
GAZİ ÜNİVERSİTESİ
Gazi
1926

OTONOM MOBİL ROBOTLAR İÇİN A* VE OLASILIKSAL YOL HARİTASI TABANLI HİBRİT YOL PLANLAMA ALGORİTMASI TASARIMI VE ANALİZİ

Barış UZUN

YÜKSEK LİSANS TEZİ
ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ ANA BİLİM DALI

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

NİSAN 2022

ETİK BEYAN

Gazi Üniversitesi Fen Bilimleri Enstitüsü Tez Yazım Kurallarına uygun olarak hazırladığım bu tez çalışmasında;

- Tez içinde sunduğum verileri, bilgileri ve dokümanları akademik ve etik kurallar çerçevesinde elde ettiğimi,
- Tüm bilgi, belge, değerlendirme ve sonuçları bilimsel etik ve ahlak kurallarına uygun olarak sunduğumu,
- Tez çalışmasında yararlandığım eserlerin tümüne uygun atıfta bulunarak kaynak gösterdiğimi,
- Kullanılan verilerde herhangi bir değişiklik yapmadığımı,
- Bu tezde sunduğum çalışmanın özgün olduğunu, bildirir, aksi bir durumda aleyhime doğabilecek tüm hak kayıplarını kabullendiğimi beyan ederim.

Barış UZUN
13/04/2022

OTONOM MOBİL ROBOTLAR İÇİN A* VE OLASILIKSAL YOL HARİTASI TABANLI HİBRİT YOL PLANLAMA ALGORİTMASI TASARIMI VE ANALİZİ

(Yüksek Lisans Tezi)

Barış UZUN

GAZİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

Nisan 2022

ÖZET

Son yıllarda gelişen teknolojiler ile robot teknolojilerinde büyük ilerlemeler kaydedilmiştir. Gelişen robot teknolojileri insan hayatının her alanında görülmektedir. Robotlar geliştikçe otonom yani kendi kendine hareket edebilecek, kendi kararlarını kendi verebilecek hale gelmiştir. Otonom teknolojisinin en önemli uygulamalarından biri en kısa rota hesaplama algoritmalarıdır. Bu tez çalışmasında robot teknolojilerinden bahsedilmiş ve en kısa rota hesaplama algoritmaları analiz edilmiştir. Öncelikle rota hesaplama algoritmaları sınıflandırılmaları ile gösterilmiş ve ilgili algoritmalar özet şeklinde açıklanmıştır. Tezin ana konusu olan A* ve PRM algoritmaları detaylıca açıklanmıştır. A* algoritması pratik uygulamalarda verimli ve stabil çalışan bir algoritmadır. Harita üzerindeki en kısa rotayı mutlaka bulur fakat harita boyutu büyüdükçe rota hesaplama süresi üstel bir şekilde artmaktadır. Bu sebeple mevcut bu tez çalışmasında PRM algoritması kullanılarak A* algoritmasının bu dezavantajının ortadan kaldırılması hedeflenmiştir. Ayrıca tez çalışmasında PRM algoritması detaylı olarak analiz edilmiş ve PRM algoritması ile A* algoritmasının birlikte çalışacağı hibrit bir algoritma geliştirilmiştir. Literatürdeki diğer çalışmalardan farklı olarak yeni oluşturulan algoritmanın verimliliğinin test edilmesi için 10 farklı haritada 2 algoritma rota oluşturma süresi, rota uzunluğu vb. gibi parametreler açısından karşılaştırılmıştır. Bu haritalarda literatür çalışmaları dikkate alınarak farklı zorluklarda performanslarının testlerinin gerçekleştirilmesi için farklı boyutlarda ve farklı engel yoğunluğunda analizler gerçekleştirilmiştir. Çalışmanın devamında test sonuçları incelenmiş ve algoritmaların avantajları ve dezavantajları görülmüştür. Ayrıca bu dezavantajların çözümü için yeni öneriler sunulmuştur.

Bilim Kodu	:	90542
Anahtar Kelimeler	:	A* algoritması, PRM algoritması, yol planlama algoritmaları, mobil robotlar
Sayfa Adedi	:	146
Danışman	:	Prof. Dr. M. Cengiz TAPLAMACIOĞLU
2. Danışman	:	Doç. Dr. Haluk GÖZDE

A* AND PROBABILISTIC ROADMAP BASED HYBRID PATH PLANNING ALGORITHM DESIGN AND ANALYSIS FOR AUTONOMOUS MOBILE ROBOTS

(M. Sc. Thesis)

Barış UZUN

GAZİ UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

April 2022

ABSTRACT

With the developing technologies in recent years, great progress has been made in robot technologies. Developing robot technologies are seen in all areas of human life. As robots develop, they have become autonomous which they can move on their own and make their own decisions. One of the most important applications of autonomous technology is the shortest route calculation algorithms. In this thesis, robot technologies are mentioned, and the shortest route calculation algorithms are analyzed. Firstly, route calculation algorithms are shown with their classification and related algorithms are explained in summary form. A* and PRM algorithms, which are the main subject of the thesis, are explained in detail. The A* algorithm is an efficient and stable algorithm in practical applications. It definitely finds the shortest route on the map, but as the map size increases, the route calculation time increases exponentially. For this reason, it is aimed to eliminate this disadvantage of the A* algorithm by using the PRM algorithm in this thesis study. In addition, the PRM algorithm was analyzed in detail and a hybrid algorithm was developed in which the PRM algorithm and A* algorithm would work together in the thesis study. Unlike other studies in the literature, in order to test the efficiency of the newly created algorithm, 2 algorithms on 10 different maps compared in terms of parameters such as route creation time, route length, etc. Considering the literature studies on these maps, analyzes of different sizes and different obstacle densities were carried out in order to test their performance at different difficulties. In the continuation of the study, the test results were examined, and the advantages and disadvantages of the algorithms were seen. In addition, new suggestions are presented for the solution of these disadvantages.

Science Code	:	90542
Key Words	:	A* algorithm, PRM algorithm, path planning algorithm, mobile robots
Page Number	:	146
Supervisor	:	Prof. Dr. M. Cengiz TAPLAMACIOĞLU
Co-Supervisor	:	Doç. Dr. Haluk GÖZDE

TEŞEKKÜR

Bu tez çalışması süresince her türlü görüş, öneri ve destekleri ile yanımda olan değerli danışmanım Prof. Dr. M. Cengiz TAPLAMACIOĞLU hocama çok teşekkür ederim.

Yardımları ve yönlendirmeleri ile bana katkıda bulunan ve çalışmalarımda daima beni destekleyen Doç. Dr. Haluk GÖZDE hocama teşekkür ederim.

Ayrıca bu süreçte maddi ve manevi desteklerini esirgemeyen, her zaman yanımda olan ve çalışmalarımda beni cesaretlendiren aileme ve tüm dostlarıma teşekkürü borç bilirim.

İÇİNDEKİLER

Sayfa

Sayfa

ÇİZELGELERİN LİSTESİ

Çizelge **Sayfa**

ŞEKİLLERİN LİSTESİ

Şekil **Sayfa**

Şekil **Sayfa**

Şekil **Sayfa**

Şekil **Sayfa**

Şekil **Sayfa**

SİMGELER VE KISALTMALAR

Bu çalışmada kullanılmış simgeler ve kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

Simgeler	**Açıklamalar**
sn	Saniye

Kısaltmalar	**Açıklamalar**
ABC	Artificial Bee Colony (Yapay Arı Kolonisi)
BFS	Breadth First Search (Genişlik Öncelikli Arama)
Bi-RRT	Bidirectional-Rapidly Exploring Random Tree (İki Yönlü Hızlı Keşfeden Rastgele Ağaçlar)
DFS	Depth First Search (Derinlik Öncelikli Arama)
DGPS	Differential Global Positioning System (Farklı Global Konumlama/Pozisyonlama Sistemi)
GA	Genetic Algorithm (Genetik Algoritma)
GBFS	Greedy Best First Search (Aç Gözlü Eniyi Öncelikli Arama)
GPS	Global Positioning System
HAS	Hierarchical A* algorithm (Hiyerarşik A* Algoritması)
ISO	International Organization for Standardization (Uluslararası Standartlar Organizasyonu)
İHA	İnsansız Hava Araçları
LACH	Late Acceptance Hill Climbing (Geç Kabul Tepe Tırmanışı)
ODE	Open Dynamics Engine (Açık Dinamik Makina)
OSRF	Open Source Robotics Foundation (Açık Kaynaklı Robotik Vakfı)
OSM	Open Street Map (Açık Sokak Haritası)
PRM	Probabilistic Roadmap (Olasılıksal Yol Haritaları)
PSO	Particle Swarm Optimization (Parçacık Sürü Optimizasyonu)

Kısaltmalar	Açıklamalar
RBFNN	Radial Basis Function Neural Network (Radyal Temel Fonksiyonlu Sinir Ağı)
ROS	Robot Operating System (Robot İşletim Sistemi)
RRT	Rapidly Exploring Random Tree (Hızlı Keşfeden Rastgele Ağaçlar)
SCARA	Selective Compliance Assembly Robot Arm (Seçilebilir Uygun Tertibatlı Robot Kolu)
SGPP	Safe Global Path Planning (Güvenli Global Yol Planlama)

1. GİRİŞ

İnsanoğlu diğer canlılar ile karşılaştırıldığında bazı bedensel kabiliyetlerden (savunma, hareket, uçma vb.) geri kaldığı görülmektedir. Ancak insanoğlunun en güçlü özelliği bedeni değil beynidir. Beynini kullanarak kendi kabiliyetlerini artıracak ve işlerini kolaylaştıracak alet, edevat ve cihazlar geliştirmiştir. Daha hızlı, daha güvenilir ve daha rahat bir şekilde yolculuk etmek için de çeşitli araçlar geliştirmiştir. Buhar gücünün keşfi ile buhar motorlu taşıtlar, petrolün keşfi ile içten yanmalı motorlu taşıtları geliştirmiştir. Elektriğin keşfi ile araçlar daha sofistike hale gelmiştir. Bilgisayar teknolojisinin keşfi ve yaygınlaşması ile bu teknolojiler taşıtlara eklenmiş ve taşıtlar daha performanslı, daha rahat ve hatta sürücüsüz otonom çalışabilecek hale gelmişlerdir.

Teknolojik gelişmelerin arkasında insanoğlunun ihtiyaçları, rahatlığı ve merakı her zaman itici bir güç olmuştur. Günlük şehir içi yolculuklarında, kıtalararası yolcuklarda, deniz ve okyanusların incelenmesinde ve özellikle de uzayın keşfinde gelişen teknoloji büyük rol oynamaktadır. Özellikle otonom (sürücüsüz) taşıt teknolojisi bu kabiliyetleri mümkün kılmaktadır.

Otonom taşıtlar herhangi bir sürücüye ihtiyaç duymadan verilen görevleri kendi kendine yapabilen taşıtlardır. Bu görevler çoğunlukla istenen bir noktadan belirlenen başka bir noktaya herhangi bir insan müdahalesi olmadan gidebilmesidir. Bu taşıtlara otonom arabalar, insansız hava araçları, denizaltı araştırma robotları, Mars gezgin robotları, derin uzay sondaları gibi insan hayatının her alanından birçok örnek verilebilir. Otonom araç teknolojisinin bu kabiliyetlere sahip olmasının en önemli özelliklerinden birisi de en kısa yol bulma algoritmalarıdır.

En kısa yol bulma algoritmaları verilen bir noktadan istenilen bir hedef noktaya en kısa ve az maliyetli yolu hesaplayıp rotasını oluşturan algoritmalardır. Bunun için birçok algoritma geliştirilmiştir [1]. İlk zamanlarda matematiksel klasik yol bulma algoritmaları geliştirilmiş ve kullanılmıştır. Klasik algoritmalar hızlı ve az maliyetli rotalar oluşturabilmektedir ancak harita boyutu büyüdükçe ve harita karmaşıklaştıkça klasik algoritmaların rota oluşturma süreleri üstel bir şekilde artmaktadır. Bu sorunları çözmek için olasılıksal ve doğadan ilham alan sezgisel algoritmalar geliştirilmiştir [2, 3]. Olasılıksal algoritmalar daha hızlı ve daha

pratik rotalar üretebilmektedir. Sezgisel algoritmalar daha karmaşık ortamlarda klasik algoritmalara göre daha kısa sürelerde rotalar oluşturabilmektedir [2, 3]. Olasılıksal ve sezgisel algoritmalar klasik algoritmalara göre bazı durumlarda daha avantajlıdırlar ancak bazı durumlarda da dezavantajlıdırlar. Örnek olarak olasılıksal algoritmalar hızlı rota üretebilmesine karşın, üretilen rotalar zikzaklı ve keskin dönüşlere sahiptir. Sezgisel algoritmalar ise çalıştığı haritada rota oluşturma garantisi verememektedir. Her tür algoritmanın avantajları ve dezavantajları vardır [3].

A* algoritması temelde klasik bir yol planlama algoritmasıdır. En kısa rota hesaplama algoritmalarının içinde en çok kullanılan algoritmalardan birisidir [4]. Verilen haritayı yoğunluğuna göre ızgara şeklinde karelere böler ve başlangıç karesinden hedef karesine doğru sıra ile kareleri inceleyerek ilerler. A* algoritması hesaplamasında başlangıç ile hedef karesi arasındaki sezgisel mesafeyi de kullanarak, fazla dallanmadan hedefine doğru ilerler. Bu özelliği sayesinde diğer klasik algoritmalara göre, çok daha hızlı rota oluşturabilmektedir [5].

PRM (Olasılıksal yol haritaları) algoritması olasılıksal bir algoritmadır ve olasılıksal algoritmaların içinde en çok kullanılanlardan birisidir. Harita üzerinde haritanın yoğunluğuna göre çeşitli sayıda rastgele noktalar atar ve rota hesaplamasını bu noktalar üzerinden yapar. Büyük haritalarda A* algoritması gibi haritanın tamamını parça parça incelemediği için çok daha hızlı rota hesaplayabilir. Fakat engel yoğunluğunun fazla olduğu ve dar koridorlara sahip haritalarda rota oluşturmada zorlanırlar. Ayrıca oluşturulan yol yukarıdaki bahsedildiği gibi yumuşak değildir.

Literatür taraması

2013 yılında ElHalawany, Abdel-Kader, TagEldeen, Elsayed ve Nossair; Standart A* algoritmasının gelişmiş bir versiyonu üzerinde çalışmışlardır. A* algoritmasının en kısa yolu bulmasına karşın, bulduğu yolun engellerin çok yakınından geçiyor olması üzerine standart A* algoritmasının daha güvenli versiyonunu geliştirmişlerdir. Bunun için iki ana konu üzerinde çalışmışlardır. Bunlar; A* algoritmasının bitişik iki engel arasındaki çapraz geçişleri kullanması ve algoritmanın rota hesaplarken robotun boyutunu dikkate almamasıdır. Bunun çözümü için algoritma kareleri incelerken çapraz yollarda yanındaki kareyi de incelemişlerdir. İkinci sorun için kare incelemelerinde robotun boyutu da

hesaplamaya dahil edilmiş ve robotun boyutundan daha büyük olacak şekilde karelerde hesaplamalar yapılmıştır. Çalışmalarının sonunda bu iki algoritma karşılaştırılmış ve değiştirilmiş A* algoritmasının, standart A* algoritmasına göre daha güvenli yollar oluşturduğu görülmüştür [6].

2014 yılında Shwail ve Karim; PRM, A* ve Genetik algoritmasını hibrit bir şekilde kullanarak yeni bir yol planlama algoritması geliştirmişlerdir. PRM algoritması ile rastgele nokta ataması yapılmış ve A* algoritması ile atanan noktalar üzerinden başlangıç ve bitiş noktaları arasındaki en kısa rotayı hesaplamışlardır. Fakat oluşturulan yol rastgele atanan noktalar üzerinden hesaplandığı için daha kısa yollar mevcuttur. Bunun için oluşturulan yolları Genetik algoritma kullanarak daha optimize hale getirmişlerdir. Çalışmanın sonunda PRM ve A* algoritmasının oluşturduğu 10 adet yol Genetik algoritması ile optimize edilip orijinal halleri ile karşılaştırılmıştır. Karşılaştırmada bütün yolların optimize edilmiş hallerinin orijinal hallerinden daha kısa olduğunu gözlemişlerdir [1].

2014 yılında Wang, Zhou, Zheng ve Liang; Google haritalar gibi global navigasyon yazılımlarında güncel hali bulunmayan bölgeler için kullanıcılara yardımcı olacak hızlı bir algoritma geliştirmeyi hedeflemişlerdir. Bunun için OSM (Open Street Map) programından güncel haritaları alıp bu haritalar üzerinden A* algoritmasını kullanarak yol bulma işlemini yapmak istemişlerdir. Fakat A* algoritması büyük ölçekli haritalarda çok uzun sürelerde sonuç verdiği için HAS (Hierarchical A* algorithm) algoritmasını geliştirmişlerdir. Bu algoritma şehir haritasını iç içe küçük parçalara bölerek klasik A* algoritmasındaki gibi baştan sona bütün ihtimalleri incelemek yerine öncelikle büyük parçalardaki rotayı bulmuştur. Sonra sırasıyla daha küçük parçalara inerek hedefine ulaşmıştır. Böylelikle aradaki gereksiz noktaları eleyerek daha kısa sürelerde sonuç elde etmiştir. Bu iki algoritmayı Çin'deki bir bölgeye uygulamışlardır. Deneyde HAS algoritmasının klasik A* algoritmasına göre çok daha kısa sürelerde rota oluşturduğunu gözlemişlerdir [5].

2015 yılında Yuan, Liang, Fu, Xu ve Ma; PRM algoritmasında iki farklı hibrit yöntem uygulamışlardır. Bunlar hibrit örnekleme stratejisi ile A* algoritmasıdır. Hibrit örnekleme stratejisini PRM algoritmasının nokta atama bölümünde kullanmakta, A* algoritmasını ise atanan noktaların birleştirilmesinde kullanmaktadırlar. Hibrit örnekleme stratejisini bridge test örneklemesi ile non-uniform örneklemesinin ortak çalışması ile oluşturmuşlardır. Bu strateji ile hem dar boğazlarda hem de sınır bölgelerinde daha fazla nokta atanması

amaçlanmıştır. Nokta birleştirme aşamasında A* algoritmasının kullanılması ile gereksiz noktaların silinip daha az nokta ile birleştirme yapılması amaçlanmıştır. Yapılan testlerde hibrit örnekleme stratejisi ile standart örnekleme yöntemine göre hem dar boğazlarda hem de sınır bölgelerinde daha fazla nokta ataması yapıldığını görmüşlerdir. Nokta birleştirme aşamasında ise A* algoritmasının kullanılması standart yönteme göre daha az sayıda nokta ile yol oluşturabilmesini sağladığını ve dolayısıyla daha kısa yol oluşturabildiğini gözlemişlerdir [7].

2016 yılında Lafcı; A* algoritmasını incelemiş ve A* algoritmasının dinamik ortamlarda verimli çalışabilmesi için mobil robota lazer sensörü eklentisi yapmıştır. Yapılan uygulamalarda rota robot harekete geçmeden önce A* algoritması ile oluşturulmuş, robotun oluşturulan rota üzerinden hareketinde önüne çıkan engellerden lazer sensörü yardımı ile uzaklaşması sağlanmıştır. Çalışmada Gazebo çoklu robot simülatörü programı tanıtılmıştır. Dinamik engel durumu çoğunlukla birden fazla robotun aynı ortamda hareket etmesi durumunda oluştuğu için; 2 ve 4 adet robotun aynı harita da farklı rotalarda hareket etmesi şeklinde simülasyonu yapılmıştır. Bu çalışmada dinamik ortamlarda verimsiz olan A* algoritmasının lazer sensör eklentisi ile verimli olabileceği gösterilmiştir [8].

2016 yılında Park ve Huh; Yol planlama algoritmaları çalışmalarında en kısa yolu hesaplanmaktan ziyade en güvenli yolu bulmaya odaklanmışlardır. Bunun için robotun boyutunu, engellerin boyutlarını, engellerin şekillerini hesaba katarak robotun engele çarpmayacak şekilde hareket etmesini dikkate almışlardır. Bunlarla birlikte algoritmalarını geliştirirken durma mesafesi, fren mesafesi ve serbest hareket mesafesini dikkate alıp robotun maksimum sürede maksimum hızında hareket etmesini amaçlamışlardır. Bu parametreleri klasik A* algoritmasına ekleyip SGPP (Safe Global Path Planning) adında yeni bir yol planlama algoritması oluşturmuşlardır. SGPP algoritmasını klasik A* ve PRM algoritması ile bir tane simülasyon üzerinde ve bir tane de gerçek ortamda karşılaştırmışlardır. 2 karşılaştırmada da SGPP algoritması klasik A* ve PRM algoritmasına göre daha güvenli yol oluşturabilmiştir. Bundan dolayı ortalama hız olarak en yüksek hıza ulaşmış ve en kısa yolculuk süresine sahip olmuştur. Ancak elde edilen yolun uzunluğunun diğer algoritmalardan daha uzun olduğu görülmüştür. İki karşılaştırma da klasik A* algoritması PRM algoritmasına göre 2 iterasyonda da en kısa yolu oluşturmuştur. Çarpışma risk durumu, yolculuk süresi ve ortalama hız açısından karşılaştırıldığında ise simülasyon

haritasında PRM algoritması, gerçek ortamdaki deneme de ise A* algoritması daha iyi sonuçlar sunmuştur [9].

2016 yılında Yetkin; A* ile alt hedef graf algoritmalarını incelemiştir. Algoritmaların sonuçları gerçek bir balıktan esinlenilen robot bir balık üzerinde uygulanmıştır. Bunun için gerçek bir balığın hareketlerini simüle edebilen, carangiform yüzüş şekline sahip bir balık robotun modeli MATLAB ortamında oluşturulmuştur. Bu balığın bir noktadan başka bir noktaya gidebilmesi için A* ve alt hedef graf algoritmalarının oluşturduğu rotalar kullanılmıştır. En kısa rotayı A* algoritmasının ürettiği gözlenmiştir. Alt hedef graf algoritmasının ürettiği rota ise daha uzun ancak engellerden daha uzak hareket ettiği için daha güvenli olduğu görülmüştür [10].

2017 yılında Güllü; Çalışmasında hem haritası bilinmeyen ortamlarda çeşitli algoritmalarla ortamın haritasını çıkarmış hem de oluşturulan harita üzerinde istenen bir noktaya en kısa rotayı oluşturan çeşitli algoritmaları karşılaştırmıştır. Bilinmeyen ortamın haritasını çıkarırken sol duvar takip, sağ duvar takip, akıllı duvar takip, DFS (Depth First Search), BFS (Breadth First Search) algoritmalarını kullanılmıştır. Sol ve sağ duvar takip algoritmaları karmaşık ortamlarda sürekli döngüye girdiği için akıllı duvar takip algoritması geliştirilmiştir. Akıllı duvar takip algoritması döngüye girdiğini anlayarak rotasını değiştirebilmektedir. Bu algoritmalar içinden en verimli sonuçlar DFS ve BFS algoritmalarından elde edilmiştir. DFS ve BFS algoritmaları Dijkstra algoritması ile optimize edilmiş ve daha verimli sonuçlar üretilmiştir. Haritanın tanımlanmasından sonra belirlenen bir hedefe en kısa rotayı oluşturmak için 2 farklı algoritma kullanılmıştır. A* ve GBFS (Greedy Best First Search) algoritmaları karşılaştırılmış ve her zaman en iyi sonucu A* algoritması vermiştir [11].

2017 yılında Santiago, Ocampo, Ubando, Bandala ve Dadios; PRM ve GA algoritmalarını incelemişler ve bu iki algoritmayı rota hesaplama süresi ve bulunan rotanın yumuşaklığı açısından karşılaştırmışlardır. Bunun için bir tane basit ve bir tane engel yoğunluğu ve boyutu daha büyük karmaşık bir harita kullanmışlardır. Testlerin sonucunda PRM algoritmasının rota oluşturma süresi bakımından GA'ya göre daha kısa sürelerde rota oluşturduğunu görmüşlerdir. Fakat iki algoritmayı oluşturulan rotanın yumuşaklığı bakımından karşılaştırdıklarında GA'nın PRM algoritmasına göre daha yumuşak rotalar oluşturduğunu görmüşlerdir [2].

2018 yılında Garip, Atalı, Karayel ve Özkan; A* algoritması ile genetik algoritmasının beraber çalıştığı yeni bir hibrit algoritma geliştirmişlerdir. Algoritma çoklu robotlar ile statik ortamda denenmiştir. Denemeler MATLAB-GUI arayüzünde yapılmıştır. Geliştirilen bu yeni hibrit algoritması zaman ve maliyet açısından A* ve genetik algoritmasının yalın çalışmasından daha verimli sonuçlar ürettiğini görmüşlerdir [12].

2018 yılında Wang ve Cai; Nükleer tesislerdeki arızalarda tesisi boşaltırken ya da basit arızaların tamiri esnasında bakım elemanlarının etrafa yayılan rasyasyondan daha az etkilenmesi için daha güvenli rotalar oluşturacak bir algoritma üzerinde çalışmışlardır. Bunun için PRM algoritmasını ve PRM algoritmasının nokta atamasından sonra rota oluşturmak için A* algoritmasını kullanmışlardır. PRM algoritması acil boşaltma durumlarında hızlı rota oluşturduğu için kullanışlı bir algoritmadır fakat oluşturulan rota üzerinde ilerlerken yüksek miktarda radyasyona maruz kalındığı için bu çalışma da en kısa yolu bulmaktan ziyade en güvenli rotayı bulmaya odaklanmışlardır. Bundan dolayı rota oluştururken doz değerlerini de hesaba katmışlardır. Bu çalışma da radyasyonun harita üzerindeki dağılımını bulmak için Monte Carlo yöntemini kullanmışlardır. Yeni oluşturulan algoritmayı MATLAB üzerinde 2 farklı durumda test etmişlerdir. 1. durumda standart bir harita da klasik PRM algoritması yeni PRM algoritması ile karşılaştırılmıştır. Haritanın belirli bir noktasına radyasyon kaynağı konulmuştur. Testin sonucunda standart PRM algoritmasının daha kısa sürede daha kısa rota oluşturduğunu görmüşlerdir. Fakat yeni PRM algoritmasının oluşturduğu rotanın çok daha az radyasyona maruz kaldığını görmüşlerdir. İkinci durumda algoritmaların rota oluşturma hızlarını karşılaştırmak için 1. duruma benzer bir harita da yeni algoritma ile standart A* algoritmasını karşılaştırmışlardır. Testin sonucunda yeni PRM algoritmasının A* algoritmasına göre çok daha hızlı rota hesaplayabildiğini görmüşlerdir [13].

2019 yılında Alpkıray; PRM algoritmasının dezavantajlarını çözmek için PRM ile ABC (Yapay arı kolonisi) algoritmalarının beraber çalıştığı bir hibrit algoritma geliştirmiştir. Oluşturulan yeni algoritma ile klasik PRM algoritması çeşitli kriterlerde karşılaştırılmıştır. Karşılaştırmalarda iki algoritmanın da oluşturduğu rotaların uzunlukları birbirine çok yakın çıkmıştır. Fakat oluşturulan rotalar güvenlik açısından karşılaştırıldığında arada yüksek bir fark görülmüştür. Klasik PRM algoritmasının rotaları engellerin çok yakınından geçerken PRM-ABC hibrit algoritmasının oluşturduğu rotalar çoğunlukla engelden güvenli bir mesafeden geçmektedir. Oluşturulan rotalar yumuşaklığı açısından karşılaştırıldığında

PRM-ABC algoritması klasik PRM algoritmasına göre çok daha yumuşak rotalar ürettiği görülmüştür [3].

2019 yılında Dere; A*, PRM, RRT ve genetik algoritmaları incelemiştir. Çeşitli örnek haritalar üzerinde bu 4 algoritma karşılaştırılmış ve en kısa yolu A* algoritmasının bulduğu görülmüştür. PRM ve RRT algoritması diğer iki algoritmaya göre çok daha hızlı sonuç bulmuştur. PRM algoritması çok hızlı sonuç üretmesine karşın örnekleme tabanlı olduğu için her işlemde farklı bir rota üretmiştir. Bu çalışma da bunun önüne geçmek için farklı bir yöntem kullanılmıştır. Rastgele nokta atama aşamasında her engelin köşelerine nokta atayarak deneme yapılmış ve hem daha kısa rotalar bulunabilmiş hem de iterasyonlar boyunca aynı veya çok yakın rotalar bulunabilmiştir. Yeni PRM algoritmasının oluşturduğu rotaların uzunluğu klasik A* algoritmasının oluşturduğu rotaların uzunluğu ile çok yakın çıkmıştır. Ayrıca A* algoritmasının daha hızlı rota hesaplaması için açık uçlu engeller kapatılmış ve çoğunlukla daha hızlı rota hesaplanabilmiştir [4].

2020 yılında Chowdhury ve Schwartz; İnsansız denizaltı araçlarında 2 konu üzerinde çalışmışlardır. İlk olarak PRM ve A* algoritmalarını hibrit bir şekilde kullanarak yeni bir algoritma geliştirmişlerdir. A* algoritması en kısa yolu hesaplamasına karşın, engele çok yakın rota üretmektedir. PRM algoritması ise rastgele nokta ataması ile rota ürettiği için geniş açılarda rota üretmektedir. Chowdhury ve Schwartz bu iki algoritmanın oluşturduğu rotaların arasında tekrardan nokta atamış ve PRM algoritmasını kullanarak engelden daha uzak ama klasik PRM algoritmasından daha kısa rota hesaplamışlardır. İkincisi ise dinamik ve hareketli engellerin olduğu durumlarda rota hesaplayabilmek için global ve yerel yol planlama algoritmalarını birlikte kullanmışlardır. Geliştirdikleri sistemde öncelikle robot harekete geçmeden önce PRM-A* algoritması ile robotun rotasını oluşturmuştur. Sonra rotanın üzerinde herhangi bir engel çıktığında robot engelin potansiyel gideceği rotanın haricindeki bölgelerde yeniden nokta ataması yapıp, PRM algoritması ile yeniden yerel rota üretmiş ve engeli aşıp orjinal oluşturulan rotasına bağlanmıştır. Bu çalışmayı 2 adet labirent benzeri olmayan ve 2 adet labirent benzeri olan harita da denemişlerdir. 4 harita da global rotalar oluşturulmuş ve haritadaki 5 hareketli engelden 2'si global rotayı kesmiştir. Geliştirdikleri yeni yerel yol planlama algoritması 4 harita üzerinde de robotun engellere çarpmadan yeni yol oluşturarak global yola bağlanmasını sağlamıştır [14].

2020 yılında Hopkins, Joy, Sheta, Turabieh ve Kar; İnsansız hava araçlarının kapalı alan uygulamalarında yol planlaması yapmak istemişlerdir. Bunun için PRM algoritmasının köşegen oluşturma aşamasından sonra oluşturduğu yol haritası üzerinden rota oluşturmak için A* ve LACH (Late Acceptance Hill Climbing) algoritmalarını kullanmışlar ve bu iki algoritmayı birbirleri ile karşılaştırmışlardır. Testleri çeşitli engel yoğunluğuna sahip haritalarda yapmışlardır. Testlerin sonucunda LACH algoritması A* algoritmasına göre daha kısa rota bulmuştur. Ancak iki algoritma rota oluşturma süresi açısından karşılaştırıldığında A* algoritmasının LACH algoritmasına göre daha kısa sürelerde rota oluşturduğunu görmüşlerdir [15].

2020 yılında Mokwele, Du ve Benade; Klasik yol planlama algoritmalarını inceleyip karşılaştıran bir çalışma yapmışlardır. Bunun için Potansiyel Alanlar Yöntemi, PRM, RRT ve A* algoritmalarını kullanmışlardır. Bu 4 algoritmayı bir harita üzerinde denemişler ve hesaplama yükü, sonuç bulma süresi, sonuç bulabilme kapasitesi, bulunan yolun uzunluğu ve tekrarlanabilirlik gibi parametreler üzerinden karşılaştırmışlardır. Testin sonucunda Potansiyel Alanlar Yöntemi sonuç bulamamış ve diğer algoritmalar sonuç bulmuşlardır. Bu 5 parametreyi dikkate alındıklarında en iyi sonucu A* algoritmasının verdiğini görmüşlerdir. Sonrasında PRM algoritması daha sonra RRT algoritması ve en son olarak Potansiyel Alanlar Yöntemi algoritmasının verdiğini görmüşlerdir [16].

2020 yılında Madridano, Al-Kaff, Flores, Martin ve Escalera; İHA (İnsansız Hava Araçları) sürülerinin güvenli hareketi üzerine çalışmışlardır. Bunun için hız engeli ve engel tanımlama isimli iki görevli bir algoritma geliştirmişlerdir. Hız engeli, İHA sürülerinin hedefine giderken dar geçit gibi yerlerde birbirine çarpmaması için geliştirdikleri bir algoritmadır. Bunun için her İHA'nın konum ve rota bilgisini merkezi bir kontrolcüye gönderirler ve çarpışma olasılığı olan İHA'lardan bazılarını yavaşlatarak çarpışmayı önlemiş olurlar. Engel tanımlama bölümünde ise PRM-A* algoritmasını kullanmışlardır. Bunun için İHA'ların hareketi esnasında daha önceden bilinmeyen bir engel karşılarına çıktığında PRM-A* algoritmasını kullanarak hızlı bir şekilde yeni bir rota planlamışlar ve engelden kaçınmışladır. Geliştirdikleri yeni algoritmayı Gazebo isimli simülasyon programında test etmişlerdir. Test esnasında 5 adet İHA kullanmışlar ve İHA'ların hareketi esnasında birbirine yakınlaşan İHA'lar olduğunda merkezi kontrolcüden komut gitmiş ve bazı İHA'lar yavaşlayarak yolculuğun güvenli bir şekilde gerçekleşmesini sağlamışladır [17].

2020 yılında Roa-Borbolla, Jimnez-Nieto, Espinosa, Villa-Paniagua, Ruiz-Martinez, Vega-Valera; Kapalı alan yangınlarında insanların panik halinde olması ve çıkış rotasını bilememelerinden dolayı kapalı alan yangınlarında en kısa kaçış rotasının oluşturulması üzerine çalışmışlardır. Bunun için A* ve PRM algoritmalarını ayrı ayrı kullanmışlar ve bu iki algoritmayı birbirleri ile karşılaştırmışlardır. Test için yangın simülasyon programı kullanmışlardır. Bu programda çeşitli engel yoğunluğuna sahip, çeşitli yangın başlama noktaları olan ve yangının yayılım hızının farklı olduğu haritalar kullanılmıştır. Testlerin sonucunda A* algoritmasının oluşturduğu yolların boyunun PRM algoritmasının oluşturduğu yoldan daha kısa olduğunu görmüşlerdir. Algoritmaları yol oluşturma süreleri üzerinden karşılaştırdıklarında sürelerin ortalamasını alarak karşılaştırmışlardır. Testin sonucunda PRM algoritmasının A* algoritmasına göre çok daha hızlı rota oluşturduğunu görmüşlerdir [18].

2021 yılında Dias, Silva C., Batista, Souza, Silva J., Ramalho ve Lima; Endüstri de kullanılan SCARA (Selective Compliance Assembly Robot Arm) robotları üzerine çalışma yapmışlardır. SCARA robotları endüstri de sıklıkla kullanılan cihazlardır ve bu robotların çalışma esnasında herhangi bir yere çarpması ya da operatöre çarpması hem can kaybına hem de yüksek maliyete sebep olmaktadır. Bunun için SCARA robotlarının istenen hedeflere ulaşırken engellerden kaçınması için PRM algoritmasını bu robotlara uygulamışlardır. Hem yuvarlak hem de dikdörtgen şekilli deneme haritalarında 100 ve 1000 adet nokta ataması yaparak 20 defa rota oluşturmuşlardır. Denemenin sonucunda her iki harita üzerinde de 100 adet nokta ataması yapılan algoritmanın daha hızlı yol oluşturduğunu görmüşlerdir. 1000 adet nokta ataması yapan algoritma daha kısa yol oluşturmuş ancak oluşan yolun engeller ile kesiştiğini görmüşlerdir [19].

2021 yılında Liu, Yao ve Dong; Bir mobil robot üzerine monteli robot kol sisteminin görev planlama algoritması üzerine çalışma yapmışlardır. Mobil aracın yol planlaması için A* algoritmasının gelişmiş bir versiyonunu, robotik kol sisteminin programlanması içinde Bi-RRT algoritmasının geliştirilmiş bir versiyonunu kullanmışlardır. Mobil robotun yol planlamasında kullanılan klasik A* algoritmasının hesaplama fonksiyonuna hareketli şasinin sapma fonksiyonunu ekleyerek yeni bir fonksiyon oluşturmuşlardır. Bu fonksiyon ile hesaplanan nokta sayısının azalmış, rota hesaplama süresinin kısalmış ve rotanın yumuşaklığının artmış olduğunu görmüşlerdir. Robot kol yol planlaması için Bi-RRT (Bidirectional Rapidly Exploring Random Tree) algoritmasının rastgele nokta tanımlaması

bölümünde adaptif yerçekimsel alanları kullanmışlardır. Bu yöntemin kullanılması ile Bi-RRT algoritmasının nokta atamasının engelden uzaklaştırılıp hedefe yakınlaşması sağlanmıştır. Böylece yol planlama verimliliğini artırmışlardır. Sonrasında oluşan yolu daha yumuşak hale getirmek için RBFNN (Radial Basis Function Neural Network) algoritmasını kullanmışlar ve daha yumuşak bir rota elde etmişlerdir. Geliştirdikleri algoritmaları MATLAB üzerinde denediklerinde geliştirdikleri A* algoritmasının klasik A* algoritmasına göre daha yumuşak bir rota elde ettiğini görmüşlerdir. Robotik kol rota planlamasında ise 10 kere deneme yapmışlar ve her seferinde geliştirdikleri yeni algoritmanın standart Bi-RRT algoritmasına göre daha kısa sürelerde ve daha yumuşak rotalar oluşturduğunu görmüşlerdir [20].

Tezin amacı ve katkısı

A* algoritması matematiksel hesaplamalar üzerinden rota hesapladığı için karmaşık haritalar üzerinde hesaplama yaparken aşırı süre harcamaktadır. PRM algoritması, karmaşık haritaları yeterli bağlantıyı sağlayacak noktalar atayarak daha basit hale getirir. Bu iki algoritmayı birleştirerek her iki algoritmanın avantajlarının birbirlerinin dezavantajlarını düzeltmesine katkı sağlayacağı beklenmektedir.

Bu tez çalışmasının amacı PRM ve A* algoritmalarının birleşimi olan bu algoritma ile klasik A* algoritmasının çeşitli haritalar üzerinden karşılaştırılmasıdır. Bu iki algoritma literatürde bu amaçla kullanılanlardan farklı haritalar üzerinde uygulanacak ve hesaplama süresi, bulunan yolun uzunluğu gibi kriterler açısından karşılaştırılacaktır.

Tezin içeriği

Tezin ikinci bölümünde otonom mobil robotların temel bileşenlerinden bahsedilmektedir. Mobil robotların donanım, yazılım ve navigasyon bileşenleri kapsamlı bir şekilde açıklanmaktadır.

Tezin üçüncü bölümünde yol planlama algoritmalarından bahsedilmektedir. Bu bölüm 2 alt başlıktan oluşmaktadır. Birinci alt başlıkta yol planlama algoritmalarının sınıflandırmasından bahsedilmektedir. Algoritmaların çalışırken çevre ve engellerin durumuna göre ya da çalışma mantığına göre nasıl ayrıldığı ve özellikleri anlatılmıştır. İkinci

alt başlıkta ise yol planlama teknikleri açıklanmıştır. Bu bölümde algoritmaların çalışma tekniklerine, çalışma mantıklarına göre nasıl ayrıldığı belirtilmiş ve her teknikten farklı algoritmalar kısaca anlatılmıştır.

Tezin dördüncü bölümünde tezin ana konusu olan algoritmalar detaylıca incelenmiştir. Bu bölüm 3 alt başlıktan oluşmaktadır. Birinci alt başlıkta tezin iki ana algoritmasından birisi olan A* algoritması kapsamlı bir şekilde anlatılmış ve bir örnek üzerinden rota hesaplaması yapılmıştır. İkinci bölümde tezin iki ana algoritmasından diğeri olan PRM algoritmasından bahsedilmiştir. PRM algoritmasının tarihçesi, aşamaları anlatılmış ve bir örnek üzerinden rota hesaplaması gerçekleştirilmiştir. Üçüncü bölümde ise PRM algoritmasında oluşturulan grafik harita üzerinden, A* algoritması kullanılarak nasıl rota oluşturulacağı gösterilmiştir.

Tezin beşinci bölümünde PRM-A* algoritması ile klasik A* algoritması çeşitli haritalar üzerinde uygulanmış ve hesaplama süresi, yol uzunluğu gibi kriterler açısından karşılaştırılmıştır.

Tezin altıncı bölümünde elde edilen veriler değerlendirilmiş ve yeni çalışmalar için gelecekte yapılması potansiyel önerilerde bulunulmuştur.

2. OTONOM MOBİL ROBOTLARIN TEMEL BİLEŞENLERİ

Robot kelimesi ilk olarak 1921 yılında Karel Capek'in RUR (Rossums Universal Robots, Rossum'un Evrensel Robotları) adlı tiyatro eserinde geçmektedir ve dünya literatürüne girmiştir. Köken olarak Çekçe'den gelmiştir ve hizmet eden, köle anlamlarına gelmektedir [21]. İnsanoğlu robotları bazı durumlarda kendilerine yardımcı olması için bazı durumlarda ise tamamen kendi işini yapması için geliştirmiştir.

Geliştirilen robotlar amacına göre çok çeşitlidirler. Bazı robotlar insan gibi 2 ayak üzerinde durup hareket edebilir, bazıları sonda gibidir ve uzay keşfinde kullanılır, bazıları uçabilir ve bazıları bu tezde de incelenen gibi tekerlekli bir araç gibidir. Tek başına hareket edip keşif, taşıma vb. görevlerini yapabilir ya da otomobil gibi insan taşıma görevinde bulunabilirler. Bu araçlar otonomdur yani insan müdahalesi olmadan kendi kendine görevini yapabilmektedir.

Otonom robotlar görevlerine göre özelleşmiş donanım ve yazılım yapılarına sahiptir. 2.1. bölümde mobil robotların donanım bileşenlerinden, 2.2. bölümde yazılım bileşenlerinden ve 2.3. bölümde ise navigasyon adımları ve yol planlamasının öneminden bahsedilmektedir.

2.1. Donanım Bileşenleri

Robotlar kullanım alanlarına göre özelleşmiş donanım yapılarına sahiptir. Hareket edebilmek için tekerleklere, paletlere ya da ayaklara; uçabilmek için aerodinamik yapıda bir gövdeye ve kanatlara; yüzebilmek için yüzgeçlere ve su içinde hareket edebilecek bir gövdeye vb. gibi çeşitli özelliklere sahiptir.

Mobil robotun (hareketli robotlar) hareket kabiliyeti donanım yapısına bağlıdır. Keskin dönüşleri yapabilmek, büyük engebeli araziyi geçebilmek, çeşitli arazi zorluklarını aşabilmek (çamur, su birikintisi vb) için özel eklem yapılarına ve hareket mekanizmalarına göre geliştirilmesi gerekmektedir.

Robotlar kullanım alanlarına, eklem yapılarına, çalışma şekillerine, boyutlarına, hareketlilik durumuna ve hareket yöntemlerine göre çeşitli kategorilere ayrılmaktadır [22].

Endüstriyel robotlar

Endüstriyel robotlar ISO 8373 standardına göre "üç veya daha fazla programlanabilir ekseni olan, otomatik kontrollü, yeniden programlanabilir, çok amaçlı, sabit veya hareketli endüstriyel manipülatör" biçiminde tanımlanmaktadır [22]. Endüstriyel robotların eklemli robotlar, kartezyen robotlar, scara robotlar, polar robotlar, delta robotlar, silindirik robotlar ve işbirlikçi robotlar gibi çeşitleri vardır. Endüstriyel robotlar bir çalışana görevini yapmakta yardımcı olan ya da görevi tek başına yapabilen, tekrarlayan görevleri yüksek verimlilikle yerine getirebilen robotlardır [23]. Bir insan için tehlikeli ya da imkânsız olabilecek işleri rahatlıkla yapabilmektedir. Bunun için endüstriyel robotlar türüne göre yüksek hareket kabiliyetine sahip, uzun süre çalışabilecek dayanıklılıkta ve ağır nesneleri kaldırabilecek kuvvete sahip olarak geliştirilmektedirler. Şekil 2.1'de farklı endüstriyel robot çeşitleri görülmektedir.

a

b

Şekil 2.1. Çeşitli endüstriyel robotlar (a: Delta Robot [24], b: Kuka-kr-240 [25])

Tekerlekli robotlar

En çok kullanılan mobil robot çeşitlerindendir. 2, 4, 6 veya 8 tekerlekli olabilir. Tekerleklerini hareket ettirebilmek için motorlara ve motorda oluşan hareketi tekerleklere iletebilmek için aktarma sistemine sahiptir. Araç kullanım yerine ve amacına göre belirli hareket kabiliyetine göre tasarlanmaktadırlar [26]. Örnek olarak daha keskin dönüşleri yapabilmek için tekerlek ve gerekirse gövde hareket açıları yüksek olacak şekilde tasarlanmaktadır. Bazılarında engebeli arazide rahat hareket etmesi için süspansiyon sistemi eklenmiştir. Şekil 2.2'de farklı tekerlekli robot çeşitleri görülmektedir (Soldan sağa sırasıyla NASA'nın Mars'a gönderdiği Perseverance isimli aracı ve Tesla'nın Model S'i).

a b

Şekil 2.2. Tekerlekli mobil araçlar (a: Perseverance [27], b: Tesla Model S [28])

Ayaklı robotlar

Çoğunlukla insanlardan ve hayvanlardan ilham alınarak geliştirilmiş robotlardır. İnsanların ve hayvanların hareket biçimleri, eklem yapıları, serbestlik dereceleri örnek alınarak tasarlanmıştır. Geliştirilen robotlar insanlara servis hizmetinde, askeri projelerde vb. alanlarda kullanılmaktadır. Ayaklı robotlarla ilgili bugüne kadar birçok çalışma yapılmıştır. Honda'nın geliştirdiği ASIMO, Waseda üniversitesinin WABIAN, KAIST Mühendisliğin KHR robotları bu alandaki önemli gelişmelerden bazılarıdır [29]. Türkiye de ise bu alanda Sabancı Üniversitesinin geliştirmiş olduğu SURALP robotu vardır [22]. Şekil 2.3'te çeşitli ayaklı robot çeşitleri görülmektedir. (Soldan sağa sırasıyla KHR-3 ve SURALP).

a b

Şekil 2.3. Ayaklı robot çeşitleri (a: KHR-3 [29], b: SURALP [30])

Uçan robotlar

İnsanlığın en eski özlemlerinden birisi uçmaktır. Uçabilmek için bugüne kadar birçok defa cihazlar geliştirilmiştir. Fakat son yıllarda gelişen teknoloji ile çok daha modern uçaklar yapılmıştır. Özellikle bilgisayar teknolojisinin gelişmesi ile uçaklar otonom yani insansız uçabilecek kabiliyetler kazanmıştır. Uçaklar keşif, taşıma, sağlık, askeri, uzay, eğlence, tarım gibi birçok alanda kullanılmaktadır. Görevlerine, hızına, irtifasına, yük taşıma kabiliyetine göre birçok türde geliştirilmiştir. Yüksek irtifa da ve yüksek hızlarda hareket etmesi için geniş kanat yapısına ve jet motoru gibi güçlü motorlara sahip uçaklar kullanılmaktadır. Keşif, gözetleme, tarım gibi havada asılı kalma ya da sabit hareket gerektiren görevler için 4, 6 veya daha fazla motordan oluşan helikopter benzeri pervaneli bir yapıda İHA'lar (İnsansız hava araçları) geliştirilmiştir [31]. Şekil 2.4'te farklı uçan robot çeşitleri görülmektedir. (Soldan sağa sırasıyla TB-2 ve Kargu).

a b

Şekil 2.4. Farklı İHA çeşitleri (a: TB2 [32], b: Kargu [33])

Yüzebilen robotlar

İnsanlar yapısı gereği su altında yaşamaya uygun değildir. Fakat insanlar her zaman su altındaki yaşamı merak etmiştir. Bunun için balıklardan ilham alarak birçok araç geliştirmişlerdir. Özellikle gelişen son teknolojilerin sayesinde insansız deniz araçları geliştirilmiştir. Bu araçların birçok çeşidi vardır fakat genel olarak suda hareket etmesi için balık yüzgeci benzeri hareket organlarına ya da pervaneli motorlara sahiptir. Gövdeleri suda istediği zaman dalıp çıkabilecek, hareket esnasında en az sürtünme ile karşılaşabilecek ve yüksek manevralar yapabilecek hareket serbestliğine sahip eklemlerden oluşacak şekilde geliştirilmiştir. İnsansız deniz araçları keşif, gözlem, askeri, sağlık gibi birçok alanda kullanılmaktadır [34]. Şekil 2.5'te farklı yüzebilen robot çeşitleri görülmektedir.

Şekil 2.5. Farklı yüzebilen robot çeşitleri [34]

Sürü robotları

Sürü robotları robotik teknolojisinin en gelişmiş alanıdır. Donanım, haberleşme ve yazılım teknolojilerinin gelişmesi ile karıncalar ve arılar gibi doğada sürü halinde hareket eden canlılardan ilham alınarak geliştirilmiştir [35]. Homojen birçok robotun ortak bir akılla koordineli bir şekilde hareket etmesi mantığına dayanan bir yöntemle tasarlanmıştır. Bir görevi sürü robotları ile yapmak görevin daha hızlı, daha kesin bir şekilde yapılmasını sağlamaktadır. Sürü robotları ile ilgili laboratuvar ortamlarında birçok çalışma mevcuttur. Harvard Üniversitesinin geliştirdiği 1024 adet robottan oluşan kilobot isimli sürü robotu topluluğu ve Fraunhofer Enstitüsünün üzerinde çalıştığı karınca robotları bu alanda yapılan

önemli çalışmalardan bazılarıdır [22, 36, 37]. Şekil 2.6'da farklı sürü robotu çeşitleri görülmektedir. (Soldan sağa sırasıyla quadrotor sürüsü ve kilobot).

Şekil 2.6. Farklı sürü robot çeşitleri [37]

2.2. Yazılım Bileşenleri

Yazılım bileşenleri robotun yapacağı görevlerin kodlandığı ya da robota öğretildiği kısımdır. Robotun görevine göre çeşitli algoritmalar kullanılmaktadır. Bu algoritmalar çeşitli yazılımlar sayesinde robota aktarılır. Bu yazılımları seçerken kabiliyeti, kullanım kolaylığı, kullanılacak robota kod atabilmenin mümkünlüğü gibi kriterlerine dikkat edilerek seçilmelidir.

Robot sistemi geliştirmek maliyetli bir görevdir. Yapılan her geliştirmenin gerçek robot üzerinde denenmesi her zaman mümkün olmamaktadır. Bunun için çeşitli simülasyon programları geliştirilmiştir. Simülasyon programları ile robot sistemi robot fiziksel olarak yapılmadan test edilebilmektedir [21].

Robot işletim sistemi (ROS)

ROS adında geçtiği gibi işletim sistemi olarak anlaşılsa da aslında bilgisayar üzerinden robot kontrol etmeye yarayan açık kaynak kodlu bir yazılım sistemidir. İlk olarak Stanford Yapay Zekâ Laboratuvarı'nda Stanford AI Robot STAIR projesiyle geliştirilmiştir. Geliştirmeler 2008 ile 2013 yılları arasında Willow Garage tarafından devam ettirilmiştir. Bu durum ROS'un gelişmesine önemli ölçüde kaynak ve uzman sağlamıştır. 2013 yılından sonra ROS'un yönetimi Open Source Robotics Foundation (OSRF) adlı kuruma aktarılmış ve

halen kurum tarafından devam ettirilmektedir. ROS dünya da binlerce kullanıcı tarafından çeşitli alanlarda kullanılmaktadır ve dünya da binlerce üniversite, şirket ve özel kullanıcılar tarafından geliştirilip açık kaynak kodlu olarak lisanslanmaktadır [38]. ROS içinde bulunan kütüphanelerin ve robotların dışında, diğer kullanıcıların geliştirmiş olduğu kütüphanelere ve araçlara da destek vermekte ve bunun sayesinde her geçen gün daha kapsamlı hale gelmektedir. ROS'un bir diğer avantajı ise yazılan kodların desteklenen her robot için geçerli olmasıdır. Bundan dolayı her robot için ayrı kod yazmaya gerek kalmamaktadır. Ayrıca ROS çeşitli simülasyon programları ile uyumlu çalışmaktadır [39].

MATLAB

MATLAB, MathWorks tarafından geliştirilmiş çok paradigmalı sayısal hesaplama yapabilen dördüncü nesil programlama dilidir. İsmi matrix laboratory kelimelerinin kısaltmalarından oluşmuştur. İsminden de anlaşılacağı gibi MATLAB matris tabanlı işlemler yapmaktadır. Dördüncü nesil programlama dili ise içinde hazır şablonları, fonksiyonları ve sihirbazları bulunan kullanımı kolaylaştırılmış dillerdir [40]. MATLAB programının özellikleri şunlardır [41];

- MATLAB ile C, C++, Java gibi dillerle hazırlanmış programlarla çalışabilinir,
- 2 ve 3 boyutlu grafik çizimlerini yapabilmektedir,
- Lineer cebir, istatistik, fourier analizi, karmaşık matris işlemleri gibi ağır matematik işlemlerini rahatlıkla yapabilmektedir,
- Sayısal analiz, görüntü işleme, yapay sinir ağları, derin öğrenme, en kısa yol planlama algoritmaları gibi algoritmalar da çoğunlukla kullanılmaktadır,

Bu tez çalışmasında algoritmalar MATLAB programı üzerinde geliştirilmiş ve geliştirilen algoritmalar MATLAB programında simüle edilmiştir.

Gazebo

Robot sistemi disiplinler arası bir teknolojidir. Bir robot sistemi geliştirilirken bütün çalışmaların gerçek ortamda yapılması zaman, para ve iş gücü bakımından maliyetli

olmaktadır. Bunun için robot sistemlerine özel gerçek ortamın denenebileceği simülasyon programları geliştirilmiştir. Gazebo bunlardan birisidir.

Gazebo 2002 yılında Güney Kaliforniya Üniversitesinde Dr. Andrew Howard ve öğrencisi Nate Koenig tarafından geliştirilmeye başlanmıştır. Hem dış mekân hem de iç mekânda robot simülasyonu yapabilme kabiliyetine sahiptir. 2009 yılında Willow Garage'ın kıdemli araştırma mühendisi John Hsu tarafından ROS ve PR2 Gazebo'ya dahil edilmesi ile birlikte Gazebo ROS topluluğunun en çok kullandığı robot simülasyon uygulaması olmuştur. 2011 yılında Willow Garage finansal olarak desteklemeye başlamıştır. 2012 yılında Open Source Robotics Foundation (OSRF) Gazebo projesinin temsilcisi olmuştur. 2018 yılında OSRF ismini Open Robotics olarak değiştirmiştir [42, 43].

Gazebo fizik motoru olarak ODE (Open Dynamics Engine), Bullet, Simbody ve Dart'ı kullanmaktadır [42]. Birden fazla robotu 3 boyutlu ortamda simüle edebilmektedir. Kullanıcısına C++ gibi nesneye yönelik diller ile geliştirme yapmasına imkân vermektedir. İçindeki kayıtlı robotlar sayesinde bir sürü robotun simülasyonu kolaylıkla yapılabilmektedir.

Webots

Webots açık kaynaklı 3 boyutlu robot simülasyon programıdır. 1996 yılında İsviçre Federal Teknoloji Enstitüsünden Dr. Olivier Michel tarafından geliştirilmeye başlanmış ve 1998 yılından itibaren Cyberbotics Ltd. tarafından patentli lisanslı bir yazılım olarak geliştirilmeye devam edilmiştir. 2018 yılında itibaren ücretsiz ve açık kaynaklı Apache 2 lisansı altında yayınlanmaktadır [44].

Webots içinde birçok farklı robot (Aibo, Lego, Mindstorms, Kpherea vb.), sensör (lidar, radar, ışık sensörü, gps, dokunmatik sensörü, basınç sensörleri vb.), nesne ve aktüatör barındırmaktadır. Bunların haricinde program içinde herhangi bir robot modellenebilmekte ya da başka bir programdan modellenmiş robotlar programa eklenebilmektedir. Webots fizik motoru olarak ODE kütüphanesini kullanmaktadır. Webots dışında robot kontrol programları yazılmak istenirse basit bir API kullanılarak C, C++, Python, ROS, Java ve MATLAB'da yazılabilir. Webots'un bir diğer artısı ise yapılan simülasyonun ekran görüntüsü ve video kaydı alınabilir ve herhangi bir sunumda rahatlıkla kullanılabilir [44].

2.3. Navigasyon Adımları ve Yol Planlamanın Önemi

Navigasyon işlemi herhangi bir başlangıç noktasından istenilen bir hedefe ulaşma aşamalarıdır. Bu aşamaların gerçekleşebilmeleri için [38];

- Çalışılacak ortamın haritası
- Robotun ve gidilecek hedef noktanın konum bilgileri
- Navigasyon ya da ortam sensör bilgileri
- Yol planlama algoritması verileri kullanılmaktadır.

Çalışılacak ortamın haritası

Robotların hareketleri ortamın haritasının bilinip bilinmemesine göre statik ve dinamik yol planlama olmak üzere 2'ye ayrılır.

Robotun hareket edeceği ortamın tamamının bilindiği duruma statik yol planlama denir. Engellerin yerleri ve tüm harita robotun hafızasına yüklenir. Robot hafızasındaki harita üzerinde sahip olduğu algoritma ile başlangıç ve bitiş arasındaki en optimum rotayı oluşturur ve hareketine öyle başlar. Bu tez çalışmasında bu yöntem kullanılmıştır.

Çalışma ortamının kısmen bilindiği ya da hiç bilinmediği durumlara dinamik yol planlama denir. Çünkü her adımda dinamik olarak ortam algılanıp haritanın çıkarılması ve çıkarılan harita üzerinde nirengi noktalarının tespit edilmesi gerekir. Daha sonra yol planlaması yapılır. Dinamik durumda harita sensör algılama mesafesi kadar oluşturulabilir ve uzak mesafe bu adımlarla gidilir. Her adımda yeniden haritalama ve yol planlaması yapılır. Bu nedenle dinamiktir.

Robot pozisyon bilgileri

Navigasyon işlemleri esnasında ihtiyaç olan bir diğer bilgi ise robot pozisyon bilgileridir. Robotun belirlenen bir rotada hareketi sırasında nerede olduğu ve hedefine ne zaman ulaşacağının bilgisi hareket esnasında önemli bilgilerdir. Bu bilgileri elde etmek için çeşitli yöntemler geliştirilmiştir. GPS, DGPS, işaretleyici tanıma (marker recognition), iç mekân

yeri tahmini (indoor location estimation), ölü hesaplar (dead reckoning) bu yöntemlerden bazılarıdır [38].

Çeşitli sensör bilgileri

Navigasyonun gerçekleşmesi için ihtiyaç olan bir diğer konu ise sensör bilgileridir. Robotların çevrenin haritasını oluşturmasında ya da hareketi esnasında önüne çıkan engellerden kaçması için çeşitli sensörlerden düzenli veri alması gerekmektedir. Bu ihtiyaçlar için çeşitli mesafe ve görüş sensörleri kullanmaktadır. Mesafe sensörü olarak; lazer sensörler (LIDAR, LDS, LRF), kızılötesi mesafe sensörleri, ultrasonik sensörleri kullanılır. Görüş sensörü olarak ise çeşitli kameralar kullanılmaktadır. Gelişen teknoloji ile günümüzde Kinect, RealSense ve AsusXtion gibi sensörler yaygın olarak kullanılmaktadır.

Yol planlama algoritmaları

Bir robotun navigasyonu için en önemli adımlardan birisi ise başlangıç ile hedef nokta arasındaki en uygun rotayı hesaplayabilmektir. Bu işlemler için özelleşmiş yazılımlara yol planlama algoritmaları denilmektedir [38]. Yol planlama algoritmalarının çalışma yöntemleri, çalışma ortamları, rota bulma süreleri, rota hedefleri, esinlenilme yöntemleri açısından birçok çeşidi vardır. Tezin 3. bölümünde bu konu hakkında detaylı bilgiler verilmektedir. Bu algoritmaların birbirlerine göre üstün ve zayıf yönleri vardır. Duruma göre en uygun algoritma seçilmelidir.

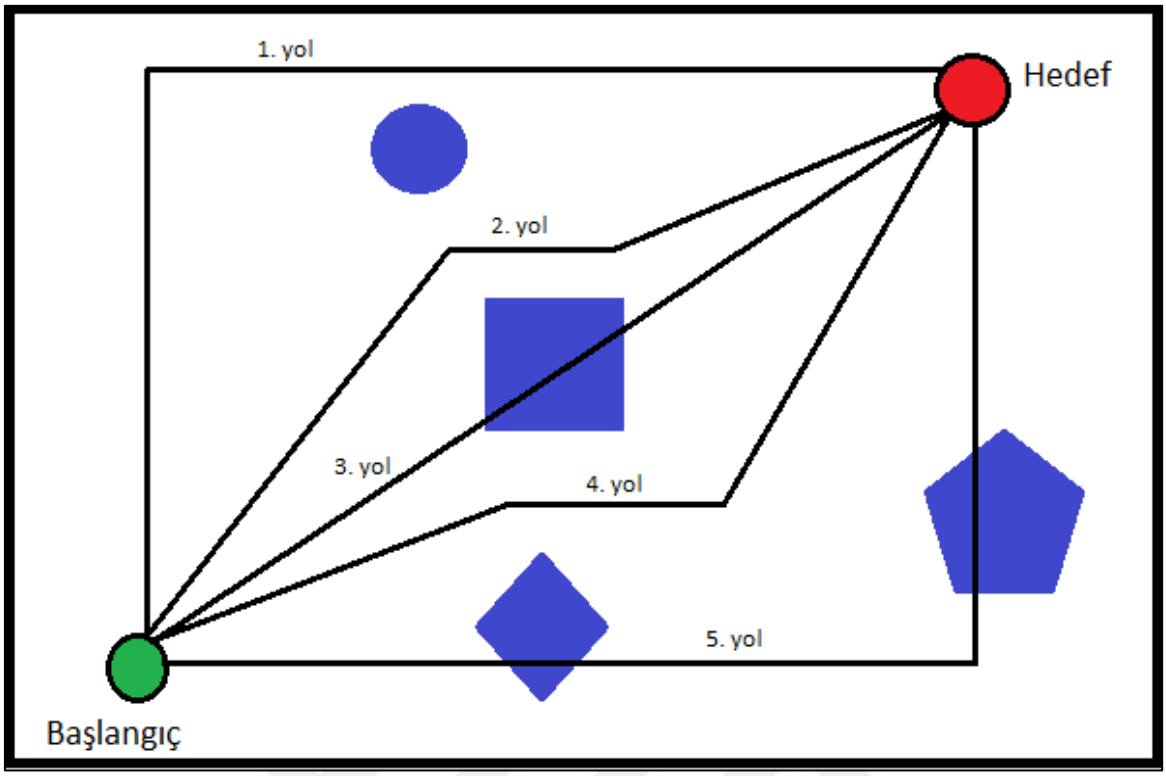
Robotun bir noktadan hedeflenen başka bir noktaya hareketinin rotası hesaplanırken dikkat edilecek birçok önemli konu vardır. Ulaşım süresi, rota hesaplama süresi, kullanılacak yolun düzgünlüğü, rotanın engelden uzak olması bunlardan bazılarıdır. Rotanın kısa olması ekonomik olarak uygundur fakat aşırı engebeli ya da zikzaklı rotanın kullanılması uygun bir seçim değildir. Ya da çok kısa bir rota bulunması için aşırı uzun süreler harcanması da uygun bir yöntem değildir. Uzun hesaplama süreleri dinamik ortamlarda kabul edilemez bir durumdur. Yol planlaması yapılırken kullanılacak algoritma büyük önem taşımaktadır. Yol planlama algoritması seçerken ortamın dinamik-statik olmasına, haritanın büyüklüğüne, çalışma ortamına vb. özellikler dikkate alınarak en uygun algoritmalar seçilmelidir.

3. YOL PLANLAMA ALGORİTMALARI

Yol planlama problemi, bir harita içerisinde bir başlangıç noktasından istenilen bir hedef noktasına, harita içindeki engellere çarpmadan en uygun uzunluktaki yolu bulmayı amaçlayan bir problem şekli olarak tanımlanabilir. Tanımda da bahsedildiği üzere yol planlama problemini çözerken dikkat edilmesi gereken bazı hususlar vardır. Bunlar;

- Oluşturulan yol engeller ile çakışmayacak ve aralarında belirli bir mesafe olacaktır. Engellerle mobil robotlar arasına konulacak mesafe belirlenirken bazı noktalara dikkat edilmesi gerekmektedir. Eğer konulan mesafe çok kısa olursa mobil robot engelin yanından geçerken engele sürtebilir hatta rotasından çıkabilir. Eğer konulan mesafe çok büyük olursa bu sefer ise oluşturulan yol optimum olmaktan uzaklaşır. Hatta rota hesaplanamayabilir.
- Oluşturulan yol robot için gidilebilir bir yol olmalıdır. Aşırı zikzaklı, aşırı engebeli ya da mobil robotun mekanik özelliklerine uygun olmayıp robotun geçemeyeceği yollar seçilmemelidir.
- Yukarıdaki koşullar sağlandıktan sonra algoritmanın bulduğu yollar içinden en kısa olanı seçilmelidir. Çünkü algoritmalar genel olarak birden fazla yol bulabilir ve bunların bazıları farklı açılardan birbirinden daha üstün olabilir (en kısa fakat en bozuk yoldan, en kısa fakat engellere çok yakın yoldan, en kısa fakat yakıt açısından yüksek maliyetli yoldan vb.). Bu yüzden yol oluştururken algoritmanın bulduğu yollar içinden yukarıdaki koşulları sağlaması şartıyla en kısa olan yol seçilmelidir. En kısa yolun seçilmesi hem zaman hem de yakıt maliyeti olarak verimlilik sağlamalıdır. Yakıt maliyeti de önemli bir giderdir ve elektrikli araçlar için önemli çalışmalar yapılmıştır [45].
- Yol oluşturulurken zamanlama optimum olmalıdır. En kısa yol oluşturulması için aşırı zaman harcanması verimsiz bir yöntemdir. Özellikle de gerçek zamanlı uygulamalarda uygulamayı zorlaştırır.

Aşağıdaki Şekil 3.1'de harita üzerinde bir başlangıç noktasından bir hedef noktasına çeşitli özelliklerde alternatif yollar verilmiştir. Bu yollar arasında değerlendirmeler yapılarak oluşturulan yolun kriterlere uygun olup olmadığı rahatça görülmektedir.



Şekil 3.1. Yol planlama problemine ait kriterler

Şekil 3.1'de harita üzerinde oluşturulmuş yollar yukarıda verilen yol planlama problemi çözülürken dikkat edilmesi gereken hususlara göre incelendiğinde; 2. yol herhangi bir engele çarpmadığı, engelle arasına belirli bir mesafe koyduğu ve yolu uzatmadan optimum uzunlukta bir yol olduğu için uygun bir yol olarak görülebilir. 3. yol haritadaki en kısa ve en düzgün yoldur fakat bu yol bir engel üzerinden geçtiği için kullanılamaz ve uygun olmayan bir yol olarak kabul edilir. 4. yol hemen hemen 2. yol ile aynı özelliklerdedir. Yol planlama algoritmaları bazen aynı uzunlukta ve aynı özelliklerde birden fazla yol bulabilirler. Bu yolların her birisi uygundur ve herhangi biri kullanılabilir. 1. yol herhangi bir engele çarpmadığı için kullanılabilir fakat hedefe daha kısa mesafelerden ulaşılabilme imkânı varken böyle uzun bir yol kullanmak optimum değildir. 5. yol hem optimum değildir hem de bir engel üzerinden geçtiği için kullanılamaz ve uygun olmayan bir yol olarak kabul edilir.

Sonuç olarak bakıldığında hedefe ulaşabilen 3 adet yol bulunmaktadır (1. yol, 2. yol ve 4. yol). Bu yollardan 1. yol uzunluk bakımından optimum olmadığı için 2. ve 4. yollardan birisi seçilmelidir. Bu 2 yolda yukarıda bahsedilen kriterler bakımından eşit olduğu için herhangi birisi seçilebilir.

Yukarıdaki uygulamada da görüldüğü üzere bir yol planlama probleminin çözümü için bir sürü parametrenin dikkate alınması ve bu parametrelerin hepsinin aynı anda sağlanmış olması gerekmektedir.

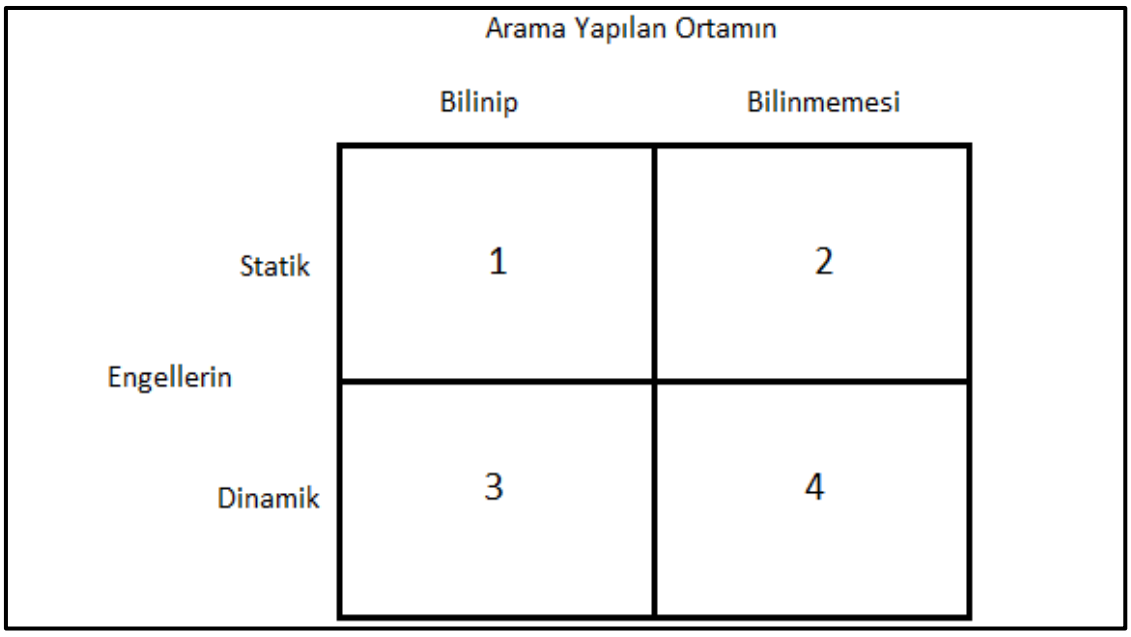
3.1. Yol Planlama Algoritmalarının Sınıflandırılması

Yol planlama problemlerinin çözümü için yıllardır birçok algoritma geliştirilmiştir [12]. Algoritmaların çalışma mantıklarına ve yöntemlerine göre birçok çeşidi vardır. Fakat araştırmacılar yol planlamada kullanılan çeşitli algoritmaları 2 ana sınıfa ayırırlar [46, 47]. Bunlar;

1. Çevre ve engellerin durumuna göre (statik, dinamik).
2. Yol planlama algoritmalarının çalışma mantığına göre (global, yerel).

Çevre ve engellerin durumuna göre

Yol planlama algoritmaları Şekil 3.2'de gösterildiği gibi arama yapılacak çevre ve çevredeki engellerin durumuna göre 4 farklı sınıfa ayrılırlar (Şekil 3.2) [46, 48]. Bunlar:



Şekil 3.2. Arama yapılacak çevrenin durumuna göre algoritmaların sınıflandırılması

- 1 numaralı bölge arama yapılan çevre ve engellerin yerlerinin bilinip engellerin statik olduğu durumu ifade eder. Mobil robot harekete geçmeden önce tüm haritayı analiz edip rotasını oluşturur ve sonra harekete geçer.
- 2 numaralı bölge arama yapılan çevre ve engellerin yerlerinin bilinmeyip ya da kısmen bilinip engellerin statik olduğu durumu ifade eder. Mobil robot elinde bulunan bilgilerle basit bir yol planlayıp harekete geçer. Robot ilerledikçe üzerindeki sensörler vasıtasıyla çevre hakkında yeni bilgiler öğrenir ve bu bilgiler sayesinde yeni yollar planlar. Bu şekilde yerel ve hızlı yol planlaması yaparak ve parça parça hareket ederek hedefine ulaşır.
- 3 numaralı bölge arama yapılan çevre ve engellerin yerlerinin bilinip engellerin dinamik olduğu durumu ifade eder. Mobil robot harekete geçmeden önce haritayı analiz edip rotasını oluşturur fakat engeller hareket edebildiği için üzerindeki sensörler (lazer, ultrasonik vb.) vasıtasıyla engellerden kaçma manevrası yapar ve engellerin çevresinden dolanarak hedefine ulaşır.
- 4 numaralı bölge arama yapılan çevre ve engellerin yerlerinin bilinmeyip ya da kısmen bilinip engellerin dinamik olduğu durumu ifade eder. Mobil robot 2. bölgedeki gibi hareket eder fakat engeller hareket edebildiği için üzerindeki sensörler (lazer, ultrasonik vb.) vasıtasıyla engellerden kaçma manevrası yapar ve engellerin çevresinden dolanarak hedefine ulaşır.

Yol planlama algoritmalarının çalışma mantığına göre

Yol planlama algoritmalarında çalışma mantığına göre global yol planlama ve yerel yol planlama olmak üzere iki farklı yöntem vardır.

Global yol planlama algoritmaları kullanılırken robotun çevre ve engellerin konumları hakkında tamamıyla bilgi sahibi olduğu varsayılır. Global yol planlama algoritması bu bilgi ile çevrimdışı çalışır ve daha robot harekete geçmeden tüm haritayı inceleyerek başlangıç ve hedef arasındaki en optimum yolu hesaplar. Böylece mobil robot hareket esnasında durmadan tek seferde hedefine ulaşabilir. Global yol planlama algoritmaları tüm haritayı ve engelleri incelediği için yerel yol planlama algoritmalarına göre daha uygun yollar bulabilir.

Yerel yol planlama algoritmaları kullanılırken robotun çevre ve engeller hakkında ya hiç bilgisi olmadığı ya da kısmen bilgisi olduğu varsayılır. Sadece başlangıç ve hedef noktalarını bilir. Robot yol planlamasını üzerinde bulunan sensörler (lazer, ultrasonik, kamera) ya da dışarıdaki bir bilgi kaynağından (GPS) elde ettiği bilgiler yardımıyla yapar. Yerel yol planlama algoritmaları çevrimiçi çalışır yani elinde bulunan bilgiler ışığında harekete geçer ve yolda elde ettiği bilgiler ile yeni rotasını parça parça oluşturarak hedefe doğru ilerler. Global yol planlama algoritmalarına göre daha hızlı çalışır fakat ona göre daha az optimum yollar bulabilir. Global ve yerel yol planlama algoritmaları arasındaki temel farklılıklar Çizelge 3.1'de gösterilmiştir [49].

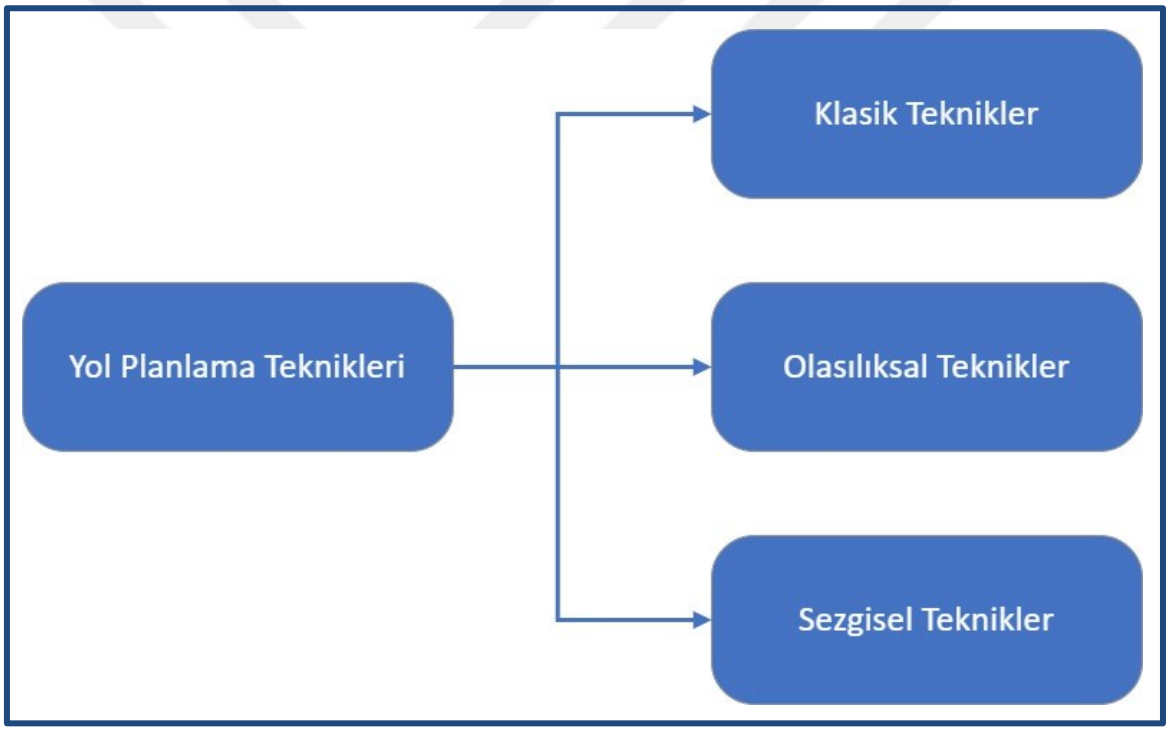
Çizelge 3.1. Global ve yerel yol planlama algoritmalarının karşılaştırılması

Global Yol Planlama	Yerel Yol Planlama
• Önceden verilmiş harita bilgilerini kullanır. • İhtiyatlı ve optimum bir şekilde hedefe gitmeyi planlar. • Süreç temkinli ve yavaştır. • Haritadaki her türlü bilginin verildiği durumlarda kullanılır. • Basit mobil robotların rota oluşturulmasında kullanılır. • Başlangıçta hedefe en uygun yolu hesaplar ve hedefe ulaşana kadar aynı rotayı takip eder.	• Sensörlerden elde edilen bilgileri kullanır. • Reaktif ve hızlı bir ulaşımı hedefler. • Süreç ani ve hızlıdır. • Haritadaki bilgilerin kısmen ya da hiç verilmediği durumlarda kullanılır. • Kompleks mobil robotların rota oluşturulmasında kullanılır. • Yerel bir şekilde yol oluşturur. Önüne çıkan engellerden kaçar ve yeni duruma göre yeni yollar oluşturup hedefine doğru ilerler.

Yukarıdaki çizelgede görüldüğü üzere global yol planlama algoritmaları yerel yol planlama algoritmalarına göre daha yavaştır ve daha uzun bir süreçte cevap verir. Bunun sebebi global yol planlama algoritmaları tüm haritayı inceler ve en uygun rotayı bulana kadar hesaplama yapar. Yerel yol planlama algoritmaları çok daha hızlı ve tepkiseldir. Fakat global yol planlama algoritmaları kadar optimum yol bulamazlar.

3.2. Yol Planlama Teknikleri

Yol planlama teknikleri uzun yıllardan beridir üzerinde çalışılan bir konudur. Teknolojinin ilerlemesi ile artan otonom araçlar bu ilerlemenin daha da hızlanmasına yol açmaktadır. Araştırmacılar daha iyi ve daha verimli yol planlama algoritmaları geliştirmek için yoğun bir çaba içerisindedirler. Araştırmacılar bu teknikleri geliştirirlerken yukarıda bahsedilen yol planlama algoritmalarının genel özelliklerinden (global-yerel, statik-dinamik engel, bilinen-bilinmeyen haritalar) yararlanmışlardır. Yol planlama teknikleri çalışma yöntemleri, kullanıldığı haritalar, engel türleri ve çözüm süreleri bakımından 3 ana başlık altında toparlanmıştır (Şekil 3.3).

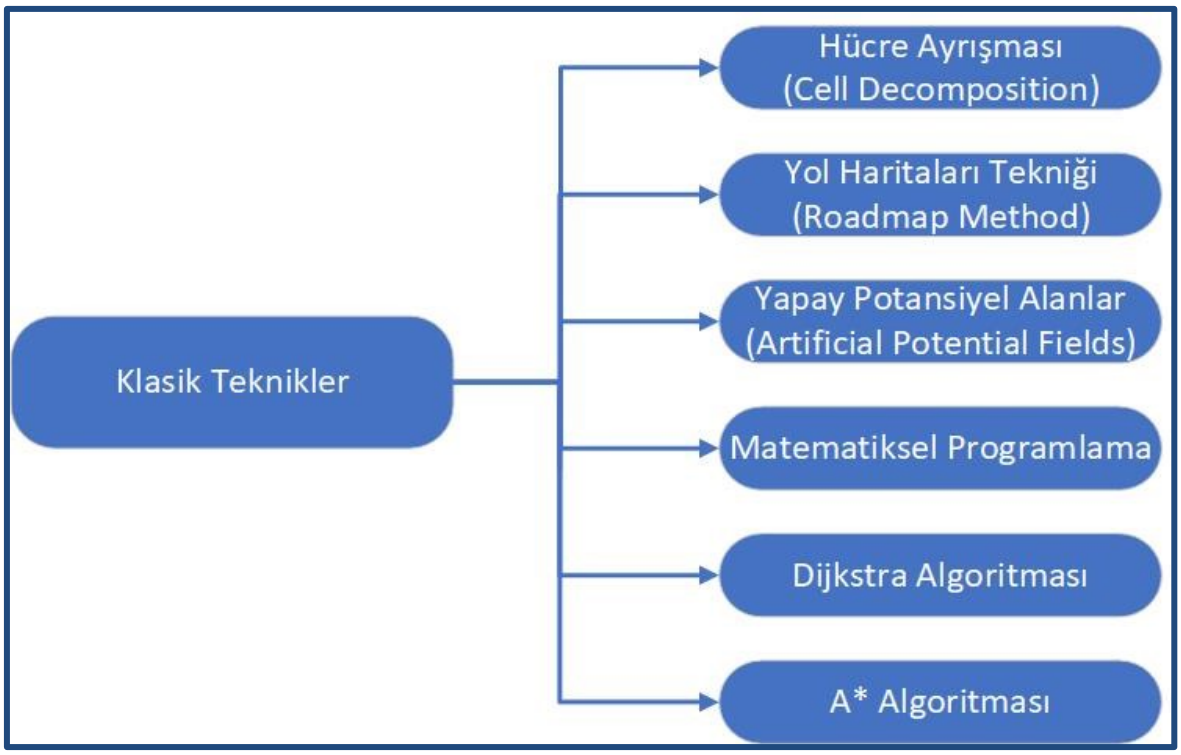


Şekil 3.3. Yol planlama teknikleri

3.2.1. Klasik teknikler

Mobil araçlarda kullanılan en eski yol planlama teknikleri klasik yol planlama algoritmalarıdır. Bu algoritmalar genellikle tamamı bilinen statik ortamlarda global yol planlama yapılacak haritalarda etkilidir. Eğer harita da uygun bir yol varsa yüksek ihtimal klasik yol planlama algoritmaları bu yolu bulur. Bu tekniğin dezavantajı ise fazla matematiksel işlem yaptığı için karmaşık ve büyük haritalar da hesaplama süresi çok

uzundur. Ayrıca az bilinen ya da hiç bilinmeyen haritalarda ve dinamik ortamlarda oldukça verimsiz bir yöntemdir [50]. Şekil 3.4'te klasik tekniklerle çalışan başlıca algoritmalar gösterilmiştir.



Şekil 3.4. Yol planlama algoritmalarında klasik teknikler

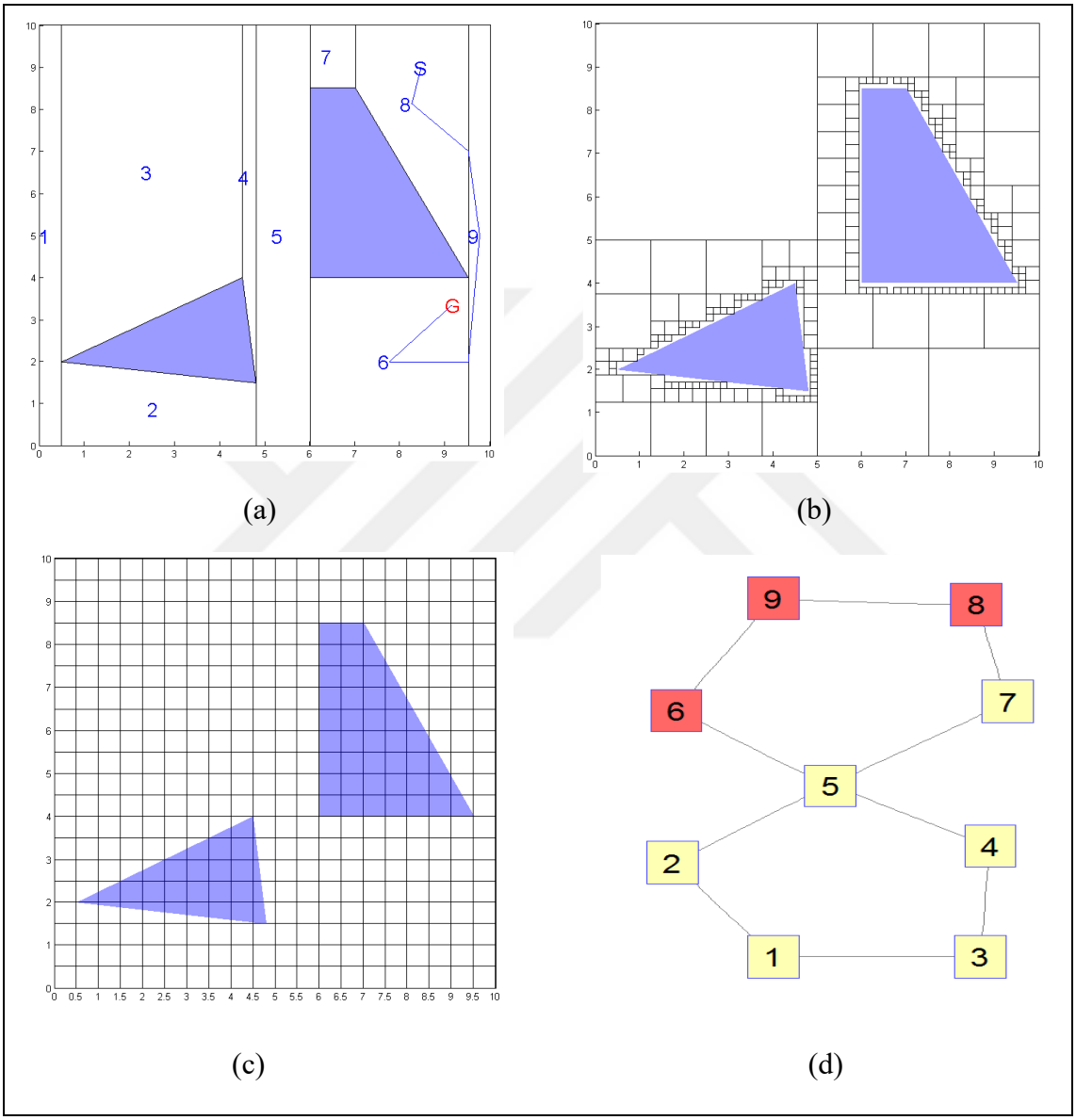
Hücre ayrışması (cell decomposition)

Hücre ayrıştırması tekniği yol planlama algoritmalarında sıklıkla kullanılan algoritmalardandır. İlk ortaya çıkan algoritmalardan olduğu için ilk zamanlarda sık sık kullanılmıştır. Hücre ayrıştırması tekniğindeki temel düşünce arama uzayındaki bölgeleri hücre şeklinde ayırarak arama alanlarını küçültmektir. Ayrıca engellerin olduğu bir haritada temiz hücreler (engellerin olmadığı alanlar) elde etmektir [51]. Hücre ayrıştırma tekniğinin çalışma prensibi şu şekildedir [52];

- Harita çeşitli yöntemlerle hücrelere ayrıştırılmaktadır.
- Hücreleri temsil edecek noktalar belirlenmektedir (genellikle hücre merkez noktasıdır).
- Başlangıç ve hedef noktalarının olduğu hücreleri de ekleyerek hücreleri komşuluk ilişkilerine göre birbirine bağlayacak şekilde bağlantısallık grafiği (connectivity graph) oluşturulmaktadır.

- Bağlantısallık grafiği uygun hareket planlama algoritmaları ile çözülerek başlangıç ile hedef nokta arası rota hesaplanmaktadır.

Şekil 3.5'te farklı şekillerde hücre ayrıştırma tekniği uygulanmış haritalar gösterilmiştir.



Şekil 3.5. Hücre ayrıştırması tekniği uygulamaları [53] (a: dikey şeritler kullanılarak oluşturulmuş hücre ayrıştırması, b: quad-tree approximation metodu, c: voxel approximation metodu, d: bağlantısallık grafiği (connectivity graph))

Hücre ayrıştırma tekniği çeşitli alt kategorilere ayrılmaktadır. Bunlar;

- Kesin hücre ayrıştırması tekniği (exact cell decomposition)
- Yaklaşık hücre ayrıştırma tekniği (approximate cell decomposition)

- Olasılıksal hücre ayrıştırma tekniği (probabilistic cell decomposition)

Yol haritaları tekniği

Yol haritaları belirlenen bir noktadan istenen bir noktaya nasıl varılacağını gösteren küçük rotalar olarak ifade edilebilir. Oluşturulan yol haritaları algoritmanın yol planlaması sırasında kullanacağı optimal yolların bir kümesi olarak kullanılır. Yol planlama algoritmaları planlama zamanında haritanın tamamını incelemek yerine kümede olan belirlenmiş özel rotaları kullanarak planlamayı gerçekleştirmektedir. Bundan dolayı yol haritaları tekniği hızlıdır ve kullanım alanları çoktur. Yol haritaları tekniği görünürlük grafiği (visibility graph), voronoi diyagramı (voronoi diagram), silüet (silhouette) ve alt hedef ağı (subgoal network) olmak üzere 4 başlık altında toplanmaktadır.

a) Görünürlük grafiği (visibility graph)

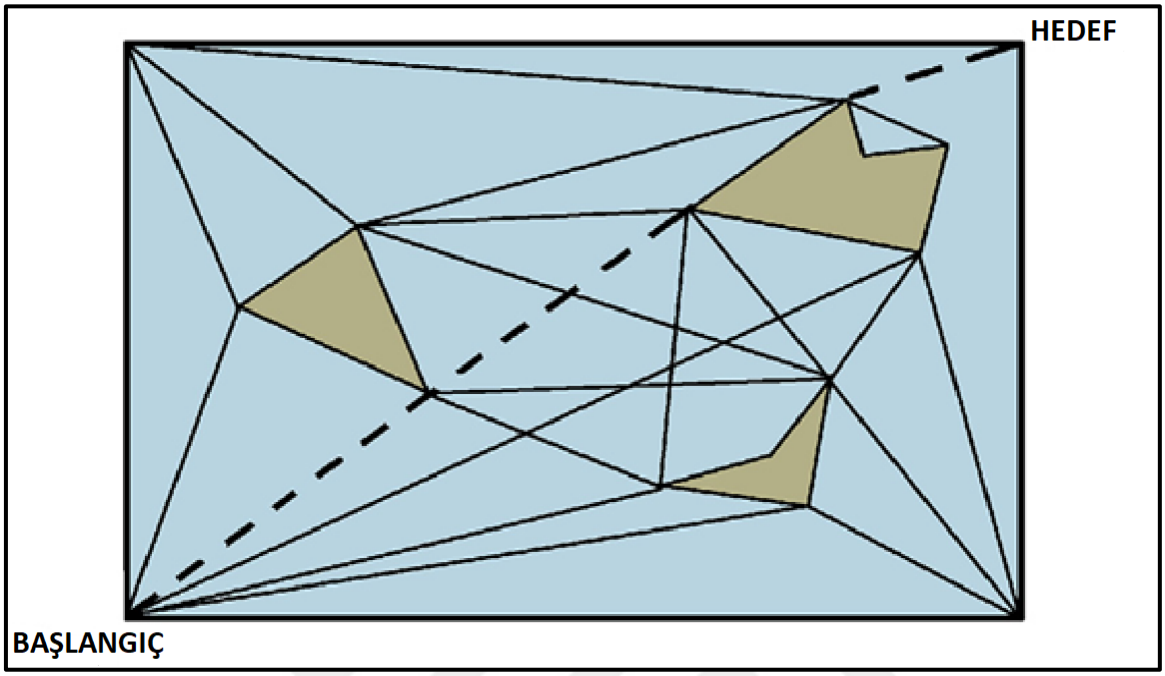
Görünürlük grafiği yol haritaları tekniklerinden biridir [51]. Bu tekniğin amacı engellerin olduğu bir harita da mobil robotun başlangıç ile hedefi arasındaki yolunu planlarken engellerden arındırılmış bir harita sunmasıdır. Bu teknik çoğu haritaya uygulanabilir fakat en iyi sonuç köşegen engellerin olduğu haritalardan alınır. Bu teknik uygulanırken her engelin köşe noktaları işaretlenir ve birbirini gören köşe noktaları düz bir çizgi ile birleştirilir. Bu tekniğe görünürlük grafiği denmesinin sebebi de budur. Bu çizgilere başlangıç ve hedef noktaları da eklenir. Sonra bu çizgiler üzerinden rota hesaplanır [54]. Bu tekniğin avantajları;

- Haritada eğer başlangıç ve hedef noktası arasında bir yol varsa mutlaka bulur.
- Basit haritalarda hızlıdır.

Dezavantajları:

- Tehlikelidir. Bulduğu rotalar engellerin çok yakınından geçtiği için kaza riski yüksektir.
- Karmaşık haritalarda maliyeti yüksektir.

Şekil 3.6'da görünürlük grafiği ile ilgili bir örnek harita gösterilmiştir. Haritada sarı alanlar engelleri göstermektedir.



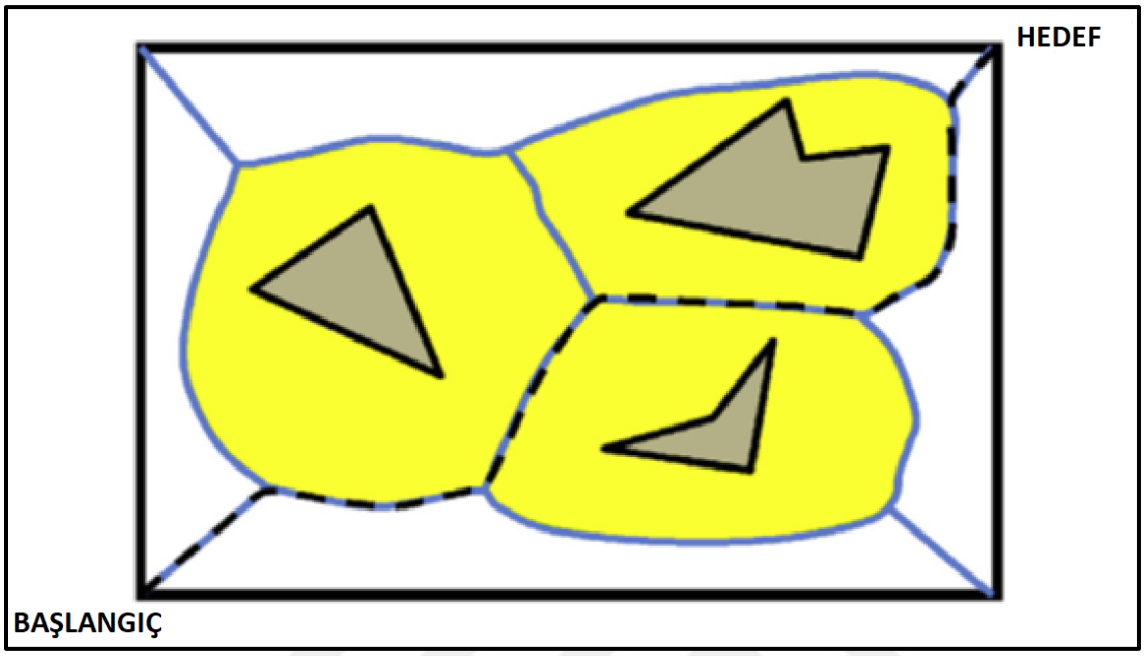
Şekil 3.6. Görünürlük grafiği [55]

b) Voronoi diyagramı (Voronoi diagram)

Voronoi diyagramı yol haritaları tekniklerinden bir diğeridir. Görünürlük grafiği ile aynı haritalarda benzer yöntemlerde çalışırlar fakat bazı açılardan birbirlerinden farklıdırlar. Voronoi diyagramı görünürlük grafiğinin aksine her noktaya eşit uzaklıkta yaylar kullanır. Bırakılacak mesafe mobil robotun güvenli bir şekilde geçebileceği kadar açık olmalıdır. Yaylar oluşturulduktan sonra 2 adet çizgi çekilir. Bunlardan birisi başlangıç noktası ile en yakın yay arasında bir diğeri ise hedef noktası ile en yakın yay arasındadır. Sonra bu çizgiler üzerinden rota planlaması yapılır [54]. Bu tekniğin avantajları;

- Haritada eğer başlangıç ile hedef noktası arasında bir yol varsa mutlaka bulur.
- Basit haritalarda hızlıdır.
- Uygulanabilirliği yüksektir. Oluşturulan yol keskin manevralar barındırmadan hafif eğimli yollar olduğu için robot rahat kullanabilir.

Şekil 3.7'de Voronoi diyagramı ile ilgili bir örnek harita gösterilmiştir. Haritada koyu sarı alanlar engelleri göstermektedir. Siyah-mavi renkli çizgiler bulunan yolu göstermektedir.

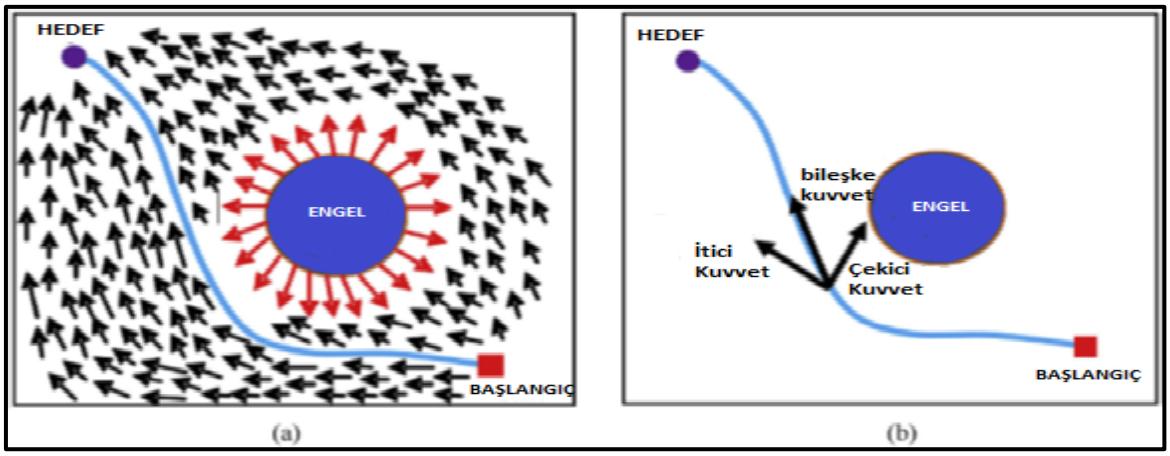


Şekil 3.7. Voronoi diyagramı [55]

Yapay potansiyel alanlar (Artificial potential fields)

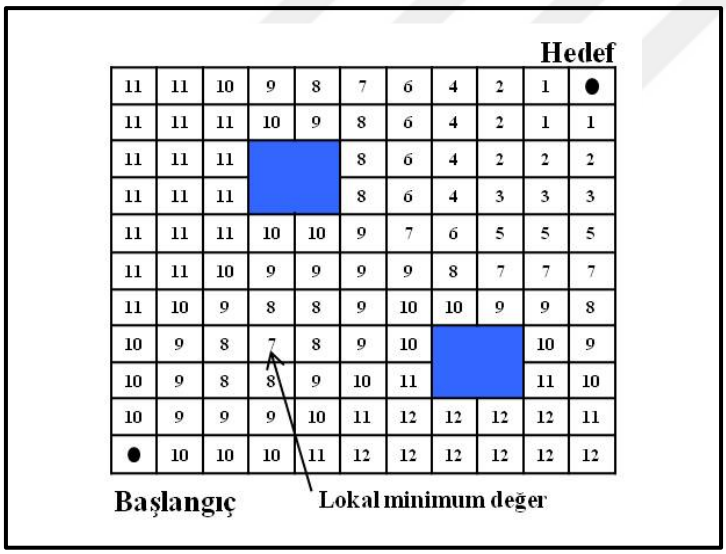
Yapay potansiyel alanlar yöntemi iticilik ve çekicilik olmak üzere iki genel kuvvetin hesaplanması ile oluşturulur. Eğimli bir arazide bırakılan topun aşağı yuvarlanması, mıknatısın aynı kutuplarının birbirini itip zıt kutuplarının birbirini çekmesi gibi doğadaki örnekleri ile benzer şekilde çalışır. Yapay potansiyel alanlar yönteminde ise mobil robota engeller yapay bir şekilde iticilik sağlarken hedef nokta çekicilik sağlar. Robot bir sonraki hareketini bu kuvvetlerin hesaplaması ile belirler ve bu şekilde ilerleyerek hedefine ulaşır [52, 55].

Şekil 3.8'de görünürlük grafiği ile ilgili örnek bir harita gösterilmiştir. Şekilde de görüldüğü gibi engeller robota iticilik sağlarken hedef noktası robota çekicilik sağlamaktadır.



Şekil 3.8. Yapay potansiyel alanlar yöntemi yaklaşımı [55]

Yapay potansiyel alanları yönteminin en büyük sıkıntısı yerel minimuma düşme olasılığının yüksek olmasıdır. Aşağıdaki şekil 3.9'da bu sıkıntının bir örneği gösterilmiştir. Yöntemin bulunmasından sonra yapılan çalışmalar genellikle bu sıkıntının çözümü üzerinedir [56].



Şekil 3.9. Yapay potansiyel alanlar yönteminde yerel minimuma düşme [56]

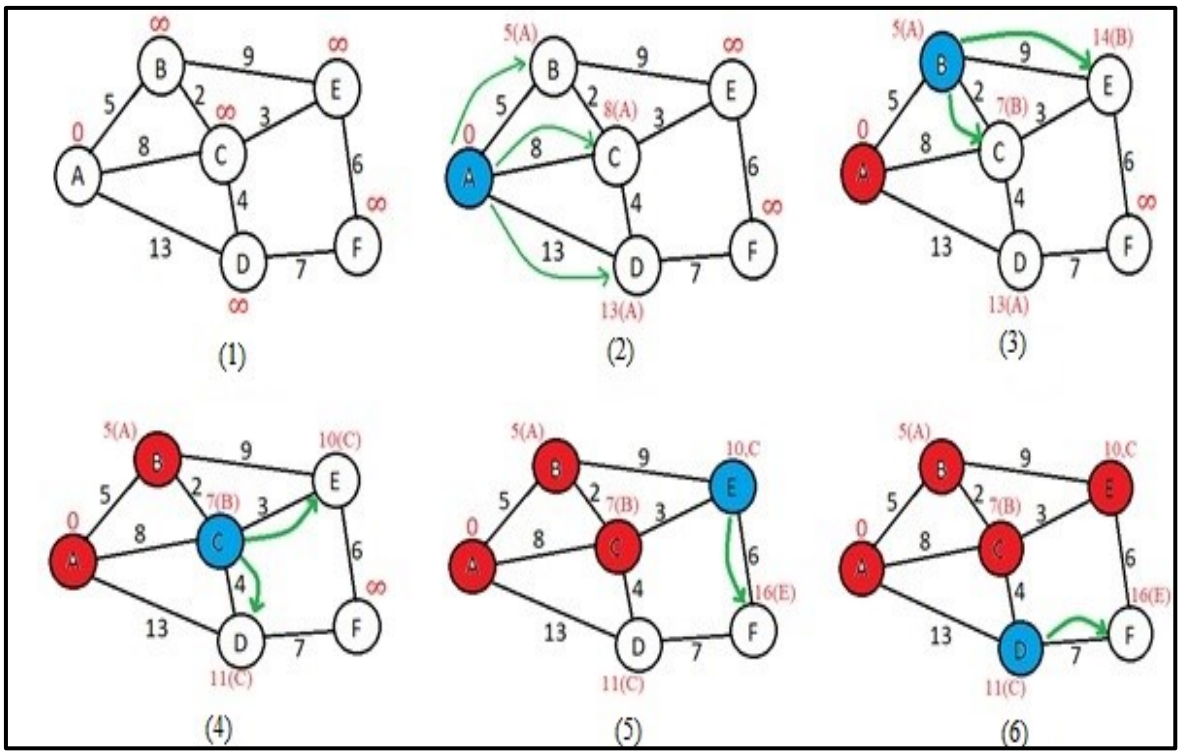
Dijkstra algoritması (dijkstra algorithm)

Dijkstra algoritması en çok kullanılan yol planlama algoritmalarından bir tanesidir. Algoritma bilgisayar bilimcisi Edsger W. Dijkstra tarafından bulunmuştur ve 1959 yılında yayımlanmıştır [57]. İsmini bulucusundan almıştır. Dijkstra algoritması başlangıç noktasından harita üzerindeki tüm noktalara en kısa yolu bulmak için kullanılmaktadır.

Algoritma başlangıç noktasından hedef noktasına giderken harita üzerindeki çoğu noktayı tek tek inceler. Bu işlemin hem avantajı hem de dezavantajı vardır. Algoritma harita üzerindeki çoğu noktayı incelediği için eğer bir yol var ise mutlaka bulur fakat büyük ölçekli haritalar da arama alanı çok artacağı için arama süresi çok yüksek olacaktır. Dijkstra algoritmasının çalışma mantığı aşağıdaki gibidir [58];

- Harita üzerindeki bütün noktalara bir maliyet değeri atanır. Başlangıç noktası için 0, diğer bütün noktalar için sonsuz değeri atanır.
- Ziyaret edilen ve edilmeyen olarak 2 adet liste oluşturulur. Bu 2 liste taranan ve taranmayan noktaları ayırmak için kullanılır ve ziyaret edilen bir noktanın tekrar ziyaret edilmesini engeller. Harita üzerindeki bütün noktalar ziyaret edilmeyen listesine atanır. Ayrıca ziyaret edilen noktaların maliyetlerinin ve ebeveynlerinin (üzerinde bulunduğu noktaya gelmeden önce bulunduğu nokta) tutulduğu liste de oluşturulmalıdır.
- Tarama işlemine ilk olarak başlangıç noktasından başlanılır. Başlangıç noktasının bütün komşuları incelenir. Komşuların maliyet değerleri hesaplanır ve ebeveyn ismi ile birlikte listeye eklenir. Başlangıç noktasının tüm komşuları incelendikten sonra başlangıç noktası ziyaret edilen listesine alınır.
- Sıradaki taranacak nokta olarak, ziyaret edilmeyen listesindeki maliyeti en düşük nokta seçilir. Sıradaki noktanın komşu noktaları incelenir. Her noktanın maliyet değerleri hesaplanır ve taranan noktanın maliyet değeri ile toplanarak komşu noktaların maliyet değeri bulunur. Eğer komşu nokta daha önce ziyaret edilmemiş ise bulunan maliyet değeri ve ebeveyn ismi listeye eklenir. Fakat komşu nokta daha önce ziyaret edilmiş ise komşu noktanın maliyet değeri maliyet listesine eklenirken 2 ihtimal söz konusudur. Komşu noktanın yeni bulunan maliyet değeri eski değerden daha büyük ise hiçbir değişiklik yapılmaz. Eğer yeni değer eski değerden daha küçük ise listedeki komşu noktanın maliyet değeri ve ebeveyn ismi yenileri ile değiştirilir.
- Taranan nokta ziyaret edilen listesine alınır ve ziyaret edilmeyen listesindeki yeni nokta ile tarama işlemine devam edilir.
- Tarama işlemi hedef nokta ziyaret edilen listesine eklenene kadar ya da ziyaret edilmeyen listesi boşalana kadar devam eder. Hedef nokta ziyaret edilen listesine eklendikten sonra hedef bulunmuş olur. Hedef noktasından başlangıç noktasına ulaşana kadar ebeveyn noktalar takip edilerek başlangıç ile hedef noktası arasındaki en kısa yol oluşturulur ve bu yolun maliyeti hesaplanır.

Aşağıdaki Şekil 3.10'da Dijkstra algoritmasının çalışmasının bir örneği gösterilmiştir. Örnekte A noktasından harita üzerindeki bütün noktalara en az maliyetle nasıl gidileceği bulunmuştur.



Şekil 3.10. Dijkstra algoritması çalışma örneği

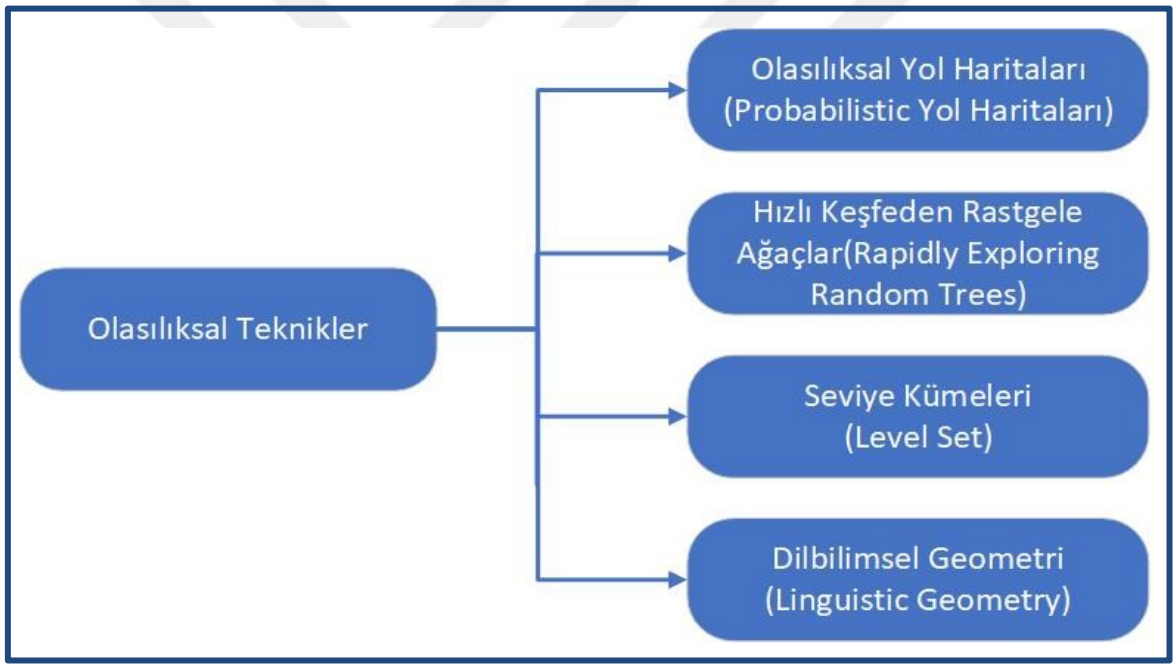
Dijkstra algoritması negatif değerli maliyete sahip haritalarda düzgün çalışmaz. Böyle durumlarda Bellman Ford algoritmasının kullanılması önerilmektedir [59].

A algoritması*

A* algoritması, sezgisel bir yaklaşım algoritmasıdır. İlk olarak Stanford Araştırma Enstitüsündeki araştırmacılardan Peter Hart, Nils Nilsson ve Bertram Raphael tarafından 1968 yılında sunulmuştur [60, 61]. Bu algoritma rota planlama uygulamalarında kullanılan en popüler yöntemlerden bir tanesidir. Dijkstra algoritmasının bir uzantısı olarak görülebilir. A* algoritmasının Dijkstra algoritmasından farkı, A* algoritmasında maliyet hesaplanırken, Dijkstra algoritmasının çalışma prensibine ek olarak sezgisel maliyette hesaba katılır. Böylece A* algoritmasında tarama alanı her yöne yayılmadan doğruca hedefe doğru yönlendiği için daha hızlı ve daha verimli rota hesaplanabilir. A* algoritması 4.1. bölümde detaylıca anlatılmıştır.

3.2.2. Olasılıksal teknikler

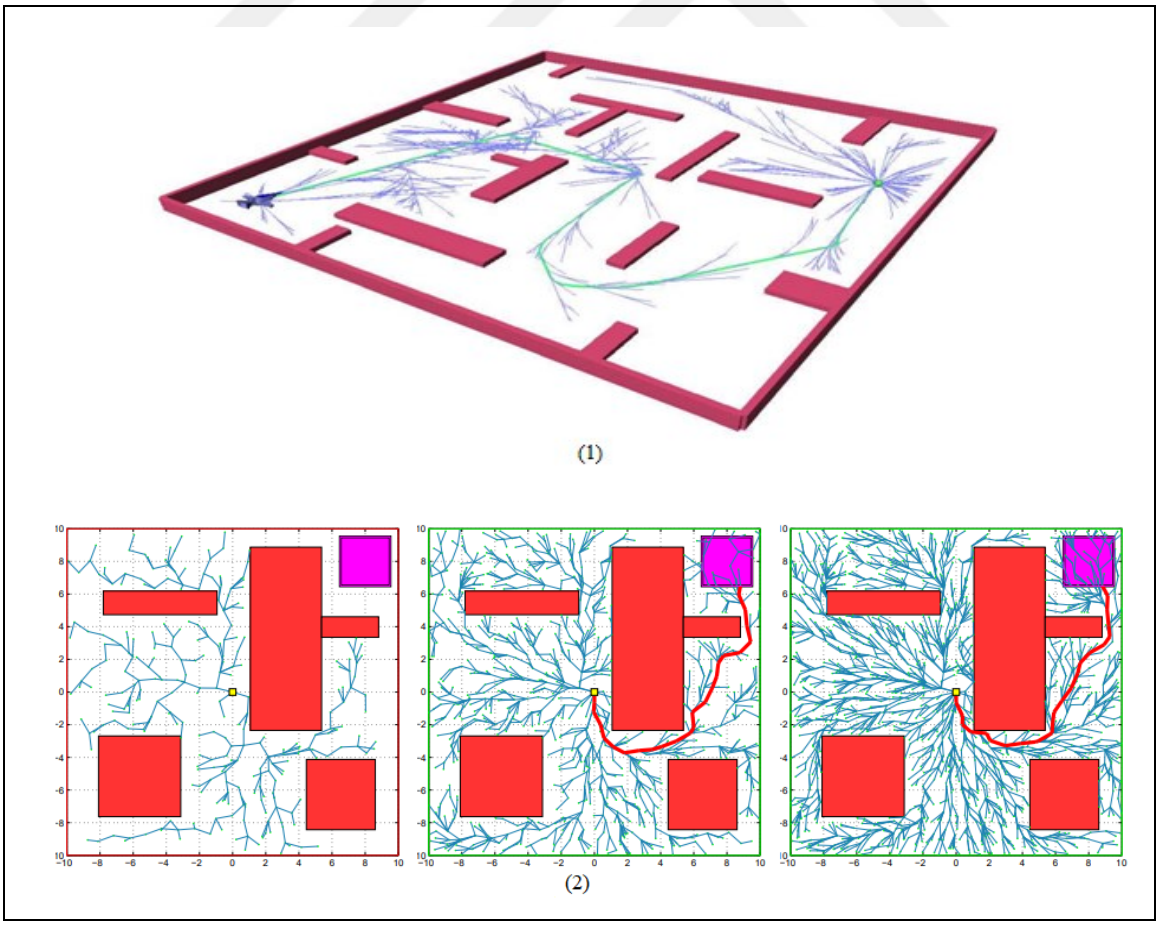
Olasılıksal teknikler yol planlama tekniklerinden bir diğeridir. Genel olarak klasik teknikler başlığı altında geçmektedir fakat çalışma mantığı klasik tekniklerden farklı olduğu için daha rahat anlaşılması açısından farklı başlık altında verilmiştir. Olasılıksal teknikler yol planlama algoritmalarında sıklıkla kullanılan tekniklerdendir. Arama alanında uygun yerlere (engellerin olmadığı vb.) rastgele noktalar atar ve yol planlamayı bu atanan noktalar üzerinden yapar. Çalışma şekli basittir ve kısa sürelerde sonuç elde ederler. Dezavantajları ise bulunan yol dalgalı ve düz olmayan bir yoldur. Olasılıksal tekniklere olasılıksal yol haritaları, hızlı keşfeden rastgele ağaçlar, seviye kümeleri, dilbilimsel geometri vb. gibi algoritmalar örnek verilebilir (Şekil 3.11).



Şekil 3.11. Olasılıksal teknikler

Hızlı keşfeden rastgele ağaçlar (RRT)

Hızlı keşfeden rastgele ağaçlar (RRT) yöntemi olasılıksal yol planlama tekniklerinde en çok kullanılan 2 yöntemden birisidir. Steven M. LaValle ve James J. Kuffner Jr. tarafından geliştirilmiştir [62]. RRT yönteminde amaç yol planlanırken rastgele noktalar seçerek başlangıç noktasından başlayan ve hedef noktasına kadar ulaşan bir ağaç oluşturmaktır. Bu yöntemi uygularken algoritma harita üzerinde rastgele yerlere noktalar atar ve bu nokta ağaç üzerinde en yakın noktadan birleştirilerek yeni dallanmalar oluşturur. Algoritma iki noktayı birleştirmek için maximum bir mesafe belirler. Bu mesafe algoritmanın oluşturduğu yolun yumuşaklığını ve algoritmanın hesaplama hızını etkiler. Eğer atanan nokta ile ağacın en yakın noktasının arasındaki mesafe belirlenen maximum mesafeden fazla ise eklenecek yeni dal ağacın en yakın noktasından atanan nokta hizasında belirlenen maximum mesafe eklenerek oluşturulur. Aşağıdaki Şekil 3.12'de çeşitli RRT algoritması uygulamaları gösterilmiştir.



Şekil 3.12. RRT algoritması [63, 64]

Algoritma hedef noktaya belirli bir mesafe yaklaşana kadar ya da bulunan yol yeterli uygunlukta (oluşan yolun zikzaklı olmaması ve mobil robotun rahat ilerleyebilecek bir şekilde olması) olana kadar yeni dallar oluşturmaya devam eder. Genişleyen ağaç haritanın her tarafına yayıldığı için harita da ulaşılmayan nokta bırakmaz. RRT algoritması eğer başlangıç ile hedef noktası arasında bir yol varsa ve yeterli zaman verilirse optimum bir yol mutlaka bulur. RRT algoritmasının hem avantajları ve hem de dezavantajları vardır. Avantajları [59];

- Geniş bir çalışma alanı vardır. Tarama esnasında tüm haritayı tarayabilir.
- Basit ve hızlı bir algoritmadır.
- Global ve yerel uygulamalarda çalışabilir.
- Dinamik ortamlarda çalışabilir.

Dezavantajları:

- Deterministik değildir. Rastgele noktalar atadığı için her çalışmasında benzer fakat farklı bir yol bulur.
- Dar alanlarda çalışma verimi düşüktür.
- Bulunan yol keskin ve zigzaglıdır.

RRT algoritması araştırmacıların sürekli üzerinde çalıştıkları bir algoritmadır. Üzerinde yukarıdaki dezavantajları iyileştirecek şekilde sürekli gelişmeler mevcuttur. RRT* ya da Bi-RRT gibi algoritmalar geliştirilmiştir. Normal bir RRT algoritması başlangıç noktasından dallanmaya başlar ve hedefe ulaşana kadar başlangıç noktasını kök olarak kullanır. Fakat Bi-RRT algoritmasında dallanma hem başlangıç noktasından hem de hedef noktasından başlar ve bu 2 kök üzerinden devam eder. 2 dallanmanın birleştiği yerde rota bulunmuş olur. Böylelikle algoritma 2 kat hızlı çalışmış olur [4].

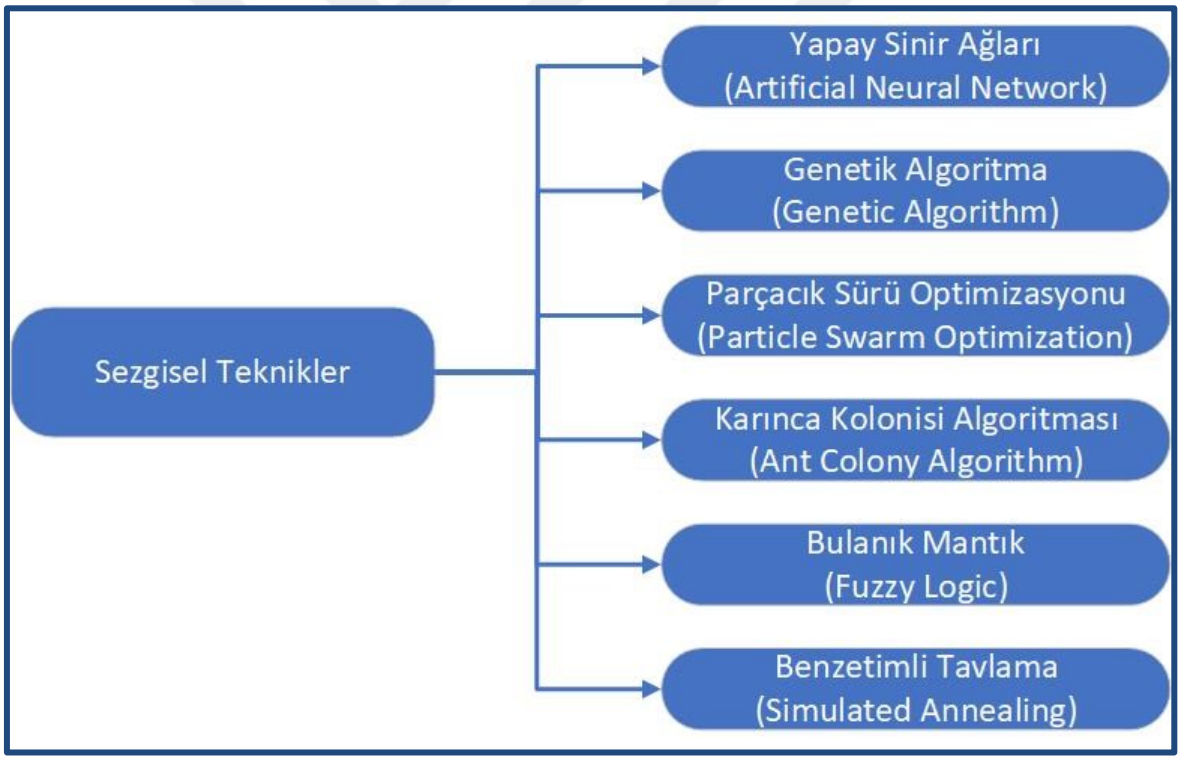
PRM algoritması

Olasılıksal yol haritaları algoritması (PRM) olasılıksal tabanlı en kısa yol planlama algoritmalarından birisidir. Lydia E. Kavraki tarafından geliştirilmiştir [65]. RRT ile birlikte olasılıksal yöntemlerin içinde en çok kullanılan yöntemlerdendir. Diğer yol planlama

algoritmalarına göre daha hızlı çalışır ve daha kısa sürelerde sonuç verir. Rastgele atanan noktalar haritanın her tarafına dağıldığı için algoritma arama alanının tümüne hâkim olabilir. PRM algoritması 4.2. bölümde detaylıca anlatılmıştır.

3.2.3. Sezgisel teknikler

Sezgisel teknikler yol planlama tekniklerinden bir diğeridir. Sezgisel algoritmalar yol planlama yaparken doğal fenomenleri kullanan algoritmalardır. Bu algoritmalar kesin çözümü garanti etmemektedirler ancak makul bir süre içerisinde kesin çözüme yakın bir çözümü garanti etmektedirler. Şekil 3.13'te gösterildiği gibi sezgisel algoritmalara yapay sinir ağları, genetik algoritma, parçacık sürü optimizasyonu, karınca kolonisi algoritması, bulanık mantık, benzetimli tavlama vb. gibi birçok algoritma örnek verilebilir.



Şekil 3.13 Sezgisel yol planlama teknikleri

Parçaçık sürü optimizasyonu (particle swarm optimization)

Parçacık sürü optimizasyonu (PSO) sezgisel tekniklerde en çok kullanılan algoritmalardan birisidir. 1995 yılında James Kennedy ve Russell Eberhart tarafından geliştirilmiştir [66]. Kuş veya balık sürülerinin yiyecek bulmak için kullandığı yöntemlerden (sürü içinde birbiri ile haberleşme, birbirlerini yiyeceğe doğru yönlendirme, güvenlik gibi) esinlenilerek geliştirilmiştir. Sürü zekasına dayanan bir algoritmadır. PSO algoritması içindeki her bir bireye parçacık denmektedir. Sürü içindeki parçacıklar her hareketinde hem kendi en iyi pozisyonuna hem de sürü içindeki en iyi pozisyondaki parçacığın pozisyonuna doğru hareket etme eğilimdedirler [67].

$$v_{id+1} = (w) \times (v_{id}) + c_1 \times rand() \times (p_{best} - x_{id}) + c_2 \times rand() \times (g_{best} - x_{id}) \quad (3.1)$$

$$x_{id+1} = x_{id} + v_{id} \quad (3.2)$$

Eşitlik 3.1 ve 3.2'deki formüller PSO algoritmasında parçacıkların uygunluk değerlerinin bulunmasını sağlayan fonksiyonlardır. Buradaki simgeler;

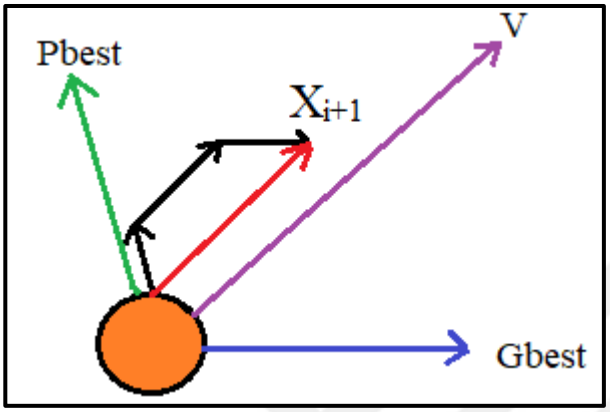
- x = Parçacığın konumu ya da değeri (kullanılan örneğe göre değişir)
- v = Parçacığın değişim hızı
- c_1, c_2= Sabit değerler (Çoğunlukla 2 tercih edilir)
- rand() = rastgele üretilen değerler
- p_{best} = Parçacığın tüm hareketi boyunca en iyi değeri (yerel eniyi)
- g_{best} = Tüm parçacıklar içindeki en iyi değer (global eniyi)
- w = Atalet ağırlık değeri

Algoritma temel olarak şu şekilde çalışmaktadır:

1. Parçacık adedi, başlangıç değeri, iterasyon sayısı, C_1, C_2 ve W değeri belirlenerek başlanır.
2. Sürü içindeki her bir parçacığın uygunluk değeri hesaplanır.
3. Sürü içindeki her bir parçacığın P_{best} değeri hesaplanır. P_{best}'lerin sayısı parçacık sayısı kadardır.

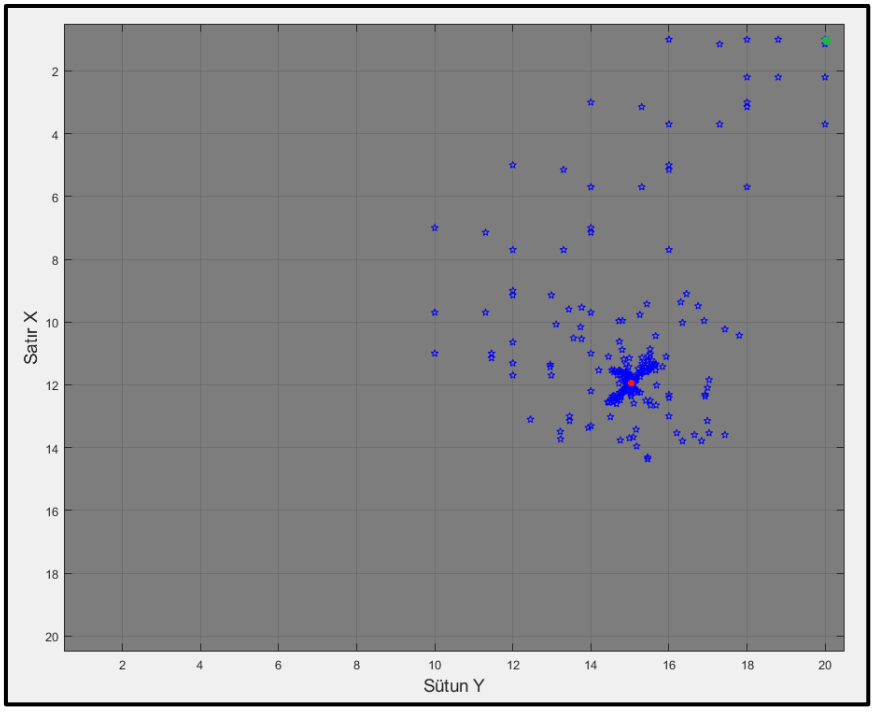
4. Mevcut iterasyonda tüm P_{best}'ler içinden en iyi değeri seçilir (G_{best}).
5. Uygunluk fonksiyonu uygulanarak konum ve hızlar yenilenir.
6. Durdurma kriteri sağlanıncaya kadar 2'den 5'e kadar ki süreçler tekrar edilir.

Aşağıdaki şekil 3.14'te değişim hızı, P_{best} ve G_{best}'i hesaplanan bir parçacığının yeni konumunun nasıl bulunduğu gösterilmiştir.



Şekil 3.14. PSO yeni konum hesaplaması

Aşağıdaki Şekil 3.15'te PSO algoritmasındaki parçacıkların hedef noktasını bulması gösterilmiştir.



Şekil 3.15. PSO algoritması ile hedefi bulma

Durdurma kriterleri; algoritmanın belirlenen iterasyon sayısına ulaşması, hedefin bulunması ya da çok yaklaşılması veya belirli bir miktar iterasyon sonrasında G_{best}'in değişmemesidir. Bu kriterlere ulaşıldığında algoritma durdurulmalıdır [52].

Genetik algoritma (genetic algorithm)

Genetik algoritma (GA) doğada bulunan doğal seçilim ilkesinin bilgisayar sistemlerine modellenilmesi ile oluşturulmuş bir algoritmadır. İlk defa 1975 yılında Michigan Üniversitesinden John Holland tarafından ortaya atılmıştır. Temel prensibi topluluktaki en güçlü üyelerin hayatta kalıp en zayıfların yok olmasına dayanmaktadır [68].

Genetik algoritmada biyolojiden esinlenilerek oluşturulmuş çeşitli kavramlar mevcuttur. Bunlar;

- Gen: Uygulanan probleme göre değişiklik göstermektedir. En kısa tabiri ile probleme ait bilgi taşıyan en küçük birimdir.
- Kromozom: Birçok genin belirlenen bir düzenle bir araya gelerek oluşturduğu diziye denir. Problem içindeki farklı çözüm adaylarıdır.
- Popülasyon: Kromozomların oluşturduğu topluluğa denir. Aday çözümlerin toplandığı çözüm kümesidir.

Genetik algoritmanın çözümünde kullanılan 2 adet genetik işlemci vardır. Bunlar;

- Çaprazlama: Popülasyon içinden seçilen 2 adet çözüm adayının (kromozom) belirli genlerinin karşılıklı değiştirilmesi işlemidir. Böylece 2 adet yeni çözüm oluşmaktadır ve çözüm çeşitliliği arttırılmıştır.
- Mutasyon: Popülasyon içindeki sadece 1 çözüm adayının belirli genlerinin değiştirilmesi işlemine denir. Bu işlem popülasyonda çeşitliliği sağlar ve algoritmanın yerel minimumlara takılmasını önler.

Genetik algoritma çoğunlukla aşağıdaki şartlarda tercih edilmektedir:

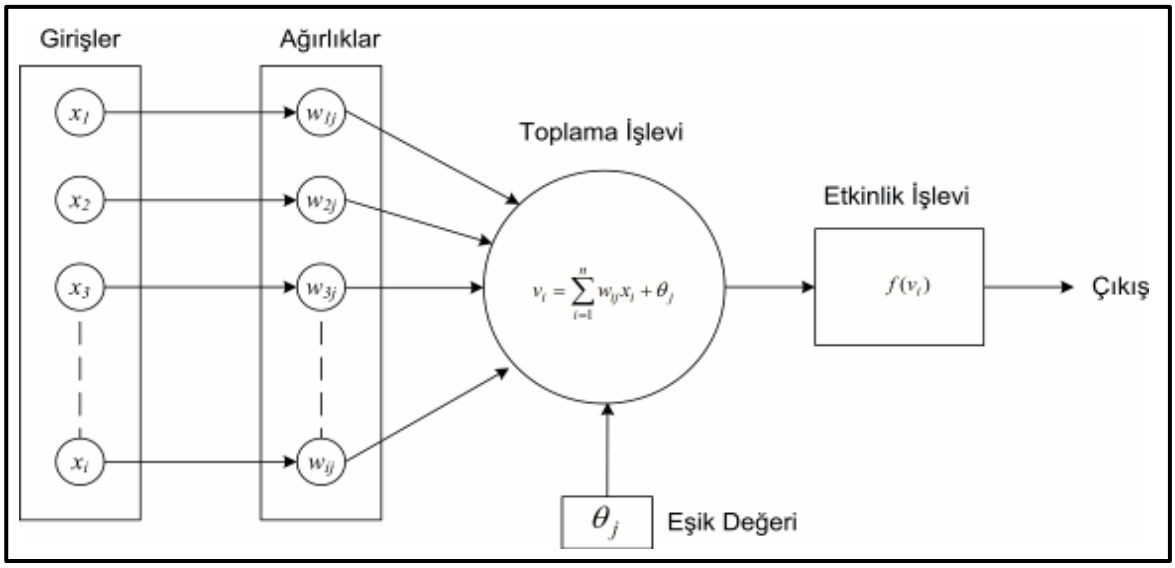
- Arama alanının geniş ve karışık olduğu ortamlarda
- Klasik yöntemler ile çözümünün mümkün olmadığı ya da çözüm süresinin problemin büyüklüğü ile üstel oranda arttığı ortamlarda
- Mevcut bilgi ile çözümün zor olduğu ortamlarda

Genetik algoritma şu şekilde çalışmaktadır:

- Aday çözümlerin (kromozom) olduğu bir çözüm kümesi oluşturulur (başlangıç popülasyonu). Popülasyonda bulunması gereken aday sayısı için bir standart yoktur. Aday sayısı problemin büyüklüğüne ve derinliğine göre değişmektedir. Problemin çözüm süresini etkiler.
- Her aday çözümünün uygunluk değeri uygunluk fonksiyonu kullanılarak hesaplanır. Uygunluk fonksiyonu problem içindeki bir adayın çözüm olup olmayacağını gösteren bir fonksiyondur. Uygunluk fonksiyonu her problem için aynı değildir ve probleme özgü oluşturulmalıdır. Genetik algoritmanın başarısının en önemli etkenlerinden birisi uygunluk fonksiyonunun iyi oluşturulmasına bağlıdır.
- Oluşturulan çözümler durdurma kriterleri üzerinden incelenir. Durdurma kriterleri; istenen çözümün bulunulması, belirlenen iterasyon sayısına ulaşılması veya iterasyonlar boyunca uygunluk değerinin aynı kalmasıdır. Eğer oluşturulan çözümlerde durdurma kriterleri sağlanırsa algoritma durdurulur. Çözüm bulunmuş ise elde edilen çözüm global çözüm olarak alınır. Eğer durdurma kriterleri sağlanmıyorsa arama işlemlerine devam edilir.
- Uygunluk değerine göre seçilen adaylara çaprazlama ve mutasyon işlemleri uygulanarak yeni bireyler oluşturulur. Algoritma 2. adıma gider ve durdurma kriterleri sağlanana kadar döngü devam eder.

Yapay sinir ağları (artificial neural network)

İnsanlar doğumlarından itibaren öğrenme sürecine girmektedirler. Öğrenme süreçleri yaşayarak, görerek, inceleyerek ve bunların sonuçlarını irdeleyerek olmaktadır. Canlılarda öğrenme nöronlar ve nöronlar arasındaki sinaptik bağlantıların düzenlenmesi sayesinde oluşur. Beyne giren her yeni bilgide sinaptik bağlantılar ayarlanır. Yapay sinir ağları ise insan beyninin bu doğal yaşayarak öğrenme sürecinin çeşitli problemleri çözmek için bilgisayar sistemlerine modellenerek oluşturulan bilgi işlem teknolojisidir. Bir yapay sinir ağının yapısı Şekil 3.16'daki gibidir.



Şekil 3.16. Yapay sinir ağı yapısı [69]

Giriş bölümü (X_i) sinir ağına dışarıdan gelen verilerin alındığı bölümdür. Biyolojik sistemlerde duyu organları, yapay sistemlerde sensörlerdir. Ağırlıklar bölümü (W_{ij}) biyolojik sistemlerdeki sinaptik bağlantılardır ve sisteme giren verilerin önemini gösterir. Bu değerin yüksek olması buradaki sinaptik bağın güçlü olduğu ve gelen verinin önemli olduğunu, düşük olması buradaki sinaptik bağın zayıf olduğu gelen verinin fazla önemli olmadığı anlamına gelmektedir. Girişten gelen veriler belirlenen ağırlıklar ile çarpılarak toplanır. Bu değer eşik değerini aşabilir ise etkinlik işlevine aktarılır ve bu bölümde işlenerek sonuç çıkışa aktarılır. Yapay sinir ağı sistemleri öğrenme süreçlerinde işlediği veriler ile probleme uygun ağırlık, toplama fonksiyonu, eşik değeri ve etkinlik fonksiyonunu belirler. Bu parametrelerin probleme en uygun şekilde oluşturulması yapay sinir ağlarının başarısının anahtarıdır [69, 70].

4. KLASİK A* ALGORİTMASI İLE PRM TABANLI A* ALGORİTMASININ İNCELENMESİ

A* algoritması bir klasik yol planlama algoritmasıdır. Çok verimli bir yöntemdir ve eğer harita da başlangıç ile bitiş noktası arasında bir yol mevcut ise A* algoritması bu yolu yüksek ihtimalle bulur. Bu özelliklerinden dolayı en kısa rota hesaplama algoritmalarının içinde en çok kullanılan algoritmalardan birisidir. Ancak A* algoritmasının büyük dezavantajlarından birisi büyük ölçekli haritalarda rota hesaplama süresinin çok yüksek olmasıdır.

PRM algoritması olasılıksal bir algoritmadır ve olasılıksal algoritmaların içinde en çok kullanılanlardan birisidir. PRM algoritması hızlı rota oluşturabilen bir algoritmadır. Bundan dolayı A* algoritmasının PRM algoritması ile beraber çalışması durumunda çok daha hızlı sonuç verebileceği beklenmektedir. Sonraki bölümlerde A* ve PRM algoritması detaylı bir şekilde anlatılmış ve son bölümde bu iki algoritmanın ortak çalıştığı algoritma oluşturulmuştur.

4.1. A* Algoritması

A* algoritması, sezgisel bir yaklaşım algoritmasıdır. İlk olarak Stanford Araştırma Enstitüsündeki araştırmacılardan Peter Hart, Nils Nilsson ve Bertram Raphael tarafından 1968 yılında sunulmuştur. [60, 61]. Bu algoritma rota planlama uygulamalarında kullanılan en popüler yöntemlerden bir tanesidir. Edsger Dijkstra'nın 1959 yılında yayınladığı algoritmasının bir uzantısı olarak görülebilir [5]. A* algoritmasının Dijkstra algoritmasından farkı, A* algoritmasında maliyet hesaplanırken Dijkstra algoritmasının çalışma prensibine ek olarak sezgisel maliyette hesaba katılır. Böylece A* algoritmasında; tarama alanı her yöne yayılmadan doğruca hedefe doğru yönlendiği için daha hızlı ve daha verimli rota hesaplanabilir.

A* algoritmasında en kısa yol bulunurken toplam maliyet aşağıda verilen formül ile hesaplanır.

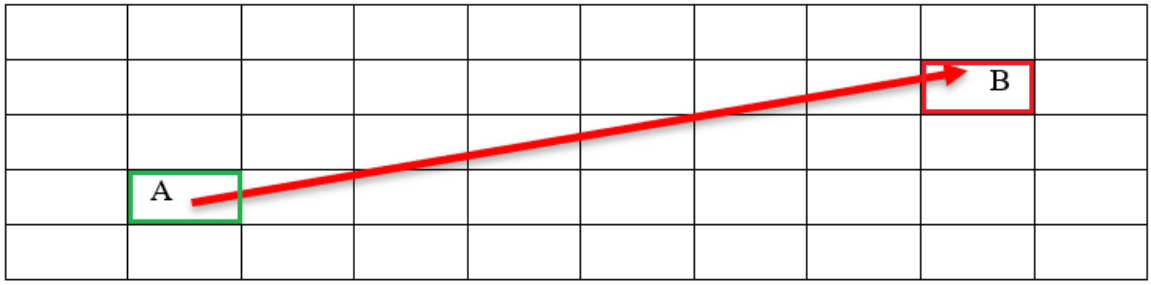
$$F(x) = G(x) + H(x) \quad (4.1)$$

F(x): Toplam maliyet

G(x): Başlangıç noktasından harita üzerinde istenen bir noktaya ulaşmak için gidilmesi gereken yolun maliyetidir. Bu zamana kadar bulunmuş olan yol kullanılarak hesaplanır.
H(x): Herhangi bir kareden hedef noktasına ulaşmak için gidilmesi gereken yolun maliyetinin tahmini ölçütüdür. Bu kavram sezgisel olarak adlandırılır ve bir tür akıllı tahminden başka bir şey değildir. Yolu bulana kadar gerçek mesafeyi bilemeyiz, çünkü yolumuza çeşitli engeller çıkabilir (duvarlar, su vb.).

H(x)'i hesaplamanın birçok yolu vardır. Bunlardan bazıları aşağıda açıklanmıştır.

Öklid mesafesi: Başlangıç ve hedef nokta arasındaki en kısa mesafedir. Kuş uçuşu mesafe olarakta adlandırılır. Şekil 4.1'de iki nokta arasındaki Öklid mesafesinin nasıl olduğu gösterilmektedir.



Şekil 4.1. 2 nokta arasındaki Öklid uzunluğu

Yukarıdaki şekilde A noktası ile B noktası arasındaki Öklid uzunluğunun nasıl alınacağı kırmızı ok ile gösterilmiştir.

Öklid uzunluğu;

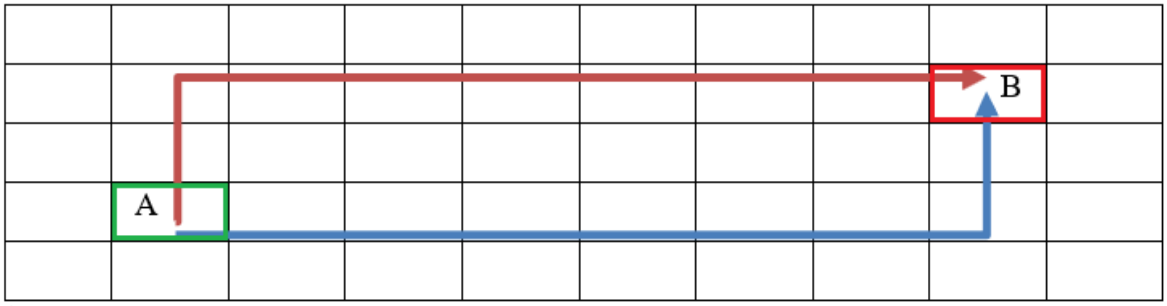
2 boyutlu haritada → $h(x) = \sqrt{(Bx - Ax)^2 + (By - Ay)^2}$

3 boyutlu haritada → $h(x) = \sqrt{(Bx - Ax)^2 + (By - Ay)^2 + (Bz - Az)^2}$

formülleri ile hesaplanır.

Manhattan mesafesi: Basitçe kare şeklindeki büyük apartman bloklarının bulunduğu bir şehirde taksi ile bir noktadan başka bir noktaya gitmek için alınması gereken mesafeyi gösterir. Yöntem ismini büyük apartman bloklarının bulunduğu Manhattan adasından

almaktadır. Bu yöntem ile başlangıç ve hedef noktası arasındaki mesafe hesaplanırken her boyutta ne kadar mesafe kat edildiği toplanarak bulunur. Şekil 4.2.'de 2 nokta arasındaki Manhattan mesafesinin nasıl olduğu gösterilmiştir.



Şekil 4.2. 2 nokta arasındaki Manhattan mesafesi

Yukarıdaki şekilde A noktası ile B noktası arasındaki Manhattan mesafesinin nasıl alınacağı gösterilmiştir.

Manhattan mesafesi;

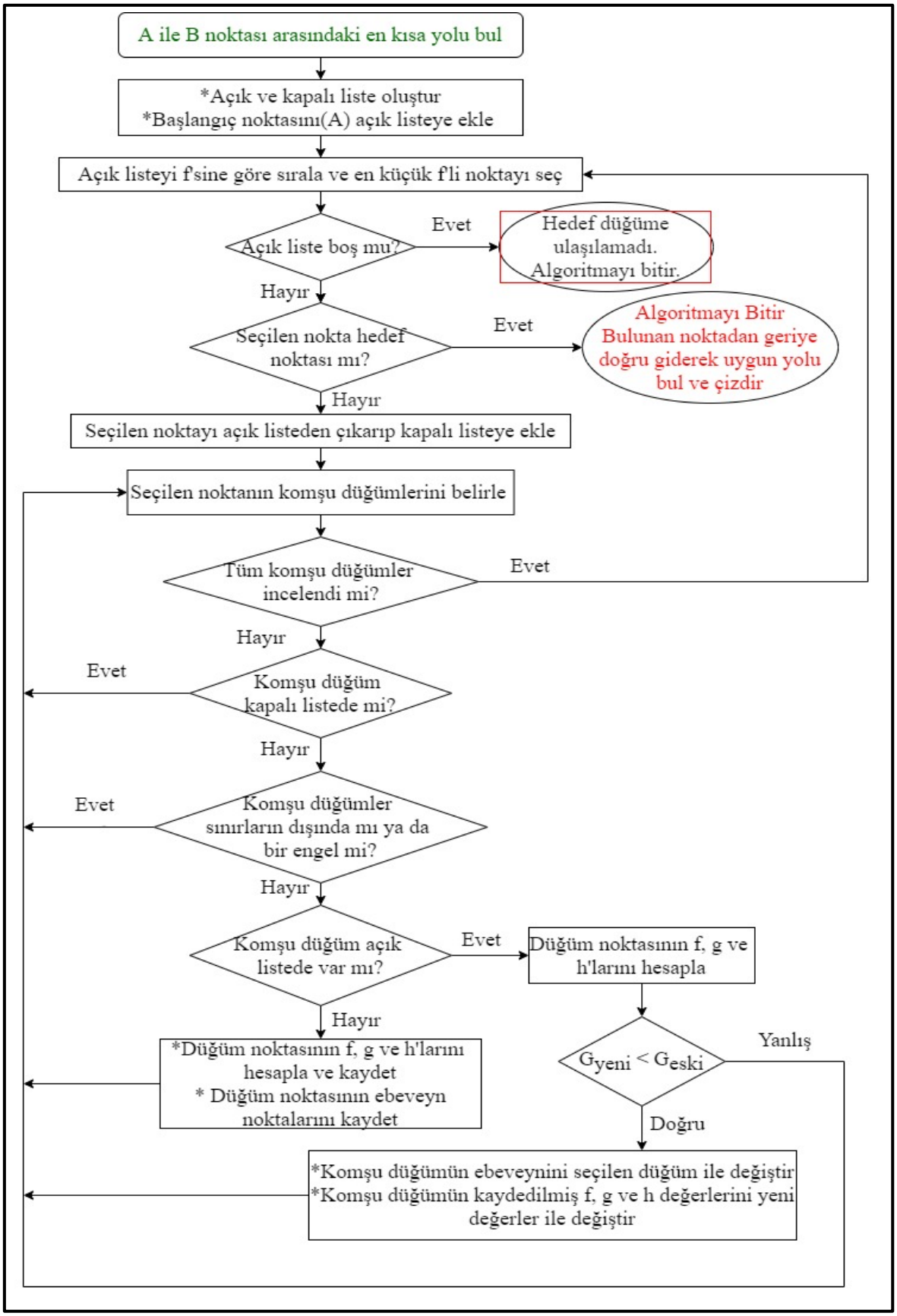
2 boyutlu haritada → $h(x) = |Bx - Ax| + |By - Ay|$

3 boyutlu haritada → $h(x) = |Bx - Ax| + |By - Ay| + |Bz - Az|$

formülleri ile hesaplanır.

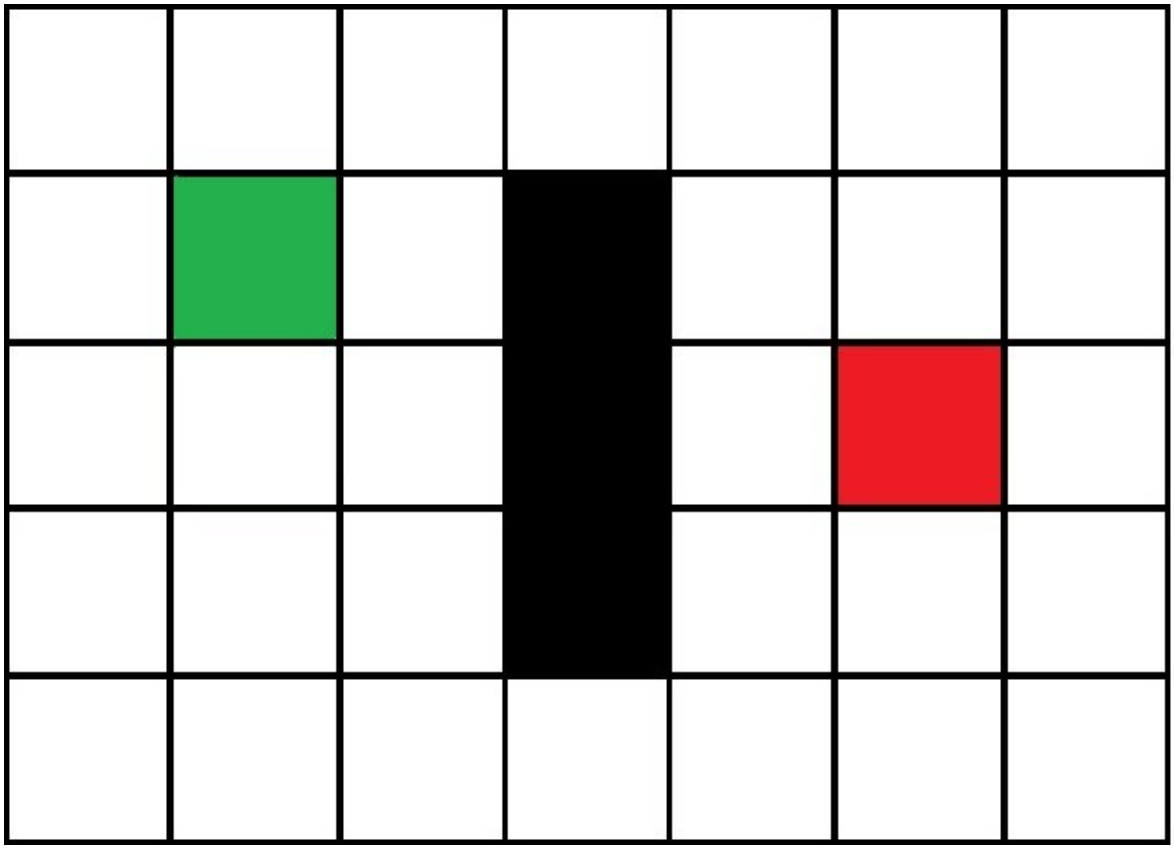
Yukarıda açıklandığı gibi daha başka sezgisel mesafe hesaplama yöntemleri mevcuttur. Sezgisel mesafe hesaplanırken önemli durumlardan bir tanesi sezgisel (heuristic) mesafe gerçeğe ne kadar yakın bulunursa A* algoritması o kadar doğru ve hızlı sonuç verir. Aşırı iyimser veya aşırı kötümser mesafe tahminleri algoritmanın verimli çalışması açısından kötü bir durumdur.

Aşağıdaki Şekil 4.3'te A* algoritmasının akış şeması verilmiştir. A* algoritması ile bir başlangıç noktasından bir hedef noktasına giden en kısa yolu bulurken aşağıdaki akış şeması kullanılır.



Şekil 4.3. A* algoritması için akış şeması

A* algoritması akış şeması aşağıda bir örnek ile açıklanmıştır. Akış şemasında geçen F, G ve H terimleri daha önce açıklanmıştır. Burada geçen açık liste ve kapalı liste terimleri örneğin ilerleyen aşamalarında açıklanacaktır. Aşağıda Şekil 4.4'te A* algoritması için örnek bir harita verilmiştir.



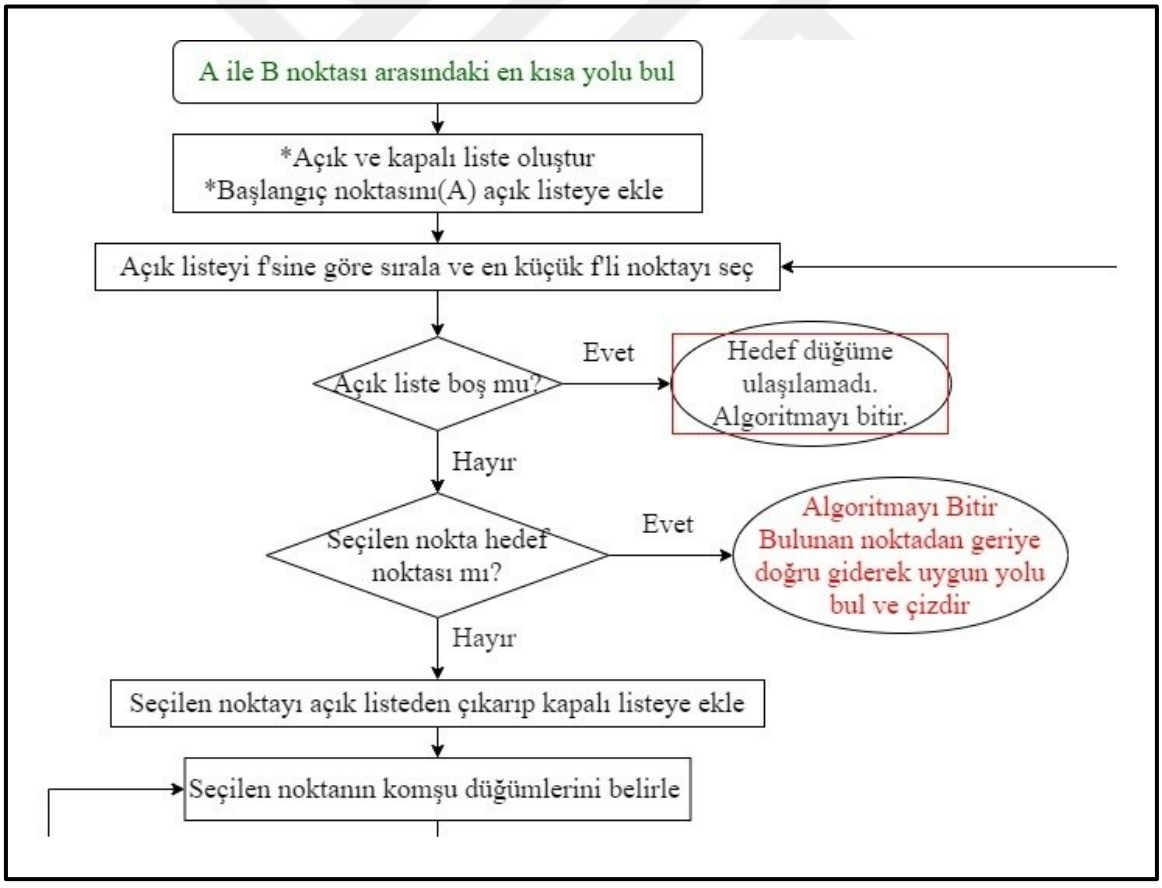
Şekil 4.4. A* örnek arama haritası

Harita da yeşil renk ile gösterilen kare başlangıç noktasını, kırmızı ile gösterilen kare hedef noktasını göstermektedir. Arada siyah ile gösterilen kareler ise engelleri (Duvar, su birikintisi gibi mobil robotun kullanamayacağı yolları) göstermektedir. Bir mobil robotun yeşil kareden kırmızı kareye engellere çarpmadan en kısa şekilde gitmek istediği varsayılırsa aşağıdaki gibi devam edilmelidir.

A* algoritması ile bu problemi çözmenin ilk adımı Şekil 4.4'teki haritayı kare ızgaralara bölerek arama alanını sadeleştirmektir. Böylelikle arama alanı iki boyutlu bir hale bürünecek ve haritayı incelemek daha kolay hale gelecektir. Haritadaki karelerin orta noktaları düğüm (node) olarak adlandırılmaktadır.

Aranılan rota başlangıçtan (yeşil kare) hedefe (kırmızı kare) ulaşılabilmesi için geçilmesi gereken karelerin elde edilmesiyle bulunur. Rota bulunduktan sonra mobil araç elde edilen karelerden sırasıyla hareket ederek hedefine ulaşabilecektir.

Harita kare ızgaralar şeklinde bölünüp sadeleştirme işlemi yapıldıktan sonra kareler üzerinden arama işlemine başlanılabilecektir. Arama işlemine başlangıç karesinden başlanılır. Komşu kareler taranarak hedefe ulaşana kadar hedefe doğru açılarak tarama işlemi yapılacaktır. Burada A* algoritmasının artısı görülebilmektedir. A* algoritmasının atası sayılan Dijkstra algoritmasında tarama dışa doğru her yöne olurken A* algoritmasında hesaplamaya sezgisel mesafe (H)'nin de katılması sayesinde tarama alanı doğruca hedefe doğru açılarak devam eder. Şekil 4.5'te A* algoritmasının 1. kısmı verilmiştir. Bu şekil yukarıda Şekil 4.3'te verilen A* algoritması akış şemasının başlangıç kısmıdır.



Şekil 4.5. A* algoritması akış şeması 1. parçası

Arama işlemi şu an A* algoritması akış şemasının Şekil 4.5'te gösterilen aşamasındadır.

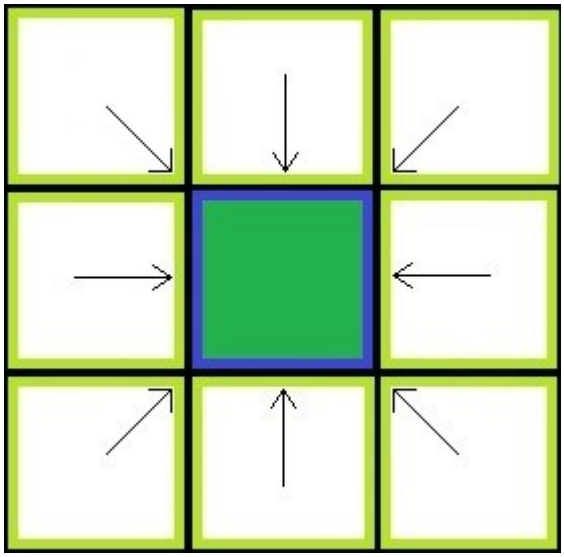
Bu kısımda sırasıyla yapılacak işlemler şu şekildedir;

- Başlangıç noktasından başla ve onu açık listeye koy.
- Başlangıç karesinin etrafında bulunan komşu kareleri incelenecekler listesine al.
- Başlangıç karesini açık listeden çıkarıp kapalı listeye al.

Burada akış şemasında geçen açık liste ve kapalı liste terimlerini açıklamak gerekmektedir.

Açık liste: Başlangıç karesi ile tarama işlemine başlandıktan sonra incelenen ve uygun olan karelerin tutulduğu listedir. Komşu noktalar incelendikten sonra karelerin maliyetleri Eşitlik 4.1'deki formüle göre hesaplanmaktadır. Açık listedeki kareler F değerlerine göre en küçükten en büyüğe doğru sıralanır. Sırasıyla en üstteki kareler (en düşük maliyetli kareler) seçilerek tarama işlemine devam edilir.
Kapalı liste: Açık listedeki karelerin incelendikten sonra konuldukları listedir. Kapalı listeye eklenecek karelerin ebeveyn bilgisi de eklenir. Algoritmanın devamında hedef nokta bulunduktan sonra hedef noktadan geriye doğru ebeveyn noktalar takip edilerek en kısa rota bulunmuş olur.

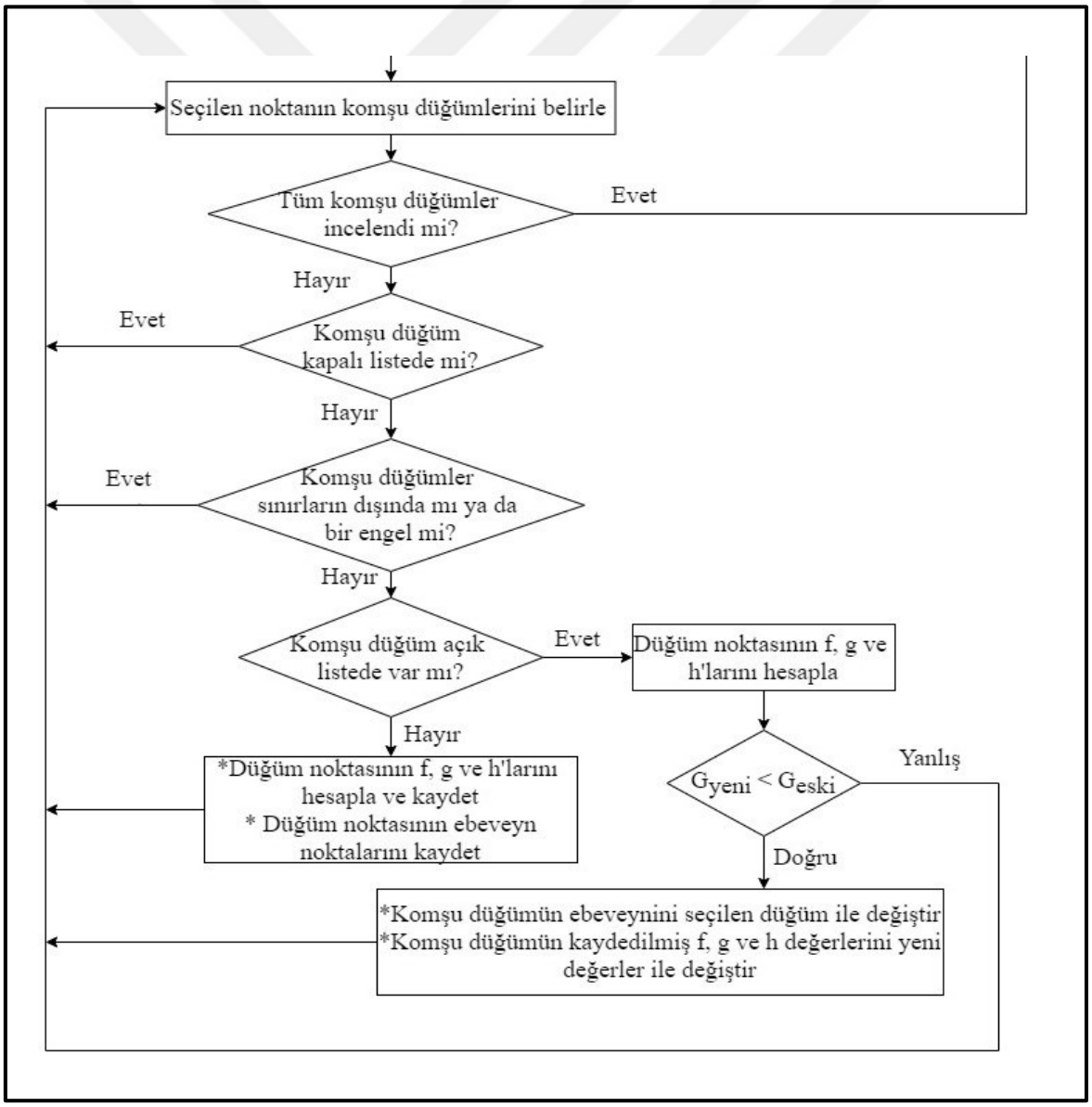
Aramaya başlangıç noktasından başlanıldıktan sonra burada Şekil 4.6'daki gibi bir yapı elde edilecektir.



Şekil 4.6. Aramada başlangıç

Şekil 4.6'da ortadaki yeşil renk başlangıç noktasını ifade etmektedir. Kapalı listeye alındığını belirtmek için de mavi çerçeve içine alınmıştır. Başlangıç noktasının çevresindeki 8 tane komşu kare herhangi bir engele takılmadığı, haritanın sınırları içinde olduğu ve kapalı listede olmadığı için açık listeye alınmıştır. Açık listede olduğunu ifade etmek için açık yeşil çerçeve içine alınmışlardır ve incelenmek için beklemektedirler. Her birinde ebeveyn karesini işaret etmek için başlangıç karesini gösteren birer ok vardır.

Başlangıç noktası incelendikten sonra sıra başlangıç noktasının komşularını incelemeye gelmiştir. Komşu noktaları incelerken akış şemasının 2. kısmı kullanılacaktır. Aşağıdaki Şekil 4.7'de A* algoritmasının akış şemasının 2. kısmı verilmiştir.



Şekil 4.7. A* algoritması akış şeması 2. parçası

Tarama sırasında seçilen karenin komşuları incelendikten sonra uygun olanlar açık listeye alınır. Komşu noktalar incelenirken Şekil 4.7'deki şemaya göre incelenir. İncelemeler şu şekilde ilerler;

- Komşu nokta haritanın sınırları dışında mı?
- Komşu nokta bir engel (duvar, çamur vb. gibi mobil robotun geçemeyeceği) noktasında mı?
- Komşu nokta daha önce kapalı listeye alındı mı?

Eğer incelenen komşu nokta bu durumlarda ise; bu nokta kullanılmadan sıradaki komşu noktalar incelenmeye devam edecektir. Eğer komşu nokta bu durumlarda değilse; bu nokta incelemeye alınacaktır. Burada komşu noktalar incelenirken komşu noktaların daha önce açık listeye alınıp alınmamasına göre iki farklı yol vardır.

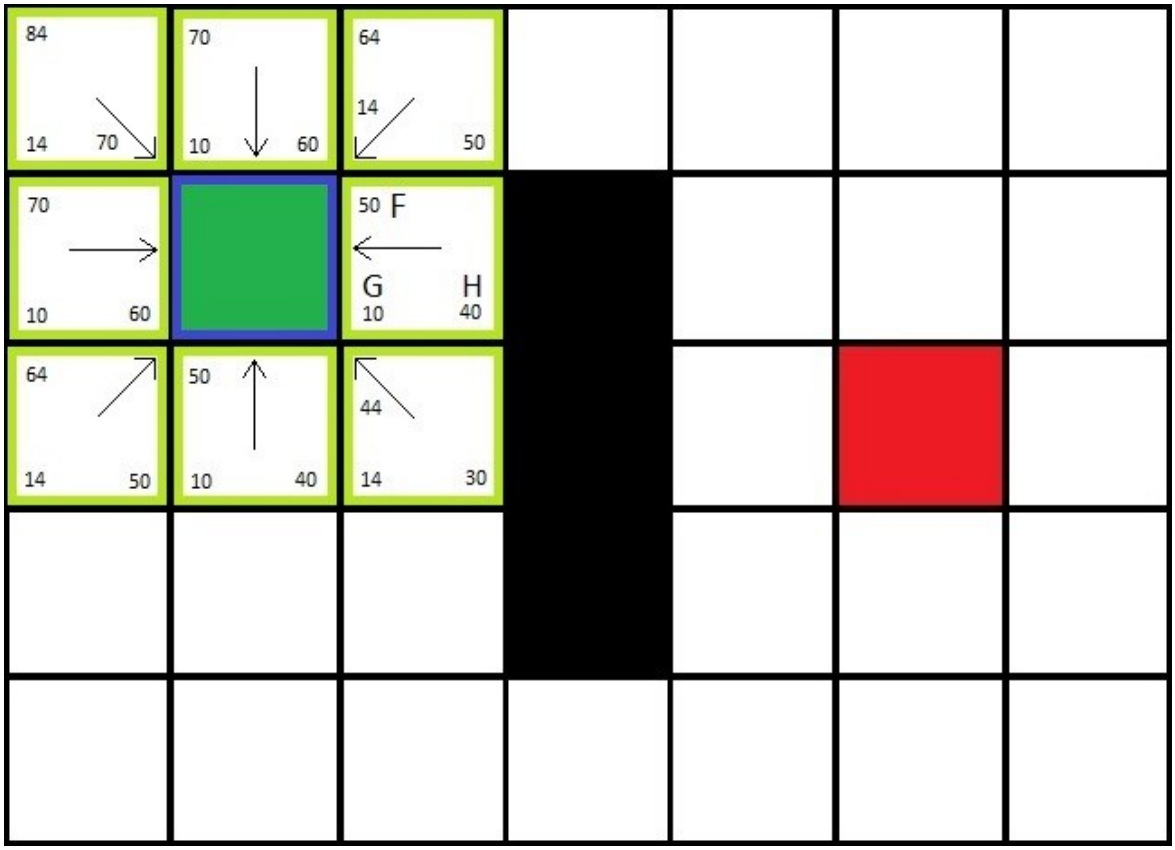
1. Eğer komşu nokta daha önce açık listeye eklenmemiş ise; komşu noktanın f, g ve h değerleri hesaplanır ve kaydedilir. Komşu noktanın ebeveyn noktasının koordinatları da kaydedilir ve sıradaki komşu noktadan işlemlere devam edilir.

2. Eğer komşu nokta daha önce açık listeye eklenmişse komşu noktanın f, g ve h değerleri hesaplanır. Eski eklenen noktanın g değeri ile yeni eklenen noktanın g değeri karşılaştırılır. Eğer eski değer daha küçük çıkarsa hiçbir şey yapılmadan sıradaki noktadan araştırılmaya devam edilir. Eğer yeni değer daha küçük çıkarsa eski değerler silinip yerine yeni noktanın f, g ve h değerleri yazılır ve eski noktanın ebeveyn değeri silinip yerine yeni noktanın ebeveyn değerleri yazılır.

Bu örnekte bir kareden başka bir kareye aynı eksende yapılan hareketler 10'ar puan, çapraz yapılan hareketler $10\sqrt{2}$ yani 14,14 puandır. Burada işlem kolaylığı açısından bu değer 14 olarak alınacaktır. 0,14 kadar küçük bir değer işlemlerde herhangi bir sıkıntı oluşturmayacağı değerlendirilmiştir.

Sezgisel maliyet hesaplanırken önemli olan aşırı iyimser ve aşırı kötümser tahminlerden uzak olup gerçeğe en yakın yolun hesaplanmasıdır. İki yöntemde verimli sonuçlar verir fakat işlem kolaylığı açısından bu örnekte Manhattan metodu kullanılacaktır.

Sezgisel maliyet hesaplanırken eğer maliyeti hesaplanacak kareler arasında engel varsa hesaplama engel yokmuş gibi yapılır. Aşağıdaki Şekil 4.8'de tarama olayının ilk adımının sonucu gösterilmiştir.



Şekil 4.8. Taramanın ilk adımının sonucu

Her karenin maliyetleri (F, G ve H) üzerlerine yazılmıştır. Başlangıç karesinin sağ tarafındaki kare de gösterildiği gibi F değeri karenin üst kısmında, G değeri karenin sol alt kısmında ve H değeri karenin sağ alt kısmında yazılmıştır.

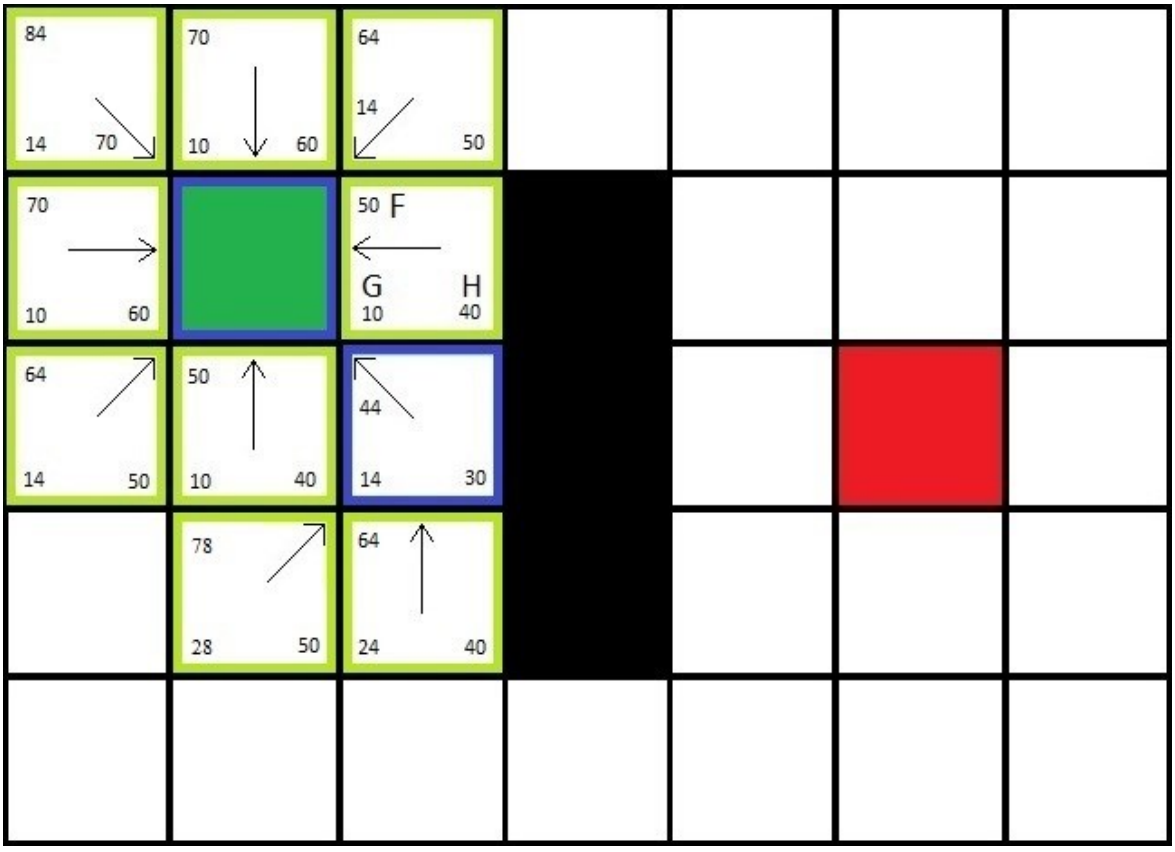
Başlangıç noktasının taraması bitmiştir. Arama işlemine açık listede f değeri en düşük olan kare ile devam edilir. F değeri en düşük olan kare 44 ile başlangıç noktasının sağ alt çaprazında bulunan karedir. Seçilen yeni kare ile sırasıyla aşağıdaki işlemler yapılacaktır.

1. Seçilen noktayı açık listeden çıkarıp kapalı listeye ekle.
2. Seçilen noktanın çevresindeki komşularını kontrol et. Engel noktasında, harita sınırları dışında ya da kapalı listede olanları yok say.

Uygun olan komşu noktaların;

- Açık listede olmayanlarının f, g ve h değerlerini hesapla ve ebeveyn noktasının bilgisi ile birlikte kaydet.
- Açık listeye daha önce eklenmiş olan komşu noktaların f, g ve h değerlerini hesapla. Eğer eski g değeri yeni g değerinden küçük ise hiçbir değişiklik yapmadan arama işlemine devam et. Eğer yeni g değeri eski g değerinden daha küçük ise yeni karenin f, g ve h değerleri ile ebeveyn noktasının bilgilerini eskileri ile değiştir.

Yukarıdaki işlemleri yeni seçilen noktaya uyguladığımızda aşağıdaki Şekil 4.9'daki gibi bir durum ortaya çıkacaktır.



Şekil 4.9. Taramanın ikinci adımının sonucu

Tarama sırasında yapılan işlemleri açıklamak gerekirse; ilk olarak seçili kare (f değeri 44 olan kare) açık listeden çıkarılıp kapalı listeye alınmıştır. Sonra komşuları incelenmeye başlanmıştır.

Sol üstteki kare başlangıç karesi olduğu ve kapalı listede olduğu için incelemeden elenmiştir. Seçili karenin sağ tarafındaki 3 kare engel noktası olduğu için bunlarda incelemeden elenmiştir. Hemen altındaki ve sol alt çaprazındaki 2 kare herhangi bir sınırlamayı ihlal

etmedikleri için f, g ve h değerleri hesaplanmıştır. Daha önce açık listeye alınmadığı için açık listeye alınmış ve maliyet değerleri harita üzerine yazılmıştır. Karelerin ebeveyn noktaları seçili kareyi gösterecek şekilde oklar vasıtasıyla gösterilmiştir.
Sırada seçili karenin hemen üstündeki kare vardır. Bu kare herhangi bir sınırlamayı ihlal etmediği için f, g ve h değerleri hesaplanmıştır. Fakat bu kare açık listededir. Yani daha önce bu kare incelenmiştir. Bu durumda yapılacak işlem şimdiki seçili kareden bu komşu kareye gitmenin daha önceki seçili kareden bu komşu kareye gitmenin maliyetinden daha az olup olmayacağını incelemektir.

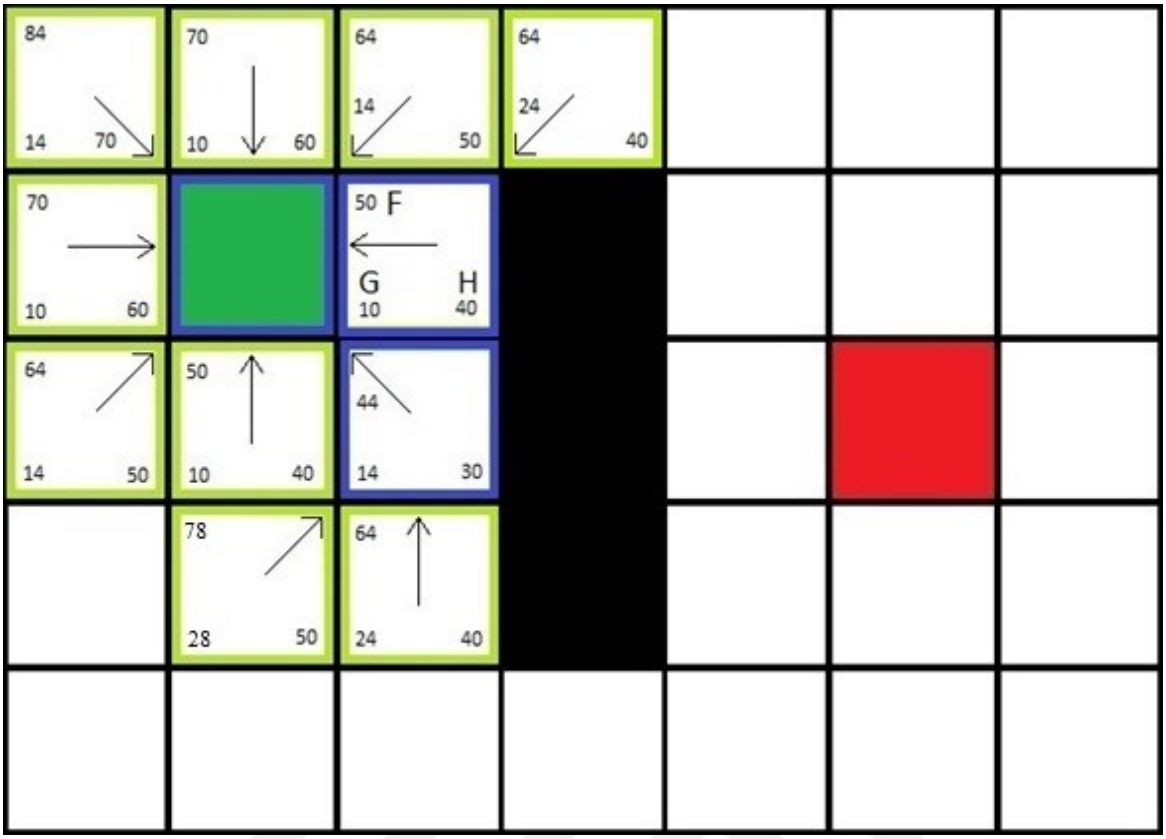
14 başlangıç karesinden seçili kareye gitmenin maliyeti, 10 ise seçili kareden komşu kareye gitmenin maliyetidir. Diğer bir değişle, şimdiki seçili karenin toplam g maliyeti 24'tür. Bu komşu karenin eski g maliyeti 10'dur. Yeni g değeri eski g değerinden daha küçük olmadığı için bu yol tercih edilmemiştir. Aynı durum seçili karenin solundaki kare için de geçerlidir.

İkinci taramanın sonucu Şekil 4.9'daki gibidir.

Sıradaki tarama için açık listeden f değeri en küçük olan kare seçilmelidir. Burada en küçük f değeri 50'dir fakat bu değere sahip 2 adet kare mevcuttur. Bu iki kareden istenilen kare seçilebilir. Eğer olumsuz olan bir kare seçilirse sadece tarama zamanını uzatır. Sonuç için bir değişiklik oluşturmaz. Örneğin devamında başlangıç karesinin sağındaki kare seçilerek tarama işlemine devam edilecektir.

Aşağıdaki Şekil 4.10'da taramanın 3. adımının sonucu gösterilmiştir. Bu işlemler sırasıyla aşağıda açıklandığı gibidir.

İlk olarak yeni seçilen kare açık listeden çıkarılıp kapalı listeye alınmıştır. Sonra sırasıyla komşu kareler incelenmeye başlanmıştır.

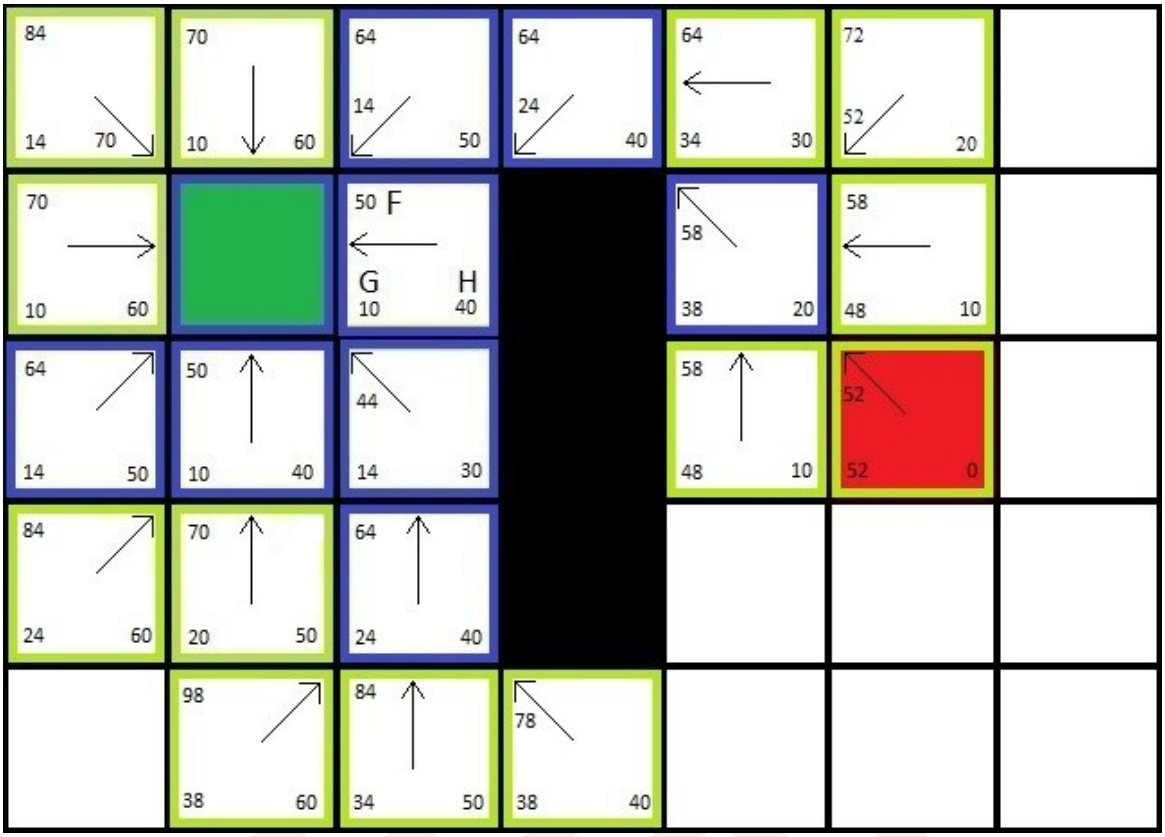


Şekil 4.10. Taramanın üçüncü adımının sonucu

Seçili karenin hemen solundaki ve altındaki kareler kapalı listede olduğu için incelenmeye alınmayacaktır. Yine, seçili karenin sağındaki ve sağ alt çaprazındaki kareler engel olduğu için bu noktalarda incelemeye alınmayacaktır. Seçili karenin üstündeki, sol alt ve sol üst çaprazındaki kareler herhangi bir sınırlamayı ihlal etmemektedirler. Ancak bu kareler açık listeye daha önce alınmışlardır. Bu komşu karelerin yeni maliyet değerleri eski maliyet değerlerinden daha küçük olmadığı için bu yollar tercih edilmemiştir. Geriye sadece seçili karenin sağ üst çaprazındaki kare kalmıştır. Bu kare herhangi bir sınırlamayı ihlal etmediği için incelemeye alınmıştır ve maliyet (f, g, h) değerleri hesaplanmıştır. Bu kare daha önce açık listeye alınmadığı için açık listeye alınmış ve maliyet değerleri harita üzerine yazılmıştır. Karenin ebeveyn noktaları seçili kareyi gösterecek şekilde oklar vasıtası ile gösterilmiştir.

Tarama işlemleri hedef kare açık listeye eklenene kadar devam etmektedir.

Aşağıdaki Şekil 4.11'de taramanın devamı ve hedef noktasının açık listeye eklendiği durumu gösterilmiştir. Bu durumda algoritma başarıya ulaşmış demektir ve algoritma sonlandırılır.

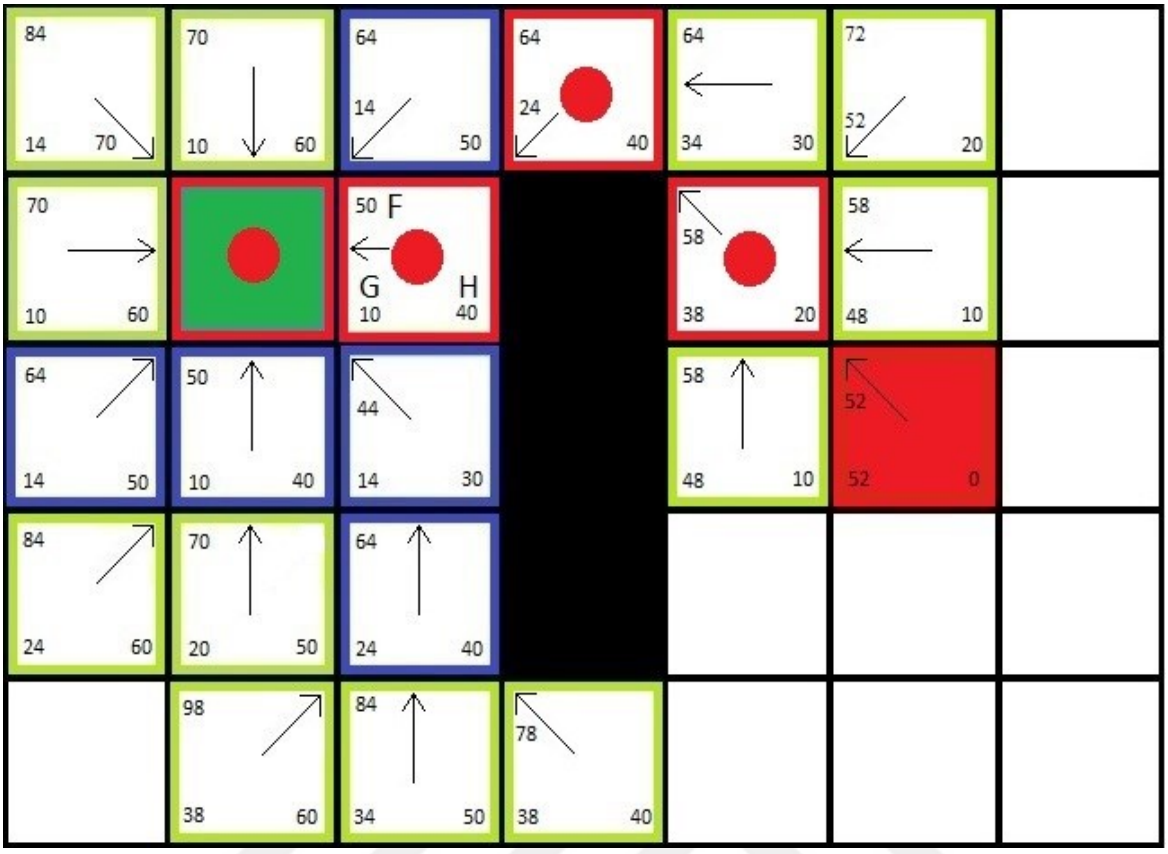


Şekil 4.11. Hedef noktanın açık listeye eklenmesi

Eğer algoritma hedef noktasını bulamasaydı ve açık liste boş kalsaydı tarama işlemi bitirilecek ve algoritma başarısız sayılacaktı.

Bu noktadan sonra ulaşılan hedef noktasından başlangıç noktasına kadar geriye doğru ebeveyn noktalarını işaret eden oklar takip edilir. Bulunan kareler kaydedilir. Bu kareler algoritmanın bulduğu rotayı gösterirler.

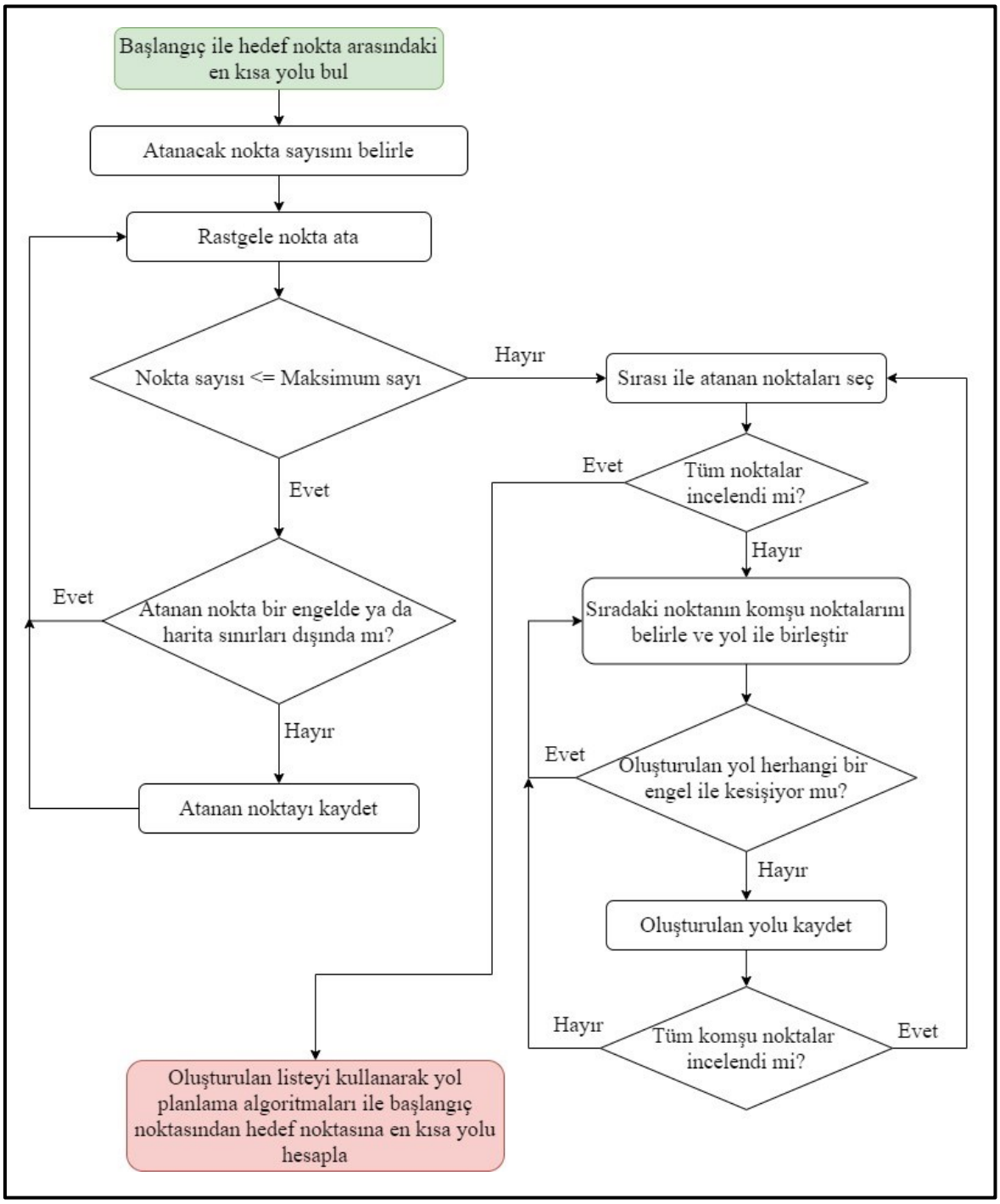
Rota aşağıdaki Şekil 4.12'de net bir şekilde görülebilmektedir. Başlangıç noktasından hedef noktaya kırmızı noktaları takip ederek gidilebilmektedir.



Şekil 4.12. Tarama sonunda yolun bulunması

4.2. Olasılıksal Yol Haritaları Algoritması (PRM Algoritması)

Olasılıksal Yol Haritaları Algoritması (PRM) olasılıksal tabanlı en kısa yol planlama algoritmalarından birisidir. Lydia E. Kavraki tarafından geliştirilmiştir [65]. Olasılıksal yöntemlerin içinde en çok kullanılan yöntemlerdendir. Diğer yol planlama algoritmalarına göre daha hızlı çalışır ve daha kısa sürelerde sonuç verir. Rastgele atanan noktalar haritanın her tarafına dağıldığı için algoritma arama alanının tümüne hâkim olabilir. Ancak PRM algoritmasında üretilen rotalar rastgele atanan yollar üzerinden oluşturulduğu için üretilen rotalar yumuşaklık ve güvenlik açısından kalite sorunlarına sahip olabilir. Daha kaliteli ve uygun bir yol oluşturulmak istenirse; PRM algoritmasından sonra oluşturulan yol cubic splines, polynomials ya da bezier eğrisi gibi yumuşatma teknikleri kullanılarak yolun kalitesi artırılabilir [3]. Cubic splines, polynomials ya da bezier eğrisi gibi yumuşatma teknikleri keskin dönüşlere sahip rotalarda yumuşatma işlemleri uygulayarak yolun kalitesini artıran tekniklerdir. Aşağıdaki Şekil 4.13'te PRM algoritmasının akış şeması verilmiştir.

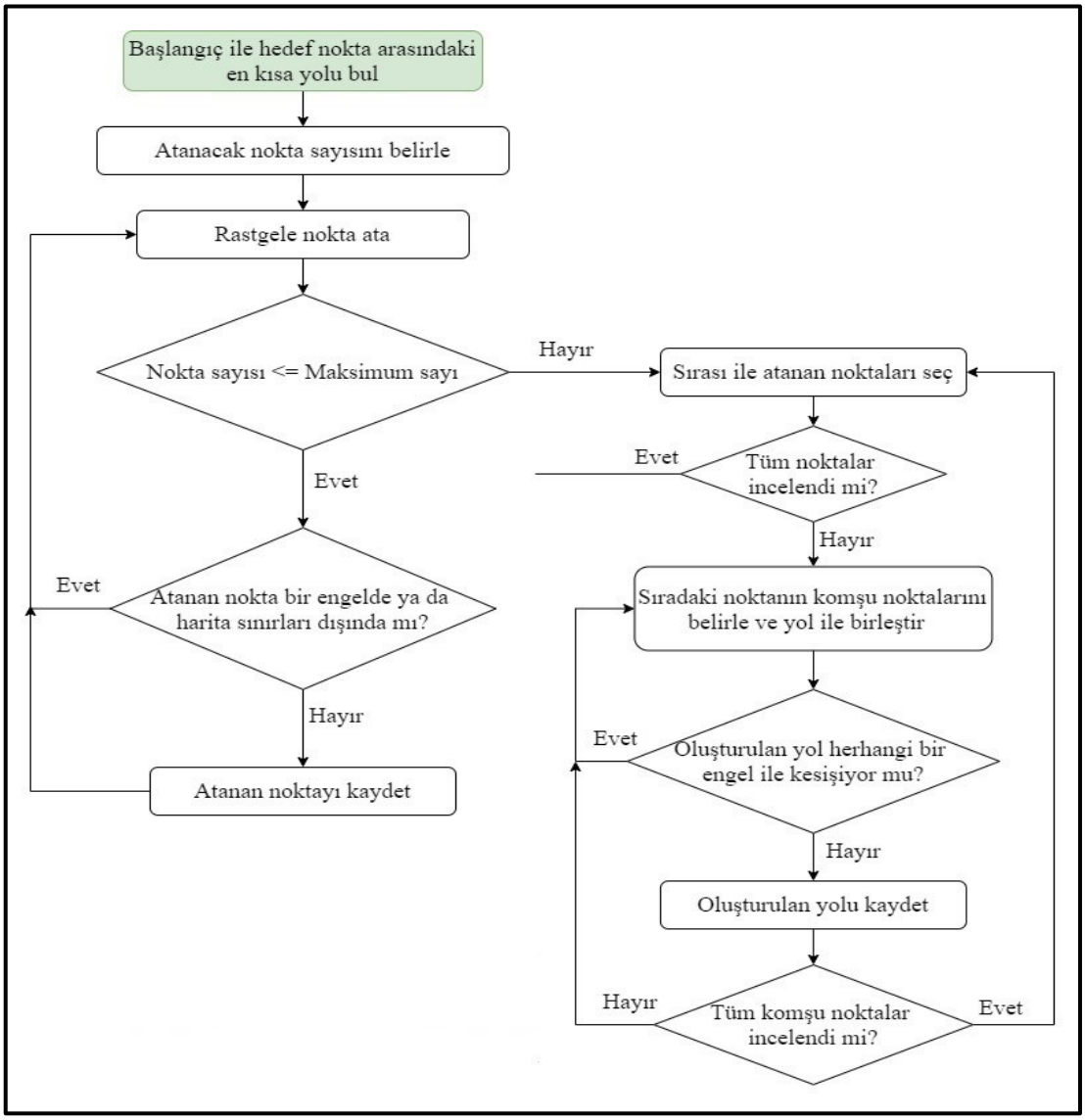


Şekil 4.13. PRM algoritması akış şeması

PRM algoritması başlangıç ve hedef arasındaki en kısa mesafeyi bulurken 2 aşama da ilerler: Öğrenme ve Sorgu aşaması.

Öğrenme aşaması

Şekil 4.14'te PRM algoritmasının öğrenme aşaması verilmiştir.



Şekil 4.14. PRM algoritması öğrenme aşaması

Öğrenme aşaması PRM algoritmasının yol haritası oluşturduğu bölümdür. Yol haritası; verilen harita üzerinde nokta atamasının yapıldığı ve bu noktaların komşuluk ilişkilerine göre birleştirilerek oluşturulan, basit tanımı ile kompleks haritaların daha basitleştirilmiş halleridir. Bu yapı kompleks haritaların taranma süresini oldukça kısaltmaktadır.

Öğrenme aşaması iki kısımdan oluşmaktadır. Birincisi rastgele nokta atama bölümü, ikincisi ise atanan noktaların komşuluk ilişkilerine göre birleştirildiği bölümdür.

Birinci bölüm algoritmanın harita üzerinde rastgele nokta atadığı bölümdür. Rastgele atanan noktalar haritanın her tarafına ulaşmayı mümkün kılacak şekilde atanmalıdır. Algoritmanın devamında arama işlemi atanan noktalar üzerinden yapılır. Eğer atanan noktalar arasında bağlantısallık (atanan noktaların başlangıç ve hedef noktası dahil haritanın her tarafına ulaşabilmesi ve noktaların birbiri ile bağlantısının mümkün olması) sorunu olursa rotanın oluşturulması zor olabilir hatta mümkün olmayabilir.

Bu bölümde algoritmanın başlangıcından itibaren belirlenen sayıda nokta harita sınırları içinde engel olmayan bölgelere atanır. Birinci bölüm atanan nokta sayısının maximum nokta sayısına ulaştığı durumda sonlanır ve ikinci bölüme geçilir. Birinci bölümün çalışma şekli şu şekildedir:

- Rastgele atanacak nokta sayısını belirle.
- Sırası ile rastgele nokta atamaya başla.
- Her atanan noktayı harita sınırları içinde ya da engelde olup olmaması durumuna göre sorgula. Eğer atanan nokta harita sınırları dışında ya da engelde ise atanan noktayı silip yeniden nokta ata.
- Harita sınırları içinde olan ve engel ile kesişmeyen bir nokta atandığında bu noktanın koordinatlarını kaydet.
- Atanan noktaların sayısının maksimum nokta sayısına ulaşıp ulaşmadığını kontrol et. Eğer ulaşmadı ise bu döngüyü maksimum nokta sayısına ulaşana kadar devam ettir. Atanan nokta sayısı maksimum değere ulaştığında birinci bölüm bitmiştir. Burada, atanan nokta sayısına harita dışına ya da engel üzerine atanan noktalar dahil edilmez.

Atanacak nokta sayısı belirlenirken, atanacak noktaların haritanın tamamına bağlantısallık (connectivity) sağlayacağı şekilde ve optimum sayıda olmasına dikkat edilmelidir. Eğer belirlenen sayı bağlantısallık sağlayamayacak kadar düşük olursa algoritma haritanın tamamına ulaşamaz veya hiç rota oluşturamaz ya da optimum rotaları kaçırarak kalitesiz ve yüksek maliyetli rota oluşturabilir. Eğer belirlenen sayı gereğinden fazla olursa algoritma daha kaliteli rota oluşturabilir ancak algoritma daha fazla noktayı inceleyeceği için rota oluşturma süresi gereksiz yüksek olur.

İkinci bölümde atanan noktalar birbirleri ile yollar vasıtasıyla birleştirilir. Birleştirmenin amacı hangi noktaların birbirlerinin komşusu olduğunun bilinmesi ve hangi noktalardan bir diğerine geçişin mümkün olduğunun keşfedilmesidir. İkinci bölümün çalışma şekli şu biçimindedir:

- Atanan noktaların listesinden sırası ile noktaları seç.
- Seçilen noktanın komşu olduğu noktaları seç. Burada komşu noktaları seçerken bazı durumlarda uzaklık sınırı konabilir. Seçilen noktadan belli bir mesafenin üzerindeki uzaklıkta noktalar komşu olsa dahi hesaba katılmaz. Bu sınırlama algoritmanın aşırı dallanmasını önler ve bazı durumlarda yararlıdır. Ancak bazı durumlarda algoritmanın tarama adımlarını artırır ve işlem süresini yükseltir. Bu seçim rota oluşturulacak haritanın durumuna göre belirlenmelidir.
- Seçilen nokta ile komşu nokta arasını yol ile birleştir. Burada oluşturulacak yol farklı şekilde olabilir. Ancak çoğunlukla en kısa mesafe yani Öklid uzunluğu şeklinde oluşturulur.
- Oluşturulan yol herhangi bir engel ile kesişiyor mu kontrol et. Eğer kesişiyor ise birleştirmeyi iptal et.
- Bütün potansiyel komşular incelendikten sonra sıradaki noktaya da aynı işlemleri uygula ve tüm noktalar bitene kadar devam et.

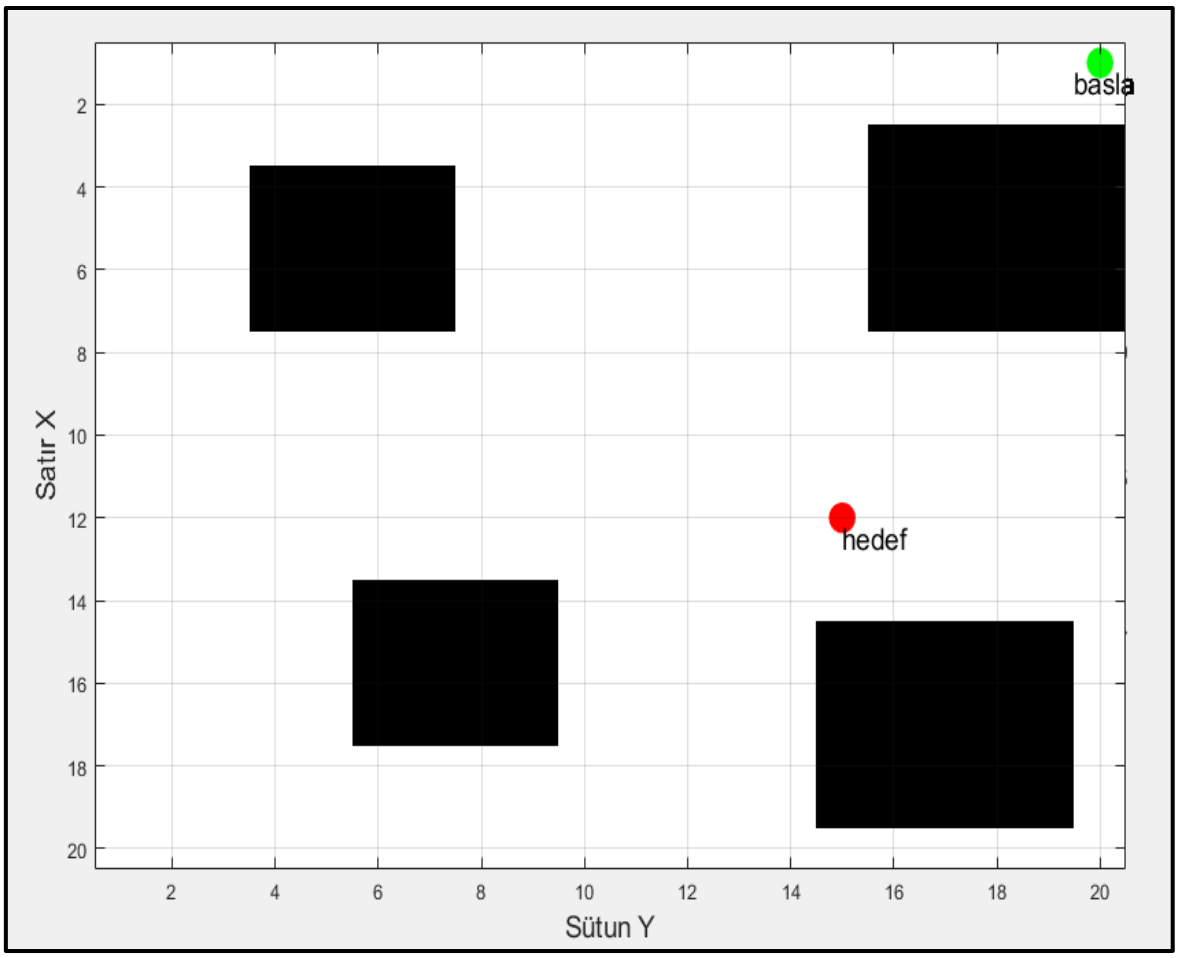
Atanan tüm noktalar komşuluk ilişkilerine göre birleştirildikten sonra yol haritası oluşturulmuş ve öğrenme aşaması bitmiştir.

Sorgu aşaması

PRM algoritması ile en kısa yol oluşturulurken, verilen bir harita üzerinde yol haritası oluşturulduktan sonra sıradaki aşama sorgu aşamasıdır. Sorgu aşamasında oluşturulan yol haritası Dijkstra, A*, D* veya çeşitli sezgisel optimizasyon algoritmaları ile taranarak, atanan noktalar ve oluşturulan yollar üzerinden en kısa rota oluşturulur.

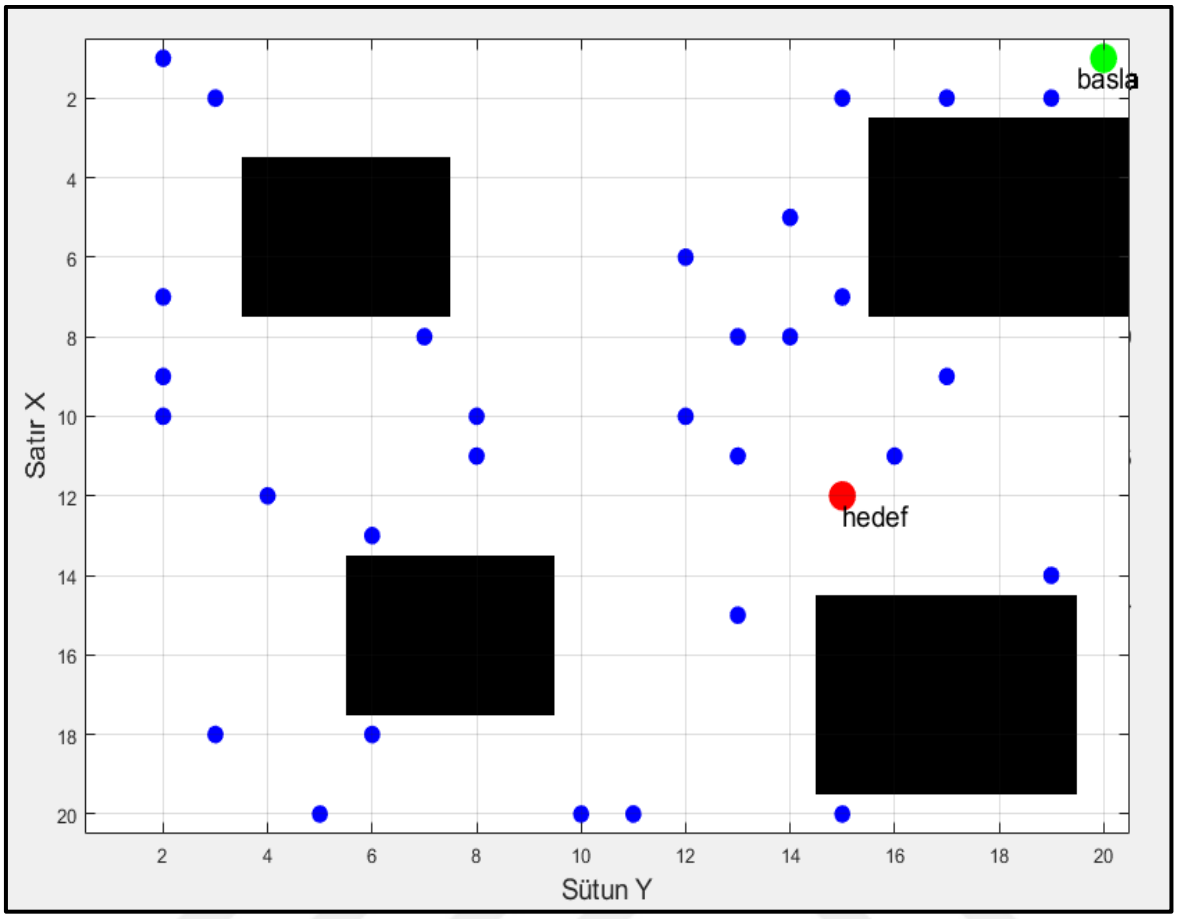
PRM algoritması bir örnek üzerinden açıklanması gerekirse;

Şekil 4.15'te verilen örnek bir harita üzerinde başlangıç ile hedef nokta arasındaki en kısa yol PRM algoritması ile bulunmak istenmektedir. Haritaya öncelikle PRM algoritmasının öğrenme aşaması uygulanmaktadır. Haritada engel yoğunluğu fazla olmadığı için atanacak nokta sayısı 30 olarak belirlenmiştir. Daha fazla nokta kullanılarak da uygun yol bulunabilir fakat fazla atanan nokta algoritmanın sonuca ulaşma süresini geciktirir.



Şekil 4.15. PRM arama alanı

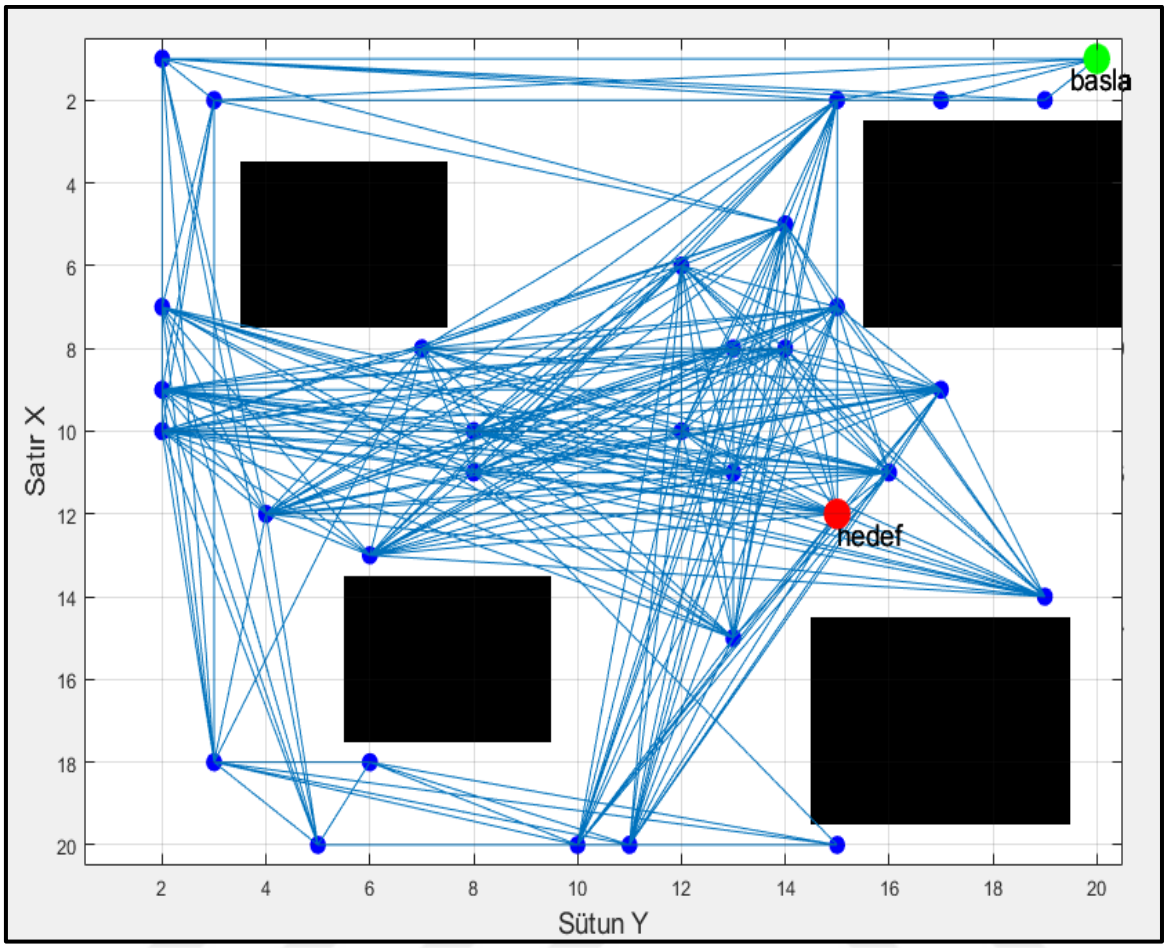
Aşağıdaki Şekil 4.16'da PRM algoritmasının öğrenme aşamasının birinci bölümünün sonucu verilmiştir.



Şekil 4.16. PRM algoritması öğrenme aşamasının birinci bölümünün sonucu

Görüldüğü gibi 30 adet nokta harita sınırları dışında olmayacak ve engeller ile çakışmayacak şekilde harita üzerine rastgele atanmıştır. Atanan noktalar harita üzerine dağıldığı için, algoritmaya yeterince bağlantısallık sağlamaktadır. Sırada öğrenme aşamasının ikinci bölümü vardır.

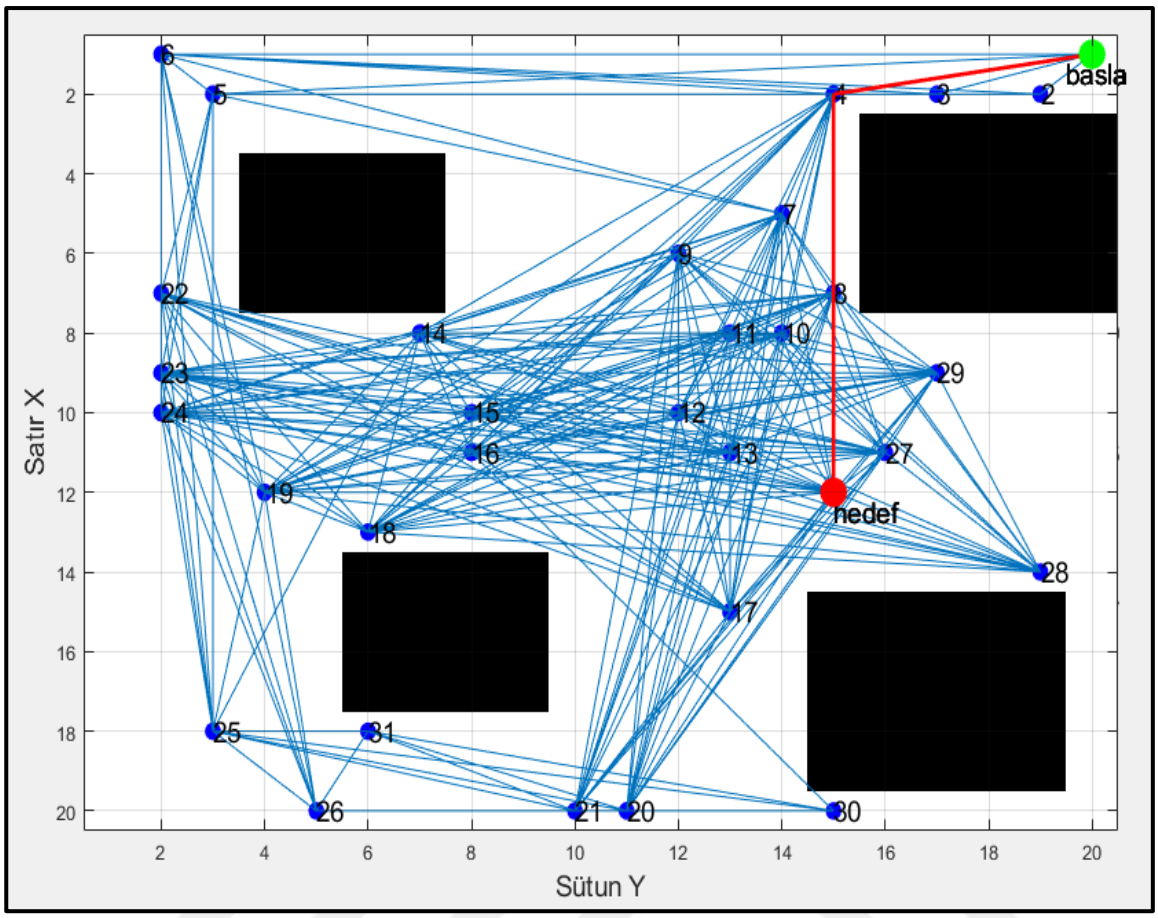
Aşağıdaki Şekil 4.17'de görüldüğü üzere öğrenme aşamasının ikinci bölümünün sonucu verilmiştir. Bu bölümde rastgele atanan noktalar arasında olası bütün yollar çizdirilmiştir. Bu aşamanın amacı mobil araç ile bir noktadan hangi noktaya ilerlemenin mümkün olduğunun belirlenmesidir. Bu örnekte komşu seçimi esnasında herhangi bir uzaklık sınırı belirlenmemiştir. Olası bütün komşular uzaklığı dikkate alınmadan değerlendirilmiştir.



Şekil 4.17. PRM algoritması öğrenme aşamasının ikinci bölümünün sonucu

Öğrenme aşaması tamamlandıktan sonra yol planlama algoritmalarının tarama alanı hazırdır. Sıradaki adım sorgu aşamasıdır. Bu aşamada oluşturulan yol haritası üzerinde Dijkstra, A*, D* veya çeşitli sezgisel optimizasyon algoritmaları kullanılarak Şekil 4.17'de oluşturulan harita üzerinden tarama işlemi yapılarak optimum yol bulunur.

Şekil 4.18'de sorgu aşamasının sonucunda algoritmanın oluşturduğu en kısa rota gösterilmiştir. 1-4-32 noktaları rota olarak belirlenmiştir. Belki aynı uzunluğa yakın daha farklı noktalarda seçilebilirdi; ancak algoritma en az dallanma ile hedefe ulaşmayı tercih etmiştir. Bunun bir avantajı da süreden tasarruftur. Algoritma her dallanmasında yeni noktanın bütün komşularını incelediği için her dallanma süre artışı demektir.

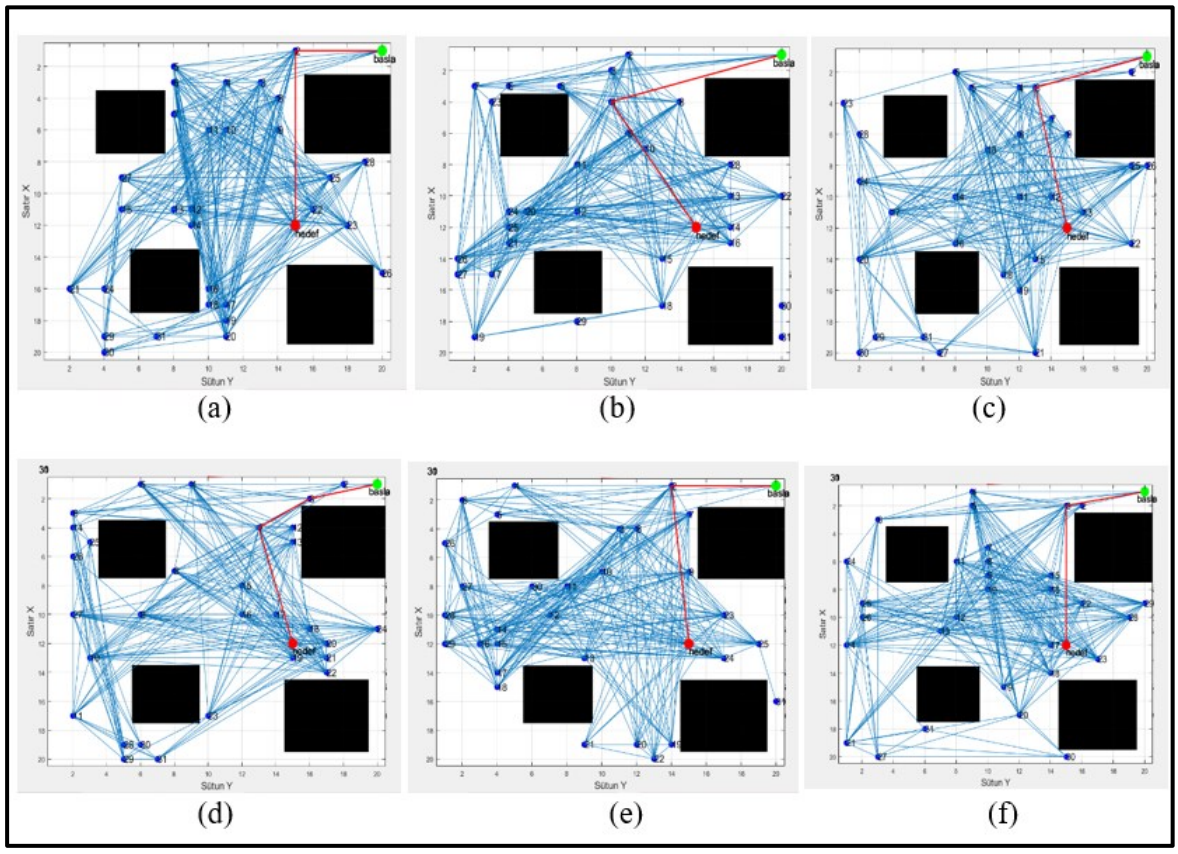


Şekil 4.18. PRM algoritması sorgu aşaması sonucu ve rota bulunması

PRM algoritmasının birçok avantajı olduğu gibi dezavantajları da vardır. Şekil 4.18'den de görüldüğü üzere oluşturulan rota keskin manevralara sahiptir. Ancak bu sıkıntıyı çözmek için çeşitli yöntemler mevcuttur.

- Atanan nokta sayısı artırılabilir ve sorgu aşamasında uzaklık sınırı konabilir. Böylelikle algoritmanın çözünürlüğü artar ve daha hafif manevralara sahip rotalar oluşturulabilir. Burada dikkat edilmesi gereken konu atanan nokta sayısının çok olmamasıdır. Eğer gereğinden fazla nokta atanır ise PRM algoritması en büyük avantajlarından birisi olan hızlı sonuç verme özelliğini kaybeder.
- Oluşturulan rota üzerine yumuşatma işlemleri uygulamak (Postprocessing). Cubic splines, polynomials ya da bezier eğrisi gibi teknikler çeşitli algoritmaların oluşturduğu rotaları yumuşatmak için kullanılan tekniklerdendir. PRM algoritması ile oluşturulan yollar böyle teknikler kullanılarak daha yumuşak ve kaliteli hale getirilebilir [3].

PRM algoritmasının bir diğer özelliği ise olasılıksal tabanlı olduğu için her çalışmasında farklı yollar üretmesidir (Şekil 4.19). Bu bazı durumlarda önemsizken bazı durumlarda sorun oluşturabilir. Ancak bu durumun çözümü için çeşitli teknikler mevcuttur. Örneğin harita üzerindeki engellerin köşe noktalarına sabit nokta atamak bu durumun bir çözümüdür [4].

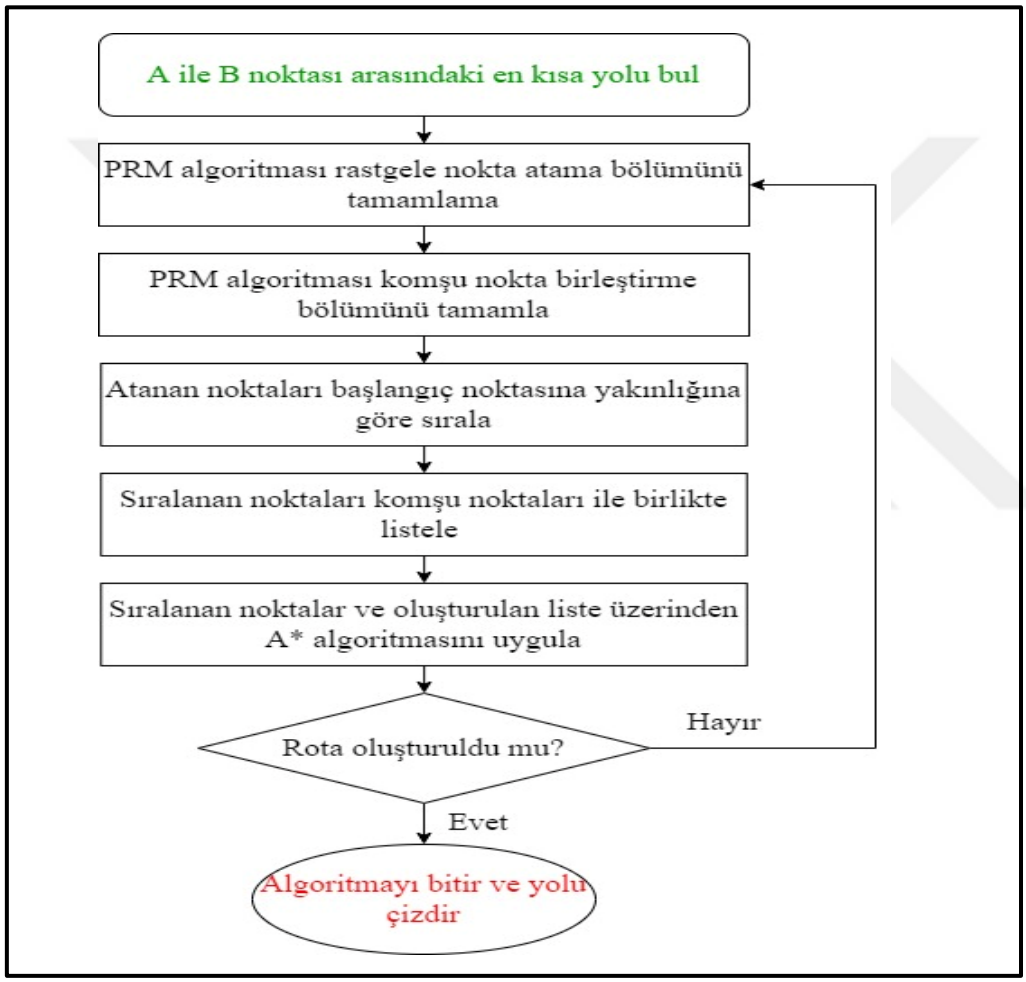


Şekil 4.19. PRM algoritması çeşitli çözüm örnekleri

Araştırmacılar PRM algoritmasının daha kaliteli, daha hızlı sonuç vermesi ve dezavantajlarının çözülmesi için sürekli geliştirme çalışmalarına devam etmektedirler. Lazy PRM, Obstacle Based PRM (Engel tabanlı PRM) vb. gibi yeni algoritmalar bunlardan bazılarıdır [3]. Bu gibi algoritmalar PRM algoritmasının dar geçiş noktalarında sıkıntı yaşaması, bazı durumlarda sorgu aşamasının gereksiz uzun sürmesi gibi durumlarına çözüm bulmaktadır.

4.3. PRM Algoritmasının A* Algoritması ile Çözümü (PRM-A*)

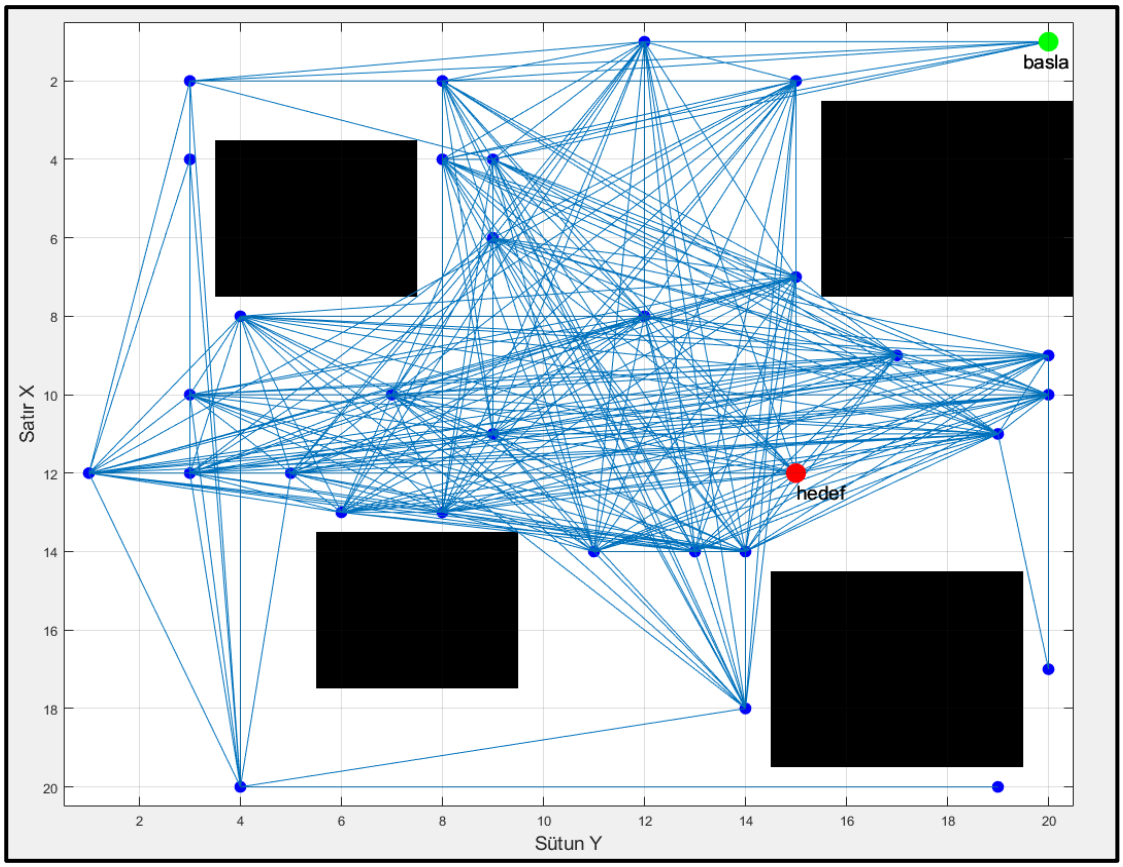
4.1. bölümde A* algoritması, 4.2. bölümde ise PRM algoritması anlatılmıştır. Bu bölümde PRM algoritması ile A* algoritmasının hibrit çalışma modeli analiz edilecektir. Şekil 4.20'de PRM-A* algoritmasının akış şeması verilmiştir. Bir A noktası ile B noktası arasındaki en kısa yol PRM-A* algoritması ile bulunmak istendiğinde aşağıdaki akış şeması kullanılmalıdır.



Şekil 4.20. PRM-A* algoritması akış şeması

Akış şemasının 2. ve 3. basamağı PRM algoritmasının akışıdır. PRM algoritması ile bir harita üzerinde bir başlangıç noktasından bir hedef noktasına en kısa yol hesaplanırken öncelikle rastgele nokta ataması yapılır. Sonrasında atanan noktalar komşuluk ilişkilerine göre birleştirilerek yol haritası oluşturulur. Oluşturulan bu harita durak noktaları olan ve birbirlerine hareket edilebilecek yolları belirtilen bir harita haline dönüştürülmüştür. Böylece

en kısa yol hesaplanırken tüm harita incelenmesi yerine sadece bazı noktalar incelenerek rota hesaplanabilir ve hem süreden hem de işlem gücünden tasarruf edilmiş olunur. Şekil 4.21'de PRM algoritmasının oluşturduğu örnek bir yol haritası verilmiştir.

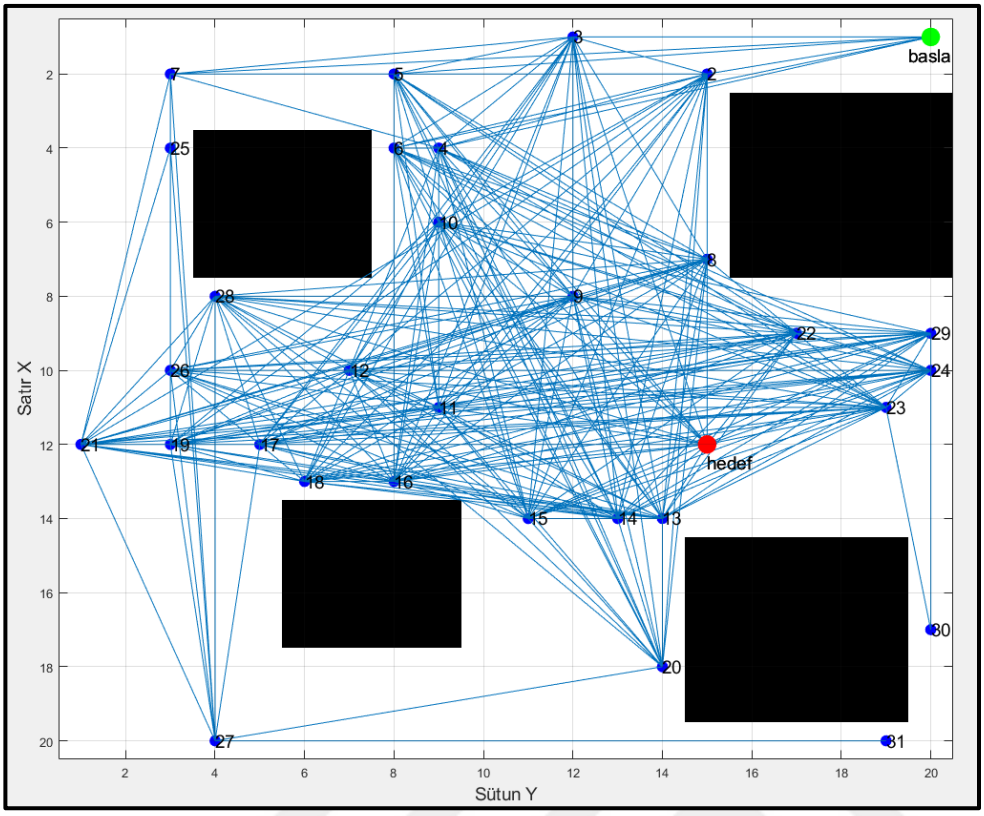


Şekil 4.21. PRM algoritmasının oluşturduğu yol haritası örneği

Akış şemasının 4. ve 5. basamaklarının amacı PRM algoritmasının oluşturduğu haritayı A* algoritmasının tarama yapabileceği hale getirmektir. Bu basamaklardan sonra hazırlanan harita üzerinde A* algoritması ile tarama yapılmaktadır.

Oluşturulan harita üzerine A* algoritması uygulanırken şu işlemler yapılmaktadır;

1. Öncelikli olarak atanan noktalar başlangıç noktasına yakınlığına göre 2'den başlayarak sıralanmalıdır. 1. nokta başlangıç noktası, sonuncu nokta hedef noktası olarak belirlenmelidir. Şekil 4.22'de bunun bir örneği gösterilmiştir. Sonrasında sıralanan noktalar komşu noktaları belirtilecek şekilde listelenmelidir.



Şekil 4.22. Atanan noktaların sıralanması

2. Bu aşamadan sonrası 4.1. bölümde anlatılan A* algoritmasının aynısıdır. Ancak, tarama sırasında yeni komşu seçimi ve maliyet hesabı işlemleri farklılık gösterir. Noktalar Şekil 4.22'deki gibi sıralandıktan sonra başlangıç noktasının komşu noktaları ve maliyetleri incelenmeye başlanır.

Aşağıdaki Çizelge 4.1'de atanan noktaların komşuluk bilgisi ve nokta konumlarının tutulduğu liste gösterilmiştir. 2 grafiğinde solundaki sütun nokta numarasını göstermektedir. Sağ tarafındaki bilgiler ise birinci grafik sırasıyla 1. noktadan hangi noktaya gidilebileceğini, ikinci grafik ise atanan noktaların harita üzerindeki konumlarını göstermektedir.

Tarama işlemine başlanırken başlangıç noktasının bütün komşu noktalarının maliyetleri hesaplanır. En düşük maliyetli nokta seçilir.

1	2	3	4	5	6	7	0	0
2	3	8	4	9	5	10	6	32
3	2	5	4	6	10	8	9	7
4	6	10	5	3	9	2	7	12
5	6	4	3	10	7	2	9	8
6	4	5	10	3	9	11	2	8
7	25	5	4	26	3	19	21	2
8	22	9	2	32	23	10	3	4
9	8	10	11	4	32	22	12	6
10	4	6	9	5	12	11	3	8
11	12	16	15	18	17	9	14	10

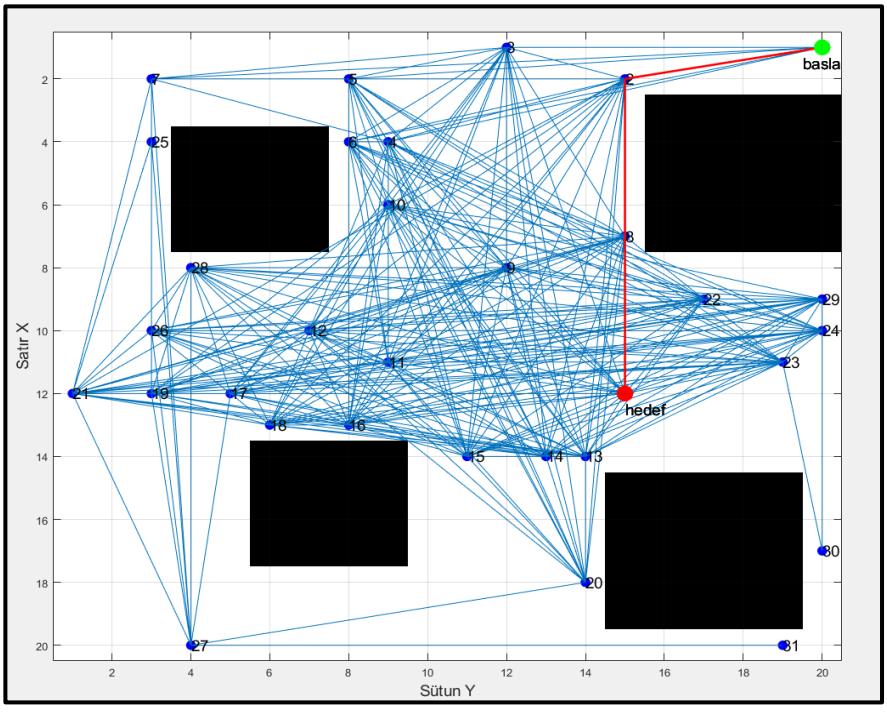
(a)

1	1	20
2	2	15
3	1	12
4	4	9
5	2	8
6	4	8
7	2	3
8	7	15
9	8	12
10	6	9
11	11	9

(b)

Çizelge 4.1. Atanan noktaların bilgisi (a: noktaların komşularının listesi, b: noktaların konumları)

3. Tarama işlemine sıradaki nokta ile devam edilir. Tarama işlemine hedef noktası açık listeye eklenene kadar ya da açık liste boşalana kadar devam edilir. Hedef nokta bulunduktan sonra hedef noktadan geriye doğru ebeveyn noktalar takip edilerek başlangıç noktasına kadar ulaşılır. Bu rota algoritmanın oluşturduğu en kısa yoldur. Şekil 4.23'te PRM-A* algoritmasının oluşturduğu rota gösterilmiştir.



Şekil 4.23. PRM-A* algoritmasının oluşturduğu rota

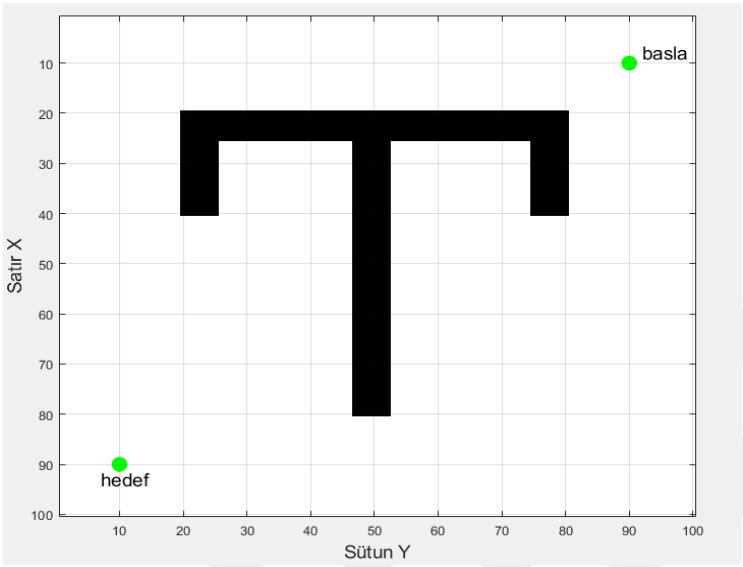
5. A* VE PRM-A* YOL PLANLAMA ALGORİTMALARININ ANALİZİ VE KARŞILAŞTIRILMASI

Önceki bölümlerde A* algoritması ile PRM-A* algoritması geniş bir şekilde açıklanmış ve örnekler verilmiştir. A* ve PRM-A* algoritmaları farklı açılardan birçok avantaj ve dezavantajlara sahiptir. Bu bölümde A* ve PRM-A* algoritmaları 100x100'lük, 200x200'lük, 300x300'lük ve 400x400'lük 10 farklı haritada çalıştırılıp sonuçları incelenmiştir. Sonuçlar bulunan yolun uzunluğu ve algoritmanın yolu oluşturma süresi kriterleri üzerinden incelenmiştir. Algoritmanın yolu oluşturma süresinde MATLAB programının algoritmaları çalıştırırken CPU üzerinden ölçtüğü süre kullanılmıştır. PRM-A* algoritması rastgele noktalar atayarak hedefini aradığı için her çalışmasında farklı rotalar hesaplamaktadır. Bunun için PRM-A* algoritması her senaryoda 5'er defa çalıştırılıp ortalaması A* algoritması ile karşılaştırılacaktır. A* algoritması da 5 sefer çalıştırılacaktır. A* algoritması her çalışmasında aynı rotayı bulduğu için rotada bir değişiklik olmayacaktır. Sadece rota oluşturma süresinde çok küçük farklar oluşmaktadır. Eşit bir karşılaştırma olması için böyle bir yol izlenmiştir.

PRM algoritması 4.2. bölümde bahsedildiği gibi rastgele nokta atadıktan sonra rota hesabı için Dijkstra, A*, D* ve çeşitli sezgisel optimizasyon algoritmaları kullanır. Bu tez çalışmasında A* algoritması kullanıldığı için bu algoritma PRM-A* algoritması olarak ifade edilmiştir.

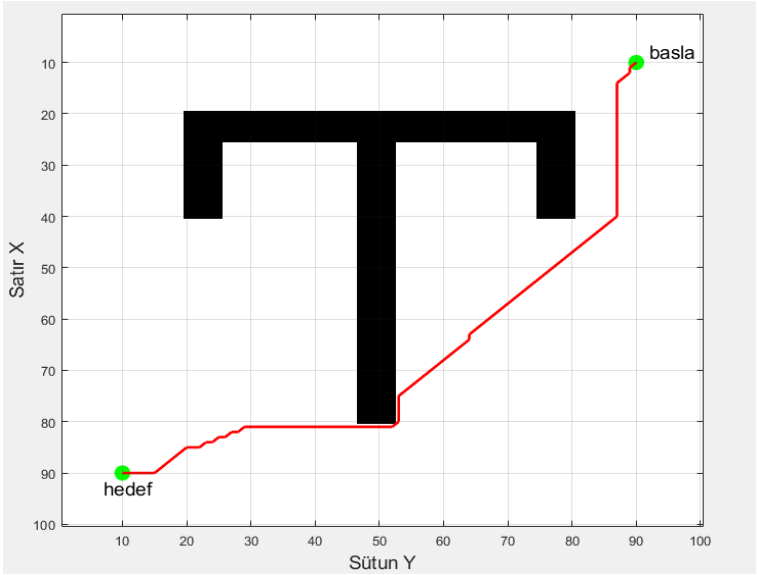
5.1. Senaryo-1

Bu bölümde Çek Teknik Üniversitesi, Sibernetik Bölümü, Akıllı ve Mobil Robotik Grubunun bir çalışmasındaki "T" isimli harita boyutları değiştirilerek kullanılmıştır [71, 3]. Senaryo-1 haritası Şekil 5.1'de görülmektedir.



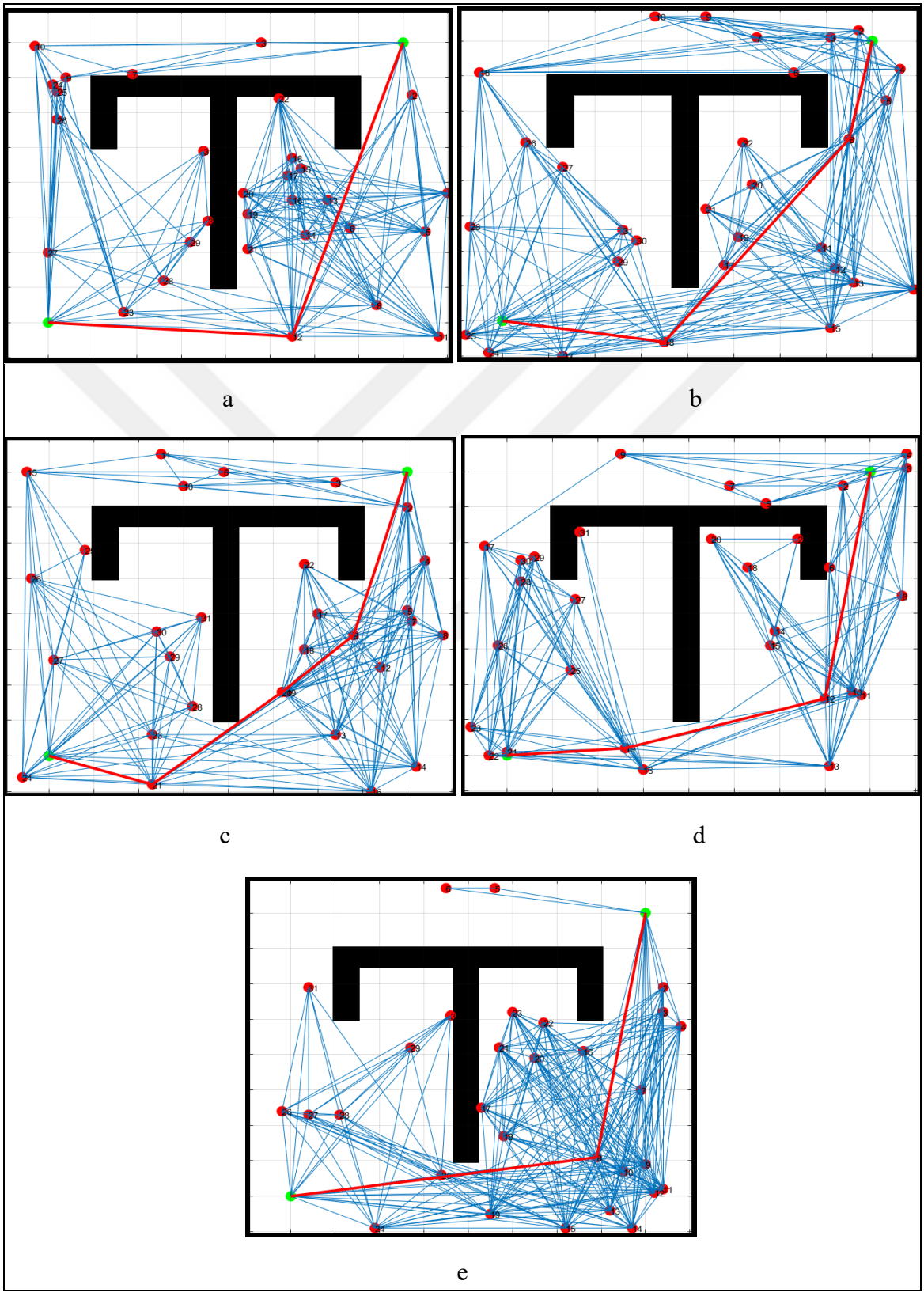
Şekil 5.1. Senaryo-1'deki arama alanı [71]

Ortamdaki siyah alanlar engelleri simgelemektedir. Harita 100x100 boyutundadır. A* algoritmasının senaryo-1 haritasında oluşturduğu rota Şekil 5.2'de görülmektedir.



Şekil 5.2. Senaryo-1 haritası için A* algoritmasının oluşturduğu rota

Şekil 5.3'te PRM-A* algoritmasının senaryo-1 haritasında oluşturduğu rotalar görülmektedir.



Şekil 5.3. Senaryo-1 haritası için PRM-A* algoritmasının 5 iterasyonlu çözümü

Senaryo-1 haritasında başlangıç noktası ile bitiş noktası arasındaki en kısa yolun A* ve PRM-A* algoritmasıyla bulunması ile bulunan yolun uzunluğu, algoritmanın yolu oluşturma süresi, algoritmaların 5 sefer çalışması sonucu bulunan veriler ve ortalama değerleri Çizelge 5.1' de gösterilmiştir.

Çizelge 5.1. Senaryo-1 haritasında A* ve PRM-A* algoritmalarının performansları

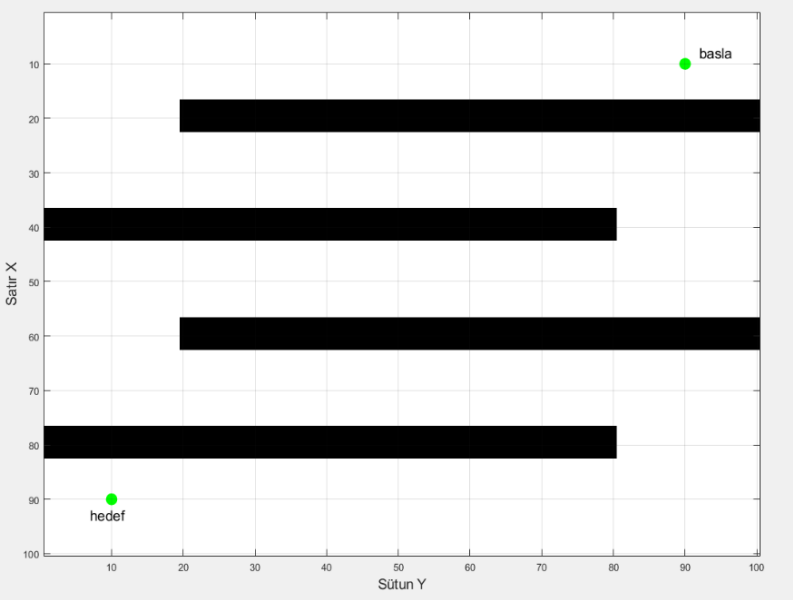
Senaryo-1	Deneme	Yol Uzunluğu	Süre (sn)	Ortalama Yol	Ortalama Süre (sn)
A*	1	132,46	10,94	132,46	10,95
	2	132,46	10,96	132,46	10,95
	3	132,46	10,92	132,46	10,95
	4	132,46	11,03	132,46	10,95
	5	132,46	10,88	132,46	10,95
PRM-A*	1	142,79	1,12	137,48	1,02
	2	134,4	0,97	137,48	1,02
	3	133,46	0,96	137,48	1,02
	4	137,03	0,91	137,48	1,02
	5	139,74	1,11	137,48	1,02

Çizelge 5.1'de görüleceği üzere senaryo-1 haritası için A* algoritmasının oluşturduğu yolun uzunluğu 132,46 birim iken yol oluşturma süresi ortalama olarak 10,95 sn sürmüştür. PRM-A* algoritmasının oluşturduğu yol ortalama 137,48 birim iken yol oluşturma süresi ortalama 1,02 sn sürmüştür. Çizelge 5.1'deki verilerin ışığında PRM-A* algoritması A* algoritmasına göre ortalama 10,74 kat daha hızlı yol oluşturmuştur. Oluşturulan yol uzunluğu arasında ise 5,02 birim fark bulunmaktadır.

Yukarıdaki verilerde görüldüğü üzere A* algoritması verimli bir algoritma olmasına rağmen büyük ölçekli haritalarda geç sonuç bulmaktadır. PRM-A* algoritması A* algoritmasına göre çok daha kısa sürelerde sonuç bulmaktadır. Fakat A* algoritmasının bir avantajı ise her çalışmasında aynı rotayı bulabilmesidir. Ayrıca A* algoritması az da olsa her denemede PRM-A* algoritmasından daha kısa yol oluşturabilmiştir. PRM-A* algoritması çalışma mantığı gereği rastgele nokta ataması yaparak rota hesapladığı için her çalışmasında aynı rotayı hesaplayamamaktadır.

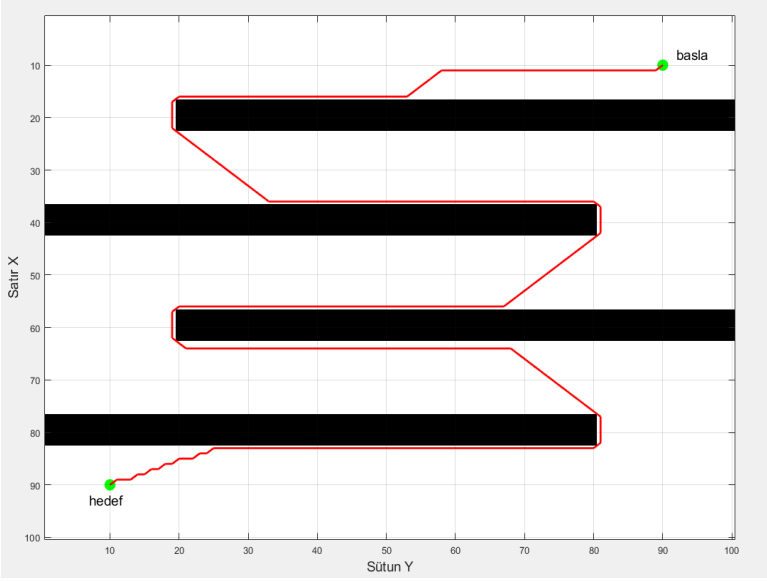
5.2. Senaryo-2

Bu bölümde yine literatürdeki “back and forth” isimli harita boyutları değiştirilerek kullanılmıştır [71, 3]. Senaryo-2 haritası Şekil 5.4’te görülmektedir.



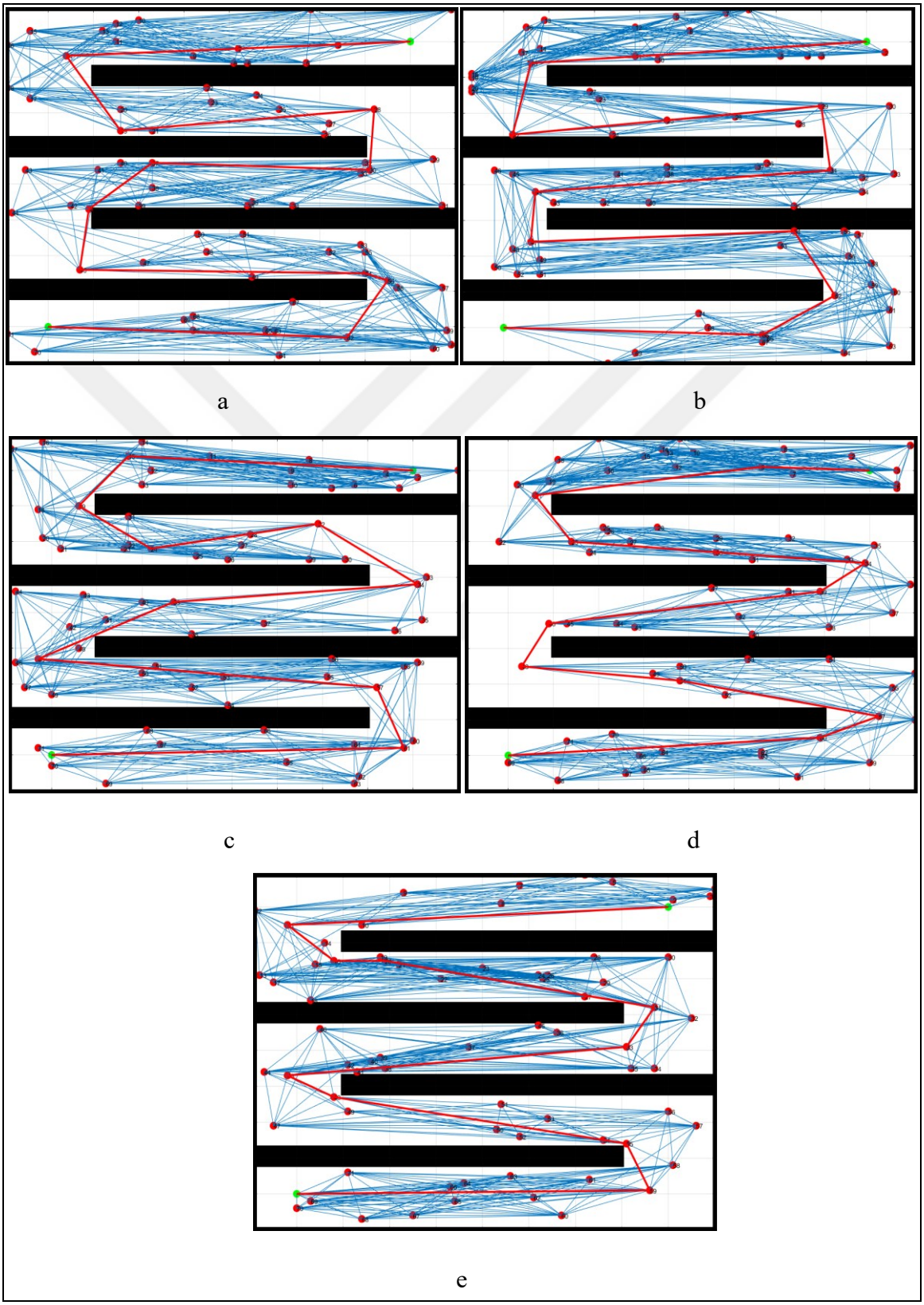
Şekil 5.4. Senaryo-2’deki arama alanı [71]

Ortamdaki siyah alanlar engelleri simgelemektedir. Harita 100x100 boyutundadır. A* algoritmasının senaryo-2 haritasında oluşturduğu rota Şekil 5.5’te görülmektedir.



Şekil 5.5. Senaryo-2 haritası için A* algoritmasının oluşturduğu rota

Şekil 5.6'da PRM-A* algoritmasının senaryo-2 haritasında oluşturduğu rotalar görülmektedir.



Şekil 5.6. Senaryo-2 haritası için PRM-A* algoritmasının 5 iterasyonlu çözümü

Senaryo-2 haritasında başlangıç noktası ile bitiş noktası arasındaki en kısa yolun A* ve PRM-A* algoritmasıyla bulunması ile bulunan yolun uzunluğu, algoritmanın yolu oluşturma süresi, algoritmaların 5 sefer çalışması sonucu bulunan veriler ve ortalama değerleri Çizelge 5.2'de gösterilmiştir.

Çizelge 5.2. Senaryo-2 haritasında A* ve PRM-A* algoritmalarının performansları

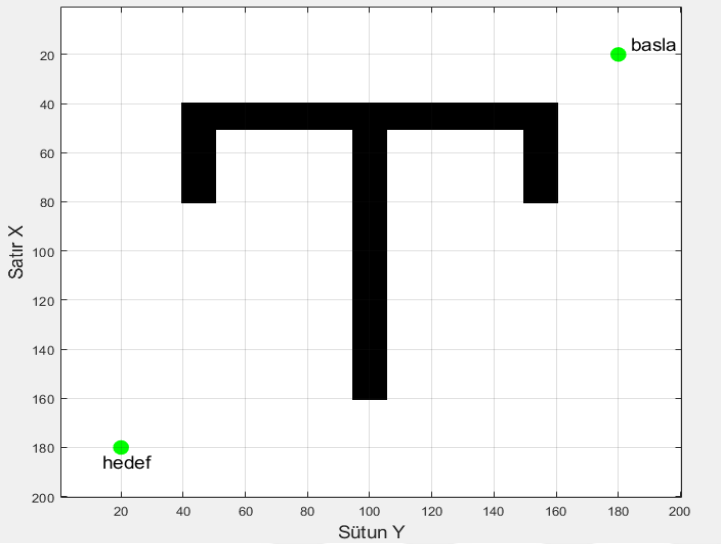
Senaryo-2	Deneme	Yol Uzunluğu	Süre (sn)	Ortalama Yol	Ortalama Süre(sn)
A*	1	372,85	20,77	372,85	20,29
	2	372,85	20,58	372,85	20,29
	3	372,85	19,5	372,85	20,29
	4	372,85	20,58	372,85	20,29
	5	372,85	20,02	372,85	20,29
PRM-A*	1	410,72	2,14	417,31	2,22
	2	415,18	2,28	417,31	2,22
	3	426,1	2,24	417,31	2,22
	4	405,94	2,22	417,31	2,22
	5	428,62	2,23	417,31	2,22

Çizelge 5.2'den de görüleceği üzere senaryo-2 haritası için A* algoritmasının oluşturduğu yolun uzunluğu 372,85 birim iken yol oluşturma süresi ortalama olarak 20,29 sn sürmüştür. PRM-A* algoritmasının oluşturduğu yol ortalama 417,31 birim iken yol oluşturma süresi ortalama 2,22 sn sürmüştür. Çizelge 5.2'deki verilerin ışığında PRM-A* algoritması A* algoritmasına göre ortalama 9,14 kat daha hızlı rota oluşturmuştur. Oluşturulan yol uzunluğu arasında 44,46 birim fark bulunmaktadır.

Yukarıdaki verilerden görüldüğü üzere Senaryo-2'de de PRM-A* algoritması A* algoritmasına göre çok daha kısa sürede sonuç elde etmiştir. Fakat bu örnekte görüldüğü gibi PRM-A* algoritmasının karmaşık haritalarda verimi azalmaktadır. Senaryo-1'deki harita da engel oranı az olduğu için 30 nokta ataması ile yol oluşturabilmişken, Senaryo-2'deki harita da 70 nokta ataması ile yol oluşturabilmiştir. Bundan dolayı rota oluşturma süresi aynı büyüklükte haritalar kullanılmasına rağmen, Senaryo-1'deki haritaya göre 2 kat fazla çıkmıştır.

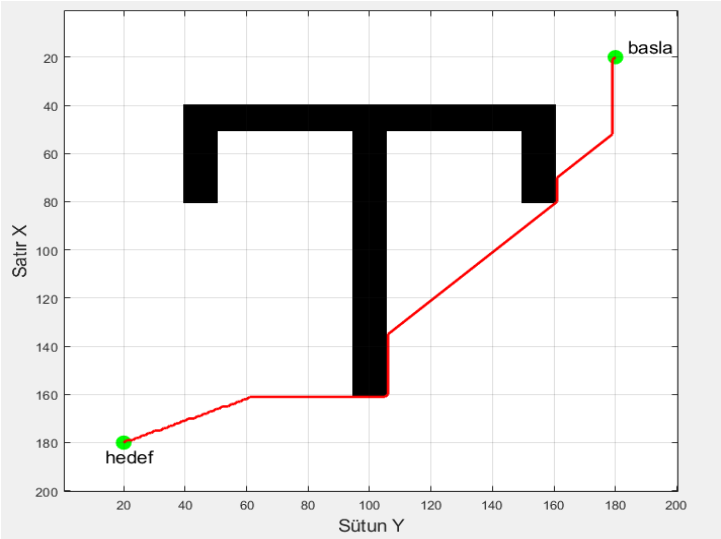
5.3. Senaryo-3

Bu bölümde yine literatürdeki “T” isimli harita boyutları değiştirilerek kullanılmıştır [71, 3]. Senaryo-3 haritası Şekil 5.7’de görülmektedir.



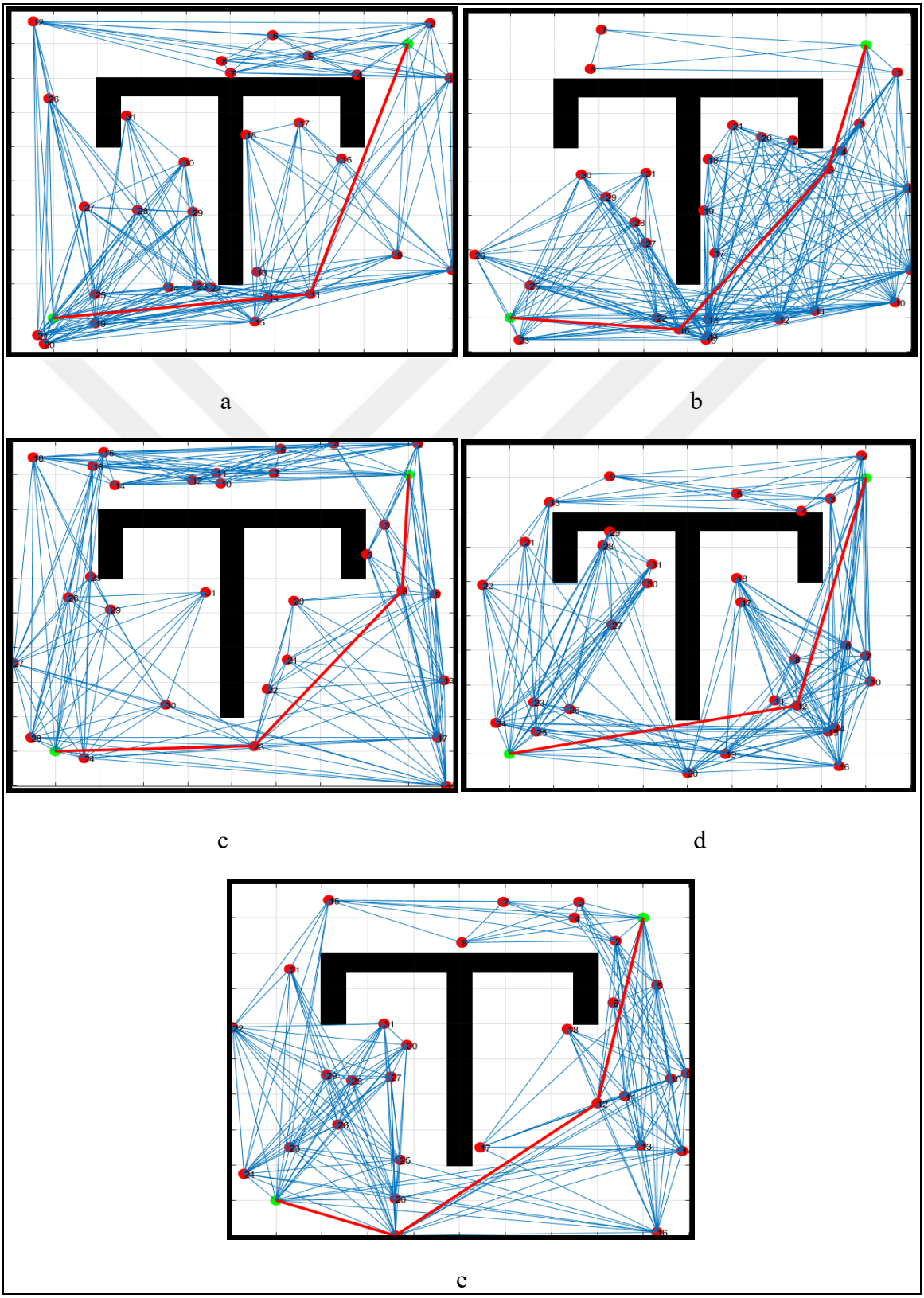
Şekil 5.7. Senaryo-3’teki arama alanı [71]

Ortamdaki siyah alanlar engelleri simgelemektedir. Burada kullanılan harita Senaryo-1’deki haritanın aynısıdır ancak boyutları 200x200’dür. A* algoritmasının senaryo-3 haritasında oluşturduğu rota Şekil 5.8’de görülmektedir.



Şekil 5.8. Senaryo-3 haritası için A* algoritmasının oluşturduğu rota

Şekil 5.9'da PRM-A* algoritmasının senaryo-3 haritasında oluşturduğu rotalar görülmektedir.



Şekil 5.9. Senaryo-3 haritası için PRM-A* algoritmasının 5 iterasyonlu çözümü

Senaryo-3 haritasında başlangıç noktası ile bitiş noktası arasındaki en kısa yolun A* ve PRM-A* algoritmasıyla bulunması ile bulunan yolun uzunluğu, algoritmanın yolu oluşturma süresi, algoritmaların 5 sefer çalışması sonucu bulunan veriler ve ortalama değerleri Çizelge 5.3'te gösterilmiştir.

Çizelge 5.3. Senaryo-3 haritasında A* ve PRM-A* algoritmalarının performansları

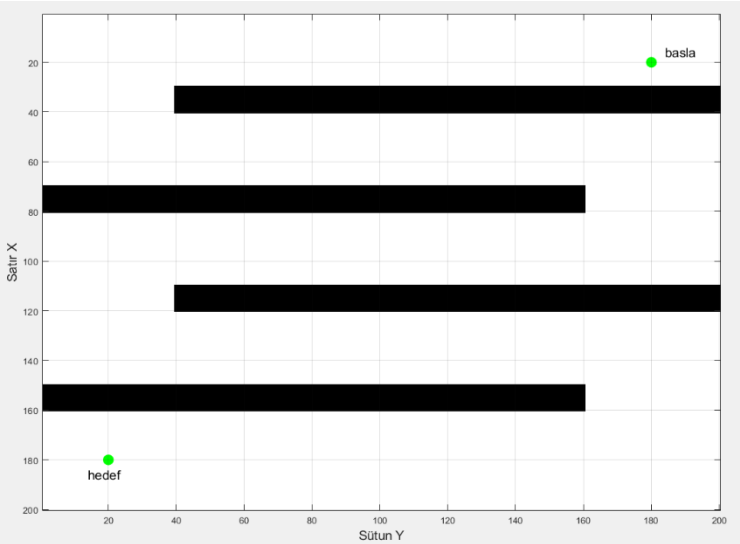
Senaryo-3	Deneme	Yol Uzunluğu	Süre (sn)	Ortalama Yol	Ortalama Süre (sn)
A*	1	264,93	158,34	264,93	157,11
	2	264,93	157,50	264,93	157,11
	3	264,93	156,68	264,93	157,11
	4	264,93	157,50	264,93	157,11
	5	264,93	155,54	264,93	157,11
PRM-A*	1	269,32	0,89	270,23	1,01
	2	266,70	1,07	270,23	1,01
	3	269,31	1,08	270,23	1,01
	4	267,60	1,01	270,23	1,01
	5	278,22	0,99	270,23	1,01

Çizelge 5.3'te görüleceği üzere Senaryo-3 haritası için A* algoritmasının oluşturduğu yolun uzunluğu 264,93 birim iken yol oluşturma süresi ortalama olarak 157,11 sn sürmüştür. PRM-A* algoritmasının oluşturduğu yol ortalama 270,23 birim iken yol oluşturma süresi ortalama 1,01 sn sürmüştür. Çizelge 5.3'teki verilerin ışığında PRM-A* algoritması A* algoritmasına göre ortalama 155,5 kat daha hızlı yol oluşturmuştur. Oluşturulan yol uzunluğu arasında 5,3 birim fark bulunmaktadır.

Senaryo-3'te Senaryo-1'deki haritanın 200x200 boyutlusu kullanılmıştır. Yukarıdaki verilerde de görüldüğü üzere yol oluşturmak için kullanılan haritaların boyutları büyüdükçe A* algoritmasının yol oluşturma süresi aşırı miktarda artmaktadır. Haritanın boyutları 4 katına da çıksa engel yoğunluğunun az olmasından dolayı PRM-A* algoritması Senaryo-1'deki gibi 30 adet nokta ataması ile yol planlaması yapmış ve yaklaşık aynı sürelerde rota oluşturabilmiştir.

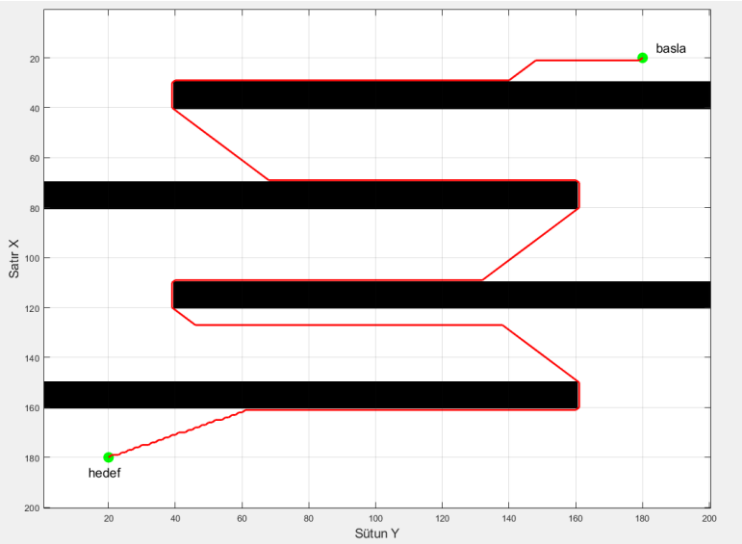
5.4. Senaryo-4

Bu bölümde yine literatürdeki “back and forth” isimli harita boyutları değiştirilerek kullanılmıştır [71, 3]. Senaryo-4 haritası Şekil 5.10’da görülmektedir.



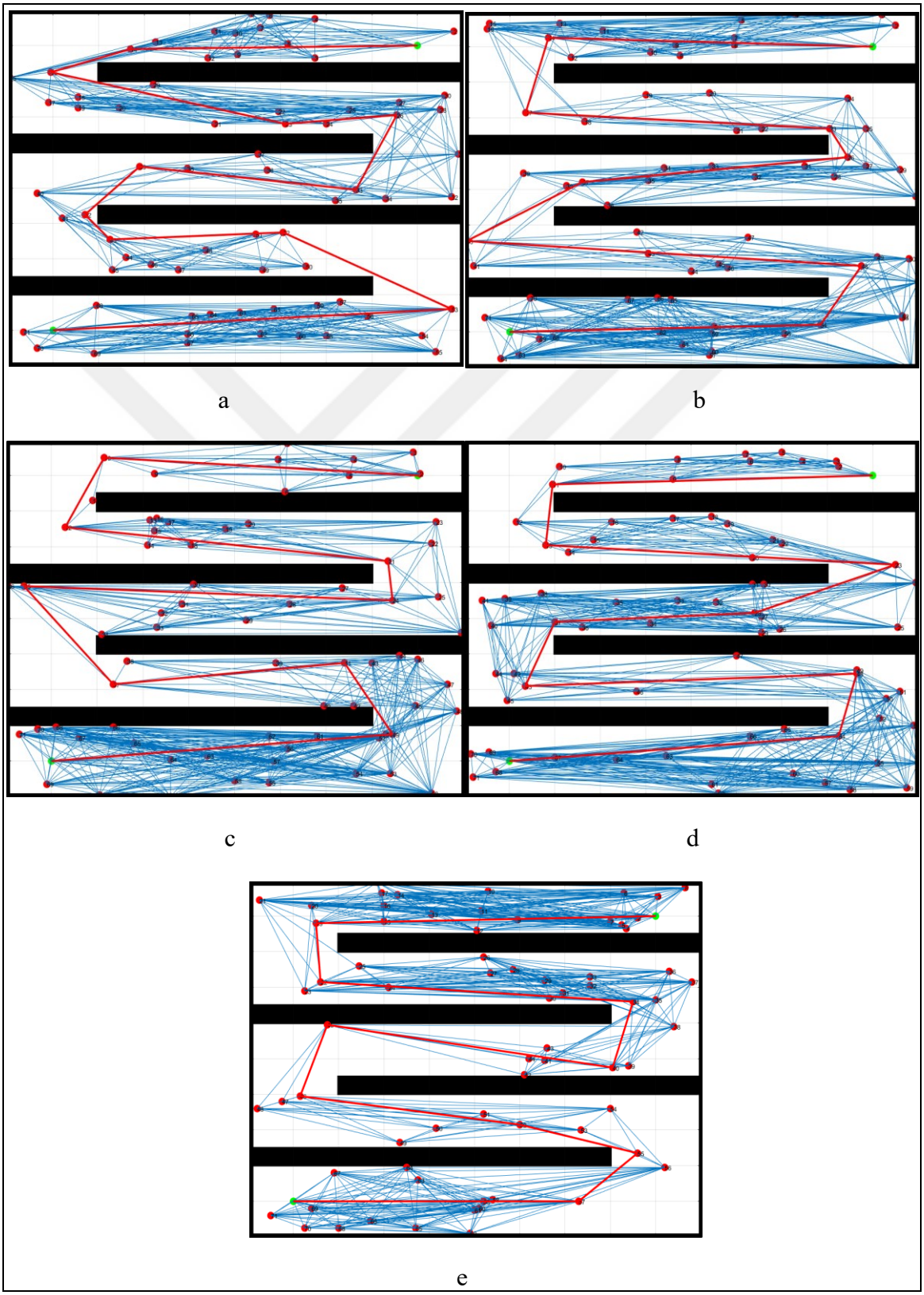
Şekil 5.10. Senaryo-4’teki arama alanı [71]

Ortamdaki siyah alanlar engelleri simgelemektedir. Burada kullanılan harita Senaryo-2’deki haritanın aynısıdır ancak boyutları 200x200’dür. A* algoritmasının senaryo-4 haritasında oluşturduğu rota Şekil 5.11’de görülmektedir.



Şekil 5.11. Senaryo-4 haritası için A* algoritmasının oluşturduğu rota

Şekil 5.12'de PRM-A* algoritmasının senaryo-4 haritasında oluşturduğu rotalar görülmektedir.



Şekil 5.12. Senaryo-4 haritası için PRM-A* algoritmasının 5 iterasyonlu çözümü

Senaryo-4 haritasında başlangıç noktası ile bitiş noktası arasındaki en kısa yolun A* ve PRM-A* algoritmasıyla bulunması ile bulunan yolun uzunluğu, algoritmanın yolu oluşturma süresi, algoritmaların 5 sefer çalışması sonucu bulunan veriler ve ortalama değerleri Çizelge 5.4'te gösterilmiştir.

Çizelge 5.4. Senaryo-4 haritasında A* ve PRM-A* algoritmalarının performansları

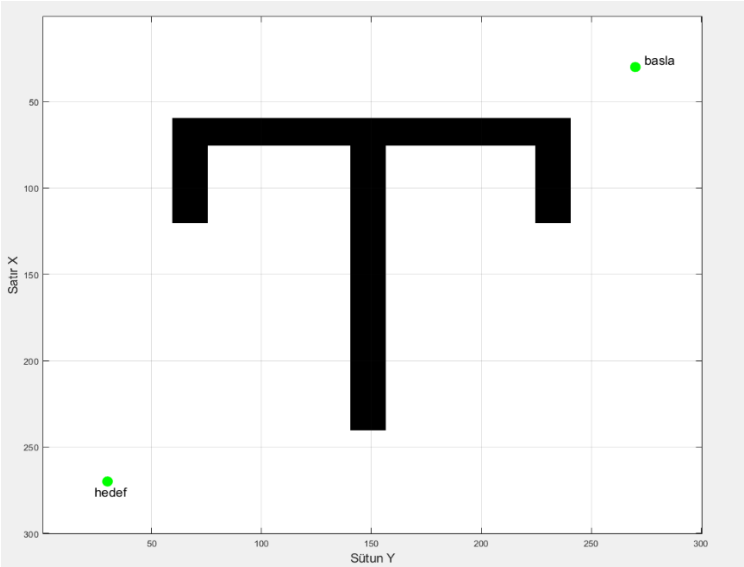
Senaryo-4	Deneme	Yol Uzunluğu	Süre (sn)	Ortalama Yol	Ortalama Süre (sn)
A*	1	737,7	243,54	737,7	240,85
	2	737,7	242,47	737,7	240,85
	3	737,7	249,35	737,7	240,85
	4	737,7	235,27	737,7	240,85
	5	737,7	233,64	737,7	240,85
PRM-A*	1	852,22	1,94	857,05	2,17
	2	866,24	2,17	857,05	2,17
	3	869,53	2,29	857,05	2,17
	4	851,4	2,22	857,05	2,17
	5	845,85	2,25	857,05	2,17

Çizelge 5.4'te görüleceği üzere senaryo-4 haritası için A* algoritmasının oluşturduğu yolun uzunluğu 737,7 birim iken yol oluşturma süresi ortalama olarak 240,85 sn sürmüştür. PRM-A* algoritmasının oluşturduğu yol ortalama 857,05 birim iken yol oluşturma süresi ortalama 2,17 sn sürmüştür. Çizelge 5.4'teki verilerin ışığında PRM-A* algoritması A* algoritmasına göre ortalama 111 kat daha hızlı rota oluşturmuştur. Oluşturulan yol uzunluğu arasında 119.35 birim fark bulunmaktadır.

Senaryo-4'te Senaryo-2'deki haritanın 200x200 boyutlusu kullanılmıştır. Haritanın boyutları 4 katına çıkmış olmasına rağmen A* algoritmasının yol oluşturma süresi Senaryo-2 haritasına göre 11,87 kat daha yüksek çıkmıştır. Fakat PRM-A* algoritması 4 kat daha büyük ve aynı engel yoğunluğuna sahip haritada yaklaşık olarak aynı sürelerde yol oluşturabilmiştir.

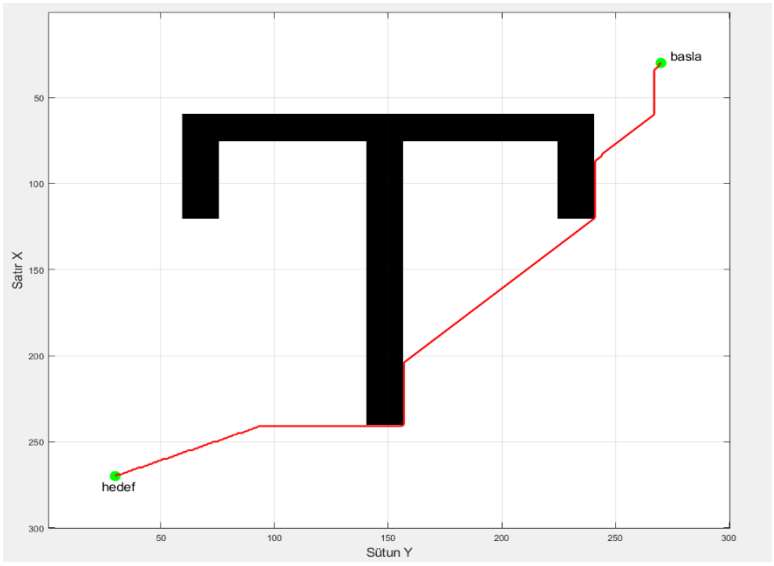
5.5. Senaryo-5

Bu bölümde yine literatürdeki “T” isimli harita boyutları değiştirilerek kullanılmıştır [71, 3]. Senaryo-5 haritası Şekil 5.13’te görülmektedir.



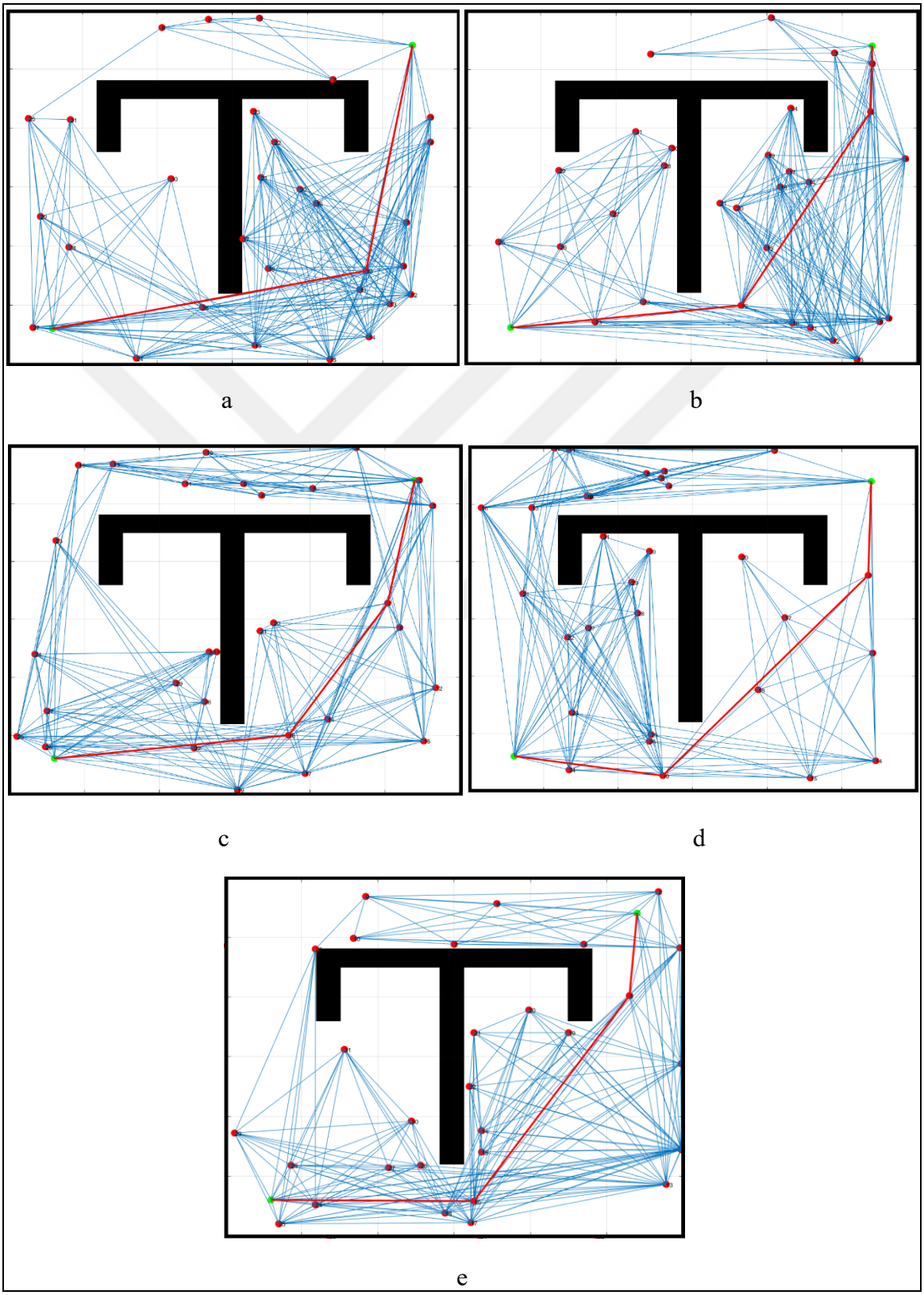
Şekil 5.13. Senaryo-5’teki arama alanı [71]

Ortamdaki siyah alanlar engelleri simgelemektedir. Burada kullanılan harita Senaryo-1’deki haritanın aynısıdır ancak boyutları 300x300’dür. A* algoritmasının senaryo-5 haritasında oluşturduğu rota Şekil 5.14’te görülmektedir.



Şekil 5.14. Senaryo-5 haritası için A* algoritmasının oluşturduğu rota

Şekil 5.15'te PRM-A* algoritmasının senaryo-5 haritasında oluşturduğu rotalar görülmektedir.



Şekil 5.15. Senaryo-5 haritası için PRM-A* algoritmasının 5 iterasyonlu çözümü

Senaryo-5 haritasında başlangıç noktası ile bitiş noktası arasındaki en kısa yolun A* ve PRM-A* algoritmasıyla bulunması ile bulunan yolun uzunluğu, algoritmanın yolu oluşturma süresi, algoritmaların 5 sefer çalışması sonucu bulunan veriler ve ortalama değerleri Çizelge 5.5'te gösterilmiştir.

Çizelge 5.5. Senaryo-5 haritasında A* ve PRM-A* algoritmalarının performansları

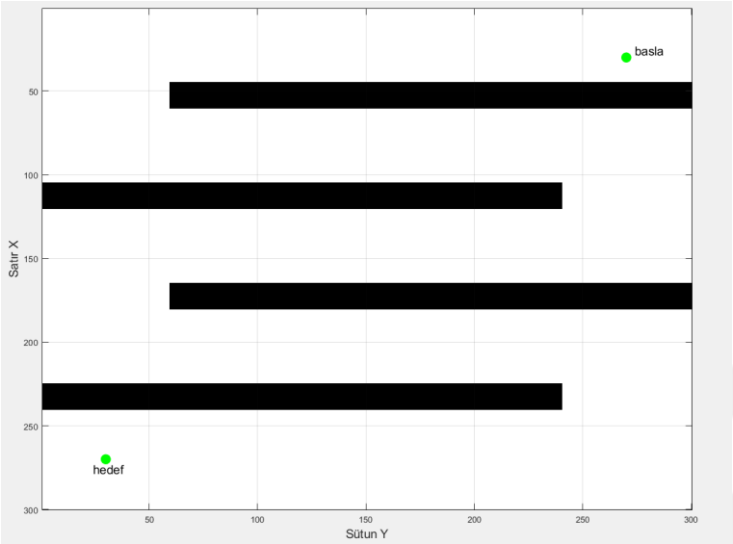
Senaryo-5	Deneme	Yol Uzunluğu	Süre (sn)	Ortalama Yol	Ortalama Süre (sn)
A*	1	396,23	685,31	396,23	684,85
	2	396,23	685,07	396,23	684,85
	3	396,23	683,56	396,23	684,85
	4	396,23	687,02	396,23	684,85
	5	396,23	683,31	396,23	684,85
PRM-A*	1	408,16	1,03	401,88	1,1
	2	396,25	1,21	401,88	1,1
	3	396,52	1,05	401,88	1,1
	4	406,32	1,09	401,88	1,1
	5	402,15	1,11	401,88	1,1

Çizelge 5.5'te görüleceği üzere Senaryo-5 haritası için A* algoritmasının oluşturduğu yolun uzunluğu 396,23 birim iken yol oluşturma süresi ortalama olarak 684,85 sn sürmüştür. PRM-A* algoritmasının oluşturduğu yol ortalama 401,88 birim iken yol oluşturma süresi ortalama 1,1 sn sürmüştür. Çizelge 5.5'teki verilerin ışığında PRM-A* algoritması A* algoritmasına göre ortalama 622,6 kat daha hızlı yol oluşturmuştur. Oluşturulan yol uzunluğu arasında 5,65 birim fark bulunmaktadır.

Senaryo-5'te Senaryo-1'deki haritanın 300x300 boyutlusu kullanılmıştır. Yukarıdaki verilerde de görüldüğü üzere yol oluşturmak için kullanılan haritaların boyutları büyüdükçe A* algoritmasının yol oluşturma süresi aşırı miktarda artmaktadır. Haritanın boyutları 9 katına da çıksa engel yoğunluğunun az olmasından dolayı PRM-A* algoritması Senaryo-1'deki gibi 30 adet nokta ataması ile yol planlaması yapmış ve yaklaşık aynı sürelerde rota oluşturabilmiştir.

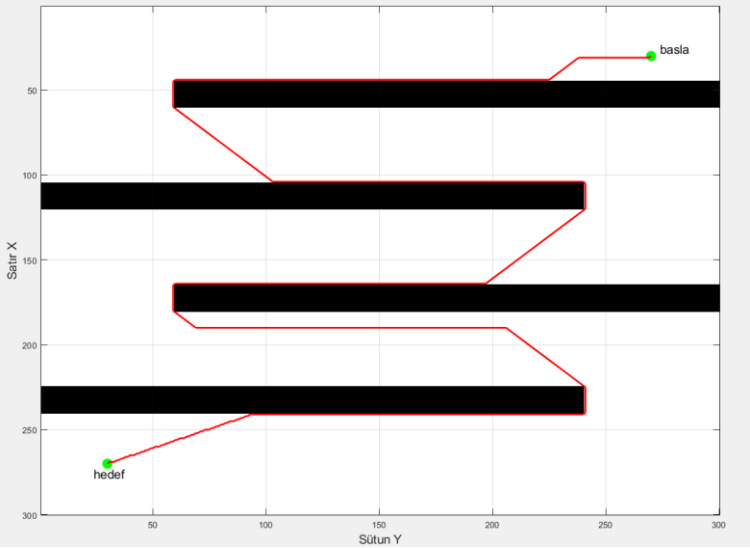
5.6. Senaryo-6

Bu bölümde yine literatürdeki “back and forth” isimli harita boyutları değiştirilerek kullanılmıştır [71, 3]. Senaryo-6 haritası Şekil 5.16’da görülmektedir.



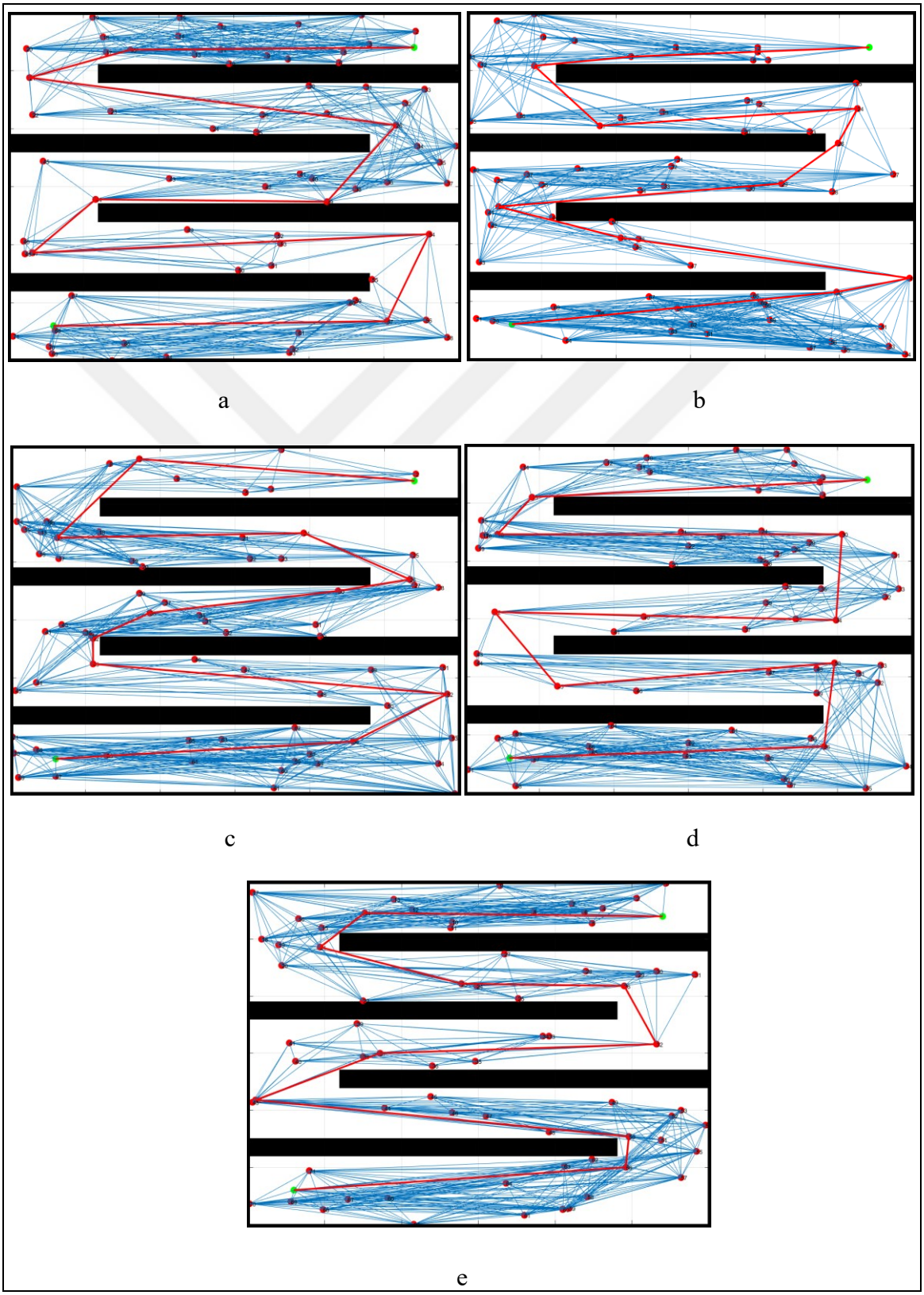
Şekil 5.16. Senaryo-6’daki arama alanı [71]

Ortamdaki siyah alanlar engelleri simgelemektedir. Burada kullanılan harita Senaryo-2’deki haritanın aynısıdır ancak boyutları 300x300’dür. A* algoritmasının senaryo-6 haritasında oluşturduğu rota Şekil 5.17’de görülmektedir.



Şekil 5.17. Senaryo-6 haritası için A* algoritmasının oluşturduğu rota

Şekil 5.18'de PRM-A* algoritmasının senaryo-6 haritasında oluşturduğu rotalar görülmektedir.



Şekil 5.18. Senaryo-6 haritası için PRM-A* algoritmasının 5 iterasyonlu çözümü

Senaryo-6 haritasında başlangıç noktası ile bitiş noktası arasındaki en kısa yolun A* ve PRM-A* algoritmasıyla bulunması ile bulunan yolun uzunluğu, algoritmanın yolu oluşturma süresi, algoritmaların 5 sefer çalışması sonucu bulunan veriler ve ortalama değerleri Çizelge 5.6'da gösterilmiştir.

Çizelge 5.6. Senaryo-6 haritasında A* ve PRM-A* algoritmalarının performansları

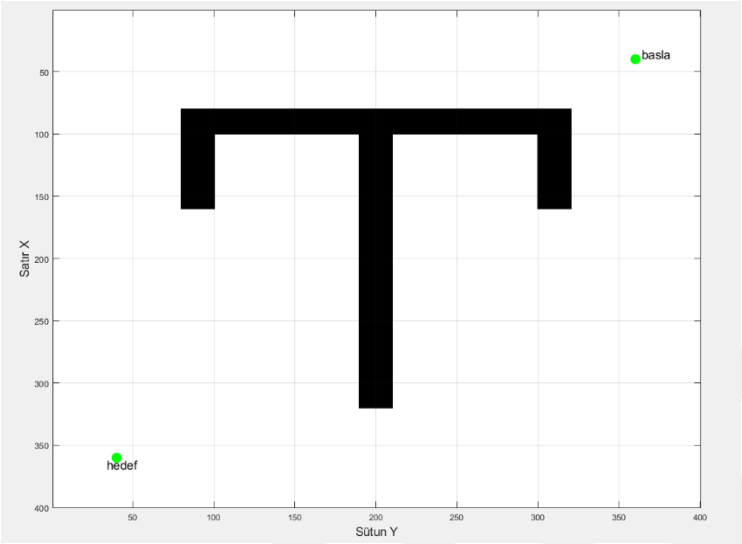
Senaryo-6	Deneme	Yol Uzunluğu	Süre (sn)	Ortalama Yol	Ortalama Süre (sn)
A*	1	1102,6	1047,16	1102,6	1047,94
	2	1102,6	1047,26	1102,6	1047,94
	3	1102,6	1050,44	1102,6	1047,94
	4	1102,6	1052,2	1102,6	1047,94
	5	1102,6	1042,66	1102,6	1047,94
PRM-A*	1	1370,9	2,37	1308,54	2,2
	2	1296,6	2,13	1308,54	2,2
	3	1274,5	2,16	1308,54	2,2
	4	1346,3	2,15	1308,54	2,2
	5	1254,4	2,2	1308,54	2,2

Çizelge 5.6'da görüleceği üzere senaryo-6 haritası için A* algoritmasının oluşturduğu yolun uzunluğu 1102,6 birim iken yol oluşturma süresi ortalama olarak 1047,94 sn sürmüştür. PRM-A* algoritmasının oluşturduğu yol ortalama 1308,54 birim iken yol oluşturma süresi ortalama 2,2 sn sürmüştür. Çizelge 5.6'daki verilerin ışığında PRM-A* algoritması A* algoritmasına göre ortalama 476,3 kat daha hızlı rota oluşturmuştur. Oluşturulan yol uzunluğu arasında 205,94 birim fark bulunmaktadır.

Senaryo-6'da Senaryo-2'deki haritanın 300x300 boyutlusu kullanılmıştır. Haritanın boyutları 9 katına çıkmış olmasına rağmen A* algoritmasının yol oluşturma süresi Senaryo-2 haritasına göre 51,64 kat daha yüksek çıkmıştır. Fakat PRM-A* algoritması 4 kat daha büyük ve aynı engel yoğunluğuna sahip haritada yaklaşık olarak aynı sürelerde yol oluşturabilmiştir.

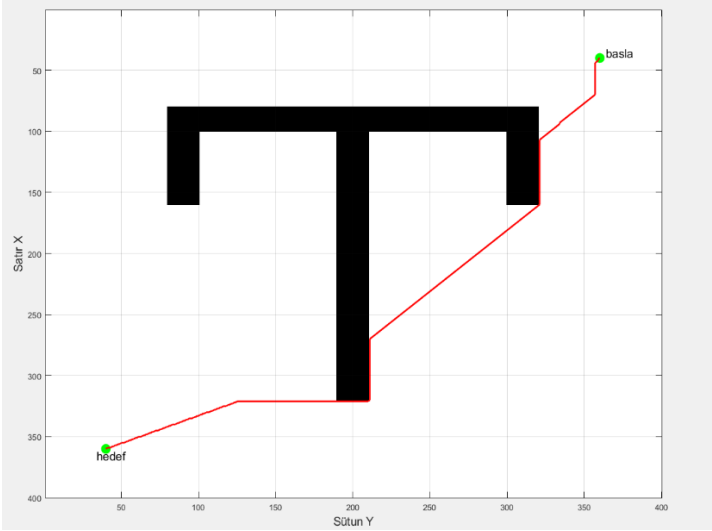
5.7. Senaryo-7

Bu bölümde yine literatürdeki “T” isimli harita boyutları değiştirilerek kullanılmıştır [71, 3]. Senaryo-7 haritası Şekil 5.19’da görülmektedir.



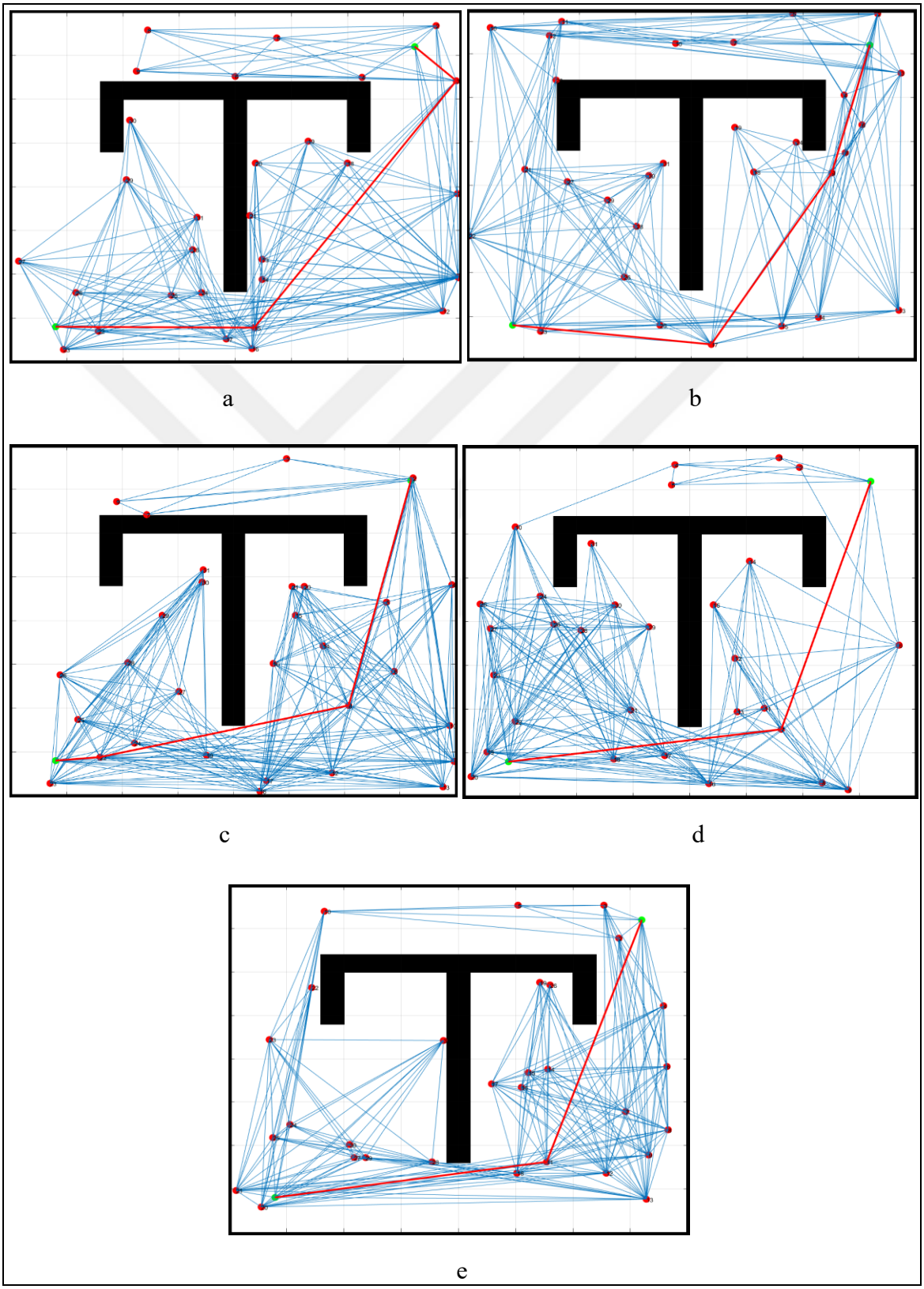
Şekil 5.19. Senaryo-7’deki arama alanı [71]

Ortamdaki siyah alanlar engelleri simgelemektedir. Burada kullanılan harita Senaryo-1’deki haritanın aynısıdır ancak boyutları 400x400’dür. A* algoritmasının senaryo-7 haritasında oluşturduğu rota Şekil 5.20’de görülmektedir.



Şekil 5.20. Senaryo-7 haritası için A* algoritmasının oluşturduğu rota

Şekil 5.21'de PRM-A* algoritmasının senaryo-7 haritasında oluşturduğu rotalar görülmektedir.



Şekil 5.21. Senaryo-7 haritası için PRM-A* algoritmasının 5 iterasyonlu çözümü

Senaryo-7 haritasında başlangıç noktası ile bitiş noktası arasındaki en kısa yolun A* ve PRM-A* algoritmasıyla bulunması ile bulunan yolun uzunluğu, algoritmanın yolu oluşturma süresi, algoritmaların 5 sefer çalışması sonucu bulunan veriler ve ortalama değerleri Çizelge 5.7'de gösterilmiştir.

Çizelge 5.7. Senaryo-7 haritasında A* ve PRM-A* algoritmalarının performansları

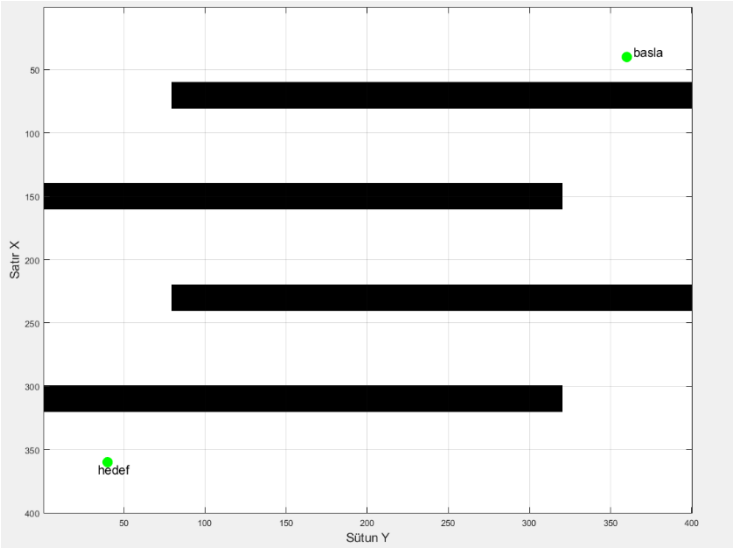
Senaryo-7	Deneme	Yol Uzunluğu	Süre (sn)	Ortalama Yol	Ortalama Süre (sn)
A*	1	529,3	2463,8	529,3	2438,98
	2	529,3	2397,68	529,3	2438,98
	3	529,3	2399,37	529,3	2438,98
	4	529,3	2472,05	529,3	2438,98
	5	529,3	2462	529,3	2438,98
PRM-A*	1	565,31	1,04	544,49	1,114
	2	553,04	1	544,49	1,114
	3	534,86	1,1	544,49	1,114
	4	537,64	1,29	544,49	1,114
	5	531,6	1,14	544,49	1,114

Çizelge 5.7'de görüleceği üzere Senaryo-7 haritası için A* algoritmasının oluşturduğu yolun uzunluğu 529,3 birim iken yol oluşturma süresi ortalama olarak 2438,98 sn sürmüştür. PRM-A* algoritmasının oluşturduğu yol ortalama 544,49 birim iken yol oluşturma süresi ortalama 1,114 sn sürmüştür. Çizelge 5.7'deki verilerin ışığında PRM-A* algoritması A* algoritmasına göre ortalama 2189,4 kat daha hızlı yol oluşturmuştur. Oluşturulan yol uzunluğu arasında 15,19 birim fark bulunmaktadır.

Senaryo-7'de Senaryo-1'deki haritanın 400x400 boyutlusu kullanılmıştır. Yukarıdaki verilerde de görüldüğü üzere yol oluşturmak için kullanılan haritaların boyutları büyüdükçe A* algoritmasının yol oluşturma süresi aşırı miktarda artmaktadır. Haritanın boyutları 16 katına da çıksa engel yoğunluğunun az olmasından dolayı PRM-A* algoritması Senaryo-1'deki gibi 30 adet nokta ataması ile yol planlaması yapmış ve yaklaşık aynı sürelerde rota oluşturabilmiştir.

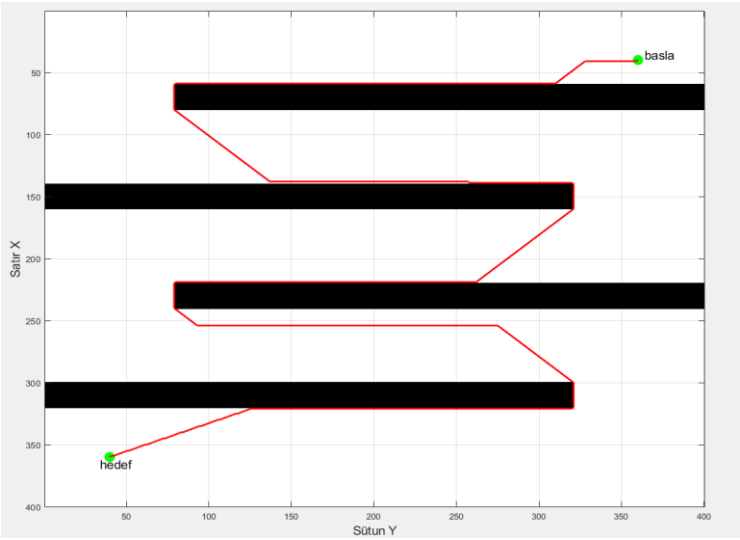
5.8. Senaryo-8

Bu bölümde yine literatürdeki “back and forth” isimli harita boyutları değiştirilerek kullanılmıştır [71, 3]. Senaryo-8 haritası Şekil 5.22’de görülmektedir.



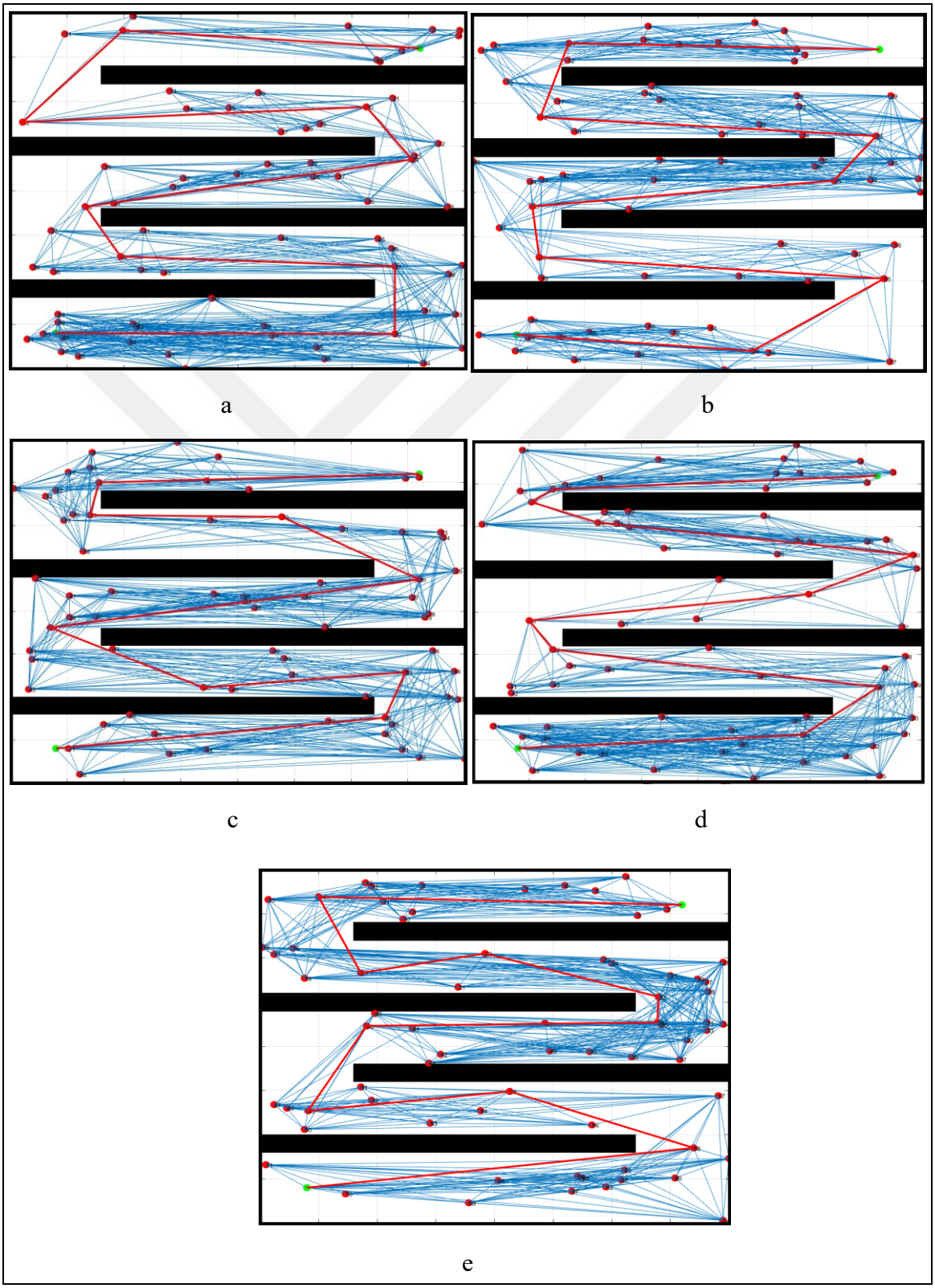
Şekil 5.22. Senaryo-8’deki arama alanı [71]

Ortamdaki siyah alanlar engelleri simgelemektedir. Burada kullanılan harita Senaryo-2’deki haritanın aynısıdır ancak boyutları 400x400’dür. A* algoritmasının senaryo-8 haritasında oluşturduğu rota Şekil 5.23’te görülmektedir.



Şekil 5.23. Senaryo-8 haritası için A* algoritmasının oluşturduğu rota

Şekil 5.24'te PRM-A* algoritmasının senaryo-8 haritasında oluşturduğu rotalar görülmektedir.



Şekil 5.24. Senaryo-8 haritası için PRM-A* algoritmasının 5 iterasyonlu çözümü

Senaryo-8 haritasında başlangıç noktası ile bitiş noktası arasındaki en kısa yolun A* ve PRM-A* algoritmasıyla bulunması ile bulunan yolun uzunluğu, algoritmanın yolu oluşturma süresi, algoritmaların 5 sefer çalışması sonucu bulunan veriler ve ortalama değerleri Çizelge 5.8'de gösterilmiştir.

Çizelge 5.8. Senaryo-8 haritasında A* ve PRM-A* algoritmalarının performansları

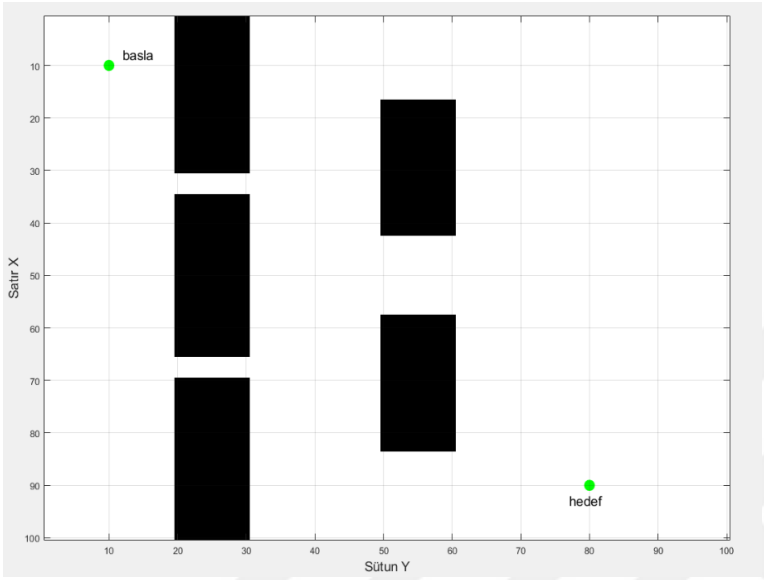
Senaryo-8	Deneme	Yol Uzunluğu	Süre (sn)	Ortalama Yol	Ortalama Süre (sn)
A*	1	1467,4	3536,31	1467,4	3532,8
	2	1467,4	3535,18	1467,4	3532,8
	3	1467,4	3524,33	1467,4	3532,8
	4	1467,4	3532,35	1467,4	3532,8
	5	1467,4	3535,8	1467,4	3532,8
PRM-A*	1	1742,1	2,24	1700,34	2,27
	2	1699,8	2,21	1700,34	2,27
	3	1640,3	2,29	1700,34	2,27
	4	1690,7	2,32	1700,34	2,27
	5	1728,8	2,29	1700,34	2,27

Çizelge 5.8'de görüleceği üzere senaryo-8 haritası için A* algoritmasının oluşturduğu yolun uzunluğu 1467,4 birim iken yol oluşturma süresi ortalama olarak 3532,8 sn sürmüştür. PRM-A* algoritmasının oluşturduğu yol ortalama 1700,34 birim iken yol oluşturma süresi ortalama 2,27 sn sürmüştür. Çizelge 5.8'deki verilerin ışığında PRM-A* algoritması A* algoritmasına göre ortalama 1556,3 kat daha hızlı rota oluşturmuştur. Oluşturulan yol uzunluğu arasında 232,94 birim fark bulunmaktadır.

Senaryo-8'de Senaryo-2'deki haritanın 400x400 boyutlusu kullanılmıştır. Haritanın boyutları 16 katına çıkmış olmasına rağmen A* algoritmasının yol oluşturma süresi Senaryo-2 haritasına göre 174,1 kat daha yüksek çıkmıştır. Fakat PRM-A* algoritması 16 kat daha büyük ve aynı engel yoğunluğuna sahip haritada yaklaşık olarak aynı sürelerde yol oluşturabilmiştir.

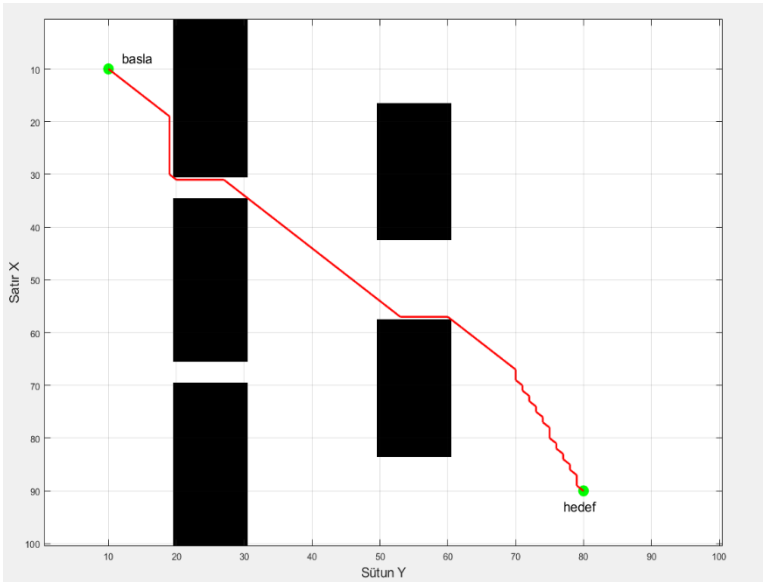
5.9. Senaryo-9

Bu bölümde yine literatürdeki “Gaps” isimli harita boyutları değiştirilerek kullanılmıştır [71, 3]. Senaryo-9 haritası Şekil 5.25’te görülmektedir.



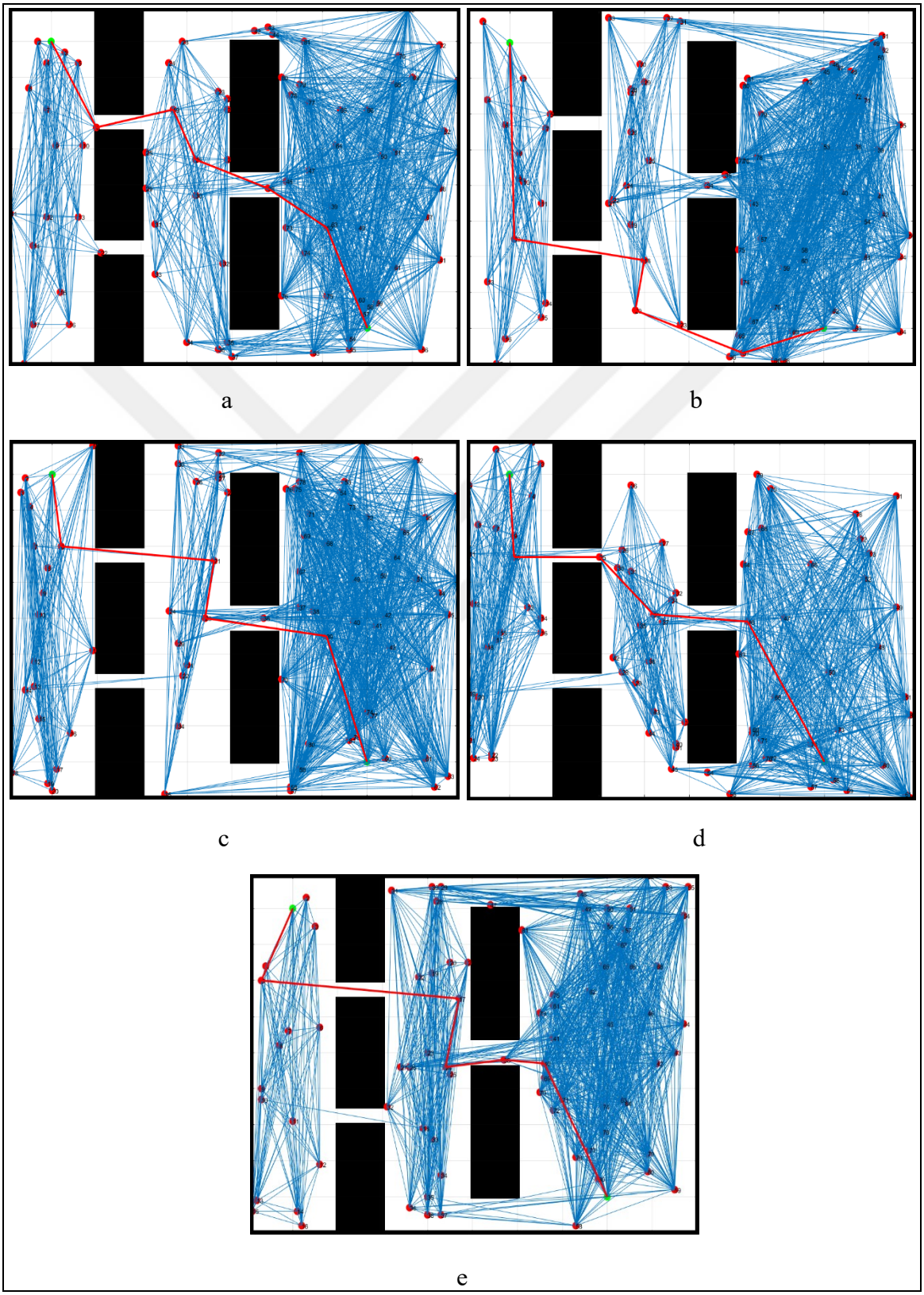
Şekil 5.25. Senaryo-9’daki arama alanı [71]

Ortamdaki siyah alanlar engelleri simgelemektedir. Harita 100x100 boyutundadır. A* algoritmasının senaryo-9 haritasında oluşturduğu rota Şekil 5.26’da görülmektedir.



Şekil 5.26. Senaryo-9 haritası için A* algoritmasının oluşturduğu rota

Şekil 5.27'de PRM-A* algoritmasının senaryo-9 haritasında oluşturduğu rotalar görülmektedir.



Şekil 5.27. Senaryo-9 haritası için PRM-A* algoritmasının 5 iterasyonlu çözümü

Senaryo-9 haritasında başlangıç noktası ile bitiş noktası arasındaki en kısa yolun A* ve PRM-A* algoritmasıyla bulunması ile bulunan yolun uzunluğu, algoritmanın yolu oluşturma süresi, algoritmaların 5 sefer çalışması sonucu bulunan veriler ve ortalama değerleri Çizelge 5.9'da gösterilmiştir.

Çizelge 5.9. Senaryo-9 haritasında A* ve PRM-A* algoritmalarının performansları

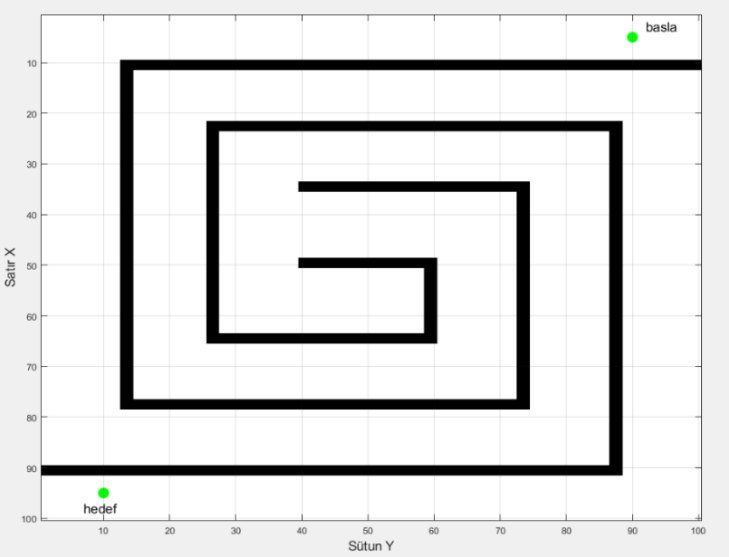
Senaryo-9	Deneme	Yol Uzunluğu	Süre (sn)	Ortalama Yol	Ortalama Süre (sn)
A*	1	117,2	6,58	117,2	6,70
	2	117,2	6,72	117,2	6,70
	3	117,2	6,71	117,2	6,70
	4	117,2	6,68	117,2	6,70
	5	117,2	6,8	117,2	6,70
PRM-A*	1	122,91	3,72	134,80	3,97
	2	144,91	4,25	134,80	3,97
	3	134,06	4,19	134,80	3,97
	4	125,66	3,57	134,80	3,97
	5	146,47	4,11	134,80	3,97

Çizelge 5.9'dan görüldüğü üzere senaryo-9 haritası için A* algoritmasının oluşturduğu yolun uzunluğu 117,2 birim iken yol oluşturma süresi ortalama olarak 6,7 sn sürmüştür. PRM-A* algoritmasının oluşturduğu yol ortalama 134,8 birim iken yol oluşturma süresi ortalama 3,97 sn sürmüştür. Çizelge 5.9'daki verilerin ışığında PRM-A* algoritması A* algoritmasına göre ortalama 1,68 kat daha hızlı yol oluşturmuştur. Oluşturulan yol uzunluğu arasında 17,6 birim fark bulunmaktadır.

Senaryo-9 haritası engel yoğunluğunun fazla ve dar geçitlerin olduğu bir haritadır. Yukarıdaki verilerde de görüldüğü üzere PRM-A* algoritması diğer senaryolarda olduğu gibi A* algoritmasına göre daha hızlı yol oluşturabilmiştir. Fakat PRM-A* ile A* algoritmaları arasındaki bu süre farkı çok azalmış durumdadır. Bunun sebebi ise PRM-A* algoritması engel yoğunluğunun fazla ve dar geçitlerin çok olduğu haritalarda yol oluşturabilmek için fazla sayıda nokta ataması yaptığından ve bundan nedenle taramanın çok olmasından dolayı daha uzun zamanlarda yol oluşturabilmektedir.

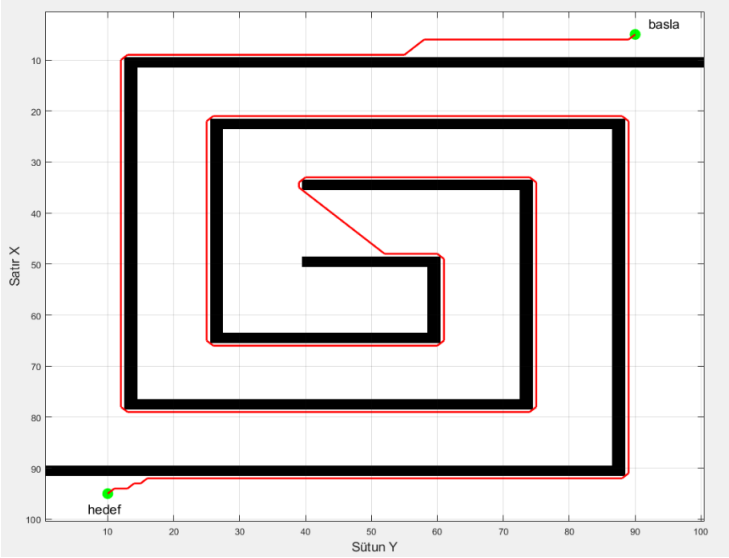
5.10. Senaryo-10

Bu bölümde yine literatürdeki “Square_Spiral” isimli harita boyutları değiştirilerek kullanılmıştır [71, 3]. Senaryo-10 haritası Şekil 5.28’de görülmektedir.



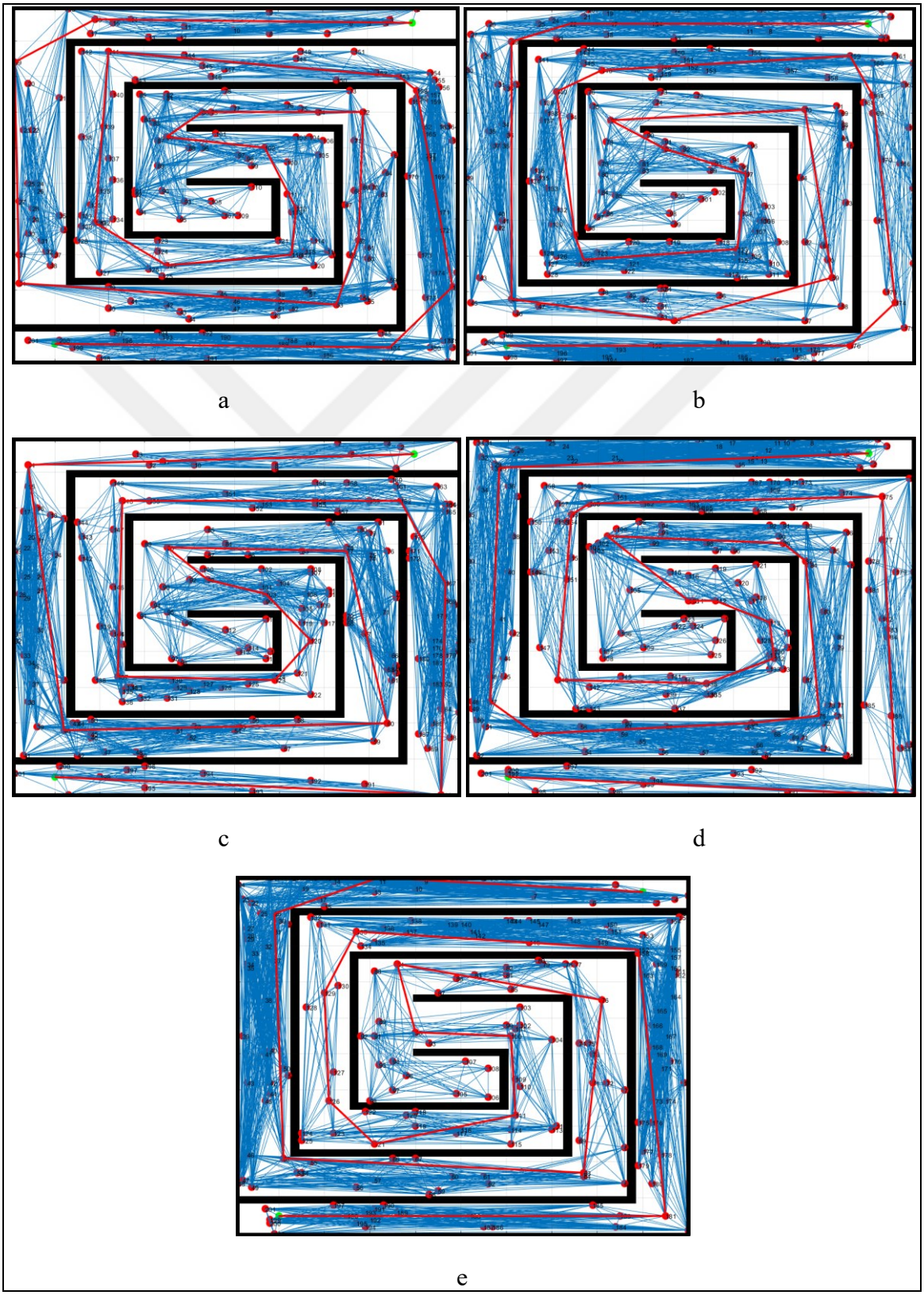
Şekil 5.28. Senaryo-10’daki arama alanı [71]

Ortamdaki siyah alanlar engelleri simgelemektedir. Harita 100x100 boyutundadır. A* algoritmasının senaryo-10 haritasında oluşturduğu rota Şekil 5.29’da görülmektedir.



Şekil 5.29. Senaryo-10 haritası için A* algoritmasının oluşturduğu rota

Şekil 5.30'da PRM-A* algoritmasının senaryo-10 haritasında oluşturduğu rotalar görülmektedir.



Şekil 5.30. Senaryo-10 haritası için PRM-A* algoritmasının 5 iterasyonlu çözümü

Senaryo-10 haritasında başlangıç noktası ile bitiş noktası arasındaki en kısa yolun A* ve PRM-A* algoritmasıyla bulunması ile bulunan yolun uzunluğu, algoritmanın yolu oluşturma süresi, algoritmaların 5 sefer çalışması sonucu bulunan veriler ve ortalama değerleri Çizelge 5.10'da gösterilmiştir.

Çizelge 5.10. Senaryo-10 haritasında A* ve PRM-A* algoritmalarının performansları

Senaryo-10	Deneme	Yol Uzunluğu	Süre (sn)	Ortalama Yol	Ortalama Süre (sn)
A*	1	631,84	27,2	631,84	26,992
	2	631,84	27,03	631,84	26,992
	3	631,84	27,09	631,84	26,992
	4	631,84	26,73	631,84	26,992
	5	631,84	26,91	631,84	26,992
PRM-A*	1	700,74	10,04	694,04	10,356
	2	681,08	10,09	694,04	10,356
	3	693,95	10,02	694,04	10,356
	4	702,85	10,48	694,04	10,356
	5	691,57	11,15	694,04	10,356

Çizelge 5.10'da görüleceği üzere senaryo-10 haritası için A* algoritmasının oluşturduğu yolun uzunluğu 631,84 birim iken yol oluşturma süresi ortalama olarak 26,992 sn sürmüştür. PRM-A* algoritmasının oluşturduğu yol ortalama 694,04 birim iken yol oluşturma süresi ortalama 10,356 sn sürmüştür. Çizelge 5.10'daki verilerin ışığında PRM-A* algoritması A* algoritmasına göre ortalama 2,6 kat daha hızlı yol oluşturmuştur. Oluşturulan yol uzunluğu arasında 62,2 birim fark bulunmaktadır.

Senaryo-10 haritası diğer haritalara göre çok daha karışık bir haritadır. Bundan dolayı iki algoritma da diğer 100x100'lük haritalara göre daha uzun sürelerde yol oluşturmuşlardır. Özellikle PRM-A* algoritması diğer haritalardaki performanslarına göre en az 2,5 kat daha geç yol oluşturmuştur. A* algoritması engel yoğunluğundan etkilenmeyen bir algoritmadır. Fakat burada PRM-A* algoritmasının bu kadar gerisinde kalmasının nedeni ise haritanın labirent şeklinden dolayı oluşan yolun uzun olmasındandır.

5.11. Senaryo-11

Bu bölümde PRM-A* algoritmasının nokta atama sayılarının algoritmanın performansını nasıl etkileyeceğini görmek için literatürdeki "Gaps" isimli harita 100x100 boyutlarında ayarlanarak kullanılmıştır (Bkz. Şekil 5.25) [71, 3]. Senaryo-11 haritasında PRM-A* algoritması 40, 80, 120 ve 160 adet nokta ataması yaparak yol oluşturmayı denemiş ve algoritmanın performansı Çizelge 5.11'de gösterilmiştir.

Çizelge 5.11. Senaryo-11 haritasında PRM-A* algoritmasının performansı

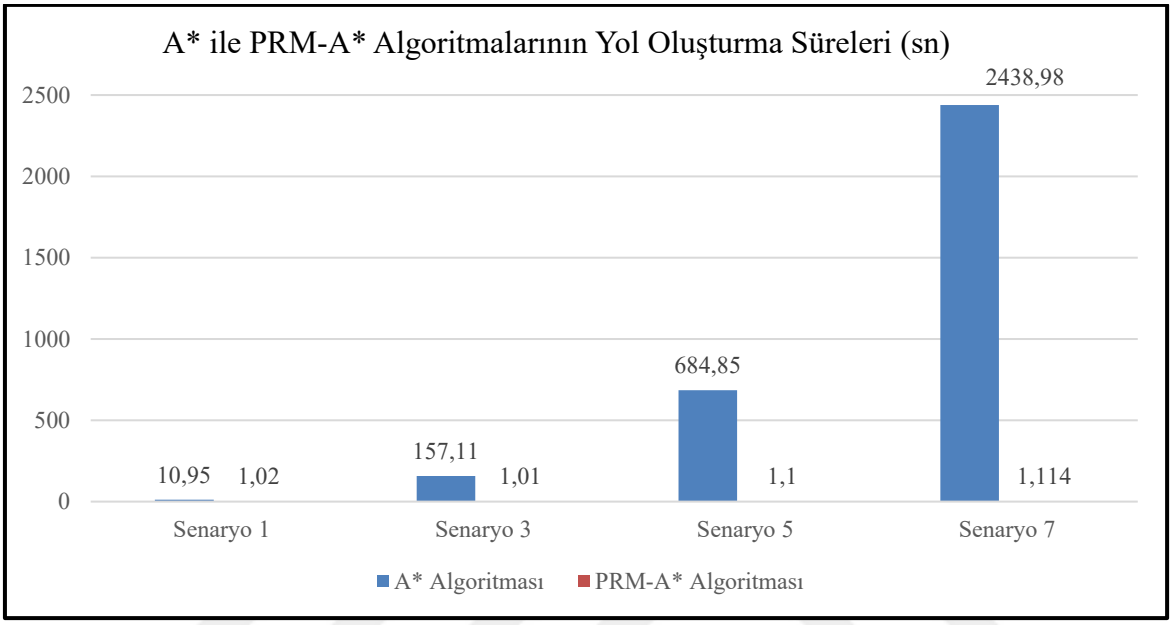
Senaryo-11	Nokta Sayısı	Deneme	Yol Uzunluğu	Süre (sn)	Ortalama Yol	Ortalama Süre (sn)
PRM-A*	40	1	140,92	2,24	132,2	2,34
		2	132,46	2,39	132,2	2,34
		3	-	-	-	-
		4	123,24	2,38	132,2	2,34
		5	-	-	-	-
	80	1	143,5	4,9	135,452	4,848
		2	132,51	5,14	135,452	4,848
		3	132,83	4,67	135,452	4,848
		4	122,39	4,56	135,452	4,848
		5	146,03	4,97	135,452	4,848
	120	1	131,37	8,9	133,312	9,664
		2	124,63	9,41	133,312	9,664
		3	135,94	9,64	133,312	9,664
		4	139,47	9,87	133,312	9,664
		5	135,15	10,5	133,312	9,664
	160	1	132,31	22,62	133,574	21,778
		2	127,18	21,27	133,574	21,778
		3	127	22,37	133,574	21,778
		4	150,47	23,1	133,574	21,778
		5	130,91	19,53	133,574	21,778

Çizelge 5.11'de görüldüğü üzere PRM-A* algoritması 40 nokta ataması ile yol oluşturmayı denediğinde 5 denemeden sadece 3'ünde yol oluşturabilmiş ve oluşturduğu yolların ortalama uzunluğu 132,2 birim iken yol oluşturma süresi ortalama 2,34 sn sürmüştür. 80 adet nokta ataması yaptığı durumda her denemesinde yol oluşturabilmiş ve oluşturduğu yolların ortalama uzunluğu 135,452 birim iken yol oluşturma süresi ortalama 4,848 sn sürmüştür. 120 adet nokta ataması yaptığı durumda da her denemesinde yol oluşturabilmiş ve oluşturduğu yolların ortalama uzunluğu 133,312 birim iken yol oluşturma süresi ortalama 9,664 sn sürmüştür. 160 adet nokta ataması yaptığı durumda da her denemesinde yol oluşturabilmiş ve oluşturduğu yolların ortalama uzunluğu 133,574 birim iken yol oluşturma süresi ortalama 21,778 sn sürmüştür.

Sonuçlardan görüldüğü üzere PRM-A* algoritmasında nokta atama sayısı arttıkça algoritmanın yol oluşturma süresi artmaktadır. Bu harita da A* algoritmasının yol oluşturma süresi 6,7 saniyedir (Bkz. Çizelge 5.9). PRM-A* algoritması A* algoritmasına göre çok daha hızlı bir algoritma iken gereğinden fazla nokta ataması PRM-A* algoritmasının performansını olumsuz yönde etkilemiştir. Oluşturulan yolların uzunluğuna bakıldığında her nokta atamasında oluşturulan yolların uzunlukları birbirine yakın çıkmıştır. PRM-A* algoritması olasılıksal bir yöntem olduğu için oluşan bu küçük farklar önemsizdir.

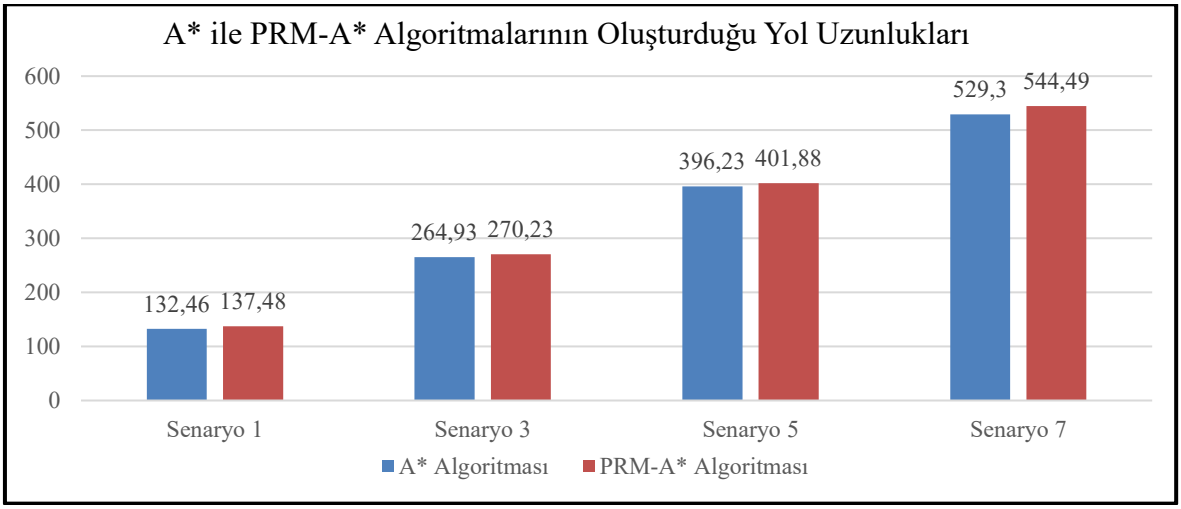
5.12. Senaryoların Bar Grafikleri ile Karşılaştırılması

Şekil 5.31'de A* ile PRM-A* algoritmalarının Senaryo 1-3-5-7 haritalarındaki yol oluşturma sürelerinin karşılaştırıldığı bir bar grafiği verilmiştir.



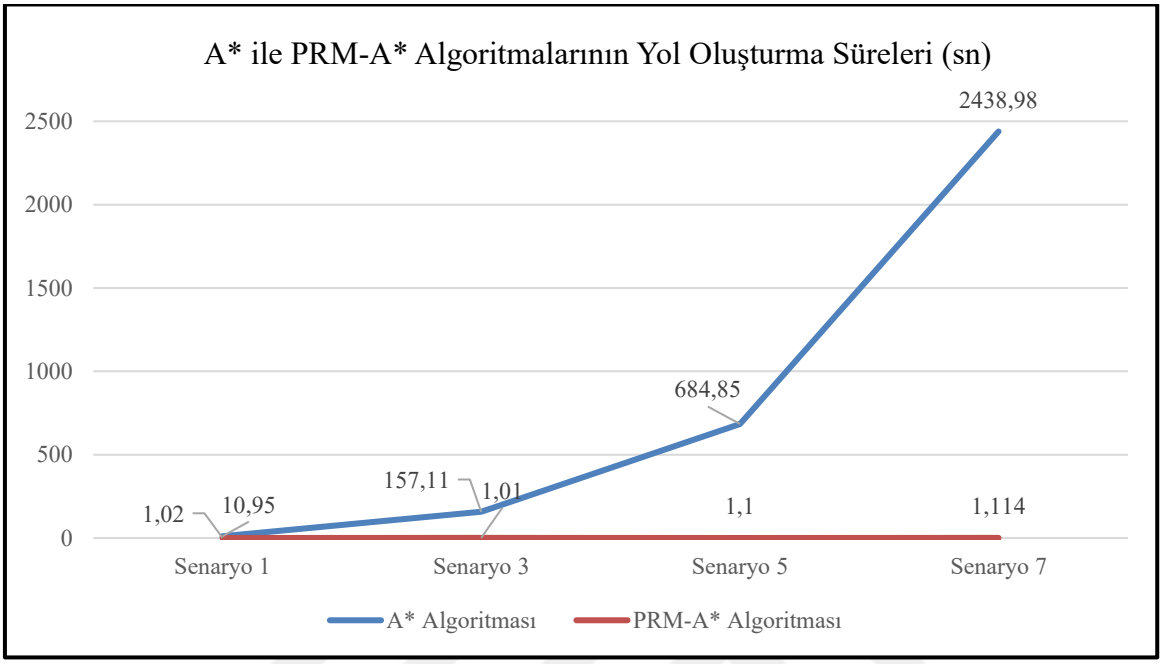
Şekil 5.31. A* ve PRM-A* algoritmalarının "T" isimli haritalarda yol oluşturma süreleri bar grafiği

Şekil 5.32'de A* ile PRM-A* algoritmalarının Senaryo 1-3-5-7 haritalarındaki oluşturduğu yol uzunluklarının karşılaştırıldığı bir bar grafiği verilmiştir.



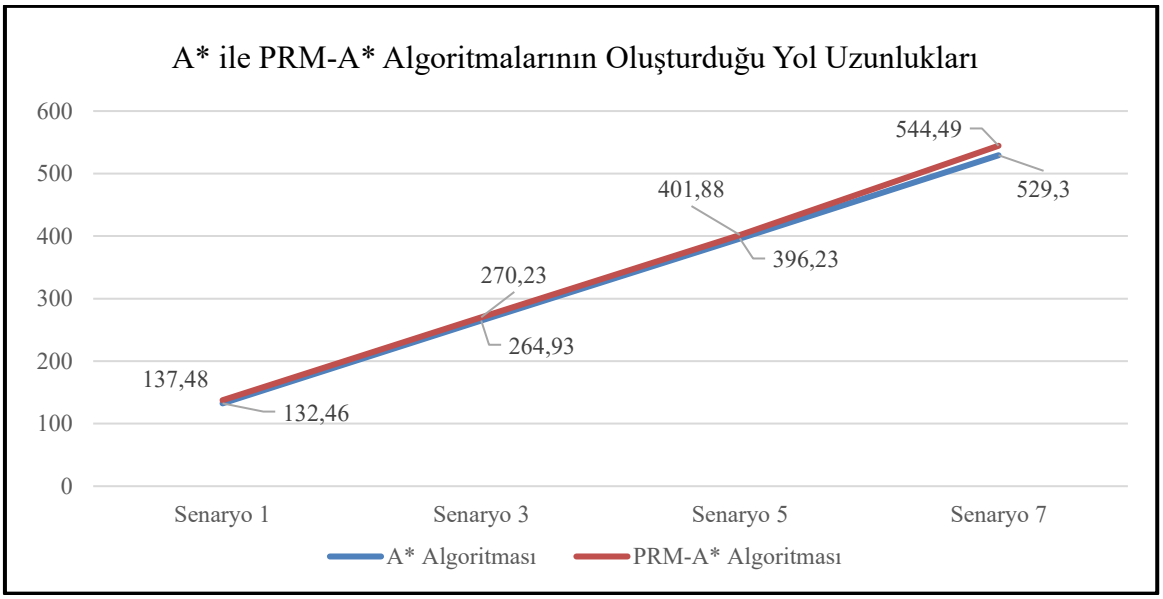
Şekil 5.32. A* ve PRM-A* algoritmalarının "T" isimli haritalarda oluşturduğu yolların uzunlukları bar grafiği

Şekil 5.33'te A* ile PRM-A* algoritmalarının Senaryo 1-3-5-7 haritalarındaki yol oluşturma sürelerinin karşılaştırıldığı bir çizgi grafiği verilmiştir.



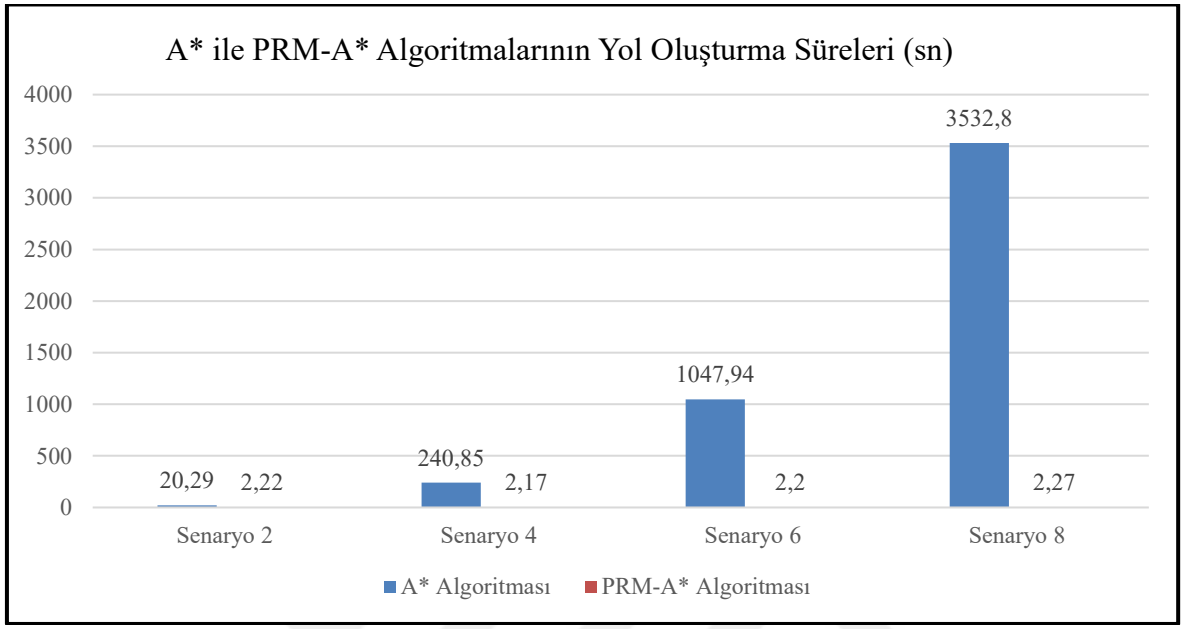
Şekil 5.33. A* ve PRM-A* algoritmalarının "T" isimli haritalarda yol oluşturma süreleri çizgi grafiği

Şekil 5.34'te A* ile PRM-A* algoritmalarının Senaryo 1-3-5-7 haritalarındaki oluşturduğu yol uzunluklarının karşılaştırıldığı bir çizgi grafiği verilmiştir.



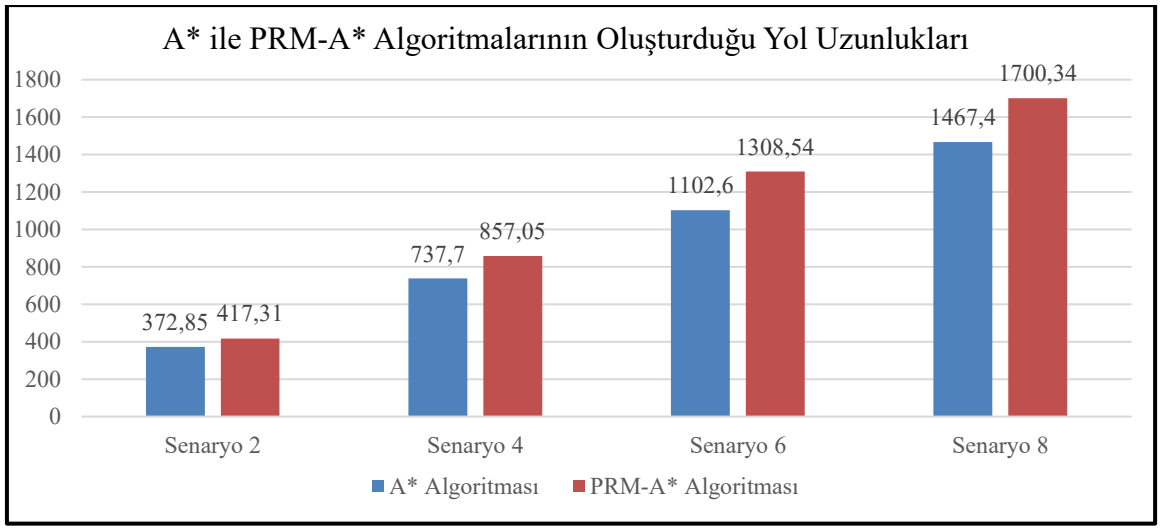
Şekil 5.34. A* ve PRM-A* algoritmalarının "T" isimli haritalarda oluşturduğu yolların uzunlukları çizgi grafiği

Şekil 5.35'te A* ile PRM-A* algoritmalarının Senaryo 2-4-6-8 haritalarındaki yol oluşturma sürelerinin karşılaştırıldığı bir bar grafiği verilmiştir.



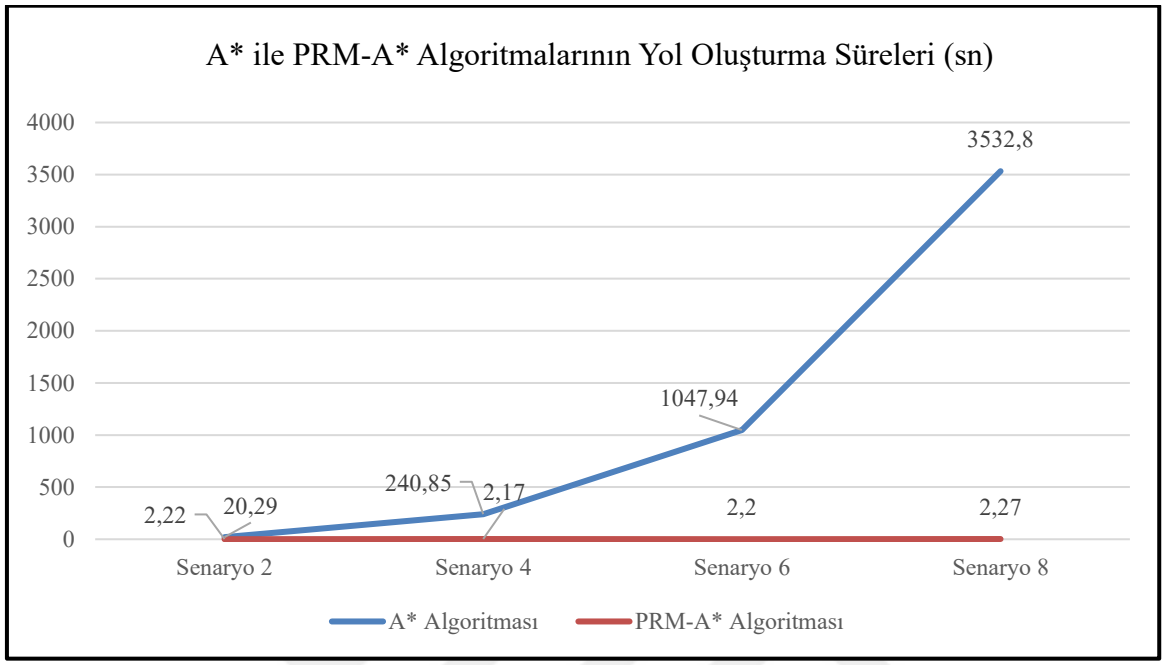
Şekil 5.35. A* ve PRM-A* algoritmalarının "Back and Forth" isimli haritalarda yol oluşturma süreleri bar grafiği

Şekil 5.36'da A* ile PRM-A* algoritmalarının Senaryo 2-4-6-8 haritalarındaki oluşturduğu yol uzunluklarının karşılaştırıldığı bir bar grafiği verilmiştir.



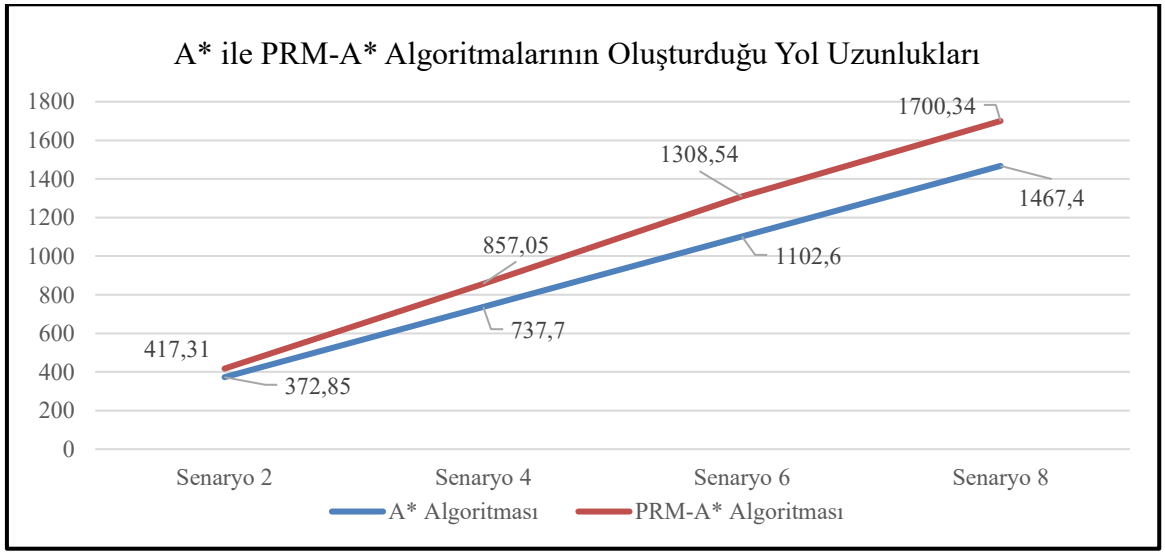
Şekil 5.36. A* ve PRM-A* algoritmalarının "Back and Forth" isimli haritalarda oluşturduğu yolların uzunlukları bar grafiği

Şekil 5.37'de A* ile PRM-A* algoritmalarının Senaryo 2-4-6-8 haritalarındaki yol oluşturma sürelerinin karşılaştırıldığı bir çizgi grafiği verilmiştir.



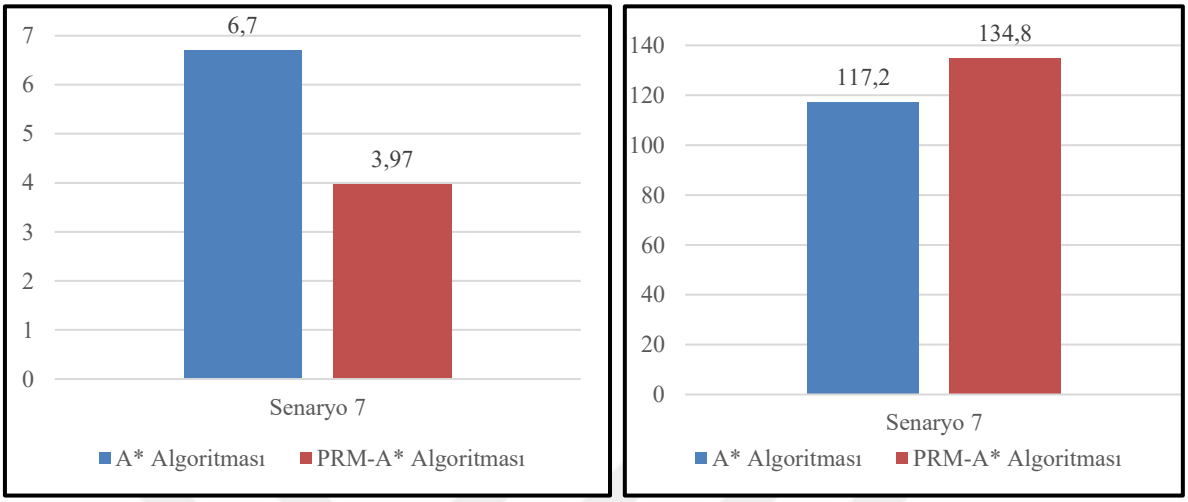
Şekil 5.37. A* ve PRM-A* algoritmalarının "Back and Forth" isimli haritalarda yol oluşturma süreleri çizgi grafiği

Şekil 5.38'de A* ile PRM-A* algoritmalarının Senaryo 2-4-6-8 haritalarındaki oluşturduğu yol uzunluklarının karşılaştırıldığı bir çizgi grafiği verilmiştir.



Şekil 5.38. A* ve PRM-A* algoritmalarının "Back and Forth" isimli haritalarda oluşturduğu yolların uzunlukları çizgi grafiği

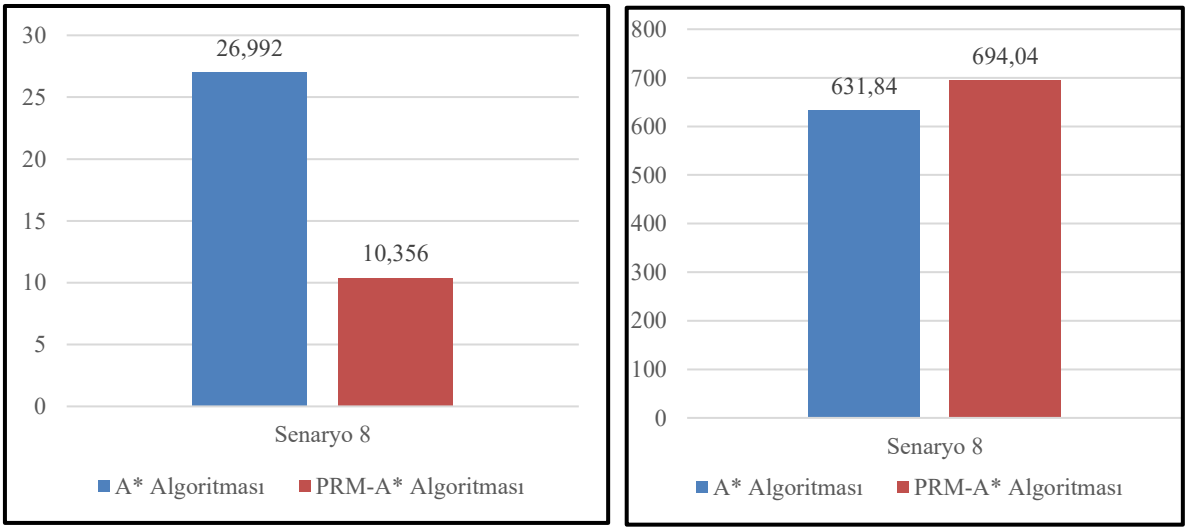
Şekil 5.39'da A* ile PRM-A* algoritmalarının Senaryo-7 haritasındaki performanslarının karşılaştırıldığı bir grafik verilmiştir.



a b

Şekil 5.39. A* ile PRM-A* algoritmalarının senaryo-7 haritasındaki performanslarının grafiği (a: A* ile PRM-A* algoritmalarının yol oluşturma süreleri (sn), b: A* ile PRM-A* algoritmalarının oluşturduğu yolların uzunlukları)

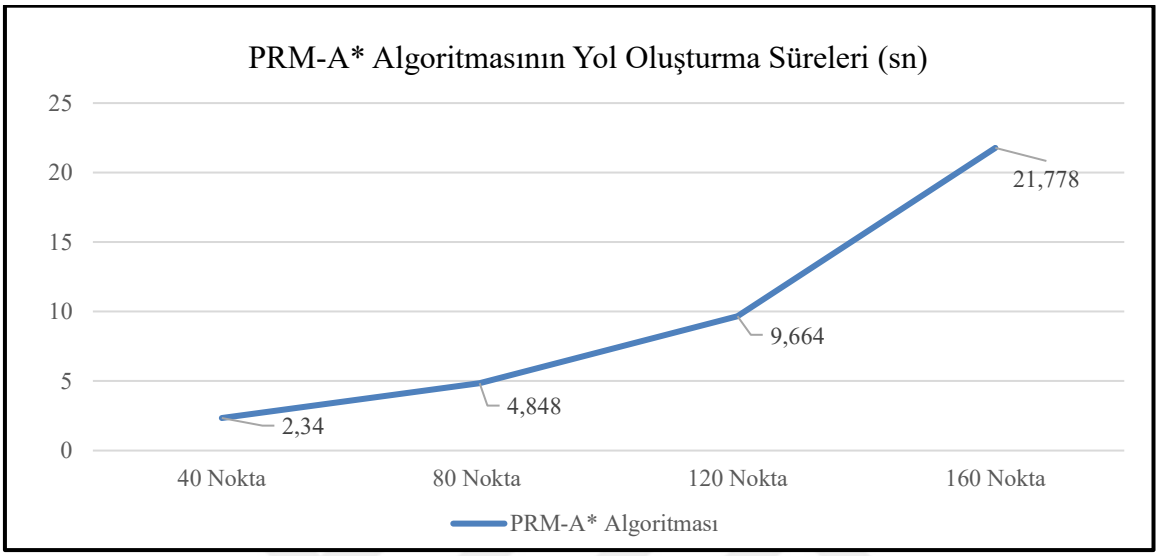
Şekil 5.40'ta A* ile PRM-A* algoritmalarının Senaryo-8 haritasındaki performanslarının karşılaştırıldığı bir grafik verilmiştir.



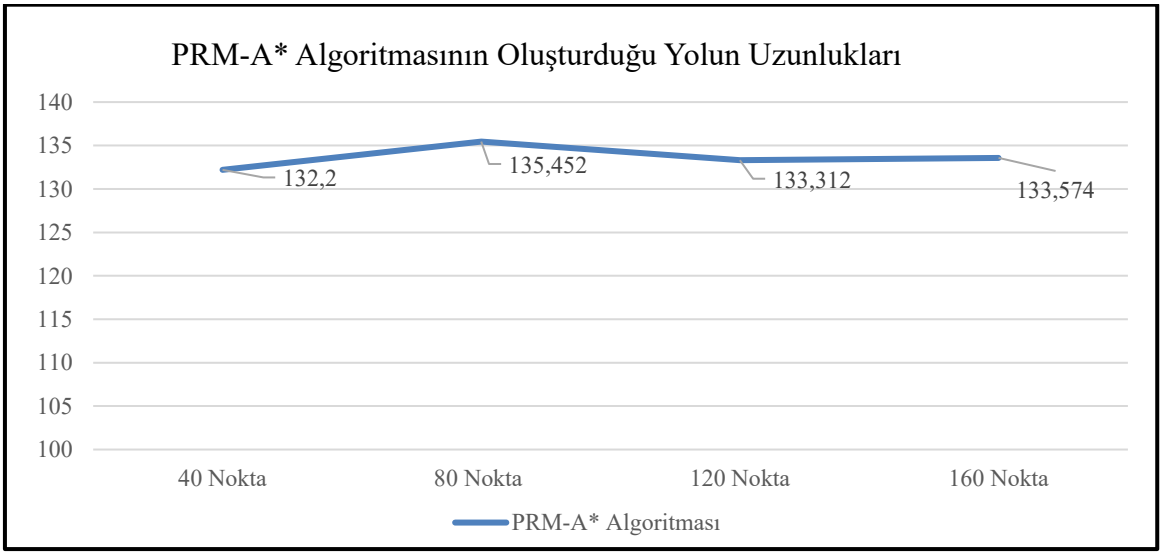
a b

Şekil 5.40. A* ile PRM-A* algoritmalarının senaryo-8 haritasındaki performanslarının grafiği (a: A* ile PRM-A* algoritmalarının yol oluşturma süreleri (sn), b: A* ile PRM-A* algoritmalarının oluşturduğu yolların uzunlukları)

Senaryo-11 haritasında PRM-A* algoritmasının farklı nokta atamalarında yol oluşturma süreleri Şekil 5.41'de, oluşturduğu yolun uzunluğu ise Şekil 5.42'de çizgi grafiği şeklinde verilmiştir.



Şekil 5.41. PRM-A* algoritmasının farklı nokta atamalarında yol oluşturma sürelerinin çizgi grafiği



Şekil 5.42. PRM-A* algoritmasının farklı nokta atamalarında oluşturduğu yolun uzunluklarının çizgi grafiği

6. SONUÇLAR VE ÖNERİLER

Bu tez çalışmasında günümüzde sıklıkla kullanılan ve kullanımı sürekli artan otonom araçlar ve kontrol algoritmaları incelenmiştir. İlk ortaya çıkan algoritmalardan (klasik algoritmalar) başlanılıp en kısa yol bulma algoritmalarının ihtiyaca göre gelişimi kısaca incelenmiştir. Her teknikten bazı algoritmalar kısaca anlatılmıştır. Sonraki bölümlerde tezin ana konusu olan 2 algoritma (A* ve PRM algoritmaları) detaylı bir şekilde incelenmiştir. PRM algoritmasının son bölümünde oluşan haritadan yol bulma işlemini A* algoritması yaptığı için bu algoritma PRM-A* algoritması olarak tabir edilmiştir.

Tezin ana konusu olarak 5. bölümde A* ve PRM-A* algoritmaları çeşitli haritalar üzerinde karşılaştırılmıştır. A* algoritması çok stabil çalışan ve verimli bir algoritmadır. Fakat büyük haritalarda yol oluşturma süreleri çok uzun olmaktadır. Bundan dolayı bu problemin çözümü için PRM algoritmasından yararlanılmaya çalışılmıştır. PRM algoritması çok büyük haritalarda dahi çok kısa sürelerde yol bulabilmektedir. Bu nedenle PRM ile A* algoritması hibrit edilmiştir.

Bu sonuçların test edilebilmesi için 5. bölümde 10 farklı harita üzerinde denemeler yapılmıştır. Farklı senaryolar denenebilmesi için bazı haritalar 100x100 birim, bazıları 200x200 birim, bazıları 300x300 birim, bazıları ise 400x400 birim olarak oluşturulmuştur. Öncelikli olarak senaryo-1'de çok karmaşık olmayan ve orta büyüklükte bir harita kullanılmıştır. Bu testin sonuçlarında PRM-A* algoritması A* algoritmasına göre 10,74 kat daha hızlı sonuç üretmiştir. Buna rağmen A* algoritmasına göre sadece %3,8 daha uzun yol oluşturmuştur.

Senaryo-2 haritası olarak karmaşık haritalarda algoritmaların verimliliğini ölçmek için yine 100x100 birim boyutlarında fakat biraz daha engel yoğunluğunun fazla olduğu bir harita seçilmiştir. Bu testin sonucunda PRM-A* algoritması A* algoritmasına göre 9,14 kat daha hızlı sonuç üretmiştir fakat A* algoritmasına göre %12 daha uzun yol oluşturmuştur. Sonuçlar da görüldüğü üzere haritanın biraz karmaşıklaştığı nokta da PRM-A* algoritmasının verimi azalmıştır. Fakat yine de çok daha kısa sürelerde yol oluşturabilmiştir.

Senaryo-3 ve senaryo-4'te ortamın büyüdüğü haritalarda algoritmaların nasıl bir sonuç vereceğinin testi için senaryo-1 ve senaryo-2 haritalarının 200x200 boyutlusu kullanılmıştır. Senaryo-3'te PRM-A* algoritması A* algoritmasına göre 155,5 kat ve senaryo-4'te ise 111 kat daha hızlı sonuç üretmiştir. Buna karşın senaryo-3'te PRM-A* algoritması A* algoritmasına göre %2 ve senaryo-4'te ise %16,2 daha uzun yol oluşturmuştur. Sonuçlar da görüldüğü üzere haritanın boyu büyüdükçe A* algoritmasının yol oluşturma süresi üstel bir şekilde artmaktadır. Haritaların boyutları 4 kat artmasına rağmen A* algoritmasının yol oluşturma süresi iki harita içinde 10 katından fazla artmıştır. Buna rağmen PRM-A* algoritmasında iki harita için de bir değişiklik olmamıştır. Sonuçlar da görüleceği üzere PRM-A* algoritması harita boyutunun büyüklüğünden etkilenmemektedir.

Aynı şekilde Senaryo-5 ve senaryo-6'da da ortamın büyüdüğü haritalarda algoritmaların nasıl bir sonuç vereceğinin testi için senaryo-1 ve senaryo-2 haritalarının 300x300 boyutlusu kullanılmıştır. Senaryo-5'te PRM-A* algoritması A* algoritmasına göre 622,6 kat ve senaryo-4'te ise 476,3 kat daha hızlı sonuç üretmiştir. Buna karşın senaryo-5'te PRM-A* algoritması A* algoritmasına göre %1,42 ve senaryo-6'da ise %18,68 daha uzun yol oluşturmuştur. Sonuçlar da görüldüğü üzere haritanın boyu büyüdükçe A* algoritmasının yol oluşturma süresi üstel bir şekilde artmaktadır. Haritaların boyutları 9 kat artmasına rağmen A* algoritmasının yol oluşturma süresi iki harita içinde 50 katından fazla artmıştır. Buna rağmen PRM-A* algoritmasında iki harita için de bir değişiklik olmamıştır. Sonuçlar da görüleceği üzere PRM-A* algoritması harita boyutunun büyüklüğünden etkilenmemektedir.

Yine aynı şekilde Senaryo-7 ve senaryo-8'de de ortamın büyüdüğü haritalarda algoritmaların nasıl bir sonuç vereceğinin testi için senaryo-1 ve senaryo-2 haritalarının 400x400 boyutlusu kullanılmıştır. Senaryo-7'de PRM-A* algoritması A* algoritmasına göre 2189,4 kat ve senaryo-8'de ise 1556,3 kat daha hızlı sonuç üretmiştir. Buna karşın senaryo-7'de PRM-A* algoritması A* algoritmasına göre %2,8 ve senaryo-8'de ise %15,87 daha uzun yol oluşturmuştur. Sonuçlar da görüldüğü üzere haritanın boyu büyüdükçe A* algoritmasının yol oluşturma süresi üstel bir şekilde artmaktadır. Haritaların boyutları 16 kat artmasına rağmen A* algoritmasının yol oluşturma süresi iki harita içinde 170 katından fazla artmıştır. Buna rağmen PRM-A* algoritmasında iki harita için de bir değişiklik olmamıştır. Sonuçlar da görüleceği üzere PRM-A* algoritması harita boyutunun büyüklüğünden etkilenmemektedir.

Senaryo-9 ve senaryo-10'da karmaşık ve dar geçitlerin olduğu haritalarda algoritmaların nasıl bir sonuç vereceğini test etmek için 2 farklı karmaşık harita kullanılmıştır. Senaryo-9'da PRM-A* algoritması A* algoritmasına göre 1,68 kat ve senaryo-10'da ise 2,6 kat daha hızlı sonuç üretmiştir. Buna karşın PRM-A* algoritması A* algoritmasına göre senaryo-9'da %15 ve senaryo-10'da ise %9,85 daha uzun yol oluşturmuştur. Sonuçlar da görüldüğü üzere PRM-A* algoritmasının verimi karmaşık ve dar geçitlerin olduğu haritalarda aşırı miktarda düşmektedir.

Senaryo-11'de PRM-A* algoritmasının nokta atama sayısının algoritmanın performansını nasıl etkileyeceğini görmek için senaryo-9'daki harita da 40, 80, 120 ve 160 adet nokta ataması yapılarak yol oluşturulmuştur. Atanan nokta sayısı arttıkça PRM-A* algoritmasının yol oluşturma süresinin lineerin üzerinde bir artış gösterdiği görülmüştür. PRM-A* algoritması hızlı bir algoritmadır. Fakat sonuçlardan da görüldüğü üzere PRM-A* algoritmasının nokta atama sayısının gereksiz bir şekilde arttırılması PRM-A* algoritmasının verimini aşırı miktarda düşürmektedir.

Sonuç olarak A* algoritması PRM algoritması ile beraber kullanıldığında çok daha kısa sürelerde rota oluşturabilmiştir. Özellikle dinamik ortamlarda ve gerçek zamanlı uygulamalarda süre hayati bir öneme sahiptir. Bu özelliğinden dolayı PRM-A* algoritması dinamik ortamlarda ve gerçek zamanlı uygulamalarda A* algoritmasına göre çok daha verimli bir şekilde kullanılabilmektedir.

Ancak PRM-A* algoritmasının bazı dezavantajları vardır. Örnek olarak karmaşık haritalarda verimi düşmektedir. Ayrıca, oluşturulan yol keskin manevralara sahiptir ve her çalışmasında farklı rotalar oluşturmaktadır. Çalışmanın devamı olarak PRM-A* algoritmasının bu dezavantajlarını iyileştirecek çalışmalar yapılabilir. Böylece PRM-A* algoritması gerçek zamanlı uygulamalarda da çok daha verimli bir şekilde çalışabilecek hale gelebilecektir.

KAYNAKLAR

1. Shwail, S. H. and Karim, A. (2014). Probabilistic Roadmap, A* and GA for Proposed Decoupled Multi-Robot Path Planning, *Iraqi Journal of Applied Physics*, 10(2), 3-9.

2. Santiago, R. M. C., Ocampo, A. L. D., Ubando, A. T., Bandala, A. A. and Dadios, E. P. (2017). Path Planning for Mobile Robots Using Genetic Algorithm and Probabilistic Roadmap, *IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, 1-5.

3. Alpkıray, N. (2019). *Otonom Keşif Amaçlı Robot Sistemleri İçin Geri Dönüş Rotası Hesaplama Algoritması Geliştirilmesi*, Yüksek Lisans Tezi, Sivas Cumhuriyet Üniversitesi Fen Bilimleri Enstitüsü, Sivas.

4. Dere, M. E. (2019). *Mobil Robotlar İçin Optimum Yol Bulma*, Yüksek Lisans Tezi, Konya Teknik Üniversitesi Fen Bilimleri Enstitüsü, Konya.

5. Wang, H., Zhou, J., Zheng, G. and Liang, Y. (2014). HAS:Hierarchical A-Star Algorithm for Big Map Navigation in Special Areas, *5th International Conference on Digital Home*, 222-225.

6. ElHalawany, B. M., Abdel-Kader, H. M., TagEldeen, A., Elsayed, A. E. and Nossair Z. B. (2013). Modified A* Algorithm for Safer Mobile Robot Navigation, *5th International Conference on Modelling, Identification and Control (ICMIC)*, 74-78.

7. Yuan, F., Liang, J., Fu, Y., Xu, H. and Ma, K. (2015). A Hybrid Sampling Strategy With Optimized Probabilistic Roadmap Method, *12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2298-2302.

8. Lafcı, M. (2016). *Dinamik Engellerin Bulunduğu Ortamda Gezgin Robot İçin Hareket Planlama*, Yüksek Lisans Tezi, Karabük Üniversitesi Fen Bilimleri Enstitüsü, Karabük.

9. Park, J. and Huh, U. (2016). Path Planning for Autonomous Mobile Robot Based on Safe Space, *Journal of Electrical Engineering and Technology*, 11(5), 1441-1448.

10. Yetkin, S. (2016). *Bir Robot Balık İçin İki Boyutlu Yol Planlama Algoritmalarının Uygulanması,* Yüksek Lisans Tezi, Fırat Üniversitesi Fen Bilimleri Enstitüsü, Elâzığ.

11. Güllü, A. (2017). *Labirentlerde Yapay Zekâ Tabanlı Yön Bulma Algoritmaları Kullanan Bir Gezgin Robot Geliştirilmesi*, Doktora Tezi, Trakya Üniversitesi Fen Bilimleri Enstitüsü, Edirne.

12. Garip, Z. B., Atali, G., Karayel, D. and Ozkan, S. S. (2018). Path Planning for Multiple Mobile Robots in Static Environment using Hybrid Algorithm, *2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, 1-4.

13. Wang, Z. and Cai, J. (2018). Probabilistic Roadmap Method for Path-Planning in Radioactive Environment of Nuclear Facilities, *Progress in Nuclear Energy*, 109, 113-120.

14. Chowdhury, M. I. and Schwartz, D. G. (2020). UUV On-Board Path Replanning Using PRM-A*, *Global Oceans 2020: Singapore – U.S. Gulf Coast*, 1-8.

15. Hopkins, J., Joy, F., Sheta, A., Turabieh, H. and Kar, D. (2020). Path Planning for Indoor UAV Using A* and Late Acceptance Hill Climbing Algorithms Utilizing Probabilistic Roadmap, *International Journal of Engineering & Technology*, 9(4), 857-862.

16. Mokwele, D., Du, S. and Benade, J. (2020). Comparative Study of Autonomous Path Planning Methods for Mobile Robot, *2020 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*, 1-6.

17. Madridano, A, Al-Kaff, A., Flores, P., Martin, D. and Escalera, A. D. L. (2020). Obstacle Avoidance Manager for UAVs Swarm*, *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, 815-821.

18. Roa-Borbolla, A. G., Jimnez-Nieto, Y. A., Espinosa, G. I. M., Villa-Paniagua, J. L., Ruiz-Martinez, M. and Vega-Valera, R. M. (2020). Algorithm Comparison between A* and PRM on Indoor Fire Simulation, *2020 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE)*, 23-28.

19. Dias, E. V. A., Silva, C. G. B. P., Batista, J. G., Souza, D., Silva, J. L. N., Ramalho, G. L. B and Lima, M. A. F. B. (2021). Trajectory Planning of a SCARA Manipulator Using PRM for Collision Avoidance, *15th Brazilian Symposium on Intelligent Automation*, 215-220.

20. Liu, X., Yao, L. and Dong, H. (2021). Path planning for mobile robotic arms incorporating adaptive gravitational fields, *2021 International Conference on Robotics and Control Engineering*, 13-18.

21. Yolal, E. (2015). *Mobil Robot Simülatörleri ve İleri Seviyeli Simülasyonlar,* Yüksek Lisans Tezi, İstanbul Teknik Üniversitesi Fen Bilimleri Enstitüsü, İstanbul.

22. Gürgüze, G. ve Türkoğlu, İ. (2018). Kullanım Alanlarına Göre Robot Sistemlerinin Sınıflandırılması, *Fırat Üniversitesi Mühendislik Bilimleri Dergisi,* 31(1), 53-66.

23. İnternet: Endüstriyel Robot Nedir? Hangi Görevlerde Kullanılır? , https://www.entes.com.tr/endustriyel-robot-nedir-hangi-gorevlerde-kullanilir/, Son Erişim Tarihi: 24.04.2022.

24. İnternet: Delta Robot Nedir? Delta Robot Nerelerde Kullanılır? , https://deltarobot.wordpress.com/2015/06/25/delta-robot-nedir-delta-robot-nerelerde-kullanilir/, Son Erişim Tarihi: 24.04.2022.

25. İnternet: KUKA KR 240, https://www.robots.com/robots/kuka-kr-240, Son Erişim Tarihi: 24.04.2022.

26. Corke, P. (2017). Robotics, Vision and Control (Second Edition). Berlin: Springer, 99-100.

27. İnternet: NASA'nın uzay aracı Perseverance Mars'a iniş yaptı, https://www.dw.com/tr/nasan%C4%B1n-uzay-arac%C4%B1-perseverance-marsa-ini%C5%9F-yapt%C4%B1/a-56622044, Son Erişim Tarihi: 24.04.2022.

28. İnternet: Tesla Model S, https://en.wikipedia.org/wiki/Tesla_Model_S, Son Erişim Tarihi: 24.04.2022.

29. Park, I. W., Kim, J. Y., Lee, J. and Oh, J. H. (2005). Mechanical Design of Humanoid Robot Platform KHR-3 (KAIST Humanoid Robot – 3: HUBO), *5th IEEE-RAS International Conference on Humanoid Robots*, 321-326.

30. İnternet: Sabancı Üniversitesi Mekatronik Mühendislerinin Büyük Başarısı! Türkiye'nin İlk İnsansı Robotu: "SURALP", https://gazetesu.sabanciuniv.edu/suralp, Son Erişim Tarihi: 24.04.2022.

31. Corke, P. (2017). Robotics, Vision and Control (Second Edition). Berlin: Springer, 114-115.

32. İnternet: Bayraktar TB2, https://www.baykarsavunma.com/iha-15.html, Son Erişim Tarihi: 24.04.2022.

33. İnternet: Kargu, https://www.stm.com.tr/tr/cozumlerimiz/otonom-sistemler/kargu, Son Erişim Tarihi: 24.04.2022.

34. Canlı, G. A., Kurtoğlu, İ., Canlı, M. O. ve Tuna, Ö. S. (2015). Dünyada ve Ülkemizde İnsansız Sualtı Araçları İSAA-AUV&ROV Tasarım ve Uygulamaları, *GİDB Dergi*, 4, 43-75.

35. Akyol, S. ve Alataş, B. (2012). Güncel Sürü Zekâsı Optimizasyon Algoritmaları, *Nevşehir Üniversitesi Fen Bilimleri Enstitüsü Dergisi*, 1 (1), 36-50.

36. Balanji, H. M., Yılmaz, E., Çakmak, Ö. ve Turgut, A. E. (2017). Sürü Robotları için Görüntü Tabanlı Mesafe, Açı ve Yönelim Sistemi Tasarımı, *Otomatik Kontrol Ulusal Toplantısı*, 386-391.

37. Seçkin, A. Ç., Karpuz, C. ve Özek, A. (2016). Sürü Robotiği, *International Conference on Computer Science and Engineering*, 414-419.

38. Atalı, G. (2018). *Dağıtık Mobil Robotlar İçin Yeni Bir Otonom Yol Planlama ve Engel Tespit Sisteminin Tasarımı*, Doktora Tezi, Sakarya Üniversitesi Fen Bilimleri Enstitüsü, Sakarya.

39. İnternet: ROS Nedir? (Robot Operating System), http://yapbenzet.kocaeli.edu.tr/ros-nedir-robot-operating-system/, Son Erişim Tarihi: 24.04.2022.

40. İnternet: MATLAB, https://tr.wikipedia.org/wiki/MATLAB, Son Erişim Tarihi: 24.04.2022.

41. İnternet: MATLAB Nedir? , https://halilozel1903.medium.com/matlab-nedir-91a904a74f45, Son Erişim Tarihi: 24.04.2022.

42. İnternet: Gazebo, https://gazebosim.org/home, Son Erişim Tarihi: 24.04.2022.

43. İnternet: Gazebo Simulator, https://en.wikipedia.org/wiki/Gazebo_simulator, Son Erişim Tarihi: 24.04.2022.

44. İnternet: Webots, https://en.wikipedia.org/wiki/Webots, Son Erişim Tarihi: 24.04.2022.

45. Soydaş, Ş. (2015). *Lityum Tabanlı Batarya Paketleri için Batarya Yönetim Sistemi Tasarımı,* Yüksek Lisans Tezi, Karabük Üniversitesi Fen Bilimleri Enstitüsü, Karabük.

46. Mnubi, S. A., (2016). Motion Planning and Trajectory for Wheeled Mobile Robot, *International Journal of Science and Research*, 5(1), 1064-1068.

47. Kunchev, V., Jain, L., Ivancevic, V. and Finn, A. (2006). Path Planning and Obstacle Avoidance for Autonomous Mobile Robots: A Review, *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, 537-544.

48. Garip, Z. (2018). *Mobil Robotların Yol Planlaması için Meta-Sezgisel Hibrit Algoritmalar Geliştirilmesi ve Uygulanması*, Doktora Tezi, Sakarya Üniversitesi Fen Bilimleri Enstitüsü, Sakarya.

49. Buniyamin, N., Sariff, N., Wan Ngah W. A. J. and Mohamad, Z. (2011). Robot Global Path Planning Overview and a Variation of Ant Colony System Algorithm, *International Journal of Mathematics and Computers in Simulation*, 5(1), 9-16.

50. Wahab, M. N. A., Nefti-Meziani, S. and Atyabi, A., (2020). A Comparative Review on Mobile Robot Path Planning: Classical or Meta-Heuristic Methods? , *Annual Reviews in Control*, 50, 233-252.

51. Atyabi, A. and Power, D. M. W. (2013). Review of Classical and Heuristic-Based Navigation and Path Planning Approaches, *International Journal of Advancements in Computing Technology*, 5, 1-14.

52. Zafar, M. N. and Mohanta, J. C. (2018). Methodology for Path Planning and Optimization of Mobile Robots: A Review, *International Conference on Robotics and Smart Manufacturing*, 133, 141-152.

53. Abbadi, A. and Prenosil, V. (2015), Safe Path Planning Using Cell Decomposition Approximation, *International Conference Distance Learning, Simulation and Communication*, 8-14.

54. Zhang H., Lin, W. and Chen, A. (2018). Path Planning for the Mobile Robot: A Review, *Symmetry,* 10(10), 450.

55. Patle, B. K., Loganathan, G. B., Pandey, A. and Parhi, D. R. K. (2019). A Review: On Path Planning Strategies for Navigation of Mobile Robot, *Defence Technology*, 15(4), 582-606.

56. Suvaydan, F. (2011). *Mobil Robotlar İçin Yol Planlama Problemi ve Karınca Kolonisi Algoritması ile Yol Planlama Problemlerinin Optimal Çözümü,* Yüksek Lisans Tezi, Düzce Üniversitesi Fen Bilimleri Enstitüsü, Düzce.

57. Dijkstra, E. W. (1959). A Note on Two Problems in Connexion With Graphs, *Numerische Mathematik*, 269-271.

58. Tuncer, A. (2013). *Otonom Araçlar İçin Yol Bulma Probleminin Genetik Algoritmalar ve FPGA ile Çözümü ve Gerçekleştirilmesi,* Doktora Tezi, Kocaeli Üniversitesi Fen Bilimleri Enstitüsü, Kocaeli.

59. Dhurkefl, E. J. D. (2019). *Üç Boyutlu Ortamda En Kısa Yol Planlama Uygulamaları*, Yüksek Lisans Tezi, Selçuk Üniversitesi Fen Bilimleri Enstitüsü, Konya.

60. Dechter, R. and Pearl, J. (1985). Generalized Best-First Search Strategies and the Optimality of A*, *Journal of the Association for Computing Machinery,* 32(3), 505-536.

61. Hart, P. E., Nilsson, N. J. and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100-107.

62. LaValle, S. M. (1998). Rapidly-Exploring Random Trees: A New Tool for Path Planning, *The Annual Research Report,* 4.

63. LaValle, S. M. and Kuffner, J. J. (2001). Randomized Kinodynamic Planning, *IEEE International Conference on Robotics and Automation,* 20(5), 378-400.

64. Karaman, S. and Frazzoli, E. (2011). Sampling-Based Algorithms for Optimal Motion Planning, *The International Journal of Robotics Research,* 30(7), 846-894.

65. Kavraki, L. E., Svestka, P., Latombe, J. C. and Overmars, M. H. (1996). Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces, *IEEE Transactions on Robotics and Automation,* 12(4), 566-580.

66. Kennedy, J. and Eberhart, R. (1995). Particle Swarm Optimization, *International Conference on Neural Networks,* 1942-1948.

67. Özsağlam, M. Y. and Çunkaş, M. (2008). Optimizasyon Problemlerinin Çözümü İçin Parçacık Sürü Optimizasyonu Algoritması, *Politeknik Dergisi,* 11(4), 299-305.

68. İnternet: Genetik Algoritma, https://tr.wikipedia.org/wiki/Genetik_algoritma, Son Erişim Tarihi: 24.04.2022.

69. Özden, S. (2013). *Modern Meyve (Bodur Kiraz) Yetiştiriciliğinde Güneş Enerjili Akıllı Damla Sulama Sistemi Tasarımı*, Doktora Tezi, Gazi Üniversitesi Fen Bilimleri Enstitüsü, Ankara.

70. Kerem, A. (2018). *Rüzgâr Parametrelerinin Değişiminin İzlenmesi ve Yapay Zekâ Algoritmaları Kullanılarak Tahmini,* Doktora Tezi, Gazi Üniversitesi Fen Bilimleri Enstitüsü, Ankara.

71. İnternet: Motion Planning Maps, http://imr.ciirc.cvut.cz/planning/maps.xml, Son Erişim Tarihi: 24.04.2022.

EKLER

EK-1. A* Algoritması

```
clear all

map_satir = 100; % harita boyutları
map_sutun = 100;MAP = int8(zeros(map_satir,map_sutun));
tic
MAP(20:25,20:80) = 1; % Engeller
MAP(20:80,47:52) = 1;
MAP(20:40,20:25) = 1;
MAP(20:40,75:80) = 1;

Connecting_Distance = 1; % Haritayı ekrana bas
figure(10);
imagesc(MAP);
colormap(flipud(gray));
hold on;

basla_X = 10; % başlangıç ve bitiş noktaları
basla_Y = 90;
plot(basla_Y,basla_X,'X r')
hedef_X = 90;
hedef_Y = 10;
plot(hedef_Y,hedef_X,'X r')
ylabel('Satır X','fontsize',14)
xlabel('Sütun Y','fontsize',14)
grid on

plot(basla_Y,basla_X,'. g','markersize',40);
plot(hedef_Y,hedef_X,'. g','markersize',40);

 text(basla_Y+2,basla_X-2,"basla",'fontsize',14);
 text(hedef_Y-3,hedef_X+3,"hedef",'fontsize',14);

iterasyon = 1000000;
open_list = zeros(map_satir,map_sutun);
closed_list = zeros(map_satir,map_sutun);
for n = 1:map_satir
   for m = 1:map_sutun
      if MAP(n,m) == 1
         closed_list(n,m) = 1;
      end
   end
end
for n = 1:map_satir
   for m = 1:map_sutun
      h(n,m) = sqrt(((n-hedef_X)^2)+((m-hedef_Y)^2));
   end
end
```

EK-1. (devam) A* Algoritması

```
g = zeros(map_satir,map_sutun);
g_new = zeros(map_satir,map_sutun);
f = ones(map_satir,map_sutun);
hf = zeros(map_satir*map_sutun,6);
t = 1;
for n = 1:map_satir
   for m = 1:map_sutun
      hf(t,2) = n;
      hf(t,3) = m;
      hf(t,1) = 20000;
      t = t+1;
   end
end
for m = 1:map_satir*map_sutun
   if (hf(m,2) == basla_X)&&(hf(m,3) == basla_Y)
      hf(m,1) = h(basla_X,basla_Y);
   end
end
f(basla_X,basla_Y)=h(basla_X,basla_Y);
open_list(basla_X,basla_Y) = 1

zz=0;
while zz < iterasyon
   if sum(sum(open_list)) == 0
      disp('Hedef düğüme ulaşılamadı!');
      disp('Açık Liste: boş..!!');
   else sum(sum(open_list)) >= 1
      hf = sortrows(hf,[1]);
      gecerli_X = hf(1,2);
      gecerli_Y = hf(1,3);
      open_list(gecerli_X,gecerli_Y) = 0;
      closed_list(gecerli_X,gecerli_Y) = 1;
      for m = 1:map_satir*map_sutun
         if (hf(m,2) == gecerli_X)&&(hf(m,3) == gecerli_Y)
            hf(m,1) = 20000;
         end
      end
   end
   if (gecerli_X == hedef_X)&&(gecerli_Y == hedef_Y)
      disp('Hedef düğüme ulaşıldı!');
      zz = iterasyon-1;
   else ((gecerli_X ~= hedef_X)&&(gecerli_Y ~= hedef_Y))
      tum_komsu_kapali = 0;
      for m = -1:1
         for n = -1:1
            komsu_X = gecerli_X+m;
            komsu_Y = gecerli_Y+n;
```

EK-1. (devam) A* Algoritması

```
        if(0<komsu_X)&&(0<komsu_Y)&&(komsu_X<=map_satir)&&(komsu_Y<=map
_sutun)&&(MAP(komsu_X,komsu_Y)==0)
            if (closed_list(komsu_X,komsu_Y)==0)
               tum_komsu_kapali = tum_komsu_kapali+1;
               if open_list(komsu_X,komsu_Y) == 0
                  open_list(komsu_X,komsu_Y) = 1;
                  for k = 1:map_satir*map_sutun
                     if (hf(k,2) == komsu_X)&&(hf(k,3) == komsu_Y)
                        hf(k,4) = gecerli_X;
                        hf(k,5) = gecerli_Y;
                     end
                  end
                  komsu_g(komsu_X,komsu_Y) = sqrt(((komsu_X-
gecerli_X)^2)+((komsu_Y-gecerli_Y)^2));
                  g(komsu_X,komsu_Y) =
komsu_g(komsu_X,komsu_Y)+g(gecerli_X,gecerli_Y);
                  f(komsu_X,komsu_Y) = g(komsu_X,komsu_Y)+h(komsu_X,komsu_Y);
                  for k = 1:map_satir*map_sutun
                     if (hf(k,2) == komsu_X)&&(hf(k,3) == komsu_Y)
                        hf(k,1) = f(komsu_X,komsu_Y);
                     end
                  end
               elseif open_list(komsu_X,komsu_Y) == 1
                  komsu_g(komsu_X,komsu_Y) = sqrt(((komsu_X-
gecerli_X)^2)+((komsu_Y-gecerli_Y)^2));
                  g_new(komsu_X,komsu_Y) =
komsu_g(komsu_X,komsu_Y)+g(gecerli_X,gecerli_Y);
                  if g_new(komsu_X,komsu_Y) < g(komsu_X,komsu_Y)
                     for k = 1:map_satir*map_sutun
                        if (hf(k,2) == komsu_X)&&(hf(k,3) == komsu_Y)
                           hf(k,4) = gecerli_X;
                           hf(k,5) = gecerli_Y;
                        end
                     end
                     g(komsu_X,komsu_Y) = g_new(komsu_X,komsu_Y);
                     f(komsu_X,komsu_Y) =
g(komsu_X,komsu_Y)+h(komsu_X,komsu_Y);
                     for k = 1:map_satir*map_sutun
                        if (hf(k,2) == komsu_X)&&(hf(k,3) == komsu_Y)
                           hf(k,1) = f(komsu_X,komsu_Y);
                        end
                     end
                  end
               end
            elseif tum_komsu_kapali == 0
               hf = sortrows(hf,[1]);
            end
```

EK-1. (devam) A* Algoritması

```
            end
        end
    end
  end
  zz=zz+1;
end
dd = 1;
yol(dd,1) = hedef_X;
yol(dd,2) = hedef_Y;
for k = 1:map_satir*map_sutun
  if (hf(k,2) == yol(dd,1))&&(hf(k,3) == yol(dd,2))
    yol(dd+1,1) = hf(k,4);
    yol(dd+1,2) = hf(k,5);
    dd = dd+1;
  end
end
yol_son = flipud(yol);
yol_son(1,:)=[];
adim_sayisi = length(yol_son)


for i=1:adim_sayisi-1

  p1=line([yol_son(i+1,2) yol_son(i,2)],[yol_son(i+1,1) yol_son(i,1)] ,'LineWidth',2);
  p1.Color='red';

end

toc

yol_uzunlugu=0;

for ix = 1:adim_sayisi-1

  uzunluk=sqrt(((yol_son(ix+1,1)-yol_son(ix,1))^2)+((yol_son(ix+1,2)-
yol_son(ix,2))^2));
  yol_uzunlugu=yol_uzunlugu+uzunluk;
end
```

EK-2. PRM Algoritması

```
clear all
map_satir = 100; % harita boyutları
map_sutun = 100;MAP = int8(zeros(map_satir,map_sutun));
tic
MAP(20:25,20:80) = 1; % Engeller
MAP(20:80,47:52) = 1;
MAP(20:40,20:25) = 1;
MAP(20:40,75:80) = 1;
 Connecting_Distance = 1; % Haritayı ekrana bas
figure(10);
imagesc(MAP);
colormap(flipud(gray));
hold on;
basla_X = 10; % başlangıç ve bitiş noktaları
basla_Y = 90;
plot(basla_Y,basla_X,'X r')
hedef_X = 90;
hedef_Y = 10;
plot(hedef_Y,hedef_X,'X r')
ylabel('Satır X','fontsize',14)
xlabel('Sütun Y','fontsize',14)
grid on

Random_map = int8(zeros(map_sutun,map_satir));
Random_map(basla_X,basla_Y)=1;
Random_map(hedef_X,hedef_Y)=1;
iterasyon=30;
it=0;
komsu(1,1)=basla_X;
komsu(1,2)=basla_Y;
komsu(iterasyon+2,1)=hedef_X;
komsu(iterasyon+2,2)=hedef_Y;

plot(basla_Y,basla_X,'. g','markersize',40);
plot(hedef_Y,hedef_X,'. g','markersize',40);

while it < iterasyon                                % Rastgele sayı atama bölümü
   random_X=randi(map_satir);
   random_Y=randi(map_sutun);

   if((MAP(random_X,random_Y)==0)&&(Random_map(random_X,random_Y)==0))
      Random_map(random_X,random_Y)=1;
      plot(random_Y,random_X,'. r','markersize',40);
      it = it + 1;
      komsu(it+1,1)=random_X;
      komsu(it+1,2)=random_Y;
   end
end
```

EK-2. (devam) PRM Algoritması

```
kombinasyon=((iterasyon+2)*(iterasyon+1));

komsu_yol=double(zeros(kombinasyon,4
a=2;
it_komsu=length(komsu);
b=1;
for ik = 1:it_komsu
   for jk = 1 : it_komsu
      if(ik~=jk)
      komsu_yol(b,1)=komsu(ik,1);
      komsu_yol(b,2)=komsu(ik,2);
      komsu_yol(b,3)=komsu(jk,1);
      komsu_yol(b,4)=komsu(jk,2);
      b=b+1;
      end
   end
end

kontrol_x = double(zeros(kombinasyon,map_satir));
kontrol_y = double(zeros(kombinasyon,map_satir));

for il = 1:length(komsu_yol)

   kontrol_y(il,:) = round(linspace(komsu_yol(il,2), komsu_yol(il,4) , 100));
   kontrol_x(il,:) = round(linspace(komsu_yol(il,1), komsu_yol(il,3) , 100));

      for iy=1:map_satir
         if (MAP(kontrol_x(il,iy),kontrol_y(il,iy))==1)
            kontrol_y(il,:)=zeros(1,map_satir);
            kontrol_x(il,:)=zeros(1,map_satir);
            komsu_yol(il,:)=zeros(1,4);

         break;
         end
      end
end

a=double(zeros(kombinasyon,1));
a=find(komsu_yol(:,1));
komsu_yol_engelsiz = double(zeros(kombinasyon,4));
kontrol_x_engelsiz = double(zeros(kombinasyon,map_satir));
kontrol_y_engelsiz = double(zeros(kombinasyon,map_satir));
```

EK-2. (devam) PRM Algoritması

```
for iy=1:length(a)

            nokta_engelsiz=a(iy,1);
            komsu_yol_engelsiz(iy,:)= komsu_yol(a(iy,1),:);
            kontrol_x_engelsiz(iy,:)= kontrol_x(a(iy,1),:);
            kontrol_y_engelsiz(iy,:)= kontrol_y(a(iy,1),:);
end
for m=1:kombinasyon
   if(komsu_yol_engelsiz(m,1)~=0)
       line([komsu_yol_engelsiz(m,2) komsu_yol_engelsiz(m,4)] ,
[komsu_yol_engelsiz(m,1) komsu_yol_engelsiz(m,3)]);
    end

end

nokta_tanim = double(zeros(iterasyon+2,3));
yol_1 = double(zeros(length(komsu_yol_engelsiz),3));
yol_2 = double(zeros(length(komsu_yol_engelsiz),2));
yol_tanim = double(zeros(iterasyon+2,iterasyon));

for it = 1:iterasyon+2
   nokta_tanim(it,1)=it+iterasyon+2;
   nokta_tanim(it,2)=komsu(it,1);
   nokta_tanim(it,3)=komsu(it,2);

end
for it =1:kombinasyon
  for il = 1:iterasyon+2
   if
((komsu_yol_engelsiz(it,1)==nokta_tanim(il,2))&&(komsu_yol_engelsiz(it,2)==nokta_tan
im(il,3)))
      yol_1(it,1)=nokta_tanim(il,1);
      yol_1(it,2)=komsu_yol_engelsiz(it,3);
      yol_1(it,3)=komsu_yol_engelsiz(it,4);
       il=iterasyon+2

   end
  end
end

for it =1:kombinasyon
  for il = 1:iterasyon+2
   if ((yol_1(it,2)==nokta_tanim(il,2))&&(yol_1(it,3)==nokta_tanim(il,3)))
      yol_2(it,1)=yol_1(it,1);
      yol_2(it,2)=nokta_tanim(il,1);
   end
  end
end
```

EK-2. (devam) PRM Algoritması

```
for it = 1:iterasyon+2

    yol_tanim(it,1)=it+iterasyon+2;
    ix=2
   for il = 1:kombinasyon
    if (yol_2(il,1)==it+iterasyon+2)

       yol_tanim(it,ix)=yol_2(il,2);
       ix=ix+1
    end
   end
end
yol_tanim2 = double(zeros(iterasyon+2,iterasyon));
yol_tanimSon= double(zeros(iterasyon+2,iterasyon));
nokta_tanim2 = double(zeros(iterasyon+2,4));
aa=length(find(yol_tanim(1,:)));
for i= 2:length(find(yol_tanim(1,:)))

    satir_sira(i-1,1)=yol_tanim(1,i);
    satir_sira(i-1,2)=sqrt(((nokta_tanim(1,2)) - (nokta_tanim((satir_sira(i-1,1)-iterasyon-
2),2)))^2 + ((nokta_tanim(1,3)) - ( nokta_tanim((satir_sira(i-1,1)-iterasyon-2),3)))^2)

end
aaa=length(satir_sira);

yol_tanim2=yol_tanim;

for i=1:length(satir_sira)
   for j=1:length(satir_sira)
      if(satir_sira(i,2)<satir_sira(j,2))
         temp(1,:)=satir_sira(i,:);
         satir_sira(i,:)=satir_sira(j,:);
         satir_sira(j,:)=temp(1,:);
      end
   end
end

for i=1: length(satir_sira)

    satir_sira(i,3)=satir_sira(i,1);
    satir_sira(i,1)=i+1;
    satir_sira(i,4)=nokta_tanim((satir_sira(i,3)-iterasyon-2),2);
    satir_sira(i,5)=nokta_tanim((satir_sira(i,3)-iterasyon-2),3);
end
nokta_tanim2(1,1)=1;
nokta_tanim2(1,2)=nokta_tanim(1,2);
nokta_tanim2(1,3)=nokta_tanim(1,3);
nokta_tanim2(1,4)=iterasyon+3;
```

EK-2. (devam) PRM Algoritması

```
for i=1: length(satir_sira)
   nokta_tanim2(i+1,1)=satir_sira(i,1);
   nokta_tanim2(i+1,2)=satir_sira(i,4);
   nokta_tanim2(i+1,3)=satir_sira(i,5);
   nokta_tanim2(i+1,4)=satir_sira(i,3);
end

for i=1: length(satir_sira)

   yol_tanim2(1,i+1)=i+1;

end

yol_tanimSon(1,:)=yol_tanim2(1,:);
yol_tanimSon(1,1)=1;
```

EK-3. PRM-A* Algoritması

```
clear all
map_satir = 100; % harita boyutları
map_sutun = 100;MAP = int8(zeros(map_satir,map_sutun));
tic
MAP(20:25,20:80) = 1; % Engeller
MAP(20:80,47:52) = 1;
MAP(20:40,20:25) = 1;
MAP(20:40,75:80) = 1;
 Connecting_Distance = 1; % Haritayı ekrana bas
figure(10);
imagesc(MAP);
colormap(flipud(gray));
hold on;
basla_X = 10; % başlangıç ve bitiş noktaları
basla_Y = 90;
plot(basla_Y,basla_X,'X r')
hedef_X = 90;
hedef_Y = 10;
plot(hedef_Y,hedef_X,'X r')
ylabel('Satır X','fontsize',14)
xlabel('Sütun Y','fontsize',14)
grid on

Random_map = int8(zeros(map_sutun,map_satir));
Random_map(basla_X,basla_Y)=1;
Random_map(hedef_X,hedef_Y)=1;
iterasyon=30;
it=0;

komsu(1,1)=basla_X;
komsu(1,2)=basla_Y;
komsu(iterasyon+2,1)=hedef_X;
komsu(iterasyon+2,2)=hedef_Y;

plot(basla_Y,basla_X,'. g','markersize',40);
plot(hedef_Y,hedef_X,'. g','markersize',40);


while it < iterasyon                                % Rastgele sayı atama bölümü
   random_X=randi(map_satir);
   random_Y=randi(map_sutun);

   if((MAP(random_X,random_Y)==0)&&(Random_map(random_X,random_Y)==0))
      Random_map(random_X,random_Y)=1;
      plot(random_Y,random_X,'. r','markersize',40);
      it = it + 1;
      komsu(it+1,1)=random_X;
      komsu(it+1,2)=random_Y;
```

EK-3. (devam) PRM-A* Algoritması

```
   end
end

kombinasyon=((iterasyon+2)*(iterasyon+1));

komsu_yol=double(zeros(kombinasyon,4
a=2;
it_komsu=length(komsu);
b=1;
for ik = 1:it_komsu
   for jk = 1 : it_komsu
      if(ik~=jk)
      komsu_yol(b,1)=komsu(ik,1);
      komsu_yol(b,2)=komsu(ik,2);
      komsu_yol(b,3)=komsu(jk,1);
      komsu_yol(b,4)=komsu(jk,2);
      b=b+1;
      end
   end
end

kontrol_x = double(zeros(kombinasyon,map_satir));
kontrol_y = double(zeros(kombinasyon,map_satir));

for il = 1:length(komsu_yol)

   kontrol_y(il,:) = round(linspace(komsu_yol(il,2), komsu_yol(il,4) , 100));
   kontrol_x(il,:) = round(linspace(komsu_yol(il,1), komsu_yol(il,3) , 100));

      for iy=1:map_satir
         if (MAP(kontrol_x(il,iy),kontrol_y(il,iy))==1)
            kontrol_y(il,:)=zeros(1,map_satir);
            kontrol_x(il,:)=zeros(1,map_satir);
            komsu_yol(il,:)=zeros(1,4);

         break;
         end
      end
end

a=double(zeros(kombinasyon,1));
a=find(komsu_yol(:,1));
komsu_yol_engelsiz = double(zeros(kombinasyon,4));
kontrol_x_engelsiz = double(zeros(kombinasyon,map_satir));
kontrol_y_engelsiz = double(zeros(kombinasyon,map_satir));
```

EK-3. (devam) PRM-A* Algoritması

```
for iy=1:length(a)

            nokta_engelsiz=a(iy,1);
            komsu_yol_engelsiz(iy,:)= komsu_yol(a(iy,1),:);
            kontrol_x_engelsiz(iy,:)= kontrol_x(a(iy,1),:);
            kontrol_y_engelsiz(iy,:)= kontrol_y(a(iy,1),:);
end
for m=1:kombinasyon
   if(komsu_yol_engelsiz(m,1)~=0)
       line([komsu_yol_engelsiz(m,2) komsu_yol_engelsiz(m,4)] ,
[komsu_yol_engelsiz(m,1) komsu_yol_engelsiz(m,3)]);
    end

end

nokta_tanim = double(zeros(iterasyon+2,3));
yol_1 = double(zeros(length(komsu_yol_engelsiz),3));
yol_2 = double(zeros(length(komsu_yol_engelsiz),2));
yol_tanim = double(zeros(iterasyon+2,iterasyon));

for it = 1:iterasyon+2
   nokta_tanim(it,1)=it+iterasyon+2;
   nokta_tanim(it,2)=komsu(it,1);
   nokta_tanim(it,3)=komsu(it,2);

end
for it =1:kombinasyon
  for il = 1:iterasyon+2
   if
((komsu_yol_engelsiz(it,1)==nokta_tanim(il,2))&&(komsu_yol_engelsiz(it,2)==nokta_tan
im(il,3)))
      yol_1(it,1)=nokta_tanim(il,1);
      yol_1(it,2)=komsu_yol_engelsiz(it,3);
      yol_1(it,3)=komsu_yol_engelsiz(it,4);
       il=iterasyon+2

   end
  end
end

for it =1:kombinasyon
  for il = 1:iterasyon+2
   if ((yol_1(it,2)==nokta_tanim(il,2))&&(yol_1(it,3)==nokta_tanim(il,3)))
      yol_2(it,1)=yol_1(it,1);
      yol_2(it,2)=nokta_tanim(il,1);
   end
  end
end
```

EK-3. (devam) PRM-A* Algoritması

```
for it = 1:iterasyon+2

    yol_tanim(it,1)=it+iterasyon+2;
    ix=2
   for il = 1:kombinasyon
    if (yol_2(il,1)==it+iterasyon+2)

       yol_tanim(it,ix)=yol_2(il,2);
       ix=ix+1
    end
   end
end
yol_tanim2 = double(zeros(iterasyon+2,iterasyon));
yol_tanimSon= double(zeros(iterasyon+2,iterasyon));
nokta_tanim2 = double(zeros(iterasyon+2,4));
aa=length(find(yol_tanim(1,:)));
for i= 2:length(find(yol_tanim(1,:)))

    satir_sira(i-1,1)=yol_tanim(1,i);
    satir_sira(i-1,2)=sqrt(((nokta_tanim(1,2)) - (nokta_tanim((satir_sira(i-1,1)-iterasyon-
2),2)))^2 + ((nokta_tanim(1,3)) - ( nokta_tanim((satir_sira(i-1,1)-iterasyon-2),3)))^2)

end
aaa=length(satir_sira);

yol_tanim2=yol_tanim;

for i=1:length(satir_sira)
   for j=1:length(satir_sira)
      if(satir_sira(i,2)<satir_sira(j,2))
         temp(1,:)=satir_sira(i,:);
         satir_sira(i,:)=satir_sira(j,:);
         satir_sira(j,:)=temp(1,:);
      end
   end
end

for i=1: length(satir_sira)

    satir_sira(i,3)=satir_sira(i,1);
    satir_sira(i,1)=i+1;
    satir_sira(i,4)=nokta_tanim((satir_sira(i,3)-iterasyon-2),2);
    satir_sira(i,5)=nokta_tanim((satir_sira(i,3)-iterasyon-2),3);
end
nokta_tanim2(1,1)=1;
nokta_tanim2(1,2)=nokta_tanim(1,2);
nokta_tanim2(1,3)=nokta_tanim(1,3);
nokta_tanim2(1,4)=iterasyon+3;
```

EK-3. (devam) PRM-A* Algoritması

```
for i=1: length(satir_sira)
  nokta_tanim2(i+1,1)=satir_sira(i,1);
  nokta_tanim2(i+1,2)=satir_sira(i,4);
  nokta_tanim2(i+1,3)=satir_sira(i,5);
  nokta_tanim2(i+1,4)=satir_sira(i,3);
end

for i=1: length(satir_sira)

  yol_tanim2(1,i+1)=i+1;

end

yol_tanimSon(1,:)=yol_tanim2(1,:);
yol_tanimSon(1,1)=1;
for xx = 2:iterasyon+2
   nokta_sira=xx;
   yeni_nokta=(nokta_tanim2(nokta_sira,4))-iterasyon-2;
   if(yeni_nokta>0)
      yeni_nokta_uzunluk=length(find(yol_tanim2(yeni_nokta,:)));
      satir_sira2=double(zeros(iterasyon/2,5));
      for i=2: length(find(yol_tanim2(yeni_nokta,:)))
         axx=yol_tanim2(yeni_nokta,i);
         for j =1:length(find(nokta_tanim2(:,1)))
            if(axx==nokta_tanim2(j,4))
                  yol_tanim2(yeni_nokta,i)=nokta_tanim2(j,1);
            end
         end
      end
      sira=1;
      for i=2:yeni_nokta_uzunluk

         if(yol_tanim2(yeni_nokta,i)~=1)
            satir_sira2(sira,1)=yol_tanim2(yeni_nokta,i);

            if(satir_sira2(sira,1)>iterasyon+2)
               satir_sira2(sira,4)= nokta_tanim((satir_sira2(sira,1)-iterasyon-2),2);
               satir_sira2(sira,5)= nokta_tanim((satir_sira2(sira,1)-iterasyon-2),3);
            end

            if(satir_sira2(sira,1)<iterasyon+2)
               satir_sira2(sira,4)= nokta_tanim2(satir_sira2(sira,1),2);
               satir_sira2(sira,5)= nokta_tanim2(satir_sira2(sira,1),3);
            end

            satir_sira2(sira,2)=sqrt(((nokta_tanim2(nokta_sira,2)) - (satir_sira2((sira),4)))^2
+ ((nokta_tanim2(nokta_sira,3)) - (satir_sira2((sira),5)))^2);
            sira=sira+1;
```

EK-3. (devam) PRM-A* Algoritması

```
    end
end
satir_sira2_boyut=length(find(satir_sira2(:,1)));
for i=1:satir_sira2_boyut
  for j=1:satir_sira2_boyut
     if(satir_sira2(i,2)<satir_sira2(j,2))
        temp2(1,:)=satir_sira2(i,:);
        satir_sira2(i,:)=satir_sira2(j,:);
        satir_sira2(j,:)=temp2(1,:);
     end
  end
end
bbbbb=length(find(nokta_tanim2(:,1)));
sira_artis=1;
for i=1:satir_sira2_boyut
    if(satir_sira2(i,1)==(2*(iterasyon+2)))
      satir_sira2(i,1)=iterasyon+2;
      satir_sira2(i,3)=2*(iterasyon+2);
    end

    if(satir_sira2(i,1)>(iterasyon+3))
       satir_sira2(i,3)=satir_sira2(i,1);
       satir_sira2(i,1)=length(find(nokta_tanim2(:,1)))+sira_artis;
       sira_artis=sira_artis+1;

    end

end

for i=1:satir_sira2_boyut
   yol_tanim2(yeni_nokta,i+1)=satir_sira2(i,1);
   yol_tanim2(yeni_nokta,1)=nokta_sira;
   yol_tanim2(yeni_nokta,satir_sira2_boyut+2)=0;

end

for i=1:satir_sira2_boyut
   nokta_tanim2_boyut=length(find(nokta_tanim2(:,1)));
   if((satir_sira2(i,1)>nokta_tanim2_boyut)&&(satir_sira2(i,1)~=(iterasyon+2)))
      nokta_tanim2(satir_sira2(i,1),1)=satir_sira2(i,1);
      nokta_tanim2(satir_sira2(i,1),2)=satir_sira2(i,4);
      nokta_tanim2(satir_sira2(i,1),3)=satir_sira2(i,5);
      nokta_tanim2(satir_sira2(i,1),4)=satir_sira2(i,3);

   end
end
```

EK-3. (devam) PRM-A* Algoritması

```
        if(xx==(iterasyon+1))
            nokta_tanim2(iterasyon+2,1)=iterasyon+2;
            nokta_tanim2(iterasyon+2,2)=hedef_X;
            nokta_tanim2(iterasyon+2,3)=hedef_Y;
            nokta_tanim2(iterasyon+2,4)=2*(iterasyon+2);
        end
        yol_tanimSon(nokta_sira,:)=yol_tanim2(yeni_nokta,:);
    end
end

nokta_tanim2(iterasyon+2,1)=iterasyon+2;
nokta_tanim2(iterasyon+2,2)=hedef_X;
nokta_tanim2(iterasyon+2,3)=hedef_Y;
nokta_tanim2(iterasyon+2,4)=2*(iterasyon+2);
xxxx=(length(nokta_tanim2)-1)
for i=2:(length(nokta_tanim2)-1)
    text(nokta_tanim2(i,3),nokta_tanim2(i,2),string(i));
end

tekrar = 10000;

open_list = zeros(map_satir,map_sutun);
closed_list = zeros(map_satir,map_sutun);
for n = 1:map_satir
    for m = 1:map_sutun
        if MAP(n,m) == 1
            closed_list(n,m) = 1;
        end
    end
end
for n = 1:map_satir
    for m = 1:map_sutun
        h(n,m) = sqrt(((n-hedef_X)^2)+((m-hedef_Y)^2));
    end
end
g = zeros(map_satir,map_sutun);
g_new = zeros(map_satir,map_sutun);
f = ones(map_satir,map_sutun);
hf = zeros(iterasyon+2,7);
for m = 1:(iterasyon+2)
        hf(m,2) = nokta_tanim2(m,2);
        hf(m,3) = nokta_tanim2(m,3);
        hf(m,1) = 20000;
        hf(m,7) = m;

end
```

EK-3. (devam) PRM-A* Algoritması

```
hf(1,1)=h(basla_X,basla_Y);
f(basla_X,basla_Y)=h(basla_X,basla_Y);
open_list(basla_X,basla_Y) = 1;
zz=0;
while zz < tekrar
   if sum(sum(open_list)) == 0
      disp('Hedef düğüme ulaşılamadı!');
      disp('Açık Liste: boş..!!');
      zz=10000;
   else sum(sum(open_list)) >= 1
      hf = sortrows(hf,[1]);
      gecerli_X = hf(1,2);
      gecerli_Y = hf(1,3);
      open_list(gecerli_X,gecerli_Y) = 0;
      closed_list(gecerli_X,gecerli_Y) = 1;
      incelenen_nokta=hf(1,7);
      for m = 1:(iterasyon+2)
         if (hf(m,2) == gecerli_X)&&(hf(m,3) == gecerli_Y)
            hf(m,1) = 20000;
         end
      end
   end
   if (gecerli_X == hedef_X)&&(gecerli_Y == hedef_Y)
      disp('Hedef düğüme ulaşıldı!');
      zz = iterasyon-1;
   else ((gecerli_X ~= hedef_X)&&(gecerli_Y ~= hedef_Y))
      tum_komsu_kapali = 0;

      komsu_boyut=length(find(yol_tanimSon(incelenen_nokta,:)));
      for m = 2: komsu_boyut
         siradaki_nokta=yol_tanimSon(incelenen_nokta,m);
         komsu_X = nokta_tanim2(siradaki_nokta,2) ;
         komsu_Y = nokta_tanim2(siradaki_nokta,3) ;

         if (closed_list(komsu_X,komsu_Y)==0)
            tum_komsu_kapali = tum_komsu_kapali+1;
            if open_list(komsu_X,komsu_Y) == 0
               open_list(komsu_X,komsu_Y) = 1;
               for k = 1:(iterasyon+2)
                  if (hf(k,2) == komsu_X)&&(hf(k,3) == komsu_Y)
                     hf(k,4) = gecerli_X;
                     hf(k,5) = gecerli_Y;
                  end
               end
               komsu_g(komsu_X,komsu_Y) = sqrt(((komsu_X-gecerli_X)^2)+((komsu_Y-
gecerli_Y)^2));
               g(komsu_X,komsu_Y) =
komsu_g(komsu_X,komsu_Y)+g(gecerli_X,gecerli_Y);
```

EK-3. (devam) PRM-A* Algoritması

```
                f(komsu_X,komsu_Y) = g(komsu_X,komsu_Y)+h(komsu_X,komsu_Y);
                for k = 1:(iterasyon+2)
                    if (hf(k,2) == komsu_X)&&(hf(k,3) == komsu_Y)
                        hf(k,1) = f(komsu_X,komsu_Y);
                    end
                end
            elseif open_list(komsu_X,komsu_Y) == 1
                komsu_g(komsu_X,komsu_Y) = sqrt(((komsu_X-gecerli_X)^2)+((komsu_Y-
gecerli_Y)^2));
                g_new(komsu_X,komsu_Y) =
komsu_g(komsu_X,komsu_Y)+g(gecerli_X,gecerli_Y);
                if g_new(komsu_X,komsu_Y) < g(komsu_X,komsu_Y)
                    for k = 1:(iterasyon+2)
                        if (hf(k,2) == komsu_X)&&(hf(k,3) == komsu_Y)
                            hf(k,4) = gecerli_X;
                            hf(k,5) = gecerli_Y;
                        end
                    end
                    g(komsu_X,komsu_Y) = g_new(komsu_X,komsu_Y);
                    f(komsu_X,komsu_Y) = g(komsu_X,komsu_Y)+h(komsu_X,komsu_Y);
                    for k = 1:(iterasyon+2)
                        if (hf(k,2) == komsu_X)&&(hf(k,3) == komsu_Y)
                            hf(k,1) = f(komsu_X,komsu_Y);
                        end
                    end
                end
            end
        elseif tum_komsu_kapali == 0
            hf = sortrows(hf,[1]);
        end
    end
  end
  zz=zz+1;
end
dd = 1;
yol(dd,1) = hedef_X;
yol(dd,2) = hedef_Y;
yol(dd,3) = iterasyon+2;
for k = 1:(iterasyon+2)
  if (hf(k,2) == yol(dd,1))&&(hf(k,3) == yol(dd,2))
      yol(dd+1,1) = hf(k,4);
      yol(dd+1,2) = hf(k,5);
      yol(dd,3) = hf(k,7);
      dd = dd+1;
  end
end
yol_son = flipud(yol);
yol_son(1,:)=[];
```

EK-3. (devam) PRM-A* Algoritması

```
adim_sayisi = length(yol_son)


for i=1:adim_sayisi-1

   p1=line([yol_son(i+1,2) yol_son(i,2)],[yol_son(i+1,1) yol_son(i,1)] ,'LineWidth',3);
   p1.Color='red';

end

aaaa=length(find(yol_tanim(1,:)));

toc



yol_uzunlugu=0;

for ix = 1:adim_sayisi-1

   uzunluk=sqrt(((yol_son(ix+1,1)-yol_son(ix,1))^2)+((yol_son(ix+1,2)-
yol_son(ix,2))^2));
   yol_uzunlugu=yol_uzunlugu+uzunluk;
end
```

GAZİ GELECEKTİR...