Gezgin Robotlarda Ultrasonik Mesafe Algılayıcılarla
Robot Davranışlarının Kontrolü ve Çevre Haritalama

Elif Eroğlu

YÜKSEK LİSANS TEZİ

Elektrik Elektronik Mühendisliği **Anabilim Dalı**

Haziran 2006

Robot Behavior Control and Environment Mapping with Ultrasonic Range Sensors in Mobile Robots

Elif Eroğlu

MASTER OF SCIENCE THESIS

Department of Electrical and Electronics Engineering

June 2006

GEZGİN ROBOTLARDA ULTRASONİK MESAFE ALGILAYICILARLA ROBOT DAVRANIŞLARININ KONTROLÜ VE ÇEVRE HARİTALAMA

Elif Eroğlu

Eskişehir Osmangazi Üniversitesi
Fen Bilimleri Enstitüsü
Lisansüstü Yönetmeliği Uyarınca
Elektrik Elektronik Mühendisliği Anabilim Dalı
Kontrol ve Kumanda Bilim Dalında
YÜKSEK LİSANS TEZİ
Olarak Hazırlanmıştır

Danışman: Doç.Dr. Osman Parlaktuna

Haziran 2006

Elif EROĞLU' nun YÜKSEK LİSANS tezi olarak hazırladığı "Gezgin Robotlarda Ultrasonik Mesafe Algılayıcılarla Robot Davranışlarının Kontrolü ve Çevre Haritalama " başlıklı bu çalışma, jürimizce lisansüstü yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

Üye : Doç. Dr. Osman PARLAKTUNA

Üye : Y. Doç. Dr. Rıfat EDİZKAN

Üye : Y. Doç. Dr. Selçuk CANBEK

Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ..... tarih ve ..... sayılı kararıyla onaylanmıştır.

Prof. Dr. Abdurrahman KARAMANCIOĞLU

Enstitü Müdürü

ÖZET

Bu çalışmada, duvar bulma, duvar takibi, öndeki ve yandaki engellerden kaçınma, içbükey ve dışbükey köşe dönüşleri, tamponların kontrolü, tekerleğin sıkışma durumundaki kontrolü gibi robotun yapması gereken temel davranışlar geliştirilmiştir. Geliştirilen davranış modeli PIONEER robotlar için tasarlanmış MobilSim simülatörü ve P3 dx robotu ile test edilmiştir.

Yapılan çalışmada, robotun çevre ile olan etkileşimi ultrasonik mesafe algılayıcıları ile sağlanmıştır. Ultrasonik mesafe algılayıcılar diğer algılayıcılara göre ucuz bir algılayıcı tipi olmasına rağmen karmaşık çevrelerde çalışırken bazı problemleri vardır.

Geliştirilen davranış modeli ile robotun hareketi sağlandıktan sonra ultrasonik algılayıcılardan ve kodlayıcıdan alınan verilerden faydalanılarak Bayes güncellemeli doluluk ızgaraları yöntemiyle çevre haritası oluşturulmuştur. Harita tespit etmekteki amacımız robotun ve etrafındaki cisimlerin konumlarını bilmek istememizdir. Robotun çevresi ne kadar doğru modellenirse ileriye yönelik planlama davranışları da o kadar başarılı olacaktır.

SUMMARY

In this study, fundamental behaviors, which has to be accomplished by a robot, are developed, such as wall finding, wall following, avoiding front and side obstacles, turning convex and concave corners, controlling bumpers and controlling stalls. The developed behavior model is simulated with MobilSim simulator which is designed for PIONEER robots, and tested with P3 dx robot.

In this application, range measurement operation is handled by ultrasonic range finders. Although these range finders are cheaper than other finders, they have some difficulties on working in complicated environments.

After making the robot move with the developed behavior model, Occupancy Grid technique with Bayesian update is used for generating the environment map by using range measurements and data obtained from encoders. Purpose of the map building is to know the location of the robot and objects around it. The more accurately environment map of the robot is modeled, the more successful planning behaviors for future will be.

TEŞEKKÜR

Bu çalışmada, bilgisi ve deneyimi ile bana yol gösteren, yönlendiren ve bu tezin oluşmasında büyük emeği ve katkıları olan değerli hocam ve danışmanım Doç. Dr. Osman PARLAKTUNA' ya çok teşekkür ederim. Ayrıca laboratuar çalışmalarımda her türlü yardımı sağlayan Araş.Grv. Metin ÖZKAN' a ve Dr. Ahmet YAZICI' ya, tez çalışmaları sırasında karşılaştığım her türlü sorunu çözmek için ortak uğraş verdiğimiz Araş.Grv. Uğur GÜREL' e, tüm bu çalışmalarım sırasında her zaman yanımda olan ve desteğini benden esirgemeyen sevgili aileme çok teşekkür ederim.

Bu tez Eskişehir Osmangazi Üniversitesi Bilimsel Araştırma Projeleri Komisyonu tarafından desteklenmiştir (Proje No: 200315030).

İÇİNDEKİLER

Sayfa

ÖZET …… iv

SUMMARY …… v

TEŞEKKÜR ……... vi

ŞEKİLLER DİZİNİ ……... x

ÇİZELGELER DİZİNİ ……. xiii

1 GİRİŞ …….. 1

2 GEZGİN ROBOTLARDA ALGILAYICILARLA HARİTA ÇIKARMA YÖNTEMLERİ ……. 5

2.1 Ultrasonik Mesafe Algılayıcılarla Harita Çıkarma Yöntemleri ….. 5

2.1.1 Moravec ve Elfes – 1985 Geniş açılı sonardan yüksek çözünürlüklü harita çıkarma ……. 5

2.1.2 Matthies ve Elfes – Izgara temelli gösterimi kullanarak sonar ve stereo mesafe verilerin birleşimi …… 10

2.1.2.1 Değişik algılayıcıların birleştirilmesi …… 10

2.1.2.2 Harita değerlerini Bayes olasılık teorisi ile güncelleme …… 11

2.1.3 Thrun -1993, Sinir ağları temelli yaklaşım ……. 13

2.1.4 Konelige -1997 Harita çıkarma için genişletilmiş doluluk ızgaraları metodu …… 13

2.2 Lazer Mesafe Algılayıcılarla Harita Çıkarma Yöntemleri ……. 14

2.3 Kamera ile Harita Çıkarma Yöntemleri ……. 18

3 DOLULUK IZGARALARI METODUNUN SONAR MODELİNE UYGULANMASI …….. 19

3.1 Sonarın Bölgeleri …….. 20

3.2 Güncelleme Yöntemleri …… 22

İÇİNDEKİLER (devam)

Sayfa

3.2.1 Bayes güncellemesi ……. 22

3.2.2 Dempster- Shafer teorisi ……. 24

3.2.2.1 Shafer fonksiyonu ……. 25

3.2.2.2 Dempster birleşme kuralı ……. 25

3.2.3 Hareket halinde histogramik harita çıkarma metodu …….. 29

4 ROBOT DAVRANIŞLARININ GELİŞTİRİLMESİ ……. 31

4.1 Duvar Bulma Davranışı ……. 31

4.2 Duvara Paralel Olma Davranışı (Duvar Takibi) …… 35

4.3 Öndeki Engelden Kaçınma (İçbükey Dönüşleri) Davranışı ……. 36

4.4 Yandaki Engelden Kaçınma Davranışı …….. 39

4.5 Dışbükey Köşe Dönüşleri ……. 39

4.6 Tampon Kontrolü …….. 41

4.7 Tekerleğin Sıkışma Durumunda Kontrolü ……. 41

4.8 Harita Oluşturma …… 41

4.8.1 Doluluk ızgara metodu ile olasılık hesaplanması ……... 41

4.8.2 Pusulanın kalibre edilmesi …….. 45

4.8.3 Verileri dosyaya yazdırma …….. 46

5 ROBOT SİSTEMİ …….. 47

5.1 Robotun Yapısı …….. 47

5.2 Kullanılan Yazılımlar ……. 48

5.2.1 ARIA 48

İÇİNDEKİLER (devam)

Sayfa

5.2.2 Simulatör ……. 50

6 ROBOT DAVRANIŞLARI UYGULAMA SONUÇLARI ……. 53

6.1 Duvar Bulma ve Duvar Takibi Davranışı Uygulamaları ……... 53

6.1.1 Simülatör testleri ……. 53

6.1.2 Gerçek ortamda yapılan testler …… 57

6.2 Harita Çıkarma Uygulamaları …… 62

6.2.1 Simülatör testleri ……. 62

6.2.2 Gerçek ortamda yapılan testler …… 64

7 SONUÇLAR ve ÖNERİLER …… 65

8 KAYNAK DİZİNİ ……. 66

EKLER

ŞEKİLLER DİZİNİ

Sayfa

2.1 Sonar Işının Modellenmesi…… 5

2.2 Sonar Işının Olasılık Dağılımı ……. 6

2.3 Çapraz Etkilenme (Min, et al., 1997) …….. 9

2.4 Bir Sonarın Yansımaları …… 10

2.5 Doluluk Izgaraları Temelli Algılayıcıların Birleşimi …….. 11

2.6 Robot Koordinat Sistemi ……... 14

2.7 Kinematik Parametreleri …….. 16

3.1 P3 DX robotun sonarlarının konumları ……... 20

3.2 Sonar Modelin Bölgelere Ayrılması …… 21

3.3 Bir Sonar Okuması Tarafından Gerçekleşen Olasılık Görüntüsü ……. 22

3.4 İki Varsayımın Sayı Çizgisinde Gösterilmesi ……... 26

3.5 Sayı Çizgisinin Birim Kareye Dönüştürülmüş Durumu ……... 26

3.6 Ağırlık Değerlerinin Birleştirilmesi ……. 27

3.7 Bel 3 Sonuç Fonksiyonu ……. 27

3.8 Normalleştirmeye Gerek Duyulan Örnek ……. 28

3.9 a. Sadece bir hücre mesafe okumada artış göstermektedir. Ultrasonik mesafe algılayıcısı ile ölçülen d mesafesinde akustik eksende hücre bulunmaktadır
b. Robot hareket halinde iken sürekli ve hızlı örneklenen algılayıcı değerleri için histogramik olasılık dağılımı elde edilir……... 30

4.1 P3 DX'in polar koordinatları …… 32

4.2 P3 DX'in -10º,+10º polar koordinatları ……... 33

4.3 P3 DX'in +1º den 0º' ye Kadar Polar Koordinatları …… 33

ŞEKİLLER DİZİNİ (devam)

Sayfa

4.4 P3 DX'in -100º den -80º' ye Kadar Polar Koordinatları ……. 34

4.5 Duvar Bulma Davranışı …… 35

4.6 Duvara Paralel Olma Davranışı …… 35

4.7 P3 DX'in -30º den +30º' ye Kadar Polar Koordinatları …… 36

4.8 Ön Engelden Kaçınma Davranışı …….. 37

4.9 Duvarın Robota Göre Durumu …….. 38

4.10 P3-DX'ın Polar Koordinatları ……. 40

4.11 90º ve 180º Dışbükey Köşe …… 40

4.12 Engel 7.Sonara Yakınsa Sonar Modeli ……... 42

4.13 Engel 8.Sonara Yakınsa Sonar Modeli ……... 42

4.14 Engel 7 ve 8.Sonara Eşit Mesafede ise Sonar Modeli …… 43

4.15 Sonarların Okuma Bölgeleri …… 44

4.16 Simülatörde Ölçülen Pusula Değerleri …….. 45

4.17 Robotta Ölçülen Pusula Değerleri …… 45

5.1 P3 DX in önden görüntüsü …….. 47

5.2 ARIA' nın Çalışma Prensibi …… 49

5.3 MobileSim programının görüntüsü ……... 50

5.4 SRIsim programının görüntüsü 51

5.5 Mapper programının görüntüsü 51

5.6 Mapper3Basic programının görüntüsü …… 52

6.1 Simülatörde Oluşturulan Dünya 1 …… 53

ŞEKİLLER DİZİNİ (devam)

Sayfa

6.2 Duvar Takibi Davranışı Sonrası Oluşan Harita …… 54

6.3 Simülatörde Oluşturulan Dünya 2 ……. 54

6.4 Duvar Bulma ve Duvar Takibi Davranışı Sonrası Oluşan Harita ……. 55

6.5 Simülatörde Oluşturulan Dünya 3 …… 55

6.6 Duvar Bulma, Duvar Takibi ve Dışbükey Köşe Dönüşleri Davranışı Sonrası Oluşan Harita …….. 56

6.7 Duvar Takibi ve Dışbükey Köşe Dönüşleri Davranışı Sonrası Oluşan Harita …….. 56

6.8 Robot Başlangıç Konumunda ……... 57

6.9 Robotun En Yakın Duvarı Aramaya Başladığı Durum …… 57

6.10 Robotun En Yakın Duvarı Bulduğu ve O Yöne Doğru Gitmeye Başladığı Durum …… 58

6.11 Robotun Duvarı Bulup Kendini Duvara Paralel Hale Getirdiği Durum….. 58

6.12 Robotun Duvara Paralel Halde Duvar Takip Ettiği Durum……. 59

6.13 Robotun 90 Derecelik Köşeye Yaklaştığı Durum …….. 59

6.14 Robotun 90 Derecelik Köşeyi Dönmeye Başladığı Durum …… 60

6.15 Robotun 90 Derecelik Köşeyi Dönmeyi Tamamladığı Durum …….. 60

6.16 Robotun 180 Derecelik Dışbükey Köşeye Yaklaştığı Durum …… 61

6.17 Robotun 180 Derecelik Dışbükey Köşeyi Dönmeye Başladığı Durum ….. 61

6.18 Robotun 180 Derecelik Köşeyi Dönmeyi Tamamladığı Durum …… 62

6.19 Simülatör Ortamında Oluşturulan Dünya ……... 63

6.20 Duvar Takibi Davranışı Sonunda Oluşturulan Harita …… 63

6.21 Laboratuar Ortamında Oluşturulan Harita …….. 64

ÇİZELGELER DİZİNİ

Şekil **Sayfa**

3.1 Sonarların Robot Üzerindeki Konum ve Yönlenmeleri ……19

1.GİRİŞ

Endüstriyel otomasyon sistemlerinde karşılaşılan ve üzerinde çalışılan temel problemler düşünüldüğü zaman ilk akla gelen konu, kontrol problemleridir. Kontrol problemleri alt başlıklara bölündüğü zaman, birinci sıraya oturan konu ise konum ve cisim algılama başlığıdır.

Konum ve cisim algılama sorunu için çözüm arayışına girildiği zaman ise dünya üzerinde söz konusu soruna en iyi çözümün, doğa tarafından getirildiği görülmektedir. Bu çözümün en iyi uygulayıcıları ise kuşkusuz yarasalardır. Yarasalar sadece karanlık ve aydınlığı algılayabilecek bir göz yapısına sahiptirler ve yaşamlarını gece avlanarak sürdürürler. Buna rağmen sahip oldukları karmaşık ultrasonik algılama sistemi sayesinde, karanlık bir odanın zeminindeki küçücük bir tırtılı bile algılar ve avlarlar (www.bilmuh.gyte.edu.tr).

Konum ve cisim algılama problemine, doğanın bulduğu çözüm kadar geçerli ve etkin olmasa da birçok çözüm üretilebilmiştir. Söz konusu soruna cevap olarak üretilen endüstriyel çözümler şu şekilde sıralanabilir:

1. Kızıl Ötesi (Optik) algılayıcılar
2. Ultrasonik algılayıcılar
3. Lazer algılayıcılar

Optik algılayıcılar, endüstriyel uygulamalar içerisinde özellikle cisim algılama için sıkça kullanılmaktadır. Bu tip algılayıcılar temel olarak kaynaktan gönderilen belirli bir frekansa sahip ışığın, yansıtıcı aynadan geri yansıtılarak alıcı tarafından algılanmasına dayanmaktadır. Algılayıcı özellikle cisim algılama yönünden etkin ve ucuz bir çözüm üretse de birçok dezavantaja sahiptir. Özellikle mesafe bu tip algılayıcılar için çok kritik bir parametre konumundadır. Mesafe arttıkça, kaynak tarafından yollanan ışının dağılımı ve geri yansıyamaması algılayıcı için önemli bir problemdir. Bu sebepten dolayı hatalı sonuçlar ortaya çıkabilmektedir.

Bunun dışında özellikle kirli ve/veya parçacıklı ortamlarda bu tip algılayıcıların yansıtıcıları sıklıkla işlev yapamaz hale gelebilmektedir. Bu tip algılayıcılar ile yapılan mesafe algılama uygulamalarında gözlenen diğer bir olumsuz etki ise yansıtıcı yüzeyin rengidir. Özellikle koyu renkli nesneler üzerinde yapılan çalışmalarda, cismin algılayıcı tarafından ya hiç algılanmadığı ya da çok geç algılandığı gözlenmiştir. Bunun sebebi ise koyu yüzeylerin ışığı emmeleri ve yansıtmamalarıdır (Murphy, 2000). Lazerin endüstriyel uygulamalarından biri olan lazer algılayıcılar ise temel olarak optik algılayıcılar ile aynı çalışma mantığına sahiptirler. En büyük farkları ise çok daha yüksek bir dalga boyundaki bir ışık ile çalışıyor olmalarıdır. Mesafe ve nesne algılama sorunu açısından incelediğimizde ise lazer algılayıcıların çok etkin bir sonuç verdiği gözlenmektedir. Lazer algılayıcıların dezavantajları ise ilk olarak sadece belirli bir düzeydeki nesneleri algılamalarıdır, o düzeyin üstündeki veya altındaki cisimleri algılayamazlar. Ayrıca hala diğer algılayıcılara göre fiyat olarak oldukça pahalıdır. Bunlara ek olarak bazı cisimler (özelikle cam gibi) lazer algılayıcı tarafından saydam olarak algılanır (Zunino, 2002).

Ultrasonik algılayıcılar ise ilk defa 1917 yılında kullanılmaya başlanmıştır. Ses dalgaları yoluyla cisimlerin yerini saptayan bu aracın temel ilkeleri Fransız fizikçi Paul Langevin tarafından oraya atılmıştır (Graff, K. F.,1981). Ses dalgasının bir noktaya gönderilip geri gelme süresine bağlı olarak ölçülen mesafe değerinden faydalanılmaktadır. Bu sistemde birden fazla sefer paketler yayımlanır ve ekonun alındığı zaman ölçülür. Bu zamana uçuş zamanı da denir. Bu zamanın mesafelerin ölçümünde kullanılmasında ses hızının bildiğimiz değerinin değişmediği ya da çevresel sıcaklığa bağlı olarak ihmal edilebilir bir biçimde değiştiği varsayılır. Sonar algılayıcılar lazer algılayıcılara göre daha ekonomiktir. Sonarla mesafe ölçümündeki ana dezavantaj nesnelerin yüzeyinden gerçekleşen yansıma ile ilgili problemlerdir. Buna aynasal yansıma adı da verilir (Min et al, 1997). Yansıma yönü gelen ses dalgasının yüzeyle yaptığı açıyla ve yüzeyin şekliyle ilgilidir. Geliş açısı ne kadar ufak olursa, sesin yansıma yapmadan yüzeyi sıyırması ihtimali o kadar yükselir ve bu şekilde hatalı bir mesafe ölçümü yapılır. Bu duruma aynasal denmesi sebebiyse, kaygan yüzeyler, yansıtıcı özellikleri ile bu sorunun büyümesine yol açarlar. Daha kaba yüzeylerde ise düzensiz yansımalardan birinin geri dönme ihtimali daha yüksektir.

Uzak mesafelerde ise ölçümlerin kesinliği büyük oranda düşecektir, bunun sebebi yanlış ölçümlerin dönmesi ihtimalinin yüksek olmasıdır. Bu dezavantajlarına rağmen sonar algılayıcılar ile ölçümler hareketli robot uygulamalarında sıklıkla uygulanmaktadır, bu uygulamalar arasında iç mekan ve dış mekan haritalarının çıkarılması da yer almaktadır (http://robot.cmpe.boun.edu.tr).

Bu tezin amacı, Eskişehir Osmangazi Üniversitesi, Elektrik Elektronik Mühendisliği Robotik ve Yapay Zeka Laboratuarında bulunan P3 – DX gezgin robotu kullanılarak bilinmeyen ortamların çevre haritalarının çıkartılabilmesi ve yine bilinmeyen ortamlarda robotun çevrede bulunan engellere çarpmadan gezinebilmesidir.

Bu çalışmada, çevre haritalama ve robot davranışlarının kontrolü işlemleri ARIA simülatöründe ve P3-DX robotunda uygulanmış ve sonuçların değerlendirilmesi yapılmıştır. Duvar bulma, duvar takibi, öndeki ve yandaki engellerden kaçınma, içbükey ve dışbükey köşe dönüşleri, tamponların kontrolü, tekerleğin sıkışma durumundaki kontrol davranışları geliştirilmiştir. Ultrasonik algılayıcılardan ve kodlayıcıdan alınan veriler doğrultusunda Bayes güncellemeli doluluk ızgaraları metodu kullanılarak çevre haritası oluşturulmuştur. Harita tespit etmekte amacımız robotun nerede olduğunu ve etrafındaki cisimlerin konumlarını bilmek istememizdir. Robotun çevresi doğru bir şekilde modellenir ve haritası çıkarılırsa birçok karmaşık görev daha hızlı ve güvenli bir şekilde robot tarafından gerçekleşir.

Harita çıkarmanın ikinci avantajı iyi bir çalışma ortamı sağlanmasıdır. Birçok algılayıcı bilgisinden birçok farklı konum ve yön için karmaşık bir plan elde edilebilinir. Değişik algılayıcıların birlikte kullanılması hata oranını azaltacaktır. Örnek olarak kamera ve sonar algılayıcıların birlikte kullanılması algılayıcıların zayıf noktalarını azaltacaktır. Kamera beyaz duvarı göremeyebilir fakat sonar bunu algılayacaktır.

Harita çıkarmada karşılaşılan en önemli problemlerden biri de konum ve yön bilgisini algılayıcılardan almasıdır. Eğer robot yanlış konum bilgileri alırsa harita güncellememiz de yanlış olacaktır. Robotun ilerleme mesafesi izlendiğinde tekerleklerin dönme sayısı ölçümü yapılır. Tekerleklerde kayma ve açısal yönlenme olursa robotun konum bilgisi de hatalı olacaktır. Bu problem öz konumlanma ile çözümlenmelidir (O'Sullivan, 2003).

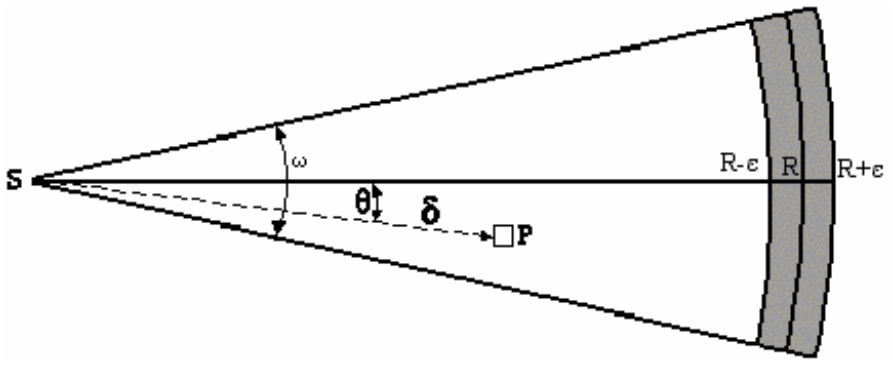
Tezin yapısı şu şekildedir: Bölüm 2'de gezgin robotlarda algılayıcılarla harita çıkarma yöntemleri anlatılmıştır. Bölüm 3'de doluluk ızgaraları metodunun sonar modeline uygulanması, bölüm 4'de robot davranışlarının geliştirilmesi, bölüm 5'de kullanılan robot sistemi ve bölüm 6 'da robot davranışları için geliştirilen algoritmanın simülatör ve P3-DX robotunda uygulanmış test sonuçları verilmiştir. 7. bölümde sonuçlar ve öneriler yer almıştır. En son olarak 8. bölümde ise yapılan çalışmalarda faydalanılan kaynaklar yer almıştır.

2. GEZGİN ROBOTLARDA ALGILAYICILARLA HARİTA ÇIKARMA YÖNTEMLERİ

2.1. Ultrasonik Mesafe Algılayıcılarla Harita Çıkarma Yöntemleri

2.1.1 Moravec ve Elfes – 1985 Geniş açılı sonardan yüksek çözünürlüklü harita çıkarma

Moravec ve Elfes (1985) *Geniş Açılı Sonardan Yüksek Çözünürlüklü Harita Çıkarma* konulu bildirileri ile başlıca metrik haritalar üzerinde çalışmışlardır. Bu yaklaşımda sabit bir konumda bulunan, birbirlerine göre konumları ve açıları bilinen sonar algılayıcılardan alınan mesafe bilgileri kullanılır. Bu mesafe bilgileri daha sonra Şekil 2.1. de görüldüğü gibi sonar ışının gördüğü alanda bir engelin var olduğu varsayılarak ve dolu olan bölümün olasılığı artırılarak iki boyutlu haritaya çevrilir. Harita çıkarmak için geliştirilen bu model uzaysal rasgele alanın dolanmasıyla oluşturulan doluluk ızgaraları olarak adlandırılır. Eğer ışınların hepsi bölgede engel olmadığını gösteriyorsa doluluk olasılığı azalır. Tek bir sonar çok az bilgi verirken, her sonardan alınan bilgi haritayı biraz değiştirir ve 100 ms de bir okuma ile güvenli ızgaralar meydana gelir. Geniş ışın açıklığı bu yaklaşımda 30º, ızgaraların doluluğu hakkında dolaylı bilgi verir. Engelin sonar ışının gördüğü mesafe (R) eksi sonar hatası (ε) , $R-\varepsilon$ veya sonar ışının gördüğü mesafe (R) artı sonar hatası (ε), $R+\varepsilon$ arasındaki bölgede olduğu varsayılır. Sonar mesafesi artıkça kesinlik değeri azalır. Aynı zamanda engel ne kadar sonarın merkez doğrultusunda ise kesinlik değeri o kadar artar.

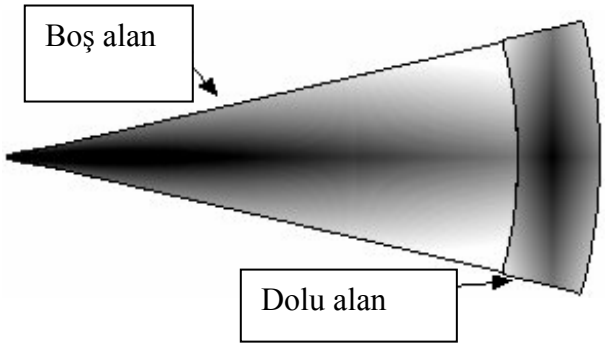


Şekil 2.1 Sonar Işının Modellenmesi

Şekil 2.1. ’de Moravec ve Elfes (1985) tarafından kullanılan sonar ışın modeli gösterilmektedir. Bu modelde:

- S, sonar algılayıcı
- P, güncellenen hücre
- ε, sonarın sapma hata ortalaması
- ω, ışın açıklığı, algılayıcı S tarafından yayılan ışının açısı
- θ, SP çizgisi ile ışının ana ekseni arasındaki açı
- R mesafe bilgisi- sonarın engele çarpana kadar gittiği yol

Bu teknik, boş harita ve dolu harita olmak üzere iki orta modelden geliştirilmiştir. Bu şekilde dolu veya boş alan olduğu tespit edilir. Algılayıcı modeli ile boşsa boş olma olasılığı, dolu ise dolu olma olasılığı hesaplanır. Son olarak boş ve dolu haritalar tek bir yerde gösterilmek üzere birleştirilir.



Şekil 2.2. Sonar Işının Olasılık Dağılımı

Şekil 2.2 de görülen koyu alanlar olasılığı yüksek olan bölümleri göstermektedir. Sol taraf boş alanı, sağ tarafı dolu alanı göstermektedir.

Herhangi bir hücrenin boş olma olasılığı, $P_b(X,Y) = E_r(\delta) * E_a(\theta)$ olarak hesaplanır. Burada $E_r(\delta)$ algılayıcıdan hücreye kadar olan uzaklığa bağlı olarak hücrenin boş olma olasılığı, $E_a(\theta)$ ışının merkez açısı ile algılayıcının gördüğü hücre arasındaki açıya bağlı olarak hücrenin boş olma olasılığıdır.

- $E_r(\delta) = \begin{cases} 1-((\delta - R_{\min})/(R-\varepsilon-R_{\min}))^2 & R_{\min} \le \delta \le R-\varepsilon \\ 0 & diger \end{cases}$

Hücrenin algılayıcıya olan uzaklığı δ, R mesafesinden büyükse veya $R_{\min}$ den daha küçük ise hücrenin boş olma olasılığı yüksektir.

- $E_a(\theta) = 1-(2\theta/\omega)^2 \qquad (-\omega/2) \le \theta \le (\omega/2)$

Eğer hücre merkeze yakınsa sonarın engeli algılama olasılığı yüksektir. Hücrenin dolu olma olasılığı, $P_d(X,Y) = O_r(\delta) * O_a(\theta)$ olarak hesaplanır. Burada $O_r(\delta)$ algılayıcıdan hücreye kadar olan uzaklığa bağlı olarak hücrenin dolu olma olasılığı, $O_a(\theta)$ ışının merkez açısı ile algılayıcının gördüğü hücre arasındaki açıya bağlı olarak hücrenin dolu olma olasılığıdır.

- $O_r(\delta) = 1-((\delta - R)/\varepsilon)^2 \qquad (R-\varepsilon) \le \delta \le (R+\varepsilon)$

Eğer hücre dolu bölgede değil ise $O_r(\delta) = 0$ olarak hesaplanır. Tek sonarın mesafe bilgisi hücrenin doluluğu hakkında çok az bilgi verir. Bu nedenle daha önce okunan sonar mesafe bilgilerinin birleştirilmesi gerekmektedir. Moravec ve Elfes bu güncelleme için birçok basamak amaçlamışlardır. Bu basamaklar boş ve dolu haritalarda simetrik değildir.
Boş haritanın güncellenmesi;

- $Boş(X,Y) = Boş(X,Y) + P_b(X,Y) - Boş(X,Y) * P_b(X,Y)$

Eğer hücrenin boş olma olasılığı 0.9 ise yeni tahminimizde $P_b(X,Y) = 0.4$ ise o halde hücrenin boş olma olasılığı güncelleme yapılırsa (0.9+0.4) - (0.9*0.4) = 0.94 dir. Daha sonraki basamak dolu haritanın güncellenmesidir.

- $P_d(X,Y) = P_d(X,Y) * (1 - Boş(X,Y))$

Örneğin, en son sonarın okumasından hücrenin dolu olma olasılığı 0.8 ise ve bir önceki sonar okumasından boş olma olasılığına hücrenin 0.5 şans veriliyorsa o halde tekrar hesaplama yapılırsa, yeni tahmin 0.8*(1-0.5)=0.4 olarak hesaplanır. Bu sonuçtan eğer bir önceki okuma en son okuma ile tutmuyorsa o halde en son okuma doğru olmayabilir.

İkinci aşamada, bütün dolu hücreler toplanarak bir yerde normalleştirilir.

- $P_d(X,Y) = P_d(X,Y) / \Sigma P_d(X,Y)$
- $Dolu(X,Y) = Dolu(X,Y) + P_d(X,Y) - Dolu(X,Y) * P_d(X,Y)$

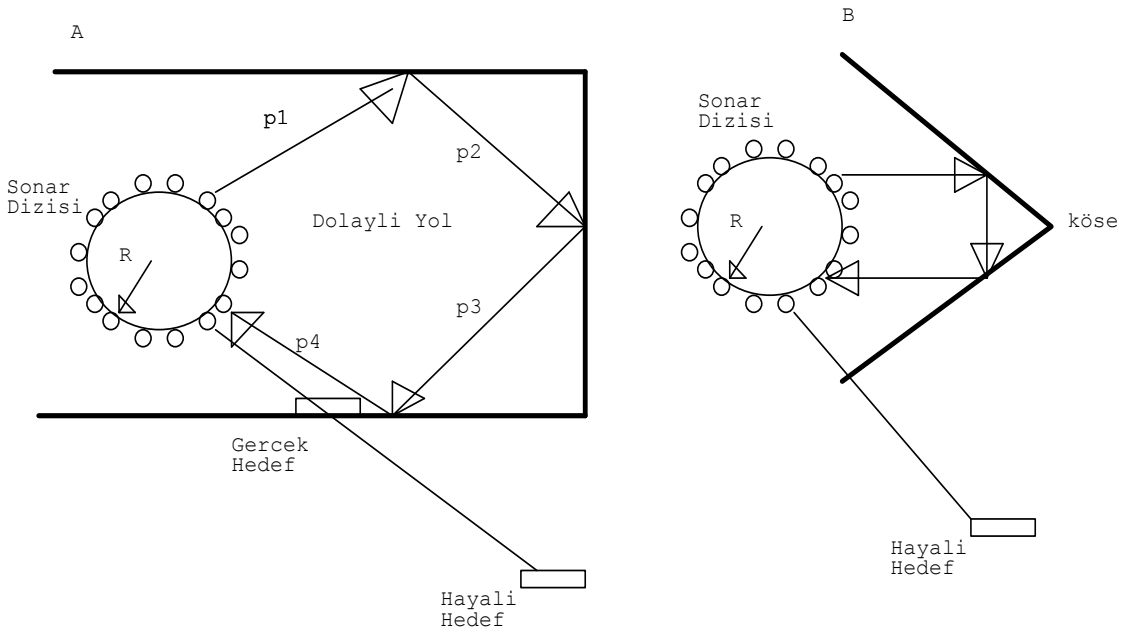
En son olarak hem boş hem de dolu harita tek haritada birleştirilir. Bu eşik değer basamağı ile gerçekleşir. Daha büyük olan ana harita için seçilir.

- $Harita(X,Y) = Dolu(X,Y)$ eğer $Dolu(X,Y) \geq Boş(X,Y)$
- $Dolu(X,Y) = Boş(X,Y)' - 1$

Boş ve dolu haritalar 0 dan 1 e kadar değer alır. 0 haritanın boş olduğunu (dolu olmadığını) belirtmektedir. Ana harita -1 den +1 e kadar değer alır. 0 değeri bilgi eksikliğini belirtir.

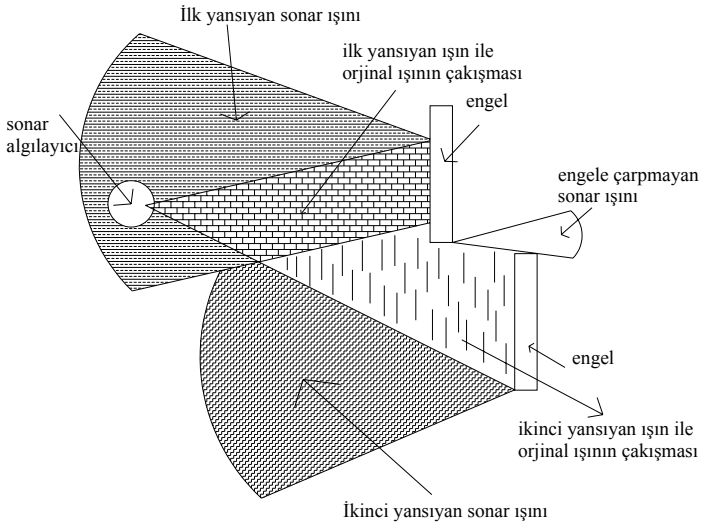
Bu yaklaşımda iki önemli dezavantaj bulunmaktadır. Bu tekniğin ilk dezavantajı yansımaların, çapraz etkilenmelerin düşünülmemesidir. Sonar ışınları birden çok yüzeye çarpar dönerse veya yüzey ışını yutarsa ışını hiç ışın dönmez ve hatalı okumalar meydana gelir. Bu nedenle de alan hatalı olarak dolu olan bir alan boş olarak düşünülebilinir (Moravec and Elfes, 1985).

Çapraz etkilenme; sonarla mesafe ölçülürken karşılaşılan istenmeyen bir olgudur ve bir algılayıcının gönderdiği ses dalgasını kendisinin değil de başka bir algılayıcının almasıdır. Bu olgu daha çok köşelerde oluşmakta ve sonarların mesafeyi olduğundan daha uzak algılamasına sebep olmaktadır (Bozma and Kuc, 1991).



Şekil 2.3 Çapraz Etkilenme (Min, et al., 1997)

Şekil 2.3' de görüldüğü gibi p4 mesafesi okunması gerekirken normalde gerçek hedef noktasına uzaklığı değil de sanki daha uzakta bir noktaya olan hayali bir nokta varmış gibi uzaklık bilgisi okunur. Bunun sebebi p1+p2+p3+p4 mesafesinin okunmasıdır. Bu tekniğin ikinci dezavantajı tek hedef varsayımıdır. Sonar tek engel varmış gibi ışını gönderir, şekil 2.4 de görüldüğü gibi engelin altında başka bir engel varsa sonar bunu algılayamaz. Bir sonarın birçok yansıması meydana gelir sonar bunlardan sadece birinciyi algılar bunların dışındakiler ihmal edilir. Kurt Konolige (1997), Muriel metodu ile bu probleme çözüm bulmuştur. Bu metot 2.1.4 bölümünde anlatılacaktır.



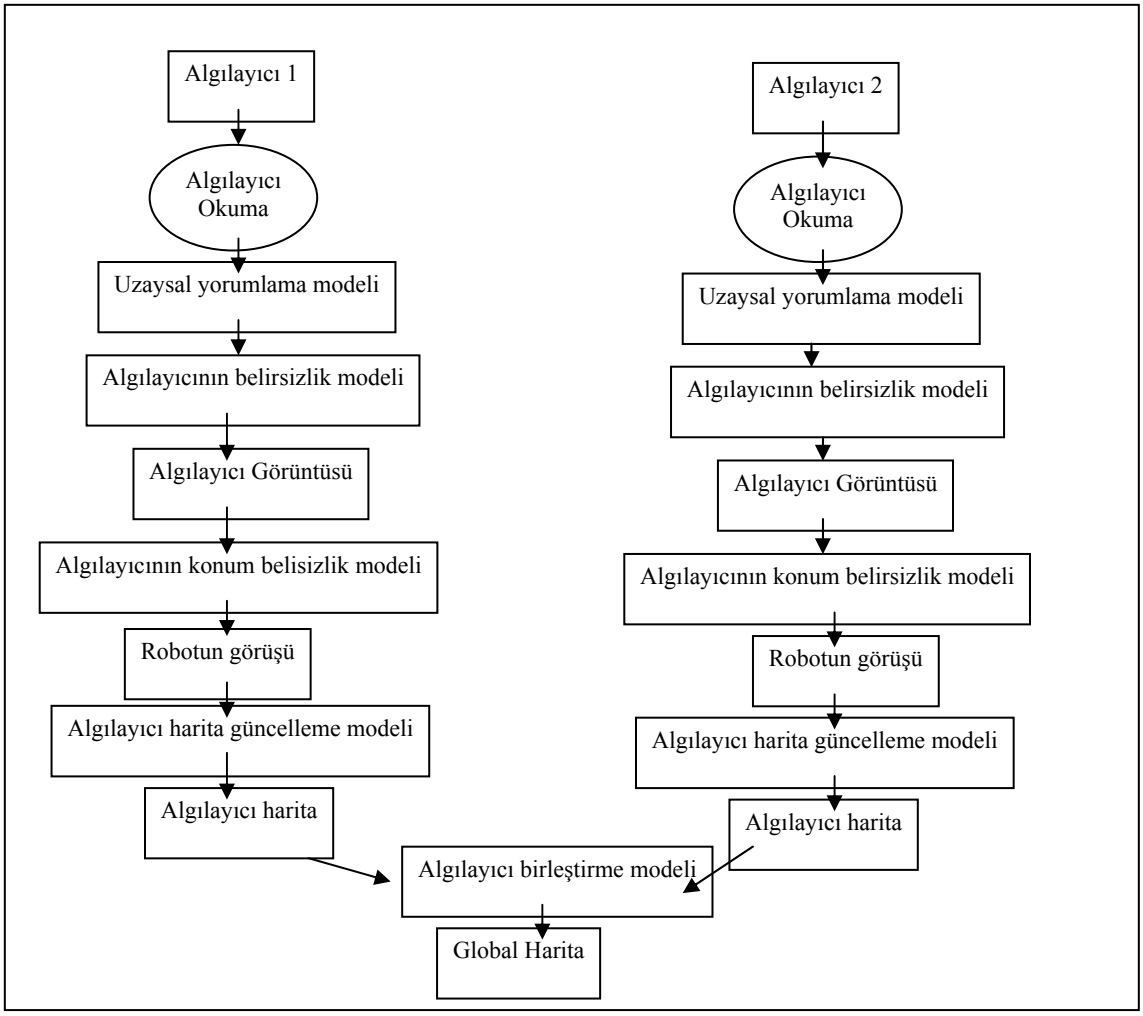
Şekil 2.4 Bir Sonarın Yansımaları

2.1.2 Matthies ve Elfes – Izgara temelli gösterimi kullanarak sonar ve stereo mesafe verilerin birleşimi

Elfes (1988), doluluk ızgaraları çalışmasını stereo görüntü ve sonar verileri ile birleştirmiştir. Bu çalışmanın anahtar noktası Moravec ve Elfes'in yaklaşımına göre güncelleme fonksiyonun farklı olmasıdır.

2.1.2.1 Değişik algılayıcıların birleştirilmesi

Şekil 2.5 den de görüldüğü gibi iki farklı algılayıcıdan bilgi alınır. Bunlar uzaysal bir modele dönüştürülür. İki algılayıcının görmesi tek bir haritada birleştirilir.



Şekil 2.5. Doluluk Izgaraları Temelli Algılayıcıların Birleşimi

2.1.2.2 Harita değerlerini Bayes olasılık teorisi ile güncelleme

Harita güncelleme Moravec ve Elfes'in geliştirdiği yaklaşıma göre daha güvenlidir. Bu yaklaşımda güncelleme sezgisel değil daha kesin olarak Bayes teoremi kullanılmıştır.

- $P(s_i \mid e) = \dfrac{P(e \mid s_i)P(s_i)}{\sum_j P(e \mid s_j)P(s_j)}$

e sonar okumasında s_i tahmin edilen durumdur. $P(s_i)$ ızgaranın dolu ya da boş olması kesinliğine göre ilk olasılık değeridir. P(Dolu) ilk dolu olma olasılığı, P(Boş) ilk boş olma olasılığıdır. Güncelleme yapılırsa aşağıdaki denklem elde edilir.

- $P(Dolu \mid R) = \dfrac{P(R \mid Dolu)P(Dolu)}{P(R \mid Dolu)P(Dolu) + P(R \mid Boş)P(Boş)}$
- $P(Boş) = 1 - P(Dolu)$ ve $P(R \mid Boş) = 1 - P(R \mid Dolu)$

Harita Güncellemede Bayes Yaklaşımını Kullanmanın Avantajları

1. Hem değişme hem de birleşme özelliği vardır bu yüzden de farklı algılayıcı sistemleri tek bir modelde birleştirilebilir.
2. Eğer ızgaranın değeri bilinmiyorsa P(Dolu)=0.5 alınır ve yeni olasılık E=P(R|Dolu) olarak hesaplanır.
3. Çatışan ölçümler birbirini götürür. Eğer dolu ve boş olma olasılığı eşitse o halde değer bilinmiyor olarak hesaplanır.

Harita Güncelleme Bayes Yaklaşımını Kullanmanın Dezavantajları

1. İlk olarak hücrenin bir kere güncellenmesi hücrenin doluluk değerinde büyük değişikliklere yol açar.

Örneğin, ilk olasılık 0.5 ve güncellenen yeni olasılık 0.1 olarak verilsin.

- $P(Dolu \mid R) = \dfrac{0.1 * 0.5}{0.1 * 0.5 + (1 - 0.1)(1 - 0.5)} = \dfrac{0.05}{0.05 + 0.45} = 0.1$

Bayes güncellemesi yapıldığında yeni olasılık 0.1 olarak bulunur. Bu sonuçtan da anlaşıldığı gibi bir tek yanlış okuma haritayı değiştirebilmektedir.

2. P(Dolu) değeri 0' ya da 1'e yakınsar.

Örneğin daha önceki hücrenin doluluk değeri 0 kabul edilsin,

- $P(Dolu \mid R) = \dfrac{P(R \mid Dolu) * 0}{P(R \mid Dolu) * 0 + (1 - P(R \mid Dolu))(1 - 0)} = \dfrac{0}{1 - P(R \mid Dolu)} = 0$

Bu hesaplamada P(R|Dolu)'nun hangi değeri aldığının önemi yoktur, eğer iki olasılık 0 ise. Aynı şey ilk olasılık değeri 1 alırsa da geçerlidir.

Ufak değişimler olasılıkta büyük sonuçlar doğurabildiği gibi hücrenin bir kere 0 ya da 1 e yaklaşması olasılığı değiştirmemektedir. Bu da gürültülü verilerin yeterli derecede doğru okuma ile kalibre edilmelerini imkansızlaştırır.

2.1.3 Thrun -1993, Sinir ağları temelli yaklaşım

Thrun (1993) tarafından geliştirilen doluluk ızgaraları harita çıkarma yaklaşımını sinir ağlarını kullanarak gerçekleştirir. Bu metotta olasılık değerleri <0…..1> arasında değişir. Bu okumalar doğrultusunda kesinlik değerleri ikinci tabaka olan güvenli ağda elde edilir. Harita güncellemesi Matthies ve Elfes'in önerdiği yaklaşım ile aynıdır. Bu yaklaşımın sınırlamaları sinir ağlarının doğasından gelmektedir. Sinir ağları ile çalışırken ağ istenen düzeye yakınsayıncaya kadar devam edilir.

2.1.4 Konelige -1997 Harita çıkarma için genişletilmiş doluluk ızgaraları metodu

Konelige'nin (1997) öne sürdüğü yaklaşımda birçok algılayıcı bilgisini iki boyutlu bir haritada birleştiren olasılık temelli bir metot üzerinde durulmuştur. Diğer yaklaşımlardan farkı iki önemli probleme, yansımalar ve tekrarlı olarak fazladan okunan sonar bilgilerine, çözüm önermiştir. Bağımsız kesinlik değerlerine birden çok gösterim sağlayan MURIEL metodunu önermiştir. Bu metot doluluk ızgaralarını iki temel çizgide ayırır. İlk olarak algılayıcı modelini dağınık ve yansıyan olarak iki gruba ayırır. Algılayıcı okumalarını dinamik olarak yeni okunan mesafe bilgileri ile birleştirir. İkinci olarak birbirinden bağımsız okunan mesafe bilgileri gereğinden fazla tekrarlı bilgilere sebep olacaktır. Bunları önlemek için bütün algılayıcıların konum ve yönlenmeleri tespit edilir. Aynı ızgarayı gören ve farklı konumlardaki algılayıcılara filtre uygulanır. Aynı ızgaranın birden fazla okunması önlenir.

Konelige metodunun güncelleme amacı dolu ve boş hücrelerin olasılığını logaritmik tekniklerle birleştirmektir. Bu yaklaşımda hücrelere ayrı ayrı tahminlerde bulunulur.

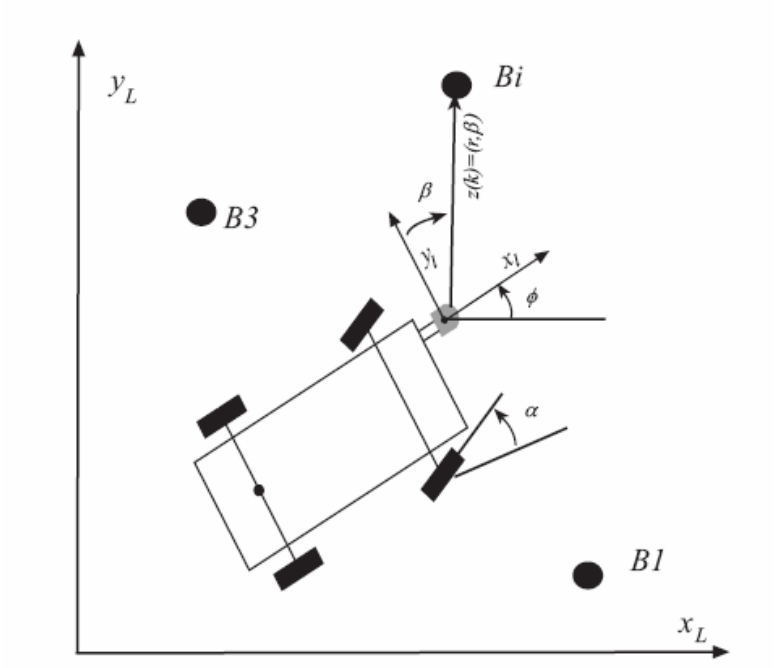
Doluluk ızgaraları metodu iki kısımda incelenir.

1. Tek hedef modeli: Doluluk ızgaralarında alan düzenli ızgaralara ayrılır. Daha önce açıklanan Bayes güncellenmesindeki hesaplamalar kullanılır.
2. Birden çok hedef modeli: Alanda birden çok engel vardır ve sonar ışını birden çok yansımaya maruz kalmaktadır. Yansımaların yoğunluğunun olasılığı da hesaba katılır.

2.2. Lazer Mesafe Algılayıcılarla Harita Çıkarma Yöntemleri

Lazer mesafe algılayıcılarla harita çıkarma yöntemleri incelendiğinde en çok uygulama alanı SLAM algoritmasıdır. SLAM, eş zamanlı konumlanma ve harita çıkarma algoritmasıdır. Bu algoritmada lazer tarafından doğal veya yapay işaretler üretilerek robotun bu işaretlere göre konumu belirlenerek, lokalizasyon yapılır. (Guivant, et al.,2000).

Global koordinatlarda robotun konumu şekil 2.6. da gösterilmektedir. α tekerleklerin yönlenmesidir. Lazer algılayıcı robotun ön tarafına yerleştirilmiş ve 50 metreye kadar nesneden mesafe bilgisi alabilmektedir.



Şekil 2.6. Robot Koordinat Sistemi

Yüksek yoğunluklu yansıma, doğal ya da yapay olarak belirlenen işaretin yansıtma yoğunluğunun yüksekliğine bağlıdır. Şekil 2.6 da işaret olarak $Bi_{(i=1....n)}$ gösterilmektedir. İşaretler robotun koordinatlarına (x_l, y_l) bağlı olarak $z(k) = (r, \beta, I)$ hesaplanır. r lazerle işaret arasındaki mesafe, β robotun koordinatlarına bağlı olarak lazer ışının açısı, I yoğunluk bilgisidir.

Robotun v_c hızıyla ve α yönlenmesiyle hareket ettiği düşünülürse, yörünge denklemi arka mile göre hesaplanırsa;

- $$\begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\phi}_c \end{bmatrix} = \begin{bmatrix} v_c.\cos(\phi) \\ v_c.\sin(\phi) \\ \frac{v_c}{L}.\tan(\alpha) \end{bmatrix}$$

Şekil 2.6 ve Şekil 2.7 de görüldüğü gibi lazer robotun ön kısmına yerleştirilmiştir, lazerden elde edilen koordinatlar robotun arka milinin merkezinin koordinatlarına dönüştürülür.

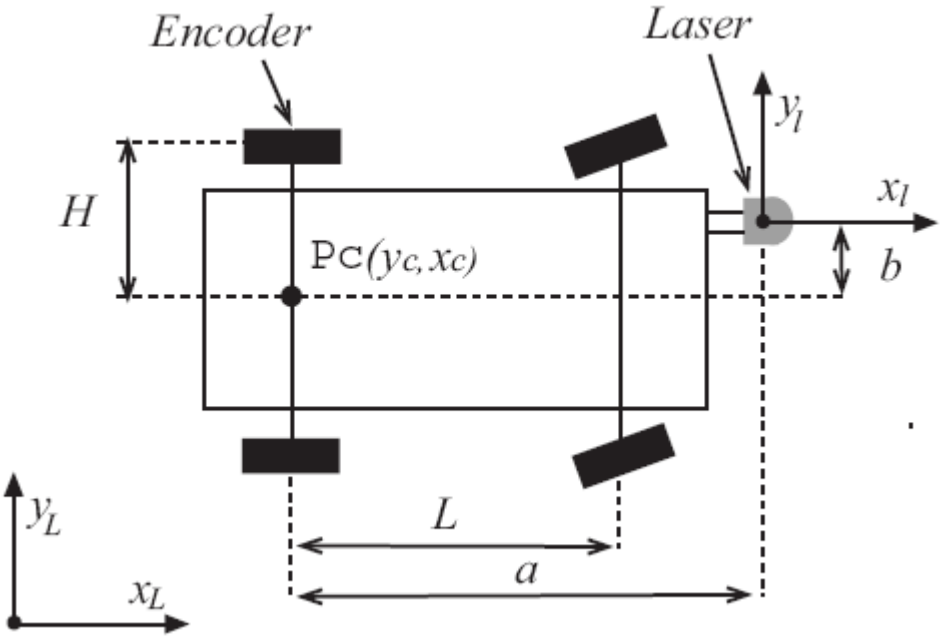
- $$P_L = P_c + a.\vec{T}_\phi + b.\vec{T}_{\phi+\frac{\pi}{2}}$$

P_L lazerin konumu ve P_c robotun arka milinin merkezinin konumudur. Dönüşüm, yönlenme açısıyla vektörel olarak gösterilir.

- $$\vec{T}_\phi = (\cos(\phi), \sin(\phi))$$

Skalar olarak gösterilirse;

- $$x_L = x_c + a.\cos(\phi) + b.\cos(\phi + \pi/2)$$
- $$y_L = y_c + a.\sin(\phi) + b.\sin(\phi + \pi/2)$$



Şekil 2.7 Kinematik Parametreleri

Son olarak tüm durum gösterilirse;

- $$\begin{bmatrix} \dot{x}_L \\ \dot{y}_L \\ \dot{\phi}_L \end{bmatrix} = \begin{bmatrix} v_c.\cos(\phi) - \frac{v_c}{L} \cdot (a.\sin(\phi) + b.\cos(\phi)) \cdot \tan(\alpha) \\ v_c.\sin(\phi) + \frac{v_c}{L} \cdot (a.\cos(\phi) - b.\sin(\phi)) \cdot \tan(\alpha) \\ \frac{v_c}{L}.\tan(\alpha) \end{bmatrix}$$

Hız denklemi de arka tekerleğin merkezinin koordinatlarına dönüştürülür.

- $$v_c = \frac{v_e}{\left(1 - \tan(\alpha).\frac{H}{L}\right)}$$

Global koordinatlarda modellenirse;

- $$\begin{bmatrix} x(k) \\ y(k) \\ \phi(k) \end{bmatrix} = \begin{bmatrix} x(k-1) + \Delta t v_c(k-1).\cos(\phi(k-1)) - \frac{v_c}{L} \cdot (a.\sin(\phi(k-1)) + \\ b.\cos(\phi(k-1))) \cdot \tan(\alpha(k-1)) \\ y(k-1) + \Delta t v_c(k-1).\sin(\phi(k-1)) + \frac{v_c(k-1)}{L} \cdot (a.\cos(\phi(k-1)) - \\ b.\sin(\phi(k-1))) \cdot \tan(\alpha(k-1)) \\ \frac{v_c(k-1)}{L}.\tan(\alpha(k-1)) \end{bmatrix}$$

Δt örnekleme zamanıdır.

Yukarıdaki denklem, işaretin yerine göre robotun konumu ve gezinme haritasının yerini belirlemede kullanılır.

Durum vektörü:

- $X = \begin{bmatrix} x_v \\ x_L \end{bmatrix}$
- $x_v = (x, y, \phi) \in R^3$
- $x_L = (x_1, y_1, ...x_n, y_n) \in R^N$

x_v robotun durumunu, x_L gerçek işaretin durumunu belirtmektedir. İşaretler bilinmeyen bir konumdaki nesnelerdir. Dinamik modelleme yapılırsa;

- $x_v(k+1) = f(x_v(k))$
- $x_L(k+1) = x_L(k)$

Dinamik durumda eğer işaret hareket etmiyorsa x_L sabittir. Jacobian matrisi genişletilmiş sistem için:

- $$\frac{\partial f}{\partial x_V} = \begin{bmatrix} \frac{\partial f}{\partial x_v} & \Phi \\ \Phi^T & I \end{bmatrix} = \begin{bmatrix} J_1 & \Phi \\ \Phi^T & I \end{bmatrix}$$

$J_1 \in R^{3x3}$, $\Phi \in R^{3xN}$, $I \in R^{NxN}$

Denklemler robotun konumuna göre elde edilmiştir.

- $r_i = h_r(X) = \|(x, y) - (x_i, y_i)\|_2 = \sqrt{(x - x_i)^2 + (y - y_i)^2}$

- $\alpha_i = h_\alpha(X) = a\tan\left(\frac{(y - y_i)}{(x - x_i)}\right) - \phi + \frac{\pi}{2}$

(x, y) robotun konumu, (x_i, y_i) işaretin konumu olarak belirtilmektedir. (r_i, α_i) vektörünün Jacobian matrisi ;

- $\frac{\partial h}{\partial X} = \begin{bmatrix} \frac{\partial h_r}{\partial X} \\ \frac{\partial h_\alpha}{\partial X} \end{bmatrix} = \begin{bmatrix} \frac{\partial r_i}{\partial(x, y, \phi, \{x_i, y_i\})} \\ \frac{\partial \alpha_i}{\partial(x, y, \phi, \{x_i, y_i\})} \end{bmatrix}$
- $\frac{\partial h_r}{\partial X} = \frac{1}{\Delta}[\Delta_x, \Delta_y, 0, 0, 0, ..., -\Delta_x, -\Delta_y, 0, ..., 0, 0]$
- $\frac{\partial h_a}{\partial X} = [-\frac{\Delta_y}{\Delta^2}, \frac{\Delta_x}{\Delta^2}, -1, 0, 0, ..., \frac{\Delta_y}{\Delta^2}, -\frac{\Delta_x}{\Delta^2}, 0, ..., 0, 0]$
- $\Delta_x = (x - x_i)$, $\Delta_y = (y - y_i)$, $\Delta = \sqrt{(\Delta x)^2 + (\Delta y)^2}$

Bu denklemler kullanılarak çevrenin haritası ve robotun gittiği konum belirlenebilir.

Lazer algılayıcısı sonara göre daha hassas ölçüm yaptığından harita çıkarmada sonara göre daha iyi sonuç vermektedir.

2.3. Kamera ile Harita Çıkarma Yöntemleri

Kamera ile harita çıkarma yöntemi sonara ve lazere göre daha ileri düzeydedir. Görme ile alınan veriler daha zengindir. Thrun (1998), robot gezinme davranışında lazer mesafe algılayıcısını ve bir kısım görmeyi birlikte kullanmıştır. Davison ve Murray'ın (1998) çalışmasında SLAM algoritması kamera kullanılarak gerçekleştirilmiştir.

3. DOLULUK IZGARALARI METODUNUN SONAR MODELİNE UYGULANMASI

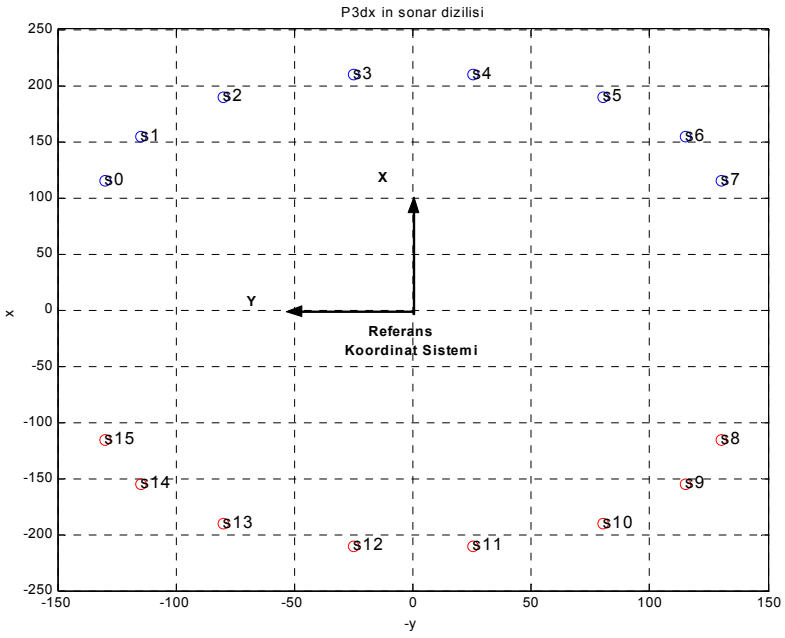
Sonar modelinde ses dalgasının bir noktaya gönderilip geri gelme süresine bağlı olarak ölçülen mesafe değerinden faydalanılmaktadır. Sonar modeli sonarın yaydığı ses dalgası yolunda bir nesnenin bulunup bulunmadığına, mesafe bilgilerine göre karar vermede kullanılır. Sonar modeli mesafe bilgileri doğrultusunda robotun etrafındaki alan ve bu alanın boş ve dolu olma olasılığı hakkında bilgi edinilmesini sağlar.

Bu çalışmada kullanılan P3 DX robotunun ön tarafında 8 adet arka tarafında da 8 adet olmak üzere toplam 16 sonar vardır. Bu sonarlarla minimum 15 cm, maksimum 7m ölçüm yapılabilmektedir. Her sonar kendi merkezine göre ± 15 derecelik bir bölgeyi taramaktadır. Sonarların robot üzerindeki referans bir koordinat sistemine göre konum ve yönlenmeleri Çizelge 3.1'de verilmiştir.

Çizelge 3.1 Sonarların Robot Üzerindeki Konum ve Yönlenmeleri

Sonar no	x(mm)	y(mm)	th(derece)
0	115	130	90
1	155	115	50
2	190	80	30
3	210	25	10
4	210	-25	-10
5	190	-80	-30
6	155	-115	-50
7	115	-130	-90
8	-115	-130	-90
9	-155	-115	-130
10	-190	-80	-150
11	-210	-25	-170
12	-210	25	170
13	-190	80	150
14	-155	115	130
15	-115	130	90

Sonarların konum ve yönlenmeleri Şekil 3.1 de görülmektedir.



Şekil 3.1 P3dx robotun sonarlarının konumları

3.1. Sonarın Bölgeleri

Şekil 3.2 de sonar modeli gösterilmektedir. Şekil 3.2 de bir tek sonar ışının modeli görülmektedir. Bu şekilden de görüldüğü gibi,

R → Sonarların maksimum okuma uzaklığı

r → Sonarın okuduğu mesafe

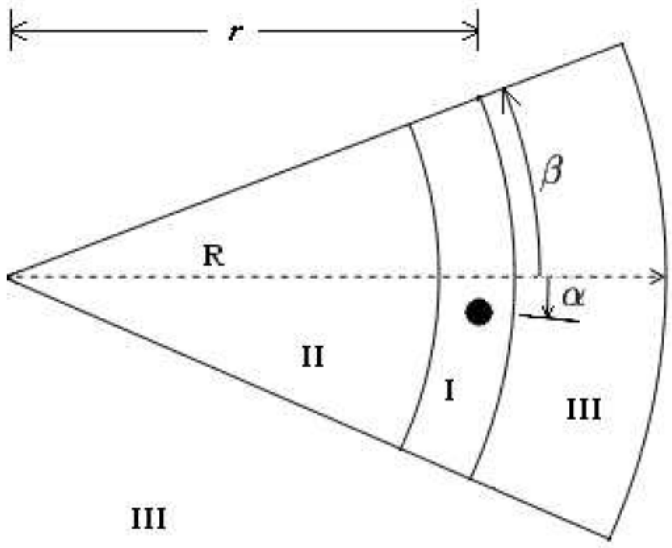
β → Sonarın görme açısının yarısı

α → Sonarın nesneye olan ışımasının açısı

1.Bölge: Bu bölgedeki bütün noktalar sonar tarafından gönderilen mesafe ile geri gelen mesafe eşittir. Bu bölgedeki ızgaralar büyük olasılıkla doludur.

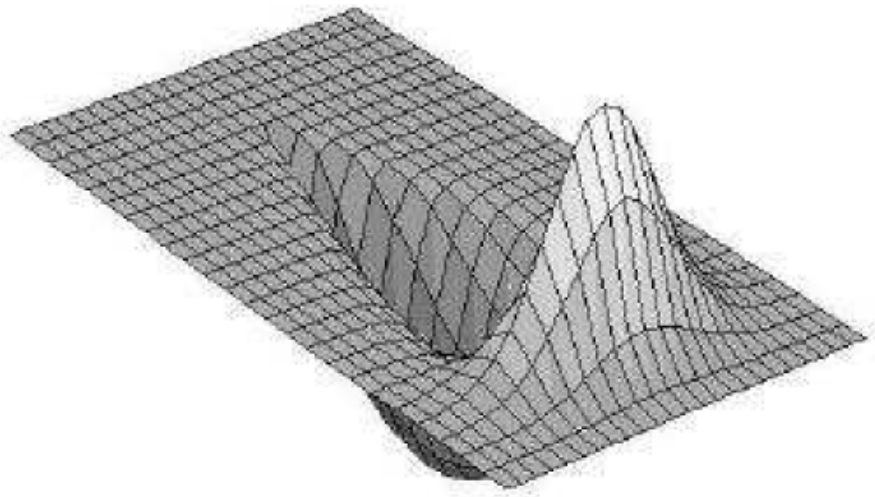
2.Bölge: Bu bölgedeki ızgaralar büyük olasılıkla boştur. Bu bölge sonarla 1.bölge arasında bulunur.

3.Bölge: Bu bölgedeki ızgaralar bilinmemektedir. Çünkü sonarın yansımasından dolayı oluşan veriler vardır.



Şekil 3.2 Sonar Modelin Bölgelere Ayrılması

Şekil 3.2 de verilen sonar ışını bitişik elemanlara ayrılır. 1. bölgede gönderilen bir ışın geri alınırsa o halde bir engel vardır. Engel olma olasılığı sonarın yansıma mesafesi ile ilgilidir. α değeri sıfıra yaklaştıkça 1. bölgede engel bulunma olasılığı artar. Başka bir sezgisel yaklaşımda eğer engel yakındaysa, sonardan alınan mesafe bilgisi küçükse hata oranı daha düşüktür.



Şekil 3.3 Bir Sonar Okuması Tarafından Gerçekleşen Olasılık Görüntüsü

r uzaklıktaki α açısındaki bir nesnenin sonar modeli olasılığını hesaplamak için ;

1. Bölge için;

$$P(dolu) = \frac{\left(\frac{R-r}{R}\right)+\left(\frac{\beta-\alpha}{\beta}\right)}{2} * maksimumdolu \quad (3.1)$$

$$P(boş) = 1 - Pdolu \quad (3.2)$$

2.Bölge için;

$$P(dolu) = 1 - P(boş) \quad (3.3)$$

$$P(boş) = \frac{\left(\frac{R-r}{R}\right)+\left(\frac{\beta-\alpha}{\beta}\right)}{2} \quad (3.4)$$

Engel sonara yakınsa r değeri küçük olacağından $\left(\frac{R-r}{R}\right)$ değeri büyüyecektir. Eğer engel sonarın bulunduğu hizaya yakınsa α değeri küçülecek $\left(\frac{\beta-\alpha}{\beta}\right)$ değeri büyüyecektir. Maksimum dolu olma katsayısı genelde 1.0 kabul edilmektedir. Dolu ızgaralarla kaplanmış bir sonar ışın modeli şekil 3.3 te görülmektedir. Yükselen bölüm ızgaraların dolu olma olasılığını, alçalan bölüm ızgaraların boş olma olasılığını belirtmektedir.

3.2. Güncelleme Yöntemleri

Doluluk ızgaraları yönteminde üç tip güncelleme yapılabilmektedir.[13] Bunlar 1. Bayes güncellemesi 2. Dempster-Shafer güncellemesi 3. HIMM güncellemesi

3.2.1 Bayes güncellemesi

3.1 bölümünde verilen denklemler olasılık hipotezi (H) oluşturulmasını sağlar.

- $H = \{dolu, boş\}$ veya $H = \{H, \neg H\}$ ve $0 \le P(H) \le 1$

Temel olasılık özelliklerine göre;

- $P(\neg H) = 1 - P(H)$

$P(H)$ ve $P(\neg H)$ şartsız olasılıklardır. Şartsız olasılıklar sadece ön bilgi sağlar, algılayıcı bilgileri bu olasılıklar kullanılarak birleştirilemez. Bayes kuralı bu probleme matematiksel bir çözüm sağlar. Bayes kuralı aşağıda açıklandığı gibi elde edilir.

$P(A)$ A'nın gerçekleşme olasılığı olsun ve $P(B)$ de B'nin gerçekleşme olasılığı olsun. $P(A \cap B)$ ise A ve B'nin birlikte gerçekleşme olasılığıdır. $P(A \mid B)$, B'nin gerçekleştiği yerde A'nında gerçekleşme olasılığıdır. Bundan dolayı çarpım kuralından denklem 3.5 elde edilir.

$$P(A \cap B) = P(A) \times P(B \mid A) \tag{3.5}$$

Denklem 3.5 tekine benzer olarak,

$$P(B \cap A) = P(B) \times P(A \mid B) \tag{3.6}$$

Fakat bilindiği üzere,

$$P(A \cap B) = P(B \cap A) \tag{3.7}$$

Denklem 3.5 ve 3.6 'yı kullanarak $P(A \mid B)$ çözümü,

$$P(A \mid B) = \frac{P(B \mid A) \times P(A)}{P(B)} \tag{3.8}$$

Genelleştirilirse bu denklem,

Sonraki olasılık= (koşullu olasılık x önceki olasılık) / tüm olasılık (3.9)

Önceki olasılık $P(H \mid s)$, Bölüm 3.1'de anlatıldığı gibi hesaplanır. Hipotezin bütün olasılıkları $P(s \mid H)P(H) + P(s \mid \neg H)P(\neg H)$ sonar modeline göre hesaplanır. Sonraki olasılık denklem 3.9'dan doluluk ızgarasının dolu olduğu durumda koşullu olasılığı verir.

İdeal olarak eğer bütün elemanlar haritalama sürecinde sadece bir kere güncellenirse yeterlidir. Bu tür bir olasılık nadiren oluşur. Bayes güncellemeli doluluk ızgaraları metodu artan bir yöntemdir. Bu nedenle bir nokta sonarın birden çok yansımasından güncellenebilir. Denklem 3.9'un yardımı ile aşağıdaki denklem 3.10 elde edilir.

$$P(H \mid s_1, s_2, ...s_n) = \frac{P(s_1, s_2 ....s_n \mid H)P(H)}{P(s_1, s_2, ...s_n \mid H)P(H) + P(s_1, s_2 ....s_n \mid \neg H)P(\neg H)} \quad (3.10)$$

Sonar okumalarının birbirinden bağımsız olduğu düşünülürse,

$$P(s_1, s_2, ..., s_n \mid H) = P(s_1 \mid H)P(s_2 \mid H).....P(s_n \mid H) \quad (3.11)$$

$P(s_n \mid H)$; n sayıda incelemede hesaplamada işlem sayısının artmasından dolayı ekonomik olamayacaktır.

$$P(A \mid B) \times P(B) = P(B \mid A) \times P(A) \quad (3.12)$$

Denklem 3.12 deki kural denklem 3.10 ile birleştirilirse, kuralın tekrarlı versiyonu elde edilir.

$$P(H \mid s_n) = \frac{P(s_n \mid H)P(H \mid s_{n-1})}{P(s_n \mid H)P(H \mid s_{n-1}) + P(s_n \mid \neg H)P(\neg H \mid s_{n-1})} \quad (3.13)$$

Böylece her sonar okumasının var olan olasılığının yeni koşullara göre güncellemesi için denklem 3.13 kullanılabilinir.

3.2.2 Dempster- Shafer teorisi

Alternatif bir teori olan Dempster-Shafer teorisi Bayes olasılık teorisine çok benzemektedir. A.P. Dempster (1960), Harward Üniversitesinde ve Glen Shafer (1987) genişletilmiş Dempster-Shafer adını verdikleri bir çalışma yapmışlardır. Bayes kuralı kanıtların olasılığına dayanırken, Dempster-Shafer kuralı kısmi kanıtlara dayanır.

3.2.2.1 Shafer fonksiyonu

Sonar Modeli bölgelere göre incelenir

1.bölge için.

- $m(Dolu) = \dfrac{\left(\dfrac{R-r}{R}\right)+\left(\dfrac{\beta-\alpha}{\beta}\right)}{2} * Maxdolu$
- $m(Boş) = 0.0$
- $m(Bilinmeyen) = 1.00 - m(Dolu)$

2.bölge için;

- $m(Dolu) = 0.0$
- $m(Boş) = \dfrac{\left(\dfrac{R-r}{R}\right)+\left(\dfrac{\beta-\alpha}{\beta}\right)}{2}$
- $m(Bilinmeyen) = 1.00 - m(Boş)$

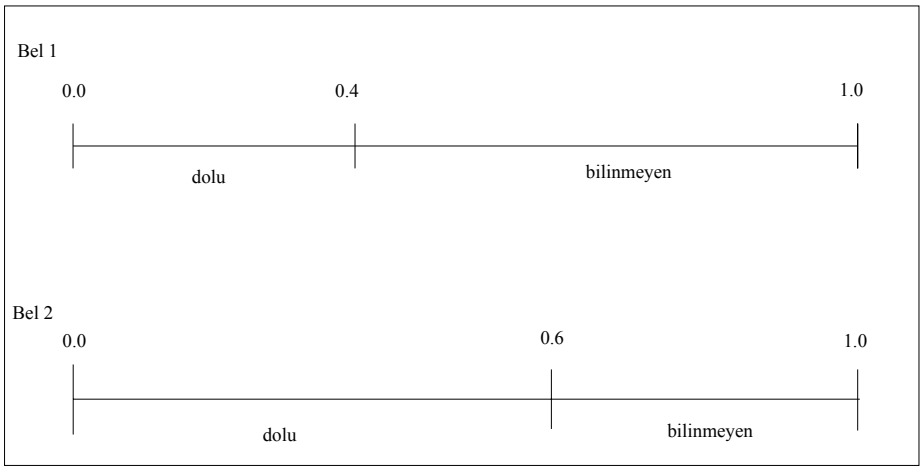
Shafer teorisinin olasılık teorilerinden kavramsal farkı kesin olmayan okumaların bilinmeyen olarak kabul edilmesidir (Murphy, 2000).

3.2.2.2 Dempster birleşme kuralı

Dempster kuralında iki bağımsız varsayım Bel1 ve Bel2 birleştirilir. Bel1 ve Bel2 iki fonksiyon olarak düşünülürse;

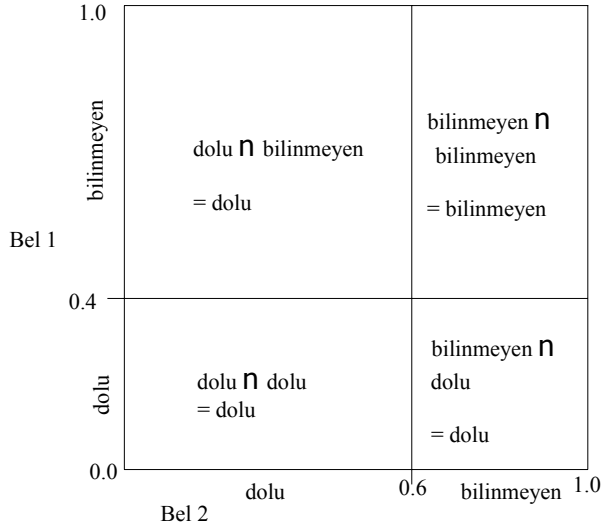
- Bel1 = m(Dolu) = 0.4, m(Boş) = 0.0, m(Bilinmeyen) = 0.6
- Bel2 = m(Dolu) = 0.6, m(Boş) = 0.0, m(Bilinmeyen) = 0.4

Şekil 3.4 de Bel 1 ve Bel2 fonksiyonları sayı doğrusunda verilmiştir.



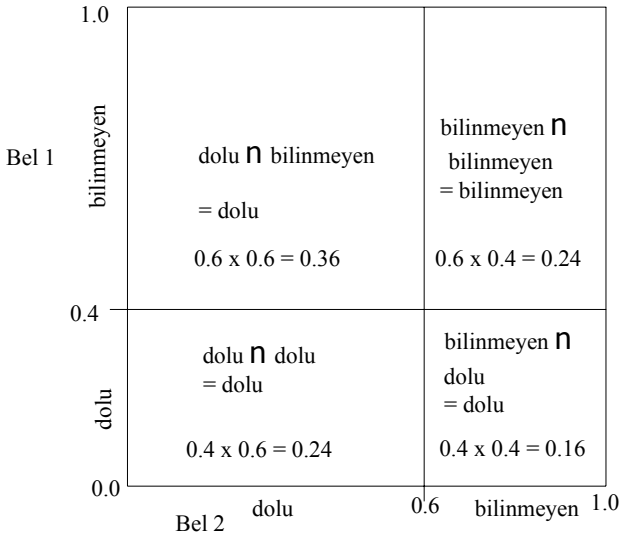
Şekil 3.4 İki Varsayımın Sayı Çizgisinde Gösterilmesi

Dempster kuralında iki sayı çizgisi dikey eksenlerden birim alanı 1 olan bir kare oluşturur.



Şekil 3.5 Sayı çizgisinin birim kareye dönüştürülmüş durumu

Küme kesişimleri şekil 3.5 te gösterilmektedir. Dört alt bölge bulunmaktadır, üç tanesi dolu, bir tanesi bilinmemektedir.



Şekil 3.6 Ağırlık değerlerinin birleştirilmesi

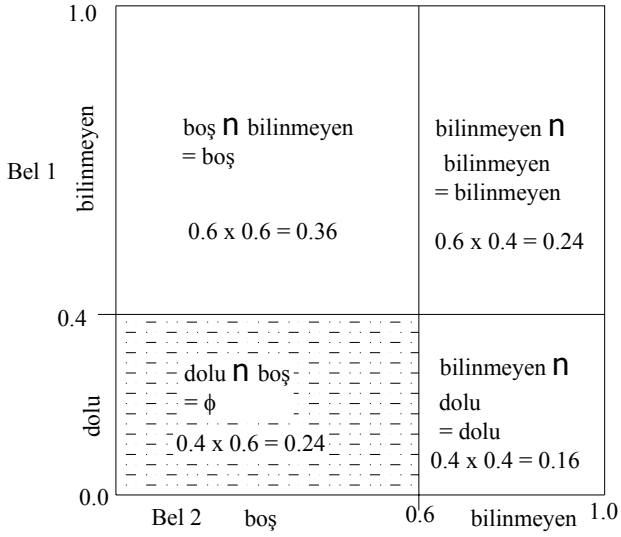
Şekil 3.6'da Bel 1 ve Bel 2 fonksiyonlarında verilen ağırlık değerleri hesaplanır. Şekil 3.7'de verilen hesaplamalar birleştirilerek sayı çizgisine dönüştürülür. Yeni Bel 3 fonksiyonu elde edilir.

Şekil 3.7 Bel 3 sonuç fonksiyonu

Bu fonksiyondan da görüldüğü gibi 0.76 ağırlıklı olarak hücre dolu, 1-0.76= 0.24 ağırlıklı olarak hücrenin değeri bilinmemektedir.

Çelişkili iki okuma olduğu durumda Bel1 ve Bel2 iki fonksiyon olarak verilirse;

- Bel1 = m(Dolu) = 0.4, m(Boş) = 0.0, m(Bilinmeyen) = 0.6
- Bel2 = m(Dolu) = 0.0, m(Boş) = 0.6, m(Bilinmeyen) = 0.4



Şekil 3.8 Normalleştirmeye gerek duyulan örnek

Şekil 3.8 de Bel 1 ve Bel 2 fonksiyonlarının birim kareye dönüştürülmüş şekli görülmektedir. Dolu ve Boş alanlar vardır. ϕ kritik olan bölgeyi göstermektedir. Shafer fonksiyonuna göre bu bölge bilinmemektedir ama 0.24 lük bir paya sahiptir. Bunun çıkarılması toplamın 1 olmasını engeller; bu nedenle de Dempster kuralına göre normalleştirme yapılır.

Eğer ϕ alanı mevcut değilse;

- $m(C) = \dfrac{\sum_{k}(m(C_k))}{1.0}$ olarak hesaplanır

Eğer ϕ alanı mevcutsa;

- $m(C) = \dfrac{\sum_{k}(m(C_k))}{1 - \sum_{p} m(\phi_p)}$ olarak hesaplanır.

Yukarıdaki denklemler örneğe uygulanırsa;

- $m(dolu) = \dfrac{0.16}{1 - 0.24} = 0.21$ olarak hesaplanır.
- $m(boş) = \dfrac{0.36}{1 - 0.24} = 0.47$ olarak hesaplanır.

- $m(bilinmeyen) = \frac{0.24}{1-0.24} = 0.32$ olarak hesaplanır.

Dempster kuralını genele indirgersek;

- $m(dolu) = \frac{\sum_{A_i \cap B_j = dolu} m(A_i)m(B_j)}{1 - \sum_{A_i \cap B_j} m(A_i)m(B_j)}$ olarak hesaplanır.

- $m(boş) = \frac{\sum_{A_i \cap B_j = boş} m(A_i)m(B_j)}{1 - \sum_{A_i \cap B_j = \phi} m(A_i)m(B_j)}$ olarak hesaplanır.

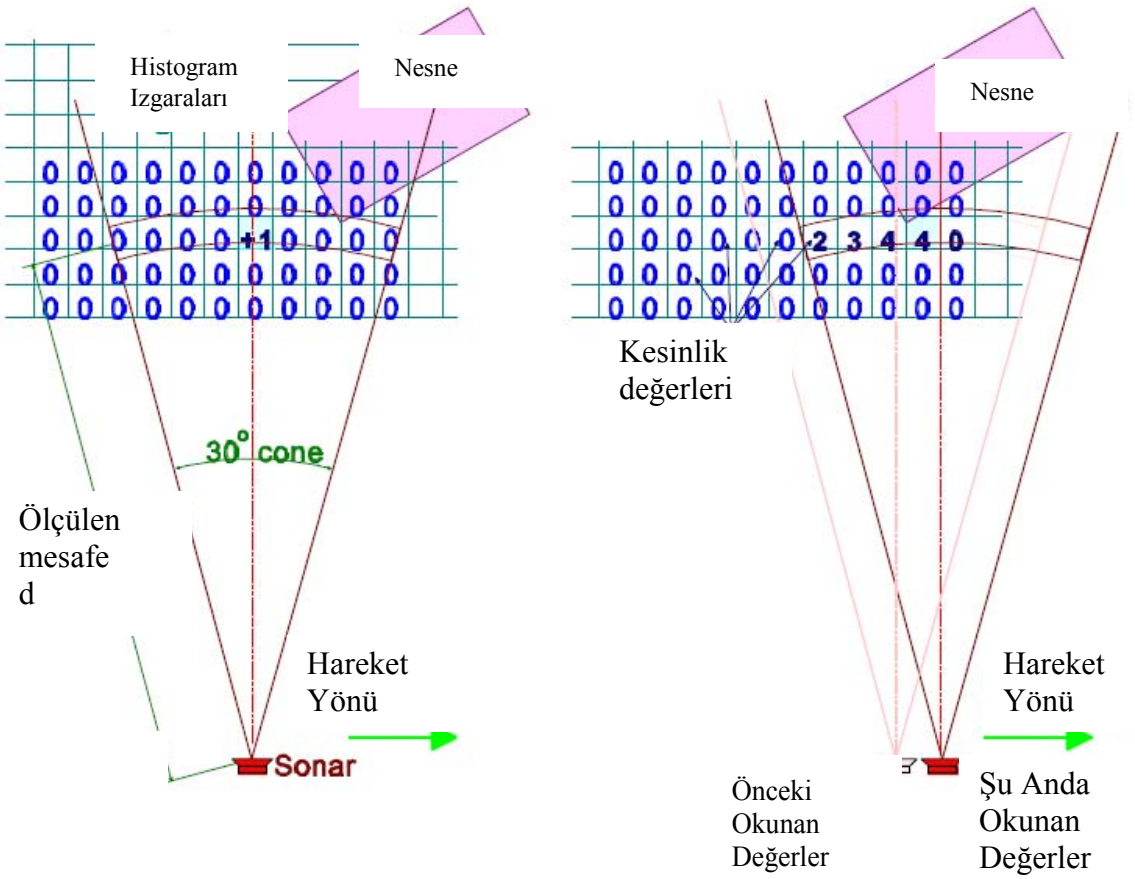
- $m(bilinmeyen) = \frac{\sum_{A_i \cap B_j = bilinmeyen} m(A_i)m(B_j)}{1 - \sum_{A_i \cap B_j = \phi} m(A_i)m(B_j)}$ olarak hesaplanır.

3.2.3 Hareket halinde histogramik harita çıkarma metodu

Gezgin robotun hareketi ile gerçek zamanlı harita çıkarabilmek için hareket halinde histogramik haritalama metodu Michigan Üniversitesinde Borenstein ve Koren (1991) tarafından geliştirilmiştir. Deneyler CARMEL robotunda denenmiştir. Robot 0.8 metre/sn maksimum hızla hareket ettirilmiştir. HIMM iki boyutlu histogramik ızgaralardan meydana gelir. Algılayıcılar tarafından alınan bilgiler doğrultusunda bu ızgaralar güncellenir. Robotun hızlı hareketinden dolayı ultrasonik algılayıcılardan alınan mesafe bilgileri sağlıklı olamayabilir. Bu nedenle hızlı harita çıkarma metodu genelde robotun engelden kaçınma davranışı için kullanılmaktadır. HIMM algoritması ile vektör alanı histogramlı engelden kaçınma algoritması bu çalışmada birleştirilmiştir. Diğer güncelleme yöntemlerindeki gibi bu yöntemde ikilik sistem kullanılmamaktadır, ızgaralar 0 ile 15 arasında değerler alır. Izgara boşsa I değeri -1 azalır. Izgara dolu ise I değeri +3 artar. (Şekil 3.9)

Genel formül;

- Izgara[i][j]=ızgara[i][j]+I $0 \leq$ ızgara[i][j] ≤ 15
- $I = \begin{cases} I^{+} dolu \\ I^{-} boş \end{cases}$



Şekil 3.9

a. Sadece bir hücre mesafe okumada artış göstermektedir. Ultrasonik mesafe algılayıcısı ile ölçülen d mesafesinde akustik eksende hücre bulunmaktadır.
b. Robot hareket halinde iken sürekli ve hızlı örneklenen algılayıcı değerleri için histogramik olasılık dağılımı elde edilir.

Histogramik ızgara metodunun dezavantajı sadece bir veya iki kez güncelleme yapılabilmesidir. Küçük cisimler ve delikler, hiçbir zaman sonara yüksek değerlilik döndürmezler.

Bayes ve Dempster Shafer biçimsel teorilerdir ve diğer algılayıcılar stereo veya lazer algılayıcı modelleri de bu yöntemle birleştirilebilir. HIMM ise sadece sonarlarla sınırlandırılmıştır.

4.ROBOT DAVRANIŞLARININ GELİŞTİRİLMESİ

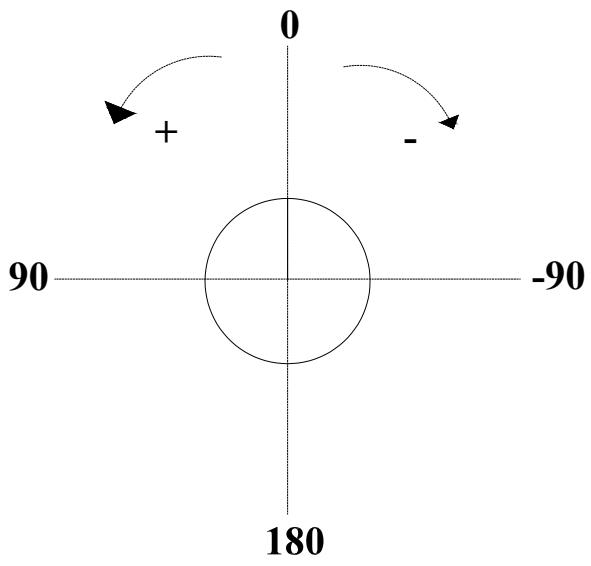
Robotun bulunduğu ortamın haritasını çıkarabilmesi için o ortamda güvenli bir şekilde dolaşabilmesi gereklidir. Bunun için bu çalışmada, gezgin bir robotun üzerindeki sonar mesafe algılayıcılardan aldığı bilgileri kullanarak bir bölgede en yakın duvarı bularak duvarı takip etme ve aynı zamanda engellerden kaçınma davranışı geliştirilmiştir. Geliştirilen davranış modeli PIONEER robotlar için tasarlanmış Mobilsim simülatörü ve P3 dx robotu ile test edilmiştir. Bu programda Aria Kütüphanesi destekli C++ komutları kullanılmıştır. Program Ek-1 de verilmektedir.

Program sekiz kısımdan oluşmaktadır. Bunlar;

1. Duvar Bulma Davranışı
2. Duvara Paralel Olma Davranışı (duvar takibi)
3. Öndeki Engelden Kaçınma (İçbükey dönüşleri) Davranışı:
4. Yandaki Engelden Kaçınma Davranışı
5. Dışbükey Köşe Dönüşleri
6. Tamponların Kontrolü
7. Tekerleğin Sıkışma Durumunda Kontrol
8. Harita Oluşturma
    - 8.1 Doluluk Izgara Metodu ile Olasılık Hesaplanması
    - 8.2 Pusulanın Kalibre Edilmesi
    - 8.3 Verileri Dosyaya Yazdırma

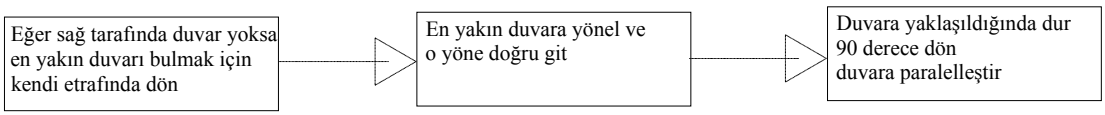
4.1 Duvar Bulma Davranışı

Robot davranışlarını geliştirmek için robotun polar koordinatlarından faydalanılır. Polar koordinatlar şekil 4.1'de verilmektedir.

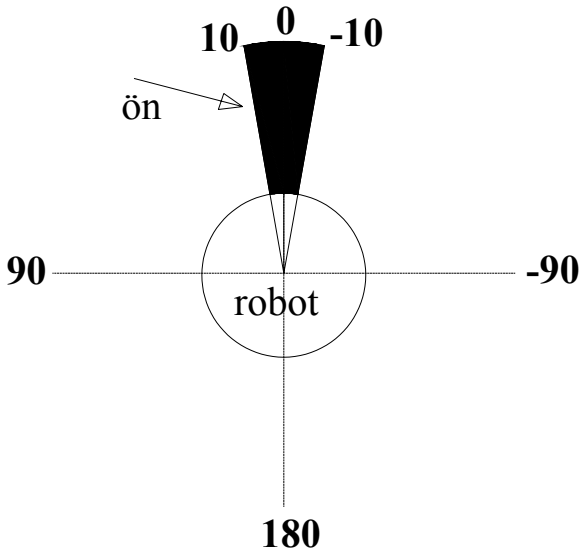


Şekil 4.1: P3 DX'in polar koordinatları

Duvar takibi için robotun öncelikle kendisine en yakın duvarı takip etmesi gerekmektedir. Bu nedenle eğer sağ tarafında 1000mm'ye kadar duvar yoksa robot duvar bulma davranışını gerçekleştirir.



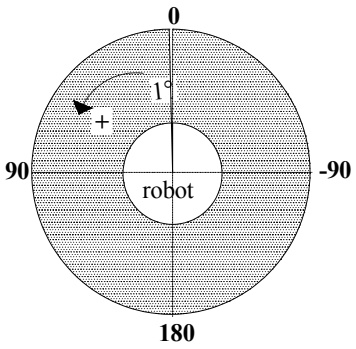
Robot saat yönüne ters yönde +5 derece açıyla döndürülür. -10º ve +10º lık polar koordinatları robotun ön kısmındaki mesafeyi göstermektedir.(Şekil 4.2)



Şekil 4.2. P3 DX'in -10º,+10º polar koordinatları

Bu polar koordinat bilgileri okunarak o bölgeye düşen sonarlardan en küçük mesafe bilgisi alınır. Bu bilgi robotun önündeki bölgeye düşen en küçük mesafe bilgisidir.

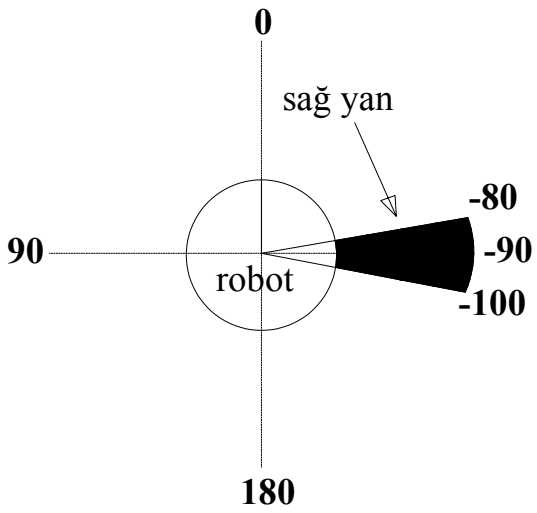
Robot kendi etrafında tüm bölgeyi tarayarak şekil 4.3 te görüldüğü gibi, en küçük mesafe bilgisini alır.



Şekil 4.3: p3dx'ın +1º den 0º' ye kadar polar koordinatları

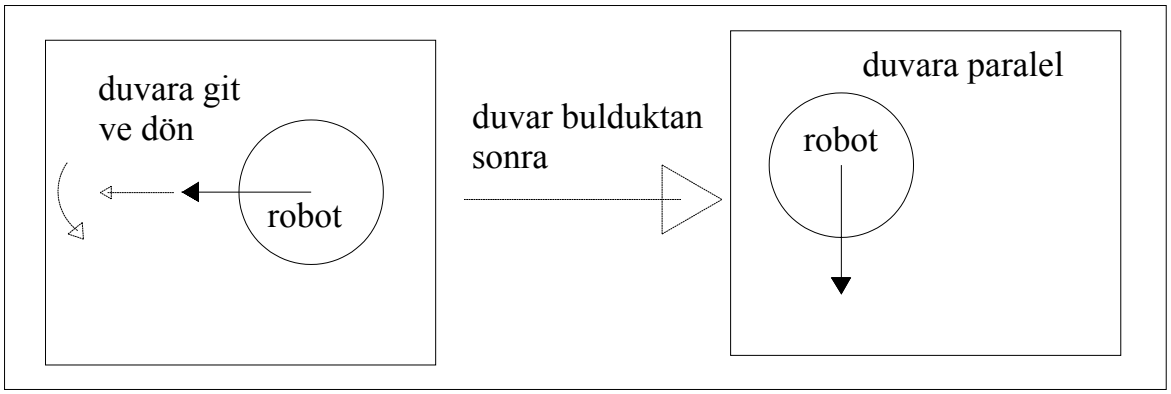
Okunan en küçük mesafe bilgisi, -10º - +10º polar bölgedeki en küçük mesafe bilgisine eşit olana kadar robotun +5º ile dönmesi sağlanır. En küçük mesafeye eşitlendiği zaman robotun dönme açısı sıfırlanır ve robot en yakın duvara doğru yönelir ve o yönde hareket eder.

Robotun hızı ön mesafe bilgisi ile ters orantılıdır. Mesafe kısaldıkça robotun hızı azaltılır. Maksimum hızı 200 mm/sn ile sınırlandırılmıştır. Robot duvara 400 mm yaklaşınca durur. -100º den -80º' ye kadar polar koordinatları sağ yandaki sonar mesafelerini göstermektedir (şekil 4.4). Sağ tarafında okunan mesafe bütün polar koordinatları tarayarak okunan en küçük mesafe bilgisine eşit olasıya kadar robot 90º ters saat yönünde döner. Böylece robot en küçük mesafede duvara paralel hale gelir.



Şekil 4.4: p3dx'ın -100º den -80º' ye kadar polar koordinatları

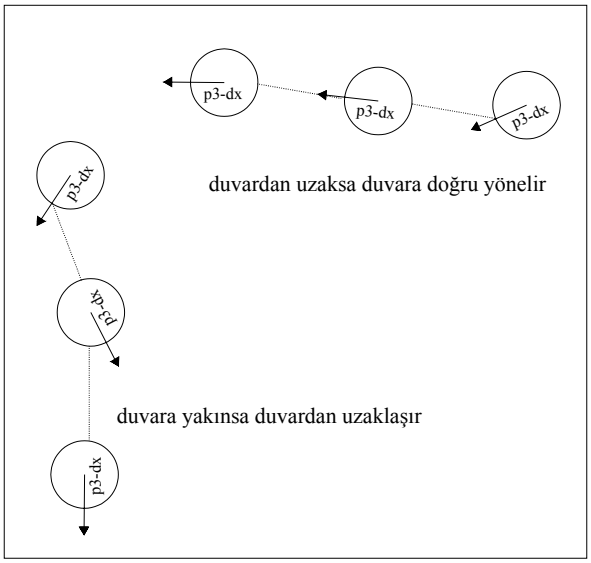
Duvar bulma davranışı gerçekleşir. Duvar bulma davranışı şekil 4.5 te gösterilmektedir.



Şekil 4.5: Duvar bulma davranışı

4.2 Duvara Paralel Olma Davranışı (Duvar Takibi)

Robot bir önceki davranışla duvarı bulduktan sonra duvara paralel olarak duvarı takip etmesi gerekmektedir (şekil 4.6).

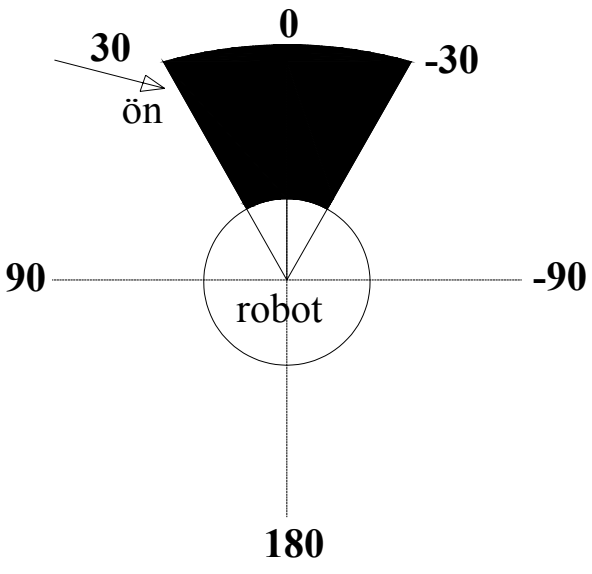


Şekil 4.6: Duvara paralel olma davranışı

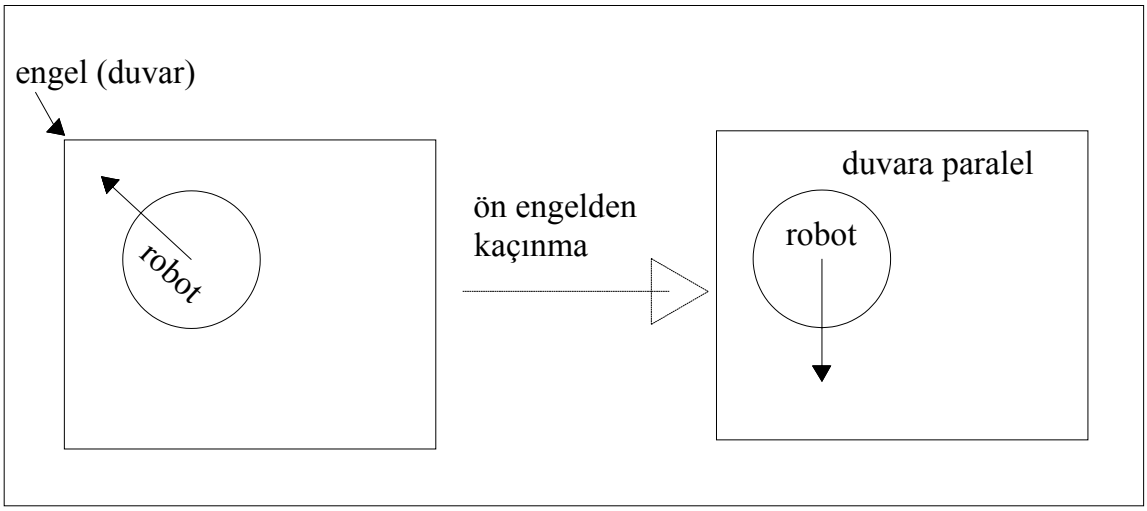
Sağ mesafe 350 mm ile 450 mm arasında ise robot istenilen eşik değerlerindedir. Bu değerlerde robot duvara en uygun mesafede bulunmaktadır. Bu mesafeler arasında ise sağ tarafında bulunan -90 derecedeki 7 ve 8 sonarlarının mesafeleri eşitlenmeye çalışılır. 7. ve 8. sonarın mesafe bilgisinin farkının ortalaması 5mm den küçükse robot duvara paralel düz gitmektedir. 7.sonarın okuduğu mesafe değeri, 8.sonardan büyükse -1 derece ile ters saat yönünde; 7.sonarın okuduğu mesafe değeri, 8.sonardan küçükse +1 derece ile saat yönünde dönerek paralelleştirme sağlanır. Sağ mesafe 350 mm'nin altında ise duvara çok yaklaşmıştır. +1 derece ile dışarı yönelir. Sağ mesafe 450 mm' nin üstünde 1000 mm 'nin altında ise uzaklaşma mesafesi ile doğru orantılı olarak robot içeri yönelir.

4.3 Öndeki Engelden Kaçınma (İçbükey Dönüşleri) Davranışı:

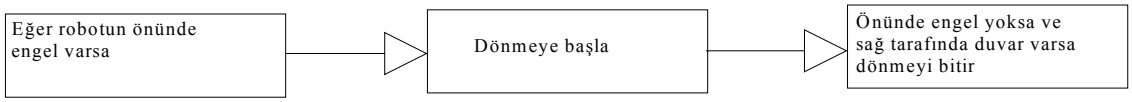
Robot, 400 mm mesafede -30º den +30º' ye kadar olan bölgede olan engellerden 90º ile dönerek kaçınır.(Şekil 4.7) 90 º içbükey köşelerden de engelden kaçınma davranışı ile döner.



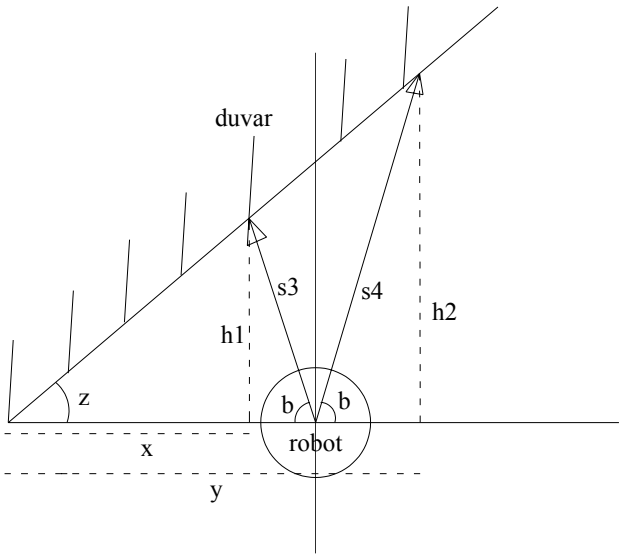
Şekil 4.7: p3dx'ın -30º den +30º' ye kadar polar koordinatları



Şekil 4.8: Ön Engelden Kaçınma Davranışı



Robotun önüne engel geldiğinde ne kadar dönmesi gerektiğini bulmak için ön tarafındaki mesafe algılayıcılarından 3. ve 4. sonar arasında ilişki kurulur. Şekil 4.9'da görüldüğü gibi b değeri sabittir. Bulmamız gereken değer z değeridir. Duvarın robota göre ne kadar eğimli olduğudur.



Şekil 4.9: Duvarın Robota göre Durumu

$$\sin(b) = \frac{h1}{s3} \tag{4.1}$$

$$\sin(b) = \frac{h2}{s4} \tag{4.2}$$

Benzerlik teoreminden,

$$\frac{h1}{h2} = \frac{x}{y} \tag{4.3}$$

Şekil 4.9 dan $y = x + s3 * \cos(b) + s4 * \cos(b) = x + (s3 + s4) * \cos(b)$ olarak hesaplanır. Hesaplanan değer denklem 4.3 te yerine konulursa, denklem 4.4. elde edilir.

$$\frac{h1}{h2} = \frac{x}{y} = \frac{x}{x + (s3 + s4) * \cos(b)} \tag{4.4}$$

Denklem 4.1 ve 4.2 , 4.4 denkleminde yerine konulursa,

- $\frac{s3 * \sin(b)}{s4 * \sin(b)} = \frac{x}{x + (s3 + s4) * \cos(b)}$
- $x * s3 + s3 * (s3 + s4) * \cos(b) = x * s3$
- $s3 * (s3 + s4) * \cos(b) = x * (s4 - s3)$

- $x = \dfrac{s3 * (s3 + s4) * \cos(b)}{(s4 - s3)}$

denklemleri elde edilir.

- $\tan(z) = \dfrac{h1}{x} = \dfrac{s3 * \sin(b)}{x} = \dfrac{s3 * \sin(b) * (s4 - s3)}{s3 * (s3 + s4) * \cos(b)}$

$$z = \tan^{-1}\left(\frac{s4 - s3}{s3 + s4} * \tan(b)\right) \tag{4.4}$$

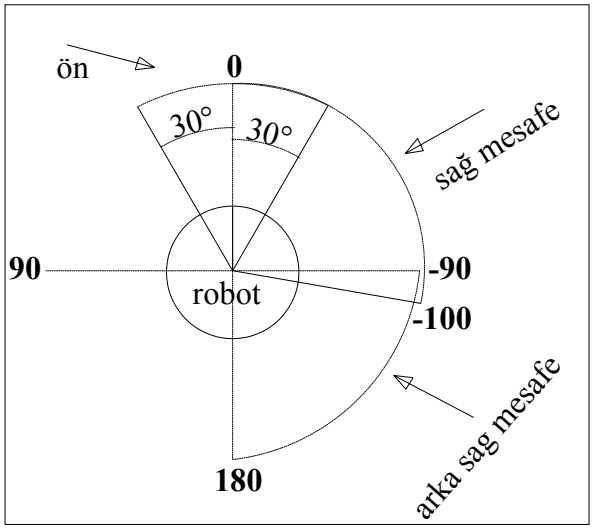
Denklem 4.4 den z değeri duvarın robotun eksenine göre açısı hesaplanır. Bulunan değer 90 dereceye eklenerek robotun duvara paralel hale gelmesi sağlanır.

4.4 Yandaki Engelden Kaçınma Davranışı:

Sol veya sağ mesafe 200 mm in altındaysa duvarlara veya herhangi bir nesneye çok yaklaşmıştır bu nedenle yandaki engelden kaçınma davranışı kullanılır. Sol mesafe 200 mm' den küçükse -5 º ile dönerek, sağ mesafe 200 mm' den küçükse +5º ile dönerek engellerden kaçınır.

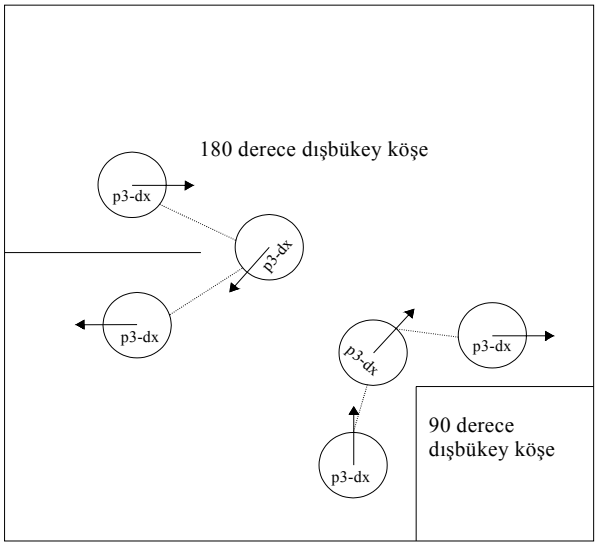
4.5 Dışbükey Köşe Dönüşleri

Sağ duvar takibinin en önemli problemlerinden biri dışbükey köşelerdir. Çünkü dışbükey köşelerde robot duvarı kaybedebilir. Bu nedenle tekrar duvar takibinin sağlanabilmesi için ön mesafe (-30º den +30 º ye polar koordinatlar) ve sağ mesafe (-100º den 0º ye polar koordinatlar) 500 mm den büyük ise ve arka sağ mesafe (-180º den -90º ye polar koordinatlar) 500 mm' den küçük ise dışbükey köşe olduğu tespit edilmiştir. (Şekil 4.10)



Şekil 4.10: p3-dx'ın polar koordinatları

Dışbükey köşelerde ortalama 180 derecelik dönme açısıyla dönerek duvarı bulması sağlanır. (Şekil 4.11)



Şekil 4.11: 90º ve 180º dışbükey köşe

4.6 Tampon Kontrolü

Sonarların göremeyeceği mesafede yani 15 cm' den yakın mesafede olan cisimlere karşı önlem olarak robotun arka tarafında bulunan tamponlar kullanıldı. Robotun arka tarafında 5 adet tampon bulunmaktadır. Robot geri geri giderken bir cisme çarparsa tamponlar bir sinyal üretmekte ve robotu durdurmaktadır. Hangi tampon sinyal üretiyorsa robot, o tamponun tersi yönünde az bir miktar ileri gitmekte ve kendini cisimden uzaklaştırmaktadır.

4.7. Tekerleğin Sıkışma Durumunda Kontrolü

Ön tarafında olabilecek çarpmalar için tekerin bir süre sıkışmasında kullanılan "ArActionRecover" komutuyla robotun biraz geri gelerek uygun olan yöne yönelmesi sağlandı.

4.8 Harita Oluşturma

4.8.1 Doluluk ızgara metodu ile olasılık hesaplanması

Uzaklığı r olan ve α açısındaki bir nesnenin sonar modeli olasılığını hesaplamak için Bölüm 3'te anlatıldığı gibi önce sonarın algılama aralığı bölgelere ayrılmıştır. 0-200 mm arası II. bölgeyi, 200-1500 mm arası I.bölgeyi, 1500-5000 mm arası III. bölgeyi oluşturmaktadır ve aşağıdaki parametreler kullanılarak olasılık hesaplaması yapılmıştır

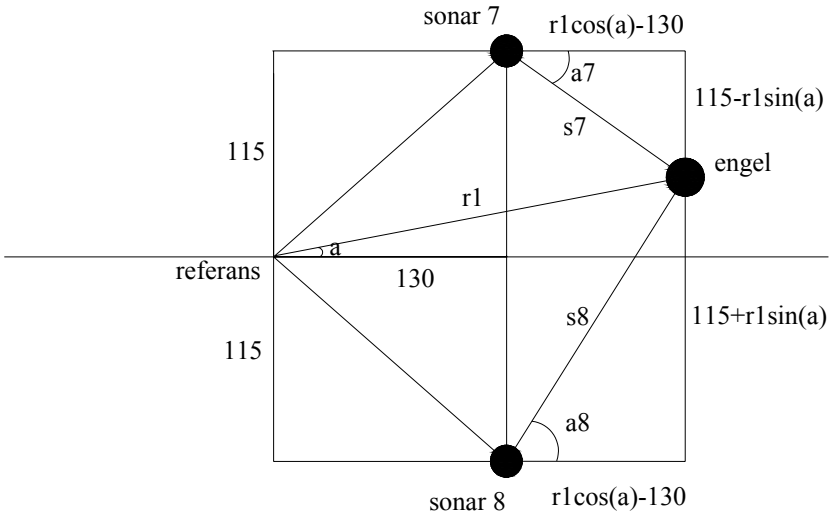
R → Sonarların maksimum okuma uzaklığı

r → Sonarın okuduğu mesafe

β → Sonarın görme açısının yarısı

α → Sonarın nesneye olan ışımasının açısı

R değeri simülatörde yapılan testler sonucu 5000 mm olarak hesaplanmıştır. β 15º alınmıştır. "Current reading polar" komutu ile referans noktasına en yakın engelin uzaklığı r1 ve açısı (a) belirlenmiştir (Şekil 4.12). Robot sağ duvar takibi davranışı gerçekleştirdiği için robotun sağ tarafında -75º ile -105º arasına gelen (7. ve 8.) sonarlar kullanılmıştır. Bu bölgedeki sonarların engele olan uzaklığı ve ışının hangi açıyla yansıdığı aşağıdaki denklemlerden hesaplanır.

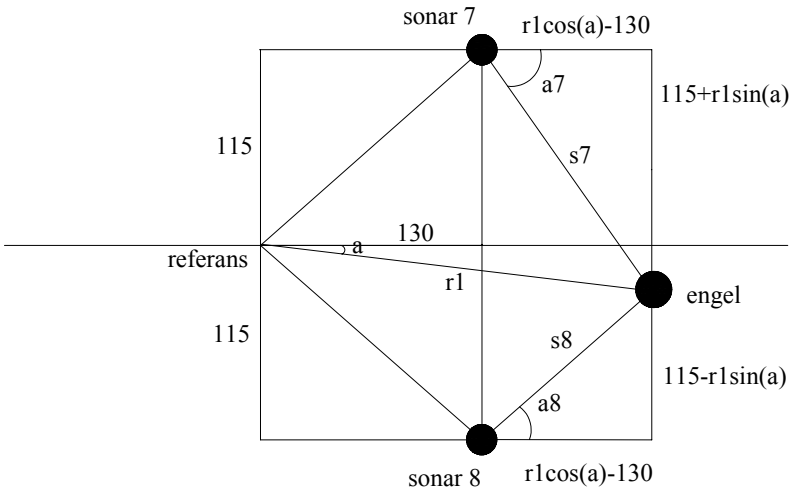


Şekil 4.12 Engel 7.sonara yakınsa sonar modeli

Engel 7.sonara yakınsa a7 değeri 4.5. denkleminden, a8 değeri denklem 4.6 dan elde edilir (Şekil 4.12).

$$a7 = \tan^{-1}\left(\frac{115 - r1\sin(a)}{r1\cos(a) - 130}\right) \tag{4.5}$$

$$a8 = \tan^{-1}\left(\frac{115 + r1\sin(a)}{r1\cos(a) - 130}\right) \tag{4.6}$$

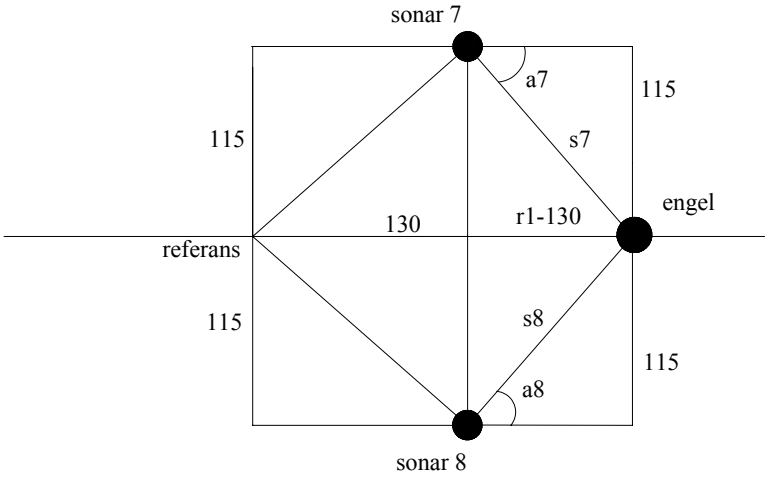


Şekil 4.13 Engel 8.sonara yakınsa sonar modeli

Engel 8.sonara yakınsa a7 değeri 4.7 denkleminden, a8 değeri denklem 4.8'den elde edilir (Şekil 4.13).

$$a7 = \tan^{-1}\left(\frac{115 + r1\sin(a)}{r1\cos(a) - 130}\right) \tag{4.7}$$

$$a8 = \tan^{-1}\left(\frac{115 - r1\sin(a)}{r1\cos(a) - 130}\right) \tag{4.8}$$



Şekil 4.14 Engel 7 ve 8.sonara eşit mesafede ise sonar modeli

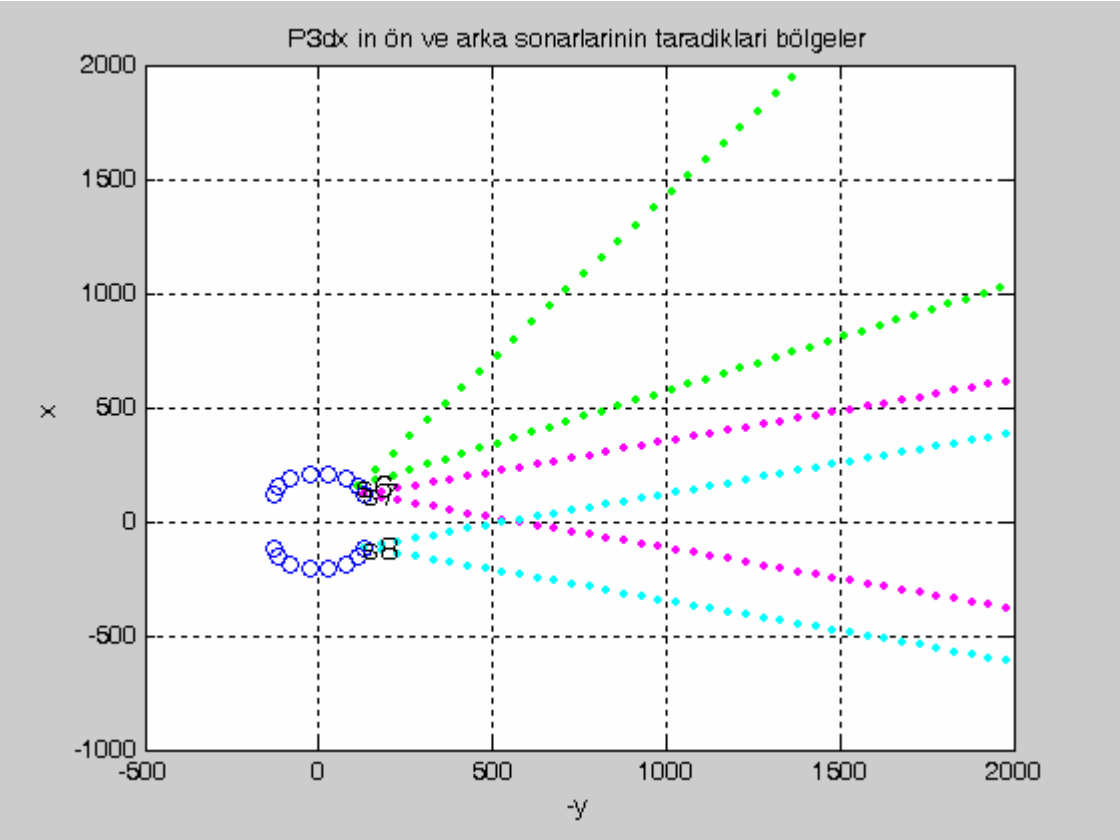
Engel 7. ve 8.sonara eşit mesafede ise a7 değeri 4.9 denkleminden, a8 değeri denklem 4.10'dan elde edilir (Şekil 4.14).

$$a7 = \tan^{-1}\left(\frac{115}{r1 - 130}\right) \tag{4.9}$$

$$a8 = \tan^{-1}\left(\frac{115}{r1 - 130}\right) \tag{4.10}$$

Üç durum içinde a7 ve a8 açıları hesaplandıktan sonra denklem 3.1'de yerine koyulmuş ve üç durum içinde olasılık fonksiyonu hesaplanmıştır. P olasılık fonksiyonumuz 0.5 ten büyükse engel olabileceği kabul edilmiştir. Doluluk ızgaraları olasılık modelini daha geliştirmek için Bayes teoremi ile güncelleme yapılmıştır.

Güncellemede amacımız eğer bir sonarın olasılığı hesaplanıp o noktada engel var kabul ediliyorsa ve aynı noktayı başka bir sonar da görüyorsa onun olasılığını da hesaba katmak daha güvenli sonuç verecektir. 6., 7. ve 8. sonarın ±15º derecelik açıda gördükleri bölgeler Şekil 4.15'te görülmektedir.



Şekil 4.15 Sonarların Okuma Bölgeleri

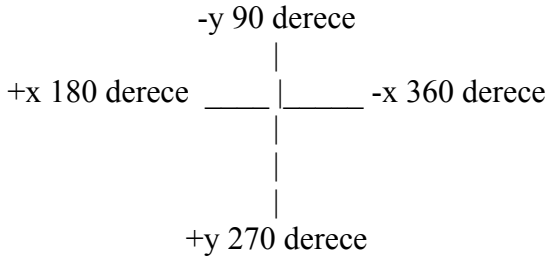
Şekil 4.15'te görüldüğü gibi 7. ve 8. sonarlar belirli bir mesafeden sonra ikisi de ortak bir alandaki engelleri görebilmektedir. 6.sonarda 7.sonara yakın olmasına rağmen ortak bir görüş alanı yoktur. Bu nedenle Bayes güncelleme hesaplamalarında 7. ve 8. sonar birlikte kullanıldı. Denklem 4.11'de doluluk ızgaraları olasılık fonksiyonu P hesaplanmaktadır.

$$P = \frac{P(7) \times P(8)}{P(7) \times P(8) + P'(7) \times P'(8)} \qquad (4.11)$$

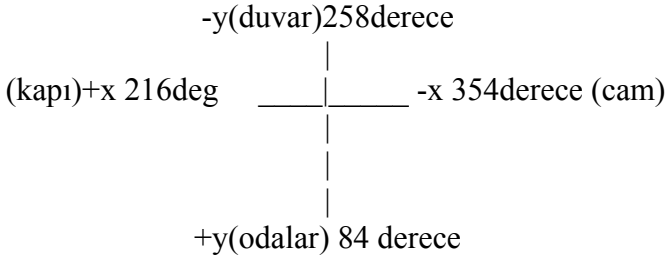
P(7), 7.sonarın dolu olma olasılığı, P(8), 8.sonarın dolu olma olasılığı, P'(7), 7.sonarın boş olma olasılığı, P'(8), 8.sonarın boş olma olasılığıdır. Hesaplanan olasılık değerleri 0.5 ten küçükse o ızgaranın boş olduğu, engel olmadığı düşünülür. 0.5 ten büyükse engel olma olasılığı yüksektir ve dolu olduğu düşünülür.

4.8.2 Pusulanın kalibre edilmesi

Pusula "getCompass" komutu ile 0 ile 360 derece arasında değerler okunur. Harita oluşturmak için güvenli pusula bilgisi değerlerine gerek vardır. İlk olarak Simulator ve P3 DX in pusula değerleri Aria' nın kendi demo programında, 4 bölgede robot 90'ar derece çevrilerek pusula değerleri tespit edilmiştir. Simülatörde ve robotta pusula değerleri şekil 4.16 ve şekil 4.17 da görüldüğü gibi belirlenmiştir.



Şekil 4.16 Simülatörde ölçülen pusula değerleri



Şekil 4.17 Robotta ölçülen pusula değerleri

Gerçek ortamda okunan pusula değerleri ile simülatör ortamında okunan pusula değerlerinin birbirinden farklılık gösterdiği tespit edilmiştir. Robotta, okunan değerlerde her bir bölüm 90 derecelik kısımlara ayrılması gerekirken ortamdaki manyetik alanlar nedeni ile sapmalar meydana gelmiştir. Bu sapmaları azaltmak için pusula kalibre edilerek 4 bölge eşit aralıklar haline getirilmiştir (Ek - 1).

Programda her bir aralığın 90 dereceye genişlemesi veya daralması sağlanmıştır. Ayrıca harita çıkarılırken de bu durum dikkate alınmıştır.

4.8.3 Verileri dosyaya yazdırma

Harita çıkarma işleminde sonarlardan aldığımız bilgi dışında olasılık fonksiyonumuz 0.5 ten büyük olduğu durumlarda kodlayıcıdan robotun pozisyon bilgileri alınır. Robotun x ve y pozisyon bilgisi, th yönlenme bilgisi ve pusula bilgisi elde edilir. Elde edilen bu bilgiler olasılık değeri, x pozisyon bilgisi, y pozisyon bilgisi, th, pusula, 7.sonar mesafe bilgisi, 8.sonar mesafe bilgisi, sonarın engeli gördüğü açı bir .txt dosyasına yazıldı. Bu .txt dosyası daha sonra MATLAB ta işlenerek harita çıkarıldı. İlgili MATLAB programı Ek-2 de verilmektedir.

5. ROBOT SİSTEMİ

5.1 Robotun Yapısı

P3-DX olarak adlandırılan Pioneer firması tarafından yapılan robotlar, hareketli, çok amaçlı olarak kullanılabilmektedir (http://robots.mobilerobots.com).

Şekil 5.1 P3 DX in önden görüntüsü

P3 DX Bileşenleri (şekil 5.1)

1. 3 adet akü
2. 2 adet tekerlek ve 1 adet döner küçük tekerlek
3. motor ve enkoder
4. sonar grubu
5. mikro kontroller
6. motor güç bölümü
7. pusula
8. tamponlar
9. tutucu
10. kamera

Robotun boyutları 44cm x 38cm x 22cm olup alüminyum gövdeye sahiptir. Gezgin robotun hareket edebilmesi için, motorlara bağlı robotun ön kısmına monte edilmiş çapı 16,5 cm olan 2 adet ve çapı 6 cm olan bir adet küçük tekerleği vardır. Pioneer 3-DX gezgin robotunun maksimum ulaşabileceği hız 1.6 m/s' dir.

Gezgin robotun hareketini sağlayan iki adet motor bulunmaktadır. Bu motorların bağlı olduğu iki adet 38.3:1 dişli oranına sahip olan dişli kutuları bulunmaktadır. Ayrıca robotun kendi konumunu belirlemesi için konum bilgisini oluşturan 500-delikli enkoder bulunmaktadır. Diferansiyel sürüş platformu ile iki tekerleği birlikte çevirir veya 32 cm yarıçapında duran tekerleğin etrafında dönme yapabilir. Arkadaki küçük tekerlek robotun dengesini sağlar. Robot %25 derece kalkarak 2.5 cm yüksekliğindeki eşiği atlayabilir. Gezgin robotun enerji ihtiyacını karşılamak için robotun içine monte edilmiş üç adet akü bulunmaktadır. Bunlara ilave olarak 180 derece önü çevreleyen 8 adet ön ultrasonik mesafe algılayıcısı ve 180 derece arkayı çevreleyen 8 adet arka ultrasonik mesafe algılayıcısı bulunmaktadır. Ultrasonik mesafe algılayıcıları yaklaşık 15 cm den 7m ye kadar okuyabilmektedir. Kullanılan robotun arka tarafında 5 adet tampon bulunmaktadır. 100 g. basınç uygulandığında bir giriş olarak algılanır. Tampon panelindeki her tamponun boyutu 10.0 cm x 2.5 cm x 1.0 cm dir.

Bilgisayar kullanıcısı ile iletişimi kablosuz modem ile sağlanır. Ayrıca ana makine ve gezgin robot arasında robota bazı komutları yükleyebilmek için TCP/IP protokolünü kullanan bir yazılım geliştirilmiştir

5.2 Kullanılan Yazılımlar

5.2.1 ARIA (http://robots.mobilerobots.com)

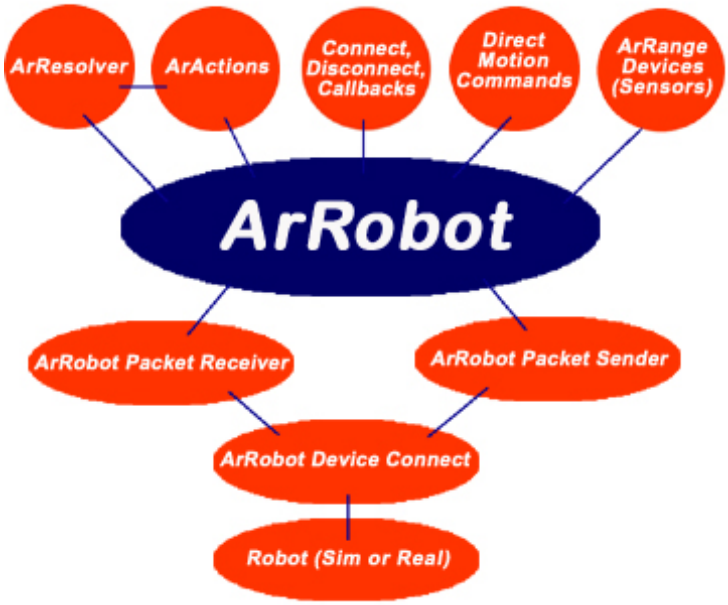
ARIA (Geliştirilmiş Robot Ara-Yüz Uygulaması) profesyonel geliştiriciler için tasarlanmıştır. ARIA, Linux veya Windows altında C++ komutları ile çalışabilen gezgin robotlar için nesneye dayalı etkili bir ara-yüzdür. ARIA robotla iletişimi alıcı/servis ilişkisi aracılığı ile, robotla haberleşecekse seri bağlantı ile, simülator ile haberleşecek ise TCP/IP bağlantı ile sağlar.(Şekil 5.2)

ARIA nın Yapısı:

ARIA esnek bir yapıya sahiptir. Tek davranışı gerçekleştirebildiği gibi bir çok davranışı da gerçekleştirebilmektedir. ARIA davranışların öncelik sırasına göre karar verir.

Anahtar Özellikler

- Hız, yönelme, bağıl yönelme dinamik olarak kontrol edilir.
- Kullanıcı Giriş / Çıkış yüzeyini, tutucuları, kamerayı sağa-sola, yukarı aşağıya çevirmeyi ve diğer aksesuarları bütünleştirir
- Bütün mesafe araçlarını tek bir fonksiyonda sorgular.
- Yüksek esnekliğe sahiptir
- Çapraz veritabanı kullanır (Linux ve Win 32)
- Kaynak dokümanları içinde mevcuttur.
- Çok yönlü uygulamalar için davranış sistemli planlama kurar.
- Engelden kaçınma için davranış oluşturabilirler



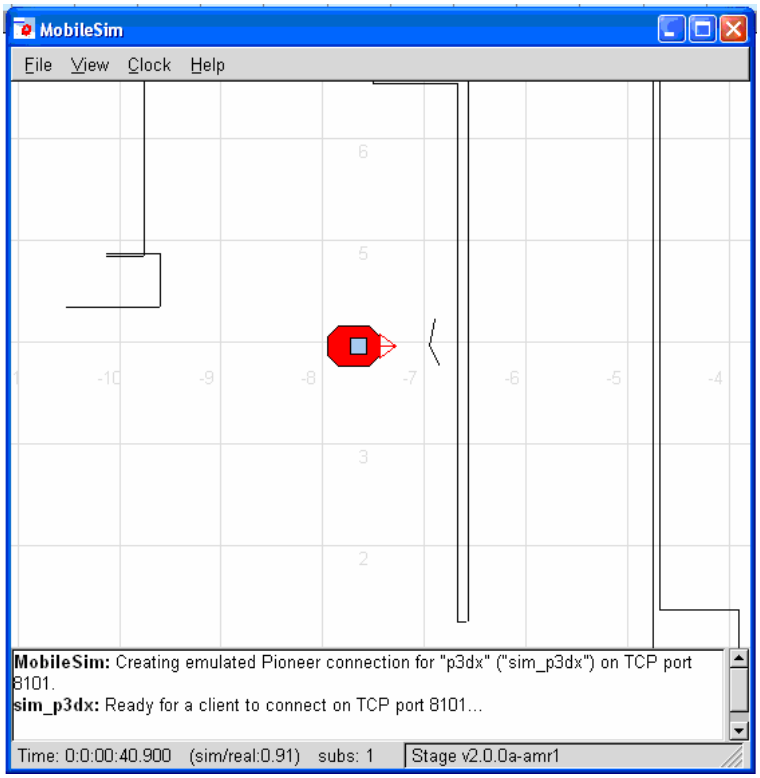
Şekil 5.2 ARIA' nın Çalışma Prensibi

Derleyici Gerekleri:

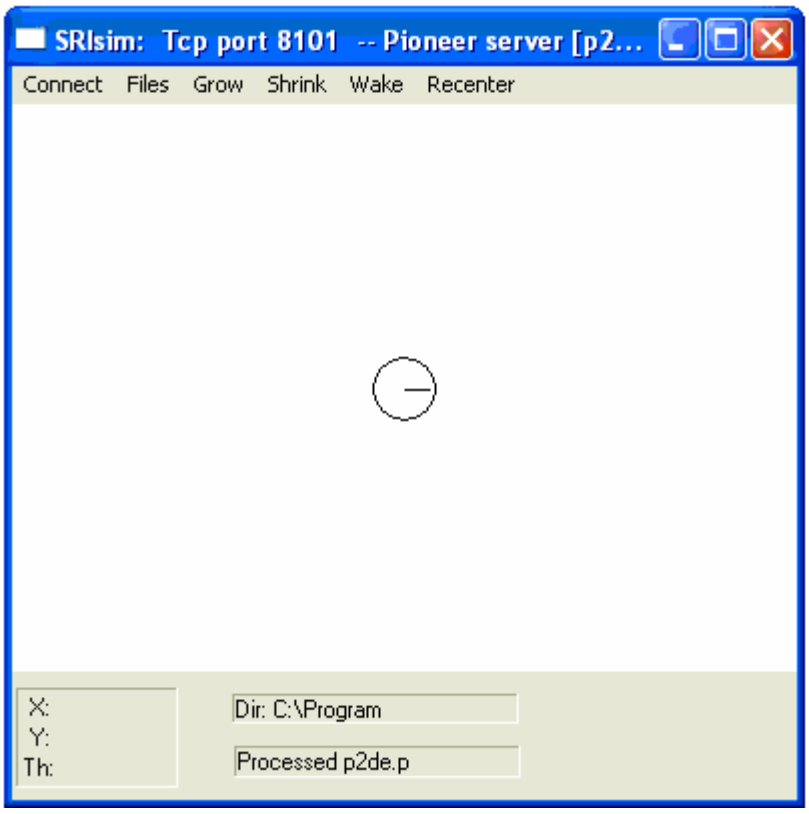
Windows kullanılıyorsa MS. Visual C++ .NET (7.1), Linux kullanılıyorsa G++ 3.x gereklidir.

5.2.2 Simulatör

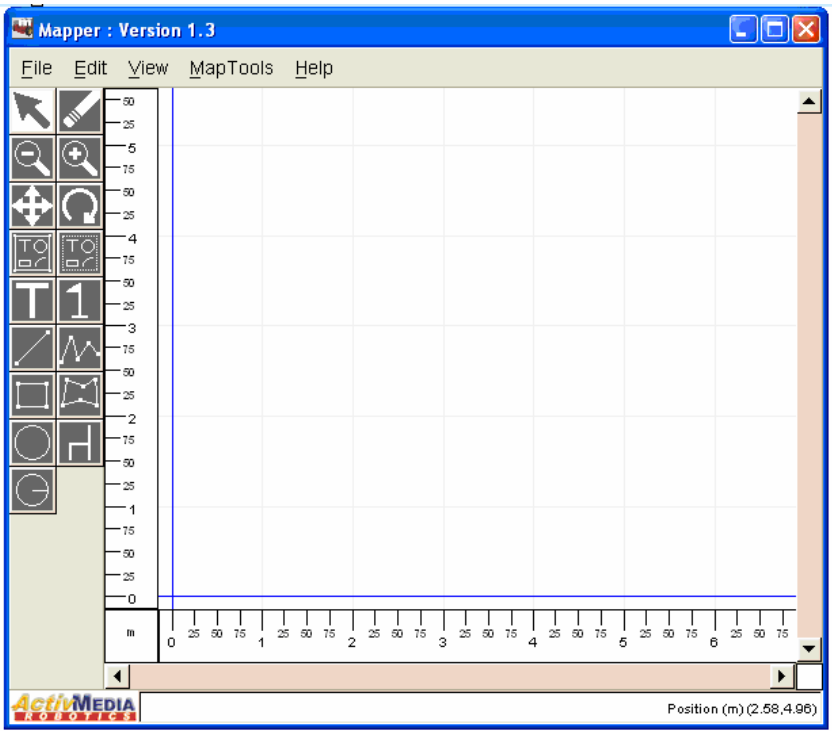
MobileSim programı ARIA ile gezgin robotları deneysel olarak bilgisayar ortamında denemek için kullanılır. SRIsim de aynı görevi yapmaktadır fakat MobileSim (Şekil 5.3) simulatörü, SRIsim (Şekil 5.4) e göre daha üstün özellikleri vardır. MobileSim, Richard Vaughan tarafından geliştirilen Stage simulatörü temeline dayanır. (2000) MobileSim çevredeki duvarları ve engelleri simule etmek için çizgi verilerini kullanır. Mapper (Şekil 5.5) veya Mapper3-Basic (Şekil 5.6) programı ile çizgi temelli harita oluşturulur. Mapper, SRIsim ile birlikte çalışıam bir çizim programıdır. Robotun çalışma ortamı için dünya modeli yaratılır. .wld dosyasındaki çizgi verilerini kullanır. Mapper3 – Basic ise MobileSim ile birlikte çalışmaktadır, .map dosyasındaki çizgi verilerini kullanmaktadır.



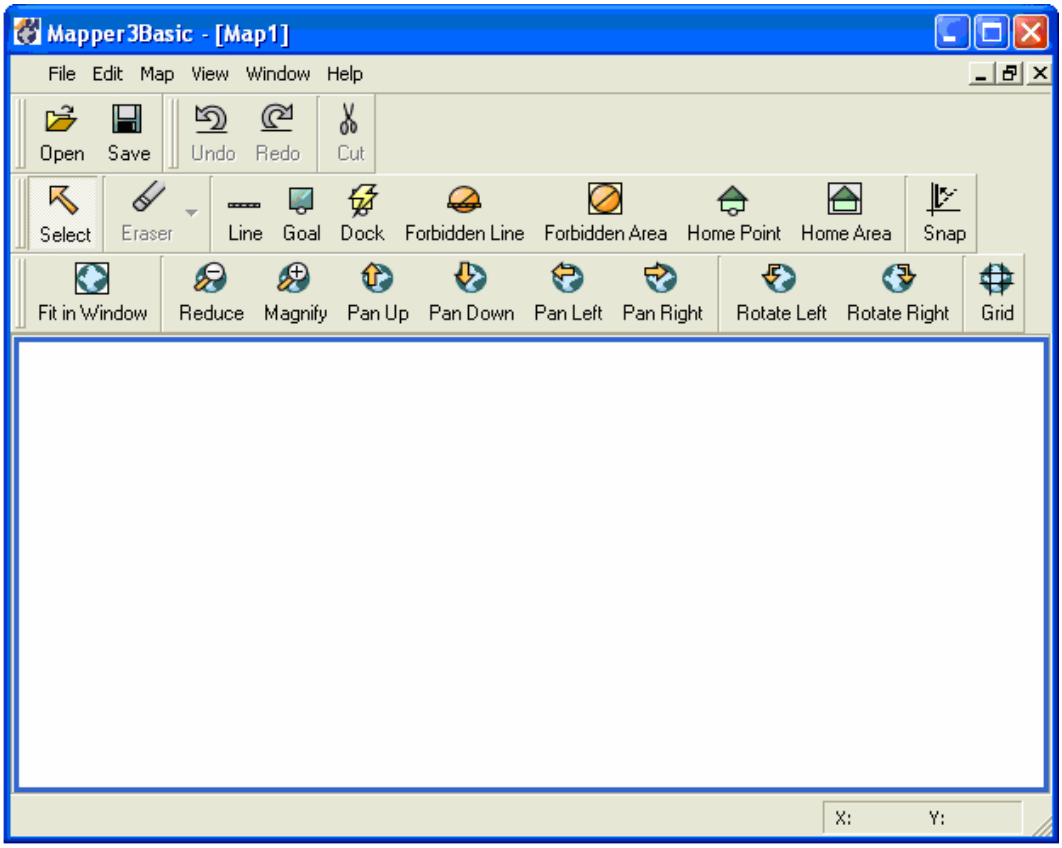
Şekil 5.3. MobileSim programının görüntüsü



Şekil 5.4. SRIsim programının görüntüsü



Şekil 5.5. Mapper programının görüntüsü



Şekil 5.6. Mapper3Basic programının görüntüsü

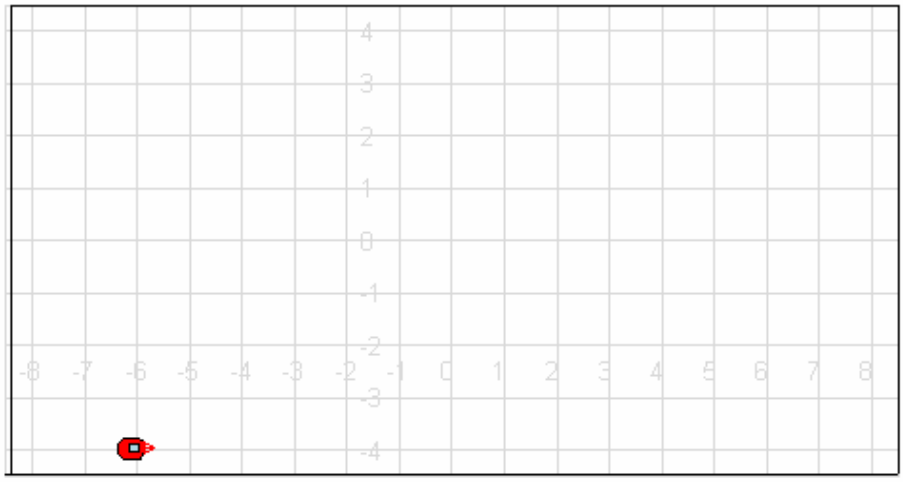
6. ROBOT DAVRANIŞLARI UYGULAMA SONUÇLARI

4. Bölümde anlatılan algoritma PIONEER robotlar için geliştirilmiş Mobilsim simülatöründe ve gerçek ortamda P3 dx robotu ile test edilmiştir.

6.1 Duvar Bulma ve Duvar Takibi Davranışı Uygulamaları

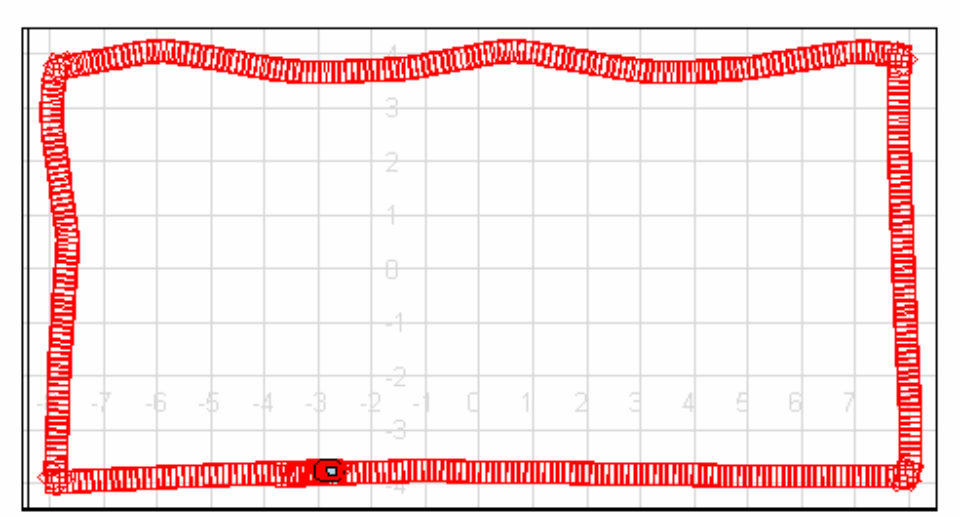
6.1.1 Simülatör testleri

1. durum: Robotun sağ tarafında duvar varsa;
Şekil 6.1'de robot için MobileSim simülatöründe çalışan Mapper3-Basic programı yardımı ile dikdörtgen bir alan çizilerek bir ortam hazırlanmıştır.
Bu durumda robot sağ tarafında duvar algıladığı için duvar bulma algoritması çalışmaz ve robot duvar takibi davranışını gerçekleştirir.



Şekil 6.1: Simülatörde oluşturulan dünya 1

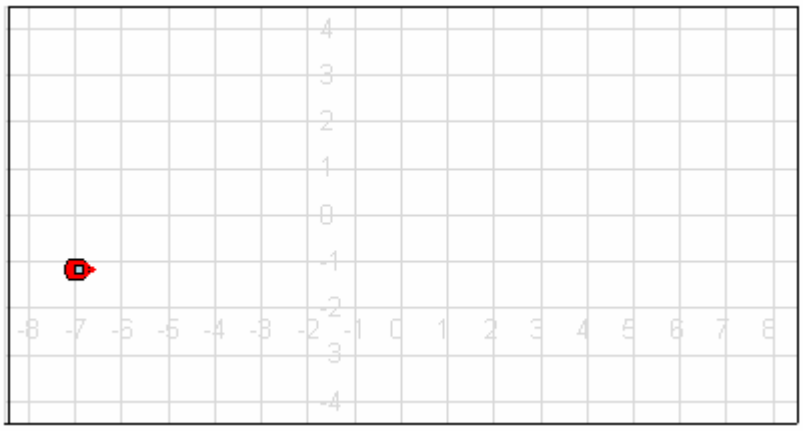
Robotun bütün ortamı dolaşması sağlanır. Robotun duvar takibi davranışı sonrası Şekil 6.2'de gösterilen harita elde edilir.



Şekil 6.2: Duvar takibi davranışı sonrası oluşan harita

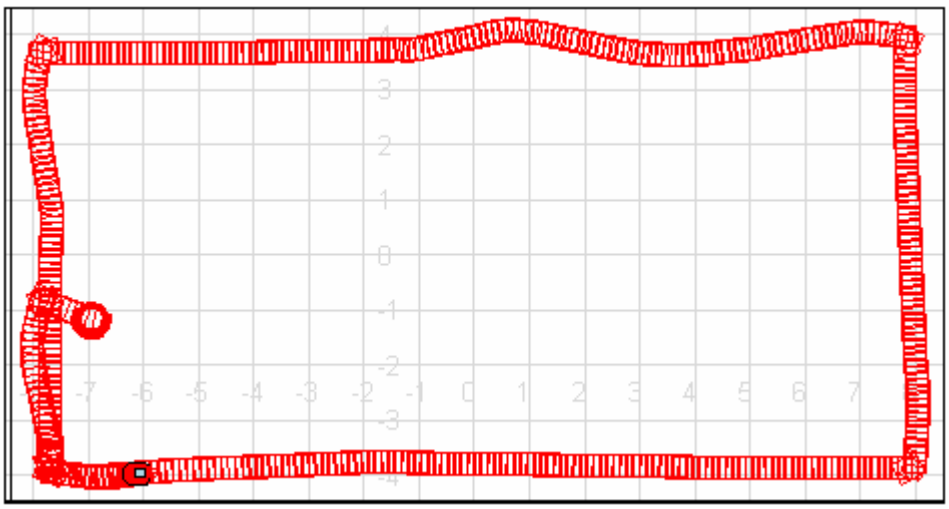
Şekil 6.2'den de görüldüğü gibi robot duvara kendini paralel hale getirmeye çalışarak duvarı takip eder. Robotun duvarı belirli bir mesafede takip etmesi istenmiştir. Robot duvara çok yaklaştığında bazı noktalarda uzaklaşma davranışını, uzaklaştığı noktalarda yaklaşma davranışında bulunmuştur. Bu ortamda engel olarak sadece duvarlar vardır. Köşelerde 90 derecelik içbükey dönüşleri gerçeklenmiştir.

2.durum: Robotun sağ tarafında duvar yoksa;
Şekil 6.3'de robot için MobileSim simülatöründe yeni bir ortam hazırlanmıştır.
Bu durumda robot sağ tarafında duvar algılamadığı için öncelikli olarak kendi etrafında dönerek duvar bulma davranışını gerçekleştirir.



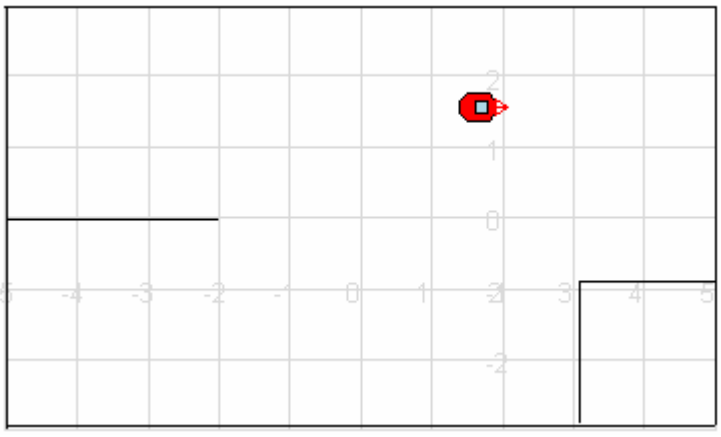
Şekil 6.3: Simülatörde oluşturulan dünya 2

Robot duvarı bulduktan sonra duvar takibi davranışını gerçekleştirir.(Şekil 6.4)



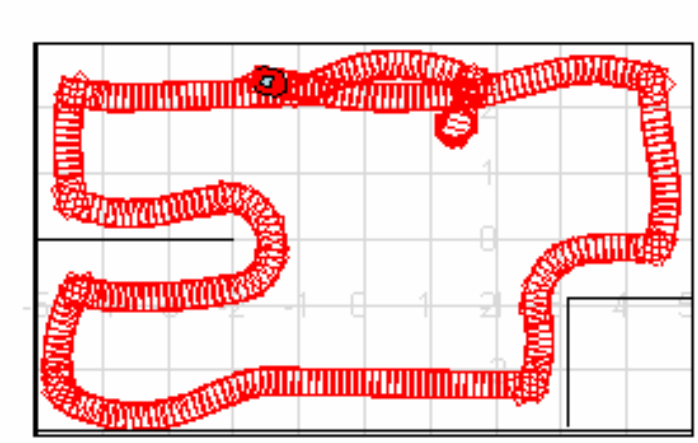
Şekil 6.4: Duvar bulma ve duvar takibi davranışı sonrası oluşan harita

3.durum: Robotun sağ tarafında duvar yok ve ortamda engeller varsa;
Şekil 6.5'den görüldüğü gibi ortamda engel olarak 180 derece ve 90 derece dışbükey köşeler vardır. Bu durumda robot öncelikle duvar bulma davranışını gerçekleştirecek daha sonra duvar takibine geçecektir. 180 derece ve 90 derece dışbükey köşelerde duvarı kaybetmemesi için dışbükey köşe dönüşleri davranışını kullanacaktır.



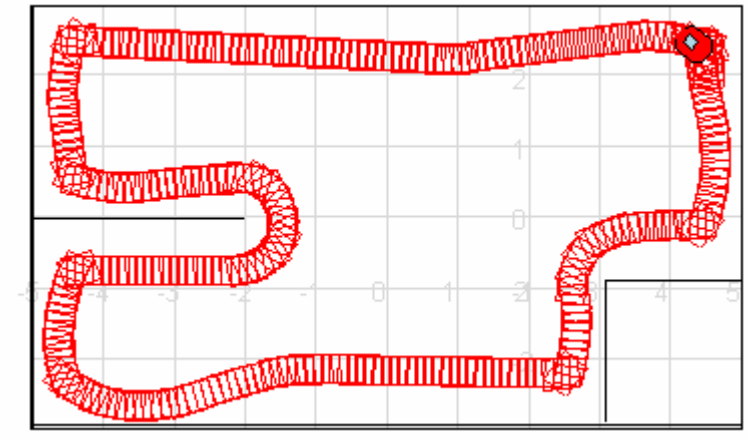
Şekil 6.5: Simülatörde oluşturulan dünya 3

Duvar bulma, duvar takibi ve dışbükey köşe dönüşleri davranışı sonrası oluşan harita Şekil 6.6'da gösterilmektedir.



Şekil 6.6: Duvar bulma, duvar takibi ve dışbükey köşe dönüşleri davranışı sonrası oluşan harita

Şekil 6.7'de ise duvar bulma davranışı yoktur. Robot duvara paralel giderken bir kesit alınmış ve aynı dünyada oluşan harita verilmiştir.



Şekil 6.7: Duvar takibi ve dışbükey köşe dönüşleri davranışı sonrası oluşan harita

6.1.2.Gerçek ortamda yapılan testler

Geliştirilen robot davranışları simülatör ortamında test edildikten sonra gerçek ortamda dinamik olarak da test edilmiştir. Laboratuarda karton kutulardan bir ortam oluşturulmuştur.

1.durum: robotun sağ tarafında duvar yoksa;

Şekil 6.8: Robot başlangıç konumunda

Şekil 6.9: Robotun en yakın duvarı aramaya başladığı durum

Şekil 6.10: Robotun en yakın duvarı bulduğu ve o yöne doğru gitmeye başladığı durum

Şekil 6.11: Robotun duvarı bulup kendini duvara paralel hale getirdiği durum

Yukarıdaki şekillerden de görüldüğü gibi robot duvar bulma davranışını gerçekleştirdikten sonra duvarı takip etmeye başlamıştır.

2.durum: Ortamda 90 derece dışbükey köşe varsa;

Şekil 6.12: Robotun duvara paralel halde duvar takip ettiği durum

Şekil 6.13: Robotun 90 derecelik köşeye yaklaştığı durum

Şekil 6.14: Robotun 90 derecelik köşeyi dönmeye başladığı durum

Şekil 6.15: Robotun 90 derecelik köşeyi dönmeyi tamamladığı durum

Yukarıdaki şekillerden de görüldüğü gibi robot 90 derecelik dışbükey köşeyi duvarı kaybetmeden dönmüştür.

3.durum: Ortamda 180 derece dışbükey köşe varsa;

Şekil 6.16: Robotun 180 derecelik dışbükey köşeye yaklaştığı durum

Şekil 6.17: Robotun 180 derecelik dışbükey köşeyi dönmeye başladığı durum

Şekil 6.18: Robotun 180 derecelik köşeyi dönmeyi tamamladığı durum

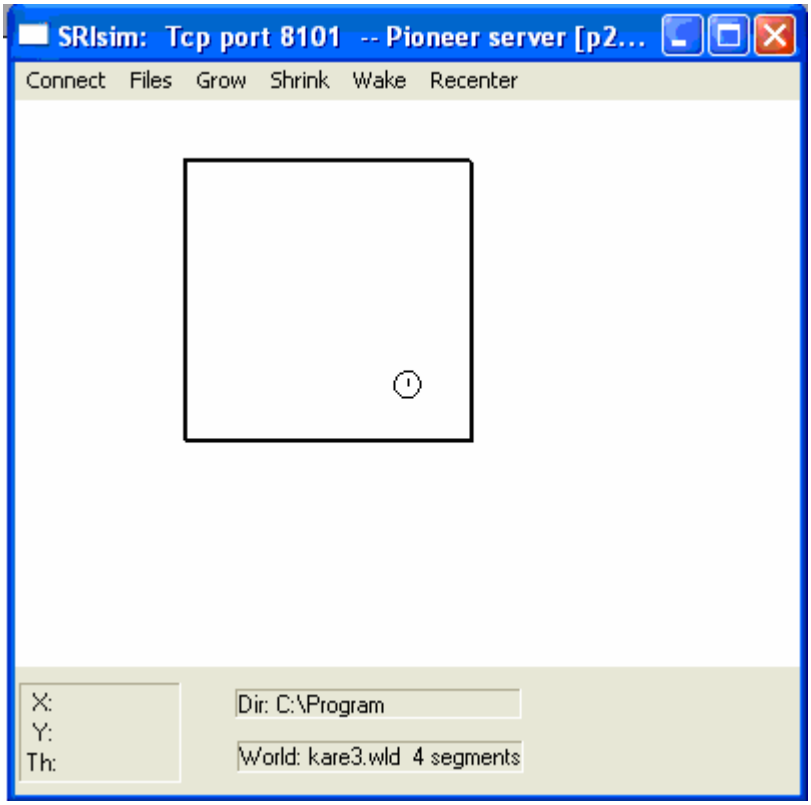
Yukarıdaki şekillerden de görüldüğü gibi robot 180 derecelik dışbükey köşeyi duvarı kaybetmeden dönmüştür.

6.2 Harita Çıkarma Uygulamaları

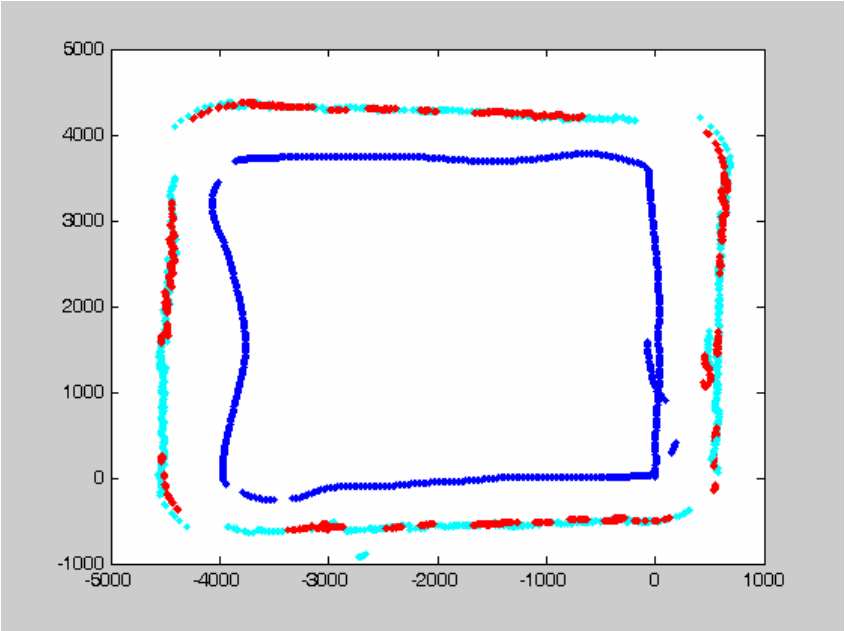
Ultarasonik mesafe algılayıcılar kullanılarak doluluk ızgaraları metodu ile robotun bulunduğu çevrenin haritası çıkarılmıştır. Harita çıkarma uygulamasında simülatör testlerinde SRIsim kullanılmıştır. Bunun sebebi MobileSim simülatörünün pusula bilgisini yanlış okumasıdır.

6.2.1 Simülatör testleri

Şekil 6.19 da simülatör ortamında robot için oluşturulan dünya görülmektedir. Robot duvar takibi davranışını gerçekleştirir ve Bayes güncellemeli doluluk ızgaraları metodu kullanılarak ortamın haritası oluşturulur. Oluşan harita Şekil 6.20 de gösterilmektedir. Mavi çizgi enkoderden alından bilgiye göre robotun gittiği yoldur. Kırmızı ve yeşil çizgiler duvarları göstermektedir.



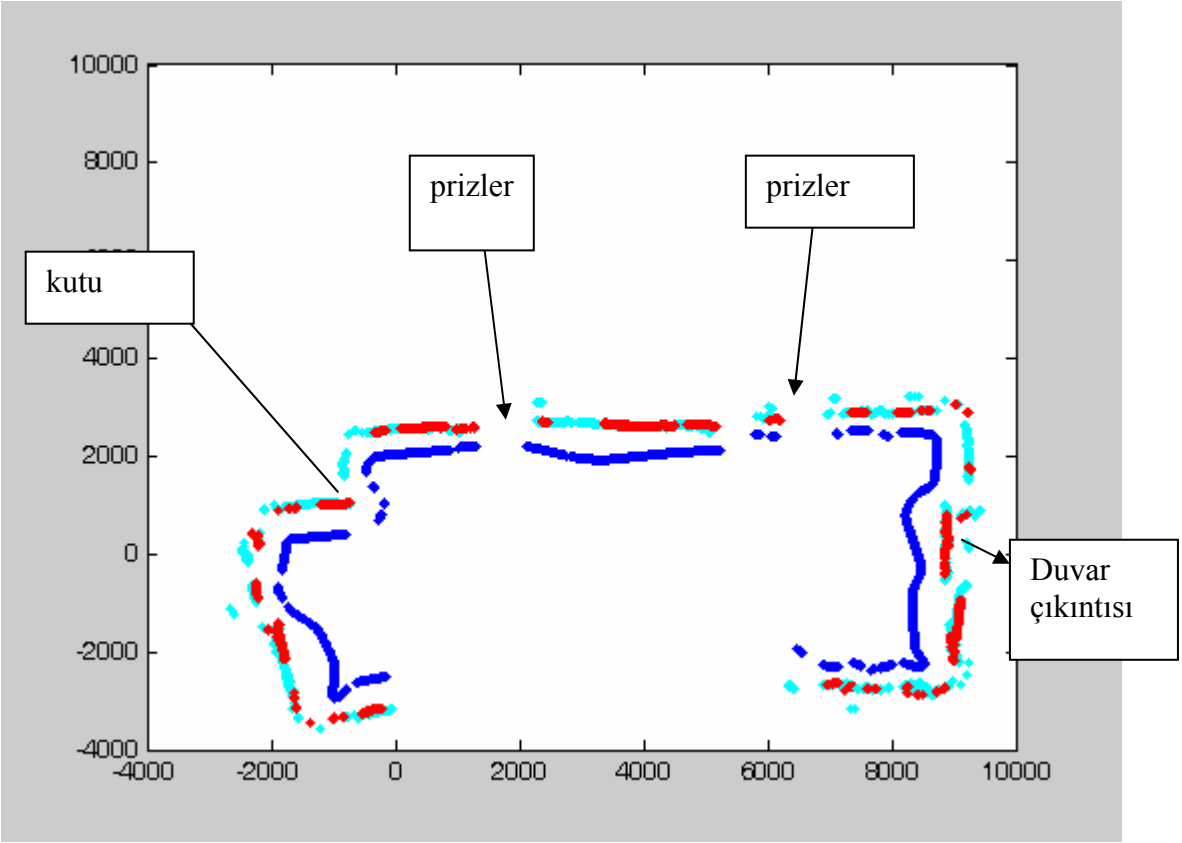
Şekil 6.19: Simülatör ortamında oluşturulan dünya



Şekil 6.20: Duvar takibi davranışı sonunda oluşturulan harita

6.2.2.Gerçek ortamda yapılan testler

Geliştirilen robot davranışları simülatör ortamında test edildikten sonra gerçek ortamda dinamik olarak da test edilmiştir. Simülatörde uygulanan çalışmadan farklı olarak pusula bilgisi kalibre edilmeye çalışılmıştır. Şekil 6.21'de laboratuar ortamında oluşan harita gösterilmektedir.



Şekil 6.21: Laboratuar ortamında oluşturulan harita

Gerek simülatör de gerekse gerçek ortamda yapılan testler sonucu şekillerde bazı yerlerde kesiklikler gözlenmiştir. Bunun sebebi harita çıkarma işlemi yapılırken yanlış okuma değerlerini azaltmak için olasılık değerinin 0.95'ten büyük ızgaraların çizdirilmesidir. Dolu olma koşulu 0,5'ten 0.95 'e çıkarıldığı içinde arada kalan bazı dolu ızgaralar boş olarak çıkmıştır.

7. SONUÇLAR VE ÖNERİLER

Bu çalışmada çevre haritalama ve robot davranışlarının kontrolü işlemleri ARIA simülatörü ve gerçek ortamda P3 dx robotunda başarılı bir şekilde uygulanmıştır.

Günümüzde birçok alanda robotik sistemler üzerinde çevre haritalama işlemleri kullanılmaktadır. Bu işlemlerin uygulandığı robotların özellikle kesintisiz, hızlı ve doğru sonuçlar verebilmesi robotların kendi aralarındaki önemli tercih sebeplerinden biridir. Robotların kesintisiz, hızlı ve doğru cevap verebilmelerini sağlayan en önemli faktörlerden biri robotun donanımı ve robot üzerinde çalıştırılan yazılımdır. Günümüzde algılayıcıların, motor sistemlerinin, mikroişlemcilerin çok hızlı geliştiği bir ortamda bu donanımların hepsini yönetecek olan yazılımlarında aynı kalitede olması istenmektedir.

P3 dx robotu için hazırlanan yazılımda robot davranışlarının kontrolü ile robotun ilk amacı engellere çarpmadan ortamda gezinmesi olmuştur. Bu davranış ile birlikte duvar bulma ve duvarı kaybetmeden, duvara kendini paralel hale getirerek duvar takibi yapılması sağlanmıştır. Ultrasonik mesafe algılayıcıları kullanılarak Bayes güncellemeli doluluk ızgaraları metodu ile ortam haritasının çıkarılması sağlanmıştır.

Gerçek ortamda yapılan testlerde pusula bilgisinin sağlıklı alınamamasında dolayı ve harita çıkarma işlemi robotun kodlayıcıdan alınan bilgisine göre oluşturulduğundan haritada kaymalar meydana geldiği görülmüştür.

Harita çıkarma işleminin daha başarılı olabilmesi için yapay veya doğal işaretlerle, lazer mesafe algılayıcı veya kamera kullanılarak daha yüksek hassasiyette lokalizasyon yapılması robotun konumunun daha doğru belirlenmesini sağlayacaktır. Birden çok algılayıcı bilgisinin bir arada kullanılarak harita çıkarmanın daha güvenli sonuçlar vereceği düşünülmektedir.

8. KAYNAKLAR DİZİNİ

Borenstein, J., Koren, Y., 1991, Histogramic in – motion mapping for mobile robot obstacle avoidance, IEEE Journal of Robotics and Automation, Vol 7 , No.4, 535-539

Bozma, Ö., Kuc R., 1991, Single sensor sonar map-building based on physical principles of reflection, IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. 13; No. 12

Davison, A.J. and Murray D.W., 1998, Mobile robot localization using active vision, In Proceedings of the 5th European Conference on Computer Vision, Freiburg, 809-825

Graff, K. F., 1981, A History of Ultrasonics, Chapter 1 of Physical Acoustics, Vol. 15, Mason and Thurston, editors, Academic Press

Guivant, J., Nebot, E., Baiker, S., 2000, Autonomous navigation and map building using laser range sensors in outdoor application, Journal of Robotic Systems, Vol 17, No. 10, 565 - 583

Konolige, K., 1997, Improved occupancy grid for map building, Autonomous Robots, 351-367

Matthies, L., Elfes, A. 1988, Integration of sonar and stereo range data using a grid-based representation, Proceedings of the 1988 IEEE International Conference on Robotics and Automation, Philadelphia, Pennsylvania, USA.

Min, B.K., Cho, D.W., Lee, J.S. and Park, Y. P., 1997, sonar mapping of a mobile robot considering position uncertainity, Robotics & Computer Integrated Manufacturing, Vol. 13, No.1, 41-49

Moravec, H. P., Elfes, A., 1985, High resolution maps from wide angle sonar, Proceedings of the 1985 IEEE International Conference on Robotics and Automation.

Murphy, R., 2000, Introduction to AI Robotics, MIT Press, London

O'Sullivan S., 2003, An emprical evaluation of map building methodologies in mobile robotics using the feature prediction sonar noise filter and metric grid map benchmarkig suite, Master of Science University of Lumerick

Thrun, S., 1993, Exploration and model building in mobile robot domains, In Proceedings of IEEE International Conference on Neural Network, Seattle, Washington,USA, IEEE Neural Network Council, 175-180

KAYNAKLAR DİZİNİ (devam)

Thrun, S., Fox, D. and Burgard, W., 1998, A probabilistic approach to concurrent mapping and localization for mobile robots, Machine Learning, Vol. 31

Vaughan, R., 2000, Stage: a multiple robot Simulator, Technical report, Institute for Robotics and Intelligent Systems IRIS-00-393, School of Engineering, University of Southern California

Zunino,G., 2002, Simultaneous localization and mapping for navigation in realistic environments, Royal Institue of Technology, Stockholm

www.bilmuh.gyte.edu.tr

http://robot.cmpe.boun.edu.tr

http://robots.mobilerobots.com

```
/* Ek-1
Yazan:Elif Eroğlu
Tarih:26/02/2006
*/
#include "Aria.h"
#include <stdio.h>

class ActionAvoidFront : public ArAction
{
public:
  // constructor, sets the obstacledistance, and turnAmount
  ActionAvoidFront(double obstacledistance,double TurnAmount);
  // destructor, its just empty, we don't need to  anything
  virtual ~ActionAvoidFront(void) {};
  // fire, this is what the resolver calls to figure out what this action wants
  virtual ArActionDesired *fire(ArActionDesired currentDesired);
  // sets the robot pointer, also gets the sonar device
  virtual void setRobot(ArRobot *robot);
protected:
  // this is to hold the sonar device form the robot
  ArRangeDevice *mySonar;
  // what the action wants to do
  ArActionDesired myDesired;
  double myobstacledistance;
  double myTurnAmount;
  // value ot hold onto so turns are smooth, which direction its turning
  int myTurning3;// -1 == left, 1 == right, 0 == none
  };
class ActionAvoidSide : public ArAction
{
public:
  // constructor, sets the startdistance, and turnAmount
  ActionAvoidSide(double startdistance,double TurnAmount);
  // destructor, its just empty, we don't need to do anything
  virtual ~ActionAvoidSide(void) {};
  // fire, this is what the resolver calls to figure out what this action wants
  virtual ArActionDesired *fire(ArActionDesired currentDesired);
  // sets the robot pointer, also gets the sonar device
  virtual void setRobot(ArRobot *robot);
protected:
  // this is to hold the sonar device form the robot
  ArRangeDevice *mySonar;
  // what the action wants to do
  ArActionDesired myDesired;
  double mystartdistance;
  double myTurnAmount;
  // value ot hold onto so turns are smooth, which direction its turning
```

```
  int myTurning2;// -1 == left, 1 == right, 0 == none
  };
class ActionFindWall : public ArAction
{
public:
  // constructor, sets the maxspeed, and stopdistance
  ActionFindWall(double maxSpeed, double stopDistance);
  // destructor, its just empty, we don't need to do anything
  virtual ~ActionFindWall(void) {};
  // fire, this is what the resolver calls to figure out what this action wants
  virtual ArActionDesired *fire(ArActionDesired currentDesired);
  // sets the robot pointer, also gets the sonar device
  virtual void setRobot(ArRobot *robot);
protected:
  // this is to hold the sonar device form the robot
  ArRangeDevice *mySonar;
  // what the action wants to do
  ArActionDesired myDesired;
  double myMaxSpeed;
  double myStopDistance;
  // value ot hold onto so turns are smooth, which direction its turning
  int myTurning;// -1 == left, 1 == right, 0 == none

  };
class ActionParallel : public ArAction
{
public:
  // constructor, sets the threshdistance1,threshdistance2,turnAmount and maxspeed
  ActionParallel(double threshdistance1,double threshdistance2,double
TurnAmount,double maxSpeed);
  // destructor, its just empty, we don't need to do anything
  virtual ~ActionParallel(void) {};
  // fire, this is what the resolver calls to figure out what this action wants
  virtual ArActionDesired *fire(ArActionDesired currentDesired);
  // sets the robot pointer, also gets the sonar device
  virtual void setRobot(ArRobot *robot);

protected:
  // this is to hold the sonar device form the robot
  ArRangeDevice *mySonar;
  // what the action wants to do
  ArActionDesired myDesired;
  double mythreshdistance1;
  double mythreshdistance2;
  double myTurnAmount;
  double myMaxSpeed;
  int SonarArray[16] ;
```

```
  int Right1S,Right2S,Left1S,Left2S,right ;
  int RightDiffS,LeftDiffS ;
  int i;
  // value ot hold onto so turns are smooth, which direction its turning
  double myTurning4;// -1 == left, 1 == right, 0 == none


  };

class ActionTurnCorner : public ArAction
{
public:
  // constructor, sets the distance and turnAmount
  ActionTurnCorner(double distance,double TurnAmount);
  // destructor, its just empty, we don't need to do anything
  virtual ~ActionTurnCorner(void) {};
  // fire, this is what the resolver calls to figure out what this action wants
  virtual ArActionDesired *fire(ArActionDesired currentDesired);
  // sets the robot pointer, also gets the sonar device
  virtual void setRobot(ArRobot *robot);
protected:
  // this is to hold the sonar device form the robot
  ArRangeDevice *mySonar;
  // what the action wants to do
  ArActionDesired myDesired;
  double mydistance;
  double myTurnAmount;
  // value ot hold onto so turns are smooth, which direction its turning
  int myTurning;// -1 == left, 1 == right, 0 == none
  };
class ActionDraw : public ArAction
{
public:
  // constructor, sets the obstacledistance, and turnAmount
  ActionDraw(double threshdistance1,double threshdistance2);
  // destructor, its just empty, we don't need to do anything
  virtual ~ActionDraw(void) {};
  // fire, this is what the resolver calls to figure out what this action wants
  virtual ArActionDesired *fire(ArActionDesired currentDesired);
  // sets the robot pointer, also gets the sonar device
  virtual void setRobot(ArRobot *robot);
  void  setComp(double a){t=a;}
protected:
  // this is to hold the sonar device form the robot
  ArRangeDevice *mySonar;
  // what the action wants to do
  ArActionDesired myDesired;
```

```
  double mythreshdistance1;
  double mythreshdistance2;
  double t,x,y,tth,compass,xl,yl;

  };

ActionFindWall::ActionFindWall(double maxSpeed, double stopDistance) :
  ArAction("FindWall")
{
  myMaxSpeed = maxSpeed;
  myStopDistance = stopDistance;
  setNextArgument(ArArg("maximum speed", &myMaxSpeed, "Maximum speed to
go."));
  setNextArgument(ArArg("stop distance", &myStopDistance, "Distance at which to
stop."));
  myTurning = 1;

 }

/*
  Sets the myRobot pointer (all setRobot overloaded functions must do this),
  finds the sonar device from the robot, and if the sonar isn't there,
  then it deactivates itself.
*/
void ActionFindWall::setRobot(ArRobot *robot)
{
  myRobot = robot;
  mySonar = myRobot->findRangeDevice("sonar");
  if (mySonar == NULL)
    deactivate();
}

/*
  This is the guts of the action.
*/
ArActionDesired *ActionFindWall::fire(ArActionDesired currentDesired)
{
  double frontRange,range,rightRange,speed;
  // reset the actionDesired (must be done)
  myDesired.reset();
  // if the sonar is null we can't do anything, so deactivate
  if (mySonar == NULL)
  {
    deactivate();
    return NULL;
  }
            // Get the front readings and right readings off of the sonar
```

```
  range=(mySonar->currentReadingPolar(1,0) - myRobot->getRobotRadius());
  frontRange = (mySonar->currentReadingPolar(-10,10) - myRobot-
>getRobotRadius());
  rightRange = (mySonar->currentReadingPolar(-100,-80) - myRobot-
>getRobotRadius());
  //sets the speed of the robot according to the front range
        speed=frontRange* .22;
        if(speed>myMaxSpeed)
        speed=myMaxSpeed;
        if(rightRange<=range)
        {
                myTurning = 0;
                myDesired.setDeltaHeading(myTurning*5);
                myDesired.setVel(speed);
        }
                myDesired.setDeltaHeading(myTurning*5);
                 if (frontRange <= range)
        {
                myTurning=0;
                myDesired.setDeltaHeading(myTurning*5);
                if(frontRange<=myStopDistance)
                        {
                                myTurning=1;
                                myDesired.setVel(0);
                                myDesired.setDeltaHeading(myTurning*90);
                                        if(rightRange<=range)
                                        {
                                                myTurning = 0;

        myDesired.setDeltaHeading(myTurning*5);
                                                myDesired.setVel(speed);
                                        }
                        }

                        else
                        {
                                myDesired.setVel(speed);
                                myTurning=0;
                                myDesired.setDeltaHeading(myTurning*5);

                        }
        }
}
ActionParallel::ActionParallel(double threshdistance1,double threshdistance2,double
TurnAmount,double maxSpeed) :
  ArAction("Parallel")
{
```

```
  mythreshdistance1 = threshdistance1;
  mythreshdistance2 = threshdistance2;
  myTurnAmount = TurnAmount;
  myMaxSpeed = maxSpeed;
  setNextArgument(ArArg("thresh distance1 (mm)", &mythreshdistance1, "The number
of mm away from obstacle to begin turnning."));
  setNextArgument(ArArg("thresh distance2 (mm)", &mythreshdistance2, "The number
of mm away from obstacle to stop turnning."));
  setNextArgument(ArArg("turn amount (deg)", &myTurnAmount, "The number of
degress to turn if turning."));
}
/*
  Sets the myRobot pointer (all setRobot overloaded functions must do this),
  finds the sonar device from the robot, and if the sonar isn't there,
  then it deactivates itself.
*/
void ActionParallel::setRobot(ArRobot *robot)
{
  myRobot = robot;
  mySonar = myRobot->findRangeDevice("sonar");
  if (mySonar == NULL)
    deactivate();

}
/*
  This is the guts of the action.
*/
ArActionDesired *ActionParallel::fire(ArActionDesired currentDesired)
{
  double leftRange, rightRange,speed,frontRange;
  // reset the actionDesired (must be done)
  myDesired.reset();
  // if the sonar is null we can't do anything, so deactivate
  if (mySonar == NULL)
  {
    deactivate();
    return NULL;
  }

for(i=0;i<=15;i++)
SonarArray[i] = myRobot->getSonarRange(i);
Right1S     = SonarArray[7] ;
Right2S     = SonarArray[8] ;
Left1S      = SonarArray[0] ;
Left2S      = SonarArray[15] ;
RightDiffS  = int((SonarArray[7]-SonarArray[8])/2);
```

```
LeftDiffS   = int((SonarArray[0]-SonarArray[15])/2);
frontRange = (mySonar->currentReadingPolar(-10,10) - myRobot->getRobotRadius());
rightRange  = (mySonar->currentReadingPolar(-100,-80) - myRobot-
>getRobotRadius());
leftRange   = (mySonar->currentReadingPolar(80,100) - myRobot->getRobotRadius());
right=(Right1S+Right2S)/2;
speed=frontRange* .22;
if(speed>myMaxSpeed)
speed=myMaxSpeed;
myTurning4=(400-rightRange)/100;
if(right<=mythreshdistance1)
{
        myDesired.setDeltaHeading(myTurning4);
        myDesired.setVel(speed);
}
if((right>mythreshdistance2)&&(right<1000))
{
        myDesired.setDeltaHeading(myTurning4);
        myDesired.setVel(speed);
}
if((right>mythreshdistance1)&&(right<mythreshdistance2))
        {
                if(abs(RightDiffS)<=5)
                {
                myTurning4=0;
                myDesired.setDeltaHeading(0);
                myDesired.setVel(speed);
                }
                if((Right1S>Right2S)&&(abs(RightDiffS)>5))
                {
                myTurning4=-1;
                myDesired.setDeltaHeading(-1);
                myDesired.setVel(speed);
                }
                if((Right1S<Right2S)&&(abs(RightDiffS)>5))
                {
                   myTurning4=1;
                   myDesired.setDeltaHeading(1);
                   myDesired.setVel(speed);
                 }
            }
}
ActionAvoidSide::ActionAvoidSide(double startdistance,double TurnAmount) :
  ArAction("AvoidSide")
{
  mystartdistance = startdistance;
  myTurnAmount = TurnAmount;
```

```
  setNextArgument(ArArg("start distance (mm)", &mystartdistance, "The number of
mm  away from obstacle to begin turnning."));
  setNextArgument(ArArg("turn amount (deg)", &myTurnAmount, "The number of
degress to turn if turning."));
}
/*
  Sets the myRobot pointer (all setRobot overloaded functions must do this),
  finds the sonar device from the robot, and if the sonar isn't there,
  then it deactivates itself.
*/
void ActionAvoidSide::setRobot(ArRobot *robot)
{
  myRobot = robot;
  mySonar = myRobot->findRangeDevice("sonar");
  if (mySonar == NULL)
    deactivate();
}
/*
  This is the guts of the action.
*/
ArActionDesired *ActionAvoidSide::fire(ArActionDesired currentDesired)
{
  double leftRange, rightRange;
  // reset the actionDesired (must be done)
  myDesired.reset();
  // if the sonar is null we can't do anything, so deactivate
  if (mySonar == NULL)
  {
    deactivate();
    return NULL;
  }
  // Get the left readings and right readings off of the sonar
  leftRange = (mySonar->currentReadingPolar(0, 100) - myRobot->getRobotRadius());
  rightRange = (mySonar->currentReadingPolar(-100, 0) - myRobot-
>getRobotRadius());
  // if neither left nor right range is within the turn threshold,
  // reset the turning variable and don't turn
  // if we're already turning some direction, keep turning that direction
  if (leftRange < mystartdistance)
  {
    myTurning2 = -1;
    myDesired.setDeltaHeading(myTurnAmount * myTurning2);
  }
  if (rightRange < mystartdistance)
  {
    myTurning2 = 1;
    myDesired.setDeltaHeading(myTurnAmount * myTurning2);
```

```
  }
}
ActionAvoidFront::ActionAvoidFront(double obstacledistance,double TurnAmount) :
  ArAction("AvoidFront")
{
  myobstacledistance = obstacledistance;
  myTurnAmount = TurnAmount;
  setNextArgument(ArArg("obstacle distance (mm)", &myobstacledistance, "The
number of mm away from obstacle to begin turnning."));
  setNextArgument(ArArg("turn amount (deg)", &myTurnAmount, "The number of
degress to turn if turning."));
}

/*
  Sets the myRobot pointer (all setRobot overloaded functions must do this),
  finds the sonar device from the robot, and if the sonar isn't there,
  then it deactivates itself.
*/
void ActionAvoidFront::setRobot(ArRobot *robot)
{
  myRobot = robot;
  mySonar = myRobot->findRangeDevice("sonar");
  if (mySonar == NULL)
    deactivate();
}

/*
  This is the guts of the action.
*/
ArActionDesired *ActionAvoidFront::fire(ArActionDesired currentDesired)
{
  double frontRange;
  // reset the actionDesired (must be done)
  myDesired.reset();
  // if the sonar is null we can't do anything, so deactivate
  if (mySonar == NULL)
  {
    deactivate();
    return NULL;
  }
  // Get the front readings off of the sonar
  float s3,s4,b1,b2,b10,bp,b;
  frontRange = (mySonar->currentReadingPolar(-30,30) - myRobot-
>getRobotRadius());

                              s3 = myRobot->getSonarRange(3);
                              s4 = myRobot->getSonarRange(4);
```

```
                        b1=ArMath::fabs(s4-s3);
                        b2=s3+s4;
                        b10=(ArMath::sin(80))/(ArMath::cos(80));
                        bp=b10*b1;
                        b=ArMath::atan2(bp,b2);

   // if neither left nor right range is within the turn threshold,
  // reset the turning variable and don't turn
  // if we're already turning some direction, keep turning that direction
  if (frontRange < myobstacledistance)
  {
   myTurning3 = 1;
   myDesired.setDeltaHeading((b+90) * myTurning3);
  }

  // return a pointer to the actionDesired, so resolver knows what to do
  return &myDesired;
}
ActionTurnCorner::ActionTurnCorner(double distance,double TurnAmount) :
  ArAction("TurnCorner")
{
  mydistance = distance;
  myTurnAmount = TurnAmount;
  setNextArgument(ArArg("distance (mm)", &mydistance, "The number of mm away
from obstacle to begin turnning."));
  setNextArgument(ArArg("turn amount (deg)", &myTurnAmount, "The number of
degress to turn if turning."));
}

/*
  Sets the myRobot pointer (all setRobot overloaded functions must do this),
  finds the sonar device from the robot, and if the sonar isn't there,
  then it deactivates itself.
*/
void ActionTurnCorner::setRobot(ArRobot *robot)
{
  myRobot = robot;
  mySonar = myRobot->findRangeDevice("sonar");
  if (mySonar == NULL)
    deactivate();
}

/*
  This is the guts of the action.
*/
ArActionDesired *ActionTurnCorner::fire(ArActionDesired currentDesired)
{
```

```
  double rightRange,backRange,frontRange;
  // reset the actionDesired (must be done)
  myDesired.reset();
  // if the sonar is null we can't do anything, so deactivate
  if (mySonar == NULL)
  {
    deactivate();
    return NULL;
  }
  // Get the right,back and front readings off of the sonar
  rightRange = (mySonar->currentReadingPolar(-100, 0) - myRobot-
>getRobotRadius());
  backRange = (mySonar->currentReadingPolar(-180, -90) - myRobot-
>getRobotRadius());
  frontRange = (mySonar->currentReadingPolar(-30, 30) - myRobot-
>getRobotRadius());
    if ((rightRange >
mydistance)&&(frontRange>mydistance)&&(backRange<mydistance))
  {
    myTurning = -1;
    myDesired.setDeltaHeading(myTurnAmount * myTurning);
   }


  // return a pointer to the actionDesired, so resolver knows what to do
  return &myDesired;
}
ActionDraw::ActionDraw(double threshdistance1,double threshdistance2) :
  ArAction("Draw")
{
  mythreshdistance1 = threshdistance1;
  mythreshdistance2 = threshdistance2;
  setNextArgument(ArArg("threshdistance1 (mm)", &mythreshdistance1, "The number
of mm away from obstacle to begin turnning."));
  setNextArgument(ArArg("threshdistance2 (mm)", &mythreshdistance2, "The number
of mm away from obstacle to stop turnning."));
}

/*
  Sets the myRobot pointer (all setRobot overloaded functions must do this),
  finds the sonar device from the robot, and if the sonar isn't there,
  then it deactivates itself.
*/
void ActionDraw::setRobot(ArRobot *robot)
{
  myRobot = robot;
  mySonar = myRobot->findRangeDevice("sonar");
```

```
  if (mySonar == NULL)
    deactivate();

}

/*
  This is the guts of the action.
*/
ArActionDesired *ActionDraw::fire(ArActionDesired currentDesired)
{
// double rightRange;
  // reset the actionDesired (must be done)
  myDesired.reset();
  // if the sonar is null we can't do anything, so deactivate
  if (mySonar == NULL)
  {
    deactivate();
    return NULL;
  }
  // Get the left readings and right readings off of the sonar
          //probability of the centre coordinate of the robot
  float reading,R,P7,P8,P27,P28,P78,B,a,a7,a8,a71,a81,s7,s8,r,rs,rc,r7,r8;
  int i;
int xc,yc,_xc,_yc;
          xc=176;
          yc=84;
          _xc=354;
          _yc=258;

 compass=myRobot->getCompass();
       if(compass<yc||compass>=_xc){
            if(compass>_xc)
                compass= 180-(((compass-_xc)/(yc+(360-_xc)))*90);
            else
                compass= 90+(((yc-compass)/(yc+(360-_xc)))*90);
       }
       else if(compass<=xc){
            compass= ((xc-compass)/(xc-yc))*90;
       }
       else if(compass<=_yc){
           compass= (((_yc-compass)/(_yc-xc))*90)+270;
       }
       else if(compass<=_xc){
           compass= ((((_xc-compass)/(_xc-_yc))*90)+180);
      }
      else{
      //Robotun yonu campass 'in kararsiz oldugu bolgede.
```

```
        compass=-1;
    }


 tth=myRobot->getTh();
 printf("compass:%f\n",compass);

for (i = -105; i < -75 ; i++)
 {
  reading=mySonar->currentReadingPolar(i,i+1);
                //printf("i=%d reading=%f\n",i,reading);
                R=5000;
                B=15;
                r=reading;
                a=ArMath::fabs(-90-i);

                        s7=myRobot->getSonarRange(7);
                        s8=myRobot->getSonarRange(8);
                        if((s7<1500)&&(s7>200)&&(s8<1500)&&(s8>200))
                        {
                        if(s7<s8)
                        {
                                rs=ArMath::sin(a);
                                rc=ArMath::cos(a);
                                a71=ArMath::atan2((115-(r*rs)),((r*rc)-130));
                                a7=ArMath::fabs(a71);
                                a81=ArMath::atan2((115+(r*rs)),((r*rc)-130));
                                a8=ArMath::fabs(a81);
                                r7=ArMath::distanceBetween(0,0,r*rc-130,115-r*rs);
                                r8=ArMath::distanceBetween(0,0,r*rc-130,115+r*rs);
                                if((r7=s7)&&(r8=s8)&&(a7<15)&&(a8<15))
                                {
                                        //7.sonarda nesne olma olasılığı
                                        P7=(((R-r7)/R)+((B-a7)/B))/2;
                                        P8=(((R-r8)/R)+((B-a8)/B))/2;
                        //bayes teoremi ile guncelleme
                  P27=(P8*P7)/((P8*P7)+(1-P8)*(1-P7));
                  x=myRobot->getX();
                  y=myRobot->getY();
                  tth=myRobot->getTh();
                 // compass=myRobot->getCompass();

       xl=((ArMath::cos(t))*x)+((ArMath::sin(-t))*y);
       yl=((ArMath::sin(t))*x)+((ArMath::cos(t))*y);
                        FILE *cfPtr;
                        if ( ( cfPtr = fopen( "maplab.txt", "a" ) ) == NULL )
                        {
```

```
                        printf( "File could not be opened\n" );
                }

                if(P27>0.95)
                {

fprintf(cfPtr,"%f %f %f %f  %f %f %f %f %f %f\n",P27,xl,yl,tth,compass,s7,s8,a7,x,y);
                }

                fclose(cfPtr);
                        }
                }
                if(s8<s7)
                {
                        rs=ArMath::sin(a);
                        rc=ArMath::cos(a);
                        a71=ArMath::atan2((115+(r*rs)),((r*rc)-130));
                        a7=ArMath::fabs(a71);
                        a81=ArMath::atan2((115-(r*rs)),((r*rc)-130));
                        a8=ArMath::fabs(a81);
                        r7=ArMath::distanceBetween(0,0,r*rc-130,115+r*rs);
                        r8=ArMath::distanceBetween(0,0,r*rc-130,115-r*rs);
                        if((r7=s7)&&(r8=s8)&&(a7<15)&&(a8<15))
                        {
                                //8.sonarda nesne olma olasılığı
                                P7=(((R-r7)/R)+((B-a7)/B))/2;
                                P8=(((R-r8)/R)+((B-a8)/B))/2;
                //bayes teoremi ile guncelleme
                                P28=(P8*P7)/((P8*P7)+(1-P8)*(1-P7));
              x=myRobot->getX();
              y=myRobot->getY();
              tth=myRobot->getTh();
             //  compass=myRobot->getCompass();

      xl=((ArMath::cos(t))*x)+((ArMath::sin(-t))*y);
      yl=((ArMath::sin(t))*x)+((ArMath::cos(t))*y);
                FILE *cfPtr;
                if ( ( cfPtr = fopen( "maplab.txt", "a" ) ) == NULL )
                {
                        printf( "File could not be opened\n" );
                }

                if(P28>0.95)
                {

fprintf(cfPtr,"%f %f %f %f  %f %f  %f %f %f
%f\n",P28,xl,yl,tth,compass,s7,s8,a8,x,y);
```

```
                        }
                        fclose(cfPtr);
                                }
                        }
                        if(s8=s7)
                        {
                                a71=ArMath::atan2(115,r-130);
                                a7=ArMath::fabs(a71);
                                a81=ArMath::atan2(-115,r-130);
                                a8=ArMath::fabs(a81);
                                r7=ArMath::distanceBetween(0,0,r-130,115);
                                r8=ArMath::distanceBetween(0,0,r-130,115);
                                if((r7=s7)&&(r8=s8)&&(a7<15)&&(a81>-15))
                                {
                                        //8.sonarda nesne olma olasılığı
                                        P7=(((R-r7)/R)+((B-a7)/B))/2;
                                        P8=(((R-r8)/R)+((B-a8)/B))/2;
                        //bayes teoremi ile guncelleme
                                        P78=(P8*P7)/((P8*P7)+(1-P8)*(1-P7));
                    x=myRobot->getX();
                    y=myRobot->getY();
                    tth=myRobot->getTh();
                   // compass=myRobot->getCompass();

        xl=((ArMath::cos(t))*x)+((ArMath::sin(-t))*y);
        yl=((ArMath::sin(t))*x)+((ArMath::cos(t))*y);
                        FILE *cfPtr;
                        if ( ( cfPtr = fopen( "maplab.txt", "a" ) ) == NULL )
                        {
                                printf( "File could not be opened\n" );
                        }

                        if(P78>0.95)
                        {
fprintf(cfPtr,"%f %f %f %f  %f %f %f %f %f %f\n",P78,xl,yl,tth,compass,s7,s8,a7,x,y);
                        }
                                        fclose(cfPtr);
                                }
                        }
                        }



   }
}
int main(void)
{
```

```
// The connection we'll use to talk to the robot
ArTcpConnection con;
// the robot
ArRobot robot;
// the sonar device
ArSonarDevice sonar;
// some stuff for return values
int ret;
std::string str;
// the behaviors from above, and a stallRecover behavior that uses defaults
ArActionStallRecover recover;
ArActionBumpers bumpers;
ActionFindWall findwall(200,400);
ActionAvoidSide AvoidSide(200,5);
ActionAvoidFront AvoidFront(400,95);
ActionParallel Parallel(350,450,2,200);
ActionTurnCorner TurnCorner(500,178);
ActionDraw Draw(300,500);
// quit the program
        ArKeyHandler keyhandler;
        printf("Press escape to quit the program\n");
        robot.attachKeyHandler(&keyhandler);
// this needs to be done
Aria::init();
// open the connection, just using the defaults, if it fails, exit
if ((ret = con.open()) != 0)
{
  str = con.getOpenMessage(ret);
  printf("Open failed: %s\n", str.c_str());
  Aria::shutdown();
  return 1;
}
// add the range device to the robot, you should add all the range
// devices and such before you add actions
robot.addRangeDevice(&sonar);
// set the robot to use the given connection
robot.setDeviceConnection(&con);
 // do a blocking connect, if it fails exit
if (!robot.blockingConnect())
 {
printf("Could not connect to robot... exiting\n");
Aria::shutdown();
return 1;
}

  // enable the motors, disable amigobot sounds
robot.comInt(ArCommands::ENABLE, 1);
```

```
  robot.comInt(ArCommands::SOUNDTOG, 0);
  // add our actions in a good order, the integer here is the priority,
  // with higher priority actions going first
  robot.addAction(&recover, 100);
  robot.addAction(&bumpers, 99);
  robot.addAction(&AvoidFront, 98);
  robot.addAction(&AvoidSide, 88);
  robot.addAction(&findwall,70);
  robot.addAction(&Parallel,85);
  robot.addAction(&TurnCorner,95);
  robot.addAction(&Draw,30);

 Draw.setComp(robot.getCompass());
  // run the robot, 'true' means to return if it loses connection
  robot.run(true);
    // now just shutdown and go away
  Aria::shutdown();
  return 0;
}
```

%Yazan: Elif EROGLU Tarih: 16.05.2006

```
%Amaç:  Gerçek ortamda P3 Dx Robotu ile Harita Çıkarma
clear
clc
ah1=fopen('maplab.txt')
aa=fscanf(ah1,'%f %f %f %f %f %f %f %f %f %f',[10 inf]);
bb=aa';
status=fclose(ah1)
[m,n]=size(bb);
for i=1:m
P=bb(i,1);
xl=bb(i,2);
yl=bb(i,3);
th=bb(i,4);
comp=bb(i,5);
if(comp>360)
   comp=0;
end
if(comp<0)
   comp=abs(comp);
end
s7=bb(i,6);
s8=bb(i,7);
a=bb(i,8);
x=bb(i,9);
y=bb(i,10);
figure(1)
xk=cos(-6*pi/180)*(-xl)+(-sin(-6*pi/180))*(-yl);
yk=sin(-6*pi/180)*(-xl)+cos(-6*pi/180)*(-yl);
plot(xk,yk,'.')
hold on
if((s7<800)&&(s8<800))
if(s7<=s8)
x71=xk+(-115*cos(comp*pi/180)-130*sin(comp*pi/180));
x7=x71+(-s7*sin(a*pi/180)*-cos(comp*pi/180)+s7*cos(a*pi/180)*-sin(comp*pi/180));
y71=yk+(130*cos(comp*pi/180)-115*sin(comp*pi/180));
y7=y71+(-s7*cos(a*pi/180)*-cos(comp*pi/180)-s7*sin(a*pi/180)*-sin(comp*pi/180));
plot(x7,y7,'c.')
hold on
end
if(s8<s7)
x81=xk+(115*cos(comp*pi/180)-130*sin(comp*pi/180));
x8=x81+(s8*sin(a*pi/180)*-cos(comp*pi/180)-s8*cos(a*pi/180)*sin(comp*pi/180));
y81=yk+(130*cos(comp*pi/180)+115*sin(comp*pi/180));
y8=y81+(-s8*cos(a*pi/180)*-cos(comp*pi/180)-s8*sin(a*pi/180)*sin(comp*pi/180));
plot(x8,y8,'r.')
hold on
end
end
end

%Yazan: Elif EROGLU   Tarih: 16.05.2006
```

```
%Amaç: Simulator Ortamında Harita Çıkarma
clear
clc
ah1=fopen('sim1.txt')
aa=fscanf(ah1,'%f %f %f %f %f %f %f %f %f %f',[10 inf]);
bb=aa';
status=fclose(ah1)
[m,n]=size(bb);
for i=1:m
P=bb(i,1);
xl=bb(i,2);
yl=bb(i,3);
th=bb(i,4);
comp=bb(i,5);
if(comp>360)
   comp=0;
end
if(comp<0)
   comp=abs(comp);
end
s7=bb(i,6);
s8=bb(i,7);
a=bb(i,8);
x=bb(i,9);
y=bb(i,10);
figure(1)
xk=cos(pi/180)*(xl)+(-sin(pi/180))*(yl);
yk=sin(pi/180)*(xl)+cos(pi/180)*(yl);
plot(xk,yk,'.')
hold on
if((s7<800)&&(s8<800))
if(s7<=s8)
x7=xk+(115*cos(comp*pi/180)+130*sin(comp*pi/180))+(-
s7*sin(a*pi/180)*cos(comp*pi/180)+s7*cos(a*pi/180)*sin(comp*pi/180));
y7=yk+(-130*cos(comp*pi/180)+115*sin(comp*pi/180))+(-
s7*cos(a*pi/180)*cos(comp*pi/180)-s7*sin(a*pi/180)*sin(comp*pi/180));
plot(x7,y7,'c.')
hold on
end
if(s8<s7)
x8=xk+(115*cos(comp*pi/180)+130*sin(comp*pi/180))+
(s8*sin(a*pi/180)*cos(comp*pi/180)+s8*cos(a*pi/180)*sin(comp*pi/180));
y8=yk+(-130*cos(comp*pi/180)-115*sin(comp*pi/180))+(-
s8*cos(a*pi/180)*cos(comp*pi/180)+s8*sin(a*pi/180)*sin(comp*pi/180));
plot(x8,y8,'r.')
hold on
end
end
end
```