonra elde ettiği sonucu veya aktarmak istediği değerleri "publish" komutu ile yayınlar, bu mesaj (topic) sistemde her an için bulunur ve bu mesajı kullanmak isteyen herhangi bir düğüm ilgili mesaj ismini kullanarak bu mesaja abone (subcribe) olur. Aynı düğümlerde olduğu gibi, her bir mesaj farklı isimlerde ve farklı tiplerde olmalıdır. Eğer 2 farklı düğüm, sisteme aynı isimde aynı tipte mesaj yayınlarsa, o mesaja o an abone olan düğüm hangisinin yayınlandığı mesajı yakalarsa onun verilerini kullanır. Burada doğru verilerin kullanılabilmesinin garantisi olmamakla beraber sistemin sonlanmasına sebep verecek bir sonuç doğurmamış olacaktır. Ancak aynı isimde farklı tipte mesajlar yayınlandığında sistem hata verecektir. Böyle bir durum mesajlara abone olan düğümler için de sorun teşkil edecektir çünkü mesaja abone olurken mesajın tipine de vurgu yapılmaktadır. Diğer bir yandan bu aynı zamanda bir avantajdır. Şöyle ki; otonom gezmekte olan bir robot için tekerleklere iletilecek /cmd_vel mesajını ilgili düğüm üretmekte ve tekerlekleri yöneten düğüm bu mesaja abone olarak değerleri işlemektedir. Robot bu sayede otonom bir şekilde gezebilmektedir. Ancak otonom gezme sırasında robotun başına gelebilecek herhangi bir olumsuzluk durumunda dışarıdan bir müdahaleye de olanak veren bir joystick ile aynı mesaj (/cmd_vel) üretilerek robotun olumsuz durumdan kurtulması sağlanabilmektedir. Bu durumda her iki farklı düğümden de aynı mesaj üretilmiş olacaktır. Bu mesaja abone olan düğüm ise sistemde o an hangi düğümden gelen /cmd_vel mesajı varsa ona göre davranacaktır. Yani otonom gezinme için komut gönderen düğümün hatalı bir gezinme rotası çizmesi durumunda joystick yardımı ile bir operatör robotun gidişine müdahale edebilecektir. Bu sebeple de aynı isimde aynı tipte mesajların sisteme birden fazla düğüm tarafından gönderilebilmesi bazen kullanışlı olmakla beraber tasarımının iyi incelenerek yapılması gerekmektedir [47].

4.2.1 Tezde Kullanılan ROS Mesajları

Bu tez kapsamında kullanılan ROS mesajlarına (topics), mesajların tiplerine ve ne bilgisi taşıdığına aşağıda maddeler halinde değinilmiştir.

- /scan (tipi: sensor_msgs/LaserScan [48]): Lazer mesafe ölçüm duyargasına /hokuyo_node isminde bir düğüm bağlanmaktadır ve mesafe ölçümlerini okuyarak bu mesajın içerisine yazmaktadır. Bu mesajın başlık kısmında (header), lazer mesafe ölçüm duyargasıyla ilgili bir takım bilgiler (maksimum ölçüm mesafesi, minimum ölçüm açısı, ölçüm frekansı vb.) tutmaktadır. Ardından başlık kısmında, sınırları ve şekli çizildiği ölçüde duyarganın ortamdan aldığı ölçümlere bir dizi şeklinde yer vermektedir.
- /joy (tipi: sensor_msgs/Joy [46]): Bu mesaj, robotu uzaktan kumanda etmekte kullandığımız "Logitech F710" isimli joystick için yazılmış "joy" tipinde /joy_node isminde düğümün yayınladığı mesajdır. İçerisinde başlık bilgisiyle beraber o an joystick'in hangi analog veya dijital butonuna basıldığını döndüren bilgiler vardır. Dijital bilgiler için; butona basılı olmadığı durumlarda 0, basıldığı durumlarda ise 1 döndürmektedir. Analog kollar hareket ettirildiğinde ise 0 ile 1 arasında belirli bir interpolasyonla değerler üretilmektedir.
- /cmd_vel (tipi: geometry_msgs/Twist [49]): Joystick'ten alınan /joy mesajı daha önce bahsedildiği gibi sadece 0 ile 1 arasında değerler üretebilmektedir. Oysaki robota git veya dur şeklinde komutlar vermek yerine farklı hızların verilebilmesi robotun sürüşü açısından daha iyi olacaktır. /joy mesajından alınan değerler yazılan bir düğüm(/robot_teleop) aracılığıyla yorumlanmakta ve joystick'ten gelen farklı değerlere göre robota m/s cinsinden sürüş hareketleri verilmektedir. /cmd_vel mesajı lineer ve açısal hız bilgilerini x, y ve z eksenlerinde taşıyan bir yapıdadır. Diğer bir yandan /cmd_vel mesajı sadece /robot_teleop düğümü ile üretilmemektedir. Robotun operatörsüz sürüldüğü durumda (otonom gezinim), robotun hareketi için üretilecek /cmd_vel mesajını, /navigation düğümü üretmektedir. Yine m/s cinsinden mesaj üreterek robotun hareket etmesini sağlamaktadır.
- /map (tipi: nav_msgs/OccupancyGrid [50]): gridMapping ve HectorMapping gibi haritalama algoritmaları tarafından yayınlanan bir mesaj türüdür. Robotun gezdiği ortamın haritasını çıkartan bu algoritmalar, haritayı, binary olarak engeller ve boşluklar olacak şekilde tutan bir mesajın (/map) içerisine

kaydederler. Mesajın başlığının içinde haritanın büyüklüğü, çözünürlüğü, zaman bilgisi ve frame_id gibi bilgiler bulunmakla beraber aynı zamanda robotun harita üzerindeki pozisyon bilgisini "geometry_msgs/Pose" tipinde tutar. Haritayı üreten düğümler tarafından haritanın ne kadar zamanda bir güncelleneceğine değinilmektedir. Her bir güncelleme adımında haritanın boyutları da dinamik olarak büyüyecek şekilde mesajın içeriği oluşturulmaktadır. Üretilen /map mesajı "map_server" paketi ile kaydedilip yayınlanabilmektedir. Buna yeri geldiğinde daha ayrıntılı bir şekilde değinilecektir. Bu mesaj ile oluşturulmuş örnek bir haritaya Şekil 4.1'de yer verilmiştir.

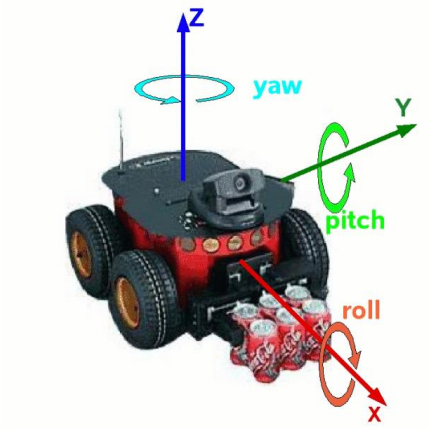


Şekil 4.1 HectorMapping ile çıkartılmış örnek bir harita

- /tf (tipi: tf/tfMessage [51]): ROS düğümlerinin ürettiği mesajların her birisinin frame_id'si vardır. frame_id, düğümlerin birbirleri ile ana-düğüm, çocuk-düğüm ilişkilerinin kurulacağı frame ağacını oluşturmak için her bir mesajda olan bir belirteçtir. Bu frame'ler gerektiğinde statik dönüşümler (transformasyonlar) kullanılarak gerektiğinde ise dinamik dönüşümler kullanılarak birbirlerine bağlanmaktır. Aynı şekilde bağlantıyı gerçekleştiren kod, programlayıcı tarafından yazılabilmekle beraber, herhangi bir düğüm ile de bağlantısı sağlanabilmektedir. Dönüşümler aynı zamanda robotun 3 boyutlu şeklinin ortaya çıkarılması için de kullanılmaktadır. Şöyle ki; robotun üzerinde bulunan duyargaların, robotun merkezinden ne kadar uzakta ve hangi açıda durduğu dönüşümler ile ayarlanabilmektedir. Dönüşümler, 6 serbestlik derecesiyle çalışmaktadır. Bunlar;

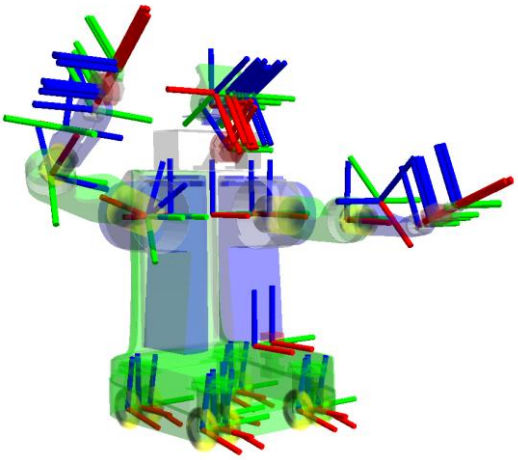
- x: robotun x ekseni üzerindeki konumu,
- y: robotun y ekseni üzerindeki konumu,
- z: robotun z ekseni üzerindeki konumu,
- yaw: robotun z eksenine olan göreli açısı,
- pitch: robotun y eksenine olan göreli açısı,
- roll: robotun x eksenine olan göreli açısıdır.

Bu 6 serbestlik derecesi örnek bir robot platformu (P3AT) üzerinde Şekil 4.2'de gösterilmiştir.



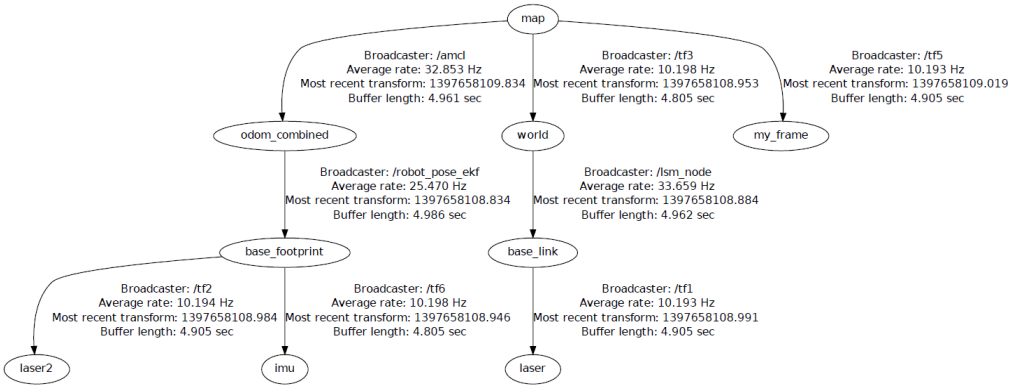
Şekil 4.2 Robot platformu (6 serbestlik dereceli)

Yine dönüşümler kullanılarak bir robotun iskeletinin çıkarılması Şekil 4.3'de gösterilmiştir.



Şekil 4.3 Robot (PR2) transformasyonları

Tez kapsamında farklı düğümler farklı şekillerde dönüşümler ile bağlanarak farklı modüllerin gerçekleştirilmesi sağlanmıştır. İlgili modüller anlatılırken her bir modül için ortaya çıkan dönüşüm ağacı (transformasyon ağacı) verilecektir. Bu sayede düğümler arasındaki ana-çocuk ilişkisi daha iyi gösterilebilecektir. Örnek bir dönüşüm ağacı Şekil 4.4'de verilmiştir.



Şekil 4.4 Örnek bir dönüşüm ağacı

Bu ağaç, ROS çekirdeği çalışırken terminale (uç birime) "view_frames" yazılması ile işaretleyicinin gösterdiği dizinde oluşturulmaktadır. Şekil 4.4'te görüldüğü üzere ana düğüm /map olacak şekilde, bütün düğümler birbirlerine ana-çocuk ilişkisiyle bağlanmışlardır. Aynı şekilde her bir bağlantıyı sağlayan düğüm ismi "broadcaster" tanımlayıcısının karşısında gösterilmiştir. /tf1, /tf2, /tf3, /tf5 ve /tf6, programcı tarafından elle verilmiş dönüşümlerdir. Diğer dönüşümler ise tanımlayıcının yanında yazan düğümler tarafından oluşturulmuştur.

Bütün düğümler düzgün bir şekilde çalıştırılsa bile aralarındaki dönüşümler geçerli bir şekilde tanımlanmadığı sürece sistem ayağa kalkamayacaktır. Bu yüzden, dönüşümler oldukça önem arz etmektedir.

- /odom (tipi: nav_msgs/Odometry[52]): Odometri mesajı, robotun belirli bir an için, içinde bulunduğu ortamın neresinde olduğu bilgisini "quaternion" formatında tutar. Quaternion formatı yukarıda bahsedilen 6 serbestlik dereceli konum bilgisini tutabilen formattır. Bunun yanında, o ortama en son geldiği noktada aldığı /cmd_vel mesajının da lineer ve açısal bilgilerini tutabilmektedir.

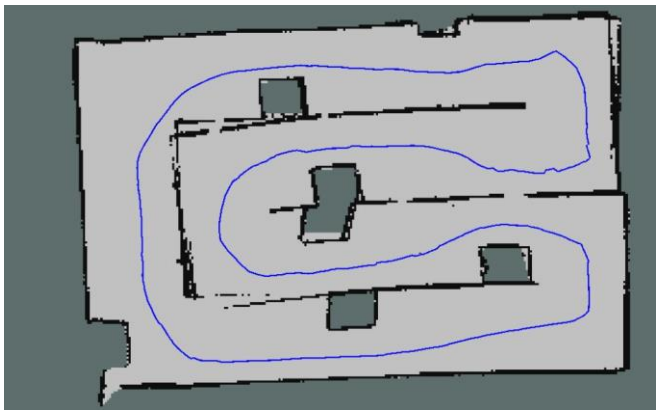
Genellikle robotun tekerleklerine bağlanan encoder'lar ile robotun ortam içerisinde, başlangıç konumundan ne kadar hareket ettiği bilgisi hesaplanarak mesaj olarak yayınlanmaktadır. Ancak her robot sisteminde robotun tekerleklerine bağlı bir encoder bulunmamaktadır ki bu tez kapsamında encoder kullanılmamıştır. Tekerleklerin dönme miktarı ile hesaplanan bu odometri çeşidine tekerlek odometrisi (wheel odometry) denilmektedir. Bu odometri çeşidi, tekerleğin dönme miktarını hesaba katarak robotun odometrisini tutmaya çalıştığı için odometri mesajı çok güvenilir olmayabilir. Çünkü robotun tekerleği belli bir zaman diliminde döndüğü halde, ortamdan kaynaklanan sürtünme gibi sebeplerden dolayı robot o an yerinden hareket etmemiş olabilir. Ama tekerlek döndüğü için odometri bilgisi hareket ettiği bilgisi ile güncellenecektir. Aynı şekilde robotun tekerlekleri dönmezken ötelenmeden dolayı robot hareket edebilmektedir. Yine odometri hesaplanırken bu hareket tekerlekler dönmediği için hesaba katılmayacaktır. Bu sebeple tekerlek odometrisi bilgisi güvenilir bir odometri bilgisi değildir.

Bunun yanı sıra kinematik denklemler kullanılarak üretilen odometri mesajları da yapılan araştırmalarda kullanılabilmektedir. Bu uygulama da ise robota verilen /cmd_vel mesajlarını robotun eksiksiz bir şekilde yerine getirdiği varsayılarak kinematik denklemler kullanılarak verilen hıza karşılık robotun ne kadar gitmiş olabileceği tespit edilmeye çalışılmaktadır. Aynı şekilde tekerlek odometrisinde olduğu gibi robota verilen her hız komutunu, robotun aynı şekilde işleyemeyeceği veya gürültülü bir şekilde işleyebileceği göz önünde bulundurulduğunda kinematik yöntemlerle odometri kestirimi de oldukça gürültülü bir mesaj içerecektir. Bu odometri tipiyle ilgili detaylı bilgiye Bölüm 8'de "Kinematik Tabanlı Gezinge Çıkarımı" bölümünde değinilecektir.

Farklı araştırmalarda, robot üzerinde bulunan lazer mesafe ölçüm duyargası kullanılarak robotun odometrisi kestirilmeye çalışılmıştır ki; bu tez kapsamında odometri bilgisi bu şekilde elde edilecektir. Temelde ardışık zamanlarda alınan lazer mesafe ölçümlerinin birbirleri ile olan ilişkileri kullanılarak konum tespiti yapılmaktadır. Bu odometri tipiyle ilgili detaylı bilgiye Bölüm 5'te Laser_Scan_Matcher paketi anlatılırken değinilecektir.

Bir diğer odometri kestirimi de görseller aracılığıyla olmaktadır. Lazer mesafe ölçüm duyargası ile yapılan odometri çıkarımındaki gibi ardışık zamanlarda alınan 2 farklı görsel birbirleriyle kıyaslanarak robotun ne kadar hareket ettiği tespit edilebilmekte ve odometri bilgisi güncellenebilmektedir. Tüm bu metotlarda çıkarılan odometri bilgisi /odom mesajı olarak "nav_msgs/Odometry" tipinde yayınlanmaktadır.

- /pose2D (tipi: geometry_msgs/Pose2D [53]): Bu mesaj laser_scan_matcher düğümü tarafından yayınlanan bir mesajdır. Bu paket, temel olarak farklı zamanlarda alınan 2 lazer taramasının birbirleri ile eşleştirilmesi ile (scan matching) robotun pozisyonuna dair bir bilgi üretmesi sonucu yayınlanmaktadır. Mesaj robotun x ve y eksenindeki konumunu ve bu konumdaki θ açısını ihtiva etmektedir. /odom mesajından farkı, 6 serbestlik derecesinde konum döndürmemesi, onun yerine 2 boyutlu bir konum döndürmesidir. Ayrıca /odom mesajında kovaryans matrisleri bilgisi varken bu mesaj kovaryans matrisi bilgisi tutmamaktadır.
- /visualization_marker (tipi: visualization_msgs/Marker [54]): Bu mesaj tipi, içerisinde x ve y eksenlerinde bilgisi olan noktalar dizisi tutmaktadır. Tez kapsamında robotun harita üzerinde gezdiği yolun (gezingenin) harita üzerine çizdirilmesi de amaçlanmıştır. Bu kapsamda, robot belirlenen bir eşik değerin üzerinde hareket ettiğinde robotun o anki odometrisinin x ve y eksenindeki bilgisi okunarak bir nokta oluşturulmakta ve bu nokta, bir noktalar dizisine eklenmektedir.



Şekil 4.5 Örnek bir gezinge

Bu mesaj, bu noktalar dizisini tutmaktadır. Bu mesajı yayınlayan düğüm noktaların ve noktaları birbirleri ile birleştiren çizgilerin renklerini de belirleyerek mesajın içerisine yazmaktadır. Harita görselleştirildiğinde robotun harita üzerindeki gezingesi gözlemlenebilmektedir. Bu mesaj ile oluşturulmuş örnek bir gezingeye Şekil 4.5'te yer verilmiştir.

- /robot_pose_ekf/odom_combined (tipi: geometry_msgs/PoseWithCovarianceStamped [55]): robot_pose_ekf paketi, farklı yöntemler (tekerlek odometrisi, lazer mesafe duyargası ile çıkarılan odometri, atalet duyargası yardımıyla çıkarılan odometri vb.) kullanılarak hesaplanmış odometrileri birbirleri ile belirli bir yöntem kullanarak birleştiren ve daha doğru odometri bilgisi üreten bir yapıya sahiptir. Yani bu paket, farklı yöntemler kullanılarak hesaplanan odometri bilgisini daha efektif hale getirebilmek amacıyla kullanılmıştır. /odom mesajı ile aynı bilgileri içermektedir. Ancak /odom mesajı o anki lineer ve açısal hız bilgilerini içerirken; bu mesaj içermemektedir.

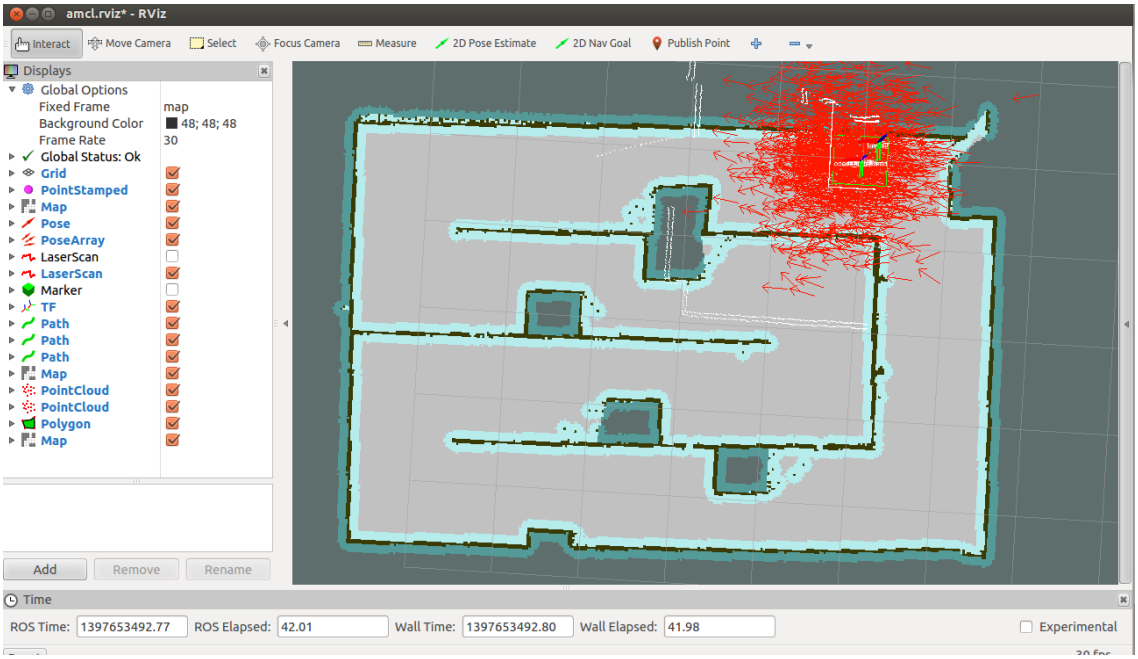
- /imu/data (tipi: sensor_msgs/Imu [56]): Microstrain 3DM-GX3-25 atalet duyargası kullanılarak robota dair açısal hız, ivme ve oryantasyon bilgilerini elde eden imu_node düğümünün yayınladığı bir mesaj tipidir. Tez kapsamında, robotun pozisyonunun daha doğru tespit edilebilmesi amacıyla kullanılmaktadır. Bu mesaj, robot_pose_ekf paketine verilerek odometri bilgisinin düzeltilmesi sağlanmaktadır. Bununla ilgili detaylı bilgiye, 8. Bölüm'de "Gezinge ve Konum Kestirim Yöntemlerinde Mesafe ve Atalet Duyargalarının Kullanımı" kısmında detaylı olarak değinilecektir.

4.3 **ROS Araçları (Tools)[46]**

ROS platformu kullanıcılarına, kullanım kolaylığı sağlamak amacıyla, platform üzerinde çeşitli araçlar geliştirilmiştir. Bu araçların büyük bir kısmından tez çalışması kapsamında da yararlanılmıştır. Özellikle haritaların incelenmesi ve çıktılarının alınarak teze yerleştirilmesi "rviz" aracı üzerinden gerçekleştirilmiştir. Bu araçlar başlıklar halinde incelenmiştir.

4.3.1 rviz

rviz, duyarga verilerinin, birleştirilmiş hallerinin, robot modellerinin ve 3 boyutlu verilerin görüntülenebileceği 3 boyutlu bir görüntüleme aracıdır. Tez kapsamında, robotun çıkardığı harita, harita üzerinde çizilen gezingeler, Monte Carlo lokalizasyonu ile robotun konumunu olasılıksal olarak tutan parçacıklar, lazer mesafe duyargasının ölçümleri, robotun başlangıç konumu ve güncel konumu, transformasyon bilgileri, navigasyon paketinin çıkardığı yerel ve genel maliyet haritaları, robota verilen hareket komutu sonucunda base_local_planner paketinin çıkarttığı hedefe nasıl gidilebileceğine dair yol rviz üzerinde gösterilmektedir. Ayrıca robotun başlangıç pozisyonu ve gitmesi istenilen harita üzerindeki konumu, rviz aracı üzerindeki modüller kullanılarak işaretlenebilmektedir. Örnek bir rviz ekranı görüntüsüne Şekil 4.6'da yer verilmiştir.



Şekil 4.6 Örnek bir rviz ekranı görüntüsü

4.3.2 rosbag ve rxbag

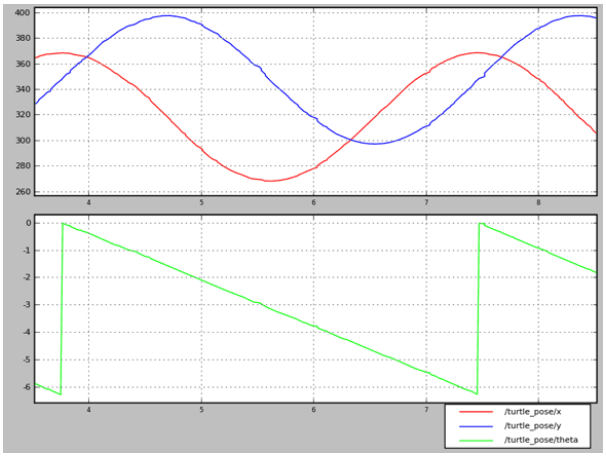
rosbag, bir komut satırı aracıdır. Robot hareket ederken yayınlanan mesajları kaydedebilmekte ve daha sonra bu mesajları oynatabilmektedir. Bu sayede robot çalıştırılmadan bag dosyası üzerinden testler yapılabilmektedir. Diğer bir yandan farklı algoritmaların aynı ortamda karşılaştırılmasında, robot her bir algoritmanın

uygulanması sırasında alanı tekrar gezmesi gerekecektir. Bu durum her ne kadar gayret edilse de robotun aynı alanı aynı şekilde ufak farklılıklarla bile olsa gezememesine yol açacaktır. Ve karşılaştırma yapılan örnekler farklı olduğundan net sonuca ulaşılamayabilir. Oysaki gezilen alanın verilerinin toplandığı mesajlar bir bag dosyasına kaydedildiğinde her bir algoritma için aynı veri kullanılabilmektedir. Bu yöntem hem pratiklik sağlamakta hem de karşılaştırma doğruluğunu arttırmaktadır. Bu tez çalışması kapsamında genellikle bag dosyaları üzerinde çalışmalar yapılmıştır.

rxbag, ise bir görselleştirme aracıdır. rosbag ile kaydedilen mesajların bilgilerini 2 boyutlu veya 3 boyutlu gösterme özelliğine sahiptir. Bu görselleştirme sayesinde kaydedilen verilerin analizleri yapılabilmekte ve varsa sıkıntılı noktalar tespit edilebilmektedir.

4.3.3 rxplot

rxplot, rxbag ve rviz gibi bir görselleştirme aracıdır. Ancak 2 boyutlu skalar verileri grafikselleştirmek için kullanılmaktadır. Bir ekran görüntüsü Şekil 4.7'de verilmiştir.

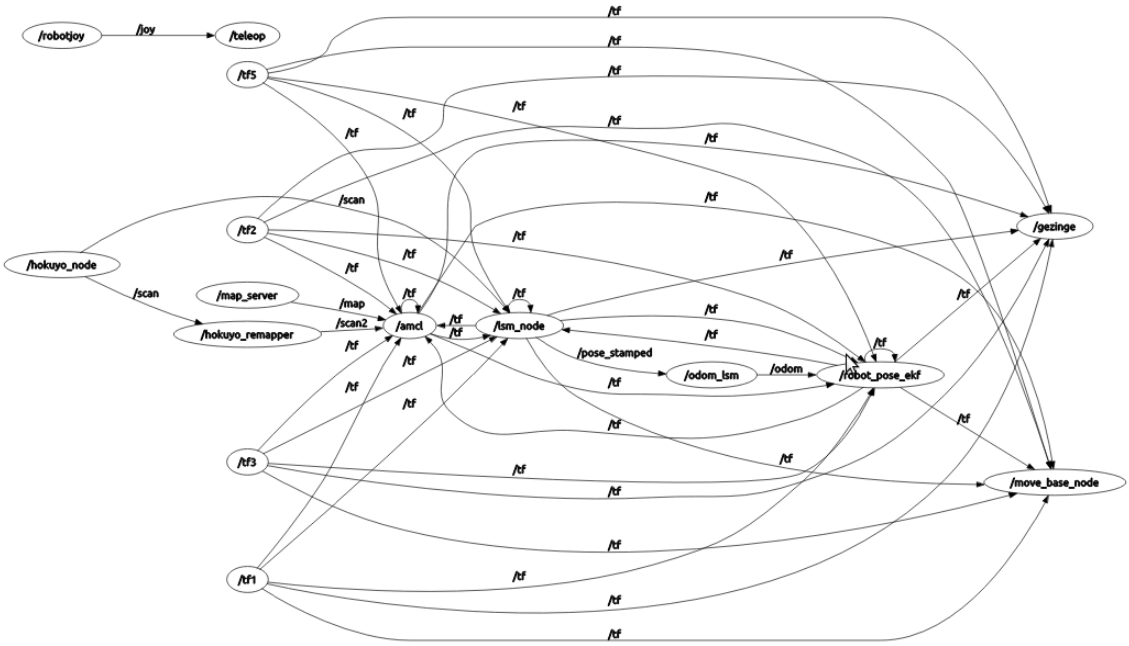


Şekil 4.7 rxplot ekranı

4.3.4 rxgraph

rxgraph ise ROS üzerinde çalışan işlemleri (process) ve bu işlemler arasındaki bağlantıları bir grafik şeklinde gösterebilen bir araçtır. Daha önce terminal ekranına "view_frames" yazıldığında dönüşüm ağacının oluşturulduğu söylenmişti. Bu araç da buna benzer şekilde bir ağaç üretmektedir. Ancak bu ağacın düğümleri, ROS üzerinde çalışan düğümlerden oluşmaktadır. Diğer bir şekilde ifade etmek gerekirse bu komut ile

ROS üzerinde çalışan düğümleri ve birbirleri ile ilişkilerini bir ağaç yapısı ile gözlemleyebilmekteyiz. Bu aracın bir görseline Şekil 4.8'de yer verilmiştir.



Şekil 4.8 rxgraph ekran çıktısı

4.3.5 roslaunch

Çoğu zaman robot sisteminin ayağa kaldırılabilmesi için birçok düğümün aynı anda çalıştırılması gerekmektedir. Her bir düğüm için terminal açılarak ilgili komut satırlarının yazılması ve çalıştırılması gerekmektedir. Öyle ki bu çalışma kapsamında zaman zaman 20'dan fazla düğümün aynı anda çalıştırılması durumu söz konusu olmuştur. Tabi ki bu kadar düğümün sırasıyla çalıştırılabilmesi için her seferinde ayrı terminal ekranlarının açılması ve komutların yazılması oldukça zaman almaktadır. Ayrıca bütün bu komutların biliniyor olmasını gerekmektedir. Bu zorluğun üzerinden gelebilmek için ROS ortamında "launch" dosyaları üzerinden çoklu düğüm çalıştırma gerçekleştirilmek istenmiştir. Bunun için "roslaunch" komutu kullanılmaktadır. "roslaunch" yereldeki veya uzaktaki ROS düğümlerini toplu olarak çalıştırmak için kullanılan SSH üzerinden erişim sağlayan bir ROS aracıdır. Bu araçla çalıştırılacak düğümlere "ROS Parameter Server" üzerinden değerler de verilebilmektedir. Herhangi bir problem sebebiyle aniden ölen işlemlerin otomatik olarak tekrardan çalıştırılmasını sağlayan bu araç bir veya birkaç farklı XML konfigürasyon dosyasından oluşur [46]. ROS

düğümlerinin çalışabilmesi için ROS çekirdeğinin (roscore) çalışıyor olması gerekmektedir. “roslaunch” aracı kullanılarak düğümler toplu olarak çalıştırılmak istendiğinde “roslaunch” komutu çalıştırıldığında öncelikle ROS çekirdeği çalıştırılmakta ardından XML dosyası içerisinde bulunan düğümler sırasıyla çalıştırılmaktadır. Yani “roslaunch” aracı ile birlikte “roscore” komutunun çalıştırılmasına gerek yoktur. Bu çalışma kapsamında yapılan deneyler için kullanılan “launch” dosyaları yeri geldiğinde referans gösterilerek tezin EK B bölümünde verilecektir.

4.4 **ROS Tarafından Desteklenen Duyarga Tipleri [46]**

ROS platformu üzerinde birçok farklı duyarga kullanımı desteklenmektedir. Bu duyargalar ana başlık olarak 9 farklı tipte incelenebilmektedir.

- 1 Boyutlu Mesafe Duyargaları: Tek bir boyutta (bir nokta olarak) mesafe ölçümü olan duyargalardır.
- 2 Boyutlu Mesafe Duyargaları: 2 boyutta ölçüm olan mesafe duyargalarıdır. Tez çalışması kapsamında 1 adet 30 metreye kadar 270°’lik ölçümlerle mesafe ölçebilen Hokuyo URG-04LX mesafe duyargası kullanılmıştır.
- 3 Boyutlu Mesafe Duyargaları ve RGB-D Kameralar: x, y ve z olmak üzere 3 farklı boyutta mesafe ölçümleri alabilen duyargalardır. Aynı zamanda derinlik ölçüsü alabilen ve bunu bir noktalar bulutu haline dönüştüren kameralar da (Kinect vb.) bu başlık altında incelenmektedir.
- Ses ve Konuşma Tanıyıcı Duyargalar: Mikrofon benzeri cihazlarla sesleri veya konuşmaları algılayabilen duyargalardır.
- Kameralar: Ortama ait görüntü bilgisine ulaşmak için kullanılan duyargalardır.
- Çevre Algılayıcı Duyargalar: Ortamın çevresinde bulunan farklı özellikleri algılayan duyargalardır. Örnek olarak karbondioksit miktarı ölçen duyargalardan veya termal kameralardan bahsedilebilir.
- Dokunma Hissetme ve Ten Algılayıcı Duyargalar: Özellikle insansı (humanoid) robotlarda kullanılan duyarga tiplerindendir. Ortamın sertliği gibi çeşitli özellikleri algılamak için kullanılmaktadır.

- Hareket Algılayıcı Duyargalar: Ortamda bulunan çeşitli nesnelerin hareketlerini algılayıp analiz eden çeşitli duyarga tipleridir.
- Konum Belirleme Duyargaları (GPS/IMU): Robotun konumunu belirlemeye dair tespitlerde bulunabilen duyargalardır. GPS ve atalet (IMU) duyargaları bunlara örnek verilebilir. (Tez çalışması kapsamında 1 adet Microstrain 3DM-GX3-25 atalet duyargası kullanıldığından bahsedilmişti).

BÖLÜM 5

LASER SCAN MATCHER (LSM)

LSM temel olarak, ardışık 2 “sensor_msgs/LaserScan” tipinde mesajı tarama eşleme (scan matching) yöntemi ile eşleyerek lazerin tahmini konumu “geometry_msgs/Pose2D” tipinde bir mesajla yayınlanmaktadır. Aynı zamanda robotun tahminin konumu bir dönüşüm (tf) olarak da yayımlanır. Bunu işlem iteratif olarak tekrarlanan iteratif en yakın nokta (Iterative Closest Point - ICP) temelli nokta-doğru ölçüt iteratif en yakın nokta (Point-to-Line Iterative Closest Point - PLICP) yöntemi temel alınarak yapılmaktadır [46]. Bu yöntem herhangi bir duyargadan üretilen odometri bilgisi kullanmadığı için tek başına bir odometri tahmini yapan düğüm olarak kullanılabilmektedir ki bu çalışma kapsamında genel olarak bu amaçla kullanılacaktır. Aşağıdaki başlıkta LSM’nin teorik altyapısına kısaca değinilmiştir.

5.1 **PLICP’ye Teorik Bakış [2]**

ICP algoritması, verilen $\{p_i\}$ noktalar kümesinden S^{ref} yüzeyine hangi $q = (t, \theta)$ roto-translation (dönüş $\big(R(\theta)\big)$ ve öteleme (t)) ile ulaşacağını yinelemeli olarak hesaplayan bir yöntemdir. $\{p_i\}$ noktalar kümesi için q roto-translation tanımı (5.1) ile verilebilir.

$$p \oplus q = p \oplus (t, \theta) \triangleq R(\theta)p + t \tag{5.1}$$

ICP algoritması q ile dönüştürülmüş p_i noktalar kümesinin S^{ref}’te Öklid karşılıkları olan noktalar ile uzaklıklarını q üzerinden en küçüklemeye çalışmaktadır. ICP için (5.2)’de verilen kısıt denkleminde, $\boldsymbol{\Pi}\{S^{ref}, p\}$ ile S^{ref} üzerine Öklid izdüşümü ifade edilmektedir.

$$\min_{q} \sum_{i} \left\| p_i \oplus q - \mathbf{\Pi}\{S^{ref}, p_i \oplus q\} \right\|^2 \tag{5.2}$$

(5.2) ile verilen denklem için kapalı formda bir çözüm olmadığından bir q_0 ilk dönüşüm durumundan hareketle ICP için yinelemeli kısıt denklemi (5.3) ile verilebilir.

$$\min_{q_{k+1}} \sum_{i} \left\| p_i \oplus q_{k+1} - \mathbf{\Pi}\{S^{ref}, p_i \oplus q_k\} \right\|^2 \tag{5.3}$$

$\mathbf{\Pi}\{S^{ref},\cdot\}$'e ilişkin tanımlara göre farklı ICP yaklaşımları tanımlanmıştır. PLICP algoritması ise karşılaştırılacak noktanın referans yüzeyi üzerinde en yakın olduğu doğruya mesafesini kapalı formda bir çözüm ile kullanmaktadır. Bu sebeple, nokta-nokta eşleştirmeleri doğrusal yakınsarken, PLICP karesel olarak yakınsar. n_i^T, referans yüzeyde en yakın doğru normalinin transpozu olmak üzere, PLICP kısıt denklemi (5.4) ile verilmektedir.

$$\min_{q_{k+1}} \sum_{i} (n_i^T [p_i \oplus q_{k+1} - \mathbf{\Pi}\{S^{ref}, p_i \oplus q_k\}])^2 \tag{5.4}$$

PLICP algoritması, y_{t-1} referans lazer taraması, y_t ikinci lazer taraması, q_0 dönüşüm değeri, i ikinci lazer ölçümü nokta indisleri, j referans lazer taraması nokta indisleri ve k yineleme adımı indisi olmak üzere;

Algoritma PLICP

Girdi: y_{t-1}, y_t, q_0

$S^{ref} \leftarrow y_{t-1}$ den oluşturulan parçalı doğru yüzeyi,
$k \leftarrow 0$
repeat
  $p_i^w \leftarrow p_i \oplus q_k$
  $j_1^i, j_2^i \leftarrow p_i^w$'ye en yakın iki nokta $\left(j_1^i, j_2^i \in y_{t-1}\right)$
  $C_k \leftarrow$ tüm $\langle i, j_1^i, j_2^i \rangle$ üçlüleri
  C_k'dan aykırı değerleri temizle
  $J(q_{k+1}, C_k) \leftarrow \sum_i \left(n_i^T \left[R(\theta_{k+1}) p_i + t_{k+1} - p_{j_1^i} \right] \right)^2$
  J'yi en küçükleyen q_{k+1} değerini bul
  $k \leftarrow k$+1
until (max _iterasyon_sayısı) or (yakınsama)

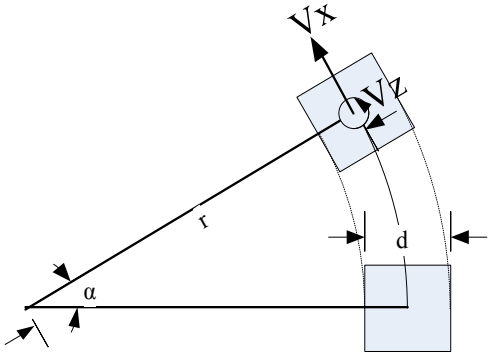
Çıktı: q

5.2 LSM'de Atalet Duyargası (Inertial Measurement Unit - IMU) ve Hız Bilgilerinin Kullanımı

ROS ortamında sunulan ve bu çalışma kapsamında kullanılan LSM paketi, LSM'yi iyileştirebilmek için, lazer tarama bilgisinin yanında atalet duyargası ve hız bilgilerini de ek olarak alabilmektedir. Bu sayede odometri tahmini sadece lazer taramalarıyla değil aynı zamanda kullanılan ek bilgiler yardımı ile de yapılabilmektedir. Alt başlıklar halinde bu ek bilgilerin ne şekilde ve ne için kullanıldığına değinilmiştir.

5.2.1 LSM'de Kinematik Bilginin (Hız) Bilgisinin Kullanımı (LSM-K)

Açık çevrim kontrollü olarak anlık hız bilgisinin zamanla çarpımından gidilen yol bilgisi elde edilmektedir. Şekil 5.1'de robotun hareketine ait kinematik model resmedilmiştir.



Şekil 5.1 Kinematik model

Robota ilişkin ilk poz durumundan (x_0, y_0, θ_0) Δt süre sonraki oluşacak poz durumuna (x_1, y_1, θ_1) geçiş (5.5) ile verilen şekilde hesaplanabilir.

$$\begin{aligned} x_1 &= x_0 + V_x \cos(\theta_0)\,\Delta t \\ y_1 &= y_0 + V_x \sin(\theta_0)\,\Delta t \\ \theta_1 &= \theta_0 + V_z \Delta t \end{aligned} \tag{5.5}$$

Burada robota verilen ötelenme (V_x) ve dönme (V_z) hızları robot üzerinde yapılan deneyler sonucunda elde edilmiştir. Bu deneyler kapsamında joystick'ten gönderilen değerlere karşılık robotun 5 metrelik bir mesafeyi ne kadar sürede gittiği 3 farklı deneyle tespit edilmiş ve bu deneylerin ortalamasında çıkan sonuç robotun joystick değerine karşılık gelen hız olarak kullanılmıştır. İlgili deneyin sonuçları Çizelge 5.1'de verilmiştir.

Çizelge 5.1 Joystick değerleri hız karşılıkları

Joystick'ten Verilen Değer	5 Metreyi Gitme Süresi (sn)			Çıkarılan Hız Bilgisi (m/sn)
	Deney 1	Deney 2	Deney 3	
12	97,00	95,55	93,77	0,052
16	49,54	48,64	49,30	0,102
20	34,22	33,67	33,65	0,148
24	26,38	26,31	26,52	0,189
28	22,41	22,42	22,15	0,224
32	19,70	19,35	19,40	0,257
64	12,71	12,69	12,67	0,394
128	10,96	10,85	10,88	0,459
256	10,25	10,30	10,44	0,484

Deney, önceden belirlenen ayrık değerler için yapılmıştır. Ancak joystick, analog değerler üretmektedir. İlgili deney verileri kullanılarak joystick'ten gelen analog değerlere karşılık bir interpolasyon fonksiyonu ile hız bilgisi üretilebilmektedir. Bu hız bilgileri "geometry_msgs/Twist" tipinde bir mesajın içerisine yazılmakta ve mesaj, LSM'nin abone olması için yayınlanmaktadır.

LSM'nin, ardışık 2 farklı taramayı iteratif olarak eşleştirmeye çalışmakta olduğuna daha önceden değinilmişti. Kinematik denklemlerden elde edilen hız bilgisi, tarama eşleşmenin ilk iterasyonunda robotun ötelenme miktarını (translation) belirlemek için kullanılmaktadır. Yani ilk iterasyonda, kinematik modelden elde edilen hız bilgisine göre robot ileri götürülmekte, ardından tarama eşleme çalıştırılmaktadır. Bu sayede eşlenecek taramalar neredeyse birbirlerinin üzerine geldikleri için çok kısa sürede tarama eşleme tamamlanmaktadır.

5.2.2 LSM'de Atalet Duyargasının Kullanımı (LSM-A)

Atalet duyargası robotun, global x,y ve z eksenlerine açısal olarak konumunu döndürmektedir. LSM, aynı hız bilgisinin kullanılmasında olduğu gibi, ardışık 2 farklı

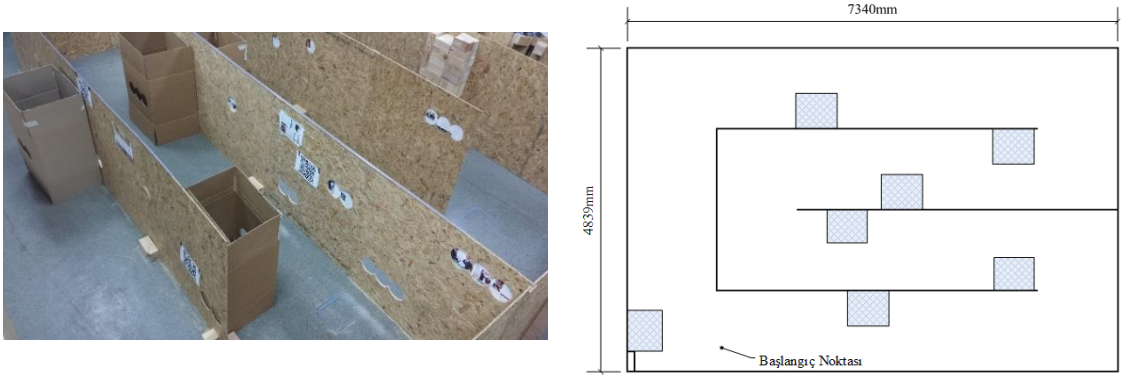
taramayı iteratif olarak eşleştirirken, ilk iterasyonda robotun açısal değişimini atalet duyargasından elde ettiği bilgileri referans alarak yapmaktadır. Bu bilgiyi alabilmek için "imu_node" düğümünün yayınladığı "imu" mesajına abone olmaktadır. Atalet duyargasından okunan değer kadar robotun konumu döndürdükten sonra tarama eşleme yapılmaktadır. Eğer 2 tarama arasında robot dönmekte ise, robotun değişen açı değeri kadar eşlenmeye çalışılan tarama döndürülmekte, ardından arasında açısal derece yokmuş gibi eşleştirilmeye çalışılmaktadır. Aynı hız bilgisi kullanıldığında olduğu gibi bu işlem de tarama eşlemenin daha az iterasyonla sonuçlanmasına sebep olacaktır.

5.2.3 LSM'de Atalet Duyargası ve Hız Bilgilerinin Kullanımı (LSM-AK)

LSM'ye hem hız bilgisinin hem de atalet duyargası bilgisinin sağladığı katkı bu başlıkta ortak olarak sağlanmaktadır. Yani ardışık 2 farklı taramayı iteratif olarak eşleştirirken, ilk iterasyonda robotun ötelenme miktarı hız bilgisinden, açısal değişimi ise atalet duyargasının ürettiği bilgiden referans alınarak ilklendirme yapılmaktadır. Her iki bilginin de kullanılmasıyla beraber robotun ne kadar döndüğü ve ne kadar ilerlediği tahmini olarak tespit edilebilmektedir. Bu tahmin üzerinden tarama eşleştirme yapılmaya çalışıldığı için oldukça az sayıda iterasyonda eşleme tamamlanmaktadır.

5.3 LSM (Laser_Scan_Matcher) Deneysel Sonuçları

ROS Hydro platformu üzerinde gerçekleştirilen ve bu çalışma kapsamında kullanılan "laser_scan_matcher" paketi IMU (Inertial Measurement Unit), tekerlek odometrisi, lazer taramaları gibi farklı düğümlere abone olarak tarama eşleme yapmaktadır. Teorik alt yapısına ve çalışma şekline bir üst başlıkta değinilmişti. Bu başlık altında ise farklı hızlarda alınan lazer taramaları için deneysel sonuçlara yer verilmiştir. İlgili deneylerin yapılabilmesi için RoboCup olimpiyatlarında kullanılan alana benzer bir labirentsel alan suntadan yapılmıştır. Labirentin boyutları 734 x 484 x 120 cm^3'dür. Şekil 5.2'de deneyler için hazırlanan alanın bir görseline ve planına yer verilmiştir. Deney kapsamında LSM'nin performansının kıyaslanabilmesi için 3 farklı hızda (10 cm/sn, 18 cm/sn ve 41 cm/sn) labirent gezilmiştir. Labirent içerisindeki robotun her bir farklı hızı için ".bag" dosyaları alınmış ve LSM çevrimdışı olarak çalıştırılmıştır. Sonucun doğruluk oranı gezingenin çevrimi tamamlamasına ve düzgünlüğüne bakılarak elde edilmiştir.



(a) (b)

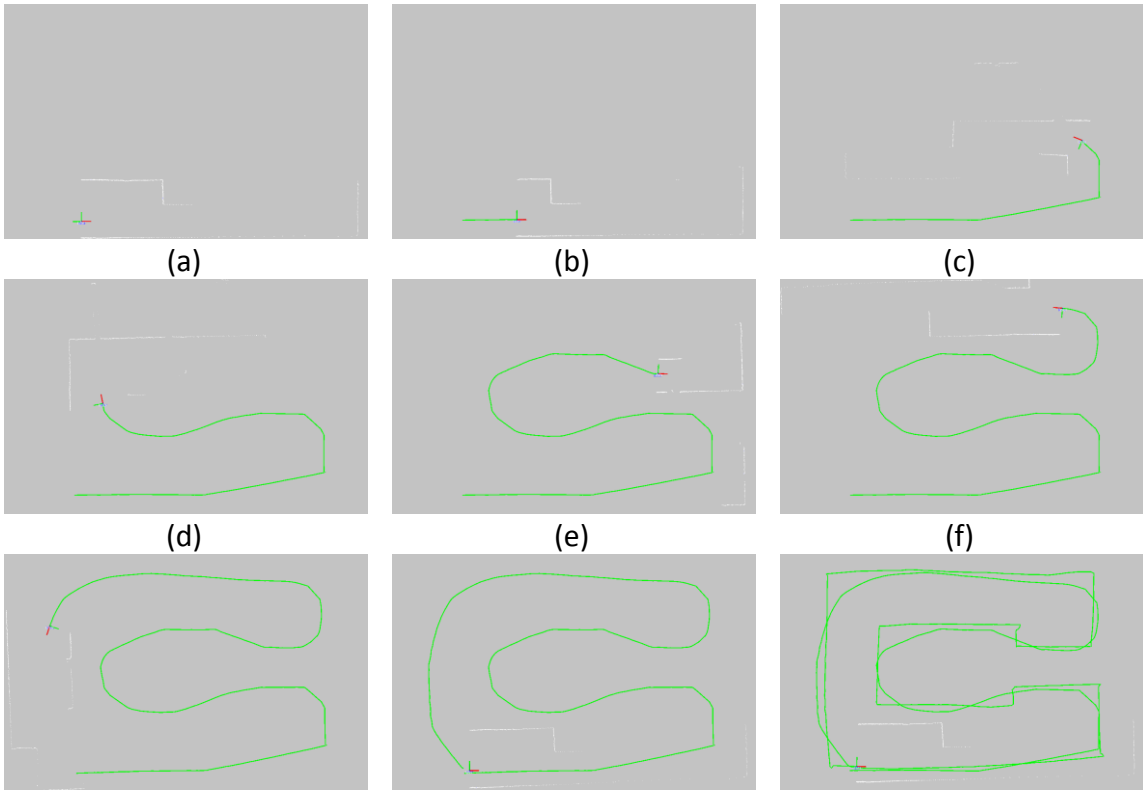
Şekil 5.2 (a) Çalışma alanına ilişkin görsel, (b) çalışma alanına ilişkin plan

Asıl olarak yapılması gereken gezingenin yer gerçeğinin çıkartılması ve bu yer gerçeğinden ne kadar saptığının hesaplanmasıdır. Bu başlık altında sadece LSM'in aynı ortam için farklı hızlarda ne şekilde çalıştığının incelenmesi gerçekleştirilmiştir. Gezingelerin yer gerçeğine göre kıyaslanmaları ise "Gezinge ve Konum Kestirim Yöntemlerinde Mesafe ve Atalet Duyargalarının Kullanımı" başlığı altında Bölüm 8'de incelenecektir.

5.3.1 10 cm/sn Hızla Yapılan Deney Sonuçları

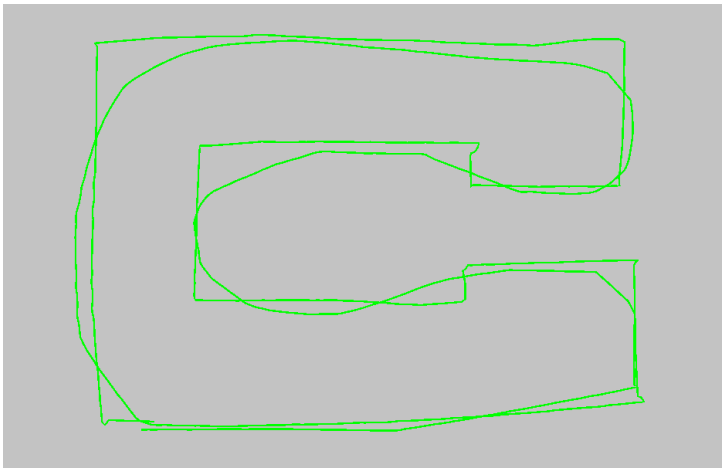
LSM, taramaları eşleştirerek robotun ortam içerisindeki pozunun bilgisini üreten bir yöntemdir. Pozun sürekli hali ise robotun gezingesini oluşturmaktadır. LSM çalıştırılırken, ROS üzerinde gerçekleştirilen paketin varsayılan parametre değerleri kullanılmıştır. LSM paketinin kullanılan parametreleri ve varsayılan değerleri EK C'de verilmiştir. LSM'nin başarım ölçütlerinin değerlendirilebilmesi için robotun gezinimine dair gezinge sonuçları görselleştirilmiştir. "rviz" ekranı üzerinde gösterilen robotun gezinimi yeşil renk ile simgelenmiştir. Kırmızı ve yeşil renklerden oluşan gezingenin baş tarafındaki hareketli belirteç ise robotun o anki pozunu göstermektedir. Beyaz noktalardan oluşan küme ise robot üzerinde bulunan lazer mesafe ölçüm duyargası tarafından algılanan lazer ölçümlerinin noktaları kümesidir. Robotun gezinimiyle beraber lazer ölçümleri farklılık göstereceği için sürekli değişen bir lazer noktaları kümesi "rviz" ekranlarında görülebilir. Şekil 5.3'de robotun labirenti 10 cm/sn hızla 2 tur gezdiği durum için gezinge çıkarımlarına adım adım yer verilmiştir. Labirentin 2 tur

gezilmesinin ardından 2 farklı gezinge çıkarımı gerçekleştirilmiştir. İlk gezinge daha yumuşak hatlara sahipken ikinci gezingede, daha sert dönüşlere yer verilmiştir.



Şekil 5.3 Varsayılan LSM parametreleri ile 10 cm/sn hızla labirentin 2 tur gezildiği bag dosyası için LSM gezingesi sonuçları

Çıkarılan gezingeden gözlemlenebilmektedir ki 2. gezingenin dönüşlerinin büyük bir kısmı 90°'lik açılarla gerçekleşmiştir. Gezinimin sonunda çıkarılan gezinge neredeyse döngüyü tamamlamıştır.

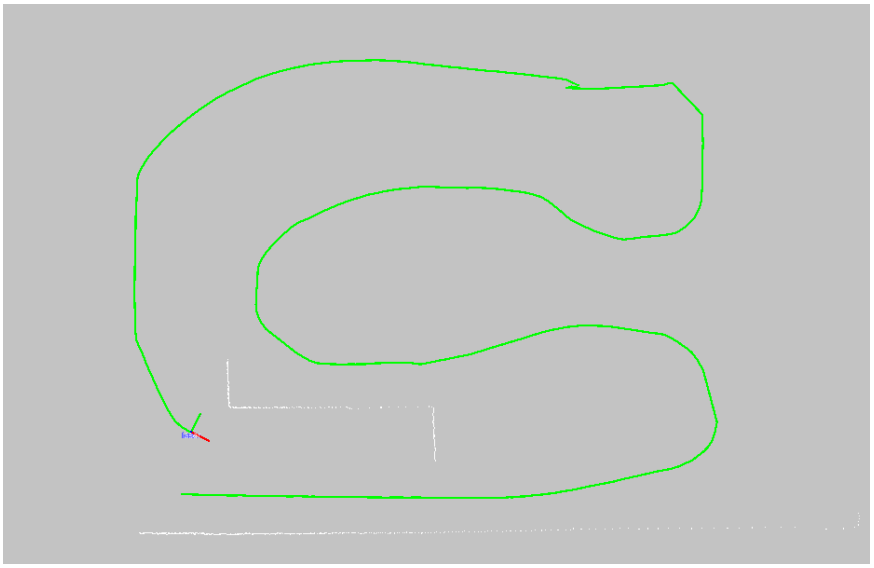


Şekil 5.4 Varsayılan LSM parametreleri ile 10 cm/sn hızla labirentin 2 tur gezildiği bag dosyası için LSM gezingesi son hali

Şekil 5.4'de görülebileceği gibi döngüyü tamamlamasına az bir mesafe kalmıştır. LSM artımsal hata içeren bir yöntem olduğu için 2 turun sonunda toplam hatanın yine de bu kadar az olması bu hızda LSM'nin gayet iyi çalıştığının bir göstergesidir.

5.3.2 18 cm/sn Hızla Yapılan Deney Sonuçları

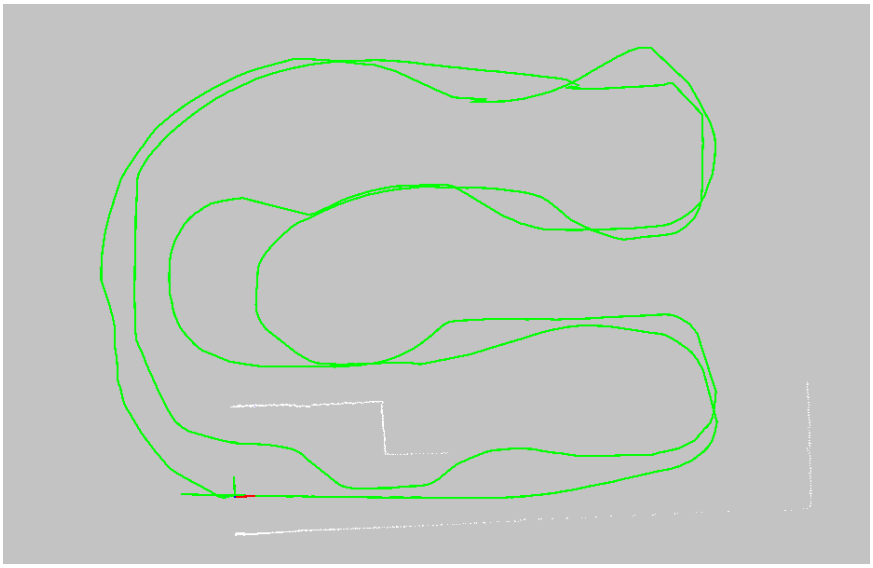
Bir diğer deney ise labirentin 18 cm/sn hızla gezildiği durum için yapılmıştır. Hızın arttırılması tarama eşlemenin gerçekleştirileceği taramalar arasındaki mesafenin artması anlamına geldiği için hızın artışıyla hatanın artması doğru orantılı olarak gerçekleşecektir. Şekil 5.5'te labirentte ilgili hızda 1 tur atılmasının sonucu görülmektedir. Fark edilebileceği üzere en üstteki koridorun gezingesi çıkartılırken bir kayma meydana gelmiştir. İlk turun bitirilmesi başlanılan yerde olmadığı için tam olarak bir karşılaştırma yapılamasa da ikinci turun bitirilmesi 1. turun başladığı yerde olduğu için 2 turun toplam hatası gezinge çevriminin kapatılmasıyla tespit edilebilmektedir.



Şekil 5.5 Varsayılan LSM parametreleri ile 18 cm/sn hızla labirentin 2 tur gezildiği bag dosyası için ilk turun LSM gezingesi sonucu

Şekil 5.6'da görülebileceği gibi 2. turun atılması sırasında da üstteki koridor gezilirken bir atlama meydana gelmiştir. Bu atlamaların toplamı gezingenin çevriminin tamamlandığı noktada, başlangıç noktasına olan uzaklık ile ölçülebilir. Yine Şekil 5.6'da görülebileceği gibi robot yaklaşık olarak 90 cm civarında başladığı konumdan daha farklı yerde gezinimi tamamlamıştır. 10 cm/sn hızla gezilirken hatanın 5 ila 10 cm arasında olduğu düşünülürse 18 cm/sn hıza çıkıldığında 90 cm civarında olan hata

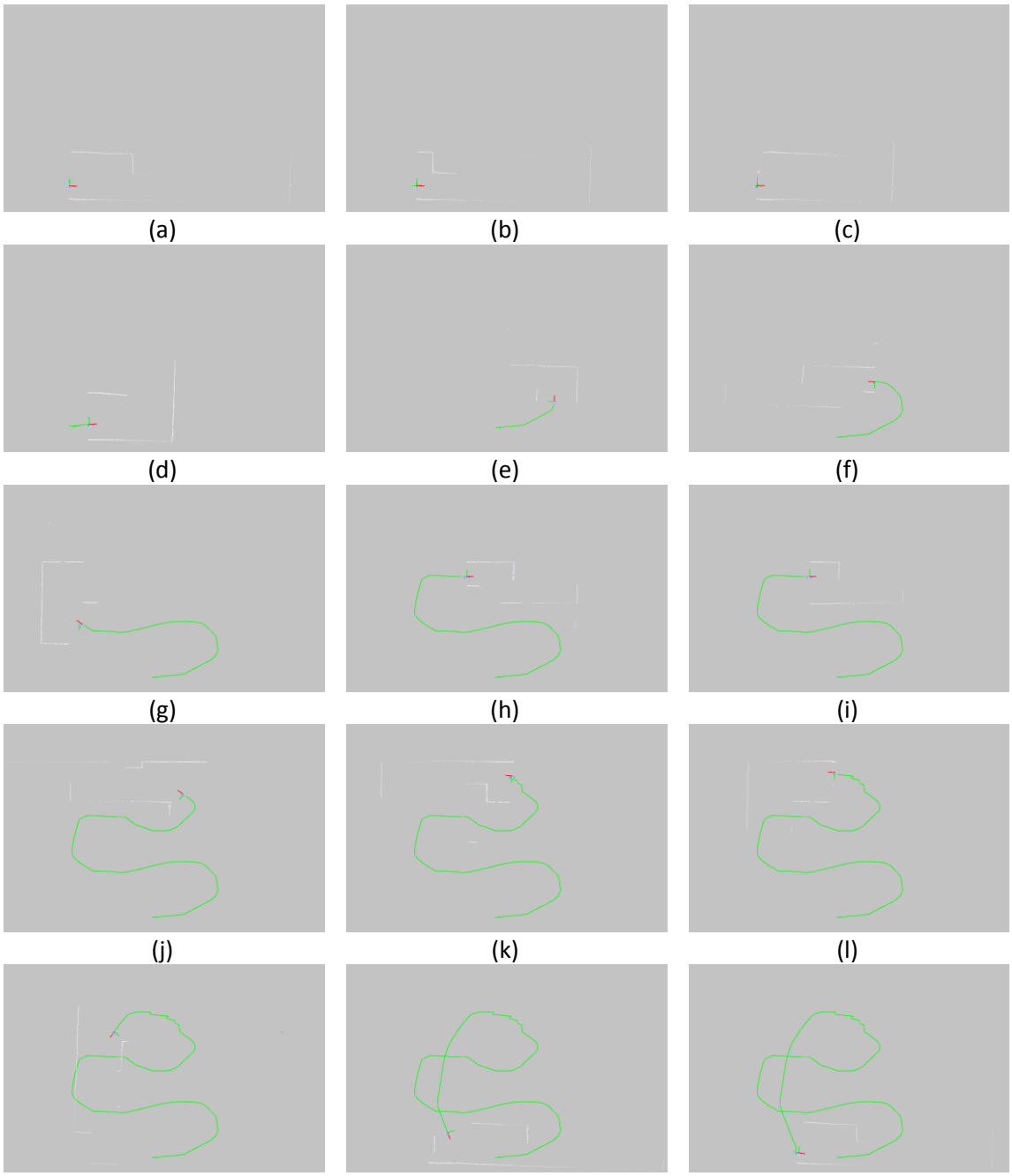
gayet yüksek bir orandır. Ama yine de gezingenin yer gerçeği şeklini içeren bir şekil ortaya çıkmıştır.



Şekil 5.6 Varsayılan LSM parametreleri ile 18 cm/sn hızla labirentin 2 tur gezildiği bag dosyası için ikinci turun LSM gezingesi sonucu

5.3.3 41 cm/sn Hızla Yapılan Deney Sonuçları

Hızın bir miktar daha arttırılarak 41 cm/sn'ye çıkarıldığı durumda ortaya çıkan LSM gezingesi sonuçları adım adım Şekil 5.7'de verilmiştir. Hızın bu miktara arttırılması sonucu LSM, taramalar arasında artan mesafeyi telafi edememektedir. Şekil 5.7-a, Şekil 5.7-b ve Şekil 5.7-c incelendiğinde gelen lazer tarama noktaları sürekli değişmekte ve robot ileriye doğru hareket halindeyken, robotun ileriye doğru gitmesini simgeleyen herhangi bir gezinge çizilmemiştir. Ancak Şekil 5.7-d'ye gelindiğinde robotun ilerleyişine dair kısa bir çizgi görülse de robot bu noktaya geldiğinde toplam 5 metre yol kat etmiştir. Bundan sonra bir süre gezingenin doğru çıkarılması gerçekleşse de Şekil 5.7-h'den Şekil 5.7-i'ye geçerken yine aynı durumla karşılaşılmıştır. Yani robot ilerlemiş olmasına rağmen (lazer tarama verilerinden anlaşılmaktadır) gezinge ilerlememektedir. Özellikle Şekil 5.7-j'den Şekil 5.7-l'ye kadar olan sürede robotun pozisyon bilgisi sürekli yer değiştirmekte ve robotun gezingesinin ilerlememesine sebep olmaktadır. Netice itibariyle Şekil 5.8 incelendiğinde, robotun geziminden ve gezinim çevrimini kapatmaktan çok uzak bir gezinge çıkarıldığı gözlemlenebilir. Bunun temel sebebi bu hızda çalıştırılan sistemin, taramalar arasındaki mesafenin fazlalığından dolayı, taramalar arası eşleme yapamaması ve atlamaların gerçekleşmesidir.

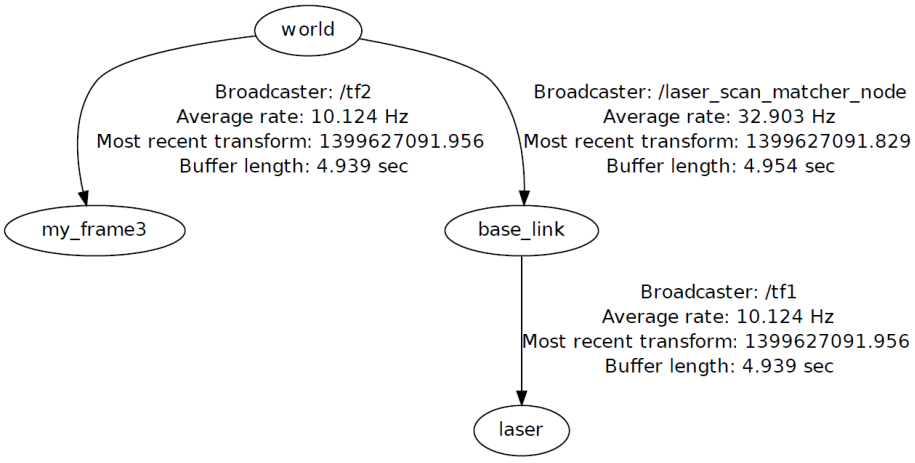


Şekil 5.7 Varsayılan LSM parametreleri ile 41 cm/sn hızla labirentin gezildiği bag dosyası için LSM gezingesi sonuçları

Gezingenin doğru çıkarılmasının tek ölçütü gezinim çevrimini tamamlamak değildir. Daha önceden de bahsedildiği gibi, yer gerçeği gezingesi ile yöntemlerin çıkarttığı gezingeler tezin farklı bir başlığı altında (Bölüm 8) detaylı olarak incelenecektir. Robotun labirenti farklı hızlarda gezdiği durumlar için çalıştırılan düğümlerin birbirleriyle bağlantısını ve ne şekilde bağlandığını gösteren dönüşüm ağacı Şekil 5.9'da verilmiştir.

Şekil 5.8 Varsayılan LSM parametreleri ile 41 cm/sn hızla labirentin gezildiği bag dosyası için LSM gezingesinin son hali

/laser frame'inden /base_link frame'ine ve /my_frame3 frame'inden /world frame'ine bağlantılar "static_transform_publisher" paketi kullanılarak statik olarak gerçekleştirilmiştir. /base_link frame'inden /world frame'ine olan bağlantıyı ise laser_scan_matcher düğümü üretmektedir. Bu yöntemleri çalıştıran "launch" dosyalarına EK B'de yer verilmiştir.



Şekil 5.9 LSM'nin farklı hızlarda çalıştırıldığı durumlar için ortak dönüşüm ağacı

BÖLÜM 6

ROBOT_POSE_EKF (RPE)

ROS Hydro sürümü üzerinde gerçeklenen Robot Pose EKF paketi robotun gezinimi esnasında farklı kaynaklardan gelen poz bilgilerini değerlendirerek robotun pozunun 3 boyutlu olarak tahmin edilmesi için kullanılmaktadır. 6D modelle (3D pozisyon, 3D oryantasyon) EKF kullanılarak tekerlek odometrisi, IMU duyarga bilgisi ve görsel odometriyi birleştirmek için geliştirilmiştir. Buradaki ana fikir, ROS mesajı olarak alınan duyarga sinyallerinin birleştirilerek bütünsel bir odometri sonucunun elde edilmesidir.

6.1 **RPE Çalışması Şekli**

Filtre düğümüne bilgi gönderen bütün duyarga kaynakları kendilerine ait referans penceresine (frame) sahip olabilir ve her bir referans penceresi zaman içinde rastgele sürüklenebilir. Böylece, farklı duyargalardan gönderilen mutlak pozlar birbirleriyle karşılaştırılamazlar. Bu düğüm, her bir duyarganın göreceli poz farklılıklarını EKF'yi güncellemek için kullanmaktadır.

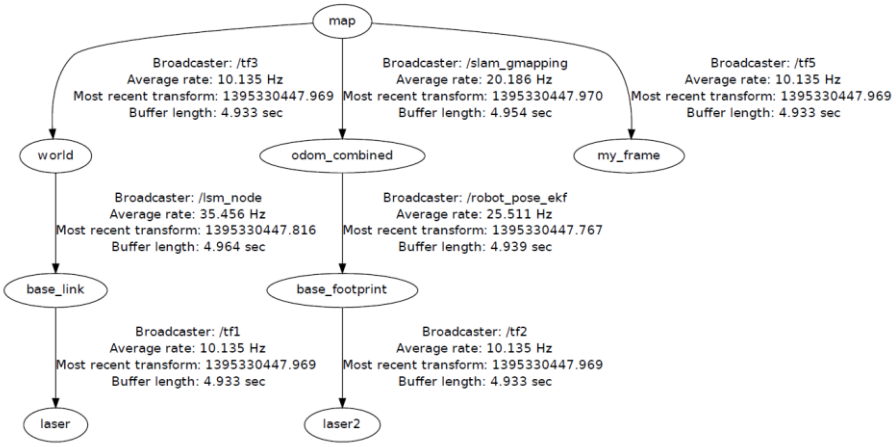
Bir robot kendi etrafında hareket ettikçe, referans yüzeyine göre pozuna dair belirsizlik git gide artmaktadır. Zamanla, kovaryanslar limit tanımaz bir şekilde büyümeye devam edecektir. Böylece, pozun kendi üzerinden kovarsyansları yayınlamak kullanışlı olmayacaktır. Onun yerine duyarga kaynaklarının zamanla kovaryanslarındaki değişimi yayınlamaları daha kullanışlı olacaktır. Mesela hız bilgisine ait kovaryanstaki değişimin yayınlanması gibi.

Robot poz filtresinin en son t_0 anında güncellendiği düşünülürse, t_0 zamanında sonra her duyargadan en az bir ölçüm gelesiye kadar robot_pose_ekf düğümü, robotun poz

filtresini güncellemeyecektir. t_0 zamanından daha sonra gelen bir t_1 anında bir odometri mesajı alındığında, IMU pozunun bilgisi de daha sonra gelen bir t_2 anına ait olduğunda, yani $t_2 > t_1 > t_0$ olduğunda ve bu an için bütün duyargalardan bilgi almak mümkün olduğunda, o zaman filtre güncellenmesi yapılabilecektir. t_1 anındaki odometri bilgisi direkt elde edilecek, IMU poz bilgisi ise IMU pozunun t_0 ve t_2 zamanları arasında yapılacak lineer interpolasyon dönüşümleri sonucunda elde edilecektir. Robotun poz filtresi, t_0 ve t_1 zamanları arasında alınan IMU ve odometri bilgisi ile göreceli olarak güncellenecektir [46].

6.2 **RPE Deneysel Sonuçları**

ROS Hydro sürümü üzerinde gerçekleştirilen Robot_Pose_EKF paketinin deneyleri için Yıldız Teknik Üniversitesi Bilgisayar Mühendisliği Bölümü'nün bodrum katı kullanılmıştır. Yaklaşık 80 metre uzunluğunda bir koridorda birkaç laboratuvarın içerisine de girerek tamamlanan gezinim yaklaşık 20 cm/sn hızla gerçekleştirilmiştir. Gezinim esnasında hem /scan hem de /imu/data mesajlarının kaydedildiği ".bag" dosyası toplanmış gezinim bittikten sonra bag dosyaları üzerinde denemeler gerçekleştirilmiştir. Deney için öncelikle sadece /scan mesajı kullanılarak üretilen /odom verisi kullanılmıştır. RPE, /odom mesajına abone olarak "base_footprint" frame'inden "odom_combined" frame'ine bir bağlantı oluşturmuştur. Bu deneyde ortaya çıkan dönüşüm ağacı Şekil 6.1'de verilmiştir.

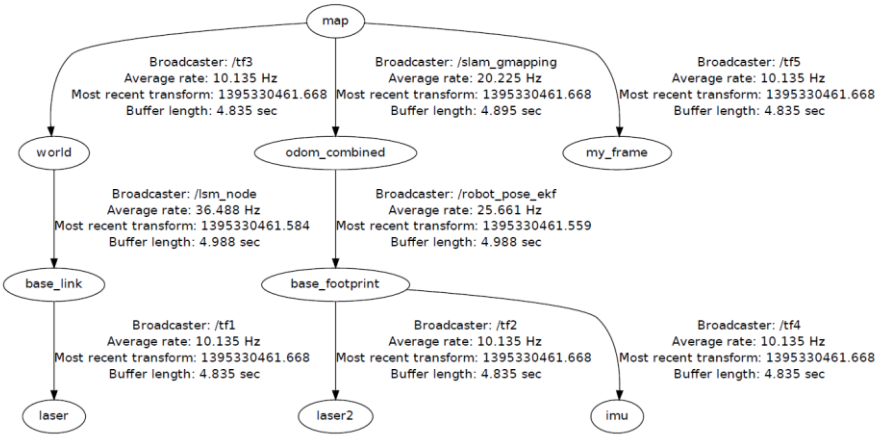


Şekil 6.1 Atalet duyargasız RPE ile dönüşüm ağacı

İlgili deney için gMapping çalıştırılmıştır. gMapping anlatılırken bahsedileceği gibi, gMapping, odometri verisiyle daha iyi sonuçlar üreten bir yöntemdir. Ne kadar iyi odometri bilgisine sahip olursa ürettiği sonuçlar da o kadar iyi olmaktadır. Bu bağlamda, sadece lazer mesafe ölçüm duyargası kullanılarak üretilen odometri verisi gMapping'e RPE filtresinden geçtikten sonra verilmiş ve ilgili bag dosyası üzerinde gMapping'in çalıştırılması sağlanmıştır. gMapping sonucu Şekil 6.2'deki gibi ortaya çıkmıştır. Şekil dikkatlice incelendiğinde sol üst taraftaki laboratuvardan başlamak üzere bir miktar kaymanın gerçekleştiği gözlemlenebilir. Bu kayma koridorda da iki tane duvar çizgisinin oluşmasına sebep olmuştur. Bu kaymanın temel sebebi, gelen odometri verisinin yetersizliğinden dolayı tarama eşleme sırasında eşlenecek taramalar arasındaki mesafelerin fazla olmasından ve bu sebeple taramaların eşlenememesinden kaynaklanmaktadır. Taramalar eşlenemediği durumda gMapping o an için en iyi parçacığın barındırdığı konum bilgisini kullanarak haritayı çizmekte (Bölüm 7'de gMapping anlatılırken bunun nasıl gerçekleştiğine detaylı bir şekilde değinilecektir) ve bu sebeple kayma gerçekleşmektedir. Burada gMapping'e verilen odometri bilgisi sadece lazer mesafe ölçüm duyargası kullanılarak LSM ile üretildiği için ve LSM de artımsal hatalar barındırdığı için gMapping sonucu belli bir oranda bozulma ile üretilmiştir.

Şekil 6.2 Atalet duyargasız RPE ile gMapping Sonucu

Bu deney kapsamında asıl gerçekleştirilmek istenilen atalet duyargası ile üretilen /imu/data bilgisini de kullanarak RPE'nin kombine bir odometri bilgisi üretmesini sağlamaktadır. Yani hem LSM tarafından üretilen /odom bilgisi RPE'ye verilecek, hem de atalet duyargasından alınan bilgi verilecektir. Daha fazla bilginin kullanıldığı bu durum için gMapping sonuçlarının sadece LSM'den üretilen /odom verisi kullanıldığı duruma göre daha iyi olması beklenmektedir. Şekil 6.3'te oluşturulan yeni kombinasyon için üretilmiş dönüşüm ağacı verilmiştir. Sağ alt tarafta görülebileceği gibi "imu" frame'i de "base_footprint" frame'ine bağlanmıştır. "base_footprint" frame'inden de /robot_pose_ekf mesajı "odom_combined" frame'ine bir bağlantı oluşturmuştur.



Şekil 6.3 Atalet duyargası kullanılarak RPE ile dönüşüm ağacı

Oluşturulan yeni kombinasyon için gMapping sonucuna Şekil 6.4'te yer verilmiştir. Tüm parametreler, bir önceki deneyle aynı olacak şekilde verilerek yapılan bu kontrollü deney sonucu Şekil 6.2 ile görsel olarak kıyaslanacak olursa sol üstteki laboratuvardaki kaymanın gerçekleşmediği ve aynı şekilde bu kaymanın koridora da etki etmediği gözlemlenebilir.

Burada RPE, kombine bir odometri bilgisi üretmiş hem tek başına LSM'nin ürettiği hem de atalet duyargasının ürettiği bilgiden daha iyi sonuçlar vermiştir. gMapping ise daha iyi odometri bilgisi ile taramalar arasındaki mesafeler azalacağı için daha az iterasyon ile tarama eşleştirmeyi gerçekleştirmekte ve daha iyi sonuçlar üretmektedir. Her iki deneme için de hazırlanan "launch" dosyası EK B'de verilmiştir.

Şekil 6.4 Atalet duyarga kullanılarak RPE ile gMapping Sonucu

Bu çalışmada, bu aşamadan sonra kombine bir odometri verisi üretebilmek için RPE kullanılacak ve gMapping'e odometri verisi bu şekilde sağlanmıştır.

BÖLÜM 7

GRIDMAPPING (gMapping)

Eşzamanlı konum belirleme ve haritalama (Simultaneous Localization and Mapping - SLAM) algoritmalarından olan gMapping (Grid Mapping), lazer verileri kullanarak harita çıkartan Rao-Blackwellized parçacık filtresi tabanlı oldukça verimli bir haritalama algoritmasıdır. Rao-Blackwellized parçacık filtreleri son zamanlarda SLAM problemlerini çözmede efektif bir yöntem olarak kullanılmaktadır [57]. Bu yöntemde, her biri bağımsız olarak kendi haritasını tutan parçacıklar kullanılmaktadır. Buradaki temel problemlerden birisi ilgili yöntemin düzgün sonuçlar verebilmesi için yüksek parçacıklarla çalışma gereksinimidir. Ancak gMapping kapsamında önerilen olasılıksal dağılım modeli sadece robotun hareketlerini değerlendirmemekte aynı zamanda robotun son gözlemlerini de hesaba katarak bir dağılım çıkartmaktadır.

Şekil 7.1 gMapping ile çıkartılan bir harita örneği

Bu yöntemle birlikte robotun konumuna olan inanç büyük oranda yükselmekte ve bu belirsizlik göreceli olarak ortadan kalkmaktadır. Bu sayede etkin bir şekilde haritalama çıkarımı yapılabilmektedir. gMapping ile yapılan örnek bir harita çıkarımına, "rviz" aracı üzerinde Şekil 7.1'de yer verilmiştir.

7.1 **Rao-Blackwellized Parçacık Filtresi ile Harita Çıkarımı**

SLAM algoritmalarında kullanılan Rao-Blackwellized Filter yöntemindeki temel fikir, m harita, $x_{1:t} = x_1, \dots, x_t$ robotun gezingesi, $z_{1:t} = z_1, \dots, z_t$ ortama dair gözlemler, $u_{1:t-1} = u_1, \dots, u_{t-1}$ mobil robot tarafından elde edilen odometri ölçümleri olmak üzere $p(x_{1:t}, m \mid z_{1:t}, u_{1:t-1})$ robotun o anki konumunu içeren bilgiyi tahmini olarak tutmaktır [58]. İlgili denklem sistemi sadeleştirilirse (6.1)'de verilen denklem elde edilir.

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t-1}) = p(m \mid x_{1:t}, z_{1:t}) \cdot p(x_{1:t} \mid z_{1:t}, u_{1:t-1}) \quad (6.1)$$

Bu sadeleştirme bize, robotun gezingesinin ilk tahminini ve bu tahmin üzerinden hesaplanan haritayı verir. Harita çıkarımı robotun poz tahminine oldukça bağlı olduğu için bu yaklaşımla verimli bir hesaplama yöntemi ortaya konulmuş olur. Potansiyel gezinge üzerinden $p(x_{1:t} \mid z_{1:t}, u_{1:t-1})$ poz tahmini için parçacık filtresi uygulanabilmektedir. Burada her bir parçacık robotun potansiyel gezinge bilgisini içermektedir. Ek olarak, her bir gezinge birbirlerinden bağımsız olarak harita bilgisini de içerir. Harita, parçacıklara karşılık gelen gezingeler ve gözlemler sonucunda elde edilir.

Haritalama için kullanılan Rao-Blackwellized Filter artımsal olarak duyargalardan okunan gözlemleri işler ve mümkün olan odometri okumalarını gerçekleştirir. Bu bilgileri kullanarak parçacıkların robotun gezingesine ait olasılıksal tahminlerini yeniden günceller. Yapılan işlemler aşağıdaki adımlardaki gibi özetlenebilir:

- Örnekleme: Gelecek nesil parçacıklar $\{x_t{}^{(i)}\}$, π önerilen dağılımı ile örneklenerek şimdiki nesilden $\{x_{t-1}{}^{(i)}\}$ elde edilir. Genellikle, olasılıksal odometri hareket modeli, önerilen dağılım için kullanılır.
- Ağırlıklandırma: (6.2) numaralı denklemde verilen önem örnekleme prensibine göre her bir parçacığa bireysel ağırlık dağılımı $w_t{}^{(i)}$ atanır.

$$w_t^{(i)} = \frac{p(x_{1:t}^{(i)} \mid z_{1:t}, u_{1:t-1})}{\pi(x_{1:t}^{(i)} \mid z_{1:t}, u_{1:t-1})} \tag{6.2}$$

- Yeniden Örnekleme: Parçacıklar kendi ağırlıklarına göre çizilir. Sürekli dağılıma yaklaşabilmek için bu adımda yeterli olacak şekilde sonlu sayıda parçacık kullanılır. Ayrıca, yeniden örnekleme hedef dağılım önerisi farklı olduğu durumlarda yeniden parçacık filtresi uygulamaya izin verir. Yeniden örneklemeden sonra bütün parçacıklar aynı ağırlığa sahip olurlar.
- Harita Çıkarımı: Her bir parçacık için, o parçacığa karşılık gelen harita tahmini $p(m^{(i)} \mid x_{1:t}^{(i)}, z_{1:t})$, $z_{1:t}$ gözlem geçmişine ve $x_{1:t}^{(i)}$ gezinge örneklerine dayanarak hesaplanır.

7.2 İyileştirmeler Sonucunda gMapping

Bir üst başlıkta anlatılan Rao-Blackwellized Filter yöntemiyle harita çıkarımına gMapping'de çeşitli eklemeler yapılmıştır. Öncelikle önerilen dağılımda hesap karmaşıklığını azaltacak şekilde normalizasyonlar yapılarak dağılımın kapsamı daraltılmış ve daha doğru sonuçlar üretilmesi sağlanmıştır. Ardından, yeniden örnekleme kısmında bir adaptasyon sağlanarak, yeniden örnekleme sırasında bütün parçacıkların değerlerini kaybetmemesi, onun yerine düşük olasılığa sahip parçacıkların yok edilmesi ve yerine yüksek olasılıklı parçacıklardan türeyen yeni parçacıkların konulması sağlanmıştır. Bu sayede yüksek olasılıklı parçacıklar yeniden örnekleme sırasında kaybolmayacağı için hesap karmaşıklığı oldukça azalacaktır [6]. Bu geliştirmeler sonucunda gMapping'in algoritma adımları aşağıdaki gibi olmuştur:

- i numaralı parçacıkla temsil edilen robotun pozu, başlangıç tahmini $x_t'^{(i)} = x_{t-1}^{(i)} \oplus u_{t-1}$ olacak şekilde bir önceki parçacığın pozundan $x_{t-1}^{(i)}$ ve en son filtre güncellenmesi sırasında toplanan odometri ölçülerinden elde edilir. Buradaki $\oplus$ operatörü standart poz birleştirme operatörünü temsil eder.
- $m_{t-1}^{(i)}$ haritası üzerinde başlangıç konumu, $x_t'^{(i)}$ olacak şekilde tarama eşleme (scan matching) algoritması çalıştırılır. Tarama başlangıç konumu olan $x_t'^{(i)}$ etrafında belirli sınırlar olacak şekilde gerçekleştirilir. Eğer tarama eşleme hata

verirse poz ve ağırlıklar hareket modeline göre hesaplanır ve 3. ve 4. adım yok sayılır.

- Tarama eşleme sonucunun bildirdiği robotun pozunun etrafında belirli bir örnekleme noktalarının kümesi seçilir. Buna dayanarak, önerinin ortalama değeri ve kovaryans matrisi, örneklenen x_j pozisyonlarında hedef dağılım $p(z_t \mid m_{t-1}^{(i)}, x_j)\, p(x_j \mid x_{t-1}^{(i)}, u_{t-1})$ noktasal olarak değerlendirilerek hesaplanır. Bu aşamada ağırlık vektörü $\eta^{(i)}$ de hesaplanır.
- i parçacığının yeni pozu $x_t^{(i)}$, önerilen dağılımın geliştirilmişi olan Gauss yaklaşımı $\mathrm{N}(\mu_t^{(i)}, \Sigma_t^{(i)})$ tarafından çizilir.
- Parçacıkların önem ağırlıkları güncelleştirilir.
- i parçacığının haritası $m^{(i)}$, çizilen $x_t^{(i)}$ pozu ve z_t gözlemine göre güncellenir.

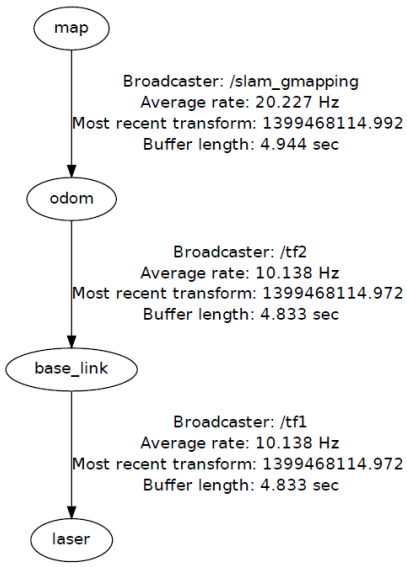
Yeni nesil örneklerin hesaplanmasından sonra yeniden örnekleme adımı N_{eff} değerine bağlı olarak gerçekleştirilir. N_{eff} değeri; $w'^{(i)}$, i parçacığının normalize edilmiş ağırlığını temsil etmek üzere (6,3) şeklinde hesaplanır.

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} (w'^{(i)})^2} \tag{6.3}$$

7.3 **ROS gMapping Paketi[46]**

gMapping eşzamanlı konum belirleme ve haritalama algoritması ROS Hydro sürümü üzerinde gerçekleştirilmiştir. Bu çalışma kapsamında “gmapping” tipinde “slam_gmapping” düğümü kullanılacak ve bunun ürettiği harita üzerinde çalışılacaktır. gMapping temelde sadece “hokuyo_node” düğümünün yayınladığı lazer mesafe ölçüm duyargasından elde edilen verileri barındıran “sensor_msgs/LaserScan” tipinde “/scan” mesajına abone olmaktadır. Bunun yanı sıra düğümlerin birbirleri ile ilişkilerini barındıran “tf/tfMessage” tipinde “/tf” mesajına abone olmaktadır. Bu mesaj içerisinde lazer, base_link ve odometri bilgilerinin birbirleri ile ilişkileri tutulmaktadır. Bu ilişkiler (dönüşümler) “slam_gmapping” düğümünün çalışması için gereklidir. Yani lazer mesafe ölçüm duyargasının konumunun (lazer) robot üzerinde nerede olduğu ve robotun

merkezinin (base_link) robotun odometrisine (odom) göre göreceli konumunun nerede olduğu bilgisini dönüşüm olarak vermek gerekmektedir. Bu dönüşümler "static_transform_publisher" düğümü kullanılarak statik olarak sağlanmaktadır. Şekil 7.2'de laser penceresinden base_link penceresine olan dönüşüm "tf1" ismi ile, base_link'ten de odom'a olan dönüşüm "tf2" ismi ile görülebilir.



Şekil 7.2 Robotun labirenti odometrisiz verisiyle gezerken oluşan dönüşüm ağacı

İlgili düğümler çalıştırıldıktan sonra slam_gmapping düğümü 2 farklı mesaj yayınlamaktadır:

- "nav_msgs/MapMetaData" tipinde "map_metadata" mesajı: Bu mesaj düğümün yayınladığı haritanın üst bilgilerini tutmaktadır. Bu bilgiler içerisinde haritanın boyutu, ne kadar sürede güncelleneceği, hangi pencere üzerinden kontrol edilebileceği ve çözünürlüğünün ne olduğu gibi bilgiler bulunmaktadır.
- "nav_msgs/OccupancyGrid" tipinde "map" mesajı: Bu mesaj içerisinde "slam_gmapping" düğümünün çıkarttığı harita bilgisi bulunmaktadır. Bu mesaj içerisinde yayınlanan haritanın sırası, pencere ismi (frame_id), yayınlandığı zamanın bilgisi ve haritanın ikili (binary) değerlerden oluşan bilgisi tutulmaktadır. Aynı zamanda robotun haritanın o an neresinde olduğunu tutan "odom" mesajını da barındıran bu mesajın ilgili parametre ile (map_update_interval) ne kadar bir sürede güncelleneceği belirlenebilmektedir.

“slam_gmapping” düğümü haritalamanın farklı değerlerle çalışabilmesi için çeşitli parametreler alabilmektedir. Bu parametrelerin neler olduğu ve ne amaçla kullanıldığı ve varsayılan değerleri EK C’de verilmiştir. Parametrelerin gMapping üzerine etkisine “gMapping Deney Sonuçları” bölümünde değinilmiştir.

7.4 gMapping Deneysel Sonuçları

ROS Hydro platformu üzerinde gerçekleştirilen ve bu çalışma kapsamında kullanılan “slam_gmapping” paketi farklı ortamlara ve duyargalara göre parametrik olarak çalışabilmektedir. Bu başlık altında ise bu parametrelerden bazılarının incelenmesine yer verilmiştir. İlgili deneyler, LSM deneylerinin yapıldığı ortam ile aynı ortamda yapılmıştır. Deney ortamının bir görseline ve planına Şekil 5.2’den ulaşılabilir.

Deney kapsamında gMapping’in performansının kıyaslanabilmesi için 4 farklı hızda (4 cm/sn, 10 cm/sn, 18 cm/sn ve 41 cm/sn) labirent gezilmiştir. Labirent içerisindeki robotun her bir farklı hızı için “.bag” dosyaları alınmış ve gMapping çevrimdışı olarak farklı parametrelerle çalıştırılmıştır. Bu sayede aynı gezinme için farklı parametrelerin kullanılması sağlanmış ve parametreler birbirleri ile kıyaslanabilir hale gelmiştir. gMapping’in çalıştırılmasında kullanılan launch dosyası EK B’de verilmiştir.

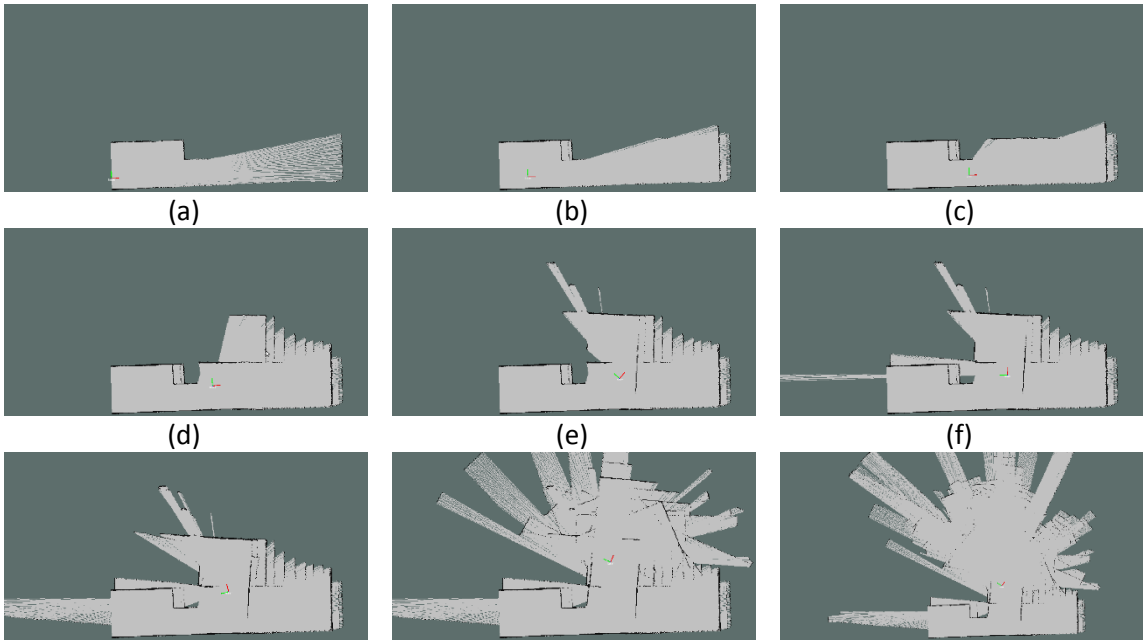
7.4.1 4 cm/sn Hızla Yapılan Deney Sonuçları

Daha önceden bahsedildiği gibi gMapping ardışık zamanlarda alınan taramaların eşleştirilmesi sonucu harita çıkarımı yapmaktadır. Robotun göreceli olarak daha yavaş gezdiği gezinimlerde taramalar arası mesafe birbirine yakın olacağı için sonuçların daha iyi çıktığı, hızın arttırıldığı gezinimlerde ise hızla doğru orantılı olarak hatanın arttığı gözlemlenmiştir. Taramalar arasındaki mesafenin artması ile ilgili taramaların birbirlerine eşleştirilmesi sırasında hesaplanan iterasyonlar doğrusal olarak artacaktır. Bu da sisteme, mesafenin artışıyla doğrusal olarak bir yük getirecek ve haritanın çıkarımı gerçek zamanlı olmaktan uzaklaşacak hatta oldukça kötü sonuçlar verebilecektir. Bu bağlamda gMapping her ne kadar odometri bilgisi olmadan da çalışabilse de (her iterasyonun başlangıcında, odometri bilgisi gelmediği durumda, odometri bilgisini boş bir veri olarak kabul etmektedir) odometri bilgisine de ihtiyaç duymaktadır. gMapping, odometri bilgisi kullanarak belli bir hata payıyla robotun

ötelenmesi (translasyon) ve dönmesi (rotasyon) ile ilgili bilgi sahibi olmaktadır. Örneğin eşleştirilecek her iki tarama arasında 5 cm ve 30° fark olduğunu düşünürsek; odometri olmadığı durumda her iki tarama böyle bir poz değişimine ulaşasıya kadar iteratif olarak hesaplanacaktır; odometri olduğu durumda ise odometri bilgisi gMapping'e belirli bir hata payıyla pozdaki değişimi verecektir (Örneğin: 4,32 cm ve 27°). Böylece gMapping'in eşleştireceği taramalar arasındaki mesafe 5 cm'den 0,68 cm'e 30°'den 3° düşecektir ki bu hesaplamanın daha az iterasyonda gerçekleşmesini sağlayacaktır.

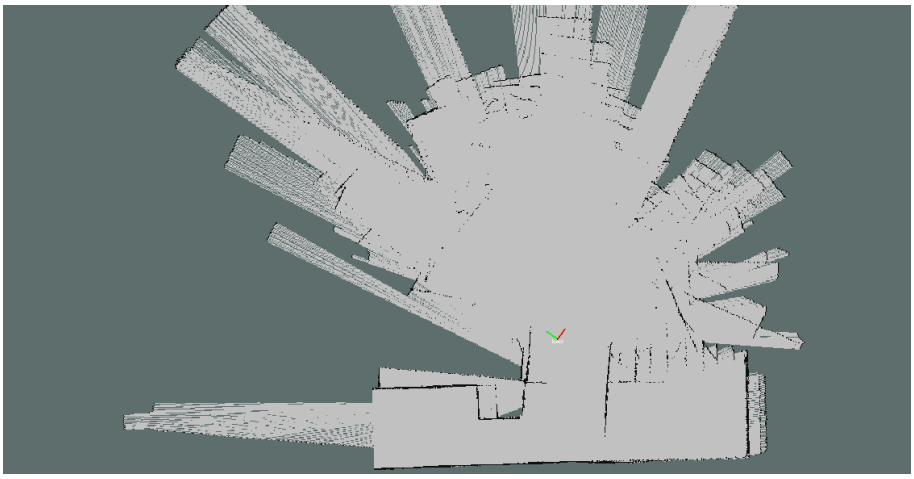
gMapping'in varsayılan parametreleri kullanıldığı durumda, 4 cm/sn hızla gezilen "bag" dosyası üzerinden odometrili ve odometrisiz gMapping sonuçları elde edilmiştir. Şekil 7.3'de odometrisiz gMapping sonuçlarına adım adım yer verilmiştir. Haritalar "rviz" ekranı üzerinden alınmıştır. Harita üzerinde kırmızı ve yeşil ile gösterilen belirteç robotu temsil etmektedir. Kırmızı renk ile gösterilen yön robotun baş tarafını temsil etmektedir. Bu şekilde robotun harita üzerindeki o anki pozu (rotasyon + translasyon) verilmiştir. Şekil 7.3-a'da robotun haritalamaya başladığı noktada kendisine 600 cm uzakta olan duvarı olması gerektiği yerde çizdiği ancak Şekil 7.3-b'ye gelindiğinde duvarı yaklaşık 20 cm kadar kaydırdığı ve aynı hatayı koridor üzerinde bulunan bir kutu yardımıyla oluşturulmuş engele de yansıttığı görülmüştür. Şekil 7.3-c'de aynı koridor üzerinde engeli geçesiye kadar bir hataya rastlamadığı gözlemlenebilir. Burada hatasız ilerlemesinin temel sebebi koridor üzerinde bulunan engele yakın olmasıdır. Her ne kadar odometri kullanılmadığı durumlarda taramalar arasındaki mesafe fazla olsa da taranan verinin robota yakın olması durumunda bu mesafede hata yapılma oranı azalmaktadır ki bu de Şekil 7.3-b ile Şekil 7.3-c arasındaki geçişten gözlemlenebilmektedir. Şekil 7.3-d'ye gelindiğinde ise hatanın büyüdüğü görülebilir. Çünkü robot karşısındaki duvara yaklaşık 300 cm uzaklıktadır ve bu uzaklıktaki taramalar arasında mesafe çok büyük olduğundan tarama eşleme yapılamamaktadır. Ardından robot koridorun sonunda dönmeye başlamıştır. Odometrisiz durumda taramalar arasındaki mesafe farkı ve hata oranı dönüşlerde daha fazla olmaktadır. Bu da Şekil 7.3-e'nin devamında görülebilmektedir. İlk koridorun sonundaki dönüşten sonra harita tamamen bozulmuş ve gerçeği hiçbir şekilde yansıtamadığı görülmüştür. Labirentin tam bir tur gezilmesi ile çıkartılan haritaya Şekil 7.4'te yer verilmiştir. Buradan da görülebileceği üzere gMapping, ilk dönüş sonrasında yaptığı hatayı

odometri bilgisinin yokluğundan dolayı bir daha düzeltememiş ve tarama verileri sürekli gelse de konumunu değiştirememiştir.



Şekil 7.3 gMapping parametrelerinin varsayılan olduğu durumda 4 cm/sn hızla gezilen bag dosyası üzerinden odometrisiz gMapping sonuçları

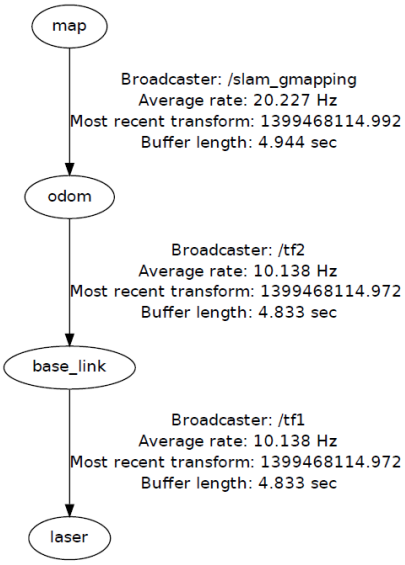
Robotun neredeyse en düşük hızı olan 4 cm/sn hızla gezildiği durumda bile gMapping odometrisiz düzgün sonuçlar verememiştir.



Şekil 7.4 gMapping parametrelerinin varsayılan olduğu durumda 4 cm/sn hızla gezilen bag dosyası üzerinden odometrisiz gMapping'in tam bir tur atılması sonucu

Robotun labirenti odometrisiz gezdiği durum için çalıştırılan düğümlerin birbirleriyle bağlantısını ve ne şekilde bağlandığını gösteren dönüşüm ağacı Şekil 7.5'te verilmiştir. /laser frame'inden /base_link frame'ine ve /base_link frame'inden /odom frame'ine

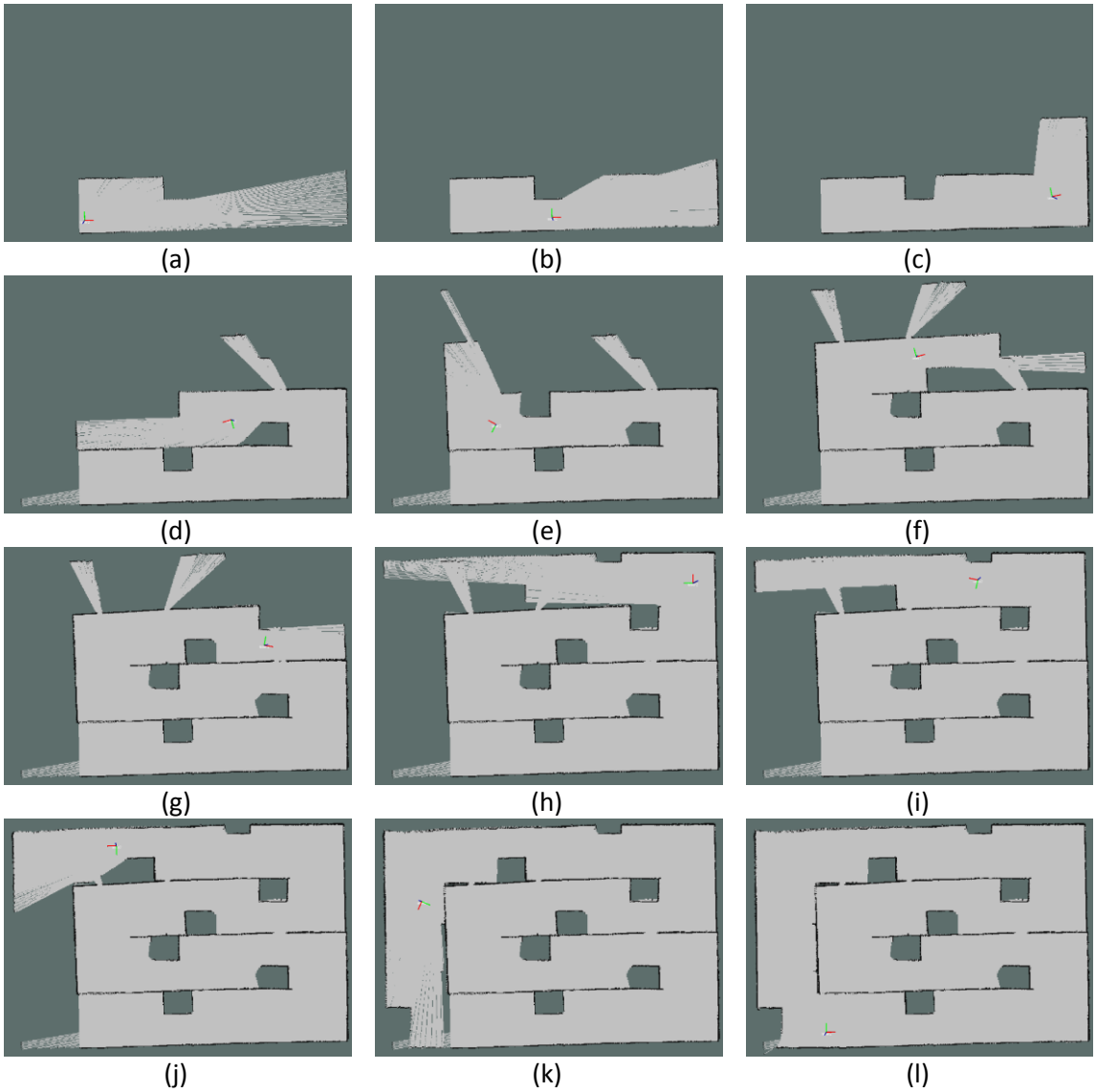
bağlantılar “static_transform_publisher” paketi kullanılarak statik olarak gerçekleştirilmiştir. /odom frame’inden /map frame’ine olan bağlantıyı ise gMapping üretmiştir.



Şekil 7.5 gMapping'in odometrisiz çalıştırıldığı durum için dönüşüm ağacı

Aynı hızla (4 cm/sn) gezilen bag dosyası üzerinden gMapping’e odometri bilgisi sağlanarak alınan sonuçlar Şekil 7.6 ve Şekil 7.7’de verilmiştir. gMapping’e odometri bilgisi tekerlek odometrisi bilgisi yerine, LSM’den elde edilen “geometry_msgs/Pose2D” tipinde “/pose2D” mesajı kullanılarak üretilen “nav_msgs/Odometry” tipindeki “/odom” mesajı ile sağlanmıştır. Bu dönüşümü yapan paket yazılırken “/pose2D” mesajı içerisinde eksik olan kovaryans bilgileri eklenerek “/odom” mesajı üretilmiştir. Mesaj LSM’nin tarama eşleme sonucunda elde ettiği robotun pozunu içermektedir. gMapping bu poz bilgisini kullanarak tarama eşleme sırasında daha az iterasyon yaptığı için doğru sonuçlara ulaşması daha mümkün olmuştur. Hiç kuşkusuz LSM’nin ürettiği odometri bilgisi artımsal gürültülü bir odometri bilgisidir. Ancak LSM o anki odometri bilgisini yayınlarken, gMapping bu mesajdan, ardışık iki odometri bilgisini çıkartarak poz bilgisi elde etmektedir. Böyle artımsal hata gMapping’e etki etmemektedir, sadece o tarama eşleştirmeye dair hata gMapping’e yansımaktadır. Şekil 7.6’da bir üstteki deneye odometri bilgisi eklendiği durumda elde edilen kontrollü deney sonuçlarına adım adım yer verilmiştir. Şekil 7.6-a’dan Şekil 7.6-l

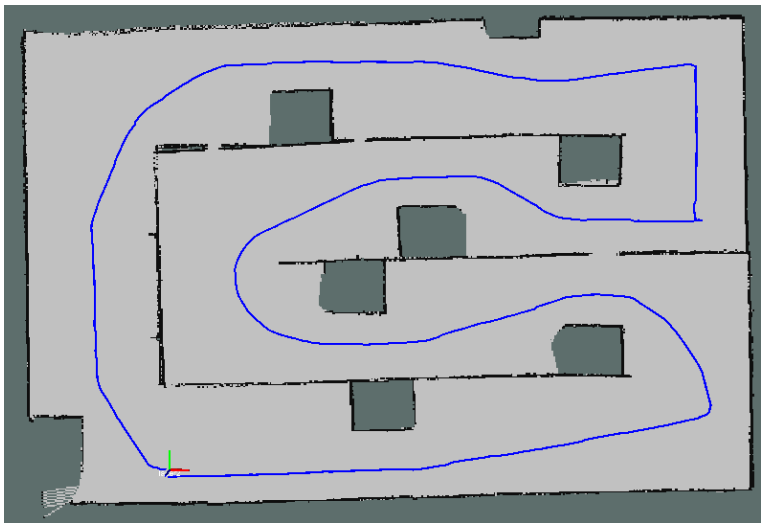
adımına kadar görülebileceği üzere haritanın bütünlüğünü bozacak şekilde bir hata oluşmamış ve ortamın planına (Şekil 5.2-b) yakın bir sonuç elde edilmiştir.



Şekil 7.6 gMapping parametrelerinin varsayılan olduğu durumda 4 cm/sn hızla gezilen bag dosyası üzerinden odometrili gMapping sonuçları

Koridorlar içerisinde engel oluşturması için konulan kutuların şekillerinin bozuk çıktığı görülebilir. Bu bozukluk robotun 180°'lik açılarda tarama ölçümleri alması sonucunda koridordan geçmesi esnasında lazer tarafından kutuların o bölgelerinin görünememesine dayanmaktadır. 180°'den daha büyük bir açıyla labirentin gezilmesi durumunda bu şekil bozukluklarının da ortadan kalkacağı görülecektir. Bu deneye tezin ilerleyen aşamalarında yer verilmiştir.

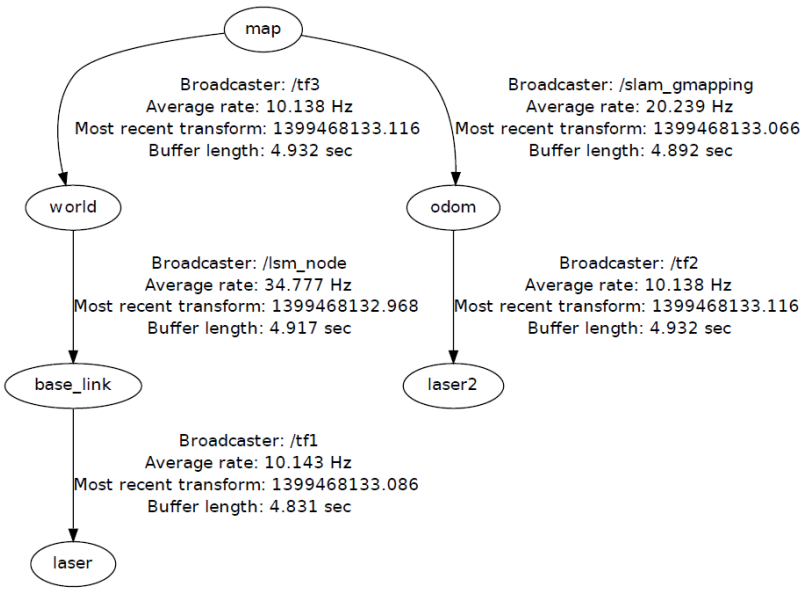
Robotun labirent içerisindeki geziniminin nasıl gerçekleştiğini daha iyi görebilmek ve deneysel sonuçları daha iyi analiz edebilmek için robotun labirent içerisinde gezingesinin çıkarılması amaçlanmıştır. Anlık odometri bilgileri (x, y) bir diziye atılmıştır. Yani anlık odometri bilgilerinin bir dizisi oluşturulmuştur. Bu dizi "visualization_msgs/Marker" tipindeki "visualization_marker" mesajı oluşturularak görselleştirilmiş ve "rviz" ekranında harita üzerinde gösterilmesi sağlanmıştır. Şekil 7.7'de ilgili deney sonucunda elde edilen gezingenin nihai şeklinin harita üzerinde gösterimi verilmiştir. Gerçekte robot başladığı noktada gezinimini bitirmiştir. Ancak gezingenin çıkarımından da anlaşılacağı gibi harita çok az bir kaydırma hatası ile üretilmiştir. Bu hatanın oluşumu, Şekil 7.6-k'da haritanın sol tarafındaki koridorun oluşturulması sırasında gerçekleştiği gözlemlenebilir.



Şekil 7.7 gMapping parametrelerinin varsayılan olduğu durumda 4 cm/sn hızla gezilen bag dosyası üzerinden odometrili gMapping'in ve gezingenin tam bir tur atılması sonucu

Robotun labirenti odometrili gezdiği durum için çalıştırılan düğümlerin birbirleriyle bağlantısını ve ne şekilde bağlandığını gösteren dönüşüm ağacı Şekil 7.8'de verilmiştir. /laser frame'inden /base_link frame'ine, /laser2 frame'inden /odom frame'ine ve /world frame'inden /map frame'ine bağlantılar "static_transform_publisher" paketi kullanılarak statik olarak gerçekleştirilmiştir. /odom frame'inden /map frame'ine olan bağlantıyı gMapping, /base_link frame'inden /world frame'ine olan bağlantıyı ise LSM üretmektedir. Yukarıda da değinildiği gibi LSM odometri bilgisi üretmek için kullanılmaktadır. Şekil 7.8'de görülebileceği üzere ağaç ikiye dallanmıştır. /laser

frame'inin bağlı olduğu dal odometri bilgisini üretmektedir. /laser2 frame'inin bağlı olduğu dal ise üretilen odometri bilgisini kullanarak haritalamayı gerçekleştirmektedir. Bir diğer dikkat edilecek husus ise tek bir lazer mesafe ölçüm duyargası kullanılmasına rağmen birden fazla /laser frame'i ağaçta görülebilmektedir. Burada hem LSM hem de gMapping üretilen tarama verilerini kullanmakta ve bu tarama mesajına ait frame'i manipüle etmektedir. Yani eğer sadece /laser frame'i olsaydı ve LSM ve gMapping ikisi de aynı frame üzerinden çalışsaydı, her ikisi de aynı frame'i manipüle edeceği için, /laser frame'inde sürekli atlamalar gerçekleşecekti. Deney kapsamında bu gözlemlendiği için lazer mesafe ölçüm duyargasından alınan taramalar çoklanarak farklı frame isimleriyle sisteme verilmiş bu sayede her iki düğümün de farklı frame'leri kullanabilmesine imkan sağlanmış ve daha düzgün sonuçlar elde edilmiştir.

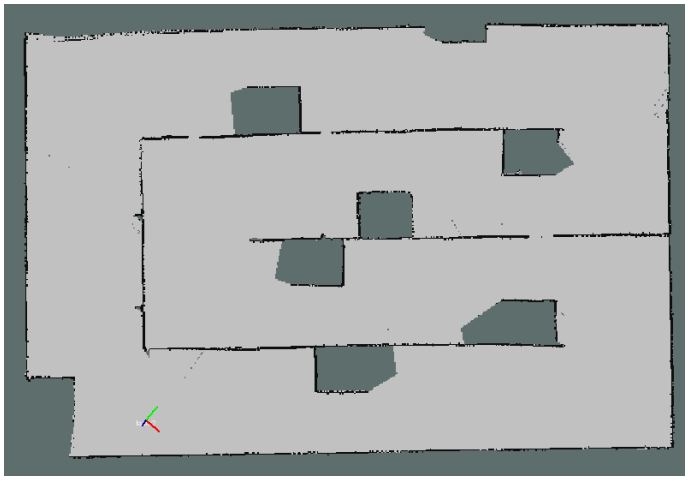


Şekil 7.8 gMapping'in odometrili çalıştırıldığı durum için dönüşüm ağacı

7.4.2 10 cm/sn Hızla Yapılan Deney Sonuçları

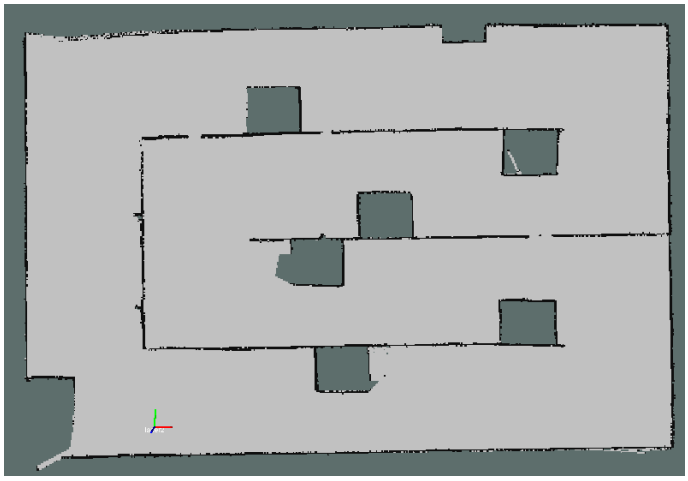
Daha önceden de bahsedildiği gibi belirli bir hıza kadar gMapping'in odometrili çalıştırılmasıyla doğru sonuçlar üretmesi, belirli bir hız değerinden sonra da sonuçların bozulması beklenmektedir. Bu kapsamda, labirent 10 cm/sn hızla 2 tur gezilmiş ve bir "bag" dosyası alınmıştır. 1. tur sonucunda haritalamanın düzgün neticeler üretmesiyle, robotun 2. bir turu atmasının nasıl bir etkisi olacağı araştırılmak istenmiş bunun için 2 turluk "bag" dosyası toplanmıştır. Şekil 7.9'da görülebileceği üzere, robotun labirentte

10 cm/sn hızla 180°'lik lazer taraması alarak 1 tam tur atması sonucunda elde edilen harita oldukça düzgün çıkmıştır. Ancak 4 cm/sn hızla gezilen haritaya göre koridorlardaki kutuların deformasyonu artmıştır. Bunun temel nedeni, iki buçuk kat daha hızlı gezildiği için taramalar arasındaki mesafe artmış ve bunun sonucu olarak kutuların çıkarımındaki hatalar fazlalaşmıştır.



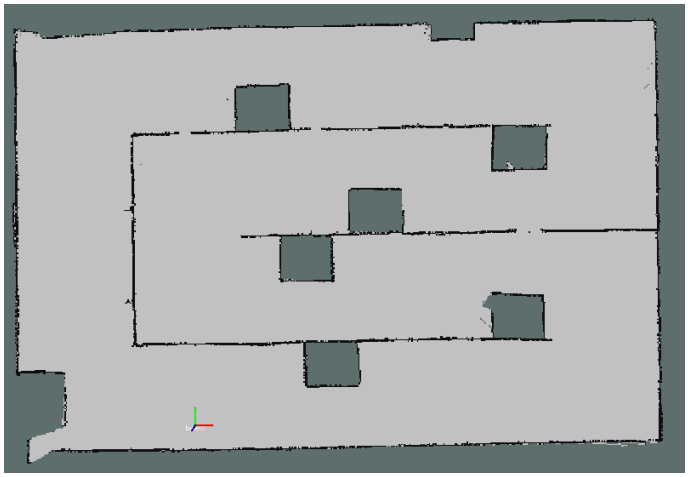
Şekil 7.9 gMapping parametrelerinin varsayılan olduğu durumda 10 cm/sn hızla gezilen bag dosyası üzerinden odometrili gMapping'in tam bir tur atılması sonucu

1. turun ardından 2. tur atılmasıyla elde edilen sonuç Şekil 7.10'da verilmiştir. Görülebileceği üzere haritanın detayları Şekil 7.9'a göre daha artmış ve koridorlardaki kutuların şekil bozuklukları oldukça giderilebilmiştir. Buradan şu sonuca varılabilmektedir: Lazerin birden fazla defa gördüğü yerleri gMapping'in harita üzerinde daha iyi çizmesi söz konusudur.



Şekil 7.10 gMapping parametrelerinin varsayılan olduğu durumda 10 cm/sn hızla gezilen bag dosyası üzerinden odometrili gMapping'in iki tur atılması sonucu

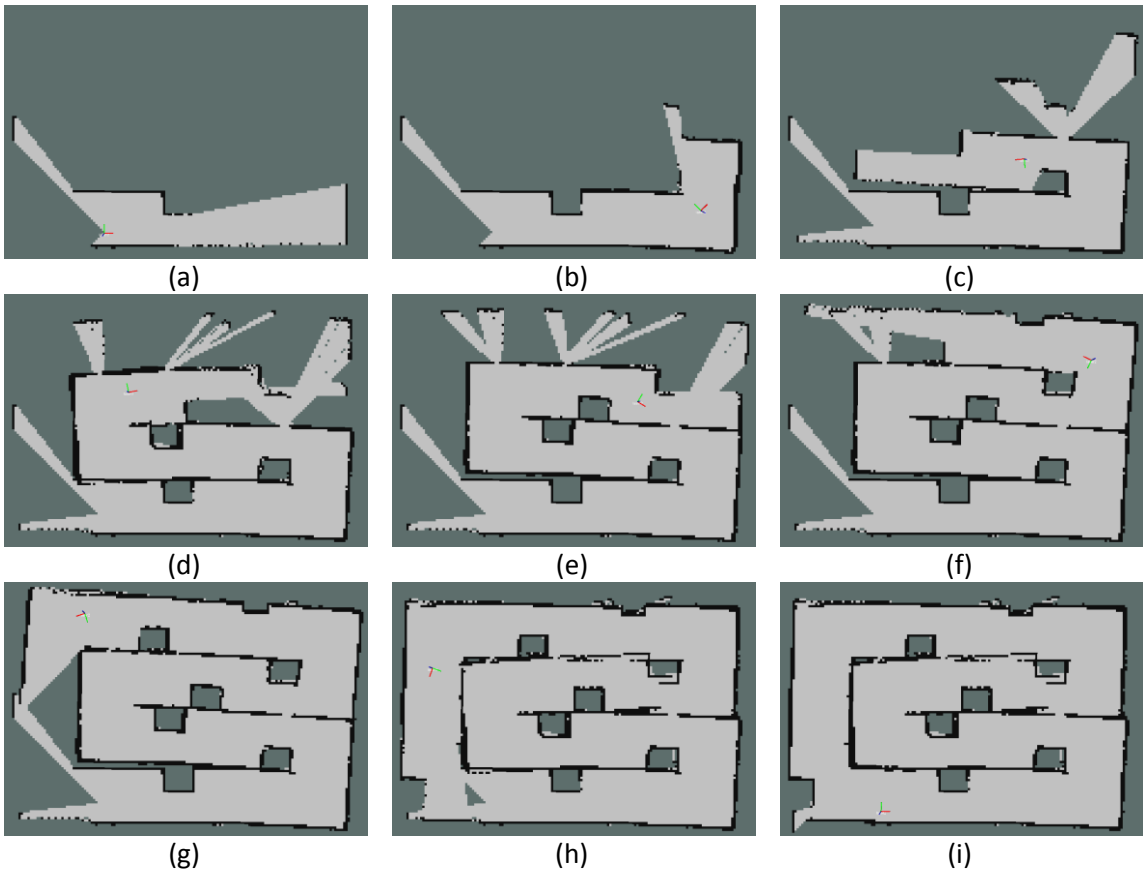
Bu sadece aynı yerden iki kere geçmek gerekmekte anlamına gelmemektedir, aynı zamanda lazerin görüş açısının genişletilmesiyle de sonuçların daha iyi çıkabileceği tahmin edilmiş ve lazerin 270°'lik açılarla tarama aldığı bir "bag" dosyası oluşturularak 10 cm/sn hızla yeniden labirent bir tam tur gezilmiş ve Şekil 7.11'deki sonuç elde edilmiştir. Şekil 7.11'den görülebileceği gibi lazerin tarama açısının genişletilmesiyle aynı noktaların lazer tarafından farklı açılarla görülebilmesi sağlanmış ve bu sayede aynı hızla aynı ortamda iki tur gezilmesinden (Şekil 7.10) bile daha iyi sonuçlar elde edildiği görülmüştür. Özellikle kutuların deformasyonu minimuma inmiş ve haritada görülemeyen (Örneğin: sol alt köşedeki duvar) noktalar lazerin açısının genişletilmesiyle tespit edilerek harita üzerinde çizilebilmiştir. Hiç kuşkusuz daha fazla lazer verisinin gelmesiyle tarama eşlemenin hesaplanması sırasında eşlenecek noktaların sayısı arttığı için sisteme ekstra bir yük yüklenmiştir. Ancak haritalamanın daha doğru ve ayrıntılı sonuçlar vermesi ile karşılaştırılacak olursa bu yük daha doğru sonuçlar alabilmek için katlanabilecek bir yük olarak değerlendirilebilir.



Şekil 7.11 gMapping parametrelerinin varsayılan olduğu durumda 10 cm/sn hızla ve 270°'lik lazer açısıyla gezilen bag dosyası üzerinden odometrili gMapping'in bir tam tur atılması sonucu

Bu aşamadan sonra gMapping'in çeşitli parametreleri değiştirilerek deneyler yapılmış ve bu parametrelerin ortama uygun seçilmesinin önemi gözlemlenmiştir. Özellikle haritası çıkartılmak istenilen ortamın büyüklüğüne göre haritanın çözünürlüğünün seçilmesinin oldukça önem arz ettiği keşfedilmiştir. Haritanın çözünürlüğü gMapping'in "delta" parametresi üzerinden ayarlanabilmektedir. Bu değerin düşük seçilmesiyle haritanın çözünürlüğü artmakta, bununla beraber engel kabul edilen nesnelerin

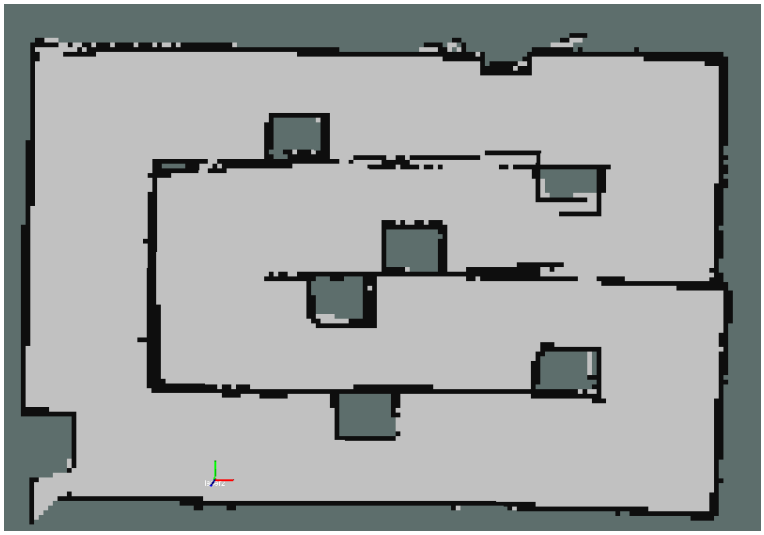
kenarlarının harita üzerinde çizimi oldukça ince olmakta ve boyut olarak küçük ortamlarda daha net haritaların çıkması sağlanmaktadır. Labirent içerisinde gezinirken varsayılan olarak 0.015 değeri alınmıştır. Çünkü labirentin duvarları 1 cm kalınlıkta olduğundan bu duvarların kalın çizilmesi sonucu haritalamanın yanlış çıkabileceği gözlemlenmiştir. Şekil 7.12'de bu deneye ait sonuçlar verilmiştir. Bu sonuçlar "delta" değerinin 0.05 seçilmesi sonucu elde edilmiştir. Görülebileceği üzere 1 cm olan duvar kalınlıklarının 5 cm olarak çizilmesi sonucunda bir önceki deney ile aynı hızda ve aynı lazer açısı ölçümleriyle deney yapıldığında haritanın daha fazla bozulmuş olarak üretildiği sonucu elde edilmiştir. Özellikle Şekil 7.12-c'den sonraki görsellerde ikinci koridorun birinci koridorla birleşik çizilmesi gerekirken araya boşluk bırakarak çizilmesi koridorun yamuk çizilmesine sebep olmuştur.



Şekil 7.12 "delta" değerinin 0.05, diğer gMapping parametrelerinin varsayılan olduğu durumda 10 cm/sn hızla ve 270°'lik lazer açısıyla gezilen bag dosyası üzerinden odometrili gMapping sonuçları

Tabi ki gMapping global olarak çalışan bir algoritmadır. Şekil 7.12-h'ye gelindiğinde bu hata fark edilip düzeltilmiş olsa da Şekil 7.13 ile Şekil 7.11 karşılaştırıldığında "delta"

değerinin 0.05 alındığında ciddi manada hatalar olduğu gözlemlenebilmektedir. Şekil 7.13'de duvar kalınlıklarının artması ile birlikte aynı duvarın birkaç kez çizildiği (en üstteki koridorun sağ köşesinde) ve bunun haritayı ciddi manada bozduğu gözlemlenebilmektedir.



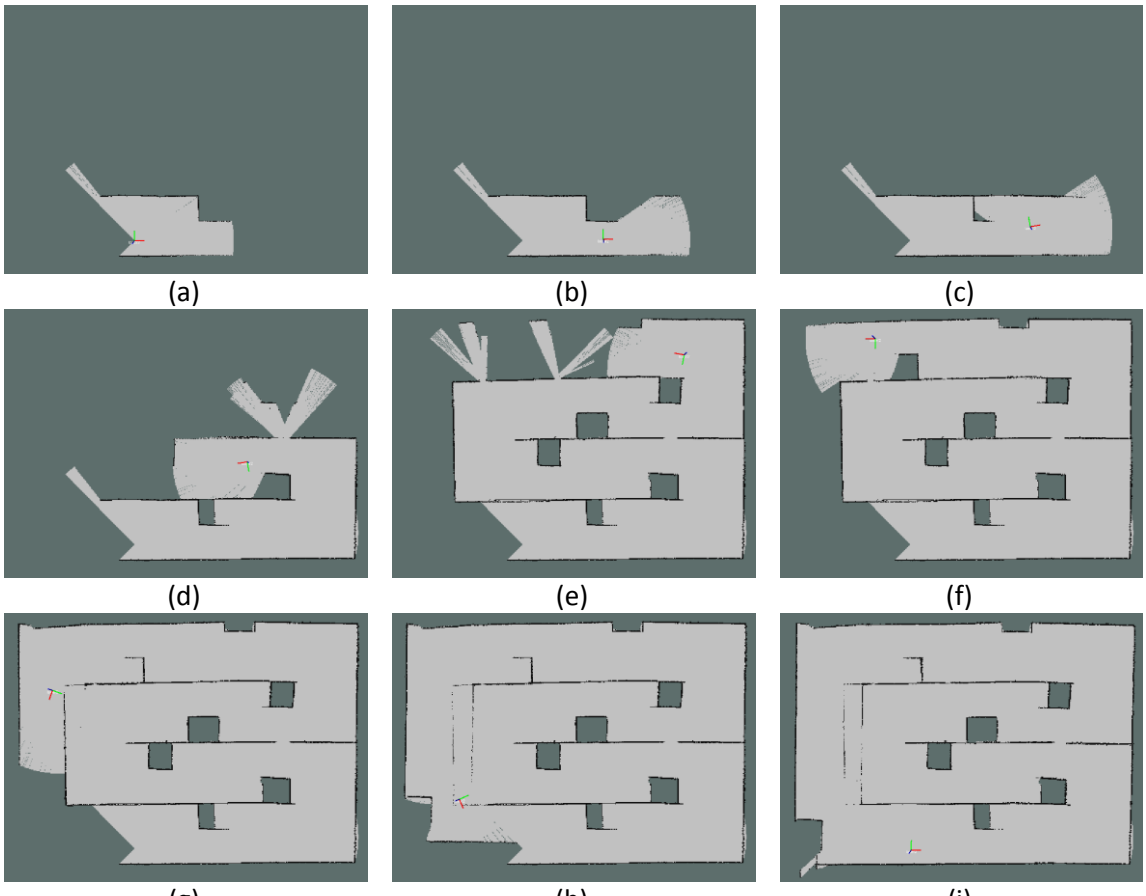
Şekil 7.13 "delta" değerinin 0.05, diğer gMapping parametrelerinin varsayılan olduğu durumda 10 cm/sn hızla ve 270°'lik lazer açısıyla gezilen bag dosyası üzerinden odometrili gMapping bir tam tur atılması sonucu

Diğer taraftan haritalanan ortamın duvarlarının kalın olduğu yerlerde "delta" değerinin büyük seçilmesi sonuçların daha başarılı çıkmasını sağlamaktadır. Bu durum Şekil 7.14'de gözlemlenebilmektedir. Bu deney alanı Yıldız Teknik Üniversitesi Bilgisayar Mühendisliği Bölümü'nün zemin katının koridorunu kapsamaktadır. Gezinme esnasında bazı odalara da girilmiştir. "delta" değerinin 0.05 olduğu bu haritalamanın gezilen alanın büyüklüğü sebebiyle (boyu yaklaşık 60 metre) bozulmadığı gözlemlenmiştir. Böyle bir ortamda küçük değerler seçildiğinde birbirleriyle eşleştirilecek az nokta olacağı için, başka bir deyişle birbirleriyle eşleştirilecek çizgilerin ince olması sebebiyle, eşleştirme de kaymalar meydana gelmekte ve harita daha başlarken bozulmaktadır. Bir diğer deney kapsamında gMapping'in "urange" parametresi değiştirilmiştir. Bu parametre ile lazer mesafe ölçüm duyargasının maksimum ölçüm mesafesi gMapping'e bildirilmektedir. Deney kapsamında 30 metreye kadar ölçüm yapabilen bir lazer kullanıldığından daha önceden bahsedilmişti. Bu aşamaya kadar yapılan deneylerde lazerden maksimum ölçüde faydalanabilmek için "urange" değeri 30 olarak ayarlanmıştı.

Şekil 7.14 “delta” değerinin 0.05, diğer parametrelerin varsayılan olarak seçildiği gMapping sonucu

Bu değer, gezilen labirentin koridorlarının boyutundan çok çok büyük olduğu için lazerin ölçüm mesafesi ile ilgili bir sıkıntı çıkmadı. Ancak lazerin ölçüm mesafesinden daha uzun koridorlar olsaydı aynı sonuçlar elde edilebilir miydi sorusunu cevaplayabilmek için lazerin “urange” parametresi 2 olarak (2 metre) ayarlandı ve koridorların tamamının lazerin görüş açısından çıkması sağlandı. Diğer parametrelerin varsayılan olduğu bu kontrollü deneyin sonucu Şekil 7.15’te verilmiştir. Şekil 7.15-a’da görüleceği gibi haritalama başladığında lazerden okunan değerlerin sadece ilk 2 metresi anlamlı olduğu için koridorun sonundaki duvar görülememektedir. Bu duvar ancak Şekil 7.15-d’ye gelindiğinde görünebilmiştir. Lazer mesafe ölçüm duyargasının maksimum ölçüm değeri koridorun uzunluğundan az olmasındaki temel problem, robotun koridor içerisinde ilerlemesiyle lazer ölçümlerinin değişmemesidir. Aynı alınan taramalar birbirine çok benzer olacağı için robot kendini ilerlemiş olarak kabul etmeyecektir. Bu durumdan odometri bilgisini kullanarak kurtulmak mümkün olabilmektedir. Mesela tekerlek odometrisi kullanılırsa, robotun ilerlediği bilgisi alınacaktır ve harita o şekilde güncellenecektir. Ancak bu tez çalışması kapsamında tekerlek odometrisi yerine LSM’den elde edilen lazer tarama eşlemeye dayalı odometri kullanıldığı için odometri bilgisi de pozun değişmediği söylemektedir. Şekil 7.15-b’de robotun koridor boyunca düzgün bir şekilde ilerlediği görülmektedir. Çünkü lazerin görüş açısında sürekli değişen bir engel vardır. Koridor içerisinde bulunan bu kutu

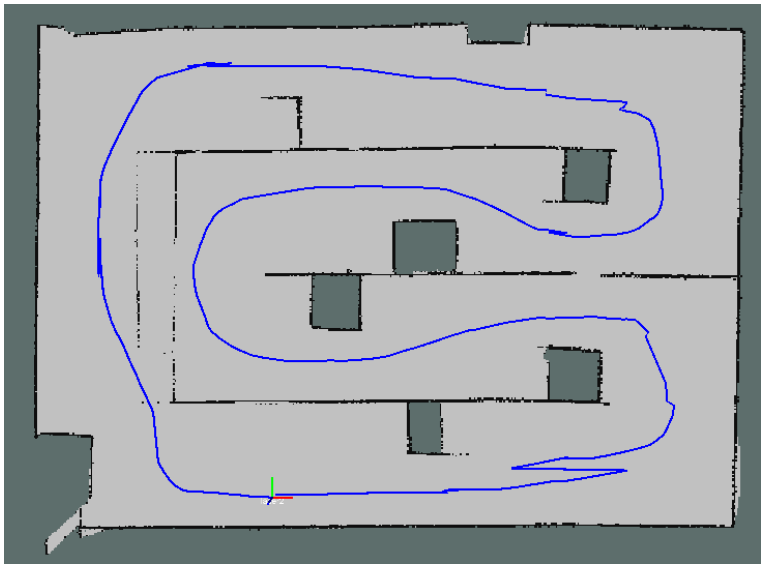
sayesinde farklı taramalar alınarak robotun ilerlediği kestirilebilmektedir. Ancak bu kutu geçildiğinde ki Şekil 7.15-c'de görülmektedir, robot kendisini ilerlemiş olarak kabul etmemekte ve haritayı da ona göre çizmektedir. Şekil 7.16'da robotun 1 tam tur dönüşü sonucunda çıkarttığı gezingeye yer verilmiştir. Bu gezingeden de görülebileceği üzere, robotun pozu ilk koridorun sonlarına doğru atlayış göstermiştir ve 6 metrelik koridorun daha kısa boyutlarda çizilmesine sebep olmuştur. Bu hata diğer koridorlar da gezilirken devam etmiştir. Şekil 7.15-g'den Şekil 7.15-h'ye geçişte görülebileceği gibi haritanın kısaltılması için bir duvar silinmiş yerine yenisi çizilmiştir.



Şekil 7.15 "urange" değerinin 2, diğer gMapping parametrelerinin varsayılan olduğu durumda 10 cm/sn hızla ve 270°'lik lazer açısıyla gezilen bag dosyası üzerinden odometrili gMapping sonuçları

İlgili deneye ait Şekil 7.16'daki gezinge incelendiğinde koridorların çeşitli yerlerinde odometri bilgisinde atlamaların olduğu ve bu atmalar sonucunda haritanın olması gerektiğinden farklı boyutlarda çizildiği görülmüştür. Sonuç olarak, lazer mesafe ölçüm duyargasının maksimum mesafesinin gezilen ortamın benzer alanlarının uzunluğundan

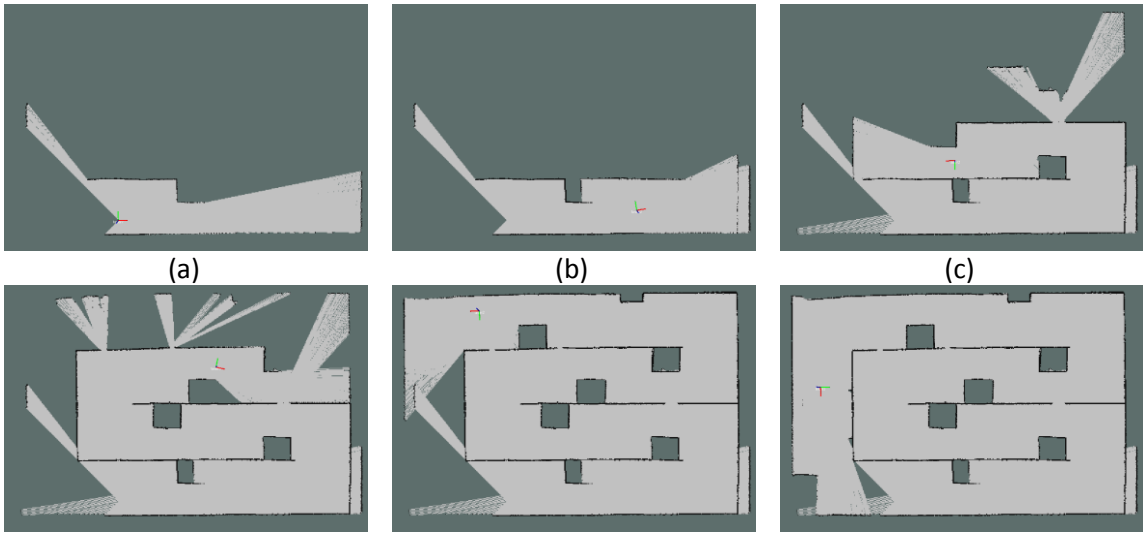
daha fazla olması gerektiği, eğer olmazsa harita çıkartılırken boyutsal farklılıkların ortaya çıkabileceği tespit edilmiştir.



Şekil 7.16 “urange” değerinin 2, diğer gMapping parametrelerinin varsayılan olduğu durumda 10 cm/sn hızla ve 270°’lik lazer açısıyla gezilen bag dosyası üzerinden odometrili gMapping’in ve gezingenin tam bir tur atılması sonucu

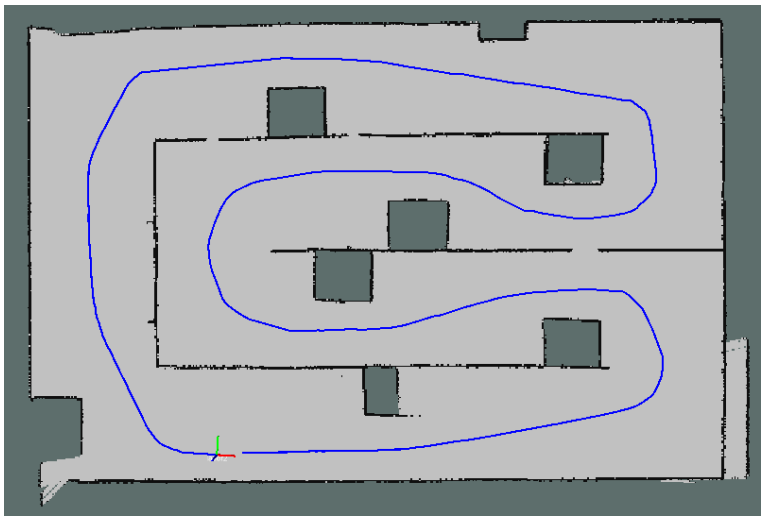
Bir diğer deneyde gMapping’in “particles” parametresi değiştirilerek farklı sonuçlar elde edilmiştir. Daha önce değinildiği gibi gMapping parçacık tabanlı çalışan bir algoritmadır. ROS Hydro ortamında gerçekleştirilen paket parametrik olarak parçacık sayısını alabilmektedir. gMapping algoritması parçacık sayısı için varsayılan olarak 30 almaktadır. Buraya kadar yapılan deneyler 30 parçacık üzerinden yapılmıştır. gMapping olasılıksal bir yöntem olduğu için tek bir parçacıkla da doğru sonuçlar üretmesi olasıdır. Ancak birden fazla parçacıkla çalışılması halinde olasılıksal olarak daha çok alternatif olacağı için doğru sonuç üretme ihtimali o kadar yükselecektir. Tabi ki parçacık sayısının artışıyla beraber işlem karmaşıklığı da artacaktır. Çünkü her bir parçacık robotun harita üzerindeki poz bilgisini ve haritayı içerisinde barındırmaktadır. Bu sebeple parçacık sayısı arttıkça hesaplanması gereken tarama eşleme sayısı da artacağından gMapping’in çalışması gerçek zamanın gerisinde gerçekleşebilmektedir. Parçacık sayısının artması her zaman daha doğru sonuçlar üretileceği anlamına da gelmemektedir. Hatta bütün değerler sabitken aynı “bag” dosyası üzerinde yapılan 2 deneme de gMapping olasılıksal bir yöntem olduğu için farklı sonuçlar üretebilmektedir. Göreceli olarak aynı değişkenlerle ürettiği sonuçlar birbirlerine yakın

olmakla beraber olasılıksal yöntemlerin bir handikabı olarak bu sorun ortada durmaktadır. Şekil 7.17'de tek bir parçacıkla yapılan deney sonuçlarına yer verilmiştir.

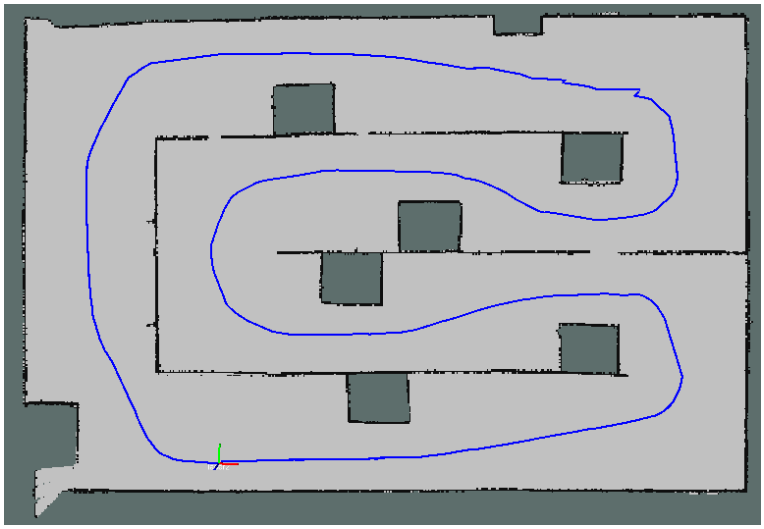


Şekil 7.17 "particles" değerinin 1, diğer gMapping parametrelerinin varsayılan olduğu durumda 10 cm/sn hızla ve 270°'lik lazer açısıyla gezilen bag dosyası üzerinden odometrili gMapping sonuçları

Görülebileceği üzere özellikle ilk koridorun sonuna gelindiğinde farklı olasılıkları tutan parçacıkların olmaması sebebiyle haritada bozulma, mesafede kısalma gerçekleşmiştir. Bu hatanın boyutu Şekil 7.17-d'den Şekil 7.17-e'ye geçişte gözlemlenebilmektedir. Başlangıçta çizilen ortamın sol duvarı, tekrar görüldüğündeki yeniden çizilen duvar arasında fark vardır. Bu fark her ne kadar kapatılmış olsa da Şekil 7.18'de de görülebileceği üzere ilk koridorda bulunan kutu, olması gereken boyutlardan küçük çizilmiştir. Sonuç olarak gMapping'in tek bir parçacıkla da çalışabileceği, fakat farklı olasılıklar tutmadığı için yapılan bir hatanın global olarak güncellenmesinin zor olduğu sonucuna varılmıştır. Şekil 7.19'da ise 5 parçacıkla yapılan sonuç verilmiştir. Özellikle labirentin en üstteki duvarının biraz daha yamuk olarak çizildiği ve gezinge de atlamaların normale göre daha fazla olduğu gözlemlenmiştir. Harita büyük ölçüde doğru sonuçlar üretmiştir. Ancak daha kompleks alanlarda haritanın olasılığını daha fazla parçayla tutmak daha iyi sonuçlar üretilmesini sağlayacaktır. Bu durum, "navigation" paketi incelenirken daha detaylı olarak ele alınacaktır.

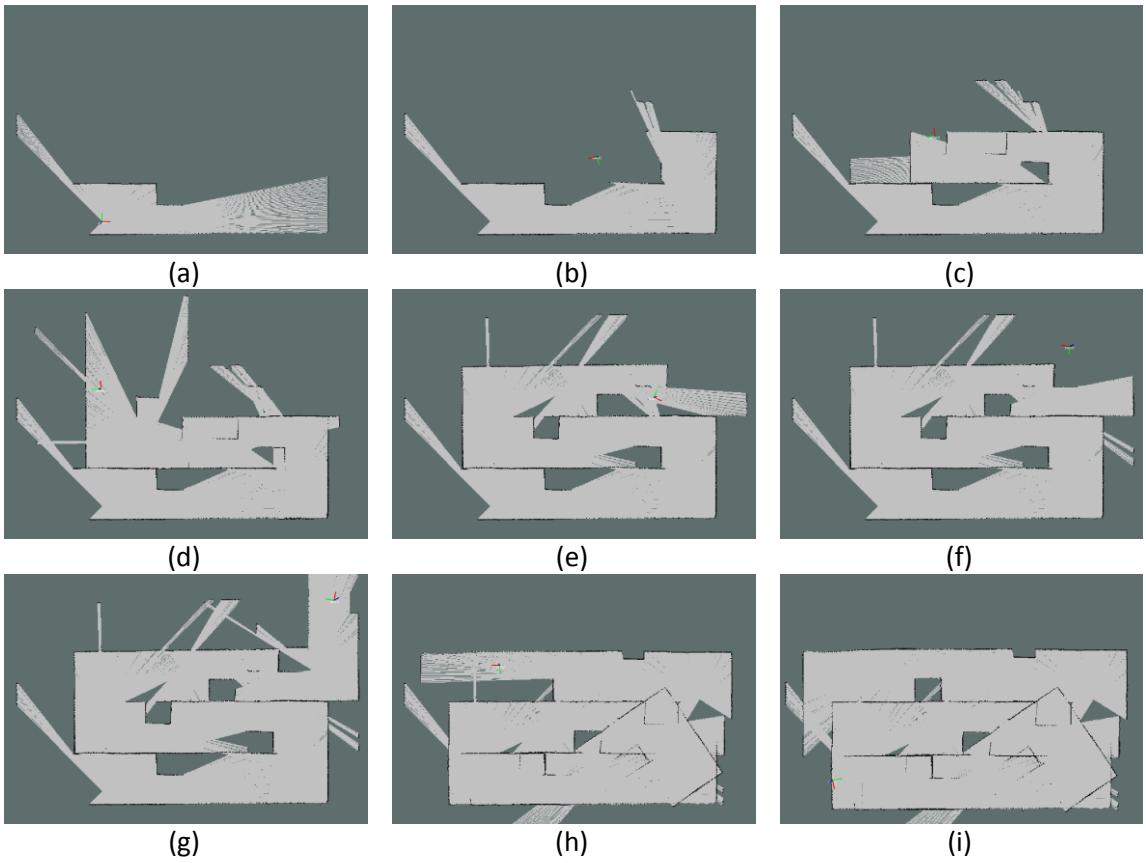


Şekil 7.18 “particles” değerinin 1, diğer gMapping parametrelerinin varsayılan olduğu durumda 10 cm/sn hızla ve 270°’lik lazer açısıyla gezilen bag dosyası üzerinden odometrili gMapping’in ve gezingenin tam bir tur atılması sonucu



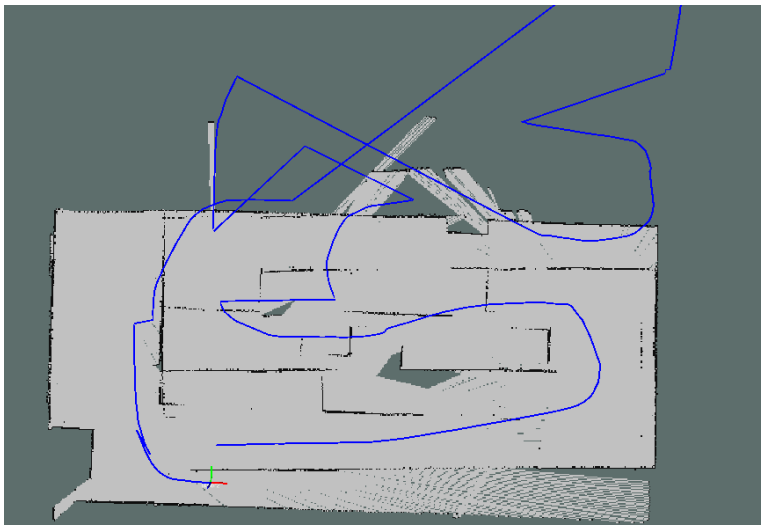
Şekil 7.19 “particles” değerinin 5, diğer gMapping parametrelerinin varsayılan olduğu durumda 10 cm/sn hızla ve 270°’lik lazer açısıyla gezilen bag dosyası üzerinden odometrili gMapping’in ve gezingenin tam bir tur atılması sonucu

Şekil 7.20 ve Şekil 7.21’de gMapping’in 100 parçacıkla çalıştırıldığı durumlar için sonuçlara yer verilmiştir. Haritalamanın üzerinde çalıştırıldığı platformun kapasitesi dolayısıyla gMapping’i 100 parçacıkla gerçek zamanlı çalıştırmak imkânsız olmuştur. Bu durum Şekil 7.20-b ve Şekil 7.20-f’de görülebilmektedir. Robotun LSM tarafından üretilen odometri bilgisi haritalamanın çok önünde gitmektedir. Böyle olduğu için Şekil 7.20-c’de de görülebileceği gibi haritalama yapılırken kayma gerçekleşmiştir. Bu kayma Şekil 7.20-d ve Şekil 7.20-g’de koridorun yan çizilmesi gerekirken dik çizilmesine sebep olmuştur.



Şekil 7.20 “particles” değerinin 100, diğer gMapping parametrelerinin varsayılan olduğu durumda 10 cm/sn hızla ve 270°’lik lazer açısıyla gezilen bag dosyası üzerinden odometrili gMapping sonuçları

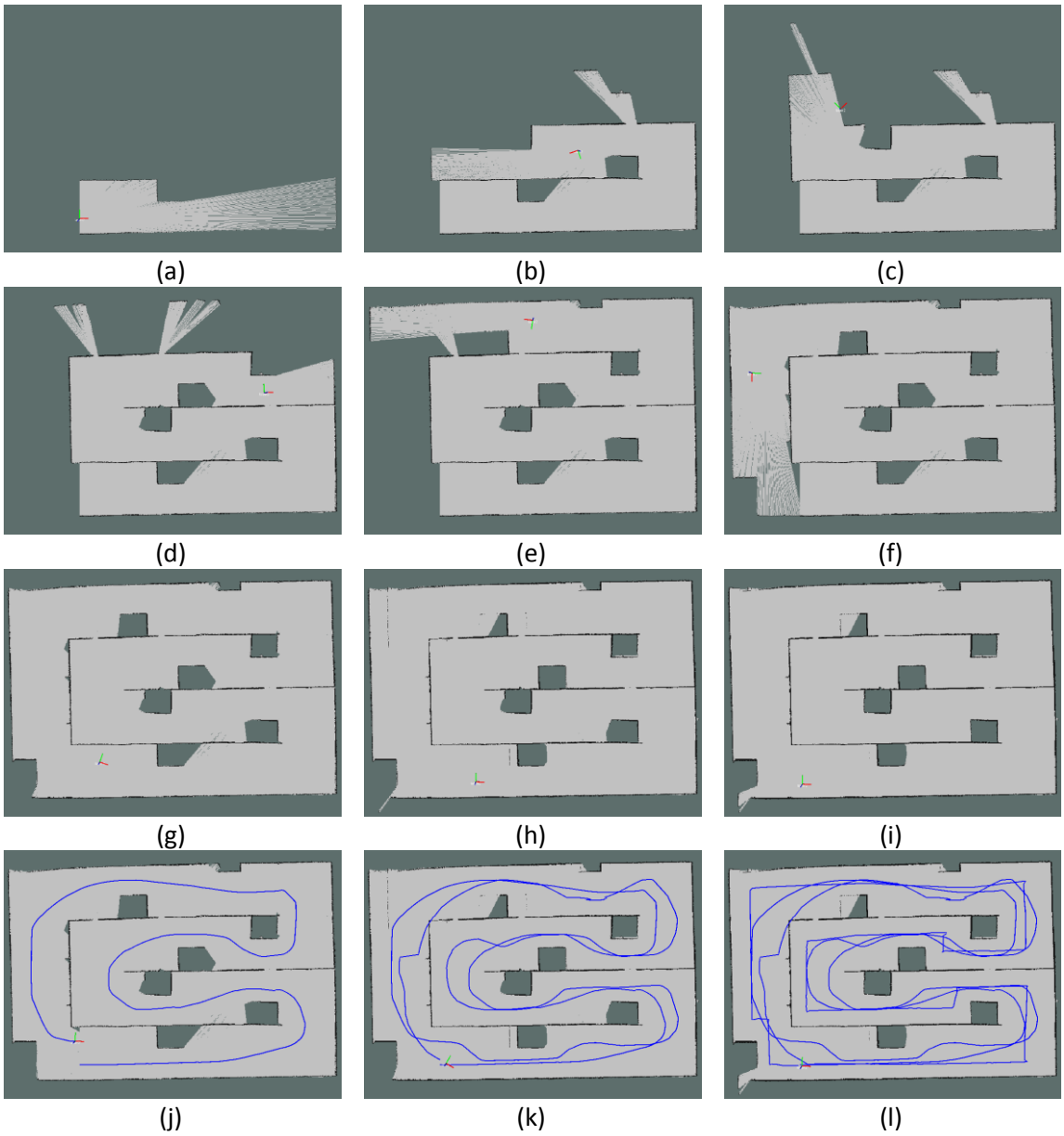
Netice itibariyle Şekil 7.21’de de görülebileceği gibi, gerçek haritadan oldukça uzak bir harita çıkarılmıştır. Bu durum robotun gezingesine de yansımıştır. Yani parçacık sayısının arttırılması platformun kapasitesi oranında iyi sonuçlar üretmekle beraber kapasite aşıldığında da sonuçların bozulmasına sebep olmaktadır. gMapping’in parçacık sayısı belirlenirken sistemin kapasitesine özellikle dikkat edilmelidir. Eğer gerçek zamanlı olarak gMapping çalıştırılmaya ihtiyaç duyulmuyorsa (çevrimdışı çalıştırılacaksa) o zaman parçacık sayısının olabildiğince fazla seçilmesi gMapping’in iyi sonuçlar üretmesini sağlayacaktır. Bu deneyler sonucunda gMapping’in standart parametresi olan 30 parçacığın, tasarlanan robot platformu için ideal olduğu gözlemlenmiştir. Ancak geliştirilen bu robot üzerinde ileride bir görüntü algılayıcısı (kamera vb.) veya bununla beraber farklı duyargaların bulunması ve bu duyargaların verilerinin işlenmesi ile gMapping işlemine ayrılacak kaynak sınırlanacağından bu parametrenin ilgili yükler doğrultusunda deneyler yapılarak optimum’a çekilmesi gerekecektir.



Şekil 7.21 “particles” değerinin 100, diğer gMapping parametrelerinin varsayılan olduğu durumda 10 cm/sn hızla ve 270°’lik lazer açısıyla gezilen bag dosyası üzerinden odometrili gMapping’in ve gezingenin tam bir tur atılması sonucu

7.4.3 18 cm/sn Hızla Yapılan Deney Sonuçları

Daha önce de değinildiği gibi gMapping’in üzerinde çalıştığı platform haritalamayı belirli bir hıza kadar düzgün bir şekilde yapabilmekte ancak belirli bir hızdan sonra bozulmanın gerçekleşmesi kaçınılmaz olmaktadır. Bu bağlamda bir diğer deney olarak robotun 18 cm/sn hızla labirent içerisinde 3 tur gezmesi sağlanmış ve bu gezinmeye dair bag dosyası alınmıştır. Şekil 7.22’de varsayılan gMapping parametreleri kullanılarak elde edilen haritalara adım adım yer verilmiştir. İlk tur sonucunda elde edilen harita Şekil 7.22-g’de, gezinge ise Şekil 7.22-j’de verilmiştir. Görülebileceği üzere haritanın kendisinde büyük bir sapma gerçekleşmemekle beraber özellikle kutuların lazer tarafından görünmeyen bölgelerinde haritalama yapılmamıştır. İkinci tur tamamlandığında ise Şekil 7.22-h’de görülebileceği gibi kutuların arka kısımlarında haritalanmayan alanlar azalmıştır. Yine aynı şekilde dikkat edilirse, en üstteki koridor gezilirken gMapping olasılıkları arasında bir atlama olmuş ve haritanın sol duvarı olması gerektiği yerden önce çizilmiştir. İlgili koridordaki kutunun sınırları da değişmiş, neredeyse yarı yarıya yok olmuştur. Üçüncü tur atıldığında üst koridorda fazla çizilen duvar silinmiş ve koridordaki sınırları bozulan kutunun sınırları tekrar çizilmiştir. Görülebileceği üzere hızın artmasıyla beraber hata olma olasılığı artmıştır. Aynı ortamda birkaç tur dönülmesiyle beraber hataların global olarak düzeltilmesi sağlansa da bu hız aşıldığında gMapping’in sonuçlarının daha da kötüleşeceği tespit edilmiştir.

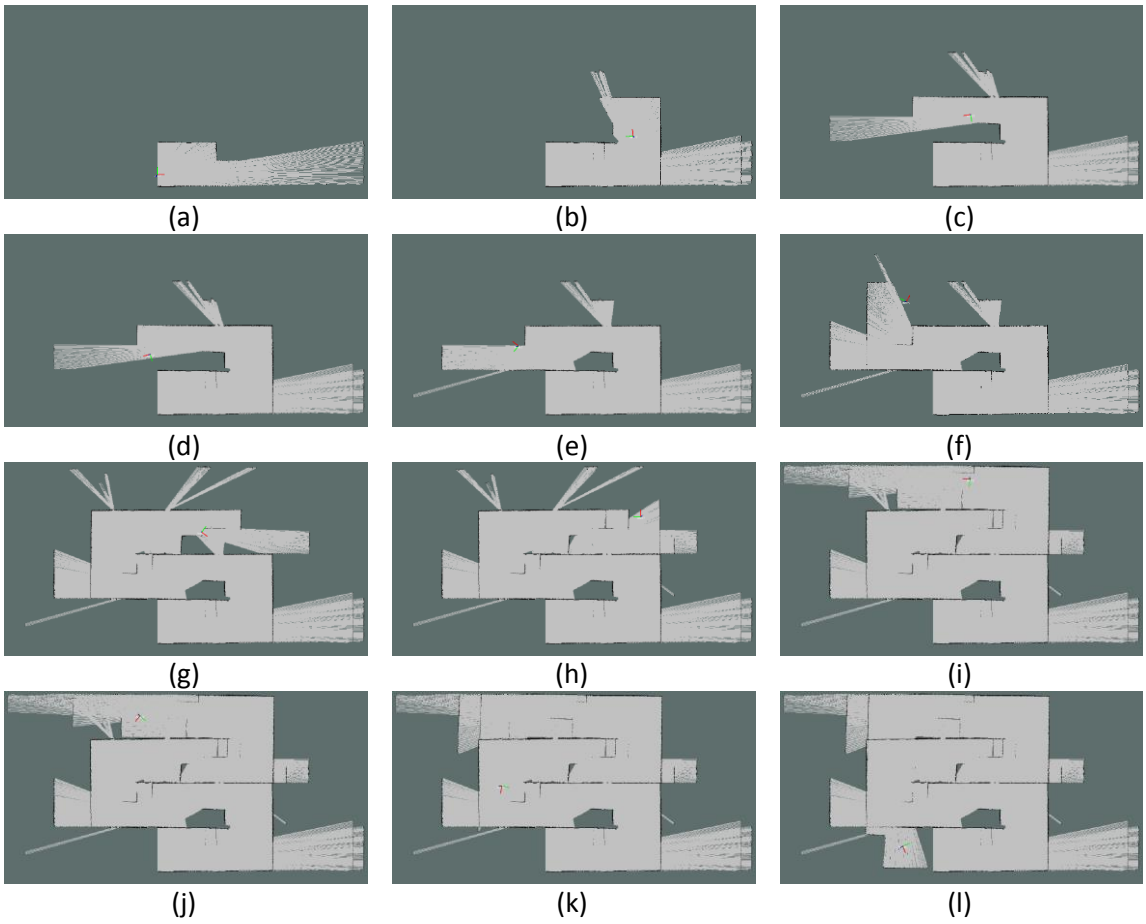


Şekil 7.22 gMapping parametrelerinin varsayılan olduğu durumda 18 cm/sn hızla 3 tur gezilen bag dosyası üzerinden odometrili gMapping sonuçları ve gezingeleri

7.4.4 41 cm/sn Hızla Yapılan Deney Sonuçları

gMapping parametreleri arasında haritanın ne kadar zamanda bir güncelleneceğini belirleyen bir değişken de mevcuttur. Bu parametre “map_update_interval” değişkeni ile gMapping’e bildirilmekte ve varsayılan olarak 5 değerini almaktadır. Yani gMapping haritası her 5 saniyede bir güncellenmektedir. Hiç kuşkusuz hızın arttırılmasıyla beraber haritalamanın bozuk çıkması pek olasıdır. Bu deney kapsamında 41 cm/sn hızla labirent içerisinde 1 tur atılmıştır. Haritalamanın 5 saniyede bir güncellendiği durumda haritanın daha başlangıçta bozulması sebebiyle 3 saniyede ve 2 saniyede yapılan farklı

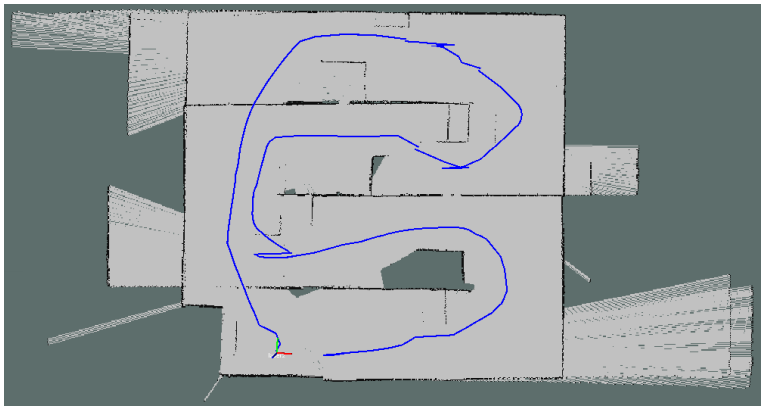
güncelleştirmelerle iki farklı harita çıkartılmıştır. Şekil 7.23'de haritanın 2 saniyede bir güncellendiği durum için gMapping sonuçlarına yer verilmiştir. Şekil 7.23-b'de görülebileceği gibi, daha ilk koridor dönüldüğünde haritalama bozulmuş, çıkarılan haritanın uzunluğu azalmıştır. Adım adım incelendiğinde her koridorun sonunda bu azalma gözlemlenebilmektedir. Şekil 7.23-l'ye gelindiğinde ise haritanın temel çizgilerinin var olduğu, ancak ölçeklemesinin ve bundan kaynaklanan kutuların yerleşiminin ve boyutlarının hatalı olarak gösterildiği görülmüştür.



Şekil 7.23 "map_update_interval" değerinin 2, diğer gMapping parametrelerinin varsayılan olduğu durumda 41 cm/sn hızla gezilen bag dosyası üzerinden odometrili gMapping sonuçları

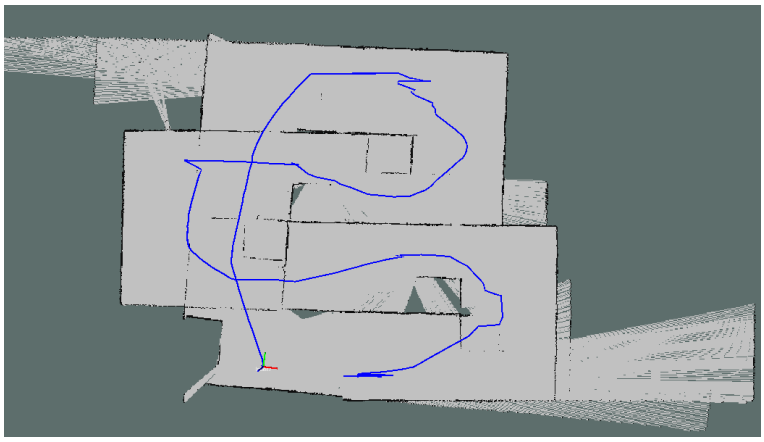
Şekil 7.24'de ilgili deneyin sonucunda elde edilen gezingeye yer verilmiştir. Odometri atlamalarının oldukça fazla olduğu bu gezingede, gezinim döngüsünün de tamamlanamadığı gözlemlenmiştir. Odometrinin bozuk üretildiği bir durum için haritalamanın doğru sonuçlar üretmesi neredeyse imkânsızdır. Bu hızda gezildiğinde

de odometri bilgisinin, tarama eşleme arasındaki mesafe fazla olduğu için, doğru çıkartılamaması sonucu haritalamanın bozuk çıkması pek olasıdır.



Şekil 7.24 “map_update_interval” değerinin 2, diğer gMapping parametrelerinin varsayılan olduğu durumda 41 cm/sn hızla gezilen bag dosyası üzerinden odometrili gMapping sonuçları ve gezingesi

Haritanın 3 saniye de bir güncellendiği durum için de deney yapılmıştır. Bu deneyin sonucu Şekil 7.25’de verilmiştir. İlgili şekilde görülebileceği üzere atlamalar ve hatalar 2 saniye de bir güncellemeye göre daha fazladır. Bu fazlalık çıkarılan gezingeye bakarak daha iyi gözlemlenebilir. Özellikle gezinim çevriminin kapatılmasına Şekil 7.24’e göre daha uzaktır. Yani haritalamanın düzgün çıkmasını sağlayan bir diğer etken de haritanın ne kadar sürede bir güncelleneceğinin belirtilmesidir.



Şekil 7.25 “map_update_interval” değerinin 3, diğer gMapping parametrelerinin varsayılan olduğu durumda 41 cm/sn hızla gezilen bag dosyası üzerinden odometrili gMapping sonuçları

Netice itibariyle gMapping’in parametrik bir yöntem olduğu, standart bir parametre ayarının olmadığı, farklı ortamlar, farklı platformlar ve farklı sonuçlar için

parametrelerin uygun girilmesi gerektiği gözlemlenmiştir. Bu deneyler kapsamında gMapping'in haritalamaya etkisi fazla olan parametreleri üzerinde değişiklikler yapılarak sonuçlar alınmıştır. Bütün parametrelerin etkisi incelenememiştir. Bazı parametreler kullanılan donanımlarla beraber standart olarak ayarlanmaktadır. Bu sebeple sadece haritalamaya etkisi olabilme ihtimali olan parametreler için deneyler yapılmıştır. Bu deneyler sonucunda her bir parametrenin çalışılan ortam ve platform koşullarına göre denenerek dikkatli bir şekilde seçilmesi gerektiğine ve haritalamanın buna göre çalışacağı neticesine ulaşılmıştır.

BÖLÜM 8

GEZİNGE VE KONUM KESTİRİM YÖNTEMLERİNDE MESAFE VE ATALET DUYARGALARININ KULLANIMI

Mobil robotlarda gezinge çıkarımı, robotun harita üzerinde izlediği yolun belirlenmesi ve bu yol üzerinden robotun hedefine olan mesafesinin tespiti, robotun engellere takılmadan güvenli bir şekilde yoluna devam edebilmesi ve robotun eğitilebilmesi açısından oldukça önemlidir. Bu çalışmada [59], belirlenen bir alan içerisinde mobil bir robot ile tekerlek odometrisi kullanılmadan gezinge çıkarımı amaçlanmıştır. Bunun için temel olarak 2 farklı yöntem kullanılmıştır. İlki, en çok kullanılan eş zamanlı konum belirleme ve haritalama (SLAM) algoritmalarından olan sadece lazer mesafe ölçüm duyargası bilgisini kullanan gMapping ile gezinge çıkarımı, ikincisi ise lazer mesafe duyargası tabanlı LSM ile gezinge çıkarımıdır. LSM ile gezinge çıkarımında hız bilgisi ve atalet duyargası da kullanılmıştır. Tüm yöntemler, alt başlıklar halinde incelenmiştir.

Mobil robotlarda gezinge çıkarımında farklı yöntemler kullanılmaktadır. Bunlardan en temeli tekerlek odometrisi kullanılarak üretilmiş çözümlerdir [1]. Bu çözümde, mobil robot üzerinde bulunan tekerleklerin dönme miktarına bakılarak kinematik denklemler aracılığıyla gezinge çıkarılmaktadır. Burada tekerleğin döndüğü kadar robotun gittiği varsayılmaktadır. Oysaki yer çekiminden veya yerin sürtünme kuvveti gibi sebeplerden kaynaklanan ötelenmeler ve yerinde saymalar robotun tekerleklerinin döndüğünden farklı miktarda hareket etmesine sebep olacaktır. İşlem karmaşıklığı az olmasına rağmen ortam bağımlı ve düşük performanslı bir yöntemdir.

Bir diğer yöntem, lazer mesafe duyargası kullanılarak yapılan, tarama eşlemeye dayalı LSM [2], [3] yöntemidir. Bu yöntemde daha önceden de bahsedildiği gibi, bir lazer

mesafe duyargası ile ardışık zamanlarda alınan ortama ait ölçüm bilgileri kıyaslanarak robotun ilerlemesinin (translation) ve dönmesinin (rotation) ne kadar olduğu hesaplanmaktadır. Sadece son iki ölçümün değerlendirilmesi ve lazer mesafe duyargasının ölçüm mesafesinin uzunluğu, özellikle birbirine çok benzeyen ortamlarda (düz bir koridor vb.) alınan taramaların birbirine benzemesi sebebiyle gezinge çıkarımı yapılamamaktadır. Ancak LSM ile birlikte robotun hız bilgisi ve/veya atalet duyargasından alınan bilgi gibi çeşitli ek bilgiler kullanılarak bu sorun en aza indirilebilmektedir.

Literatürde yaygın olarak kullanılan yöntemlerden bir diğeri de SLAM algoritmaları ile gezinge çıkarımıdır [6], [7], [8], [9]. Bu yöntemlerde LSM'de olduğu gibi tarama eşleme ile konum belirleme yapılmaktadır. Önemli noktalardan birisi konum belirlemenin sadece ardışık zamanlarda alınan tarama verileriyle olmaması aynı zamanda önceden alınan verilere göre hesaplanan konumların bilgisinin gezinge çıkarımında kullanılmasıdır.

Özellikle son araştırmalarda görsel odometrinin gezinge çıkarımında kullanıldığı görülmektedir [10], [11]. Bu çalışmalarda bir kamera yardımı ile farklı zaman aralıklarında ortamın görüntüsü alınmakta ve bu görüntüler birbirleri ile eşlenmeye çalışılarak robotun ilerleme ve dönme miktarı hesaplanabilmektedir. Derinlik bilgisi okuyabilen kameralar (Kinect vb.) ile lazer mesafe duyargasının işlevine ek olarak görüntü bilgisi de kullanıldığı için daha doğru sonuçlar üretilmektedir.

Bu çalışma kapsamında RoboCup olimpiyatlarında robotların yarıştığı ortamlarda kullanılması zor olan tekerlek odometrisi ve görsel odometri üzerinde durulmamış bunun yerine LSM ve SLAM tabanlı gezinge çıkarım yöntemleri üzerinde çalışılmıştır. Deney kapsamında daha önceden Bölüm 5'te verilen Şekil 5.2'deki ortam kullanılmış ve sonuçlar bu ortam üzerinden alınmıştır.

8.1 **Deneyde Kullanılan Gezinge Çıkarım Yöntemleri**

Robotun gezinmesi esnasında 7 farklı gezinge çıkartılmaktadır. İlki yer gerçeği gezingesidir. Bu gezinge, robota yerleştirilen bir işaretleyici yardımı ile çalışma alanı içerisinde robotun gezdiği gerçek konumun yere çizilmesi, ardından diğer gezingelerle

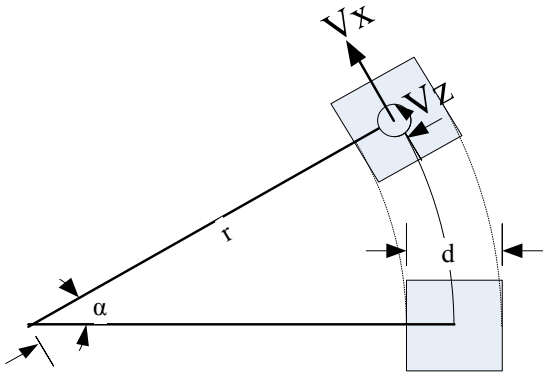
kıyaslayabilmek için çizimin sayısallaştırılmasından oluşmaktadır. Bunun dışında 6 farklı gezinge çıkarım yöntemi daha uygulanmıştır. Bunlara ait yapılandırmalar ve gezinge yöntemlerini temsil eden renkler Çizelge 8.1'de verilmiştir.

Çizelge 8.1 Gezinge çıkarım yöntemlerine dair yapılandırma tablosu

Gezinge Çıkarım Yöntemi	**Kullanılan Duyargalar ve Bilgiler**	**Gezinge Rengi**
Kinematik	Hız	
gMapping	Lazer	
LSM	Lazer	
LSM-K	Lazer + Hız	
LSM-A	Lazer + Atalet	
LSM-AK	Lazer + Hız + Atalet	
Yer Gerçeği	-	

8.2 **Kinematik Tabanlı Gezinge Çıkarımı**

Kinematik tabanlı gezinge çıkarımında robotun hız bilgisinden yararlanılmıştır. Açık çevrim kontrollü olarak anlık hız bilgisinin zamanla çarpımından gidilen yol bilgisi elde edilmektedir. Anlık hız bilgileri deneysel sonuçlarla elde edilmiştir. Ayrık zamanlar arasında olan değerler için interpolasyon işlemi yapılmaktadır. Şekil 8.1'de robotun hareketine ait kinematik model resmedilmiştir.



Şekil 8.1 Kinematik model

Robota ilişkin ilk poz durumundan (x_0, y_0, θ_0), Δt süre sonraki oluşacak poz durumuna (x_1, y_1, θ_1) geçiş (8.1) ile verilen şekilde hesaplanabilir.

$$\begin{aligned} x_1 &= x_0 + V_x \cos(\theta_0)\, \Delta t \\ y_1 &= y_0 + V_x \sin(\theta_0)\, \Delta t \\ \theta_1 &= \theta_0 + V_z \Delta t \end{aligned} \tag{8.1}$$

8.3 **LSM Tabanlı Gezinge Çıkarımı**

Bölüm 5'de LSM tabanlı gezinge çıkarımının nasıl gerçekleştiğine ve deneysel sonuçlara yer verilmişti. Burada ise LSM ile birlikte LSM'nin parametre olarak aldığı kinematik ve atalet duyargası bilgisinin LSM'ye eklenmesi sonucunda ortaya çıkan gezingelerin yalnız LSM kullanıldığı durumdaki gezinge ile karşılaştırılmasına yer verilmiştir.

LSM yönteminde kinematik bilgisinin dâhil edilmesinde (LSM-K), ilk dönüşüm tahmini olan q_0 için öteleme parametresi (t) ilk durmunun kinemetik model ile belirlenmesi yaklaşımı kullanılır. Atalet duyargası bilgisinin dâhil edilmesinde (LSM-A), ilk dönüşüm tahmini olan q_0 için rotasyon parametresi ($\mathrm{R}(\theta)$) ilk durumunun atalet duyargası ile belirlenmesi yaklaşımı kullanılır. Atalet duyargası ve kinematik bilgisinin birlikte dâhil edilmesinde (LSM-AK), ilk dönüşüm tahmini olan q_0 için öteleme parametresi (t) ve rotasyon tahmini ($\mathrm{R}(\theta)$) ilk durumunun kinematik model ve atalet duyargası ile birlikte belirlenmesi yaklaşımı kullanılır.

8.4 **gMapping Tabanlı Gezinge Çıkarımı**

Bölüm 7'de gMapping tabanlı haritalamanın ve bu haritalamaya bağlı olarak gezingelerin nasıl çıkarılacağına ve deneysel sonuçlarına değinilmişti. gMapping'in deneysel sonuçları anlatılırken çıkarttığı gezingenin çevrimi tamamlayıp tamamlayamadığına dair karşılaştırma yapılmıştı. Bu bölümde gMapping'in çıkarttığı gezingenin yer gerçeği gezingesine göre hata oranları verilmiştir.

8.5 **Deneyin Yapılış Şekli ve Neticeleri**

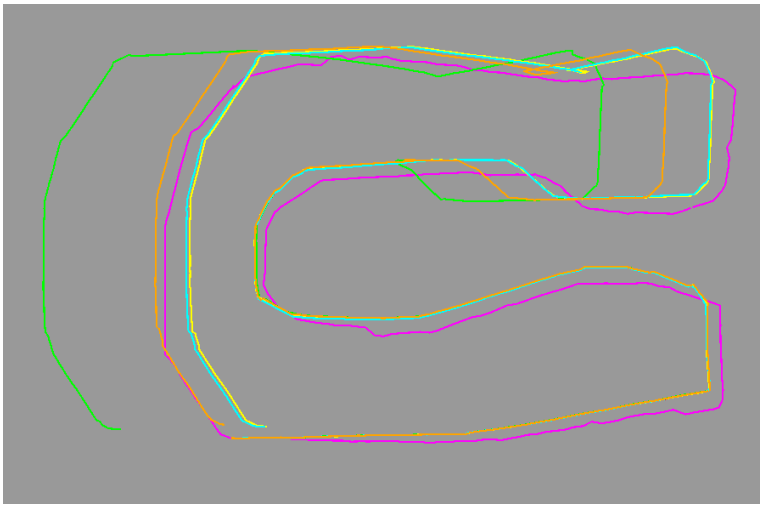
Deney sonuçları, robotun alanda 10 cm/sn hızla bir tur gezdiği durum için elde edilmiştir. Tüm yöntemler için gezingeler eşzamanlı olarak hesaplanmıştır. Yer gerçeği

gezingesi ise gezinim bittikten sonra sayısallaştırılmıştır. Gezingeler hesaplanırken kullanılan bilgilerin çalışma frekansları aynıdır. Şekil 8.2'de yer gerçeğine karşılık, hız bilgisinin kullanıldığı durumda elde edilen kinematik gezinge verilmiştir. Sadece hız bilgisinin kullanıldığı durumda yerinde saymalar ve takılmalardan dolayı bulunan gezingenin yer gerçeğinden oldukça saptığı görülmektedir. Sapmalar genellikle dönme noktalarında meydana gelmektedir. Şöyle ki, hem yer gerçeği gezingesi hem de kinematik modele ait gezinge bir ip gibi düşünülüp uçlarından tutarak düz bir doğru oluşturacak şekilde gerilirse, her iki gezingenin boylarının birbirinden çok farklı olmadığı görülecektir. Ancak dönmeler esnasında tekerleklerin gürültülü dönmesi sonucunda kinematik model robotun dönüşünü tam olarak örnekleyemediğinden robot gerçek dönme açısından daha farklı derecede dönmüştür. Bu sebeple kinematik modelin çıkarttığı gezinge yer gerçeği gezingesinden oldukça farklı görünmektedir. Eğer bu gezingeler kullanılarak gMapping gibi bir yönteme odometri üretilmek istenirse, bu yöntemin kullanılması, ürettiği odometri verisinin çok kötü olmasından dolayı, gMapping'in çıkarttığı haritaların oldukça bozuk olmasına sebep olacaktır. Bu yüzden, RoboCup olimpiyatlarında ve yapılan haritalama çalışmalarında kinematik modelden yararlanılarak çıkarılan gezingenin kullanılmamasına karar verilmiştir.

Şekil 8.2 Kinematik tabanlı gezinge ve yer gerçeği

Şekil 8.3 ile LSM tabanlı gezinge çıkarımları verilmektedir. LSM tabanlı gezinge çıkarımı yöntemlerinin, artımlı ve yerel tahminler yaptıkları için, çevrimi tamamlayamadıkları

gözlenmektedir. LSM tabanlı yöntemlerin dönüşlerde ve lazer ölçümünün değişmediği düz koridorlarda dahi birbirlerinden farklılaştıkları gözlenmektedir.



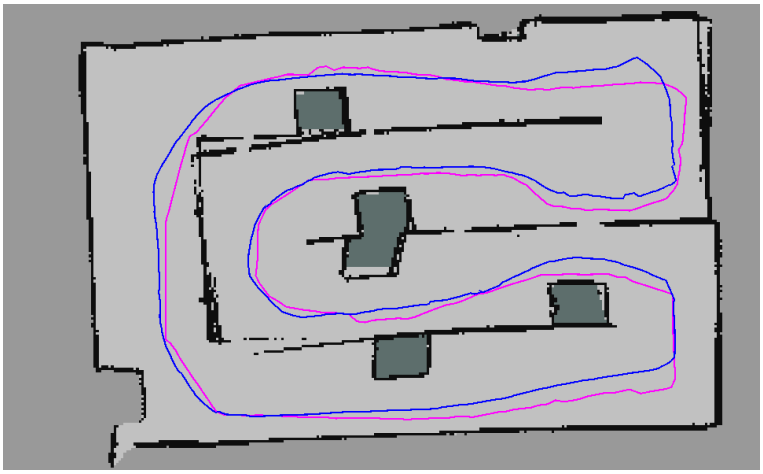
Şekil 8.3 LSM tabanlı gezingeler ve yer gerçeği

Çizelge 8.1'deki yapılandırmalardan da görülebileceği gibi en fazla hataya sahip yeşil gezinge sadece LSM kullanılması sonucu ortaya çıkmıştır. 3. koridorun ortasında ve sonunda dönerken hatalı dönmüş ve bu hatayı bir daha düzeltemeyerek gezinimin sonuna kadar aynı hatayı artımsal olarak devam ettirmiş ve sonucunda gezinim çevrimini tamamlamaktan oldukça uzak bir noktada kalmıştır.

Turuncu renk ile gösterilen LSM+A gezingesinde atalet duyargasından elde edilen bilgiler kullanılmıştır. Bu gezinge çıkarımında da sadece LSM gezingesinin çıkarımı sırasında olan 3. koridordaki hatalar meydana gelmiş ancak nispeten daha az hatayla dönüşler gerçekleştirilmiştir. Bunun temel sebebi bahsedildiği gibi LSM'nin tarama eşleme yapması sırasında ilk iterasyon için atalet duyargasından elde edilen dönme bilgisini kullanmasıdır. Bu sayede, yalnız LSM kullanılan yöntemden daha az hatalı bir dönüş gerçekleştirmiştir. Hatanın büyük olmamasından dolayı 4. koridoru (en üstteki koridor) gezerken bu hatayı telafi edebilmiş ve gezinimin sonuna varıldığında diğer LSM'li yöntemlere göre çevrimi tamamlamaya en yakın yöntem olmuştur.

Sarı (LSM-K) ve cyan (LSM-AK) renkleri ile çizilmiş yöntemlere gelindiğinde ise yer gerçeğine göre daha benzer oranda gezinge çıkarttıkları görülmüştür. Ancak birazdan her ikisinin de fark alan oranları incelendiğinde LSM-AK'nın daha başarılı olduğu görülecektir.

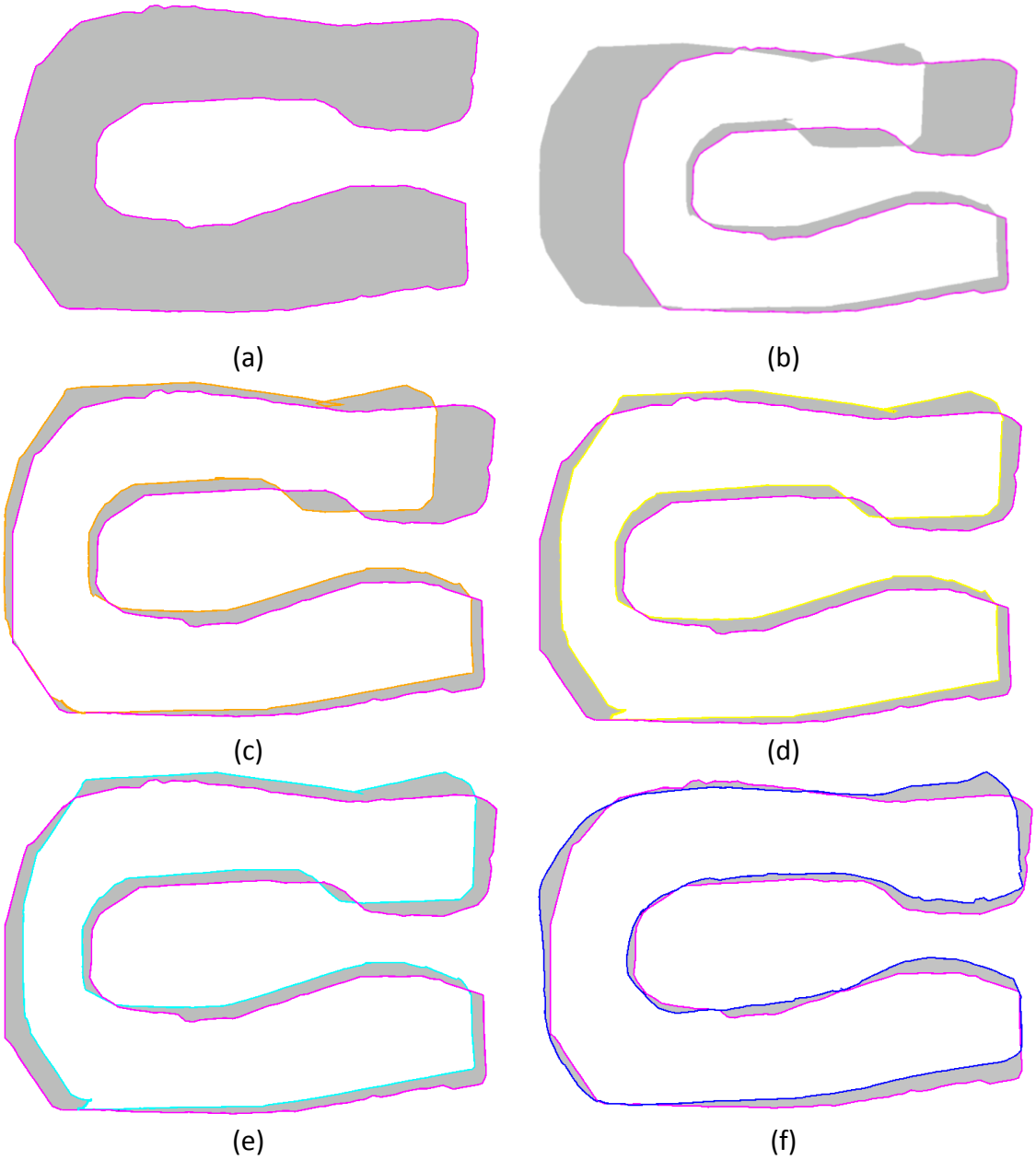
Şekil 8.4'te gMapping tabanlı gezinge çıkarımı yer gerçeği ile birlikte verilmektedir. gMapping uygulamasında parçacık adedi 10 olarak kullanılmıştır. gMapping ile oluşturulan gezinge, harita bağımlı olarak elde edilmekle birlikte, çevrimin başarılı bir şekilde tamamlanabildiği görülmektedir.



Şekil 8.4 gMapping ve yer gerçeği gezingeleri

Gezinge incelendiğinde, zaman zaman lazer taramalarının geç eşleştirilmesinden kaynaklanan hataların olduğu ve bu geç eşleştirmenin bazen yer gerçeği gezingesinde olan kıvrımlı yerleri göremediği gözlemlenmiştir.

Görsel değerlendirmeler yanında, elde edilen gezingeler için, yer gerçeğine benzerlik ölçütü olarak kapsanan alan farkı oranı verilmiştir. Bu değerlendirme ile uygulanan yöntemin gezinim çevrimini tamamlasa bile, gezinim esnasında içerdiği hatalar görülebilecektir. Diğer taraftan farklı yöntemlerin çıkarttığı gezingeler birbirleri ile daha kolay kıyaslanabilecektir. Çevrimi tamamlayamayan gezingeler uzatılarak çevrimi tamamlaması sağlanmıştır. Böylece her bir gezingenin yer gerçeği gezingesi ile arasında olan alan farkı bulunabilmesi amaçlanmıştır. Gezinge kapalı alanlarının yer gerçeğinden farkının yer gerçeği alanına bölünmesi ile bu ölçüt hesaplanmıştır. Bu ölçüt hata oranı olarak değil fark alan oranı olarak oluşturulmuştur. Yani % olarak üretilen sonuçlar hatanın ne kadar olduğunu göstermemekte; sadece gezingelerin yer gerçeğine göreceli olarak fark alan oranını göstermektedir. İdeal bir durum için, gerçek eğri ile hesaplanan eğri birbirine yaklaştıkça, bu oranın sıfıra yaklaşması beklenmektedir. Şekil 8.5 ile söz konusu alan farkları görsel olarak verilmiştir.



Şekil 8.5 Alan farkları ölçüsü: (a) Yer Gerçeği, (b) LSM, (c) LSM-A, (d) LSM-K, (e) LSM-AK, (f) gMapping

Çizelge 8.2 ile fark olan alan ile yer gerçeği alan oranları verilmektedir. Tablodaki sonuçlar incelendiğinde, en başarılı gezinge çıkarımının %19,88 fark oranı ile gMapping yöntemi ile elde edildiği görülmektedir. LSM tabanlı gezinge çıkarımlarında ise beklenildiği gibi kullanılan bilgiler arttıkça daha iyi sonuçlar elde edildiği gözlemlenmiştir. Bu deney kapsamında tekerlek odometrisi kullanmadan, lazer mesafe ve atalet duyargaları kullanarak oluşturulan gezingeler, SLAM tabanlı gMapping gezingesi ile karşılaştırmalı olarak incelenmiştir.

Çizelge 8.2 Gezingeler için alan fark oranları

Gezinge Çıkarım Yöntemi	**Fark Oranı (% Olarak)**
gMapping	19,88
LSM-AK	31,98
LSM-K	33,17
LSM-A	34,40
LSM	74,91

Artımsal yerel uygulamalar olan LSM tabanlı gezingelerde çevrimin tamamlanmadığı, buna rağmen gMapping yönteminin, ortam haritası ile gezinge oluşturduğu için çevrimi kapatabildiği ve fark alan ölçüsü olarak en iyi sonucu verdiği görülmüştür. Kullanılan yöntemlerin yanı sıra görsel odometri dâhil edilerek duyarga birleştirme ile gezinge tahmininin daha başarılı bir şekilde elde edilmesi beklenmektedir.

BÖLÜM 9

MOBİL ROBOTLARDA LOKALİZASYON

Mobil robotlarda, robotun pozisyonunun tahmin edilmesi temel problemlerden birisidir. Robotun pozisyonunun bilinmesi, ortamın o an için var olan veya belirli bir süre öncesi için var olan haritasını kullanabilmek ve bunu ofis içi teslimatlar gibi yararlı bir hale getirebilmek için gereklidir. Mobil robotlarda, robotun pozisyonunun tahmini için kullanılan yöntemler kabaca robotun gezingesini tutan yöntemler ve robotun global pozisyonunu tutan yöntemler olmak üzere 2 başlık altında incelenebilir [60]. Yapılan birçok araştırmada robotun gezingesinin üzerinden konum ve duruş kestirimi problemi çözülmeye çalışılmıştır. Bu metotta robotun başlangıç konumunun bilindiği varsayılmaktadır. Bunun üzerinden artımsal odometri hatalarının giderilmesi için çözüm önerileri getirilmektedir. Bu yöntemde, kullanılan birkaç tane güçlü ve hızlı metotlar olsa da; 2 önemli zayıf nokta vardır:

- Gezinge izleme teknikleri, robotun global pozisyonunu tahmin edebilmekten oldukça uzaktırlar ki; bu durum otonom robotlarda oldukça önemlidir. Bunun yerine, bu yöntemlerde robotun başlangıç konumu önceden bilinmektedir.
- Ayrıca gezinge izleme metotları konum kestiriminden kaynaklanan kalıtsal hataları tespit etme ve gidermeye elverişli değildir. Bir operatör tarafından takip edilmeyi ve hata ile karşılaşıldığında durdurulmayı gerektirmektedirler.

Bir diğer yöntemde ise robotun global pozisyonu üzerinden lokalizasyonun gerçekleştirilmeye çalışıldığından bahsedilmişti. Bu yöntem için kullanılan metotlarda, robotun başlangıç konumunun bilinmesine gerek olmadan robotun konumunun tespiti

gerçekleştirilmektedir. Bu yeterlilik gerçekten otonom çalışan bir mobil robot için temel bir önkoşuldur.

Gezinge izleme teknikleri ve global lokalizasyon metotları yürütüldükleri durum uzayları tarafından ayırt edilebilir. Gezinge izleme teknikleri, robotun tahmini pozisyonunun çevresini merkez alarak ufak bir durum uzayında güncellemeler yapmaktadır. Global pozisyon tahmini metotları ise tam bir yapılandırma uzayı ile baş etmektedirler [61]. Gezinge izleme yöntemleri arama uzaylarının küçüklüğünden dolayı global yöntemlere göre daha uygun görünse de konum kestirimi hatalarının telafisini gerçekleştirmeden yoksun oldukları için son araştırmalarda daha çok global yöntemlere ağırlık verilmiştir. Bu çalışma kapsamında da global yöntemlerden olan Monte Carlo lokalizasyonunun (MCL) iyileştirilmiş bir versiyonu olan Augmented Monte Carlo lokalizasyonu (AMCL) üzerinde durulmuştur.

9.1 **Monte Carlo Lokalizasyonu (MCL)**

MCL, mobil robotlarda global konum kestirimi ve kaçırılmış robot problemini güçlü ve verimli bir şekilde çözen olasılıksal lokalizasyon algoritmasıdır. Rastgele gürültü dağılımlarının (ve robotun hareketlerindeki ve algılarındaki nonlineerliğin) yerini tutabilen bir yöntemdir. Bu yüzden MCL, duyarga verilerinden özellik çıkarımına duyulan ihtiyacı ortadan kaldırır. Genellikle sadece lazer mesafe ölçüm duyargası kullanılarak çalıştırılır. Algoritmaya ismini veren şehirden (Monte Carlo: kumarhaneleriyle ünlü bir şehir) de anlaşılabileceği gibi rastgele olasılıksal dağılımların, duyargalardan okunan değerlerle ağırlıklandırılması prensibiyle çalışmaktadır. Yani başlangıçta robotun pozunu temsil eden belirli sayıdaki parçacık haritanın üzerine önerilen bir dağılımla dağıtılır, ardından her bir parçacığa duyargalardan okunan değerlerle önem katsayısı verilir. Bu önem katsayısı, parçacığın robotun pozisyonunu temsile ne kadar yakın olduğunu tutmaktadır.

MCL'in ana fikri örnekler kümesinin (parçacıkların) inançlarını temsil etmek ve robot pozları üzerinden bir sonraki dağılımı buna göre çizmektir. MCL, istenilen dağılıma yakınsayan bir rastgele ağırlıklı parçacıklar koleksiyonu tarafından posterior'ları temsil eder. Bu yöntemler literatürde genellikle parçacık filtresi olarak anılır [18]. MCL algoritmasının sözde kodu; X parçacıkların şu anki inanç durumu, a verilen komut

kümesini, o okunan duyarga verilerini, X' parçacıkların bir sonraki adımdaki inanç durumlarını temsil etmek üzere aşağıdaki gibi verilebilir. Bu çalışma kapsamına MCL'nin ROS Hydro sürümü üzerinde gerçeklenmiş AMCL versiyonu kullanılacaktır.

Algoritma: MCL

Girdi: X, a, o

$X' = 0$
$for\ i = 0\ to\ m\ do$
  $w_1, \dots, w_m\ göre\ X'ten\ rastgele\ x'ler$ üret
  $rastgele\ x' \sim p(x'|a, x)$ üret
  $w' = p(o|x')$
  $ekle\langle x', w' \rangle\ to\ x'$
$endfor$
önem $faktörleri\ w'leri\ X'e\ normalize\ et$
$return\ X'$

Çıktı: X'

9.2 **Augmented Monte Carlo Lokalizasyonu (AMCL)**

AMCL ile adaptasyon geçiren MCL algoritması konum kestirimi hatalarının daha iyi giderilebilmesi için rastgele parçacık seçimi kullanmaktadır. Özellikle robotun kaçırılması problemlerinin üstesinden gelebilmek için bu adaptasyon gerçekleştirilmiştir. Lokalizasyon çalışırken, belirli bir anda sadece belirli bir noktanın yakınlarında bütün parçacıklar toplanabilir ve eğer bu tahmini poz yanlış ise algoritmanın lokalize olması imkansız hale gelmektedir. Bir başka deyişle şu şekilde açıklamak mümkündür. Robotun pozisyonu doğru olarak tahmin edildiğinde bütün parçacıklar aynı yere toplanmaktadır, robot bu anda yerinden kaldırılıp başka bir yere konulduğunda (robotun kaçırılması problemi) robotun yeni yerini temsil edebilecek, düşük olasılıkta olsa da, bir parçacık olmadığı için algoritma lokalize olamayacaktır. Bu önemli bir problemdir. Çünkü özellikle bu çalışma kapsamında da üzerinde durulduğu gibi arama kurtarma robotlarında sistemin çalışması esnasında sıkça böyle problemler ortaya çıkabilmektedir.

Bu yöntemde ise böyle bir durumla karşılaşılabileceği düşünülerek, parçacıkların yeniden örneklendirilmesi sırasında rastgele parçacıklar da parçacık kümesine eklenmektedir. Robotun kaçırılmadığı durumlar için bu yöntem aynı zamanda MCL'nin güçlülük düzeyini arttırmaktadır. Ancak rastgele parçacık eklemek de şu 2 soruyu akla getirmektedir.

- Algoritmanın her bir iterasyonunda ne kadar rastgele parçacık eklenmelidir?
- Bu rastgele parçacıklar hangi dağılım modelini kullanarak üretilmelidir?

İlk soru için, belirli sayıda parçacığın her iterasyon sonucunda eklenmesinin mümkün olabileceği gibi, konu kestirimi performansına göre belirlenen sayıda yeni rastgele parçacık eklenmesi daha mantıklı bir yaklaşım olacaktır. Bu parçacık filtrelerinde kullanılan önem faktörü gibi bir yaklaşımla basitçe çözülebilecek bir sorundur. AMCL'de ölçüm olasılıklarının kısa dönem (short-term) ortalamalarını sürdürmek ve yeni rastgele parçacıklar üretileceği zaman bunları uzun dönem (long-term) ortalamalarla ilişkilendirmek gibi bir yöntem izlenmiştir.

İkinci soru için, poz uzayı ve ağırlıkları üzerinden tekdüze (uniform) dağılım kullanılabileceği gibi bazı duyarga modellerine göre de parçacıklar direkt olarak üretilebilir [62]. Anlatılanlar toparlandığında AMCL algoritması aşağıdaki gibi verilebilir.

Algoritma: AMCL

Girdi: X_{t-1}, u_t, z_t, m

$static\ w_{slow}, w_{fast}$
$X_t = X'_t = 0$
$for\ m = 1\ to\ M\ do$
  $x_t^{[m]} = hareket\_modeli(u_t, x_{t-1}^{[m]})$
  $w_t^{[m]} = olcum\_modeli(z_t, x_t^{[m]}, m)$
  $X'_t = X'_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
  $w_{ort} = w_{ort} + \frac{1}{M} x_t^{[m]}$
$endfor$
$w_{slow} = w_{slow} + a_{slow}(w_{ort} - w_{slow})$
$w_{fast} = w_{fast} + a_{fast}(w_{ort} - w_{fast})$
$for\ m = 1\ to\ M\ do$
  $do \max\left\{0.0, 1.0 - \frac{w_{fast}}{w_{slow}}\right\} olasılığıyla$
    $X_t\ ye\ rastgele\ poz\ ekle$
  $else$
    $x_t^{[i]}\ yi\ X_t\ ye\ ekle$
  $enddo$
$endfor$
$return\ X'$

Çıktı: X'

AMCL'de MCL'den farklı olarak, 2. döngü içerisinde örnekleme işlemi sırasında rastgele pozların üretilmesi $\max\left\{0.0, 1.0 - \frac{w_{fast}}{w_{slow}}\right\}$ olasılığı ile sağlanmaktadır. Aksi takdirde

örnekleme alışık olunan şekilde yapılmaktadır. Rastgele örnekler ekleme olasılığı, kısa ve uzun dönem ölçüm olasılıkları ortalamaları arasındaki farklılıklar hesaba katarak yapılır. Eğer kısa dönem olasılığı, uzun dönem olasılığından daha iyi veya eşitse rastgele parçacık eklemesi gerçekleşmez. Daha kötü olduğu durumda ise denklemde belirlenen oranlarda rastgele parçacıklar eklenir. Böylece, ölçüm olasılıklarındaki ani bir bozulma rastgele örnek sayısının artışına sebep olacaktır.

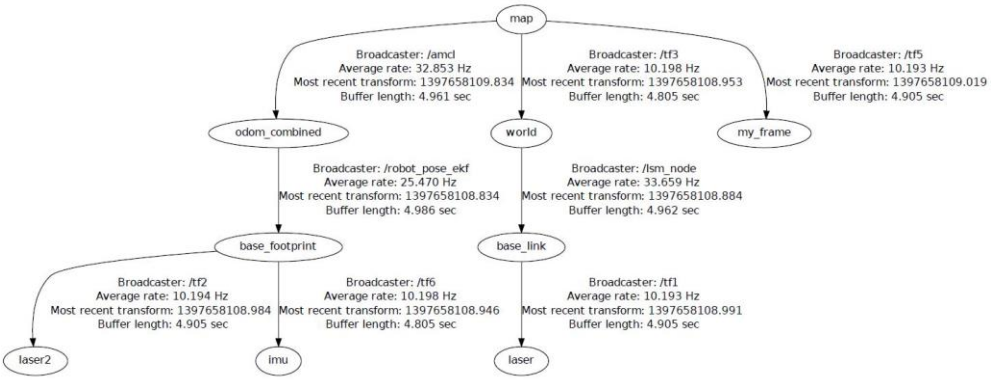
Çalışma kapsamında ROS ortamında gerçekleştirilmiş "amcl" paketi kullanılmıştır.

9.3 **ROS Hydro Ortamında Gerçekleştirilmiş "amcl" Paketinin Özellikleri**

ROS ortamında gerçekleştirilen "amcl" paketi Sebastian Thrun'ın "Probabilistic Robotics" kitabında anlatılan algoritmayı temel almıştır. Çalışma gerçekleştirilirken kullanılan paketin bu sürümünde sadece lazer mesafe ölçüm duyargası tarafından çıkarılan harita kullanılmıştır. Başka herhangi bir duyargadan yararlanılmamıştır.

"amcl" paketi "sensor_msgs/LaserScan" tipindeki /scan mesajına, "tf/tfMessage" tipindeki /tf mesajına, "geometry_msgs/PoseWithCovarianceStamped" tipindeki /initialpose mesajına ve "nav_msgs/OccupancyGrid" tipindeki /map mesajına abone olarak "geometry_msgs/PoseWithCovarianceStamped" tipinde /amcl_pose mesajı ve "geometry_msgs/PoseArray" tipindeki /particlecloud mesajlarını yayınlar [46]. Buradan da anlaşılabileceği gibi paketin çalıştırılabilmesi için bir haritaya ihtiyaç vardır. Ardından bu harita üzerinde robotun başlangıç konumuna dair bir bilgi verilmesi gerekmektedir. Konfigürasyon dosyaları hazırlanırken bu başlangıç konumunun her zaman haritanın (0,0) noktası olabileceğini belirtilebileceği gibi, bu nokta aynı zamanda harita "rviz" ekranı üzerinde açıldığında işaretçi yardımı ile de belirlenebilir. "amcl" paketi aynı zamanda bir dönüşüm (tf) de yayınlamaktadır. Bu mesaj içerisinde /odom frame'inden /map frame'ine bir bağlantı bulunmaktadır. Daha önceden de hatırlanabileceği üzere /odom'dan /map'e bağlantı gMapping üzerinden yapılıyordu. Ancak haritanın önceden var olması sebebiyle gMapping'in çalıştırılmasına gerek olmayacak ve bu bağlantı "amcl" paketi üzerinden yapılabilecektir. Şekil 9.1'de bu bağlantıyı gösteren bir dönüşüm ağacına yer verilmiştir. Diğer bağlantılar önceki başlıklarda verilen dönüşüm ağaçlarında olduğu gibidir. Her zamanki gibi LSM, poz bilgisi üretmek, RPE de kombine

odometri bilgisi üretmek için kullanılmaktadır. Her ne kadar bu deneyde IMU kullanılmasa da, dönüşüm ağacındaki bağlantıyı sağlaması için RPE kullanılmıştır.

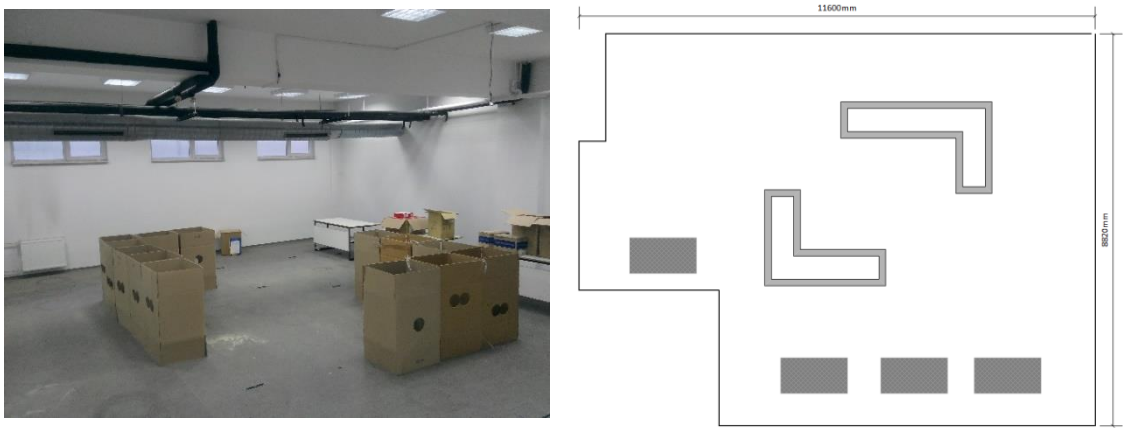


Şekil 9.1 /odom_combined penceresinden /map penceresine olan bağlantının "amcl" tarafından sağlandığı örnek bir dönüşüm ağacı

Çalışma kapsamında “amcl” paketinin kullanılmasındaki temel amaç bu pakete, ROS Hydro sürümü üzerinde gerçekleştirilen, robotun otonom sürümünü sağlayan “navigation_stack” paketinin kullanılabilmesi ihtiyaç duyulmasıdır.

9.4 **AMCL Deneysel Sonuçları**

ROS Hydro sürümü üzerinde gerçekleştirilen “amcl” paketinin çalıştırılabilmesi için bir takım paketlerin önceden çalıştırılması gerekmektedir. Yani daha önce de değinildiği gibi “amcl” paketinin abone olmak istediği mesajlar vardır. Bunlar /scan, /tf ve /map mesajlarıdır. Bu deney kapsamında farklı deneyler için dinamik olarak şekli değiştirilebilecek bir ortam hazırlanmıştır. Farklı denemeler yapılmış ve konum kestirimini nispeten zorlayacak bir ortam oluşturulmuştur. Bu, aynı geometrik şekilli nesnelerin ortama yerleştirilmesi ile oluşturulmuştur. Şekil 9.2-a’da çalışma ortamının son haline ilişkin görsele, Şekil 9.2-b’de ise plana yer verilmiştir. Şekil 9.2-b’den de görülebileceği gibi çalışma ortamının ortasında birbirine çok benzeyen “L” şeklinde iki tane nesne oluşturulmuş ve konum kestirimi için “amcl” paketine verilecek haritanın bu ortamda çıkarılması sağlanmıştır. Kullanılacak harita için ortamda robot bir tur gezdirilmiş ve gezdirilmesi esnasında gMapping çalıştırılmıştır. Gezinim sonunda Şekil 9.3’de verilen harita ortaya çıkmıştır.



(a) (b)

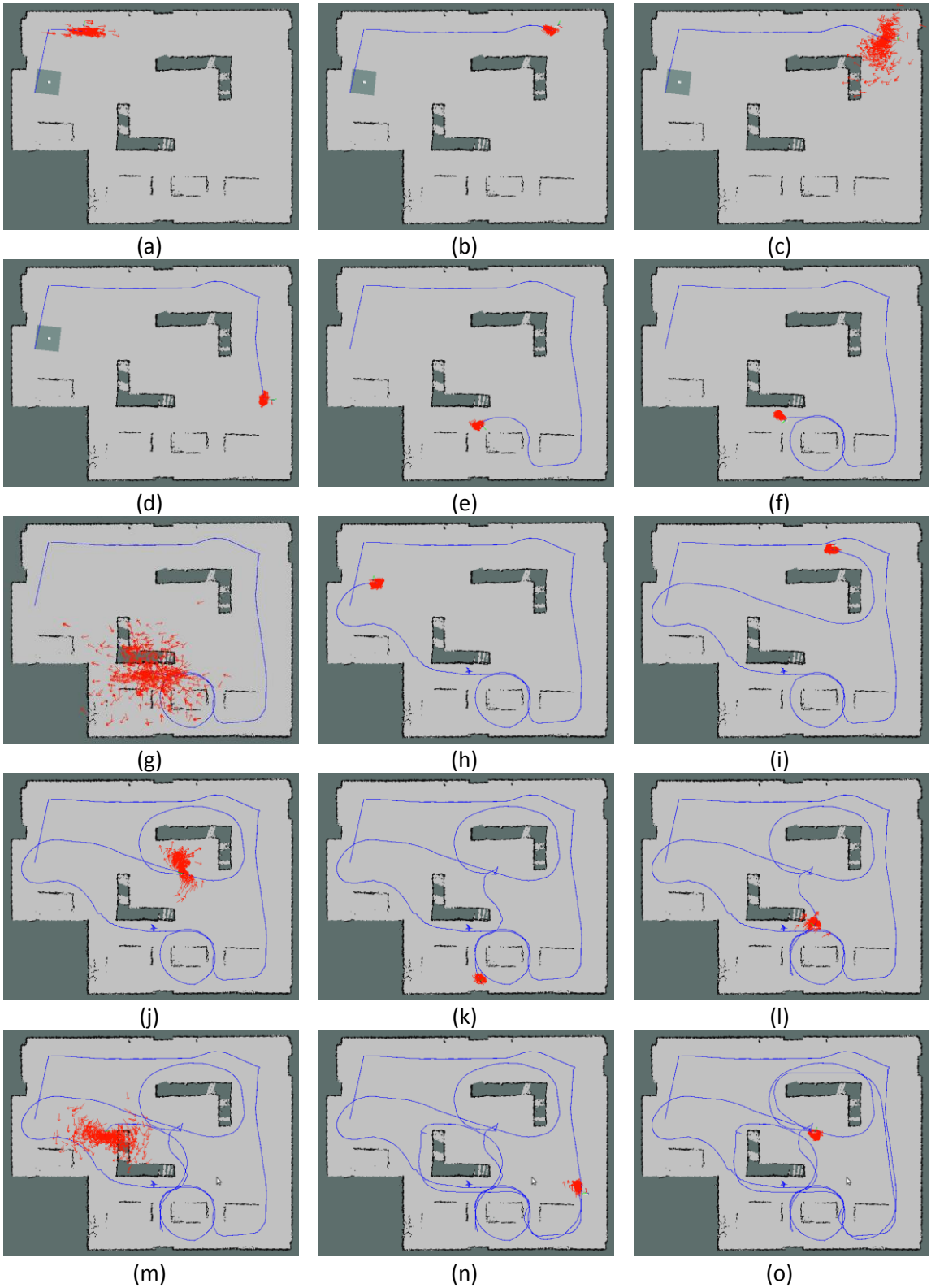
Şekil 9.2 (a) Çalışma alanına ilişkin görsel, (b) çalışma alanına ilişkin plan

Şekil 9.3 "amcl" deney ortamının gMapping ile haritalanması

Bu sonuç ROS'un "map_server" paketi kullanılarak ".yaml" formatında kaydedilmiştir. Yine Şekil 9.3'den görülebileceği üzere Şekil 9.2-b'deki ortam planına benzer özelliklerde bir harita çıkarımı yapılmıştır. "amcl" paketinin çalıştırılabilmesi için diğer gerekli olan şey ise /scan mesajıdır. Robot lokalize olmak için ortamda gezinirken lazer mesafe ölçüm duyargası da çalıştırılarak bu mesajın üretilmesi sağlanmıştır. Aynı zamanda /tf mesajına ihtiyaç duyan paket için çeşitli dönüşümler yazılmış ve bu mesaj üretilmiştir. Bu dönüşümlerin ne şekilde gerçekleştirildiği EK B'de launch dosyası olarak verilmiştir. Son olarak robotun konum kestirimiyle düzelteceği odometri mesajının LSM tarafından üretilmesi sağlanmıştır. Bütün düğümlerin çalıştırılması gerçekleştikten sonra robot alan içerisinde gezdirilmiştir. Bu gezinim için "rviz" ekranından alınan

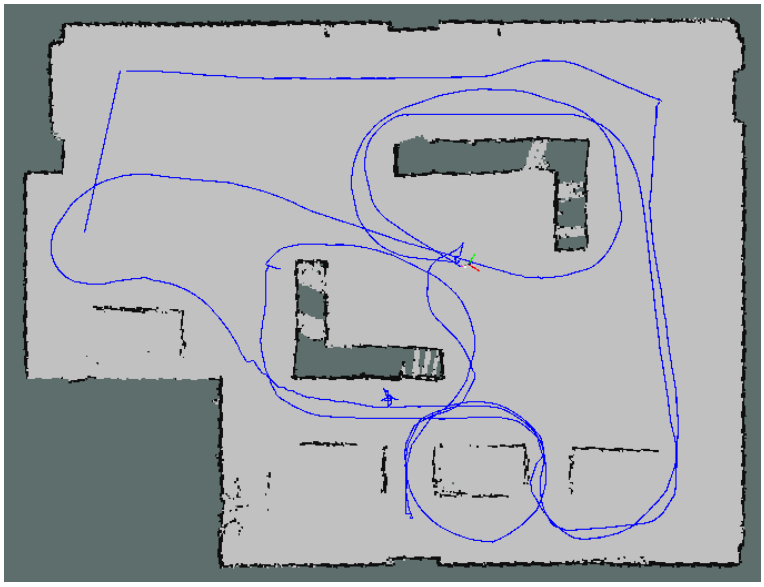
görüntüler Şekil 9.4'te adım adım verilmiştir. Mavi renkle gösterilen gezinge "amcl" paketi tarafından üretilen /amcl_pose mesajının sürekli halidir. Kırmızı ile gösterilen oklar ise, "amcl" paketinin /particlecloud mesajının gösterimidir. Bunlar AMCL'nin parçacıklarıdır ve robotun konumuna dair olasılıksal olarak tahminler içermektedirler. Her bir parçacık robotun konumunu tahmin ettiği yerde "rviz" ekranı üzerine çizdirilmiştir. Yeşil-kırmızı-mavi kombinasyonu ile robotun konumu gösterilmektedir. Robotun konumu "amcl" paketinin ürettiği parçacıkların en yüksek olasılıklı olanından alınmaktadır. Bu sebeple zaman zaman gezingeden de görülebileceği gibi atlamalar gerçekleşmiştir. Bu atlamalar esnasında daha düşük olasılıkta olan bir parçacığın olasılığı yükselmiş ve robotun yeni konumu, bu parçacığın tahmini olacak şekilde güncellenmiştir. Robotun başlangıç pozisyonu haritanın (0,0) noktası olacak şekilde parametre olarak verilmiştir. Ancak robotun başlangıç konumu haritanın (0,0) konumu değildir. Bu sebeple robotun başlangıç konumu "rviz" üzerindeki "2D Pose Estimate" butonu ile aniden değiştirilerek gerçekte olduğu konuma getirilmiştir. Bunun yapılması sırasında robotun pozisyonunda başlangıçta bir atlama meydana gelmiştir. Bu atlama Şekil 9.4-a'da görülebilmektedir. "amcl" paketi parçacıklarının tamamını harita üzerine dağıtmamaktadır. Onun yerine başlangıç pozisyonu olarak verilen yerin etrafında bir normal dağılım kullanarak parçacıkları dağıtmaktadır. Bu sebeple başlangıçta parçacıklar bütün haritaya dağılmamaktadır. Şekil 9.4-a'da görülebileceği gibi "rviz" ekranından verilen başlangıç konumu etrafında parçacıklar toplanmıştır. Belirli bir süre robot ilerledikten sonra bütün parçacıklar olasılıksal olarak robotun yerini tahmin etmiştir. Bu durum Şekil 9.4-b'de görülebilmektedir. Şekil 9.4-c'ye geçildiğinde ise robot kendi etrafında hızlı bir şekilde dönmeye başlamıştır ve bu sayede parçacıkların önem faktörlerinin azalması sağlanmıştır. Algoritmanın adımları detaylı bir şekilde anlatılırken de değinildiği gibi parçacıkların önem faktörleri azaldığı durumda AMCL yöntemi rastgele yeni parçacıklar atamaktadır. Şekil 9.4-c'de görülen parçacıkların dağılmasının sebebi de budur. Şekil 9.4-g'ye kadar robot ortam içerisinde sürülmüş ardından yine kendi etrafında dönmesi sağlanmıştır. Bu sefer görülebileceği gibi daha fazla parçacık rastgele olarak daha uzak noktalara atanmıştır. Bu durum parçacıkların önem faktörünün daha da azaldığı anlamına gelmektedir. Yine Şekil 9.4-h'ye geldiğinde ise bu dağılımın toparlanabildiği görülebilmektedir. Aynı şekilde robotun kendi

etrafında hızla dönmesi Şekil 9.4-j'de ve Şekil 9.4-m'de de gerçekleştirilmiştir ve her seferinde parçacıkların olasılıklarının yenilenerek robotun gerçek konumda birleştiği gözlemlenmiştir.



Şekil 9.4 "amcl" paketinin gezinge çıkarımı için çalıştırılmasının aşamalı gösterimi

Şekil 9.5'te çıkartılan gezingenin son haline yer verilmiştir. Gezinge incelendiğinde görülebileceği üzere ufak atlamalar gerçekleşmiştir. Bu durum farklı parçacıkların önem faktörlerinin artmasıyla robotun pozunu belirler hale gelmesinden kaynaklanmıştır.

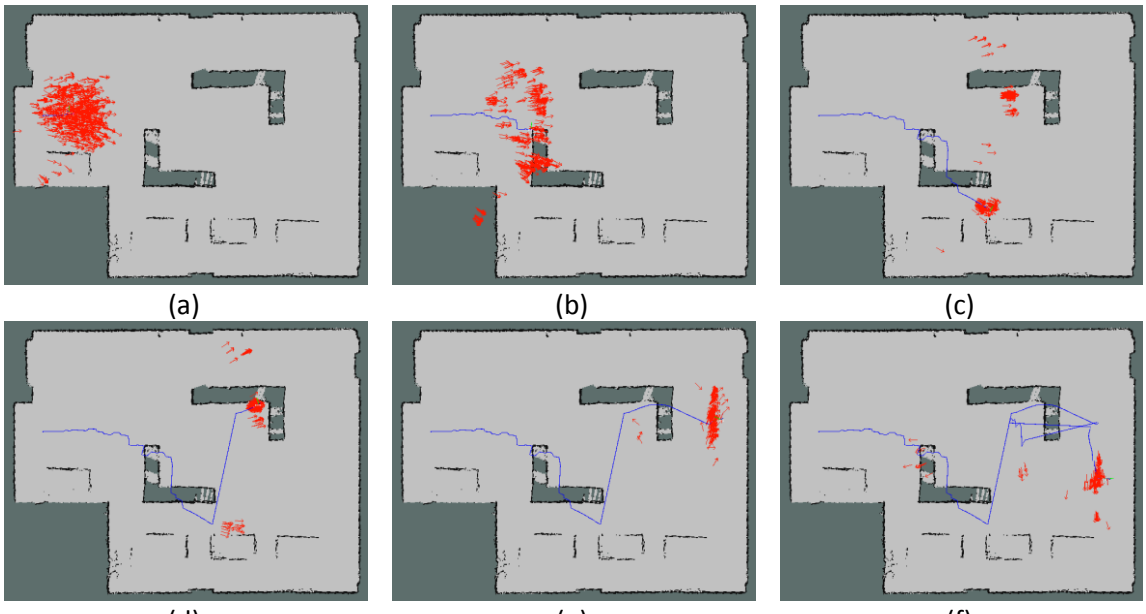


Şekil 9.5 "amcl" paketinin çıkarttığı gezingenin son hali

Şuana kadar anlatılan "amcl" deneyinde robotun başlangıç pozisyonu düzgün bir şekilde verilmiş ve robotun kaçırılması gerçekleştirilmemiştir. Şekil 9.6'da adım adım görsellerinin verildiği deneyde ise robotun başlangıç konumu, gerçek başlangıç konumu olarak verilmemiş, haritanın (0,0) noktasından başlaması sağlanmıştır. Bu deneyde, yukarıdaki deney yapılırken alınan lazer verileri kullanılmıştır. Bu sayede her iki deney için kullanılan verinin aynı olası sağlanarak kontrollü bir deney gerçekleştirilmiştir. Ayrıca "rviz" ekranındaki "2D Pose Estimation" butonu kullanılarak (Şekil 9.6-g'de görülebileceği gibi) robot lokalize olduğu zamanlarda kaçırılmış ve yeniden lokalize olması için serbest bırakılmıştır.

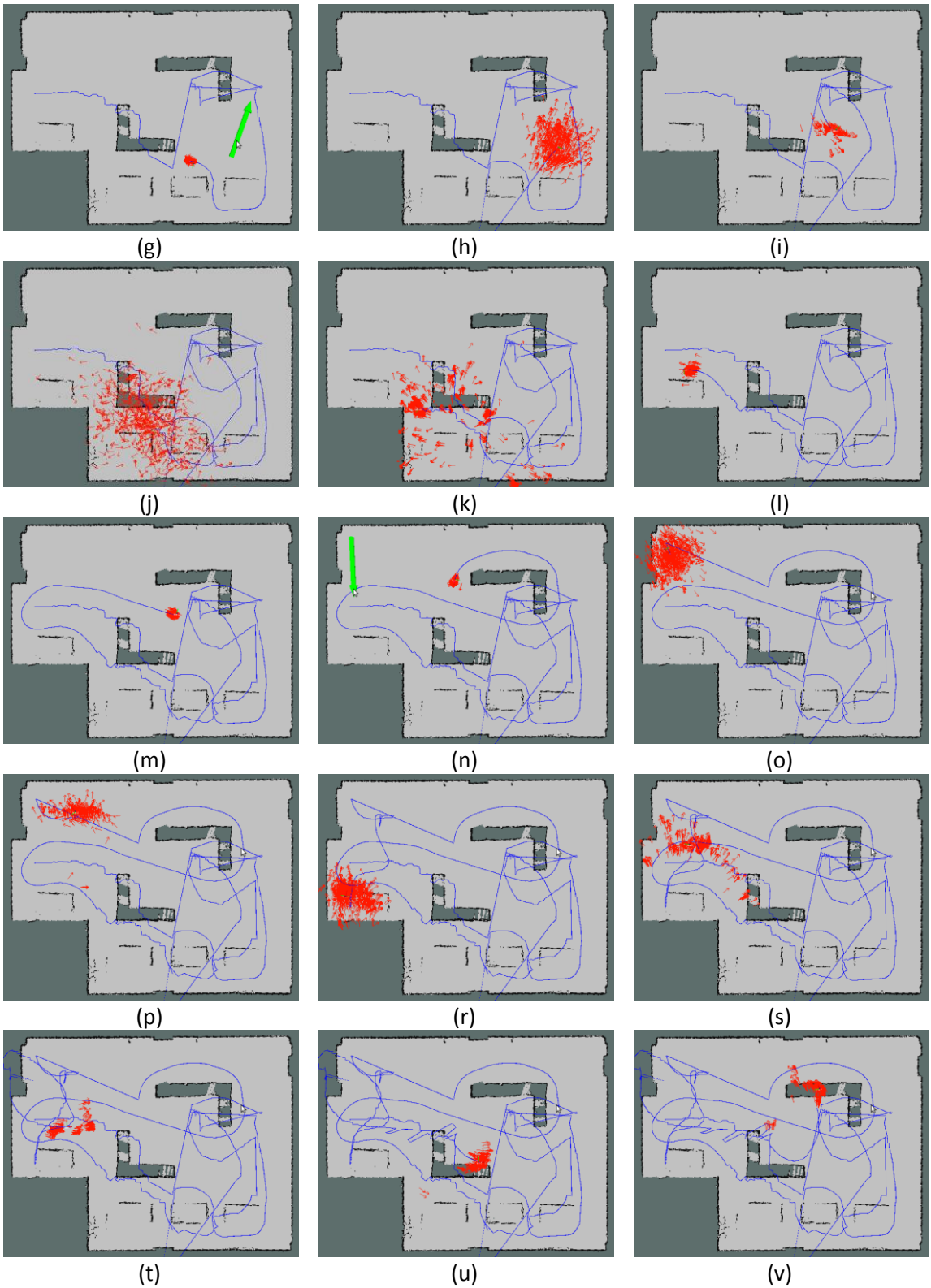
Şekil 9.6-a'dan Şekil 9.6-b'ye geçişte bir önceki deneyde olduğu gibi parçacıkların lokalize olamadığı ve farklı yerlerde yoğunlaştıkları görülebilmektedir. Çünkü bu ana kadarki hiçbir parçacık robotun gerçek konumu temsil edememiştir. Şekil 9.6-c'ye gelindiğinde rastgele atanan parçacıklardan bir kısmı (haritanın üst tarafındaki parçacıklar) robotun gerçek konumuna yakın yerlerde bulunmaktadırlar. Böyle olduğu için Şekil 9.6-d'ye gelindiğinde robotun pozisyonunun aniden değiştiği ve en azından belirli bir yerde daha fazla yoğunlaştığı gözlemlenmiştir. Ancak robotun gerçek

pozisyonu orası da değildir. Bu durum Şekil 9.6-e'den Şekil 9.6-f'ye geçişte gözlemlenebilir. Şekil 9.6-f'deki gezingenin birkaç kez atlamalar gerçekleştirdiği ve sonunda robotun gerçek pozisyonuna kavuştuğu görülebilir. Şekil 9.6-g'ye gelindiğinde Şekil 8.4-e'de olduğu kadar yol gidilmiştir ve parçacıklardan da görülebileceği gibi konum kestirimi gerçekleştirilmiştir. Bu aşamada robotun kaçırılması sağlanarak Şekil 9.6-g'deki işaretleyicinin gösterdiği yere pozisyonu taşınmıştır. Bununla beraber Şekil 9.6-h'de görülebileceği gibi parçacıklar dağılmıştır. Şekil 9.6-j'ye gelindiğinde parçacıkların önem faktörlerinin iyice azaldığı ve bu sebeple birçok rastgele parçacık atandığı görülebilir. Yapılan bu işin AMCL'nin avantajlarından olduğuna tekrardan değinmek gerekir. Çünkü parçacıklar bu şekilde rastgele dağıtılmazlarsa kaçırılan robot problemi için tekrardan lokalize olmaları imkansız hale gelecektir. Şekil 9.6-l'ye gelindiğinde konum kestiriminin gerçekleştiği görülebilir. Robot, Şekil 9.6-n'ye geldiğinde yaklaşık olarak Şekil 8.4-i kadar yol gitmiştir. Bu aşamada yine Şekil 9.6-n'deki işaretçi ile gösterildiği gibi robot kaçırılmış ve tekrardan lokalize olması beklenmiştir. Ancak gezinimin bu noktadan sonra kısa sürmesi sebebiyle robot lokalize olamadan bag dosyasının oynatımı tamamlanmıştır. Şekil 9.6-p'den Şekil 9.6-v'ye kadar olan görsellerde robotun lokalize olamadığı durumlar gösterilmiştir. Bunlar incelendiğinde parçacıkların bir yerde toplanamadığı ve zaman zaman önem faktörleri değişerek rastgele parçacıkların artmasına sebep oldukları gözlemlenebilir.



(a) (b) (c)

(d) (e) (f)



Şekil 9.6 Robotun kaçırıldığı "amcl" deneyinin adım adım gösterilmesi

Netice itibariyle bu deney kapsamında robot lokalize olduğu durumlarda zaman zaman kaçırılmış ve tekrardan lokalize olması için serbest gezinime bırakılmıştır. Bu durumlardan bazılarında tekrardan konum kestirimi gerçekleşmiş bazılarında ise

gerçekleşememiştir. Gerçekleşemeyen durumlar için yeterli zaman verilseydi onların da lokalize olması parçacıkların rastgele dağıtımından dolayı gerçekleşecekti. Şekil 9.7'de bu gezinim için çıkarılan gezingenin ve parçacıkların dağılımının son haline yer verilmiştir. Görüldüğü gibi parçacıklar birbirlerinden farklı olasılıklar taşımaktadır ve çıkarılan gezinge parçacıkların önem faktörlerinin değişmesiyle robotun pozisyonunu belirleyen parçacığın değişmesi sonucunda atlamalar gerçekleştiği için oldukça bozuktur.

Şekil 9.7 Robotun bir kaç kez kaçırıldığı deney için son gezinge

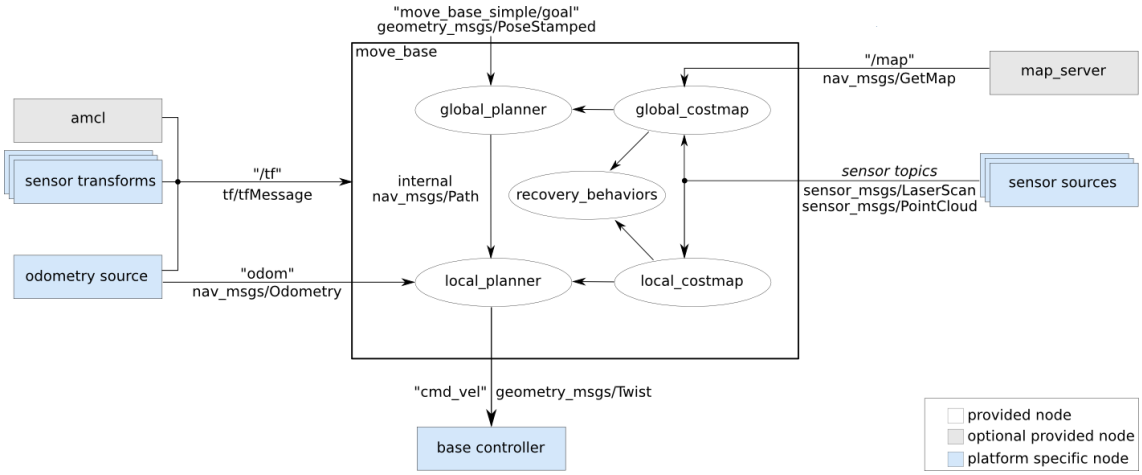
BÖLÜM 10

ROS NAVIGATION STACK PAKETİ

ROS Hydro sürümü üzerinde gerçeklenen “Navigation Stack” paketi kavramsal seviyede oldukça basit bir uygulamadır. Temel olarak mobil robottan odometri ve duyarga bilgilerini alıp bir hız komutu üreterek bu hız komutunu mobil robota gönderir. Ancak rastgele olasılıksal hareket eden robotlarda bu durum biraz daha karmaşıktır. ROS üzerinde “navigation stack” paketinin çalışabilmesi için /tf dönüşüm ağacına ve ROS mesajları tipinde yayımlanan duyarga bilgilerini içeren mesajlara ihtiyaç vardır. Bunun yanında, robotun boyutlarının ve şeklinin ayrıntılı bir şekilde yapılandırma dosyaları içerisine yazılması gerekmektedir. Çünkü “navigation stack” robotun navigasyonu esnasında, robotun boyutlarının getirdiği kısıtlar çerçevesinde hareketlerine karar verecektir. Örneğin robotun geçemeyeceği kadar dar olan boşluklardan robotu geçirmeyecek onun yerine alternatif güzergahlar arayacaktır. Bu paket bütün robot platformlarının genel kullanımı için üretilmiştir. Ancak temelde birkaç tane donanım gereksinimine ihtiyaç duymaktadır [46]:

- Robotun hareket modeli, (x, y, θ) hızları şeklinde olmalıdır.
- Robot platformu üzerinde, konum ve duruş kestirimi veya haritalama için bir lazer mesafe ölçüm duyargası bulunmalıdır.
- Bu paket, kare robotlar üzerinde gerçekleştirilmiştir. Yani en iyi sonucu kare veya çember robot platformları üzerinde verecektir. Platformlar karemsi şekillerden farklı olduğu durumlarda, robotun ana hatlarının yapılandırma dosyasında çok iyi belirtilmesi gerekmektedir.

Bu aşamadan sonra “navigation stack” paketinin ne şekilde çalıştığına değinilecek ve bu Şekil 10.1’de verilen ilgili paketin yapılandırma şeması üzerinden gerçekleştirilecektir.



Şekil 10.1 "Navigation stack" yapılandırma şeması [63]

Şekil 10.1’in sağ alt köşesinde de belirtildiği gibi, beyaz arka plandaki modüller, “navigation stack” tarafından sağlanan düğümler, gri renkle gösterilenler modüller opsiyonel olarak “navigation stack” tarafından sağlanan düğümler ve mavi renkle gösterilenler robot platformunun navigasyon paketine sağladığı düğümlerdir. “amcl” paketi anlatırken, bu şekilde gri ile gösterilen “amcl” ve “map_server” düğümlerinin nasıl kullanılacağı bahsedilmişti. Aynı şekilde, robot platformu tarafından sağlanan /tf, /odom, /scan ve /cmd_vel mesajlarının nasıl yayınlandığına ve içlerinde ne şekilde mesajlar barındırdıklarına değinilmişti. Bu başlık altında ise beyaz arka planla gösterilen “navigation stack” paketi tarafından oluşturulan modüller üzerinde durulacaktır. Bu modüllerin hepsi “move_base” düğümünün çalıştırılması ile oluşmaktadır. Bu düğümü çalıştıran launch dosyasına EK B’de yer verilmiştir.

“Navigation stack” 2 farklı şekilde çalıştırılabilir:

- Haritanın önceden çıkarılıp “map_server” düğümü ile yayınlandığı durum için.
- Haritanın navigasyon esnasında gMapping veya hectorMapping gibi haritalama algoritmaları tarafından eşzamanlı olarak çıkarılıp yayınlandığı durum için.

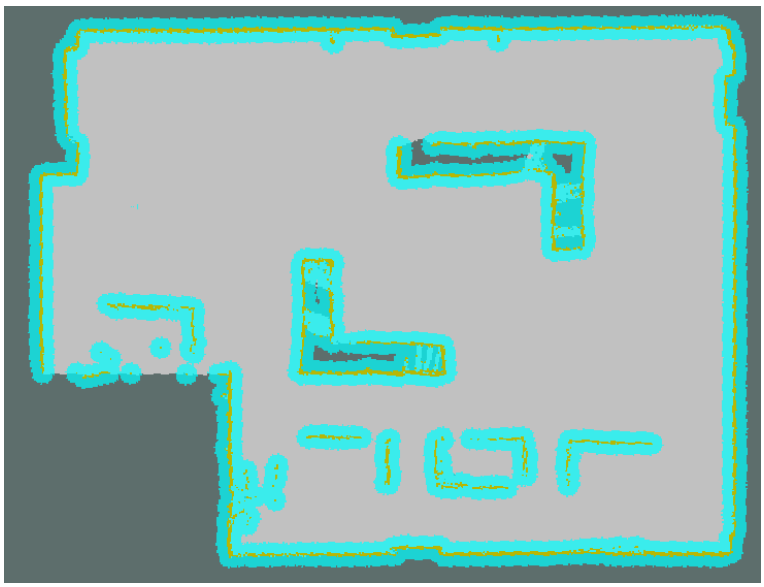
Bu tez çalışması kapsamında sadece haritanın önceden bilindiği durumlar için çalışmalar ve deneyler yapılmıştır.

10.1 **CostMap Yapılandırmaları**

“navigation stack” gezilen ortamdaki engellerle ilgili bilgi depolayan 2 farklı “costmap” (maliyet haritası) kullanmaktadır. İlki “global_costmap” olarak isimlendirilir ve bunun ile tüm ortamın uzun dönemli engel haritası çıkartılır, ikincisi ise daha çok engelden kaçınmak için kullanılan, “global_costmap” haritasında olmayan engelleri içeren “local_costmap” olarak isimlendirilen maliyet haritalamasıdır. Her ikisini de ortak olarak kapsayan yapılandırma ayarları (engel kalınlıkları, renkleri, engel eşik değerleri) olmakla beraber, her ikisi için de ayrı ayrı yapılandırmalar yapılabilecek dosyalar oluşturulabilir. Çalışma kapsamında kullanılan her 3 yapılandırma dosyası (costmap_common_params.yaml, global_costmap_params.yaml, local_costmap_params.yaml) EK B’de bulunabilir.

10.1.1 “global_costmap” Yapılandırması

Global maliyet haritasının çıkarımında “map_server” ile yayınlanan /map mesajı alınır ve yapılandırma dosyasında belirlenen parametreler kullanılarak duvarların/engellerin kenarlarına robotun girmesi yasak bölgeler çizilir. Bunun bir örneği Şekil 10.2’de gösterilmiştir. Burada duvarlardan itibaren girilmesi yasak bölgeler 30 cm olarak belirlenmiştir ve harita üzerinde “cyan” rengi ile gösterilmiştir. Sarı renkle gösterilen bölgeler ise duvarlardır.



Şekil 10.2 global_costmap sonucu

Robot navigasyonu esnasında gitmesi için çizeceği yolu bu yasak bölgelerin dışında kalacak şekilde çizmektedir. Bu sayede global engellere takılmadan varmak istediği yere gidebilecektir.

10.1.2 "local_costmap" Yapılandırması

Lokal maliyet haritasının oluşturulmasının temel nedeni aniden oluşan engellerden kaçınmaktadır. Mesela robot navigasyona devam ederken önüne bir kişi geçtiğinde, onu laser mesafe ölçüm duyargası ile algılayarak lokal maliyet haritasına eklemektedir. Maliyet haritası bu sayede navigasyon ile eş zamanlı olarak güncellenebilmektedir. Maliyet haritası güncellendiğinde gidilecek hedefe olan yol tekrardan hesaplanmakta ve robotun hareketi bu şekilde değiştirilmektedir. Şekil 10.3'te Şekil 10.2'de verilen global maliyet haritasına ek olarak lokal maliyet haritası da verilmiştir. Görülebileceği gibi Şekil 10.2'de olmayan engeller Şekil 10.3'te ortaya çıkmıştır.

Şekil 10.3 local_costmap sonucu

Lokal maliyet haritasındaki engeller sabit olabilmekle beraber aynı zamanda değişken de olabilmektedir. Mesela Şekil 10.3'teki robotun sol tarafındaki lokal maliyetler aynı zamanda global maliyetlerdir. Yani odanın duvarlarını görmektedir. Ancak sağ tarafında algıladığı ve çizdiği engeller değişken engellerdir ve robotun aynı yere bir sonraki gelişinde bu engelin orada olmama olasılığı da vardır. Bu gibi durumlar "Navigation Paketinin Deneysel Sonuçları" bölümünde detaylı olarak ele alınacaktır.

10.2 **“global_planner” ve “local_planner” Yapılandırması**

Oluşturulan lokal ve global planlar ile robotun belirlenen hedefe gitmesi sağlanmaktadır. Yine global planlar verilen hedefe maliyet haritalarına göre yolu çizerken, lokal planlar engelden kaçınmayı içermektedir. Bu planların çıkarılması için de çeşitli yapılandırma parametreleri kullanılmaktadır. Bu parametreleri (maksimim, minimum hız, dönme açısal hızlarının artış miktarları, vb.) barındıran yapılandırma dosyası (base_local_planner_params.yaml) EK B’de verilmiştir.

Tüm düğümler çalıştırıldığında ortaya çıkan dönüşüm ağacı “amcl” paketi çalıştırıldığında ortaya çıkan dönüşüm ağacı ile (Şekil 9.1) aynıdır.

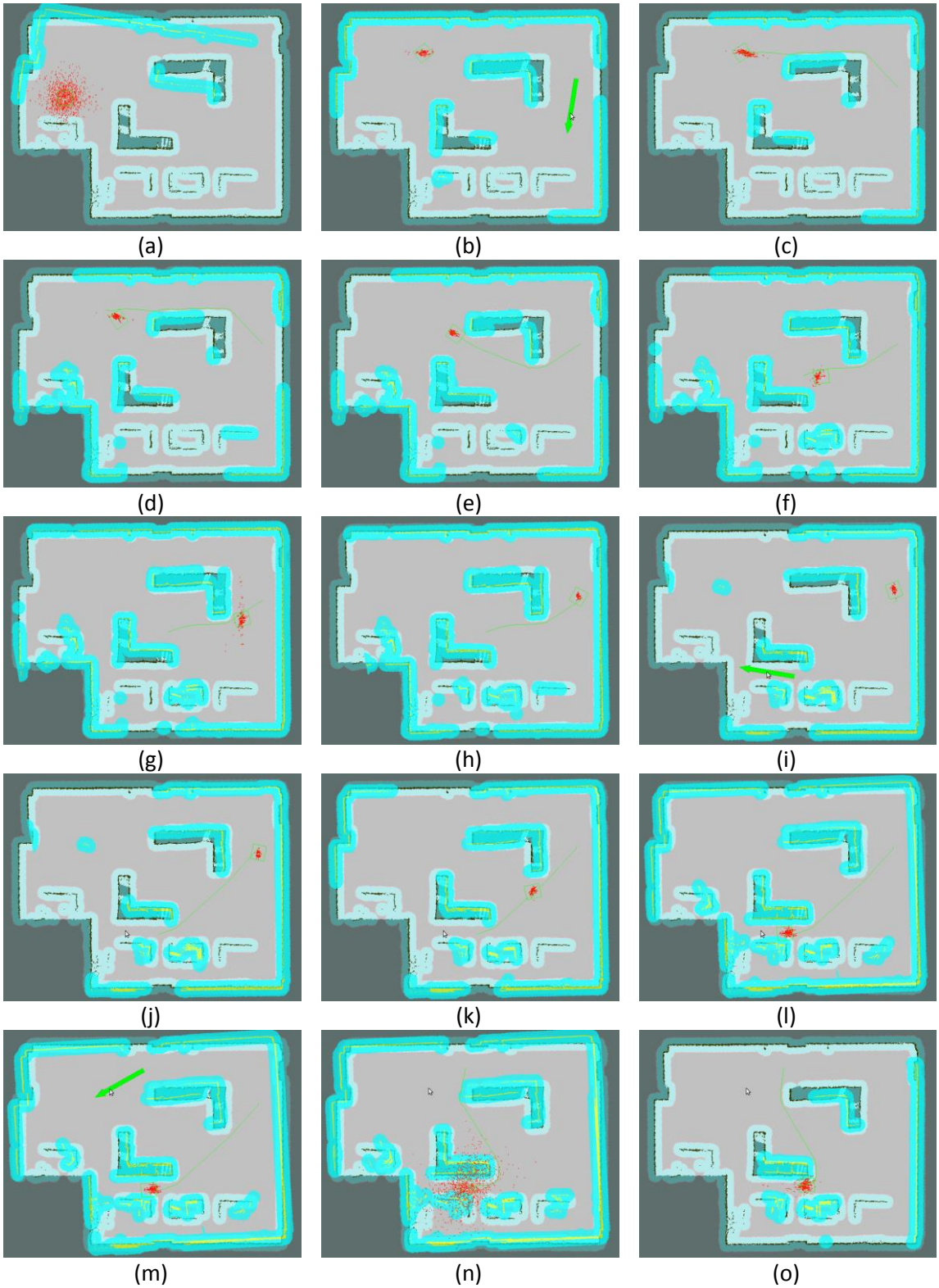
10.3 **Navigation Paketinin Deneysel Sonuçları**

ROS Hydro sürümü üzerinde gerçeklenen “navigation stack” paketinin nasıl çalıştırılacağını ve ne çeşit gereksinimleri olduğu bir önceki bölümde anlatılmıştı. Bu başlık altında, “amcl” paketinin deneylerinin yapıldığı ortamda “navigation stack” paketinin ortaya çıkan deneysel sonuçları verilmiştir. Deney kapsamında kullanılan çalışma ortamı Şekil 9.2-a’de, planı ise Şekil 9.2-b’de verilmiştir. Deneyde 2 farklı bilgisayar kullanılmıştır. Birisi üzerinde düğümlerin çalıştığı, robot platformunu kontrol eden ve navigasyon esnasında robotun üzerinde bulunan bilgisayar, ikincisi ise robot üzerindeki bilgisayara uzaktan bağlanarak robota komut vermekte kullanılan ve deney kapsamında ekran görüntülerinin alındığı bilgisayardır. “rviz” üzerinden alınan ekran çıktılarında yeşil ok ile gösterilen işaretçi, robotun gitmesi istenilen hedefi vermek için kullanılmıştır. Genellikle parçacıkların içerisinde toplandığı yeşil dikdörtgen şeklindeki poligon ise robotu temsil etmektedir. Robotun boyutlarından ikişer santimetre fazla olacak şekilde oluşturulmuştur. Hedef gösterildikten sonra navigasyon paketi hedefe olası gidebileceği global sonucu çıkartmaktadır. Bu yol yeşil renk ile gösterilmiştir.

Şekil 10.4-a’da robotun başlangıç pozisyonu, parçacıkların dağılımı, global maliyet haritası ve lokal maliyet haritası görülebilmektedir. Robotun başlangıç pozisyonu, robotu temsil eden parçacığın başlangıç pozisyonu ile aynı olmadığı için lokal maliyet haritasındaki duvarlar belirli bir açıyla eğik çıkmıştır. Şekil 10.4-b’ye gelindiğinde robot lokalize olmuş ve lokal maliyet haritasındaki duvarlar global maliyet haritasındaki

duvarlarla örtüşmüştür. Aynı görsel üzerindeki yeşil ok ile görülebileceği gibi robota gitmesi için bir hedef verilmiş ve “navigation” paketi hedefe gidebilmek için Şekil 10.4-c’de görülen şekilde bir yol çıkartmıştır. Bu yol, robotun o a en yüksek olasılığa sahip olan parçacığına göre çizilen yoldur. Başka parçacıklar için farklı şekillerde yollar da oluşturulabilir ve zaman içerisinde farklı bir parçacığın olasılığı yükseldiğinde yol değiştirilebilir. Şekil 10.4-e’de bu durum görülebilmektedir. Robotun gideceği hedef noktası yine aynı kalmış ancak gidilmesi planlanan yol değişmiştir. Şekil 10.4-h’ye gelindiğinde robot hedefine ulaşmıştır ve robot boşta beklemektedir. Yeni bir hedef Şekil 10.4-i’de gösterildiği gibi verilmiştir. Bu hedef diğerine göre biraz daha zordur. Çünkü robotun geçebileceği aralık neredeyse robot poligonu boyutundadır. Bu arada şuna değinmek gerekir ki: robot açık alanda navigasyonu daha hızlı yapmakta, ancak engellere yakın noktalardan geçmesi gereken yerlere geldiğinde gezinimi iyice yavaşlamakta, engellere çarpmamak için bir dizi (ileri-geri şeklinde) hareketler yapmaktadır. Ardından gidilecek yolun planı yapılmış ve Şekil 10.4-l’ye gelindiğinde robot hedefine ulaşmıştır. Burada görülebileceği gibi başlangıçta çizilen yol hiç değiştirilmemiş, sonuna kadar aynı yol kullanılarak hedefe gidilmiştir. Bu durumda, olasılığı en yüksek olan parçacığın hiç değişmediğine ve her zaman robotun pozisyonunu temsil eden parçacık olduğu sonucuna varılabilir. Bu hedefe de ulaşmasının ardından Şekil 10.4-m’de görüldüğü gibi tekrardan bir hedef verilmiştir. Robot öncelikle konum ve duruş bilgisini yitirmiş, ardından lokalize olarak çıkarttığı yol doğrultusunda hedefine ulaşmaya çalışmıştır. Ancak Şekil 10.4-r’den de görülebileceği gibi robotun çizdiği yolun tam ortasına dikilinmiş ve robotun gitmesi engellenmeye çalışılmıştır. Robotun önüne geçen kişi, robot tarafından lokal maliyet haritasında bir engel olarak görülmüş ve Şekil 10.4-s’de görülebileceği gibi hedefe gidebilmek için engelin etrafından dolaşarak yeni bir yol çizmiştir. Bu yolu takip ederek robot platformu verilen hedefe ulaşmıştır. Yani bu sayede “navigation stack” paketinin lazer mesafe duyargası kullanarak çıkartmış olduğu lokal maliyet haritasının da kullanıldığı ve robotun hareketlerinde öncelikle buna bakıldığı görülmüştür. Robotun gezinimi elle sürmede olduğu gibi akıcı değildir. Özellikle engellere yakın geçilen yerlerde robot birçok kez hareket yaparak geçme işlemini yerine getirmektedir. Hatta bazen çok uzun süre aynı hareketleri yaptığı görülmüştür. Ama netice itibariyle eğer hedefe

gidilebilecek bir yol çizilmişse, robot hedefe ulaşmayı her zaman başarabilmiştir. Tabi ki her zaman verilen hedef robotun gidebileceği yerlerde olmamaktadır.



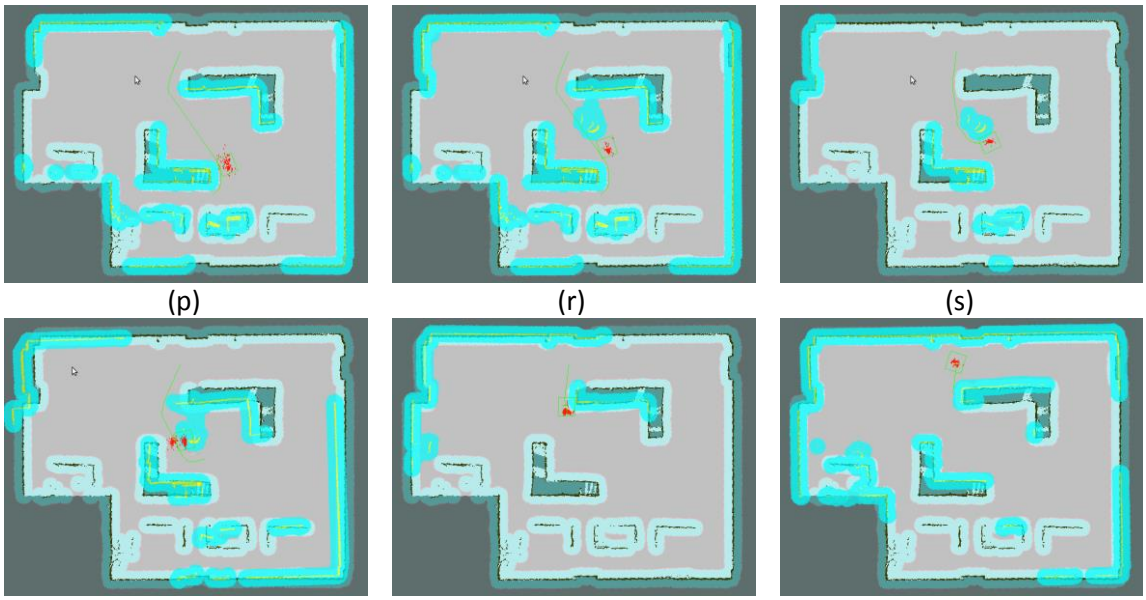
(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

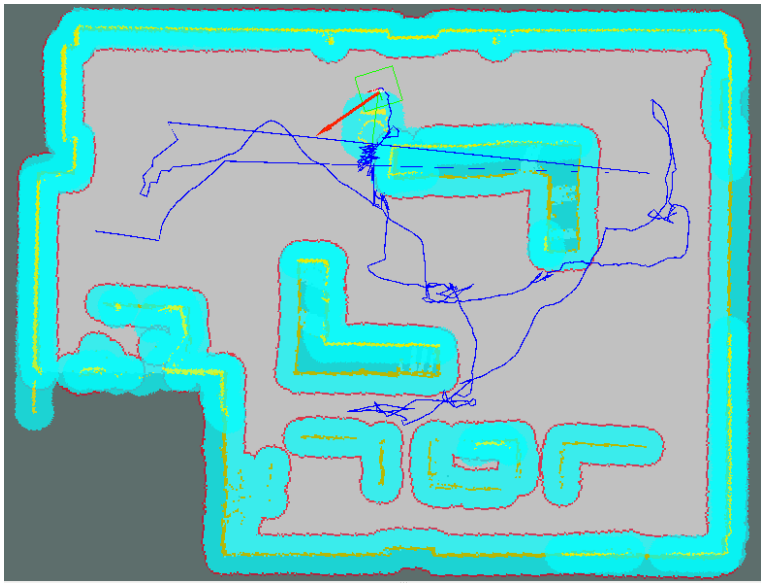
(j) (k) (l)

(m) (n) (o)



Şekil 10.4 “navigation stack” paketi çalışırken adım adım alınan ekran görüntüleri

Eğer bir engelin içerisi hedef olarak gösterilirse, “base_local_planner” modülü tarafından yol çıkartılmayacağı için robot hedefi geçersiz saymaktadır. Bazı durumlarda başlangıçta geçerli olan bir hedef verilmesine rağmen dinamik bir şekilde değişen haritanın hedefe gidilmesini engellediği görülmüştür. Bu tarz durumlarda robot, gidebilmek için birkaç alternatifi denedikten sonra, “hedefe gitme başarısız oldu” uyarısı vererek görevi iptal etmektedir. Robotun, özellikle dar yerlerden geçerken çok fazla kontrol hareketleri yaptığına değinmiştik. Şekil 10.5’de mavi ile gösterilen gezinge incelendiğinde bu hareketlerin sıklığı ve şekli görülebilmektedir.



Şekil 10.5 Navigasyon sonucunda ortaya çıkan gezinge

Netice itibariyle, ROS Hydro sürümü üzerinde gerçekleştirilen “navigation stack” paketi çalıştırılmış ve yapılandırma ayarları dışındaki parametreleri standart olacak şekilde kullanılmıştır. Dar ortamlarda gezilirken, duvardan olan engel miktarı ve robot platformundaki robotun kenarlarından olan fazlalıklar azaltılarak denemeler yapılabilir. Çünkü aksi takdirde robotun geçemeyeceği aralıklar oluşabilmekte ve bundan dolayı yol planı çıkarılamamaktadır. Geniş ortamlarda gezildiğinde ise, engellere çarpma ihtimalini minimuma indirmek için bu değerler arttırılabilir. Yine robotun maksimum hızı yüksek bir değer belirlendiğinde engellere daha çok çarptığı gözlemlenmiştir. Bunun için robotun gezdiği ortama göre hız değerlerinin verilmesi uygun olacaktır.

BÖLÜM 11

SONUÇ VE ÖNERİLER

Bu çalışma kapsamında, arama kurtarma amacıyla tasarlanan diferansiyel sürüşlü, tekerlekli bir robot platformu kullanılarak ROS Hydro sürümü üzerinde gerçekleştirilen çeşitli algoritmaların uygulanması ve testlerinin yapılması sağlanmıştır. Öncelikle, robotun gezinimi esnasında tekerlek ve görsel odometri kullanılmadan sadece lazer mesafe ölçüm duyargası ve atalet duyargası yardımı ile odometri bilgisinin doğru bir şekilde üretilmesi amaçlanmıştır.

Konuyla ilgili olarak ROS ortamında gerçeklenen LSM yöntemi detaylı bir şekilde incelenerek robot platformu üzerinde test edilmiştir. LSM, global bir yöntem olmadığı için yayınladığı poz bilgisine dayalı olarak üretilen odometri bilgisi de artımsal hatalar içermektedir. Bu sebeple, LSM'nin farklı bilgiler içerecek şekilde de kullanımı test edilmiş ve LSM'ye kinematik ve atalet duyargası gibi bilgiler eklendiğinde daha iyi sonuçlar alındığı tespit edilmiştir. Lazer mesafe duyargasının ürettiği lazer taramalarını eşlemeye dayalı bu yöntemin hesap karmaşıklığının ve robot sistemine olan yükünün az olduğu gözlemlenmiştir. Diğer bir yandan lazer duyargasının maksimum ölçüm menziliyle ters orantılı olacak şekilde hatanın arttığı sonucuna varılmıştır. Lazer sistemlerinin çalışmasını etkileyen, çok yansıyan ortamlarda (ayna gibi) yöntemin göreceli olarak daha fazla hatalar barındırdığı gözlemlenmiştir.

LSM ile birlikte üretilen odometrinin robot_pose_EKF yardımıyla diğer odometri bilgileriyle (atalet duyargası vb.) kombine edilerek daha düzgün bir odometri bilgisinin üretilmesi gerçekleştirilmiştir. Özellikle odometrinin daha iyi olduğu durumlarda daha iyi sonuçlar üreten gMapping haritalama algoritmasında kullanılacak odometri

bilgisinin robot_pose_EKF paketi tarafından üretilmesi sağlanmıştır. Bu paket hem LSM'den gelen hem de atalet (IMU) duyargasından gelen odometri bilgilerini birleştirerek yeni bir odometri bilgisi üretmektedir. Bu sayede her iki yönteme göre daha iyi bir odometri bilgisinin üretildiği tespit edilmiş ve haritalama algoritmalarında kullanılmak üzere üretilecek odometri bilgisinin bu paket tarafından yayınlanması gerektiği sonucuna varılmıştır. Robot_Pose_EKF paketi aynı zamanda görsel ve tekerlek odometrisi bilgilerini de odometriyi düzeltmek için almaktadır. Farklı odometri tiplerinin bu metoda girdi olarak verilmesi durumunda daha iyi sonuçlar elde edileceği tahmin edilmektedir.

Çalışma kapsamında bir diğer ele alınan konu sıkça kullanılan eş zamanlı konum belirleme ve haritalama (SLAM) algoritmalarından olan gMapping yöntemidir. Öncelikle teorik alt yapısı incelenerek çalışma şekli anlaşılmış ardından ROS üzerinde gerçekleştirilen gMapping paketinin robot platformu üzerinde denemeleri yapılmıştır. gMapping parçacık tabanlı bir yöntemdir. Bu bağlamda gMapping'in tahmin ettiği konumun daha fazla parçacık tarafından tutulmasıyla daha iyi sonuçların elde edildiği ancak bu durumun sisteme oldukça fazla hesap karmaşıklığı getirdiği gözlemlenmiştir. Hatta belirli bir eşik değerden fazla parçacık kullanıldığında, algoritmanın gerçek zamanlı çalışmaktan uzak olduğu tespit edilmiştir. gMapping'in çalışmasına etki eden bir diğer parametrenin haritanın çözünürlüğü olduğu sonucuna varılmıştır. Yüksek çözünürlüklü haritaların daha küçük ve ayrıntının bol olduğu ortamlarda, düşük çözünürlüklü haritaların ise ayrıntıların az ve göreceli olarak daha fazla alan kapsayan haritalarda kullanılması sonucunun daha iyi olduğu deneysel olarak tespit edilmiştir. Lazer mesafe ölçüm duyargasının maksimum menzilinin haritalamaya oldukça etkisinin olduğu görülmüştür. Öyle ki lazer mesafe ölçüm duyargasının maksimum mesafesi (uRange) parametre olarak 3 metre verildiğinde çıkan sonuçlar 30 metre olduğu duruma göre oldukça kötüdür.

Taramaların birbirleri ile eşleştirilmesiyle çalışan gMapping yönteminde taramalar arasındaki mesafenin belirli bir değerden fazla olması durumunda eşlemenin yapılamadığı gözlemlenmiştir. Yani gMapping ile gerçek zamanlı ortam haritalaması yapılırken robotun gezdiği hız önemlidir. Çok hızlı gezildiğinde taramalar arasında mesafe arttığı için sonuçların kötü çıktığı, çok yavaş gezildiğinde haritalama süresinin

çok fazla uzadığı görülmüştür. Deneysel olarak platformun donanımsal yapısına da bağlı olarak belirlenen bir hızda gezmesinin uygun olduğu tespit edilmiştir. Bu yöntemde test edilen bir diğer parametre ise gMapping'in yayınladığı haritanın ne kadar sürede bir güncelleneceğidir. Yine belirli bir değerin üstündeki sürelerde güncellendiğinde tarama eşlemelerin güncellemelere eklenemediğinden haritanın bozulduğu gözlemlenmiştir. Minimum güncelleme süresine ise sistemin performansı karar vermektedir. Yani neticede güncelleme süresinin azalmasıyla daha düzgün haritaların çıktığı tespit edilmiştir.

Çalışma kapsamında incelenen diğer bir yöntem Augmented Monte Carlo lokalizasyonudur. Yine parçacık tabanlı çalışan bu lokalizasyon (konum kestirimi) algoritmasında parçacıkların arttırılmasıyla daha iyi sonuçlar elde edilmektedir. Ancak birbirine çok benzer ortamlarda gezildiğinde algoritmasının kısıtı gereği tam lokalize olunamamaktadır. İçerisinde hiçbir şey olmayan kare bir odada gezildiğinde, yöntemin 4 farklı yerde lokalize olduğu gözlemlenmektedir. Bu durum algoritmanın mantığı gereği pek olasıdır. Bunun dışında birbirinden gözle ayırt edilebilecek ortamlarda konum kestiriminin başarılı sonuçlar ürettiği ve navigasyon paketi için kullanılmasının uygun olduğu sonucuna varılmıştır.

Statik harita üzerinde lazer mesafe ölçüm duyargası da kullanılarak ROS üzerinde gerçeklenmiş "navigation stack" paketi kullanılmıştır. Paketin ihtiyaç duyduğu konum ve duruş bilgisi AMCL tarafından üretilmiş ve navigasyonun başarılı bir şekilde gerçekleştirildiği görülmüştür. Özellikle lokal maliyet haritalarıyla birlikte engelden sakınmanın başarıyla hayata geçirildiği, robotun navigasyonu sırasında aniden önüne geçildiğinde yönünü değiştirebildiği gözlemlenmiştir.

Sonuç olarak, ROS Hydro sürümü üzerinde gerçeklenen paketlerin bir kısmı tasarlanan robot platformu üzerinde denenmiştir. Bütün metotlar parametrik olduğu için, metotların doğru çalışmalarıyla ilgili kesin bir yapılandırmanın olmadığı, ortama göre ayarlanması gereken parametrelerin olduğu gözlemlenmiştir. ROS framework'ünün robot platformlarının tasarlanmasında oldukça iyi bir çatı olduğu ve iş yükünü azalttığı tespit edilmiştir. Bir arama kurtarma amaçlı çalışacak robot platformu geliştirmek için gereksinimlerin neler olduğu ve bu gereksinimlerin ne şekilde karşılanacağı, daha iyi

sonuçlar elde edilmesi için takip edilecek yöntemlerin neler olduğu bu çalışma hayata geçirilirken tecrübeyle kavranmış ve konuyla ilgili literatürel bilgiler edinilerek bir konuda ileri düzey bilgi sahibi olunmuştur.

KAYNAKLAR

[1] Ganganath, N. ve Leung, H., (2012). "Mobile robot localization using odometry and kinect sensor".

[2] Censi, A., (2008). "An ICP variant using a point-to-line metric".

[3] Shu, L. Xu, H. ve Huang, M., (2013). "High-speed and accurate laser scan matching using classified features".

[4] Borrmann, D. Elseberg, J. Lingemann, K. Nüchter, A. ve Hertzberg, J., (2008). "Globally consistent 3D mapping with scan matching", Robotics and Autonomous Systems, 56: 130 - 142.

[5] Diosi, A. ve Kleeman, L., (2005). "Laser scan matching in polar coordinates with application to SLAM".

[6] Grisetti, G. Stachniss, C. ve Burgard, W., (2007). "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters", Robotics, IEEE Transactions on, 23: 34-46.

[7] Grisetti, G. Stachniss, C. ve Burgard, W., (2005). "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling".

[8] Kohlbrecher, S. Von Stryk, O. Meyer, J. ve Klingauf, U., (2011). "A flexible and scalable SLAM system with full 3D motion estimation".

[9] Esenkanova, J. İlhan, H.O. ve Yavuz, S., (2013). "Pre-Mapping System with Single Laser Sensor Based on Gmapping Algorithm", International Journal of Electrical Energy, 1: 97-101.

[10] Dryanovski, I. Valenti, R.G. ve Xiao, J., (2013). "Fast visual odometry and mapping from RGB-D data".

[11] Nowicki, M. ve Skrzypezynski, P., (2013). "Combining photometric and depth data for lightweight and robust visual odometry".

[12] Huang, A.S. Bachrach, A. Henry, P. Krainin, M. Maturana, D. Fox, D. ve Roy, N., (2011). Visual odometry and mapping for autonomous fight using an RGB-D camera.

[13] Burgard, W. Cremers, A.B. Fox, D. Hähnel, D. Lakemeyer, G. Schulz, D. Steiner, W. ve Thrun, S., (1999). "Experiences with an interactive museum tour-guide robot", Artificial Intelligence, 114: 3 - 55.

[14] Schiele, B. ve Crowley, J.L., (1994). "A comparison of position estimation techniques using occupancy grids".

[15] Weiss, G. Wetzler, C. ve Von Puttkamer, E., (1994). "Keeping track of position and orientation of moving indoor systems by correlation of range-finder scans".

[16] Roumeliotis, S.I. ve Bekey, G.A., (2000). "Bayesian estimation and Kalman filtering: a unified framework for mobile robot localization".

[17] Jensfelt, P. ve Kristensen, S., (2001). "Active global localization for a mobile robot using multiple hypothesis tracking", Robotics and Automation, IEEE Transactions on, 17: 748-760.

[18] Thrun, S. Fox, D. Burgard, W. ve Dellaert, F., (2001). "Robust Monte Carlo localization for mobile robots", Artificial Intelligence, 128: 99 - 141.

[19] Fox, D. Burgard, W. ve Thrun, S., (1999). "Markov localization for mobile robots in dynamic environments", Journal of Artificial Intelligence Research, 11: 391-427.

[20] Eliazar, A.I. ve Parr, R., (2003). "DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks".

[21] Dissanayake, G. Durrant-Whyte, H. ve Bailey, T., (2000). "A computationally efficient solution to the simultaneous localisation and map building (SLAM) problem".

[22] Thrun, S. ve Leonard, J., (2008). Simultaneous Localization and Mapping, B. Siciliano ve Khatib, O., ed. Springer Handbook of Robotics. Springer Berlin Heidelberg, 871-889.

[23] Montemerlo, M. Thrun, S. Koller, D. ve Wegbreit, B., (2002). "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem".

[24] Huang, S. ve Dissanayake, G., (2007). "Convergence and Consistency Analysis for Extended Kalman Filter Based SLAM", Robotics, IEEE Transactions on, 23: 1036-1049.

[25] Bailey, T. Nieto, J. Guivant, J. Stevens, M. ve Nebot, E., (2006). "Consistency of the EKF-SLAM Algorithm".

[26] Beinhofer, M. Müller, J. ve Burgard, W., (2013). "Effective landmark placement for accurate and reliable mobile robot navigation", Robotics and Autonomous Systems, 61: 1060 - 1069.

[27] Thrun, S., (1998). "Learning metric-topological maps for indoor mobile robot navigation", Artificial Intelligence, 99: 21 - 71.

[28] Conrad, D. ve DeSouza, G.N., (2010). "Homography-based ground plane detection for mobile robot navigation using a Modified EM algorithm".

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

[30] Andre, T. ve Bettstetter, C., (2013). "Assessing the value of coordination in mobile robot exploration using a discrete-time Markov process".

[31] Couceiro, M.S. Rocha, R.P. ve Ferreira, N.M.F., (2011). "A novel multi-robot exploration approach based on Particle Swarm Optimization algorithms".

[32] Andriluka, M. Schnitzspan, P. Meyer, J. Kohlbrecher, S. Petersen, K. Von Stryk, O. Roth, S. ve Schiele, B., (2010). "Vision based victim detection from unmanned aerial vehicles".

[33] Kleiner, A. ve Kummerle, R., (2007). "Genetic MRF model optimization for real-time victim detection in search and rescue".

[34] Carballo, A. Ohya, A. ve Yuta, S., (2009). "Multiple People Detection from a Mobile Robot using Double Layered Laser Range Finders".

[35] Fod, A. Howard, A. ve Mataric, M., (2002). Laser-Based People Tracking, 3024-3029.

[36] Pham, Q.-C. Gond, L. Begard, J. Allezard, N. ve Sayd, P., (2007). "Real-Time Posture Analysis in a Crowd using Thermal Imaging".

[37] Quigley, M., Gerkey B. Conley K. Faust J. Foote T. Leibs J. Berger E. Wheeler R. ve Ng A., (2009). ROS: an open-source Robot Operating System.

[38] The Robot Operating System, ROS Tanıtımı, www.ros.org, 5 Aralık 2013.

[39] Willow Garage, PR2 Robot Platformu Tanıtımı, www.willowgarage.com/pages/pr2/overview, 12 Aralık 2013.

[40] TurtleBot, TurtleBot Tanıtımı, www.turtlebot.com, 12 Aralık 2013.

[41] MobileRobots Pioneer 3-AT (P3AT) research robot platform, P3AT Tanıtımı, www.mobilerobots.com/ResearchRobots/P3AT.aspx, 5 Ocak 2014.

[42] Wikipedia, RoboCup Tanıtımı, en.wikipedia.org/wiki/RoboCup, 10 Ocak 2014.

[43] Sheh, R. Jacoff, A. Virts, A. Kimura, T. Pellenz, J. Schwertfeger, S. ve Suthakorn, J., (2012). "Advancing the State of Urban Search and Rescue Robotics through the RoboCupRescue Robot League Competition", Japan.

[44] Tandy, M.J. Winkvist, S. ve Young, K., (2010). "Competing in the RoboCup Rescue Robot League".

[45] Arduino Türkiye, Arduino Mikro Denetleyicisi Tanıtımı, www.arduinoturkiye.com/arduino-nedir-ve-ne-degildir, 15 Ocak 2014.

[46] Martinez, A. ve Fernandez, E. (2013). Learning ROS for Robotics Programming, Livery Place 35 Livery Street Birmingham B3 2PB, UK: Packt Publishing Ltd.

[47] ROS Wiki, ROS Topic'lerinin Anlatımı, wiki.ros.org/ROS/Tutorials/UnderstandingTopics, 15 Ocak 2014.

[48] ROS Dökümanları, /scan Mesajının Özellikleri, docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html, 15 Ocak 2014.

[49] ROS Dökümanları, /cmd_vel Mesajının Özellikleri, docs.ros.org/api/geometry_msgs/html/msg/Twist.html, 20 Şubat 2014.

[50] ROS Dökümanları, /map Mesajının Özellikleri, docs.ros.org/api/nav_msgs/html/msg/OccupancyGrid.html, 20 Şubat 2014..

[51] ROS Dökümanları, /tf Mesajının Özellikleri, docs.ros.org/api/tf/html/msg/tfMessage.html, 1 Mart 2014.

[52] ROS Dökümanları, /odom Mesajının Özellikleri, docs.ros.org/api/nav_msgs/html/msg/Odometry.html, 1 Mart 2014.

[53] ROS Dökümanları, /pose2D Mesajının Özellikleri, docs.ros.org/api/geometry_msgs/html/msg/Pose2D.html, 10 Mart 2014.

[54] ROS Dökümanları, /visualization_marker Mesajının Özellikleri, docs.ros.org/api/visualization_msgs/html/msg/Marker.html, 15 Mart 2014.

[55] ROS Dökümanları, /robot_pose_ekf/odom_combined Mesajının Özellikleri, docs.ros.org/api/geometry_msgs/html/msg/PoseWithCovarianceStamped.html 15 Mart 2014.

[56] ROS Dökümanları, /imu/data Mesajının Özellikleri, docs.ros.org/api/sensor_msgs/html/msg/Imu.html, 5 Nisan 2014.

[57] OpenSLAM, gMapping Tanıtımı, openslam.org/gmapping.html, 20 Nisan 2014.

[58] Murphy, K., (1999). Bayesian map learning in dynamic environments, 1015-1021.

[59] Çakmak, F., Uslu, E., Yavuz, S. Amasyalı, M.F. Balcılar, M. ve Altuntaş, N., (2014). Using Range and Inertia Sensors for Trajectory and Pose Estimation.

[60] Borenstein, J. Everett, H. R. Feng, L. ve Wehe, D. (1996). Navigating Mobile Robots: Sensors and Techniques: A. K. Peters, Ltd., Wellesley, MA.

[61] Burgard, W. Derr, A. Fox, D. ve Cremers, A.B., (1998). "Integrating global position estimation and position tracking for mobile robots: the dynamic Markov localization approach".

[62] Thrun, S. Wolfram, B. ve Dieter, F., (2005). Probabilistic Robotics: The MIT Press.

[63] ROS Wiki, Navigation Stack İçin Robot Ortamının Hazırlanmasının Tanıtımı, wiki.ros.org/navigation/Tutorials/RobotSetup, 25 Mayıs 2014.

EK-A

JOYSTICK TUŞ TAKIMI

Tez çalışması kapsamında kullanılan Logitech F710 kablosuz joystick'in analog çalışan 6 adet axes butonu, 12 adet dijital butonu vardır. Her bir butona basıldığına /joy mesajı içerisindeki ilgili alanların (axes ve buttons) değiştiği gözlemlenmiştir. Bunların kullanımları ile ilgili bilgiler aşağıdaki tabloda verildiği gibidir.

Axes[0]	Sol analog kolun sağa-sola hareket ettirilmesi sonucunda değişime uğramaktadır. Sola hareket ettirildiğinde negatif değerler, sağa hareket ettirildiğinde pozitif değerler aldığı gözlemlenmiştir.
Axes[1]	Sol analog kolun aşağı-yukarı hareket ettirilmesi sonucunda değişime uğramaktadır. Yukarı hareket ettirildiğinde negatif değerler, aşağı hareket ettirildiğinde pozitif değerler aldığı gözlemlenmiştir.
Axes[2]	Sağ analog kolun sağa-sola hareket ettirilmesi sonucunda değişime uğramaktadır. Sola hareket ettirildiğinde negatif değerler, sağa hareket ettirildiğinde pozitif değerler aldığı gözlemlenmiştir.
Axes[3]	Sol analog kolun aşağı-yukarı hareket ettirilmesi sonucunda değişime uğramaktadır. Yukarı hareket ettirildiğinde negatif değerler, aşağı hareket ettirildiğinde pozitif değerler aldığı gözlemlenmiştir.
Axes[4]	Joystick'in sol üst tarafında bulunan yön tuşlarının sağa-sola hareket ettirilmesiyle değişime uğramaktadır. Sola hareket ettirildiğinde negatif değer, sağa hareket ettirildiğinde pozitif değer aldığı gözlemlenmiştir.
Axes[5]	Joystick'in sol üst tarafında bulunan yön tuşlarının yukarı-aşağı hareket ettirilmesiyle değişime uğramaktadır. Yukarı hareket ettirildiğinde negatif değer, aşağı hareket ettirildiğinde pozitif değer aldığı gözlemlenmiştir.
Button[0]	Joystick üzerinde bulunan mavi renkli "X" tuşuna basıldığında değiştiği

	gözlemlenmiştir.
Button[1]	Joystick üzerinde bulunan yeşil renkli “A” tuşuna basıldığında değiştiği gözlemlenmiştir.
Button[2]	Joystick üzerinde bulunan kırmızı renkli “B” tuşuna basıldığında değiştiği gözlemlenmiştir.
Button[3]	Joystick üzerinde bulunan sarı renkli “Y” tuşuna basıldığında değiştiği gözlemlenmiştir.
Button[4]	Joystick'in arka kısmının sol üst tarafında bulunan “LB” tuşuna basıldığında değiştiği gözlemlenmiştir.
Button[5]	Joystick'in arka kısmının sağ üst tarafında bulunan “RB” tuşuna basıldığında değiştiği gözlemlenmiştir.
Button[6]:	Joystick'in arka kısmının sol alt tarafında bulunan “LT” tuşuna basıldığında değiştiği gözlemlenmiştir.
Button[7]	Joystick'in arka kısmının sağ alt tarafında bulunan “RT” tuşuna basıldığında değiştiği gözlemlenmiştir.
Button[8]	Joystick'in analog kollarının üst kısmında bulunan “BACK” tuşuna basıldığında değiştiği gözlemlenmiştir.
Button[9]	Joystick'in analog kollarının üst kısmında bulunan “START” tuşuna basıldığında değiştiği gözlemlenmiştir.
Button[10]	Joystick'in sol analog kolluna basıldığında değiştiği gözlemlenmiştir.
Button[11]	Joystick'in sağ analog kolluna basıldığında değiştiği gözlemlenmiştir.

EK-B

LAUNCH DOSYALARI

Bu başlık altında deneylerin yapılması sırasında kullanılan launch dosyalarına yer verilmiştir.

B-1 LSM Yönteminde Kullanılan Launch Dosyası

```
<!-- lsm.launch -->
<launch>
    <param name="/use_sim_time" value="true"/>
    <node pkg="tf" type="static_transform_publisher" name="tf1"
        args="0.0 0.0 0.0 0.0 0.0 0 /base_link /laser 100" />
    <node pkg="tf" type="static_transform_publisher" name="tf2"
        args="0.0 0.0 0.00 0.0 0.0 0 /world /my_frame3 100" />
    <node pkg="laser_scan_matcher" type="laser_scan_matcher_node"
        name="laser_scan_matcher_node" output="screen">
        <param name="max_iterations" value="10"/>
        <param name="use_imu" type="bool" value="true" />
        <param name="use_odom" type="bool" value="false" />
        <param name="fixed_frame" type="string" value="world" />
        <param name="base_frame" type="string" value="base_link" />
    </node>
    <node pkg="trajectory" type="draw_laser_path" name="laser_path"/>
    <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
        robot_joy)/launch/rviz/laser_scan_matcher_path.rviz"/>
</launch>
```

B-2 RPE ve gMapping Yönteminde Kullanılan Launch Dosyaları

```
<!-- gMapping_without_rpe.launch -->
<launch>
    <param name="/use_sim_time" value="true"/>
    <node pkg="tf" type="static_transform_publisher" name="tf2"
        args="0.0 0.0 0.0 0.0 0.0 0.0 /odom /base_link 100" />
    <node pkg="tf" type="static_transform_publisher" name="tf1"
        args="0.0 0.0 0.0 0.0 0.0 0.0 /base_link /laser 100" />
    <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping"
        output="screen">
        <param name="map_udpate_interval" value="2.0"/>
```

```
        <param name="maxUrange" value="16.0"/>
        <param name="sigma" value="0.05"/>
        <param name="kernelSize" value="1"/>
        <param name="lstep" value="0.05"/>
        <param name="astep" value="0.05"/>
        <param name="iterations" value="5"/>
        <param name="lsigma" value="0.075"/>
        <param name="ogain" value="3.0"/>
        <param name="lskip" value="0"/>
        <param name="srr" value="0.1"/>
        <param name="srt" value="0.2"/>
        <param name="str" value="0.1"/>
        <param name="stt" value="0.2"/>
        <param name="linearUpdate" value="1.0"/>
        <param name="angularUpdate" value="0.5"/>
        <param name="temporalUpdate" value="3.0"/>
        <param name="particles" value="10"/>
        <param name="xmin" value="-20.0"/>
        <param name="ymin" value="-20.0"/>
        <param name="xmax" value="20.0"/>
        <param name="ymax" value="20.0"/>
        <param name="delta" value="0.015"/>
        <param name="llsamplerange" value="0.01"/>
        <param name="llsamplestep" value="0.01"/>
        <param name="lasamplerange" value="0.005"/>
        <param name="lasamplestep" value="0.005"/>
    </node>
    <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
        robot_joy)/launch/rviz/gmapping_from_bag.rviz"/>
</launch>


<!—gMapping_with_rpe.launch -->
<launch>
    <param name="/use_sim_time" value="true"/>
    <node pkg="laser_scan_matcher" type="laser_scan_matcher_node"
        name="lsm_node" output="screen">
        <param name="use_imu" type="bool" value="false" />
        <param name="use_odom" type="bool" value="false" />
        <param name="laser_topic" type="string" value="scan" />
        <param name="publish_pose_stamped" type="bool" value="true" />
    </node>
    <!-- lsm launcher -->
    <node pkg="tf" type="static_transform_publisher" name="tf1"
        args="0.0 0.0 0.0 0.0 0.0 0 /base_link /laser 100" />
    <!-- lsm'nin yayınladığı /pose2D topic'ini kullanarak /odom
        topic'i publish eder  -->
    <node pkg="robot_joy" type="pose2d_to_odom" name="odom_lsm"/>
    <!-- robot_pose_ekf launcher -->
    <node pkg="robot_pose_ekf" type="robot_pose_ekf"
        name="robot_pose_ekf">
        <param name="odom_used" value="true"/>
        <param name="imu_used" value="false"/>
    </node>
    <!-- hokuyo_node'un yayınladığı /scan topic'ini /scan2 diye
        çoklar. /scan2'nin frame'ini laser2 yapar. -->
    <node pkg="laser_remapper" type="remapper2" name="hokuyo_remapper"
        />
    <node pkg="tf" type="static_transform_publisher" name="tf2"
        args="0.0 0.0 0.0 0.0 0.0 0 /base_footprint /laser2 100" />
```

```
    <!-- gmapping launcher -->
    <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping"
        output="screen">
        <param name="scan_topic" type="string" value="scan2" />
        <param name="odom_frame" type="string"value=" odom_combined"/>
        <param name="map_udpate_interval" value="5.0"/>
        <param name="maxUrange" value="29.5"/>
        <param name="sigma" value="0.05"/>
        <param name="kernelSize" value="1"/>
        <param name="lstep" value="0.05"/>
        <param name="astep" value="0.05"/>
        <param name="iterations" value="5"/>
        <param name="lsigma" value="0.075"/>
        <param name="ogain" value="3.0"/>
        <param name="lskip" value="0"/>
        <param name="srr" value="0.1"/>
        <param name="srt" value="0.2"/>
        <param name="str" value="0.1"/>
        <param name="stt" value="0.2"/>
        <param name="linearUpdate" value="1.0"/>
        <param name="angularUpdate" value="0.5"/>
        <param name="temporalUpdate" value="3.0"/>
        <param name="particles" value="30"/>
        <param name="xmin" value="-20.0"/>
        <param name="ymin" value="-20.0"/>
        <param name="xmax" value="20.0"/>
        <param name="ymax" value="20.0"/>
        <param name="delta" value="0.02"/>
        <param name="llsamplerange" value="0.01"/>
        <param name="llsamplestep" value="0.01"/>
        <param name="lasamplerange" value="0.005"/>
        <param name="lasamplestep" value="0.005"/>
    </node>
    <node pkg="tf" type="static_transform_publisher" name="tf3"
        args="0.0 0.0 0.0 0.0 0.0 0 /map /world 100" />
    <!-- lsm trajectory launcher , duzeltilecek !!! -->
    <node pkg="trajectory" type="draw_lsm_imu_r_p_ekf"
        name="gezinge"/>
    <node pkg="tf" type="static_transform_publisher" name="tf5"
        args="0.0 0.0 0.0 0.0 0.0 0 /map /my_frame 100" />
    <!-- rviz launcher -->
    <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
        robot_joy)/launch/rviz/lsm_imu_robot_pose_ekf.rviz"/>
</launch>
```

B-3 AMCL Yönteminde Kullanılan Launch Dosyaları

```
<!-- amcl.launch -->
<launch>
    <node pkg="robot_joy" type="robot_teleop" name="teleop"/>
    <node respawn="true" pkg="joy" type="joy_node" name="robotjoy" >
        <param name="dev" type="string" value="/dev/input/js1" />
        <param name="deadzone" value="0.01" />
    </node>
    <node pkg="rosserial_python" type="serial_node_ard_mega_2.py"
        name="serial_motor"/>
    <node pkg="hokuyo_node" type="hokuyo_node" name="hokuyo_node"
        output="screen">
        <param name="frame_id" value="/laser"/>
```

```
        <param name="port" value="/dev/sensors/hokuyo_H1304463" />
    </node>
    <node name="map_server" pkg="map_server" type="map_server"
        args="$(find robot_joy)/launch/map_for_amcl/amcl_map.yaml" />
    <node pkg="laser_scan_matcher" type="laser_scan_matcher_node"
        name="lsm_node" output="screen">
        <param name="use_imu" type="bool" value="false" />
        <param name="use_odom" type="bool" value="false" />
        <param name="publish_pose_stamped" type="bool" value="true" />
        <param name="laser_topic" type="string" value="scan" />
    </node>
    <node pkg="tf" type="static_transform_publisher" name="tf1"
        args="0.0 0.0 0.73 0.0 0.0 0 /base_link /laser 100" />
    <node pkg="robot_joy" type="pose2d_to_odom" name="odom_lsm"/>
    <node pkg="robot_pose_ekf" type="robot_pose_ekf"
        name="robot_pose_ekf">
        <param name="imu_used" value="false"/>
    </node>
    <node pkg="laser_remapper" type="remapper2"
        name="hokuyo_remapper"/>
    <node pkg="tf" type="static_transform_publisher" name="tf2"
        args="0.0 0.0 0.73 0.0 0.0 0 /base_footprint /laser2 100" />
    <node pkg="tf" type="static_transform_publisher" name="tf3"
        args="0.0 0.0 0.0 0.0 0.0 0 /map /world 100" />
    <node pkg="trajectory" type="draw_lsm_imu_r_p_ekf"
        name="gezinge"/>
    <node pkg="tf" type="static_transform_publisher" name="tf5"
        args="0.0 0.0 0.0 0.0 0.0 0 /map /my_frame 100" />
    <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
        robot_joy)/launch/rviz/amcl.rviz"/>
    <include file="$(find
        robot_joy)/launch/map_for_amcl/amcl.launch.xml"/>
</launch>
```

B-4 Navigation Yönteminde Kullanılan Launch ve Yapılandırma Dosyaları

```
<!—move_base.launch -->
<launch>
    <node pkg="move_base" type="move_base" respawn="false"
        name="move_base_node" output="screen">
        <param name="footprint_padding" value="0.1" />
        <param name="controller_frequency" value="2.0" />
        <param name="controller_patience" value="3.0" />
        <param name="oscillation_timeout" value="30.0" />
        <param name="oscillation_distance" value="1.0" />
        <rosparam file="$(find
            robot_joy)/launch/map_for_amcl/costmap_common_params.yaml"
            command="load" ns="global_costmap" />
        <rosparam file="$(find
            robot_joy)/launch/map_for_amcl/costmap_common_params.yaml"
            command="load" ns="local_costmap" />
        <rosparam file="$(find robot_joy)/launch/map_for_amcl/
            global_costmap_params.yaml" command="load" />
        <rosparam file="$(find robot_joy)/launch/map_for_amcl/
            local_costmap_params.yaml" command="load" />
        <rosparam file="$(find robot_joy)/launch/map_for_amcl/
            base_local_planner_params.yaml" command="load" />
    </node>
</launch>
```

```
# base_local_planner.yaml
TrajectoryPlannerROS:
    acc_lim_th: 0.08
    acc_lim_x: 0.05
    acc_lim_y: 0.05
    max_vel_x: 0.1
    min_vel_x: 0.05
    max_vel_theta: 0.15
    min_vel_theta: -0.15
    min_in_place_vel_theta: 0.15
    escape_vel: -0.1
    holonomic_robot: false
    y_vels: [-0.3, -0.1, 0.1, -0.3]
    xy_goal_tolerance: 0.1
    yaw_goal_tolerance: 0.2
    sim_time: 1.7
    sim_granularity: 0.025
    vx_samples: 3
    vtheta_samples: 20
    goal_distance_bias: 0.8
    path_distance_bias: 0.6
    occdist_scale: 0.01
    heading_lookahead: 0.325
    dwa: false
    oscillation_reset_dist: 0.1
    prune_plan: true



#costmap_common_params.yaml
map_type: voxel
    origin_z: 0.0
    z_resolution: 0.2
    z_voxels: 10
    unknown_threshold: 9
    mark_threshold: 0
    transform_tolerance: 0.3
    obstacle_range: 15.0
    max_obstacle_height: 2.0
    raytrace_range: 20.0
    footprint: [[-0.31, -0.28], [-0.31, 0.28], [0.31, 0.28], [0.31, -
        0.28]]
    footprint_padding: 0.02
    inflation_radius: 0.30
    cost_scaling_factor: 5.0
    lethal_cost_threshold: 100
    base_scan: {sensor_frame: laser2, data_type: LaserScan, topic:
        scan2, expected_update_rate: 0.4,
    observation_persistence: 0.0, marking: true, clearing: true}



#global_costmap_params.yaml
global_costmap:
    global_frame: /map
    robot_base_frame: base_footprint
    update_frequency: 2.0
    publish_frequency: 0.0
```

```
    static_map: true
    rolling_window: false
    footprint_padding: 0.02



#local_costmap_params.yaml
local_costmap:
    publish_voxel_map: true
    global_frame: odom_combined
    robot_base_frame: base_footprint
    update_frequency: 2.0
    publish_frequency: 2.0
    static_map: false
    rolling_window: true
    width: 20.0
    height: 20.0
    resolution: 0.025
    origin_x: 0.0
    origin_y: 0.0
plugins: {name: static_map, type: "costmap_2d::StaticLayer"}
plugins: {name: obstacles,  type: "costmap_2d::VoxelLayer"}
plugins: {name: inflater,   type: "costmap_2d::InflationLayer"}
```

EK-C

ROS ÜZERİNDE ÇALIŞTIRILAN PAKETLERİN PARAMETRELERİ

Bu bölümde, ROS üzerinde gerçeklenen LSM ve gMapping parametre bilgilerine ve varsayılan değerlerine yer verilmiştir.

C -1 gMapping Parametreleri

1. throttle_scans: Varsayılan olarak 1 değerini alır. Eğer lazerden gelen her scan işleme tabi tutulmak istenmiyorsa bu değer istenilen ölçüde büyütülür. Mesela 3 yapılırsa; 3 scan'de bir scan işleme tabi tutulur.

2. map_update_interval: Varsayılan olarak saniye cinsinden 5.0 değerini alır. "gmapping" tarafından oluşturulan haritanın kaç saniye de bir güncelleneceği bu parametre ile ayarlanır. Bilgisayarın işlem kapasitesine göre değişmekle birlikte, bu süre azaltıldığında bilgisayarın kaynaklarının birim zamanda daha fazla kullanılacağına dikkat edilmelidir.

3. maxUrange: Varsayılan olarak metre cinsinden 80.0 değerini alır. Lazerin kullanılabilen maksimum mesafesini bu parametre üzerinden ayarlanabilir.

4. sigma: Varsayılan olarak 0.05 değerini alır. Bitiş noktasının eşleştirilmesi için kullanılan bir eşik değerdir.

5. kernelSize: Varsayılan olarak 1 değerini alır. Haberleşme/mesajlaşma arayan çekirdeğin büyüklüğünü ayarlar.

6. lstep: Varsayılan olarak 0.05 değerini alır. "Translation" adımının optimize edilmesi için kullanılır.

7. astep: Varsayılan olarak 0.05 değerini alır. "Rotation" adımının optimize edilmesi için kullanılır.

8. iterations: Varsayılan olarak 5 değerini alır. "Scan Match" yapılırken kaç maksimum iterasyon kullanılacağı bu parametre ile ayarlanır.

9. Lsigma: Varsayılan olarak 0.075 değerini alır. Olasılık hesaplaması sırasında karşılaşılan kirişlerin (beam) yok sayılmasının eşik değeridir.

10. ogain: Varsayılan olarak 3.0 değerini alır. Olasılıkların değerlendirilmesinde kullanılan kazancın parametresidir. Örnekleme etkisinin yumuşatılması için kullanılır.

11. lskip: Varsayılan olarak 0 değerini alır. Her bir scan'de atlanılan kirişlerin sayısının parametresidir.

12. linearUpdate: Varsayılan olarak 1.0 değerini alır. Robot verilen parametre kadar translate olduğunda scan'den alınan veriyi işlemek için kullanılır.

13. angularUpdate: Varsayılan olarak 0.5 değerini alır. Robot verilen parametre kadar rotation olduğunda scan'den alınan veriyi işlemek için kullanılır.

14. temporalUpdate: Varsayılan olarak -1.0 değerini alır. Saniyeler içerisindeki güncelleme zamanı son scan'in işlenmesinden daha büyükse bir scan daha işlemek için kullanılır. 0'dan küçük değerler bu güncelleştirmenin yapılmaması için kullanılır.

15. particles: Varsayılan olarak 30 değerini alır. Parçacık filtresi için kullanılacak parçacık sayısının parametresidir.

16. delta: Varsayılan olarak 0.05 değerini alır. Haritanın çözünürlüğü ile ilgili ilerleme parametresidir.

17. llsamplerange: Varsayılan olarak 0.01 değerini alır. Olasılık için öteleme (translational) örnekleme aralığının parametresidir.

18. llsamplestep: Varsayılan olarak 0.01 değerini alır. Olasılık için öteleme örnekleme adımının parametresidir.

19. lasamplerange: Varsayılan olarak 0.005 değerini alır. Olasılık için açısal (angular) örnekleme aralığının parametresidir.

20. lasamplestep: Varsayılan olarak 0.005 değerini alır. Olasılık için açısal (angular) örnekleme adımının parametresidir.

21. transform_publish_period: Varsayılan olarak saniye cinsinden 0.05 değerini alır. Dönüşümlerin yayınlanmasının ne kadar süreceğinin parametresidir.

22. occ_thresh: Varsayılan olarak 0.25 değerini alır. "gmapping" in işgal ettiği sürenin eşik değeridir. Doluluk oranı fazla olan hücreler meşgul (occupied) olarak kabul edilir.

23. maxRange: Duyarganın maksimum mesafesidir. Şu şekilde bir ayarlama yapılması uygundur: maxUrange < duyarganın gerçek maksimum mesafesi <= maxRange.

C -2 LSM Parametreleri

1. max_iterations: Varsayılan olarak 10 değerini alır. ICP'nin maksimum iterasyon sayısını belirtmek için kullanılır.

2. max_correspondence_dist: Varsayılan olarak 0.03 değerini alır. Eşlemelerin geçerli kabul edilebilmesi için aralarında olması gereken maksimum mesafeyi belirler.

3. max_angular_correction_deg: Varsayılan olarak 45 değerini alır. Taramalar arasında derece olarak maksimum yer değiştirmeyi belirler.

4. max_linear_correction: Varsayılan olarak 0.5 değerini alır. Taramalar arasında metre cinsinden maksimum translasyon değerini belirler.

5. epsilon_xy: Varsayılan olarak 0.000001 değerini alır. LSM'nin eşlemelerinin durması için metre cinsinden eşik değeri belirler.

6. epsilon_theta: Varsayılan olarak 0.000001 değerini alır. LSM'nin eşlemelerinin durması için radyan cinsinden açısal eşik değeri belirler.

7. outliers_maxPerc: Varsayılan olarak 0.90 değerini alır. Taramaların değerlendirilecek karşılık yüzdesini belirler.

8. clustering_threshold: Varsayılan olarak 0.25 değerini alır. Gelen tarama verilerinin aynı kümede değerlendirilmesi için kullanılan maksimum eşik değerdir.

9. orientation_neighbourhood: Varsayılan olarak 10 değerini alır. Oryantasyonu tahmin etmek için kullanılan komşuluk sayısıdır.

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı :Furkan ÇAKMAK

Doğum Tarihi ve Yeri :12.01.1989 Şefaatli/YOZGAT

Yabancı Dili :İngilizce

E-posta :furkan.cakmak@windowslive.com

ÖĞRENİM DURUMU

Derece	Alan	Okul/Üniversite	Mezuniyet Yılı
Lisans	Bilgisayar Mühendisliği	Yıldız Teknik Üniversitesi	2013
Lise	Matematik-Fen	Düzce Fen Lisesi	2007

İŞ TECRÜBESİ

Yıl	Firma/Kurum	Görevi
2013-devam	Yıldız Teknik Üniversitesi / Bilgisayar Mühendisliği	Araştırma Görevlisi

YAYINLARI

Bildiri

1. **Çakmak, F.; Uslu, E.; Yavuz, S.; Amasyalı, Balcılar, M; Altuntaş, N., " Using Range and Inertia Sensors for Trajectory and Pose Estimation," Signal Processing and Communications Applications Conference (SIU), 2014 22st , 22-25 April 2014**

Proje

1. **Otonom Arama Kurtarma Robotları Için Keşif, Haritalama ve Afetzede Tespit Algoritmalarının Geliştirilmesi. TÜBİTAK 1001 EEEAG 113E212. Bursiyer**

2. **Afet Ortamlarında Kullanılacak Otonom Robot Tasarımı. YTU KAP 2013-04-01-KAP02. Araştırmacı**