KOCAELİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ELEKTRONİK VE BİLGİSAYAR EĞİTİMİ
ANABİLİM DALI

DOKTORA TEZİ

OTONOM ARAÇLAR İÇİN YOL BULMA PROBLEMİNİN GENETİK ALGORİTMALAR VE FPGA İLE ÇÖZÜMÜ VE GERÇEKLEŞTİRİLMESİ

ADEM TUNCER

KOCAELİ 2013

KOCAELİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ELEKTRONİK VE BİLGİSAYAR EĞİTİMİ
ANABİLİM DALI

DOKTORA TEZİ

OTONOM ARAÇLAR İÇİN YOL BULMA PROBLEMİNİN GENETİK ALGORİTMALAR VE FPGA İLE ÇÖZÜMÜ VE GERÇEKLEŞTİRİLMESİ

ADEM TUNCER

Doç.Dr. Mehmet YILDIRIM
Danışman, Kocaeli Üniv. .....

Prof.Dr. Yunus Emre ERDEMLİ
Jüri Üyesi, Kocaeli Üniv. .....

Doç.Dr. Celal ÇEKEN
Jüri Üyesi, Sakarya Üniv. .....

Yrd.Doç.Dr. Serhat YILMAZ
Jüri Üyesi, Kocaeli Üniv. .....

Yrd.Doç.Dr. Ali ÇALHAN
Jüri Üyesi, Düzce Üniv. .....

Tezin Savunulduğu Tarih: 29.04.2013

ÖNSÖZ ve TEŞEKKÜR

Teknolojik gelişmeler ışığında, her alanda olduğu gibi otonom gezgin robotlar için de araştırma geliştirme çalışmaları hızlı bir şekilde ilerleme kaydetmektedir. Otonom gezgin robotların yerine getirmesi gereken görevlerden bir tanesi de yol planlama problemini başarılı bir şekilde çözmeleri ve bu doğrultuda kendilerine verilen hedef noktaya doğru hareket edebilmeleridir. Yol planlama problemleri için geliştirilen diğer yöntemlerin yanı sıra, genel bir optimizasyon yöntemi olan genetik algoritmalar (GA) da sıkça kullanılmaktadır. Sahada programlanabilir kapı dizileri (FPGA) teknolojisinin gelişimi ile birlikte, GA'nın donanım olarak gerçekleştirilmesi de mümkün hale gelmiştir. Bu tez çalışmasında, FPGA üzerinde çalışan bir GA donanımı gerçekleştirilerek, yol bulma probleminin çözüm hızı artırılmış, gezgin robotun taşıması gereken yük ve hacim azaltılmaya çalışılmıştır. Hazırlamış olduğum tezin, ileride yapılacak olan diğer çalışmalara katkı sağlamasını umuyorum.

Tez çalışması süresince gösterdiği destek ve ilgiyle, çalışmamın tamamlanmasında büyük emeği olan değerli danışman hocam Doç.Dr. Mehmet YILDIRIM'a, çalışmalarım sırasında fikirleriyle bana yol gösteren ve destekleyen değerli hocam Prof.Dr. Kadir ERKAN'a, doktora tez izleme jürisinde bulunan hocalarım Doç.Dr. Celal ÇEKEN ve Yrd.Doç.Dr. Serhat YILMAZ'a, desteklerini her zaman hissettiğim ve sağladıkları huzurlu çalışma ortamı için başta oda arkadaşım Arş.Gör. Faruk AKTAŞ ve diğer tüm değerli çalışma arkadaşlarıma teşekkürü bir borç bilirim.

Kocaeli Üniversitesi, Bilimsel Araştırma Projeleri Birimi'ne sağladıkları proje desteği için teşekkür ederim.

Ayrıca, bu süreçte verdikleri destekler için tüm aileme ve göstermiş olduğu sabır ve anlayış için sevgili eşim Aslı'ya teşekkürlerimi, biricik kızım Beyza Nur'a sevgilerimi sunarım.

Nisan - 2013 Adem TUNCER

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

SİMGELER DİZİNİ VE KISALTMALAR

α : Robotun bütünsel düzleme göre açısı
d : İki nokta arasındaki uzaklık
f : Yol bulma problemi için kullanılan amaç fonksiyonu
g : Mevcut düğümle hedef arasındaki mesafe
h : Sezgisel fonksiyon
n : Kromozomdaki gen sayısı
MHz : Mega Hertz
p_i : Kromozomdaki genler (herhangi bir i noktası)
P : Robotun koordinat ve açıya bağlı olarak konumu
S : Genetik algoritmada bireyin seçilme olasılığı
U : Genetik algoritma için uygunluk değeri
μs : Mikro saniye

Kısaltmalar

ARCOS : Advanced Robot Control and Operations Software (Gelişmiş Robot Kontrol ve Operasyon Yazılımı)
ARIA : Advances Robotics Interface for Applications (Uygulamalar için Gelişmiş Robot Arayüzü)
ASIC : Application Specific Integrated Circuits (Uygulamaya Özgü Tümleşik Devreler)
BSB : Base System Builder (Temel Sistem Oluşturucu)
CIP : Create or Import Peripheral (Çevre Birimi Oluştur ya da Ekle)
CPLD : Complex Programmable Logic Devices (Karmaşık Programlanabilir Lojik Cihazlar)
CPU : Central Processing Unit (Merkezi İşlem Birimi)
DA : Doğru Akım
DLMB : Data Side Local Memory Bus (Veri Yerel Bellek Yolu)
DVI : Digital Visual Interface (Sayısal Görüntü Arayüzü)
EDK : Embedded Development Kit (Gömülü Sistem Geliştirme Aracı)
FIFO : First In First Out (İlk Giren İlk Çıkar)
FPGA : Field Programmable Gate Array (Sahada Programlanabilir Lojik Kapılar)
FSL : Fast Simplex Link (Hızlı Tekyönlü Bağlantı)
GA : Genetik Algoritmalar
GUI : Graphical User Interface (Grafiksel Kullanıcı Arayüzü)
HDL : Hardware Description Language (Donanım Tanımlama Dili)
IEEE : Institute of Electrical and Electronics Engineers (Elektrik ve Elektronik Mühendisleri Enstitüsü)
IIC : Inter Integrated Circuit (Entegre Arası Cihaz)
ILMB : Instruction Side Local Memory Bus (Komut Yerel Veri Yolu)
IP CORE : Intellectual Property Core (Fikri Mülkiyet Çekirdeği)

IPIF	: Intellectual Property Interface (Fikri Mülkiyet Arayüzü)
ISE	: Integrated Software Environment (Tümleşik Yazılım Ortamı)
JTAG	: Joint Test Action Group (Ortak Test Eylem Grubu)
LCD	: Liquid Crystal Display (Sıvı Kristal Ekran)
LFSRs	: Linear Feedback Shift Registers (Doğrusal Geri Beslemeli Kaydırma Yazmaçları)
LMB	: Local Memory Bus (Yerel Bellek Yolu)
LUT	: Look Up Table (Doğruluk Tablosu)
MUX	: Multiplexer (Çoklayıcı)
PCI	: Peripheral Component Interconnect (Çevresel Bileşen Arayüzü)
PLB	: Processor Local Bus (İşlemci Yerel Veri Yolu)
RAM	: Random Access Memory (Rastgele Erişimli Bellek)
SDK	: Software Development Kit (Yazılım Geliştirme Kiti)
SIP	: Server Information Packet (Sunucu Bilgi Paketi)
SoC	: System on Chip (Tek Yongada Sistem)
USB	: Universal Serial Bus (Evrensel Seri Veriyolu)
VHDL	: Very High-Speed Integrated Circuit Hardware Description Language (Çok Hızlı Tümleşik Devreler için Donanım Tanımlama Dili)
XPS	: Xilinx Platform Studio (Xilinx Platform Aracı)

OTONOM ARAÇLAR İÇİN YOL BULMA PROBLEMİNİN GENETİK ALGORİTMALAR VE FPGA İLE ÇÖZÜMÜ VE GERÇEKLEŞTİRİLMESİ

ÖZET

Bu tez çalışmasında, otonom bir gezgin robot için yol bulma problemi Genetik Algoritmalar (GA) kullanılarak çözüldü. GA ile yol bulma problemine ilişkin yeni bir mutasyon operatörü geliştirildi ve bu yeni mutasyon operatörü ile daha önce literatürde sunulmuş olan farklı mutasyon operatörleri karşılaştırıldı. Karşılaştırmalar sonucunda yeni mutasyon operatörünün daha başarılı performans sergilediği sonucuna varıldı.

GA ile yol bulma algoritması Pioneer 3-DX gezgin robot üzerinde denendi. Robot ile haberleşmede Matlab programı kullanıldı. Görüntü almak amacıyla, laboratuvar tavanına bir kamera yerleştirilerek lokalizasyon gerçekleştirildi, robotun ortamdaki engelleri ve kendi konumunu belirleyebilmesi sağlandı. Böylece, lokalizasyon ve yol bulma işlemleri bütünleşik bir sistem olarak gerçek laboratuvar ortamında denendi.

Bir bilgisayar üzerinde çalıştırılan GA, daha sonra Sahada Programlanabilir Kapı Dizisi (Field Programmable Gate Array, FPGA) üzerinde donanımsal olarak gerçekleştirildi. GA önce yarı yazılımsal yarı donanımsal bir şekilde hibrit olarak daha sonra tamamı donanımsal olarak tasarlandı. Hibrit yapıdaki GA'nın, sadece amaç fonksiyon hesaplayan bölümü donanımsal Intellectual Property Core (IP çekirdek) ile, diğer operatörleri de Microblaze adı verilen yazılımsal işlemci üzerinde çalışacak şekilde tasarlandı ve performans katkısı incelendi. Daha sonra da, sistem tamamen IP çekirdek ile donanımsal olarak gerçekleştirildi ve performans katkısı incelendi.

Kameradan görüntü alıp işleyerek harita oluşturan, GA ile yol bulan, robot ile haberleşen, robotun GA ile bulunan bir yol bilgisine göre veya kullanıcı tarafından belirlenen bir yol koordinatlarına göre ilerleyebilmesini sağlayan Matlab-GUI tabanlı bir arayüz yazıldı.

Tez çalışması kapsamında sistemin tamamı gerçek bir laboratuvar ortamında uygulandı, böylece çalışmanın sadece benzetim ortamında kalmayıp, gerçek dünyada uygulanabilirliği gösterildi.

Anahtar Kelimeler: FPGA, Genetik Algoritmalar, Gezgin Robot, IP Çekirdek, Yol Bulma.

THE SOLUTION AND IMPLEMENTATION OF PATH PLANNING PROBLEM WITH GENETIC ALGORITHMS AND FPGA FOR AUTONOMOUS VEHICLES

ABSTRACT

In this thesis, the problem of the path planning for an autonomous mobile robot was resolved by using Genetic Algorithms. A new mutation operator was developed to address the problem of path planning by GA. This mutation operator was compared with various other mutation operators documented in research literature. In light of the comparisons, it was found that the new mutation operator showed superior performance than the other operators.

Path planning with GA was tested on the Pioneer 3-DX mobile robot. Matlab was used to communicate with the robot. On the part of localization, a camera was installed on the ceiling of the laboratory to receive images and to assist the robot with the ability to determine its own location and the obstacles in the environment. This way, procedures of localization and path planning were tested in the real lab environment.

First run on a computer, GA was later performed in hardware on the Field Programmable Gate Array (FPGA). First, GA was designed as a hybrid fashion, in which a section was designed in software and the other sections were designed in hardware. Then, it was designed as entirely in hardware. In the hybrid design, only the component, that is calculating the objective function, was designed with hardware, namely IP (intellectual property) core. The rest of the operators were designed in software to work on the Microblaze processor, which is a soft-processor. Performance contribution was evaluated by comparing with the fully software design. Subsequently, the system was designed entirely in hardware with an IP core and performance contribution was evaluated.

A Matlab-GUI based interface was written for fetching images from the camera and processing them to form a map, path planning with GA, communicating with the robot, moving the robot according to the path found by the GA or the path, coordinates of which are provided by the user.

Within the frame of the dissertation study, the entire system was performed in a real lab environment, so as to demonstrate that the study is applicable in the real world, besides being simulated.

Keywords: FPGA, Genetic Algorithms, Mobile Robot, IP Core, Path Planning.

GİRİŞ

Günümüzde robotlar, askeri alanlardan bilimsel alanlara, endüstriden günlük yaşama kadar çok geniş bir yelpazede kullanılmaktadır. Teknolojik gelişmelere paralel olarak, robot konusundaki çalışmalar çoğalmakta ve robotların kullanım alanları gün geçtikçe artmaktadır. Robotlar görevlerini yerine getirirken, yapacakları işlerin tamamı insanlar tarafından kontrol edilebileceği gibi, belli hareketleri de kendi başlarına otonom olarak yapabilirler. Otonom olma özelliği, bir robotun karşısına çıkabilecek önceden bilinen ya da beklenmedik durumlar karşısında, insan yardımı olmadan kendi kendine karar verebilme yeteneğine sahip olmasıdır.

Otonom robotlar pek çok görevi kendi kendine yerine getirebilecek şekilde geliştirilmektedir. Özellikle askeri alanlarda, otonom robotlar önemli görevler üstlenmektedir. Savaş alanını izleme, veri toplama, tehlikeli durumları bildirme, tehlikeli bölgelerden sakınma, haberleşme, istihbarat, keşif gibi görevleri yerine getirmek gibi pek çok çalışmada kullanılmaktadır. İnsan hayatı için zararlı veya tehlikeli görev ve durumlarda, otonom robotlar önemli bir rol oynamakta ve bu rolü giderek artmaktadır. Bunların dışında, otonom robotlar tarım, jeolojik araştırmalar, arama kurtarma çalışmaları vb. sivil uygulamalar için de geniş bir alanda kullanılmaktadır.

Gezgin bir robotun otonom olabilmesi için birtakım özelliklere sahip olması gerekmektedir. Bu özelliklerden bazıları; kendi kendine karar verebilme ve verdiği kararı uygulama yeteneğine sahip olma, yol planlaması yapabilme ve planlanan yol boyunca hareket edebilme, engel bulma, tanıma ve işleme kabiliyeti, algılayıcılardan gelen verileri okuyabilme ve değerlendirebilme, görüntü işleme, keşif ve gözetleme yapabilme olarak sıralanabilir.

Otonom gezgin robotların engellere çarpmadan, güvenli bir şekilde hareket edebilmesi ve doğru kararlar verebilmesi için bulunduğu çevre hakkında tam olarak bilgi sahibi olması, dinamik veya kesin olarak bilinmeyen çevrelerde beklenmedik engelleri keşfedebilme ve belirleme özelliğine sahip olması gerekmektedir. Robotun

bu bilgileri elde edebilmesi için, üzerinde algılayıcılar bulunması veya kamera gibi cihazlardan yararlanması gerekmektedir.

Otonom bir gezgin robotun yerine getirmesi gereken üç temel görev bulunmaktadır;

- Lokalizasyon; en basit ifadeyle, robotun "ben nerdeyim?" sorusuna cevap bulması, içinde bulunduğu çevrede kendi konum bilgisine sahip olmasıdır. Ayrıca, ortamda bulunan engellerin, hedefin ve ilgili nesnelerin algılanması gerekir. Bu bilgiye de üzerinde bulunan lazer, sonar, yaklaşım vb. algılayıcılar ya da kamera sayesinde erişebilmektedir. Robotun, kendisi de dahil olmak üzere, çevredeki tüm nesneleri algıladıktan sonra, çevrenin bir haritasını oluşturması, nesnelerin harita üzerinde koordinat veya konumlarını hesaplaması gerekir.

- Yol planlama; robotun içinde bulunduğu çevrede engellere çarpmayacak şekilde hareket edebileceği yolu planlaması ve bu yol üzerinde hedef doğrultusunda ilerleyebilmesidir. Yol planlaması için kullanılan bazı matematiksel veya sezgisel yöntemler mevcuttur. Belirlenen yolun kalitesi ve belirlenme hızı kullanılan yöntem ile doğrudan ilişkilidir.

- Engelden sakınım; robotun planlamış olduğu yolda ilerlerken, engellere çarpmamak için, engel ile arasında bir güvenlik mesafesini korumasıdır. Robotlar, her ne kadar engelsiz bir yol planlaması yapabilseler de, zeminin veya hareket mekanizmasının kaygan olması, sürtünme, engellerin yer değiştirmesi vb. nedenlerden dolayı planlanan yoldan sapabilmekte veya engele çarpabilmektedirler. Robotlar, algılayıcılardan aldıkları işaretleri bir takım yöntemlerle değerlendirerek engelden sakınımı sağlayabilirler.

Otonom gezgin robotların lokalizasyon, yol planlaması ve engelden sakınımlarıyla ilgili olarak literatürde pek çok çalışma yapılmıştır. Bu tez çalışmasının amacı, otonom gezgin robotların yol bulma problemine ilişkin çözümler üretmek olduğundan, tez çalışmasında özellikle yol planlaması üzerinde durulmuştur.

Yol planlama, engellerin olduğu bir ortamda, otonom gezgin robotun bir başlangıç noktasından başka bir hedef noktasına gidebilmesi için engellere çarpmayacağı uygun bir yolu bulma işlemidir [1]. Otonom gezgin robotlar için yol planlama

konusu halen aktif olarak çalışılan bir araştırma alanıdır ve bu alanda kullanmak üzere pek çok yöntem geliştirilmiştir. Uygulama tipine ve ortamına bağlı olarak her yöntemin zayıf ve güçlü olduğu yönleri bulunmaktadır. Bazı yöntemler kapalı alanlar için daha uygun iken, bazıları dış ortam için daha uygundur. Bazıları yöntemler statik ortamlar için daha uygun iken, bazıları dinamik ortamlar için uygundur. Bazıları karasal gezgin robotlar için uygun iken, bazıları da hava araçları için uygun olabilmektedir [2].

Geleneksel arama ve optimizasyon metotları ile karşılaştırıldığında, evrimsel algoritmalar, problem hakkında az bilgi olduğu ya da hiçbir ön bilgiye sahip olunmadığı durumlarda, daha genel, sağlam ve basit bir yapıya sahiptir [3]. GA karmaşık problemler için en güçlü arama algoritmalarından biri olarak tanımlanır. Bu tip problemlerin çözümünde GA'yı cazip yapan temel karakteristik, özelliği gereği paralel arama metotlarına sahip olması ve dinamik ortamlarda dahi optimum çözüm bulabilme yeteneğine sahip olmasıdır [4].

GA pek çok uygulamada çok iyi sonuçlar bulabilmektedir. Paralel bir arama algoritması olmasına rağmen, yazılımda seri olarak kodlandığı için hız duyarlı problemlerde dezavantajlı duruma düşmektedir. Bu durum, optimizasyon sürecinde kabul edilemez gecikmelere neden olabilmektedir [5]. Hız veya gecikme problemlerinin üstesinden gelebilmek için kullanılabilecek önlemlerden birisi, GA'yı donanımsal olarak gerçekleştirmektir. Paralel hesaplama özelliğinden dolayı, donanımsal olarak gerçekleştirilen GA'nın yazılımsal olarak gerçekleştirilen GA'ya göre daha yüksek performansa sahip olacağı düşünülmektedir [6].

Genel amaçlı tasarlanan GA'nın tasarımı esnek ve kolay yapılandırılabilir olmalıdır. Bu esneklik yazılım olarak kolay gerçekleştirilebildiği halde, donanım olarak pek de kolay olmamaktadır. Bu bakımdan, Sahada Programlanabilir Kapı Dizileri (Field Programmable Gate Arrays, FPGA) geliştirilene kadar GA'nın donanımsal olarak gerçekleştirilmeleri mümkün olmamıştır [7].

FPGA'ler düşük maliyet ve yüksek hızlarından dolayı, popülerliği her geçen gün artan elemanlardır. FPGA'ler donanım tasarımcılarına büyük bir esneklik sağlamaktadır. FPGA'deki kapasite, performans ve mimari özelliklerin artmaya devam etmesi, pek çok tasarımın FPGA kullanılarak gerçekleştirilmesine olanak

sağlamaktadır [8]. Kullanıcılar kendilerine özgü, etkin ve esnek donanım tasarımları oluşturmak için FPGA kullanabilirler. Paralel işlem yapabilme yeteneğine ve kullanıcı açısından tasarım kolaylığına sahip FPGA'ler, özellikle hız gerektiren sistemleri geliştirmek için kullanılmaktadır. Literatürde, FPGA'lerin kullanıldığı pek çok çalışma bulunmaktadır. FPGA kullanarak gerçekleştirilen ve ilk donanım tabanlı GA olan HGA (hardware based genetic algorithm) çalışması 1995 yılında açıklanmıştır [7]. O tarihten bu yana, FPGA üzerinde GA'nın kullanıldığı ve yol bulma probleminin çözümüne ilişkin fazla çalışma bulunmamaktadır.

Literatürde, yol bulma problemlerinin çözümüne ilişkin pek çok çalışma bulunmaktadır. Bunların önemli olanları, özellikle GA'nın yol bulma problemlerinde kullanılması ile ilgili olanlar ve GA'nın FPGA üzerinde donanımsal olarak gerçekleştirilmesi ile ilgili olanlar aşağıda verilmektedir.

Hu ve Yang [1] tarafından gerçekleştirilen çalışmada, standart GA kullanmak yerine, robotun yol bulma problemine özel, bilgi tabanlı bir GA kullanılmıştır. Benzetim çalışmaları sonuçlarına göre, amaçlanan bilgi tabanlı GA'nın, hem statik hem de dinamik ortamlarda en iyi veya en iyiye yakın bir yolu bulma yeteneğine sahip olduğu belirtilmiştir.

Al-Taharwa ve diğ. [9] tarafından gerçekleştirilen çalışmada, çalışma alanı bilinen statik bir ortamda, gezgin robotun yol bulma problemini çözmek için GA'yı kullanması sunulmuştur. Çalışmada, yol uzunluğunun, basitleştirilmiş bir uygunluk fonksiyonu olarak kullanılması önerilmiştir. Farklı büyüklükteki nüfus yapıları için, algoritmanın performansı araştırılmıştır. Deneysel sonuçlara dayanarak, amaçlanan yaklaşımın farklı statik ortamlarda etkili olduğu belirtilmiştir. Çalışmada, robot sadece x veya y yönünde hareket etmekte, çapraz olarak ilerleyememektedir. Bu bakımdan algoritma en kısa yolu bulamamaktadır.

Nagib ve Gharieb [10] tarafından gerçekleştirilen çalışmada, engellerle dolu iki boyutlu statik bir ortamda, en kısa yolu bulmayı sağlayan gezgin bir robot için, ikili sayı sistemi ile kodlanmış ve sabit kromozom uzunluklu bir GA geliştirilmiştir. Algoritmanın diğer algoritmalardan daha az hesaplama gücü gerektirdiği belirtilmiştir. Kromozom uzunluğu sabit olmakla beraber, ortamdaki engellerin

sayısına bağlı olarak belirlenmiştir. Değişken uzunluklu kromozom yapılarının dinamik ortamlarda gerekebileceği vurgulamıştır.

Guo ve diğ. [11] tarafından gerçekleştirilen çalışmada, çok sayıda engel bulunan iki boyutlu bir çalışma alanı modeli tasarlanmıştır. Çalışma alanı düzgün ve ızgaralı bir yapı olarak ifade edilmiştir. Optimum yol, 16 yöndeki komşu hücrelerde arama yapan iyileştirilmiş bir D* algoritması kullanarak planlanmıştır. Algoritma WiRobotX80 gezgin robot ile test edilmiştir. Test sonuçlarında, iyileştirilen D* algoritmasının geleneksel D* algoritmasına oranla daha iyi sonuçlar verdiği gözlemlenmiştir. Robotun yön için en küçük dönme açısı Π/8 alınmıştır. Açı miktarının azalmasından dolayı, gereksiz yol maliyetinin azaldığı ve robotun yol bulma hassasiyetinin arttığı ifade edilmiştir.

Çınar ve diğ. [12] tarafından gerçekleştirilen çalışmada, otonom gezgin bir robotun laboratuvar ortamında gezinebilmesi için bir yöntem geliştirilmiştir. Robotun yol bulma işlemi için A* algoritması, engelden sakınma için de potansiyel alan yönteminin iki farklı versiyonu kullanılmış ve bunlar karşılaştırılmıştır. Robotun gideceği hedef nokta bir engelin yakınında olduğu durumda, klasik potansiyel alan yönteminin robotu hedefe ulaştıramadığını ancak geliştirilen yeni potansiyel alan yöntemi ile bu problemin ortadan kaldırıldığını belirtmişlerdir. Çalışmada iki farklı potansiyel alan yöntemi aynı probleme uygulanmış ve yeni potansiyel alan yaklaşımının daha etkin olduğu belirtilmiştir.

Pakkan ve Ermiş [13] tarafından gerçekleştirilen çalışmada, Hava Kuvvetleri'nin gözlem amaçlı insansız hava araçlarının görev planlanmasını, çevrimdışı olarak, daha hızlı ve daha etkin şekilde yapılabilmesine yardımcı olabilecek GA tabanlı bir çözüm yöntemi geliştirilmiştir. İnsansız hava araçları için, etkin rotaların bulunması amacıyla geliştirilen sistemde, görsel bir kullanıcı arayüzü oluşturulmuştur. Planlama subayının, bu arayüzü kullanarak, istenilen hedeflerin coğrafi koordinatlarını, çevrimiçi internet aracılığıyla Google Maps sunucusundan Delphi programına alıp, burada veri paketleri haline getirerek ve bunları Matlab Simulink ile işleyerek, geliştirilen GA tabanlı çözüm tekniğiyle en uygun rotaları bulabilmekte ve görsel olarak izleyebilmekte olduğunu ifade etmişlerdir.

Chen ve Qin [14] tarafından gerçekleştirilen çalışmada, robot yol planlamasında kullanmak üzere hibrit bir iyileştirilmiş GA yaklaşımı sunulmuştur. Çalışma ortamı hücre tabanlı olarak modellenmiştir. GA'nın performansını attırmak için GA ve benzetilmiş tavlama (simulated annealing) algoritmaları beraber kullanılmıştır. Standart GA ve hibrit GA karşılaştırılmış ve önerilen çalışmanın daha başarılı olduğu ifade edilmiştir.

Changan ve diğ. [15] tarafından gerçekleştirilen çalışmada, gezgin robot için iyileştirilmiş GA tabanlı dinamik yol planlaması amaçlanmıştır. Yerel optimumdan kaçınmak için rastgele mutasyon kullanmak yerine, tepe tırmanma (hill climbing) metodunun kullanıldığı daha iyi bir mutasyon metodu geliştirilmiştir. Çözümün yakınsama hızını arttırmak için nüfus, parçacık sürüsü (particle swarm) algoritması kullanılarak güncellenmiştir. Çalışmanın sonucunda, yol bulma algoritmasının geçerli ve etkili olduğu ifade edilmiştir.

Kala [16] tarafından gerçekleştirilen çalışmada, çoklu robotlar için bir yol planlama çalışması sunulmuştur. Çalışma ortamı olarak labirent şeklinde bir harita kullanılmıştır. Farklı sayıda robot, başlangıç konumu olarak haritanın farklı yerlerine yerleştirilmiş ve her biri için farklı hedef noktaları belirlenmiştir. Ana evrimsel algoritma tüm optimum yolları hesaplarken, her bir robot için gramer tabanlı GA kullanılmıştır. Bireysel robotların evrimsel algoritmaları arasındaki işbirliği ile tüm optimum yolların üretildiği belirtilmiştir. Yazılım olarak Java kullanılmıştır. Deneysel sonuçların, algoritmanın çeşitli senaryolar altında kullanılabilirliğini doğruladığı ifade edilmiştir.

Fernando ve diğ. [17] tarafından gerçekleştirilen çalışmada, daha önce donanımsal olarak gerçekleştirilen GA'da, GA parametrelerinin programlanabilirliğinin eksikliği, önceden tanımlanmış sistem mimarilerinin esnek olmaması ve birden fazla amaç fonksiyonunu desteklemede yetersiz kalmaları gibi eksikliklerin olduğu ifade edilmiştir. Çalışmada, bu problemlerin çözüm için, GA'nın FPGA üzerinde genel amaçlı olarak gerçekleştirilmesi amacıyla IP çekirdeği tasarlanmıştır. Çalışmada Xilinx Virtex II Pro FPGA cihazı (xc2vp30) kullanılmıştır. Tasarlanan IP çekirdeğinde; GA'nın nüfus sayısı, jenerasyon sayısı, çaprazlama ve mutasyon oranları, rastgele sayı üretimi için gerekli olan başlangıç değeri dinamik olarak

değiştirilebilmektedir. GA çekirdeği performansının standart optimizasyon test fonksiyonlarıyla test edildiği belirtilmiştir. Deneylerde, IP çekirdeği ile, optimum yolun bulunduğu ya da %3,7 hata ile optimuma yakın bir yol bulunduğu ifade edilmiştir.

Yan-cong ve diğ. [18] tarafından gerçekleştirilen çalışmada, tipik bir optimizasyon problemi olan gezgin satıcı probleminin, FPGA tabanlı GA ile gerçekleştirilmesi sunulmuştur. GA tabanlı gezgin satıcı problemlerinin yazılımsal olarak çok yavaş olduğu ve bu yüzden GA'nın donanımsal olarak gerçekleştirildiği ifade edilmiştir. Algoritma yapısındaki sıralı ve paralel işlemlerin başarıyla gerçekleştirildiği ve çalışma hızının iyileştirildiği belirtilmiştir. Deneysel sonuçlara dayanarak, sistem kaynaklarının tüketiminin azaldığı ve donanımın çalışma hızının yazılıma göre 2-3 kat arttığı ifade edilmiştir. Çalışmada, Altera EP2C70F896C6 cihazı ve donanım dili olarak ta Verilog kullanılmıştır.

Kim ve Lee [19] tarafından gerçekleştirilen çalışmada, yüksek oranda çeşitlilik sağlayan yeni bir mutasyon olasılığı ve bu mutasyonu kullanan donanım tabanlı bir GA önerilmiştir. Matematiksel problemler ve örüntü tanıma üzerine gerçekleştirilen deneylerin sonuçlara göre, geleneksel metotla karşılaştırıldığında, amaçlanan metodun yakınsanma kapasitesinin %34,5'e kadar iyileştiği ve hesaplama gücünün yaklaşık %40 civarında azaldığı ifade edilmiştir. Donanım olarak gerçekleştirilen GA işlemcisinde geçen süre, C dilindeki yazılımsal benzetimle karşılaştırıldığında %82,4 azalma görüldüğü belirtilmiştir. Çalışmada, Xilinx XC6VLX130 FPGA cihazı ve donanım dili olarak da Verilog kullanılmıştır.

Deliparaschos ve diğ. [20] tarafından gerçekleştirilen çalışmada, yazılım tabanlı GA ile karşılaştırıldığında, etkileyici bir hıza ulaşan GA IP çekirdeği tasarlandığı ve gerçekleştirildiği belirtilmiştir. Amaçlanan mimari FPGA üzerinde gerçekleştirilmiş ve donanım dili olarak VHDL kullanılmıştır. GA, gezgin satıcı problemine uygulanmıştır. Donanımsal GA için çalışma frekansı 92 MHz olarak seçilmiştir.

Allaire ve diğ. [2] tarafından gerçekleştirilen çalışmada, GA tabanlı yol bulma algoritmasının FPGA üzerinde gerçekleştirilmesi amaçlanmıştır. İnsansız hava araçlarının, beklenmedik durumlarla karşı karşıya kalmaları halinde, gerçek zamanlı

olarak yeniden planlama yapabilme yeteneğine sahip bir yol bulma algoritması olarak GA gerçekleştirilmiştir. Ancak, çalışma benzetim düzeyinde kalmıştır.

Bu tez çalışmasının amacı ve özgün katkıları üç ana başlık altında toplanmaktadır. İlki, otonom gezgin robotlar için yol bulma algoritmalarının incelenmesi ve daha etkili bir algoritmanın geliştirilmesi; ikincisi, yol bulma algoritmasının donanımsal olarak gerçekleştirilmesi ve üçüncü olarak da gerçekleştirilen sistemin gerçek bir laboratuvar ortamında gezgin bir robot üzerinde denenmesidir. Tez çalışması dahilinde gerçekleştirilen işlemler ve bunların katkıları aşağıda verilmektedir:

- Otonom bir gezgin robot için yol bulma problemi GA kullanılarak çözüldü. GA'da, robotun kullanacağı yol tanımına ilişkin farklı kromozom kodlama yöntemleri denendi ve aralarındaki performans farkları belirlendi. GA ile yol bulma problemine ilişkin yeni bir mutasyon operatörü geliştirildi ve bu yeni mutasyon operatörü ile daha önce literatürde sunulmuş olan farklı mutasyon operatörleri karşılaştırıldı. Karşılaştırmalar sonucunda yeni mutasyon operatörünün daha başarılı performans sergilediği sonucuna varıldı.

- Pioneer 3-DX gezgin robot ile GA'nın üzerinde çalıştığı bir bilgisayar arasında, RS-232 seri haberleşme standardı aracılığıyla haberleşme sağlandı ve GA'nın ürettiği yol üzerinde robot denendi. Robot ile haberleşmede Matlab programı kullanıldı. Görüntü almak amacıyla, laboratuvar tavanına bir kamera yerleştirilerek lokalizasyon gerçekleştirildi, robotun ortamdaki engelleri ve kendi konumunu belirleyebilmesi sağlandı. Böylece, lokalizasyon ve yol bulma işlemleri bütünleşik bir sistem olarak gerçek laboratuvar ortamında denendi.

- Bir bilgisayar üzerinde çalıştırılan GA, daha sonra FPGA üzerinde donanımsal olarak gerçekleştirildi. GA önce yarı yazılımsal yarı donanımsal bir şekilde hibrit olarak daha sonra tamamı donanımsal olarak tasarlandı. Hibrit yapıdaki GA'nın, sadece amaç fonksiyon hesaplayan bölümü donanımsal IP çekirdek (intellectual property core) ile, diğer operatörleri de Microblaze yazılımsal işlemci üzerinde çalışacak şekilde tasarlandı ve performans katkısı incelendi. Daha sonra da, sistem tamamen IP çekirdek ile donanımsal olarak gerçekleştirildi ve performans katkısı incelendi.

- Kameradan görüntü alıp işleyerek harita oluşturma, GA ile yol bulma, robot ile haberleşme ve robotun GA ile bulunan bir yol bilgisine göre ya da kullanıcı tarafından belirlenen bir yol koordinatlarına göre ilerleyebilmesini sağlayan Matlab-GUI (Graphical User Interface) tabanlı bir arayüz yazıldı.

- Tez çalışması kapsamında sistemin tamamı gerçek bir laboratuvar ortamında uygulandı. Böylece çalışmanın sadece benzetim ortamında kalmayıp, gerçek dünyada uygulanabilirliği gösterildi.

Bu tez çalışması altı ana bölümden oluşmaktadır. Giriş bölümünde otonom gezgin robotlar ve genel özellikleri hakkında bilgiler sunulmaktadır. Literatürde konu ile ilgili yer alan çalışmalardan, tez çalışmasının amacından ve özgün katkılarından bahsedilmektedir. Tezin diğer bölümlerinin kapsamı ise aşağıda verilmektedir.

Birinci bölümde, gezgin otonom robotlar için yol planlaması problemi tanıtılmakta ve bu planlama için geliştirilmiş olan algoritmalardan en çok kullanılanları kısaca açıklanmaktadır. GA'nın genel özelliklerinden bahsedilmektedir.

İkinci bölümde, GA'nın yol bulma problemlerine uyarlanması anlatılmaktadır. GA ile yol bulmada kullanılan farklı kromozom kodlama yöntemleri karşılaştırılmaktadır. Bu tez çalışmasının katkılarından birisi olan, yeni geliştirdiğimiz mutasyon operatörü tanıtılmakta ve çalışmada önerilen yeni mutasyon operatörü ile mevcut diğer mutasyon operatörlerinin performansları karşılaştırılmaktadır.

Üçüncü bölümde, FPGA teknolojisinden genel olarak bahsedilmekte, tez çalışmasında kullanılan FPGA geliştirme kartı ve özellikleri tanıtılmakta ve FPGA üzerinde gömülü sistem geliştirme konusu anlatılmaktadır. FPGA üzerindeki yazılımsal sanal mikroişlemci, IP çekirdekleri ve donanım tanımlama dili olan VHDL (Very high speed integrated circuit Hardware Description Language) hakkında bilgiler verilmektedir.

Dördüncü bölümde, tez çalışmasında kullanılan Pioneer 3-DX gezgin robot tanıtılmaktadır. Robotun genel özellikleri ve robot-bilgisayar haberleşmesinin nasıl sağlandığı açıklanmaktadır.

Beşinci bölümde, tez çalışmasıyla amaçlanan GA ile FPGA'de yol bulma uygulaması anlatılmaktadır. GA ile yol bulma algoritmasının FPGA üzerinde yazılımsal sanal mikroişlemci ile gerçekleştirilmesi, algoritmanın bir kısmının sanal mikroişlemci, bir kısmının da IP çekirdeği ile gerçekleştirilmesi ve tamamen IP çekirdeği ile gerçekleştirilmesi anlatılmakta ve bunların kıyaslamaları yer almaktadır.

Altıncı bölümde, tez çalışmasının sonuçları açıklanmakta ve ileriye dönük çalışmalar için önerilere yer verilmektedir.

1. YOL BULMA ALGORİTMALARI

Yol bulma, otonom gezgin robot uygulamalarında önemli bir rol oynamaktadır ve robot uygulamalarında en önemli problemlerden biridir. Yol bulma, engellerin olduğu bir ortamda, otonom robotun bir başlangıç noktasından hedef noktasına, engellere çarpmadan gidebilmesi için uygun bir yolun bulunması işlemidir.

Yol bulma problemlerine çözüm üretmek için Dijkstra [21], A* (A-yıldız) [22], D* (D-yıldız) [23], potansiyel alan metodu [24] gibi pek çok algoritma ve yöntem geliştirilmiştir. Bu algoritmalar, genellikle graf teorisi ve yapay zeka tabanlı algoritmalardır. Metotlar birbirleri ile karşılaştırıldığında, her bir metodun kendine özgü avantajları olmasına rağmen, dezavantajları da bulunmaktadır. Bunlar; yavaş hesaplama hızı, yerel optimuma takılma ve uyum eksikliği gibi dezavantajlar olabilmektedir. Yol bulma problemlerinde önemli olan, uygulamaya göre uygun algoritmayı seçmektir. Örneğin, en popüler metotlardan biri olan potansiyel alan metodu, statik ortamda bile sadece bir çözüm üretir, fakat bulunan çözüm her zaman en kısa yol olmayabilir [3]. Bununla birlikte bulunan çözümün yerel minimuma takılma ihtimali yüksektir [2], bu da robotun hedef konuma ulaşmasını engelleyebilmektedir.

Yol bulma algoritması gerçekleştirilirken özellikle dikkat edilmesi gereken durumlar bulunmaktadır. Bunlar;

- Yalnız bir tane çözüm var olduğunda, yol bulma algoritması bu çözümü bulabiliyor mu?
- Birden fazla çözüm var olduğunda, algoritma en iyi yolu bulabiliyor mu?
- Algoritma en iyi yolu bulmak için ne kadar zaman harcıyor?
- Arama işlemi ne kadar bellek gerektiriyor?

Yol bulma algoritmalarında uygulama çevresi ile ilgili bilgilerin bilinip bilinmemesi önemli bir konudur. Araştırmacılar, yol bulma problemlerinin çözümünde hangi metotları kullanacaklarını çeşitli faktörlere bağlı olarak belirlemektedirler. Örneğin; çevrenin dinamik ya da statik yapıda olması, yol bulma algoritmasının yerel ya da genel olması, engel sayısının sürekli değişmesi, engellerin hareketli ya da sabit olması, çevrenin önceden bilinmemesi ya da kısmen bilinmesi.

Statik çevre, otonom araç dışında hareket eden herhangi bir nesnenin olmadığı çevre olarak tanımlanmaktadır. Dinamik çevre ise statik çevrenin aksine hareket eden nesnelerin olduğu, değişim gösteren çevre olarak tanımlanmaktadır. Statik çevrede tüm engeller hakkında bir ön bilgi mevcuttur. Dinamik çevrede robot, engeller hakkında bilgi sahibi değildir ya da kısmen bilgi sahibidir ve tüm bilgileri keşif esnasında öğrenmektedir.

Yol bulma algoritmaları için çevrimiçi ya da çevrimdışı olmak üzere iki yaklaşım bulunmaktadır. Çevrimdışı algoritmalar, hedef çevrenin iyi bilindiği ve statik olduğu varsayılan durumlar için kullanılırken; çevrimiçi algoritmalar ise hedef çevrenin tam olarak bilinmediği, robotun çevredeki hareketi esnasında yapılan planlamalar için kullanılmaktadır [25].

Çevrimdışı algoritmalar çalışma başlamadan önce bütün çözümü bulurlar. Çevrimdışı algoritmalar bilgili arama ya da bilgisiz arama olarak iki bölüme ayrılmaktadır. Bilgisiz arama algoritmaları, arama için alana özgü herhangi bir bilgi (sezgisel fonksiyon) kullanmaz. Dijkstra algoritması iyi bilinen bir bilgisiz çevrimdışı arama algoritmasıdır. Öte yandan bilgili arama, alana özgü bilgi (sezgisel fonksiyon) kullanır. A* algoritması en iyi bilinen bilgili çevrimdışı arama algoritmasıdır. Yol bulma algoritmalarında sezgisel fonksiyon olarak, genellikle Euklid ya da Manhattan mesafesi kullanılmaktadır [26].

Dijkstra ve A* gibi çevrimdışı algoritmaların, çok fazla hesaplama zamanına ihtiyaç duymalarından dolayı, büyük dinamik çevrelerde kullanımı zordur [27]. Bu duruma bir çözüm olarak, çevrimdışı algoritmaların tekrarlı planlamalar yapması yerine bu algoritmaların artımsal olarak düzenlenmesi önerilmektedir. Artımsal algoritmalar, problemlere daha hızlı çözüm bulmak için, önceki aramalardan elde edilen bilgileri kullanan sürekli planlama teknikleridir [28]. D* , Focused D* [29] ve D* Lite [30]

iyi bilinen artımsal sezgisel arama algoritmalarıdır. Bu algoritmalarda, uygun bir başlangıç yolu çevrimdışı olarak üretilir ve daha sonra araç bu yolu izler. Yol boyunca, araç çevreyi gözlemler ve herhangi bir çevresel değişim belirlediğinde, mevcut çözümü kısmi olarak yeniden planlar [26]. Bu algoritmalar çoğu zaman etkili sonuçlar üretebilir, fakat bazen çevrede meydana gelen ufak bir değişiklikte tüm yolu yeninden planlamak zorunda kalabilir. Bu algoritmalar dışında; GA, rastgele ağaçlar algoritması ve olasılıksal yol haritası kullanan olasılıksal çevrimdışı algoritmaları da mevcuttur [27].

Çevrimiçi algoritmalarda ise, araç sürekli olarak, bir sonraki hareketini sınırlı bir zaman içerisinde planlar ve planı uygular. Algoritma, aynı yol üzerinde, gerçek zamanlı olarak plan ve işlem safhasını birleştirir. Gerçek zamanlı algoritmalardan bazıları; Real-Time A* (RTA*) [31], Learning Real-Time A* (LRTA*) [31], Real-Time Horizontal A* (RTHA*) [26] algoritmalarıdır. Gerçek zamanlı algoritmalar, en iyi çözümler için tasarlanmamıştır ve genellikle yol uzunluğu konusunda zayıf çözümler bulurlar [26].

Yol bulma algoritmaları genel ve yerel yol bulma olmak üzere de sınıflandırılabilirler [32-34]. Genel yol bulma algoritması, arama çevresi hakkında tam bilgi gerektirir ve arama çevresi statik yapıda olmalıdır. Bu yaklaşımda, algoritma, araç harekete başlamadan önce, başlangıç noktasından hedef noktaya kadar olan bütün yolu üretir. Diğer yandan, yerel yol bulma, yol bulma işlemi araç hareket halinde iken gerçekleştiriliyor demektir; diğer bir ifadeyle, algoritma çevresel değişmelere yanıt olarak yeni bir yol üretmeye yeteneklidir.

Yol bulma algoritmaları, kesinlik durumlarına göre de; tam ve sezgisel algoritmalar olarak iki kategoride sınıflandırılabilir. Tam algoritmaların amacı, eğer mevcut ise optimum sonucu bulmak veya mümkün bir kesin sonuç olmadığını ispatlamaktır. Diğer yandan sezgisel algoritmaların amacı, kısa zamanda kaliteli bir sonuç elde etmek için arama yapmaktır. Tam algoritmaların genellikle hesaplama maliyetleri yüksektir. Diğer taraftan, sezgisel algoritmalar da zor problemler için iyi yolu bulmakta yetersiz kalabilirler [4].

Yol bulma problemi, en kısa yolun ya da en az maliyetli yolun bulunması olarak düşünüldüğünde bir optimizasyon problemi olarak ta ele alınabilir. Son yıllarda,

sezgisel bir algoritma olan GA kullanılarak en kısa yol problemine ilişkin çözüm aramaları yapılmaktadır [35-38].

Tez çalışmasında, özellikle kısa yol bulma konusu üzerinde durulduğu için, en kısa yol algoritmalarından birkaç tanesi aşağıda açıklanmaktadır.

1.1. Dijkstra Algoritması

Dijkstra algoritması, yol bulma problemlerinde kullanılan algoritmalardan en çok bilinen ve kullanılanıdır. Algoritma 1959 yılında Edsger Dijkstra tarafından açıklanmıştır [21]. Dijkstra algoritması belirlenen bir başlangıç düğümünden, ortamdaki diğer bütün düğümlere giden en kısa yolların bulunması için kullanılmaktadır. Başlangıç düğümünden diğer tüm düğümlere giden bütün yolları sırayla kontrol eder. Bu nedenle, hedefe olan mesafe uzak olduğunda, uzun bir arama zamanına sahiptir ve bu verimsiz arama, bir dezavantaj oluşturmaktadır [39]. Dijkstra algoritmasının işlem basamakları aşağıda verilmektedir [40];

1. Her düğüm için geçici olarak bir mesafe değeri atanır. Bu değer başlangıç düğümü için 0, diğer tüm düğümler için sonsuzdur.

2. Ziyaret edilmemiş tüm düğümler işaretlenir. İlk düğüm dışındaki tüm düğümler arasından ziyaret edilmemiş bir düğüm listesi oluşturulur.

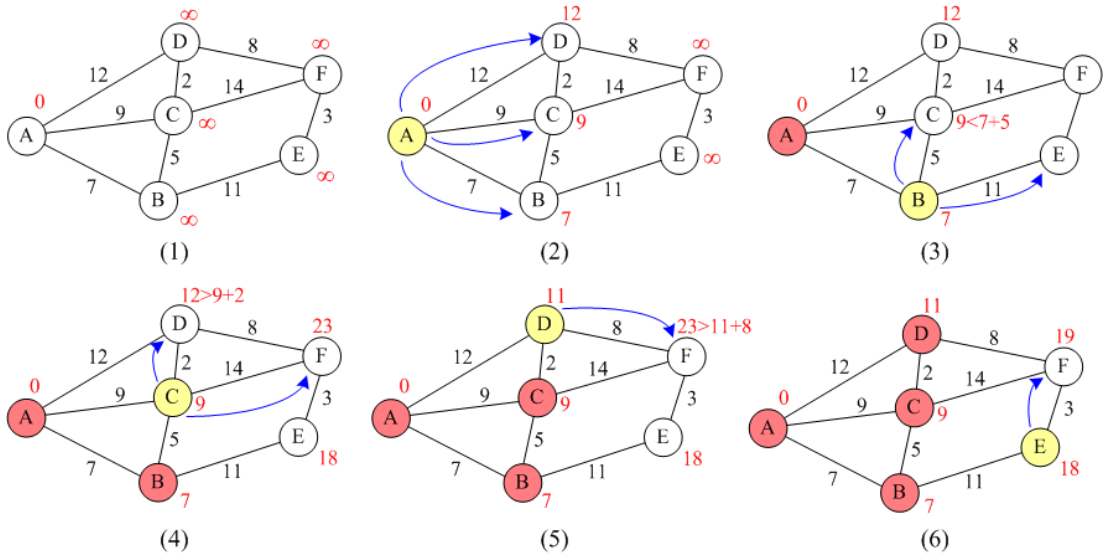
3. Mevcut düğüm için, ziyaret edilmemiş komşu düğümlerin hepsi göz önünde bulundurulur ve bunların geçici mesafeleri hesaplanır. Örneğin; herhangi bir A düğümü için mesafe 0 olarak verilsin. A düğümüne komşu bir B düğümü için de aralarındaki mesafe 7 olarak verilsin. Bu durumda A üzerinden B'ye gitme mesafesi 0+7=7 olacaktır. Bu mesafe, B üzerinde daha önceden kaydedilmiş geçici mesafeden daha az ise, yeni mesafe geçici mesafenin üzerine kaydedilir.

4. Mevcut düğümün tüm komşuları incelendiğinde, mevcut düğüm ziyaret edilmiş olarak işaretlenir ve ziyaret edilmemiş düğümler listesinden çıkartılır. Ziyaret edilen bir düğüm tekrar kontrol edilmez.

5. Hedef düğüm ziyaret edilmiş olarak işaretlendiyse veya ziyaret edilmemiş düğümler listesindeki düğümler arasında en küçük geçici mesafe sonsuz ise algoritma bitirilir.

6. Ziyaret edilmemiş düğümler listesindeki en küçük geçici mesafeli düğüm seçilerek mevcut düğüm olarak atanır ve 3. adıma geri dönülür.

Şekil 1.1'de, bir yol bulma probleminin Dijkstra algoritması ile aşama aşama çözümü örnek olarak gösterilmektedir. Örnekteki mesafe bilgilerine göre, A noktasından F noktasına gitmek için bulunan en kısa yol A-C-D-F düğümlerini takip etmektedir ve toplam mesafe 19 birimdir.



Şekil 1.1. Dijkstra algoritması örneği

1.2. A* (A-yıldız) Algoritması

A* algoritması, başlangıç düğümü ile bir hedef düğüm arasındaki en az maliyetli yolu bulmak için kullanılır. Bu algoritma ilk olarak 1968 yılında Peter Hart, Nils Nilsson, ve Bertram Raphael tarafından açıklanmıştır [22]. İlk olarak A algoritması olarak adlandırılsa da, daha sonraları kullanmış olduğu sezgisel yaklaşımdan dolayı A* olarak adlandırılmıştır. En kısa yol hesaplanırken; toplam maliyet fonksiyonu f(x), uzaklık fonksiyonu g(x) ile sezgisel fonksiyon h(x) toplanarak elde edilir (Denklem 1.1). Uzaklık fonksiyonu g(x), başlangıç düğümü ile üzerinde bulunulan mevcut düğüm arasındaki gelinen yol uzunluğu olarak tanımlanmaktadır. h(x) sezgisel fonksiyonu ise mevcut düğüm ile bitiş düğümü arasındaki düz bir çizginin

uzunluğunu temsil eder. Bu, mevcut düğüm ile hedef arasındaki kuş bakışı mesafe olarak da adlandırılmaktadır. Algoritma başlangıç düğümünden başlayarak, en düşük f(x) değerli düğüme doğru yayılır. Dijkstra algoritmasındaki mevcut düğümden dışarıya doğru bütün yönlerde ilerlemek yerine, hedef düğüme doğru yayılım gösterir;

$$f(x) = g(x) + h(x) \tag{1.1}$$

A* algoritmasının işlem basamakları aşağıda verilmektedir [41];

1. Başlangıç düğümü olarak A düğümü açık listeye eklenir. Açık liste kontrol edilmesi gereken düğümleri içermektedir.

2. Başlangıç düğümünün komşuları incelenir. Komşu düğümlerden, örneğin engel gibi yasak olanlar var ise bunlar göz ardı edilerek, yasak olmayanlar açık listeye eklenir. Eklenen bu düğümlerin her biri A düğümünün "çocuk düğümleri" olarak kaydedilir.

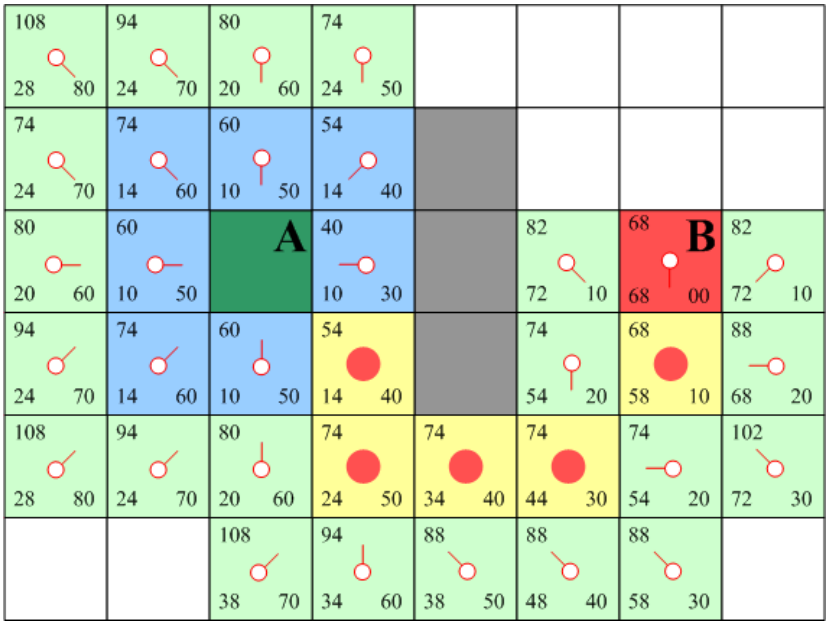
3. A düğümü açık listeden çıkartılıp kapalı listeye eklenir. Artık A düğümü tekrar incelenmeyecektir.

4. Açık listedeki en düşük değerlikli düğüm yeni mevcut düğüm olarak işaretlenir. Bir komşu düğüm önceden açık listeye eklenmiş ise, o düğümün önceki ve şimdiki g(x) maliyetlerine bakılır ve maliyeti düşük olan yol bilgisi korunur.

5. Bu şekilde, hedef düğüme erişilene kadar 2-4 adımları tekrar edilir.

Şekil 1.2, A* algoritması kullanılarak, A başlangıç düğümünden B hedef düğümüne gitmek için gerekli olan adımları göstermektedir. Şekilde, gri renkli kareler yasak olan düğümleri, beyaz renkli kareler hiç kontrol edilmemiş olan düğümleri, mavi renkli kareler kapalı listedeki düğümleri, açık yeşil renkli kareler açık listede yer alan düğümleri ve sarı renkli kareler ise A noktasından B noktasına gitmek için gereken en kısa yolu ifade etmektedir. Her bir kare üzerinde kendi ebeveynini gösteren işaretçi bulunmaktadır. Yine şekilde görüldüğü gibi, maliyet fonksiyonları kare üzerinde belirtilmektedir. f(x) üst solda, g(x) sol altta ve h(x) değeri ise sağ alta gösterilmektedir. Karelerin kenar uzunlukları 10'ar birim ve hesap kolaylığı

açısından $\sqrt{2} \cong 1.4$ olarak alınmaktadır. h(x) sezgisel fonksiyonunun hesaplanmasında Manhattan mesafesi kullanılmaktadır.



Şekil 1.2. A* algoritması örneği [41]

1.3. D* (D-yıldız) Algoritması

D* algoritması, ilk olarak 1994 yılında Anthony Stentz tarafından açıklanmıştır [23]. D* ismi, Dinamik A* algoritmasından gelmektedir. D*, D*-Lite, Focused D* gibi algoritmaların hepsi sezgisel varsayıma dayanan yol bulma algoritmalarıdır. Otonom gezgin araç, kısmi olarak bilinen ya da hiç bilinmeyen bir arazi üzerinde varsayımlar yaparak, verilen hedefi bulmak zorundadır. Araç, arazinin bilinmeyen parçası hakkında varsayımda bulunur. Örneğin, başlangıçta hiçbir engel olmadığını varsayar. Bu ve benzeri varsayımlar altında, mevcut koordinattan hedef koordinata en kısa yolu bulmaya çalışır. Araç, önceden bilinmeyen engellerin artık biliniyor olması gibi yeni bir harita bilgisi keşfettiğinde, bu bilgiyi kendi haritasına ekleyerek güncelleme yapar ve gerekli ise mevcut konumundan hedef konuma en yakın yolu yeniden planlar. Hedef konuma ulaşana kadar bu işlemleri tekrarlar veya hedef konuma ulaşılamayacağını belirler. Araç, bilinmeyen bir araziye gelindiğinde, sık sık yeni engellerle karşılaşabilir ve yeni planlamalar yapmak zorunda kalabilir. Bu durumda, planlamanın çok hızlı bir şekilde yapılması gerekmektedir. Sezgisel arama algoritmaları, mevcut durum için önceki problemlerden elde edilen deneyimleri kullanarak aramayı hızlandırabilirler.

1.4. Genetik Algoritmalar

GA, temel ilkeleri John Holland [42] tarafından ortaya atılmış, genetik bilimine dayanan bir optimizasyon tekniğidir. GA, iteratif çalışan ve rastlantısal olarak arama yapan bir algoritmadır. Genetik alanındaki çaprazlama, mutasyon, doğal seçilim gibi biyolojik süreçlerden esinlenerek geliştirilmiş ve bu süreçleri matematiksel olarak modelleyerek fonksiyonları optimize eden bir algoritmadır. GA, doğadaki "güçlü olan birey hayatta kalır [43]" prensibine bağlı kalarak, nüfustaki iyi bireylere yaşama şansı vermekte ve nüfusu oluşturan bireylerin, yani aday çözümlerin kuşaktan kuşağa iyileşmesini sağlamaktadır. Bu özelliğiyle GA, tamamen rastgele bir arama değil, geçmiş kuşaklardaki genetik verilere dayalı olarak bir arama gerçekleştirmektedir.

Şekil 1.3'de akış diyagramı gösterilen GA'nın çalışma adımları kısaca aşağıdaki gibi açıklanabilir;

- Çözüm uzayındaki olası tüm çözümler bir dizi olarak kodlanır. Bu kodlanmış diziler GA'nın kromozomlarını veya diğer ismiyle bireylerini ifade eder. Kromozomu oluşturan genler, çözümü oluşturan parametrelerdir.

- Kodlanmış kromozomlar içinden rastgele bir grup çözüm alınarak, başlangıç nüfusu oluşturulur. Nüfus içerisindeki kromozom sayısı nüfus büyüklüğünü gösterir. GA için en yaygın kullanılan kromozom kodlama biçimleri ikili (binary) kodlama ve gerçel (real) sayı kodlamalarıdır.

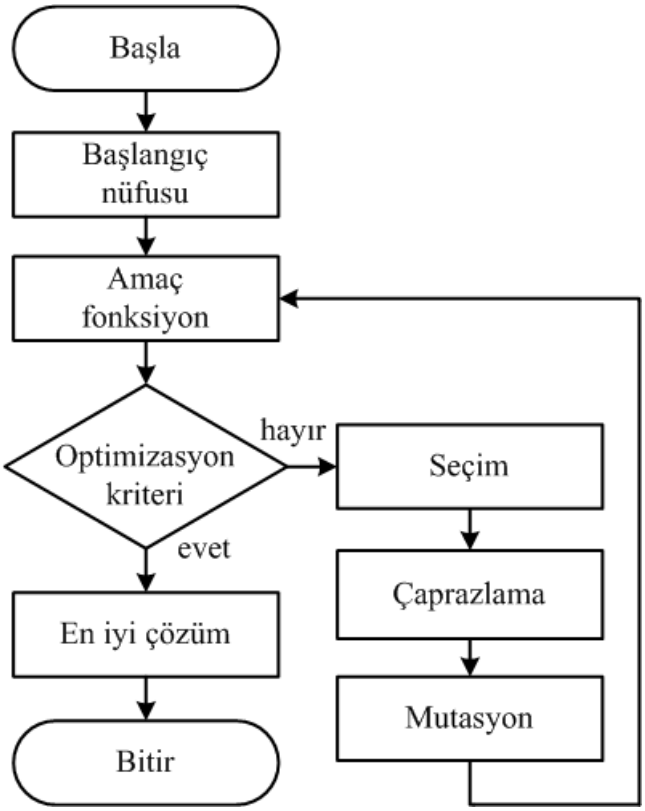
- Nüfus içerisindeki her bir bireyin amaç fonksiyon değeri hesaplanır. GA'nın amacı en iyi amaç fonksiyonuna sahip bireyi bulabilmektir.

- Amaç fonksiyon değeri, herhangi bir bireyin çözüm için uygunluğunun bir ölçüsüdür. Amaç fonksiyon değeri, problemin başında belirlenen optimizasyon kriterini sağlıyor ise algoritma sonlandırılır, aksi halde algoritma çalışmaya devam ettirilir.

- Amaç fonksiyon değerlerine göre bireylerin seçilme olasılıkları belirlenir. Bireyler, seçilme olasılıklarına göre rastgele olarak seçim işlemine tabi tutulur. Seçim işlemi ile, problemin çözümü için en çok uyum sağlayan bireylerin genlerinin gelecek kuşaklara aktarılması sağlanmış olur.

- Çaprazlama ve mutasyon işlemleri uygulanarak daha iyi ve çözüm için daha uyumlu bireyler bulunur. Çaprazlama ve mutasyon işlemleri, problemin başında çözüm parametresi olarak verilen olasılıklar dahilinde gerçekleştirilir.

- Yeni bireylerin amaç fonksiyon değerleri hesaplanır.



Şekil 1.3. GA'nın akış diyagramı

GA, paralel ve genel arama yapan algoritmalardır. Aynı anda arama uzayının birçok yerinde paralel olarak arama yapabilmekte ve çözüme daha hızlı bir şekilde ulaşabilmektedir. GA problemlere tek bir çözüm üretmez. Bunun yerine farklı çözümlerin olduğu bir çözüm kümesi üretir. Bu sayede, arama alanındaki birçok nokta kontrol edilebilmekte ve çözüme ulaşma olasılığı artmaktadır. GA sezgisel bir algoritma olduğu için problemlerin çözümünde her zaman en iyi çözümü bulamayabilir, fakat geleneksel yöntemlerle çözülemeyen ya da çözüm süresi uzun olan problemler için en iyi çözüme yakın çözümler üretebilirler.

GA'nın en büyük avantajı, diğer sezgisel algoritmaların aksine daha karmaşık problemleri çözebilmeleridir. GA'yı diğer algoritmalardan ayıran en önemli özellikler aşağıdaki gibi sıralanabilir [43];

- GA tek bir çözümle değil, çözümlerden oluşan bir nüfus ile arama işlemine başlar. Bir çözüm başarısız olmuş ise, bunun yerine nüfus içerisindeki diğer başarılı çözümlerin özelliklerinden yararlanarak yeni bir çözüm üretir. Böylece, algoritmaya paralel çözüm arama yeteneği katılmış olur. Paralellik nedeniyle arama uzayını hızlı arar.

- GA'da problem değişkenleri veya parametreleri doğrudan kullanılmazlar. Bunların yerine problemin kodları kullanılır. Çözümün kendisi ile değil, çözüm grubunun kodlarıyla çalışır.

- GA türev bilgisi ve çözüm uzayı hakkında başlangıç bilgisi gerektirmeyen, rastlantısal yapısı gereği çözüm uzayının tamamında arama yapabilen bir algoritmadır. Klasik yöntemlerin aksine GA gürültülü, süreksiz ve zamanla değişen fonksiyonları da optimize edebilirler.

Bu tez çalışmasında, yol bulma algoritması olarak GA kullanılmış olup algoritmanın probleme uygulanışı ileriki bölümlerde detaylıca anlatılmaktadır.

2. OTONOM GEZGİN ROBOTLAR İÇİN GENETİK ALGORİTMA TABANLI YOL BULMA

GA paralel arama özelliği sayesinde, aynı anda çalışma ortamının tamamında arama yapabilmekte ve böylece çözüme daha hızlı bir şekilde ulaşabilmektedir. Son yıllarda GA, güçlü optimizasyon yeteneğinden dolayı optimum yol bulmada yaygın bir şekilde kullanılmaktadır [9]. GA ile yol planlaması problemine çözüm ararken, çalışma ortamının belirlenmesi, bireylerin kodlanması ve GA operatörlerinin probleme uyarlanması en başta yapılması gerekenlerdir. GA'yı probleme uyarlamak için yapmış olduğumuz çalışmalar aşağıda detaylıca verilmektedir.

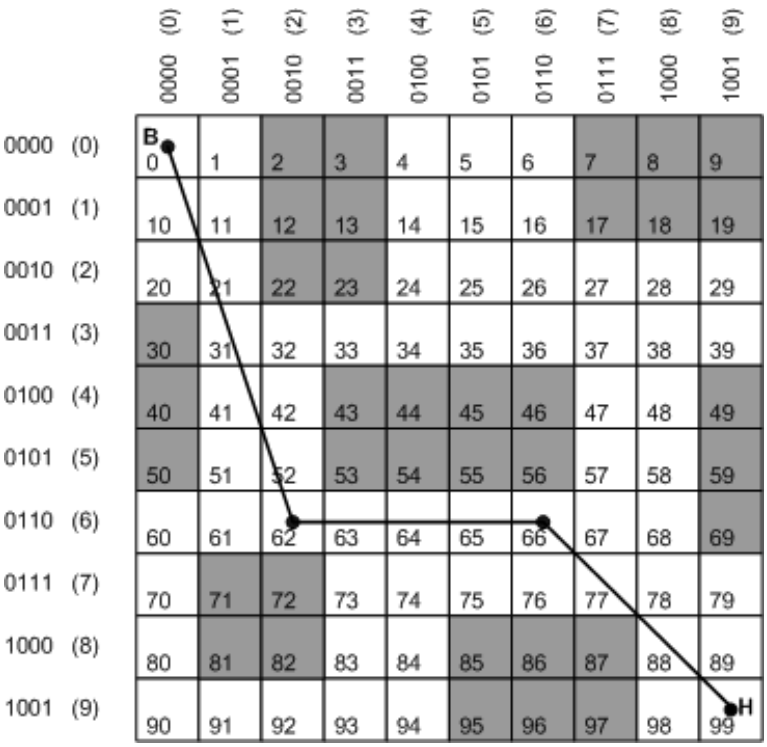
2.1. Kodlama

GA'da nüfusun her bir üyesi bir kodla temsil edilir [44]. GA için ikili kodlama, gray kodlama, tam sayı kodlama, gerçel sayı kodlama, vektör kodlama ve ağaç kodlama gibi çeşitli kodlama biçimleri mevcuttur. Hangi tür kodlamanın kullanılacağı problemin yapısına uygun olarak belirlenmektedir. Sıklıkla kullanılan kodlama biçimleri ikili kodlama ve gerçel sayı kodlamalardır [45].

2.1.1. Çalışma ortamının belirlenmesi

Yol bulma metotlarının pek çoğu, çalışma alanını temsil etmek için hücre tabanlı bir model kullanır [46]. Literatürde, engellerin çalışma alanında ifade edilmesi ve yol mesafesinin hesaplanmasında hücre tabanlı gösterimin uygulanmasının daha kolay olduğu saptanmıştır. Hücre tabanlı çalışma alanları; koordinat düzlemi [4,46] şeklinde veya seri numaralı [1, 37, 38] bir şekilde temsil edilebilmektedir. Koordinat düzleminde, her bir hücrenin koordinatlarını gösteren (x,y) çiftleri hem ikili hem de tamsayı şeklinde ifade edilebilir. Seri numaralı temsil yöntemi de literatürde yaygın olarak kullanılan bir yöntemdir. Şekil 2.1'de çalışma ortamının hem koordinat düzlemi (ikili ve tamsayı) hem de seri numaralı temsili gösterilmektedir. Gri renkli hücreler engelleri, boş hücreler robotun gidebileceği uygun alanları temsil etmektedir.

Şekilde başlangıç ve hedef noktaları arasında robotun gidebileceği engelsiz örnek bir yol gösterilmektedir.



Şekil 2.1. Çalışma alanının (x,y) koordinatı ve seri numaralı temsili

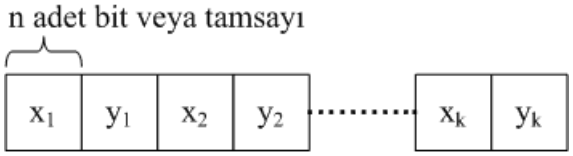
2.1.2. Kromozom kodlama yöntemleri

Yol bulma problemlerinde kromozomlar, başlangıç ile hedef nokta arasında, üzerinden geçilen yolları tanımlamaktadır. Oluşturulan her bir kromozom hedefe giden engelli ya da engelsiz bir yolu ve kromozom içerisinde yer alan her bir gen de yol üzerindeki bir adımı ifade etmektedir. Kromozomları oluştururken, kullanılan çalışma alanı gösterim şekline bağlı olarak farklı kodlama yöntemleri kullanılabilmektedir.

2.1.2.1. Koordinat düzlemi şeklinde kodlama

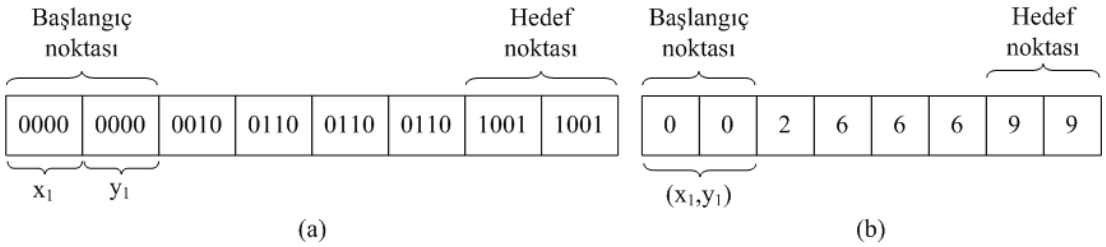
Koordinat düzleminde yer alan hücrelerin koordinatları, hem ikili hem de tamsayı olarak kodlanabilen (x,y) çiftleridir. İkili kodlamada, hücrenin koordinatları ikili sayı sistemine göre kodlanmaktadır. Örneğin, 16x16 olan bir çalışma alanı için, ikili kodlamada her bir gen 4 bit ile ifade edilirken, tam sayı kodlamada ise her bir gen [0,15] arasında bir tamsayı olacak şekilde kodlanırlar. Şekil 2.2'de, (x_1,y_1)

noktasından başlayıp (x_k,y_k) noktasında biten yol koordinatlarına sahip bir kromozom gösterilmektedir. "k" burada adım sayısını ifade etmektedir. Her bir x veya y koordinatı n adet bit olarak kodlanabileceği gibi tamsayı olarak da kodlanabilir.



Şekil 2.2. Koordinat çiftlerinden (x,y) oluşan bir kromozom yapısı

Örnek olarak Şekil 2.3, Şekil 2.1 ile gösterilen çalışma ortamındaki yolun ikili ve tamsayı olarak kodlanmış kromozomlarını göstermektedir. Şekilden de anlaşılacağı üzere, gezgin robot (0,0) noktasından harekete başlayıp, sırasıyla (2,6) ve (6,6) noktalarından geçerek (9,9) hedef noktasına varacaktır.

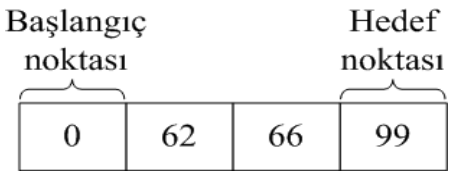


Şekil 2.3. İkili (a) ve tamsayı (b) kodlanmış kromozom örnekleri

2.1.2.2. Seri numaralı kodlama

Seri numaralı kodlamada, kromozomdaki genler ondalık tamsayılardan oluşmaktadır. Bu gösterimde genler seri numaralar şeklinde ifade edilir. Her bir gen çalışma alanının toplam hücre sayısı kadar değer alabilir. Boyutları NxN olan bir çalışma alanı için toplam hücre sayısı N^2 dir. Bu durumda her bir genin alabileceği değer [0, N^2-1] arasında olacaktır. Seri numaralı kullanım daha kısa ve hafızada daha az yer kaplar. Bu yüzden, yapılan çalışmalarda seri numaralı kodlama kullanımı yaygınlaşmaktadır. Şekil 2.4'de, Şekil 2.1 ile gösterilen çalışma ortamındaki yolun seri numaralı olarak kodlanmış kromozom örneği gösterilmektedir. Bu yol bilgisine göre, gezgin robot 0 numaralı hücreden başlayıp, sırasıyla 62 ve 66 numaralı hücrelerden geçerek 99 numaralı hücreye ulaşacaktır.

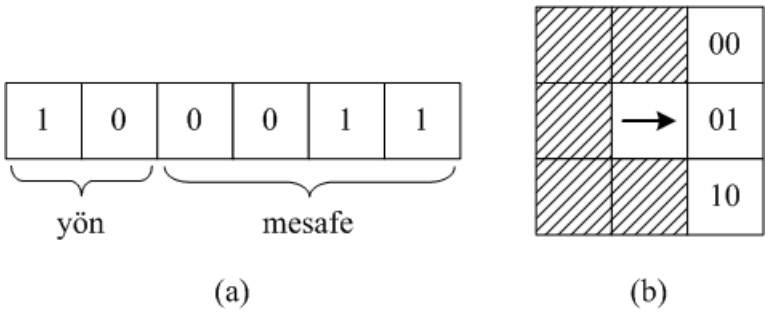
Kromozomların uzunluğu veya içerdiği gen sayısı, çalışma ortamlarına ve ortamdaki engellerin durumuna göre değişmektedir. Literatürde yapılan çalışmalarda kromozom uzunluğu genellikle sabit olarak alınmıştır. Fakat, bazı uygulamalarda değişken uzunlukta kromozomlar da kullanılmaktadır. Özellikle, dinamik yapıdaki çalışma alanları için değişken uzunlukta kromozomların daha etkili olduğu belirtilmektedir [3].



Şekil 2.4. Seri numaralı kodlanmış kromozom örneği

2.1.2.3. Yön-mesafe kodlama

Kromozomlar oluşturulurken, üzerinden geçilen hücrelerin koordinat bilgilerini kullanmak yerine, bir sonraki gidilecek hücrenin mevcut hücreye göre yön ve mesafe bilgileri de kullanılabilmektedir. Fries [47] yapmış olduğu çalışmasında, kromozom içerisinde yön ve mesafe bilgisini kullanmıştır.



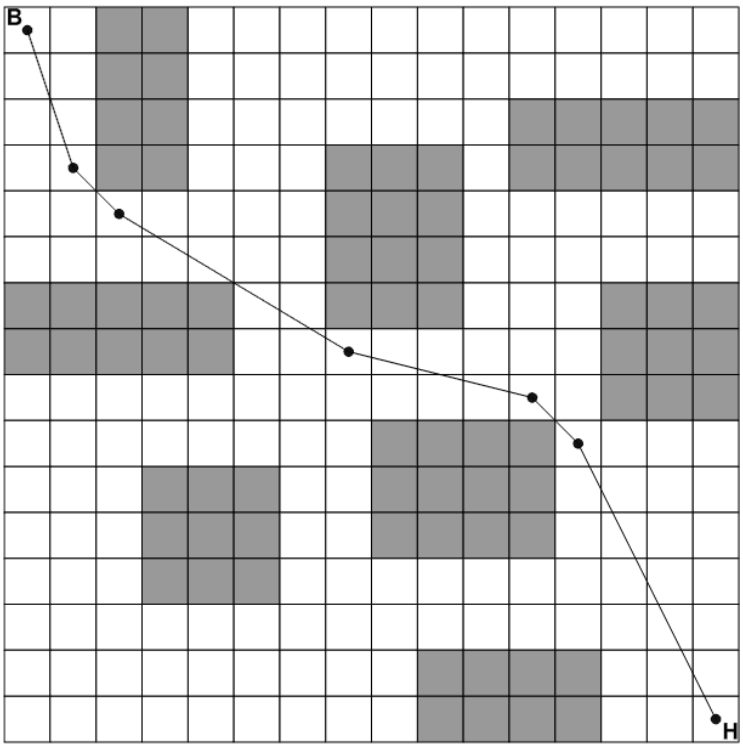
Şekil 2.5. Yön-mesafe kodlama yöntemi [47]

Şekil 2.5a'da görüldüğü gibi, ikili kodlanmış bir genin ilk iki biti yön bilgisi için, sonraki dört biti ise mesafe bilgisi için kullanılmaktadır. Yön bilgisi, Şekil 2.5b'de doğrultusu verilen bir gezgin robot için; "00" kodu 45 derece sola dönüş için, "01" kodu ileri yön için ve "10" kodu 45 derece sağa dönüş için tanımlanmaktadır. Mesafe için ise ikili kodun ondalık karşılığı bulunarak hesaplama yapılmaktadır. Şekildeki örnek için, gezgin robot 3 hücre ilerleyecektir. Bu şekilde, yön ve mesafe bilgileri arka arkaya eklenerek bir kromozom oluşturulmaktadır.

2.1.3. Kromozom kodlama yöntemlerinin karşılaştırılması

Tez çalışmasında ikili, tamsayı ve seri numaralı kodlama yöntemleri için örnek uygulamalar yapılarak kodlama metotlarının birbirlerine göre avantaj ve dezavantajları belirlenmeye çalışıldı. Aşağıda bu amaçla yaptığımız iki örnek çalışmanın detayları verilmektedir.

Şekil 2.6'da gösterilen çalışma ortamı için, yol bulma problemi GA ile Matlab yazılımı kullanılarak çözüldü. Çevre, 16x16'lık bir hücresel çalışma alanından ve bu alan üzerinde 8 adet engel bölgesinden oluşmaktadır. GA'nın kromozomlarını üretmek için ikili, tamsayı ve seri numaralı kodlama yöntemleri denendi. GA parametrelerinden nüfus büyüklüğü 60, çaprazlama oranı 1 ve mutasyon oranı 0,1 olarak belirlendi. Her bir kodlama yöntemi için GA 10 sefer çalıştırıldı.



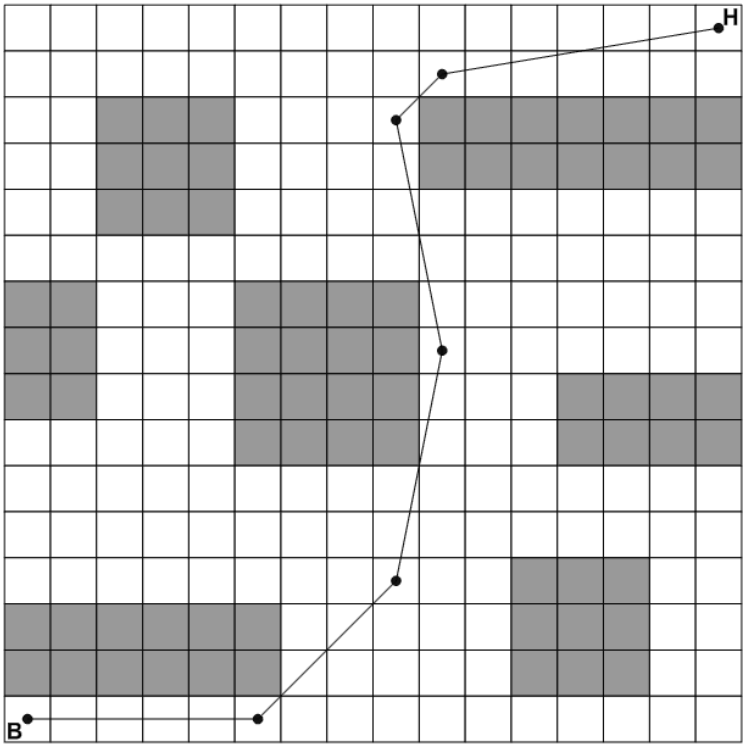
Şekil 2.6. Kromozom kodlama yöntemleri karşılaştırma için çalışma alanı-1

Elde edilen ortalama yol mesafeleri, jenerasyon sayıları ve çözüm bulma süreleri Tablo 2.1'de verilmektedir. Tablodaki verilere göre, tamsayı kodlama yöntemi en kısa yolu bulmak için daha uygundur ve seri numaralı kodlama yöntemi de diğer kodlama yöntemlerinden daha kısa sürede uygun bir çözüm bulabilmektedir.

Tablo 2.1. Kodlama yöntemlerini karşılaştırmak için 1. örnek sonuçları

	İkili	Tamsayı	Seri numaralı
Amaç fonksiyon	28,40	25,07	26,30
Jenerasyon sayısı	153	75	54
Çözüm süresi (sn)	2,7	1	0,7

Kromozom kodlama yöntemlerini ikinci bir örnekle karşılaştırmak için farklı bir çalışma alanı kullanıldı. Şekil 2.7'de gösterilen çalışma ortamı 16x16 boyutunda ve 7 adet engel bölgesinden oluşmaktadır. İlk karşılaştırmada kullanılan GA parametre değerleri bu örnekte de kullanıldı ve her bir kodlama yöntemi için GA 10 sefer çalıştırıldı.



Şekil 2.7. Kromozom kodlama yöntemleri karşılaştırma için çalışma alanı-2

Elde edilen ortalama yol mesafeleri, jenerasyon sayıları ve çözüm bulma süreleri Tablo 2.2'de verilmektedir. Tablodaki verilere göre, ilk uygulamada olduğu gibi tamsayı kodlama yöntemi en kısa yolu bulmak için daha uygundur ve seri numaraları kodlama yöntemi de diğer kodlama yöntemlerinden daha kısa sürede uygun bir çözüm bulabilmektedir.

Tablo 2.2. Kodlama yöntemlerini karşılaştırmak için 2. örnek sonuçları

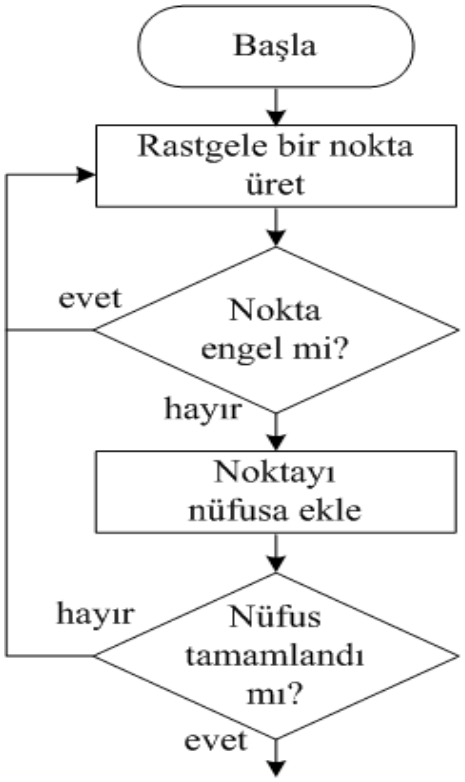
	İkili	Tamsayı	Seri numaralı
Amaç fonksiyon	31,9	29,6	30,6
Jenerasyon sayısı	101	44	32
Çözüm süresi (sn)	1,5	0,55	0,47

2.2. Başlangıç Nüfusunun Oluşturulması

GA çalışmalarında, başlangıç nüfusundaki bireyler genellikle rastgele üretilmektedir. Bu yöntem, nüfusu oluşturmanın basit ve pratik bir yoludur. Fakat, rastgele oluşturulan kromozomlar engel üzerinden geçen ve uygun olmayan yol bilgilerini içerebilmektedir. Hatta, engel içeren yol bilgilerine sahip iki bireyin çaprazlanması ile yine engel içeren yola sahip bir birey meydana gelme olasılığı oldukça fazladır. Böyle bir durumda da, algoritmanın arama hızı düşecek ve yol bulma süresi artacaktır. Nüfus rastgele oluşturulsa dahi, çok iyi planlanmış genetik operatörler sayesinde bu olumsuz durum telafi edilebilir ve iyi bir çözüm bulunabilir.

Bu problemi çözmek için, başlangıç nüfusu oluşturulurken, engel içeren noktaların kromozomlarda yer alması engellenebilir ve tüm yollar engel üzerinden geçmeyen uygun noktalardan rastgele olarak seçilebilir. Yao ve Ma [37], Li ve diğ. [38] çalışmalarında, engelli noktaların oluşmasını önleyen bir başlangıç nüfusu hazırlamışlardır. Şekil 2.8'de, başlangıç nüfusu oluşturulurken engel kontrolü yapan bir algoritma gösterilmektedir.

Tez çalışmasında, başlangıç nüfusu oluşturulurken önce rastgele sonra da engel kontrolü yapılarak engelsiz bir nüfus oluşturuldu. Her iki başlangıç nüfusu da, ayrı ayrı GA ile Matlab yazılımı kullanılarak yol bulma problemine uygulandı. 10'ar kez çalıştırılarak ortalama değerleri alındı. Aynı koşullar altında elde edilen amaç fonksiyon, jenerasyon sayısı ve çözüm süresi ortalama değerleri Tablo 2.3'de verilmektedir. Tabloda görüldüğü gibi, nüfus oluşturma işleminde engelli noktaların önlenmesi, yol bulma zamanını azaltmakta ve algoritmayı daha başarılı hale getirmektedir.



Şekil 2.8. Engelsiz başlangıç nüfusu oluşturma

Tablo 2.3. Rastgele ve engel kontrollü başlangıç nüfuslarının GA performansına etkileri

Başlangıç nüfus metodu	Rastgele	Engel kontrollü
Amaç fonksiyon	28,34	28,29
Jenerasyon sayısı	81	36
Çözüm süresi (sn)	0,87	0,49

2.3. Seçim

GA, doğal seçilim prensibini esas alarak, iyi bireylerin yaşaması için bunların özelliklerinin yeni kuşaklara aktarılmasını sağlar. Nüfus içerisindeki bireylerin hangilerinin ve ne derecede iyi olduklarının tespiti için bir seçim mekanizması kullanır. Seçim mekanizması üç aşamadan oluşmaktadır. İlk aşamada tüm bireylerin ne derecede iyi olduklarını gösteren amaç fonksiyon değerleri bulunur. İkinci aşamada bireylere, kendisinin ve nüfus içerisindeki diğer bireylerin amaç fonksiyon değerlerine bağlı olarak uygunluk değerleri atanır. Üçüncü aşamada ise bireyler

uygunluk değerlerine göre seçilerek, yeni bireyler üretmek üzere eşleştirme havuzuna atılırlar [48].

2.3.1. Amaç fonksiyon

Amaç fonksiyon, nüfus içerisindeki bireylerin genetik gelişimini sağlayan mekanizmanın temelini oluşturmaktadır. GA ile çözümü aranan problem arasındaki tek bağlantıdır. Amaç fonksiyon, bireyi bir giriş olarak alır ve çözümü ne derece karşıladığının ölçüsünü gösteren bir sayı üretir. Ürettiği sayının aralığı problemden probleme değişmektedir [48].

Yol bulma problemlerinde amaç, engellere çarpmadan başlangıç ile hedef arasındaki en uygun yolu bulmaktır. Burada uygunluk ölçüsü; en kısa yol, minimum zaman veya en az enerji tüketimi olabilir. Yol bulma çalışmalarında, genellikle en kısa yol amaç fonksiyon olarak ele alınmaktadır. Bu tez çalışmasında da, amaç fonksiyon en kısa yol olarak belirlenmiştir. Denklem (2.1) ve (2.2) çalışmada kullanılan amaç fonksiyonu göstermektedir.

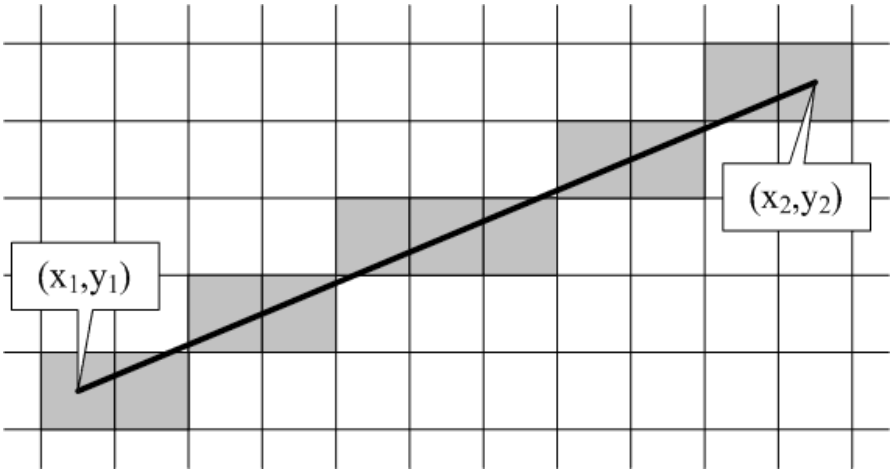
$$f = \begin{cases} \sum_{i=1}^{n-1} d(p_i, p_{i+1}) & \text{, engel yoksa} \\ \sum_{i=1}^{n-1} d(p_i, p_{i+1}) + \sum_{j=1}^{m} ceza & \text{, engel varsa} \end{cases} \tag{2.1}$$

$$d(p_i,p_{i+1}) = \sqrt{(x_{(i+1)} - x_i)^2 + (y_{(i+1)} - y_i)^2} \tag{2.2}$$

Burada; f amaç fonksiyonu, p kromozomdaki genleri, n kromozomdaki gen sayısını, d iki nokta arasındaki mesafeyi, m iki nokta arasındaki engel sayısını ifade etmektedir. Denklem (2.1)'de görüldüğü gibi, amaç fonksiyon kromozomda yer alan genler arasındaki mesafelerin toplamı olarak hesaplanmaktadır. İki nokta arasında herhangi bir engel var ise amaç fonksiyona ceza puanı eklenmektedir. En iyi birey amaç fonksiyon değeri en küçük olan birey olarak tanımlandığı için, engel üzerinden geçen birey cezalandırılarak seçilme olasılığı azaltılmaktadır. Ceza puanı çalışma alanındaki en uzak mesafeden daha büyük seçilmelidir. Tez çalışmasında ceza puanı olarak 100 değeri kullanıldı.

Literatürdeki bazı yol bulma çalışmalarında, gezgin robotun sadece yatay ve dikey, bazılarında da ilaveten 45° çapraz açılarla hareket etmesine izin verilmektedir. Bu tez çalışmasında ise, robotun her yönde ve 0-360° açılarla hareket edebilmesi göz önünde bulunduruldu. Çalışmada, noktalar arasındaki doğrusal mesafenin hangi hücrelerden geçtiği, Bresenham [49] doğru çizme algoritması kullanılarak belirlendi.

Bresenham algoritması diğer algoritmaların aksine, ondalık sayılar yerine sadece tam sayılarla işlem yapmaktadır. Tam sayılarla işlem yapılması algoritmanın daha hızlı çalışmasını sağlamaktadır. Şekil 2.9'daki örnekte, Bresenham algoritmasına göre iki nokta arasında çizilen doğruyu ve bu doğrunun üzerinden geçtiği hücreler gösterilmektedir.



Şekil 2.9. Bresenham algoritması ile iki nokta arasında çizilen çizgi

2.3.2. Uygunluk değeri

GA'da bireylerin amaç fonksiyon değerleri arasındaki dağılım farklılıklarını gidermek için, amaç fonksiyon değerleri, uygunluk değerlerine dönüştürülürler. Uygunluk değerine dönüştürme işlemi, nüfus içerisindeki tüm bireylerin seçilme olasılıklarını yeniden düzenleme anlamına gelmektedir. Bireylere uygunluk değeri atanmasında, kendisinin ve nüfus içerisindeki diğer bireylerin amaç fonksiyon verileri kullanılır. GA'da yaygın olarak kullanılan uygunluk değeri atama yöntemleri arasında orantısal uygunluk atama ve sıra tabanlı uygunluk atama yer almaktadır. Bu tez çalışmasında, sıra tabanlı uygunluk atama yöntemi kullanıldı. Her bir bireye, amaç fonksiyon değeri sıralamasına göre eşit aralıklı uygunluk değeri atandı.

2.3.3. Seçim

Bireyler uygunluk değerlerine göre, yaygın olarak kullanılan rulet tekerleği seçim yöntemi ile seçilerek, yeni birey üretmek üzere eşleştirme havuzuna atılırlar. Seçim, eşleştirme havuzunda kullanılacak birey sayısına ulaşılıncaya kadar tekrarlanan bir işlemdir. Rastlantısal bir yöntem olan rulet tekerleği seçiminde, bireylerin seçilme olasılıkları sürekli bir çizgi üzerine parçalar halinde yerleştirilirler. Bireylerin parça boyları, uygunluk değerlerine bağlı olarak hesaplanan seçilme olasılıklarıyla eşit uzunluktadır. Parçaların toplam uzunluk aralığında rastgele bir sayı üretilir ve bu sayı hangi parça üzerine düşüyor ise o parçaya sahip birey seçilmiş olur.

Tablo 2.4'de, 10 bireyden oluşan, maksimum uygunluk değeri 10 olan ve uygunluk değerleri sıra tabanlı uygunluk atamasıyla belirlenmiş olan bir nüfus örneği görülmektedir. Tabloda yer alan seçilme olasılıkları Denklem (2.3) ile belirlenmektedir.

$$S_i = \frac{U_i}{\sum U} \tag{2.3}$$

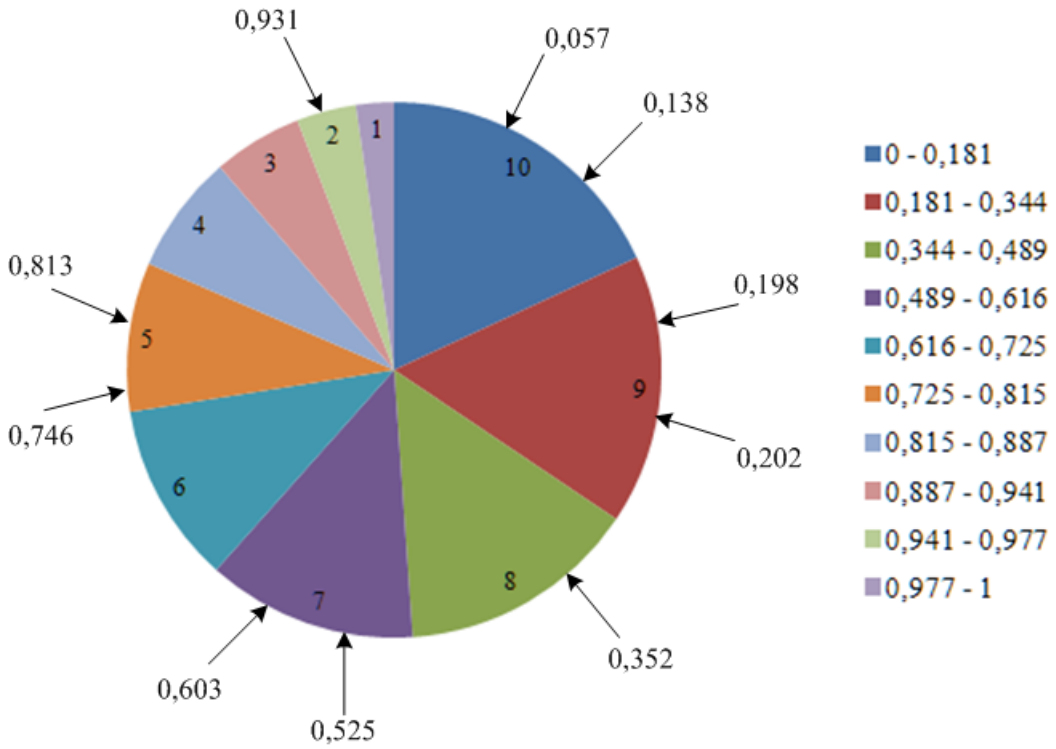
Burada; S_i birey i'nin seçilme olasılığını, U_i birey i'nin uygunluk değerini, $\sum U$ nüfus içerisindeki bireylerin toplam uygunluk değerlerini göstermektedir.

Tablo 2.4. Rulet tekerleği seçilme olasılıkları

Amaç fonksiyon değerleri	10	11	18	21	45	120	121	132	207	212
Uygunluk değerleri	10	9	8	7	6	5	4	3	2	1
Seçilme olasılıkları	0,181	0,163	0,145	0,127	0,109	0,090	0,072	0,054	0,036	0,018

Tablo 2.4'de amaç fonksiyon değeri 10 olan birey, en kısa yol uzunluğuna sahip ve nüfus içerisindeki diğer bireylerle karşılaştırıldığında en uygun bireydir. Bu nedenle seçilme olasılığı en yüksek bireydir. Amaç fonksiyon değeri 212 olan birey, en uzun yola sahiptir ve nüfus içerisinde seçilme olasılığı en az olan bireydir. Tablodaki değerlere göre, eşleştirme havuzuna seçilecek 10 adet bireyi belirlemek için, 0 ile 1

arasında düzgün dağılımlı rastgele 10 adet sayı üretilir. Bu rastgele sayıların, seçilme olasılıkları doğrusu üzerinde hangi parçaya isabet ettiği belirlenir ve o parçaya ait birey seçilmiş olur (Şekil 2.10). Şekil üzerinde de görüldüğü gibi, uygunluk değerleri büyük olan, yani yol uzunluğu kısa olan bireyler daha sıklıkla seçilmişlerdir.



Şekil 2.10. Rulet tekerleği seçimi

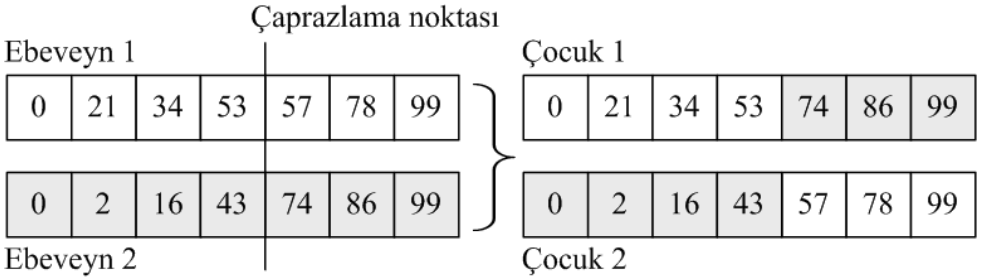
2.4. Çaprazlama

Çaprazlama, ebeveyn bireylerin farklı pozisyonlardaki iyi olan genlerinin aynı çocuk bireyde birleştirilerek, ebeveynlerden daha iyi bir birey üretilebilmesi amacıyla yapılmaktadır. Bu yeni bireylerin, ortama ebeveynlerinden daha iyi uyum göstermeleri yani optimum çözüme daha yakın olmaları beklenmektedir. Bütün çaprazlama operatörlerinin amacı aynı olmakla beraber, kullanılan kodlama türüne göre uygulanışı farklılık arz edebilmektedir.

GA'da bireyler sabit uzunlukta bilgi dizilerinden oluşmaktadır ve bireylere çaprazlama operatörünün uygulanması ile üreme gerçekleştirilmektedir. Ebeveyn birey çiftlerinin karşılıklı parçaları yer değiştirilerek çocuk bireyler üretilmektedir. Kaç tane parçanın değiştirileceğini belirleyen çaprazlama noktası sayısı genellikle birkaçı geçmez. Çaprazlama birey seviyesinde yapılan bir işlemdir. Çaprazlamaya

tabi tutulacak olan ebeveyn bireyler nüfus içerisinden rastgele seçilmektedir. Ayrıca, ebeveyn bireyleri sağ ve sol iki parçaya ayıran çaprazlama noktası da rastgele belirlenmektedir [48].

Bu tez çalışmasında, tek noktalı tam sayı çaprazlama operatörü kullanıldı. Her bir çaprazlama işlemi için farklı bir çaprazlama noktası rastgele olarak belirlendi. Şekil 2.11 yol bulma için örnek bir çaprazlama işlemini göstermektedir.



Şekil 2.11. Yol bulma problemi için tek noktalı çaprazlama işlemi

2.5. Mutasyon

Çaprazlama ebeveyn bireylerden çocuk bireyler üreterek, mevcut çözümler civarında daha iyi çözümler bulmayı sağlamaktadır. Dolayısı ile arama yüzeyindeki bir tepenin eteklerinden zirvesine doğru aramayı yönlendirmektedir. Ancak, bu işlem nüfus içerisindeki bireylerin aynı tepe civarında toparlanmasına yani birbirlerine benzemesine neden olmaktadır. Çaprazlamanın bu sakıncası GA'nın paralel çözüm arama özelliğini sınırlamakta ve çözümün yerel en iyiye takılmasıyla sonuçlanmaktadır. İstenmeyen bu durumu gidermek için yapılacak olan tek şey, nüfus içerisindeki bireylerin çeşitliliğini artırmaktır. Bireylerin arama uzayına tekrar dağılmasını ve yeni alanların keşfedilmesini sağlayacak olan ise mutasyon işlemidir [50]. Mutasyon operatörünün işlevi, bireylerin birbirlerine benzemeye başlamasını engellemek için bireylerin genlerini düşük bir olasılıkla rastgele değiştirmektir. Nüfus içerisindeki genlerin ne kadarının değişikliğe uğratılacağı mutasyon oranıyla kontrol edilmektedir. Mutasyon oranının 1 olduğu durumda nüfus içindeki bireylerin yaklaşık olarak tamamı değişime uğrar, 0 olması durumunda ise hemen hemen hiçbir birey değişmez. Literatürde yaygın olarak kullanılan mutasyon oranları, nüfus içerisindeki toplam gen sayısına bağlı olmakla beraber, 0,1 - 0,01 veya 0,001 gibi çok küçük değerlerdir.

Tablo 2.5, tamsayı kodlanmış bir kromozom için örnek bir mutasyon işlemini göstermektedir. Örnek mutasyon işleminde, bütün bireylerin her bir geni için rastgele mutasyon olasılıkları üretilmiştir. Mutasyon olasılığı mutasyon oranından daha düşük olan gen, genin alabileceği en küçük ve en büyük değerler arasında rastgele bir değerle değiştirilmiştir.

Tablo 2.5. Tamsayı kodlanmış bir kromozom için mutasyon işlemi

Mutasyon oranı			0,01				
Mutasyon olasılıkları	0,231	0,438	**0,003**	0,019	0,752	0,928	0,563
Mutasyondan önce	0	21	**34**	53	74	86	99
Mutasyondan sonra	0	21	**45**	53	74	86	99

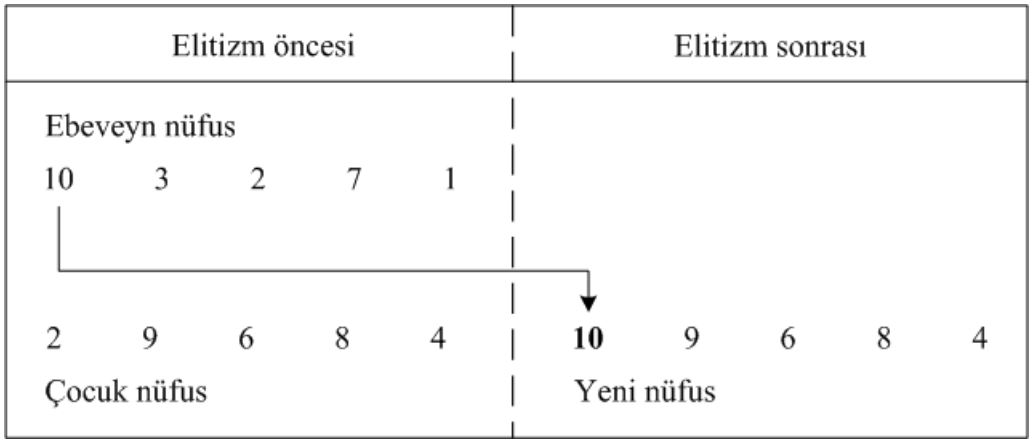
2.6. Elitizm

Uygunluk değeri iyi olan ebeveyn bireylerin özelliklerinin bir sonraki kuşağa aktarılması gerekir. Fakat, çaprazlama ve mutasyon işlemleri sonrasında, ebeveynlerin iyi özellikleri çocuklarda bozulmuş olarak ortaya çıkabilir veya tamamen kaybolabilir. İyi genlerin kaybolmasını engellemek için, iyi ebeveynleri (elit bireyler) çaprazlama ve mutasyon işlemleri sonrasında, olduğu gibi yeni jenerasyona aktarmak gerekir. Elitizm, bu bireylerin özelliklerinin değişmeden bir sonraki kuşağa aktarılması işlemidir.

Standart GA'da, mutasyon işleminden sonra elitizm işlemi uygulanmadan, çocuk bireyler yeni nüfusu oluşturmaktadır. Elitizm işlemi uygulandığında, her bir kuşakta ebeveyn bireyler içerisinden önceden belirlenmiş sayıda en iyi birey, en kötü çocuk bireylerle yer değiştirilerek yeni nüfusa katılmaktadır. Böylece, iyi bireylerin bir kuşak yerine, birkaç kuşak veya daha iyi bireyler üretilene kadar yaşaması sağlanmış olmaktadır. Bu tez çalışmasında GA'da elitizm işlemi uygulandı.

Şekil 2.12'de, ebeveyn bireyler içerisinde uygunluk değeri en yüksek olan bireyin, elit birey olarak yeni nüfusa eklendiği bir örnek gösterilmektedir. Örnekte, ebeveyn bireyler ile çaprazlama ve mutasyona uğratılmış çocuk bireylerin uygunluk değerleri verilmektedir. Çocuk bireylerin tümü yeni nüfusa aktarılırken, elitizm işlemi

uygulanarak 2 uygunluk değerine sahip kötü çocuk birey yerine, 10 uygunluk değerine sahip iyi ebeveyn birey yeni nüfusa katılmaktadır.



Şekil 2.12. Elitizm uygulaması [48]

2.7. Yol Bulma için Yeni Bir Mutasyon Operatörü

GA'da mutasyon operatörünün en yaygın kullanımı, mutasyona uğrayacak genin değerini rastgele üretmektir. Fakat yol bulma problemlerinde rastgele üretilen mutasyon, uygun olmayan yollar üretebilmektedir. Mutasyondan önce engel içermeyen bir kromozomun mutasyon ile üretilen yeni geni, engelli bir alan üzerine düşebilir (Şekil 2.13a). Bu durum optimizasyonu yavaşlatır ve jenerasyon sayısının artmasına sebep olur.

Bu problemi çözmek için yapılacak işlemlerden bir tanesi, başlangıç nüfusu oluşturma işleminde yapıldığı gibi, mutasyon operatörü ile değiştirilen genlerin engel bölgesine düşmesinin engellenmesidir. Örneğin, mutasyona uğrayacak olan genin yerine gelecek olan ve rastgele üretilen yeni değeri hemen o an kontrol edilir ve o noktanın engel noktası olup olmadığına bakılır. Yeni üretilen değer engel noktasına denk geliyor ise mutasyon işlemi uygun bir gen bulunana kadar tekrar edilir.

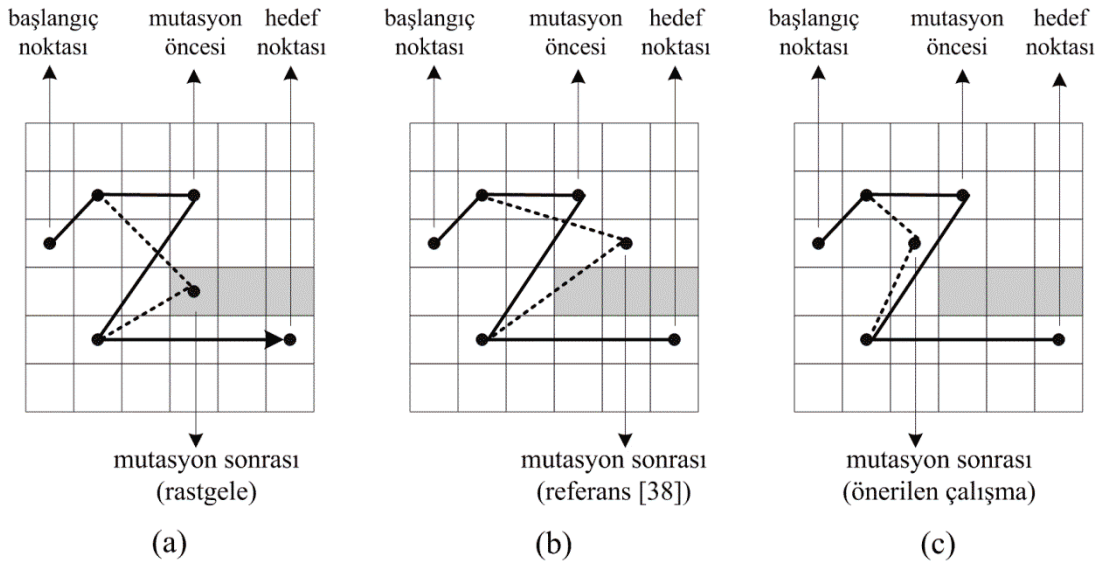
Tez çalışmasında, mutasyon operatörü için hem rastgele üretim hem de engel kontrollü üretim işlemlerinin her ikisi de denendi ve aralarındaki farklar incelendi. Aynı çalışma ortamında ve koşullarda yapılan denemelerde, GA ile yol bulma sonuçları Tablo 2.6'da verildiği gibi elde edildi. Her iki işlem için de 10'ar denemenin ortalamaları alındı. Tabloda görüldüğü gibi, mutasyon işleminde engelli noktaların önlenmesi, algoritmayı daha başarılı hale getirdi.

Tablo 2.6. Rastgele ve engel kontrollü mutasyon işlemlerinin GA performansına etkisi

Mutasyon metodu	Rastgele	Engel kontrollü
Amaç fonksiyon	28,35	27,78
Jenerasyon sayısı	49	41
Çözüm süresi (sn)	1,25	1,12

Mutasyon operatörünü geliştirmeye yönelik literatürde farklı çalışmalar yapılmıştır [37, 38]. Bu çalışmalardan en yaygını, yukarıda anlatılan mutasyon ile yeni oluşan kromozomun engel içerip içermediğinin kontrol edilmesidir. Bundan farklı olarak, Changan ve diğ. [15] yapmış oldukları bir çalışmada, tepe tırmanma (hill-climbing) algoritmasını içeren bir mutasyon operatörü kullanmışlardır. Algoritma, mevcut gen etrafında daha iyi bir gen için arama yapar ve mutasyona uğratılacak gen bulunan daha iyi gen ile değiştirilir.

Otonom robotların yol bulma problemlerini çözmek için daha etkili bir diğer mutasyon yöntemi Li ve diğ. [38] tarafından önerilmiştir. Bu metotta, mutasyon geninin komşularının oluşturduğu kümeden rastgele bir komşu seçilir, bu komşu mevcut noktadan daha ileri yöndeki (hedef yönünde) bir komşu ise yeni gen olarak kabul edilir, eğer değilse rastgele başka bir komşu seçilir (Şekil 2.13b). Bu işlem, hedef yönünde mevcut noktadan daha yakın bir komşu bulunana kadar devam eder.



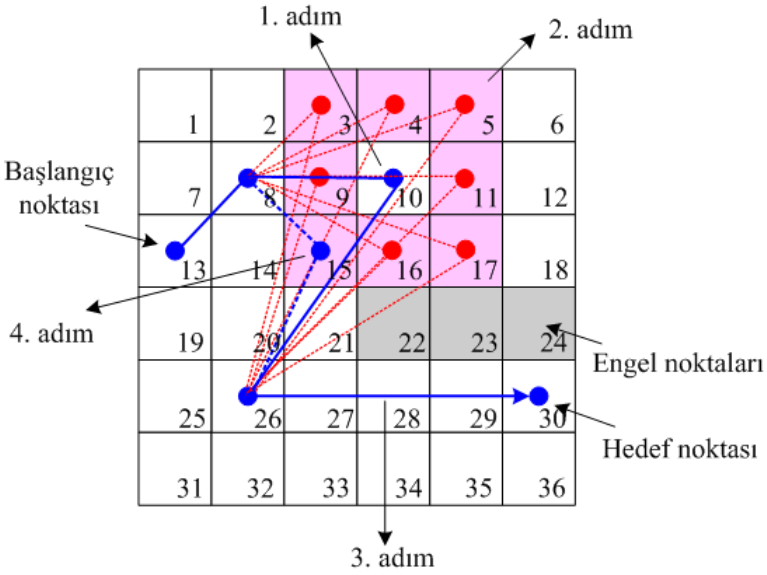
Şekil 2.13. Farklı mutasyon operatörlerinin karşılaştırılması

Bu tez çalışmasında, literatürde yer alan diğer mutasyon operatörlerinden daha etkili yeni bir mutasyon operatörü önerilmektedir. Önerdiğimiz mutasyon metodunun işlem basamakları Tablo 2.7'de açıklanmaktadır. Metodumuz, Li ve diğ. [38]'nin açıkladığı çalışmaya benzetilebilir, ancak o çalışmadan iki farklı yönden ayrılmaktadır. Birincisi, önerdiğimiz mutasyon metodu, komşu düğümleri teker teker rastgele seçip kontrol etmek yerine, tüm komşu düğümleri bir kerede kontrol eder. Rastgele seçim, en iyi düğümü bulmadan önce, mevcuttan daha iyi bir düğüm bulduğunda işlemi sonlandırırken, bizim önerimiz en iyi düğümü bulmayı garanti eder. İkincisi, önerdiğimiz metot, yeni mutasyon düğümünü kabul edip etmemeyi, hedefe doğru hareket yönüne bakarak değil toplam yolun amaç fonksiyon değerine göre belirler (Şekil 2.13c). Yeni mutasyon düğümü başlangıç-hedef doğrultusundan farklı bir yönde olsa dahi, eğer en uygun toplam yolu veriyor ise, önerdiğimiz mutasyon operatörü bu yeni mutasyon düğümünü kabul eder.

Tablo 2.7. Yeni mutasyon operatörünün işlem basamakları

Adım	İşlem
1	Başlangıç ve hedef düğümü haricinde, belirlenen mutasyon oranına göre rastgele bir tane mutasyon düğümü seç.
2	Mutasyon düğümünün engel düğümü olmayan tüm komşularını bir kümeye aktar.
3	Her bir komşu düğümü yeni mutasyon geni olarak alıp amaç fonksiyonunu bul.
4	En iyi amaç fonksiyona sahip kromozomdaki mutasyon genini yeni mutasyon geni olarak belirle.

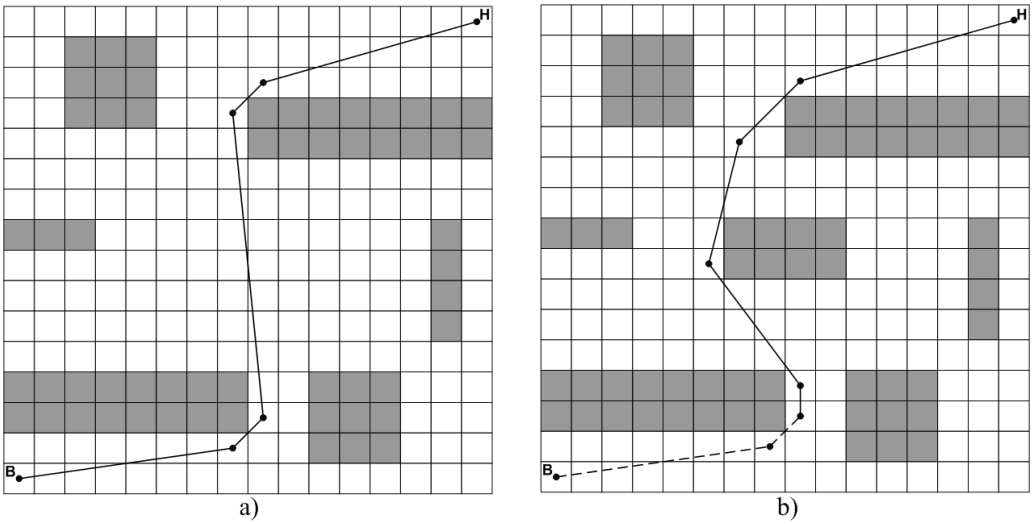
Tablo 2.7'de işlem basamakları verilen yeni mutasyon önerimiz Şekil 2.14'de görsel olarak sunulmaktadır. Şekildeki örnekte; (13, 8, 10, 26, 30) düğümlerinden geçen bir yol verilmektedir. Birinci adımda, yolu tanımlayan düğümler içerisinden rastgele seçilen 10 düğümü mutasyon düğümü olarak belirlenmektedir. İkinci adımda, 10 düğümünün engel olmayan tüm komşuları (3, 4, 5, 9, 11, 15, 16, 17) bir kümeye aktarılır. Üçüncü adımda, küme içerisindeki tüm komşu düğümlerin amaç fonksiyon değerleri hesaplanır. Dördüncü adımda, 15 numaralı düğüm en uygun toplam yol uzunluğuna sahip olduğu için yeni mutasyon geni olarak alınır. Mutasyon işlemi sonucunda yeni yol (13, 8, 15, 26, 30) olarak belirlenir.



Şekil 2.14. Yeni mutasyon operatörünün adımları

2.8. Yeni Mutasyon Operatörünün Performans Değerlendirmesi

Bu tez çalışmasında önerdiğimiz yeni mutasyon operatörü, literatürdeki geçmiş GA çalışmalarıyla karşılaştırıldı. Karşılaştırmalar, Matlab yazılımı kullanılarak benzetim ortamında gerçekleştirildi. Hem statik hem de dinamik olmak üzere iki farklı çalışma ortamı kullanıldı. Birinci karşılaştırma için, referans [38]'deki çalışma ortamı seçildi.



Şekil 2.15. Uygulama-1 için başlangıç (a) ve değiştirilmiş (b) çalışma ortamı

Şekil 2.15a, 16x16 boyutunda hücrelerden ve 6 adet engel bölgesinden oluşan, robotun hareketinden önceki statik çalışma alanını göstermektedir. Şekil 2.15b ise robot harekete başladıktan sonra, yeni bir engel bölgesi eklenerek değiştirilmiş

dinamik bir çevreyi göstermektedir. GA parametreleri; nüfus büyüklüğü 60, çaprazlama oranı 1 ve mutasyon oranı 0,3 olarak alındı. Hem statik hem de dinamik çalışma ortamında, rastgele mutasyon, referans [38] ile verilen mutasyon, referans [15] ile verilen mutasyon ve çalışmada önerdiğimiz mutasyon olmak üzere, GA dört farklı mutasyon operatörünün her biri için 100'er kez çalıştırıldı.

Tablo 2.8. Uygulama-1'in başlangıç ortamı için deneysel sonuçları

	Rastgele	Referans [38]	Referans [15]	Çalışmada önerilen
Optimum çözüm	1	3	2	54
Optimuma yakın çözüm	86	69	69	44
Fizibıl olmayan çözüm	13	28	29	2
Amaç fonksiyon	31,78	29,25	29,91	27,82
Jenerasyon sayısı	21	23	22	11
Çözüm süresi (sn)	0,28	0,31	0,46	0,89

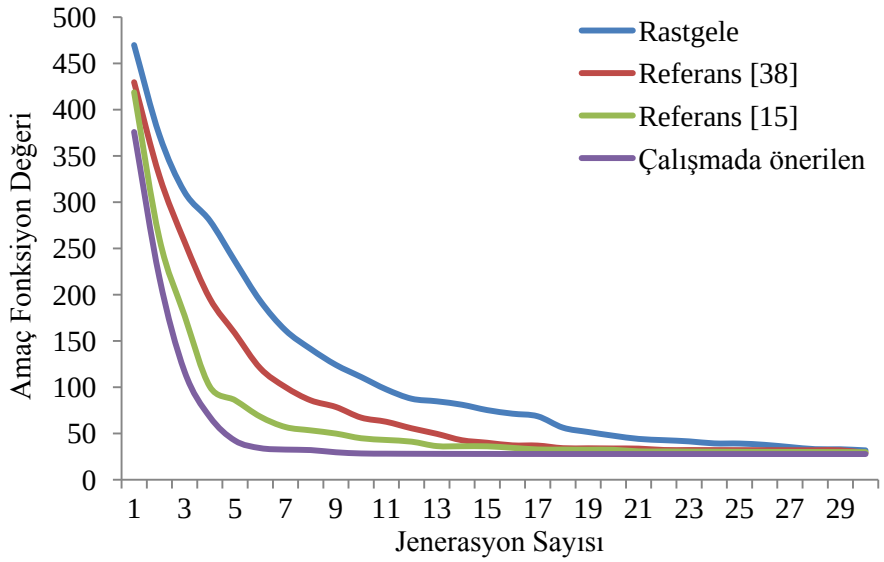
Tablo 2.9. Uygulama-1'in dinamik ortam için deneysel sonuçları

	Rastgele	Referans [38]	Referans [15]	Çalışmada önerilen
Optimum çözüm	0	0	0	44
Optimuma yakın çözüm	95	95	89	56
Fizibıl olmayan çözüm	5	5	11	0
Amaç fonksiyon	35,37	31,21	30,87	29,08
Jenerasyon sayısı	23	22	23	11
Çözüm süresi (sn)	0,20	0,26	0,41	0,86

Tablo 2.8 ve 2.9, sırasıyla statik ve dinamik ortam için ayrı ayrı deneysel sonuçları göstermektedir. Tablolar 100'er deneme sonunda, en iyi sonuçların sayısını, en iyiye yakın sonuçların sayısını ve uygun (fizibıl) olmayan yolların sayısını göstermektedir. Tablolar ayrıca, 100'er deneme sonunda, en iyi ve en iyiye yakın olmak üzere bulunan tüm yollar için ortalama amaç fonksiyon değerini, ortalama jenerasyon sayısını ve ortalama çözüm süresini göstermektedir. Sonuçlar açıkça göstermektedir ki, diğer metotlar en uygun yolu yalnızca birkaç kez bulabiliyorken, çalışmada önerdiğimiz mutasyon operatörü en uygun yolu statik ortamda 54 ve dinamik ortamda 44 kez bulabilmektedir. Önerilen metodun ortalama amaç fonksiyon değerleri ve ortalama jenerasyon sayılarının, diğer metotlarda bulunan sonuçlardan

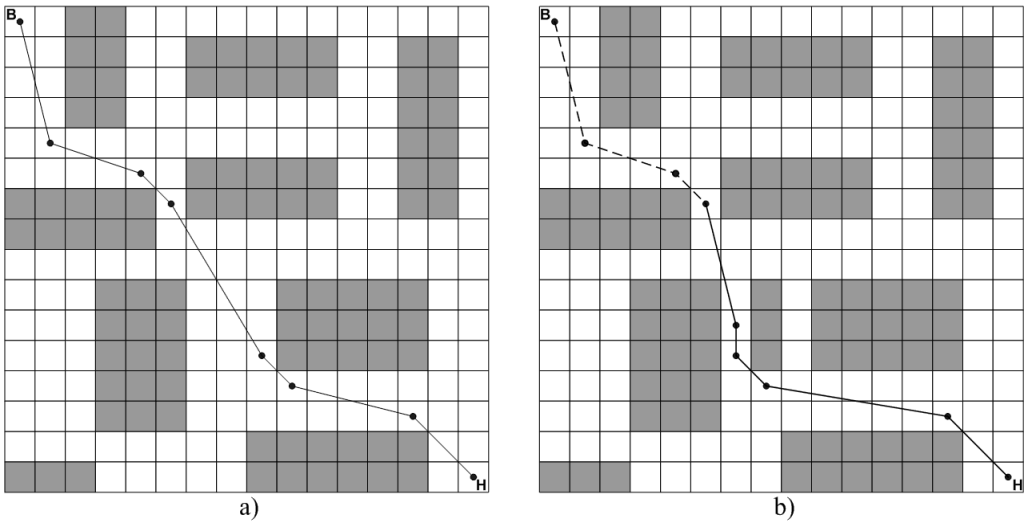
daha başarılı olduğu da ayrıca görülmektedir. Diğer taraftan, önerilen metodun çözüm süresi, diğer metotlarda olduğu gibi bir saniyenin altında olmasına rağmen, yine de onlardan daha yavaştır.

Şekil 2.16, farklı mutasyon metotları kullanan GA'nın amaç fonksiyon ve jenerasyon sayısı grafiklerini göstermektedir. Grafikte de görüldüğü gibi, önerdiğimiz mutasyon metodunu kullanan GA, diğer metotlardan daha hızlı bir şekilde optimum çözüme yakınsamaktadır.



Şekil 2.16. Uygulama-1 için optimum çözüme yakınsama grafiği

Başka bir karşılaştırma yapmak için, önceki uygulamada kullanılan çalışma ortamından daha karmaşık bir çalışma ortamı seçildi (Şekil 2.17).



Şekil 2.17. Uygulama-2 için başlangıç (a) ve değiştirilmiş (b) çalışma ortamı

Şekil 2.17a, 16x16 boyutunda hücrelerden ve 9 adet engel bölgesinden oluşan, robotun hareketinden önceki statik çalışma alanını göstermektedir. Şekil 2.17b ise robot harekete başladıktan sonra, yeni bir engel bölgesi eklenerek değiştirilmiş dinamik bir çevreyi göstermektedir. GA parametreleri nüfus büyüklüğü 80, çaprazlama oranı 1 ve mutasyon oranı 0,2 olarak alındı. Önceki uygulamada olduğu gibi her bir metot 100'er kez çalıştırıldı.

Tablo 2.10. Uygulama-2'nin başlangıç ortamı için deneysel sonuçları

	Rastgele	Referans [38]	Referans [15]	Çalışmada önerilen
Optimum çözüm	0	1	2	9
Optimuma yakın çözüm	46	68	39	78
Fizibıl olmayan çözüm	54	31	59	13
Amaç fonksiyon	29,69	25,02	25,93	24,68
Jenerasyon sayısı	81	65	47	16
Çözüm süresi (sn)	1,22	1,03	0,85	1,68

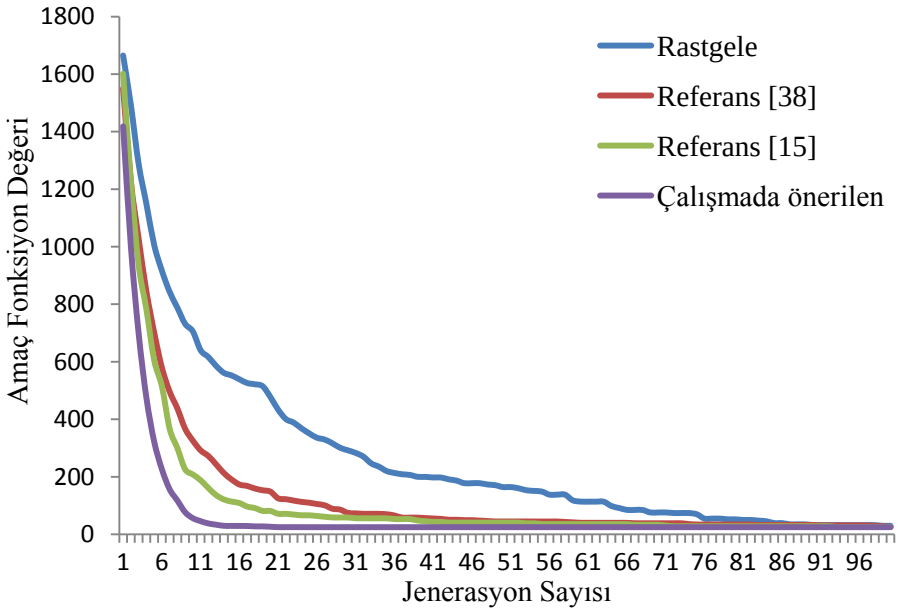
Tablo 2.11. Uygulama-2'nin dinamik ortam için deneysel sonuçları

	Rastgele	Referans [38]	Referans [15]	Çalışmada önerilen
Optimum çözüm	6	6	15	32
Optimuma yakın çözüm	94	92	77	68
Fizibıl olmayan çözüm	0	2	8	0
Amaç fonksiyon	25,48	25,17	25,84	24,71
Jenerasyon sayısı	73	31	38	12
Çözüm süresi (sn)	0,74	0,34	0,49	0,69

Tablo 2.10 ve 2.11, sırasıyla statik ve dinamik ortam için ayrı ayrı deneysel sonuçları göstermektedir. Tablolar 100'er deneme sonunda, en iyi sonuçların sayısını, en iyiye yakın sonuçların sayısını ve uygun olmayan yolların sayısını göstermektedir. Tablolar ayrıca, 100'er deneme sonunda, en iyi ve en iyiye yakın olmak üzere bulunan tüm yollar için ortalama amaç fonksiyon değerini, ortalama jenerasyon sayısını ve ortalama çözüm süresini göstermektedir. Sonuçlar açıkça göstermektedir ki, diğer metotlardan en iyisi, en uygun yolu statik ortamda 2 ve dinamik ortamda 15 kez bulabiliyorken, çalışmada önerdiğimiz mutasyon operatörü en uygun yolu statik ortamda 9 ve dinamik ortamda 32 kez bulabilmektedir. Önerilen metodun ortalama

amaç fonksiyon değerleri ve ortalama jenerasyon sayılarının, diğer metotlarda bulunan sonuçlardan daha başarılı olduğu da ayrıca görülmektedir. Fakat, önerdiğimiz metodun çözüm süresi diğer metotların çözüm süresinden daha yavaştır.

Şekil 2.18, farklı mutasyon metotları kullanan GA'nın amaç fonksiyon ve jenerasyon sayısı grafiklerini göstermektedir. Grafikte de görüldüğü gibi, önerdiğimiz mutasyon metodunu kullanan GA, diğer metotlardan daha hızlı bir şekilde optimum çözüme yakınsamaktadır.



Şekil 2.18. Uygulama-2 için optimum çözüme yakınsama grafiği

Her iki uygulama sonucunda, önerdiğimiz mutasyon metodunu kullanan GA, en iyi yolu diğer metotlardan daha fazla bulabilmektedir.

3. FPGA

İşlem yükü fazla olan çalışmalar standart bilgisayarlar ile değil, daha alt seviyede donanımsal olarak gerçekleştirilebilmektedir. Bu, çalışmaya hem hız kazandırmakta hem de mobil uygulamalarda taşınması gereken yükün ağırlığını ve hacmini azaltmaktadır. Problemlerin karmaşıklığı arttıkça, yazılımsal olarak gerçekleştirilen GA'nın çözüm zamanı da uzun sürebilmektedir. Yapmış olduğumuz tez çalışmasında, GA'yı donanımsal olarak gerçekleştirerek çözüm süresinin kısaltılması ve gezgin robotun yükünün hafifletilmesi hedeflenmektedir.

FPGA, donanım hızı ve yazılım esnekliğinin faydalarına sahiptir. Bu yüzden, pek çok bilimsel ve mühendislik uygulamaları için iyi bir seçimdir [51]. Bir sistemin tamamı, FPGA üzerinde C veya C++ kullanarak yazılımsal olarak ya da VHDL veya Verilog gibi donanım tanımlama dilleri kullanarak donanımsal olarak gerçekleştirilebildiği gibi, sistemin bir bölümü yazılımsal diğer bölümü donanımsal olarak da gerçekleştirilebilmektedir. Mikroişlemcileri kullanan tek yonga donanım sistemlerinden farklı olarak yazılımsal işlemci çekirdekleri tasarımcılara, uygulamanın ihtiyaç durumuna göre tek bir FPGA üzerinde farklı sayıda yazılımsal işlemciyi birlikte kullanma olanağı vermektedir [8].

Tasarımcılar, donanım tasarımlarını hızlı bir şekilde gerçekleştirmek için FPGA kullanabilmelerine rağmen, pek çok sistemde yazılım ve donanımın birlikte kullanılması gerektirmektedir. 1990'ların sonlarında, FPGA satıcıları piyasaya tek-yonga mikroişlemci/FPGA cihazlar sunmaya başlamışlardır [8]. Yazılımsal çekirdekler donanımsal çekirdeklerle karşılaştırıldığında; daha portatif ve esnek yapıdadırlar, fakat FPGA kullanarak gerçekleştirilen yazılımsal işlemci çekirdekleri daha yüksek güç tüketir, daha düşük performansa sahiptir ve daha uzun hesaplama zamanına ihtiyaç duyarlar. Tasarımcılar, yazılımın performansını arttırmak ve güç tüketimini düşürmek için imkan dahilinde donanım ve yazılımı beraber kullanabilirler [8].

3.1. FPGA Teknolojisi

Teknolojik gelişmeler çok büyük sayıdaki transistörlerin tek bir yonga üzerine entegre edilmesini mümkün hale getirmiştir. Böylece; mikroişlemciler, uygulamaya özgü tümleşik devreler (Application Specific Integrated Circuits, ASIC), bellek birimleri ve çevre birimleri kapsayan tüm sistemler tek bir yongaya dahil edilebilmektedir [52]. FPGA, programlanabilir mantık blokları, giriş-çıkış blokları ve bu bloklar arasındaki ara bağlantılardan oluşan sayısal bir tümleşik devredir. Bu devreler, çok geniş bir uygulama alanına sahiptir ve herhangi bir uygulamaya özel olarak programlanabilen genel amaçlı yongalardır. Tasarımcının ihtiyaç duyduğu mantık fonksiyonlarını gerçekleştirme amacına yönelik olarak üretilmiştir. Dolayısıyla, her bir mantık bloğunun fonksiyonu kullanıcı tarafından düzenlenebilmektedir. Sahada programlanabilir lojik kapılar olarak adlandırılmalarının sebebi, tasarımcının imalat sürecinden sonra, problemin ihtiyacına göre mantık fonksiyonlarını düzenleyebilmesi ve gerçekleştirebilmesidir. Bu yönüyle FPGA, ASIC teknolojisinden ayrılmaktadır.

ASIC teknolojisinde devre, üretim aşamasında sadece belli bir fonksiyonu gerçekleştirmek üzere tasarlanmaktadır. Geri dönüşleri yoktur, yani bir kez tasarlandıktan sonra tasarım değiştirilemez. Farklı bir sisteme ihtiyaç duyulduğunda, tekrar tasarlanamadıkları için yeni bir ASIC yapısı gerekmektedir. Bu yönüyle ASIC'ler, oldukça pahalı ve zaman harcayan devrelerdir [53]. Fakat, FPGA'de tasarımın son aşamasında bile donanımı kolayca değiştirebilmek mümkündür. FPGA üzerinde gerçekleştirilen sistemler, ihtiyaca göre yeniden tasarlanabilmektedir.

FPGA içindeki yapılandırılabilir lojik kapasitesinin artması ve FPGA maliyetlerinin düşmesi, tasarımcıların FPGA'lere olan ilgilerini artırmaktadır [8]. FPGA günümüzde oldukça yaygın kullanım alanına sahiptir. FPGA'ler sayısal işaret işleme, radyo haberleşme sistemleri, uzay, savunma sistemleri, medikal görüntüleme, robotik, ses tanıma, şifreleme, bilgisayar donanımı gibi pek çok alanda kullanılmaktadır.

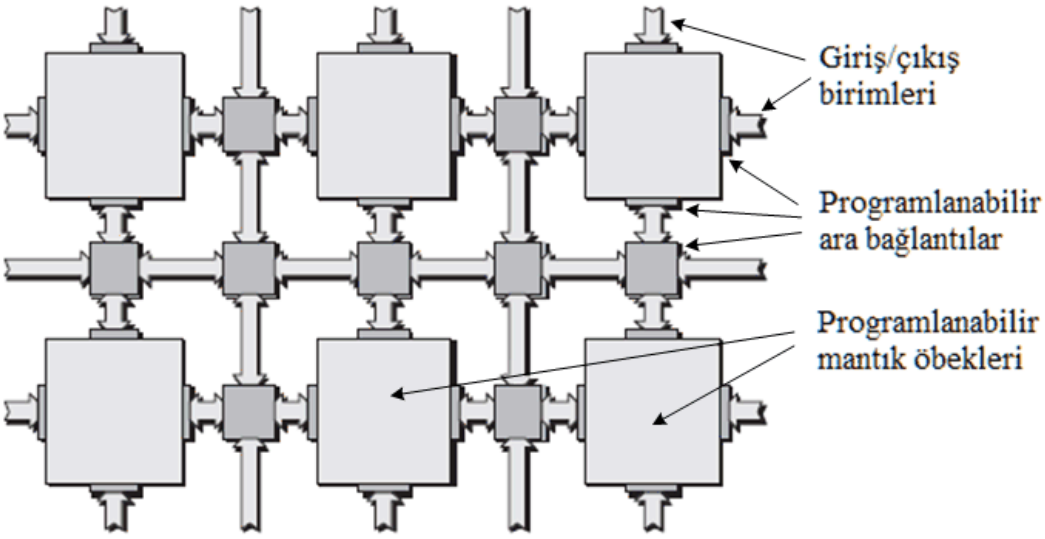
Yüksek hız ve performansa sahip olmaları nedeniyle, birçok alanda diğer mikroişlemci ve sayısal işaret işlemcilere göre çok daha fazla tercih edilir hale

gelmişlerdir. Özellikle hızlı tasarım, düşük maliyet ve esnek programlanabilir olma özellikleri tercih edilme nedenlerinin başında gelmektedir.

FPGA'lerin en önemli özelliklerinden bir tanesi de paralel işlem yapabilme yeteneğine sahip olmalarıdır. Özellikle yüksek performans gerektiren görüntü işleme, sayısal işaret işleme gibi uygulamalar için paralel işlem yapabilme yeteneği tasarımcıların FPGA'i tercih etme nedenleri arasında yer almaktadır.

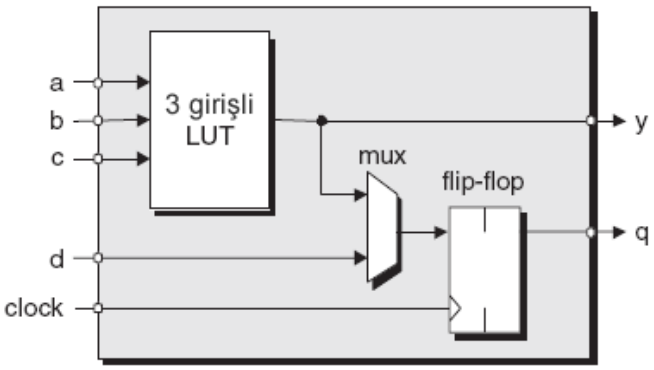
Şekil 3.1'de genel mimarisi verilen FPGA temel olarak üç ana bölümden oluşmaktadır;

- Mantık blokları veya öbekleri, düzenlenebilir mantıksal fonksiyonları ifade eder.
- Giriş-çıkış birimleri.
- Ara bağlantılar, mantık blokları ile giriş-çıkış birimleri arasındaki gerekli bağlantılardır.



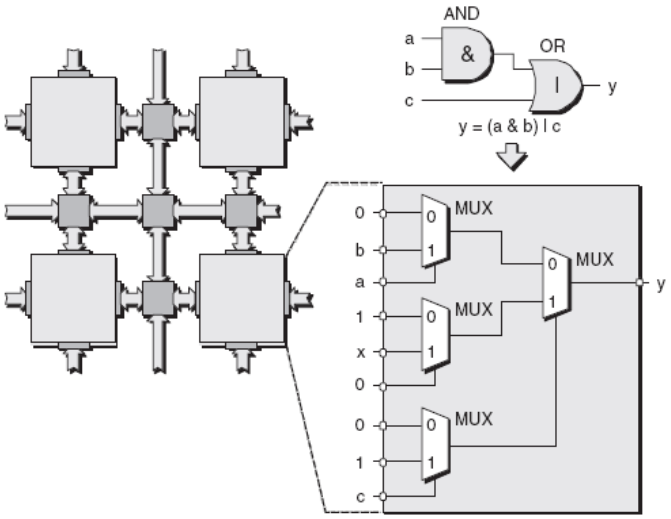
Şekil 3.1. Genel FPGA mimarisi [54]

FPGA içindeki programlanabilir mantık blokları, basit lojik kapılar olabileceği gibi, daha karmaşık fonksiyonlar da olabilir. Programlanabilir mantık blokları; LUT (look-up table) adı verilen doğruluk tablosu tabanlı ve MUX (multiplexer) adı verilen çoklayıcı tabanlı olmak üzere iki gruba ayrılabilir. LUT tabanlı bir mantık bloğu birer adet LUT, çoklayıcı ve D tipi flip-flop'tan oluşur. Şekil 3.2'de basit bir LUT tabanlı mantık bloğunun içeriği gösterilmektedir.



Şekil 3.2. LUT tabanlı mantık bloğu [54]

Şekil 3.3'de MUX tabanlı bir mantık bloğunun yapısı gösterilmektedir. MUX tabanlı yaklaşıma örnek olarak, üç girişli bir lojik fonksiyon olan y=(a&b)|c'nin sadece çoklayıcı içeren bir mantık bloğuyla gerçekleştirilmesi verilebilir. Mantık bloğunun her bir girişi a, b, c girişlerinin kendisi veya tersi, lojik-0, lojik-1 veya başka bir bloğun çıkışından gelen değer olabilir. Bu, bir bloğun birçok olası fonksiyonu gerçeklemek için sayısız yolla yapılandırılabilmesini sağlar. Şekildeki x girişi "fark etmez (don't care)" anlamındadır.

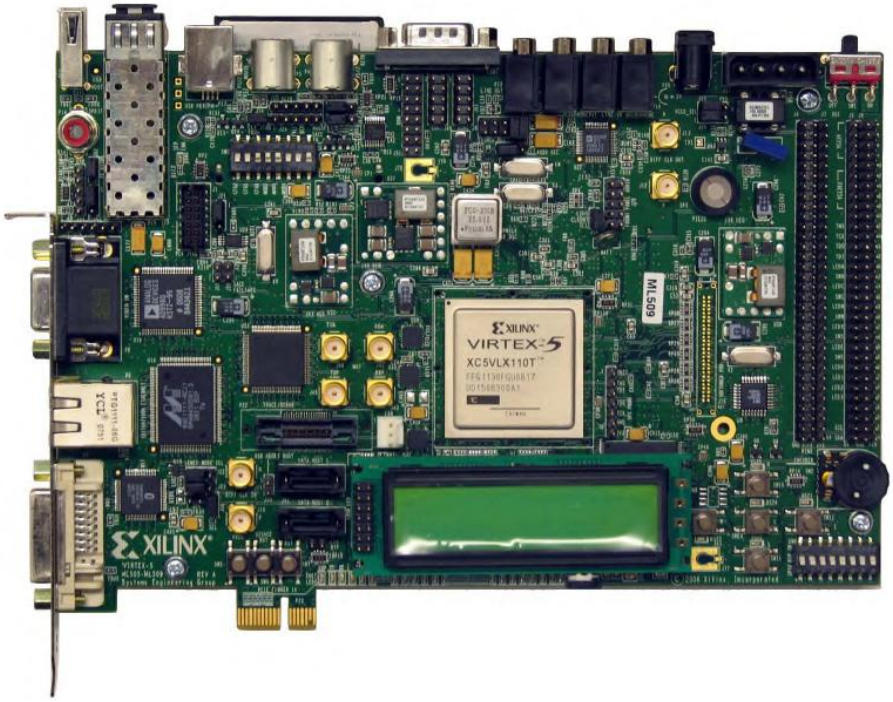


Şekil 3.3. MUX tabanlı mantık bloğu [54]

FPGA'in programlanması, içerisindeki lojik kapıların yapılandırılması ve programlanabilen anahtarlar vasıtası ile ara bağlantıların oluşturulmasıyla gerçekleştirilir. FPGA'de, herhangi bir tasarımı donanım olarak gerçekleştirmek için donanım tanımlama dili (hardware description language, HDL) kullanılmaktadır. Donanım tanımlama dillerinden en yaygın kullanılanları Verilog veya VHDL'dir.

3.2. FPGA Geliştirme Kartı

Piyasada farklı FPGA üretici firmaları bulunmaktadır. Bunlar; Xilinx [55], Altera [56], Atmel [57] vb. firmalardır. Tez çalışmasında, Şekil 3.4'de gösterilen Xilinx firmasına ait Virtex-5 XUPV5-LX110T FPGA geliştirme kartı kullanıldı. XUPV5-LX110T akademik çalışmalar için geliştirilmiş, dahili hafıza ve standart bağlantı ara yüzlerine sahip bir geliştirme platformudur. Bu geliştirme kartı; sayısal tasarım, gömülü sistemler, işaret işleme ve haberleşme, bilgisayar mimarisi, işletim sistemleri, görüntü ve resim işleme gibi araştırma alanlarında kullanılmaktadır [58].

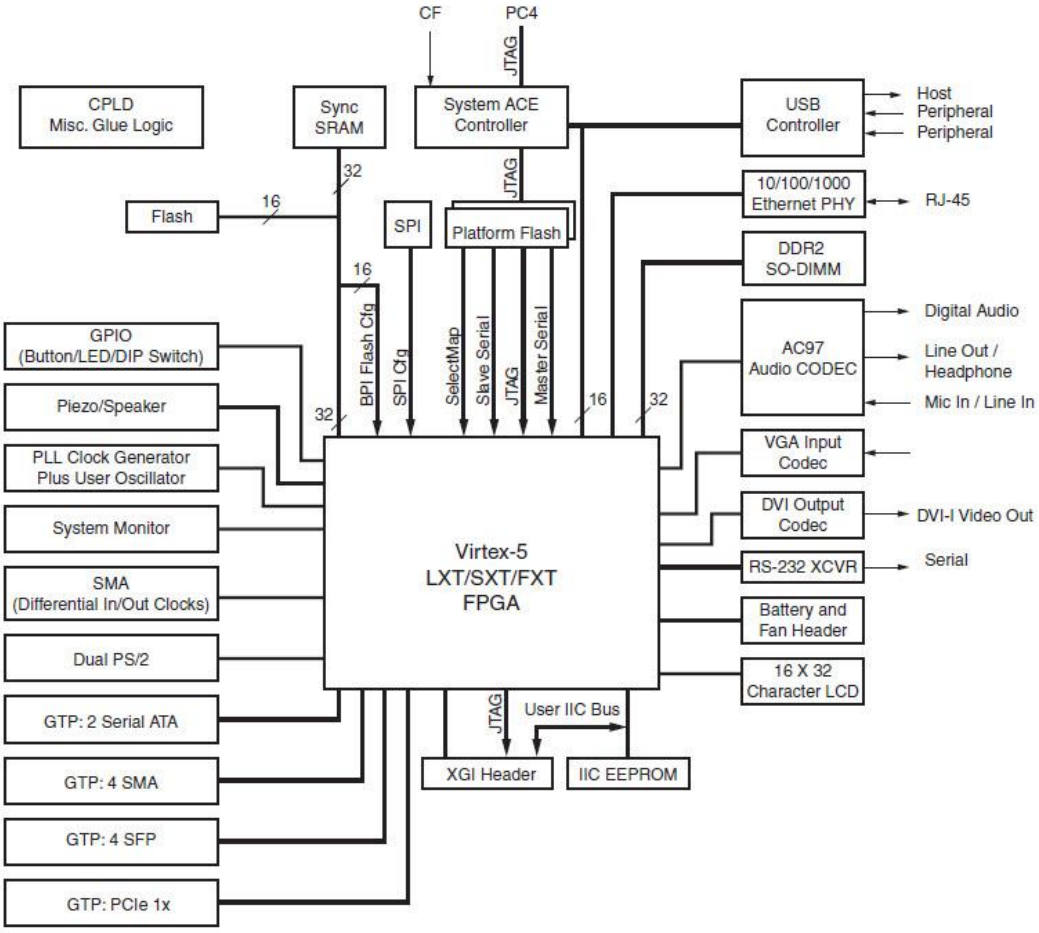


Şekil 3.4. Virtex-5 FPGA geliştirme kartı

Şekil 3.5 ile blok şeması verilen Virtex-5 geliştirme kartının özellikleri şunlardır:

- Virtex-5 XC5VLX110T FPGA,
- 64 bit genişliğinde 256 MB DDR2 SODIMM,
- Osilatörler,
- LCD için parlaklık ve kontrast ayarı,
- Genel amaçlı giriş/çıkış anahtarları,
- Kullanıcı ve hata ledleri,
- Kullanıcı butonları,

- CPU reset butonu,
- Stereo AC97 ses,
- RS-232 iletişim portu,
- 2 x 16 karakter LCD,
- 8 Kb EEPROM'lu IIC,
- DVI, USB ve JTAG portları,
- PS/2 fare ve klavye portları,
- 1 GB Flash Bellek,
- SRAM,
- 10/100/1000 üç-hızlı Ethernet bağlantısı,
- Piezo/Hoparlör,
- VGA video giriş,
- PCI arayüzü.



Şekil 3.5. Virtex-5 geliştirme kartının blok şeması [59]

3.3. FPGA Üzerinde Gömülü Sistem Geliştirme

FPGA'ler son yıllarda, gömülü sistemlerde ve yüksek performanslı uygulamalarda çok popüler olmaya başlamıştır. FPGA içerisine sadece donanım fonksiyon birimleri yerleştirilmez, bununla beraber gömülü işlemler de yerleştirilebilmektedir [60]. Gömülü sistemler, bir sistemin tüm bileşenlerinin tek bir yonga üzerinde toplanmasını ifade eder. Tek yonga üzerindeki sistem (SoC, system on chip) olarak da ifade edilen gömülü sistemler daha az güç tüketimi, daha yüksek hız ve düşük maliyet sağlamaktadır. Gömülü donanım platformu, bir veya daha fazla işlemci, çevre birimleri ve hafıza blokları ile bunların birbirlerine bağlanmasını sağlayan veri yollarından oluşmaktadır [61]. Ayrıca, dış dünya ile iletişim kurmayı sağlayan portlara da sahiptir.

FPGA üzerinde sistem tasarımı, HDL tabanlı veya mikroişlemci tabanlı olmak üzere iki farklı durumda gerçekleştirilebilmektedir. İşlemci kullanmadan gerçekleştirilen tasarımlarla karşılaştırıldığında, işlemci kullanarak gerçekleştirilen gömülü sistemler tasarıma esneklik kazandırmaktadır. Bu esneklik sayesinde, tasarım aşamasından sonra bile tasarımın istenen parametreleri üzerinde değişiklik yapabilmek mümkün olmaktadır. Fakat, sadece HDL tabanlı olarak gerçekleştirilen sistemlerde, tasarım aşamasından sonra herhangi bir değişiklik yapabilmek için sistemin tekrar tasarlanması gerekmektedir. Ayrıca, işlemci tabanlı sistemlerde hazır IP çekirdeklerinin de kullanılabilmesi, sistemi tasarım açısından daha basit bir hale getirmektedir.

Tez çalışmasında, FPGA üzerinde gömülü sistem oluşturmak için Xilinx Embedded Development Kit (EDK) [61] kullanıldı. Xilinx EDK sistem araçları, FPGA içerisinde işlemci tabanlı gömülü sistemler gerçekleştirmek için kullanılmaktadır. Xilinx EDK platformu aşağıdakileri içermektedir;

- Xilinx Platform Studio (XPS); kullanıcı tarafından gömülü bir işlemci donanımını gerçekleştirmeyi sağlar. XPS ile, mikroişlemci tabanlı sistem tasarlanır ve çevre birimleri ile mikroişlemci arasındaki gerekli olan bağlantılar gerçekleştirilir.

- Software Development Kit (SDK); gömülü bir yazılım uygulamasını gerçekleştirmek için kullanılır. Açık kaynak kodlu Eclipse tabanlıdır. SDK, aynı zamanda bağımsız programlar için de kullanılabilir.

- Çevre birimleri ve işlemcileri kapsayan Intellectual Property Cores (IP core - fikri mülkiyet çekirdekleri).

Şekil 3.6'da, bir gömülü sistemin tasarımında kullanılan araçlar gösterilmektedir.

- İşlemci Donanımı Geliştirme (Processor Hardware Development): Xilinx FPGA teknolojisi, tasarlanan işlemci donanımında, uygulama ihtiyacına göre uyarlamalar ve değişikliklere olanak sağlanmaktadır. Bu tarz donanım değişiklikleri, standart kullanıma hazır mikroişlemci veya mikrodenetleyici yongalarında mümkün değildir. Donanım Platformu (Hardware Platform) Xilinx'de hazırlanmış, esnek gömülü işlemciyi tanımlamaktadır.

- Yazılım Geliştirme (Software Development): Yazılım sürücüleri ve isteğe bağlı olarak, uygulamanın üzerinde çalışacağı işletim sisteminin birleşimidir. Oluşturulan yazılım görüntüsü, sadece gömülü tasarımda kullanılan Xilinx kütüphanesinin bölümlerini içerir.

- Doğrulama Dosyası Üretimi (Verification File Generation): EDK, hem donanım hem de yazılım için doğrulama araçları içermektedir. Tasarlanan donanım platformunun işlevselliğini doğrulamak için, bir benzetim modeli oluşturulur ve bu model HDL simülatöründe çalıştırılır. Benzetim yapılırken, geliştirilen yazılım işlemci veya işlemciler üzerinde çalıştırılır. Benzetimde; davranışsal, yapısal veya zaman doğrulama modellerinden bir tanesi seçilebilir.

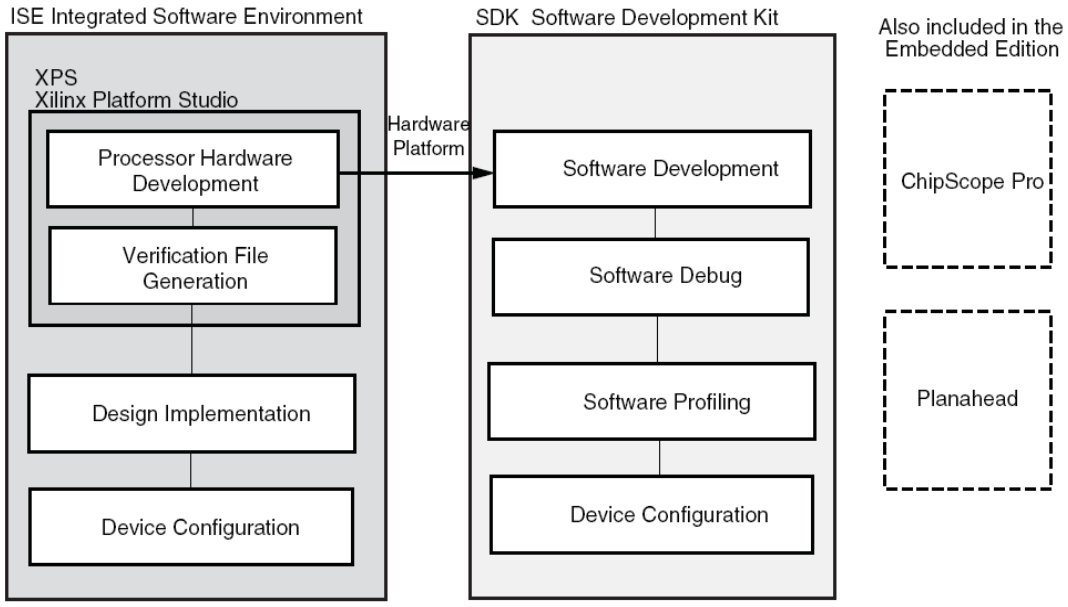
- Yazılım Ayıklama (Software Debugging): Tasarlanan yazılım, geliştirme kartına yüklenir ve ayıklama araçları kullanılarak hedeflenen işlemci kontrol edilir. Ayrıca, bir bilgisayarda çalıştırılan komut seti simülatörü ile de hazırlanan kod ayıklanabilir.

- Cihaz Konfigürasyonu (Device Configuration): Donanım ve yazılım platformları tamamlandığında, hedef FPGA için konfigürasyon bit dizisi oluşturulur. Prototip olarak kullanmak için, FPGA'e bağlı bir bilgisayardan, gömülü sistem platformu

üzerinde çalışan herhangi bir yazılım aracılığı ile FPGA'e yüklenir. Seri üretim için ise, konfigürasyon bit dizisi FPGA'e bağlı bir kalıcı belleğe yüklenir.

- Planahead: Pek çok alternatif yaklaşım sunan bir araçtır. Kolay kullanım ve güçlü pin planlama, performansı arttırmak için mantık bloklarının el ile yerleşimi ve en iyi bağlantıların seçimi, tasarımın analizi ve hata ayıklama gibi yeteneklere sahiptir.

- ChipScope Pro: FPGA üzerindeki sistemin çalışması esnasında, tasarımda kullanılan tüm veri yolları ve işaretlerin izlenmesine olanak sağlayan bir programdır.



Şekil 3.6. Basit bir gömülü sistem tasarımının akış diyagramı [61]

3.3.1. FPGA içinde yazılımsal mikroişlemci

Yazılımsal işlemciler genellikle VHDL veya Verilog gibi donanım tanımlama dilleriyle yazılmış ve mikroişlemcilerin tüm görevlerini yerine getiren yazılımlardır. Bu yazılımlar FPGA gibi programlanabilir donanımlara gömülerek, FPGA'in gerçek donanımsal bir mikroişlemci gibi çalışması sağlanır. Yazılımsal işlemcilerin bir avantajı esnek yapıya sahip olmalarıdır. Özel bir uygulama için, sadece gerekli olan işlemci özellikleri kullanılabilir. Yazılımsal işlemcilerin diğer bir avantajı ise, IP çekirdekleriyle kolay bir şekilde bütünleşerek birlikte çalışabilme yeteneğine sahip olmasıdır [53]. IP çekirdekleri, yazılımdaki gibi sıralı işlem yerine, donanım üzerinde

paralel olarak işletilen algoritmalar için yüksek hız sağlayan özgün tasarlanmış yazılımlardır. Donanım işlemcilerini kullanan tek yonga FPGA sistemlerinden farklı olarak, yazılımsal işlemci çekirdekleri tasarımcılara, uygulamanın ihtiyaç durumuna göre tek bir FPGA üzerinde farklı sayılarda işlemcileri birleştirme imkanı verir [8]. FPGA'ler için hazırlanmış mikroişlemci yazılımlarından en sık kullanılanları; Xilinx firmasının Microblaze, Picoblaze ve Altera Firmasının Nios işlemcileridir [62]. Tez çalışmasında, yazılımsal işlemci olarak Microblaze sanal işlemcisi kullanıldı.

Microblaze, FPGA üzerinde gömülü uygulamalar için optimize edilmiş, IP çekirdek bloklarını birleştirerek uygulama geliştirmeyi sağlayan yazılımsal bir işlemcidir. Microblaze işlemcisi, tek bir FPGA üzerinde mümkün olan en düşük maliyetle, gerekli olan tüm sistemi kullanıcıya sunmaktadır. Tek bir FPGA geliştirme kartı ile büyük bir proje tamamlanabilmektedir. Karmaşık durum makineleri kullanmak yerine, mikroişlemci kullanmak tasarım açısından da kullanıcıya kolaylık sağlamaktadır.

Microblaze işlemcisi, 32 bitlik 32 adet genel amaçlı kaydedicilere ve 32 bitlik adres yoluna sahiptir [63]. Microblaze işlemcisi C/C++ programlama dilleri aracılığıyla kullanılmakta ve bu özellik kullanıcı için büyük avantaj sağlamaktadır. Xilinx EDK, Microblaze işlemcisi için gerekli olan derleyiciyi içermektedir. Tasarımın boyutuna göre Microblaze, blok RAM veya harici bellek üzerinde çalışabilmektedir. Microblaze çevresel birimler olmadan tek başına da kullanılabilir, fakat bu işlem gereksizdir. EDK sistemi içerisinde, yaygın olarak kullanılan çevresel birimlerin pek çoğu bulunmaktadır. Bu çevresel birimler kullanılarak pek çok farklı sistem tasarlanabilir. Bunun dışında, kullanıcılar EDK kütüphanesinde olmayan, kendi kullanım amaçlarına göre çevresel birim geliştirebilir ve bu çevresel birimi kendi işlemcisinde kullanabilirler.

Microblaze işlemcisi blok RAM'lere ve çevresel birimlere ulaşmak için farklı veri yolu bağlantıları kullanmaktadır. Microblaze işlemcisi Blok RAM'e erişimde, program ve veri bölümleri için ayrı ayrı yerel bellek yolu (Local Memory Bus, LMB) [64] kullanır. Bu bellek yolları, Xilinx tarafından tasarlanmış olan veri yerel bellek yolu (Data Side LMB, DLMB) ve komut yerel bellek yolu (Instruction Side LMB, ILMB) denetleyicileri tarafından kontrol edilen 32 bitlik yollardır. Blok RAM'e

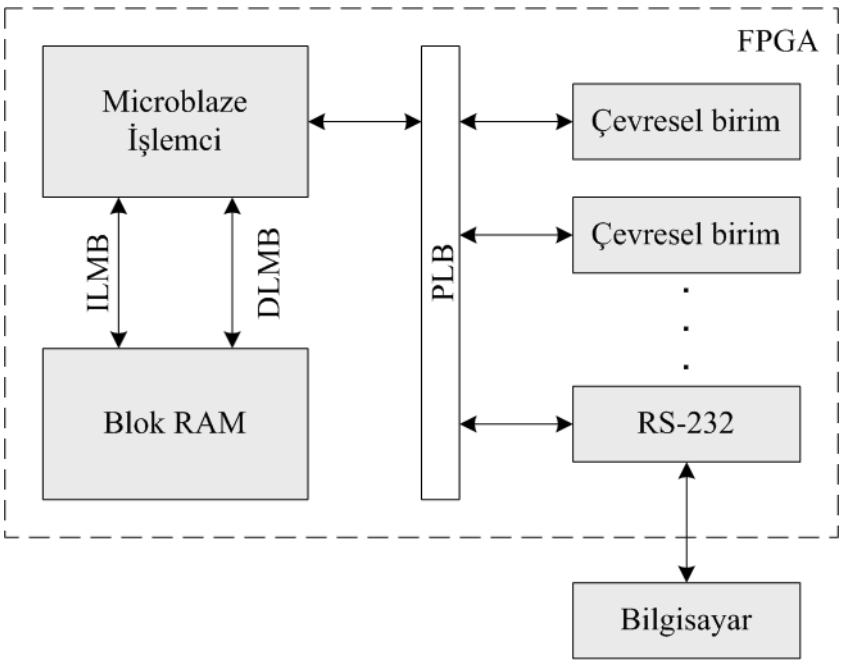
üzerinde bulunan bir tasarım, Microblaze tarafından ILMB ve DLMB yolları üzerinden başlatılmaktadır.

Microblaze tabanlı sistemlerde, çevresel birimlerle haberleşme için kullanıcı tarafından belirlenen farklı veri yolları bulunmaktadır;

- FSL (Fast Simplex Link); FIFO (first in - first out) benzeri, noktadan noktaya bir veri akışı için kullanılan bir arayüzdür. Microblaze 8 adet giriş ve 8 adet çıkış FSL arayüzünü içermektedir. Microblaze üzerindeki FSL arayüzleri 32 bit genişliğindedir [53].

- PLB (Processor Local Bus); IBM firması tarafından tek yonga sistemler için geliştirilmiş bir veri yoludur. IBM PLB, 128 bit genişliğe sahiptir ve yonga içerisinde yüksek performans gerektiren elemanlar ile işlemci arasında yüksek hızda bir arayüz sağlamaktadır. Tez çalışmasında, Microblaze için PLBv4.6'nın 32 bitlik versiyonu kullanıldı. PLBv4.6 hem PowerPC hem de Microblaze işlemcilerinde kullanılmaktadır [65].

Şekil 3.7'de, Microblaze işlemcisinin çevresel birimlerle PLB üzerinden bağlantısı ve Blok RAM ile LMB üzerinden bağlantı şeması gösterilmektedir.



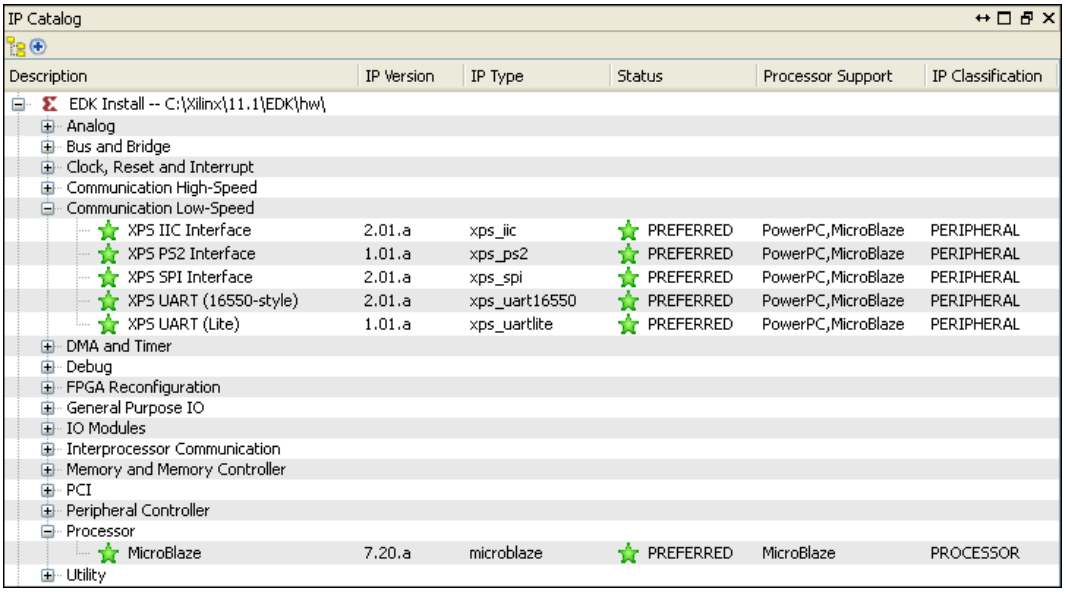
Şekil 3.7. Microblaze işlemci ve çevresel birimlerle bağlantıları

3.3.2. IP çekirdekleri (Intellectual property cores)

IP çekirdekleri, hız ve performans dikkate alınarak optimize edilmiş, karmaşık elektronik devre bloklarını içeren hazır fonksiyonlardır. Hazır IP çekirdekleri, kullanıcının istediği gömülü sistemi çok hızlı bir şekilde gerçekleştirmesine olanak vermektedir. IP çekirdekleri sayesinde, tasarım geliştirme aşaması daha kolay olmakta, yazılım derleme işlemleri azalmakta ve son kullanıcıya ulaşım daha hızlı olmaktadır. Hazır olarak kullanılabildikleri için işlevsellikleri artmaktadır.

IP donanımsal çekirdeklerin avantajı, yazılım ile sıralı işlemler yapmak yerine, donanım içerisindeki paralel işlem yapabilme yeteneğinden dolayı etkileyici bir hıza sahip olmasıdır.

FPGA ile karmaşık sistemlerin tasarımını daha basit hale getirmek için önceden tanımlanmış hazır IP çekirdekleri bulunmaktadır. Örneğin, önceden tanımlı IIC, RS-232 ve Ethernet haberleşme protokolü gibi modüller, IP çekirdekleri halinde bulunmaktadır. Tasarımcılar, bu çekirdekleri çalışmalarında uygun bir şekilde kullanabilmektedirler. FPGA üretici firmaları veya üçüncü parti satıcılardan hazır IP çekirdekleri de satın alınabilmektedir. Bunların dışında, kullanıcılar kendi özgün IP çekirdeklerini tasarlayarak, geliştirdikleri sistemlerde kullanabilmektedirler.



Şekil 3.8. Xilinx EDK'da gelen hazır IP çekirdekleri kataloğu

Şekil 3.8, Xilinx EDK platformunda hazır olarak gelen IP çekirdekleri kataloğunu göstermektedir. Kullanıcılar, buradan istedikleri IP çekirdeklerini, kendi geliştirdikleri sisteme entegre ederek, gerekli bağlantıları ve tanımlamaları yaparak kullanabilmektedirler.

IP çekirdekleri, donanımsal ve yazılımsal olmak üzere iki çeşittir. Donanımsal çekirdekler IP tasarımının fiziksel yapılarıdır. Yazılımsal çekirdekler, mantıksal yapılardır ve donanımsal çekirdeklerden daha portatif ve esnektirler.

3.4. Donanım Tanımlama Dili VHDL

VHDL (Very high speed integrated circuit Hardware Description Language), donanım tanımlama dillerinden biridir. VHDL, sayısal sistemlerdeki fiziksel donanımların modellenmesinde kullanılmaktadır. VHDL hem devre benzetimi hem de devre sentezleme için tasarlanmıştır. VHDL'de tüm sistemin benzetimi yapılabilir, fakat sistem içerisindeki bütün yapılar sentezlenemez. VHDL'de komutlar klasik programlama dillerindeki gibi ardışıl değil, aksine paralel olarak çalışmaktadır. Fakat belli yapılarla, sistemin belirli kısımlarının ardışıl olarak ta çalışması sağlanabilmektedir.

VHDL, 1980'li yıllarda Birleşik Devletler Savunması tarafından geliştirilmiştir. IEEE-1076 standardı ile Elektrik ve Elektronik Mühendisleri Enstitüsü (IEEE) tarafından standartlaştırılan bir donanım tanımlama dilidir. Bu standarda daha sonra, çok değerli lojik sistemleri tanıtmak için, IEEE-1164 isimli ek bir standart eklenmiştir. VHDL'in temel yapısal elemanları aşağıdaki gibidir [66];

Library (Kütüphane): Tasarımda kullanılacak olan kitaplıkların listesidir. Tasarımda kullanılacak olan tüm varlıklar, mimariler ve paketler kütüphane içerisinde yer almaktadır. Şekil 3.9'da örnek bir IEEE kütüphanesi gösterilmektedir. Şekilde görüldüğü gibi, bir kütüphane "LIBRARY" anahtar sözcüğü ile kütüphane içindeki paketler de "USE" ifadesiyle tanımlanmaktadır.

```
LIBRARY ieee;                    -- Kitaplık adı
USE ieee.std_logic_1164.all;     -- Paket adı
```

Şekil 3.9. VHDL'de kütüphane tanımlama

Tasarımlarda genellikle üç farklı kitaplıktan üç paket kullanılmaktadır. Bunlar;

- ieee kitaplığından std_logic_1164 paket (çok seviyeli lojik)
- std kitaplığından standart paket (veri tipleri, metin, giriş/çıkış)
- work kitaplığından work paket (tasarımın saklanacağı yer)

IEEE kitaplığı aşağıdaki paketleri içermektedir:

- std_logic_1164: STD_LOGIC (8 seviye) ve STD_ULOGIC (9 seviye) çok değerli lojik sistemleri içerir.
- std_logic_arith: SIGNED ve UNSIGNED veri tipleri ilişkili aritmetik ve karşılaştırma işlemlerini içerir.
- std_logic_signed: SIGNED tipli STD_LOGIC_VECTOR verisi ile yapılabilecek işlemler için fonksiyonları içerir.
- std_logic_unsigned: UNSIGNED tipli STD_LOGIC_VECTOR verisi ile yapılabilecek işlemler için fonksiyonları içerir.

Entity (Varlık): Devrenin tüm giriş/çıkış bacaklarının belirtildiği listedir. Devrenin portlar aracılığıyla dış dünya ile bağlantısını sağlamaktadır. Her VHDL tasarımı en az bir tane entity tanımlaması içermektedir. Şekil 3.10, bir entity tanımlamasının nasıl yapılması gerektiğini göstermektedir.

```
ENTITY varlık_adı IS
    PORT (
        port_adı : işaret_modu  işaret_tipi;
        port_adı : işaret_modu  işaret_tipi;
        … );
END varlık_adı;
```

Şekil 3.10. VHDL'de varlık tanımlama

Şekilde görüldüğü gibi, "ENTITY" ile "IS" arasında varlık adı belirtilmeli, tasarımdaki giriş/çıkış portları da "PORT" başlığı altında belirtilmelidir. İşaret_modu in (giriş), out (çıkış), inout (giriş-çıkış) ve buffer (ara bellek) olmak üzere dört farklı şekilde tanımlanabilir. İşaret_tipi ise BIT, STD_LOGIC, INTEGER gibi değerler alabilir.

Architecture (Mimari): Devrenin nasıl davranacağını belirten VHDL kodunu içermektedir. Tasarımın yerine getirmek zorunda olduğu işlemler bu bölümde gerçekleştirilir. Şekil 3.11 genel bir mimari yapısının nasıl tanımlanması gerektiğini göstermektedir.

```
ARCHITECTURE mimari_adı OF varlık_adı IS
    [bildirimler]
BEGIN
    (kod)
END mimari_adı;
```

Şekil 3.11. VHDL'de mimari tanımlama

Şekilde görüldüğü gibi "ARCHITECTURE" ile "OF" arasında mimari adı, "OF" ile "IS" arasında da "ENTITY" kısmında tanımlanan varlık adı yazılmalıdır. "bildirimler" bölümü isteğe bağlı olarak sinyaller ve sabitlerin tanımlandığı bölümdür. Tasarımın ana kodu "kod" ifadelerinin yer aldığı bölüme yazılır.

Şekil 3.12, VHDL ile tüm tanımlamaların yapıldığı, örnek bir "VE (AND)" kapısı tasarım örneğini göstermektedir. Şekilde görüldüğü gibi, "a" ve "b" olmak üzere tek bitlik iki giriş ve "c" adında tek bitlik bir çıkış portu tanımlanmıştır. Aşağıdaki tasarım ile, FPGA içerisinde donanımsal olarak çalışacak olan bir VE kapısı gerçekleştirilmiş olur.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------------------------------
ENTITY ve_kapisi IS
    PORT ( a,b : IN BIT;
           c : OUT BIT);
END ve_kapisi;
-----------------------------------------------------------------
ARCHITECTURE kapi_mimari OF ve_kapisi IS
BEGIN
    c <= a AND b;
END kapi_mimari;
```

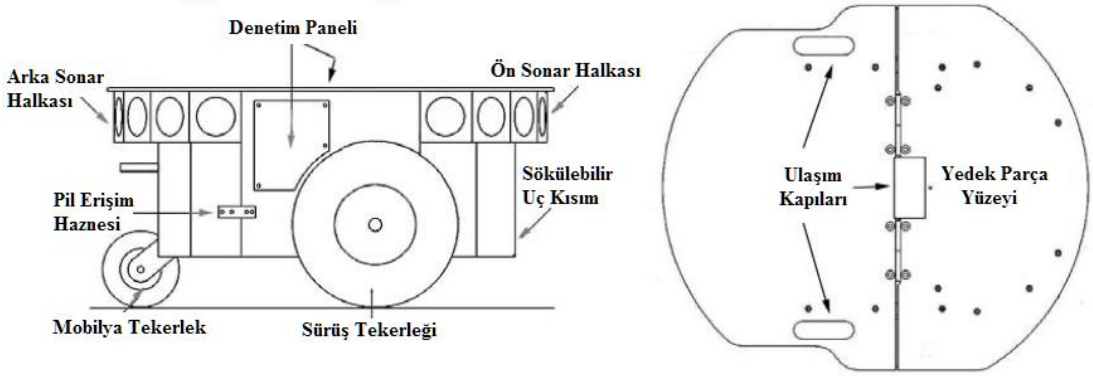
Şekil 3.12. VE (AND) kapısının VHDL ile tasarımı

4. PIONEER 3-DX GEZGİN ROBOT

4.1. Gezgin Robotun Genel Özellikleri

Pioneer 3-DX [67] akademik çalışmalar için kullanılan son derece popüler bir araştırma geliştirme gezgin robotudur. Pek çok araştırmacı bu robotu akademik çalışmalarında kullanmışlardır [68-71]. Pioneer 3-DX gezgin robot Adept MobileRobots firması tarafından üretilmektedir. Sert alüminyum gövdeye sahip robot üzerinde iki adet diferansiyel ve bir adet mobilya tekerlek bulunmaktadır. Şekil 4.1 ve 4.2'de, tez çalışmasında kullanılan Pioneer 3-DX gezgin robotun genel görünümü ve genel özellikleri gösterilmektedir.

Şekil 4.1. Pioneer 3-DX gezgin robot



Şekil 4.2. Pioneer 3-DX gezgin robotun genel özellikleri [72]

Pioneer 3-DX gezgin robotta bulunan standart özellikler aşağıda verilmektedir;

- Yüksek çözünürlükte kodlayıcılar,
- 1 adet RS-232 seri haberleşme portu,
- 3 adet batarya,
- 8 adet ön sonar algılayıcı,
- Reset ve motor butonları,
- Kullanıcı denetim paneli,
- Yüksek performanslı mikrodenetleyici.

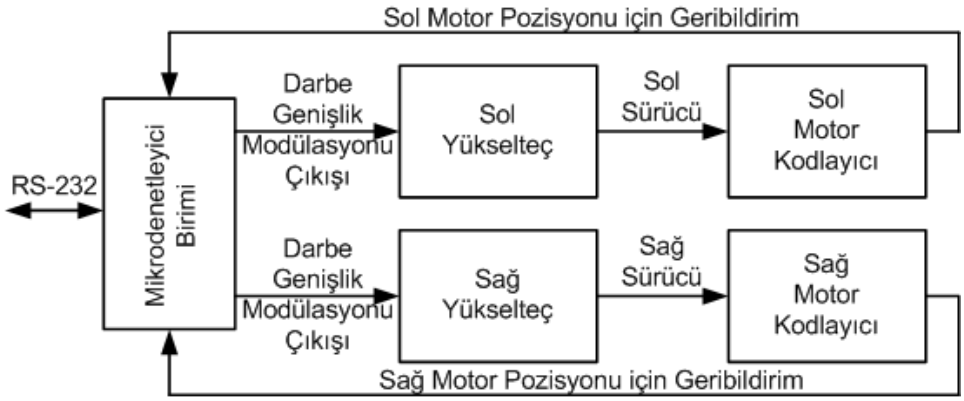
Ayrıca yukarıda verilmiş olan standart özellikler yanında, isteğe bağlı olarak ek donanımlar da robota eklenebilmektedir. Bu donanımlardan bazıları;

- 8 adet arka sonar algılayıcı
- Uzaktan kontrol için kablosuz seri/ethernet dönüştürücü,
- Robot kolları ve tutucular (gripper),
- Lazer mesafe ölçücü,
- Mono ve stereo kamera,
- Kontrol için kumanda kolu (joystick),
- Pusula,
- Tümleşik bilgisayar.

Pioneer 3-DX gezgin robot 9 kg ağırlığındadır ve uzunluğu 44,5cm, genişliği 39,3cm, yüksekliği 23,7cm'dir. Sonar algılayıcıların yerden yüksekliği 18,5cm ve robotun kendi etrafında dönüş yarıçapı 26cm'dir. Ulaşabileceği maksimum hız 1,6m/s'dir. Gezgin robot düşük hızda 23kg'a kadar yük taşıyabilmektedir. Tam dolu 3 batarya ile aralıksız olarak 8-10 saat arası çalışabilmektedir [72].

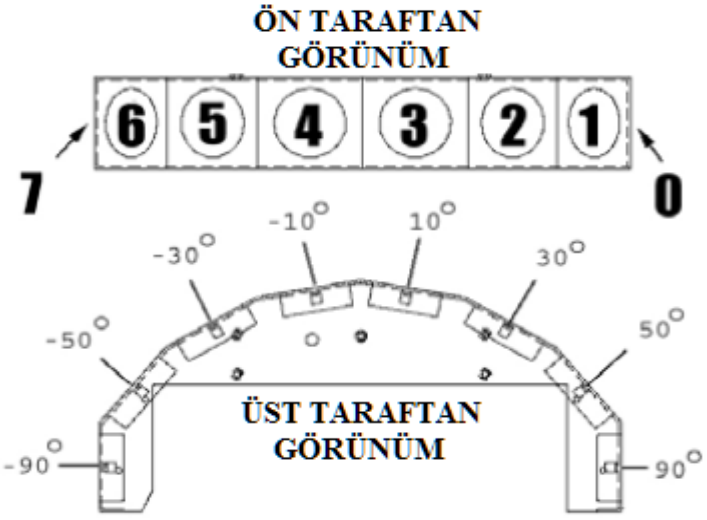
Pioneer 3-DX gezgin robot, yüksek hızlı, yüksek torklu ve çift yönlü doğru akım (DA) motorlara sahiptir. DA motorların üzerinde her biri hassas konum, hız algılama ve gelişmiş konum tahmini için yüksek çözünürlüklü optik şaft kodlayıcılar bulunmaktadır. Kullanılan bu kodlayıcılar bir dönüşte 500 adet darbe (pals) üretebilme özelliğine sahiptirler. Bu motorların bağlı olduğu, 38,3:1 dişli oranına sahip, iki adet dişli kutusu bulunmaktadır. Robot üzerinde dolma teker

kullanılmaktadır. Bu tekerleklerin çapları 19,53cm'dir [72]. Şekil 4.3'de robotun basitleştirilmiş iç yapısı görülmektedir.



Şekil 4.3. Pioneer 3-DX gezgin robotun basitleştirilmiş iç yapısı [73]

Pioneer 3-DX gezgin robotun önünde, nesne algılama ve tanıma, çarpışmaları önlemek için mesafe bilgisine sahip olma, lokalizasyon ve gezinim gibi işlemleri gerçekleştirmek için kullanılan sonar algılayıcılar standart olarak bulunmaktadır. Bunun dışında isteğe bağlı olarak robotun arkasına da sonarlar eklenebilmektedir. Bu ses ötesi algılayıcılar sayesinde, hemen hemen 360 derecelik bir görüş açısı sağlanmaktadır. Algılayıcıların hassasiyet aralığı yaklaşık 10cm ile 5m arasındadır. Şekil 4.4'de robotun ön tarafında yer alan 8 adet sonar algılayıcının dizilimi görülmektedir. Aynı dizilim, isteğe bağlı olarak, robotun arka tarafına da yerleştirilebilmektedir.



Şekil 4.4. Pioneer 3-DX gezgin robot sonar dizisi [72]

4.2. Gezgin Robot ile Haberleşme

Pioneer 3-DX, tüm hareketlerinin denetlenebilmesi için, ARIA (Advances Robotics Interface for Applications) isimli bir yazılım geliştirme aracını kullanmaktadır. ARIA otonom gezgin robotlar için, nesneye yönelik C++ tabanlı açık-kaynak kodlu bir yazılımdır. ARIA ile çalışan programcılar; sınıflar, basit kalıtım ile nesneleri, işaretçiler, hafıza yönetimi ve derleme gibi tipik C++ kavramlarına aşina olmalıdırlar [74]. Robotun kendisi herhangi bir üst seviye görevi gerçekleştirmez. Engel tanıma ve kaçınma, lokalizasyon, haritalama, akıllı navigasyon ve kamera kontrolü gibi görevleri gerçekleştirmek için robota bağlı bir bilgisayar üzerinde çalışan akıllı istemciye ihtiyaç vardır.

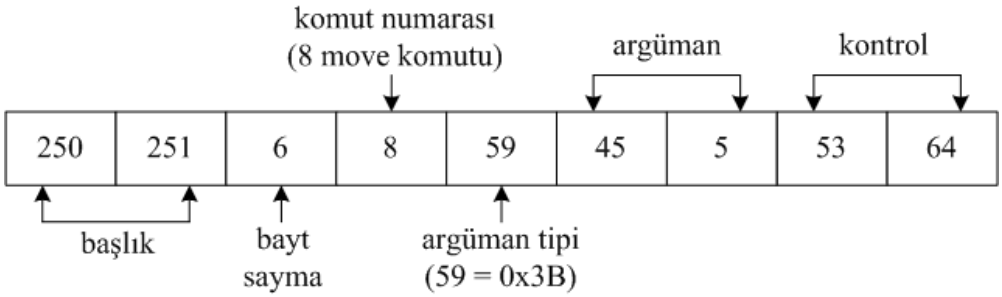
ARIA dinamik olarak robotun hızını, başlangıç açısını, başlangıç pozisyonuna göre mevcut açı bilgilerini ve diğer hareket parametrelerini, ya basit düşük seviye komutlarla ya da üst seviye hareketlerle kontrol edebilmektedir. ARIA ayrıca, robot tarafından sağlanan pozisyon tahmin verilerini, sonar algılayıcı verilerini ve diğer tüm güncel işletim verilerini alıp değerlendirebilmektedir [75].

ARIA dışında robot, ARCOS (Advanced Robot Control and Operations Software) istemci-sunucu arayüzü ile de kontrol edilebilmektedir [72]. Pioneer 3-DX platformu, ARCOS işletim sistemine özel istemci-sunucu haberleşme protokolü aracılığıyla, bir istemci ile haberleşebilmektedir. İstemciden robot üzerindeki sunucuya komut paketleri ve sunucudan istemciye sunucu bilgi paketleri (Server Information Packets, SIPs) ile haberleşme sağlanmaktadır. İstemciden sunucuya gönderilen komut paketlerinin bileşenleri, genişlikleri ve alabileceği değerler Tablo 4.1'de gösterilmektedir.

Tablo 4.1. İstemci komut paket protokolü

Bileşen	Genişlik (bayt)	Değer
Başlık	2	0xFA, 0xFB
Bayt sayma	1	N
Komut numarası	1	0 – 255
Argüman tipi	1	0x3B, 0x1B, 0x2B
Argüman	n	veri
Kontrol	2	hesaplanan

Paketin başlangıcını ifade eden iki bayt uzunluğundaki başlık alanı, hem istemci komutu hem de SIP için aynıdır ve sırasıyla 0xFA (250) ve 0xFB (251) değerlerini içerir. Bayt sayısı alanı, başlık ve kendisi hariç, kendisinden sonra gelen tüm alanların toplam kaç bayt olduğunu göstermek için kullanılmaktadır. Komut numarası alanı, ilerleme ve dönme gibi robotun yapacağı eylemin numarasını ifade etmektedir. Argüman tipi alanı, robotun ileri yönde ya da geri yönde hareket etmesini belirlemek için pozitif tamsayı veya negatif tamsayı gibi değerler içermektedir. Yine robotun dönüş yaparken, sol taraftan mı yoksa sağ taraftan mı dönmesi gerektiği argüman tipi alanı içerisinde yer almaktadır. Argüman alanı, robotun yerine getireceği herhangi bir eylemin miktarını belirtmek için kullanılmaktadır. Örneğin, robotun kaç derece döneceği ya da ne kadar ilerleyeceği bu alan içerisinde verilmektedir. Kontrol alanı ise, gönderilmek üzere olan bu paketin hata kontrol bilgilerini (error check-sum) içermektedir. Bu alan sayesinde, sunucu ve istemci arasında taşınan paketin kablo üzerindeki taşıma güvenliği sağlanmış olmaktadır. Şekil 4.5, örnek bir istemci komut paketini göstermektedir. Şekilde de görüldüğü gibi, kontrol alanı, komut numarası ile argümanın ilk baytı toplamı olan 53 ve argüman tipi ile argümanın ikinci baytı toplamı olan 64 değerlerini içerecek şekilde oluşturulmaktadır.



Şekil 4.5. İstemci komut paketi

Robot ile iletişim, üzerinde bulunan seri port üzerinden RS-232 bağlantısı kurularak sağlanmaktadır. Ancak iletim kurulduktan sonra, istemci robot üzerindeki sunucuya hareket komutlarını göndermekte ve sunucudan da SIP paketleri ile işletim bilgilerini almaktadır.

Robot ilk çalıştırıldığında ya da sıfırlandığında, ARCOS istemci-sunucu bağlantısını kurmak için haberleşme paketlerini özel bir bekleme durumunda dinler. Bağlantı oluşturmak için istemci uygulaması, sırası ile SYNC0, SYNC1 ve SYNC2

komutlarını içeren senkronizasyon paketleri serisi yollar ve sunucudan cevap bekler. Senkronizasyon için gerekli olan komut dizisi aşağıda gösterilmektedir;

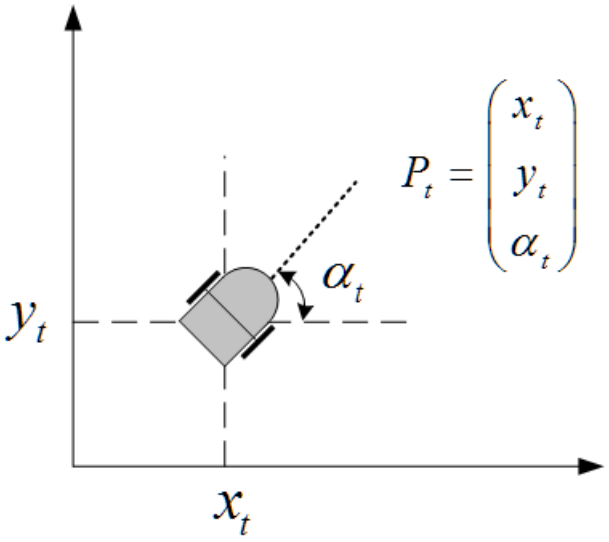
- SYNC0: 250, 251, 3, 0, 0, 0
- SYNC1: 250, 251, 3, 1, 0, 1
- SYNC2: 250, 251, 3, 2, 0, 2

Tez çalışmasında, Pioneer 3-DX gezgin robot ile iletişim için ARIA'nın C++ kütüphanesi yerine Matlab programı kullanıldı. Matlab sayısal hesaplama, görüntüleme ve programlama için yüksek seviyeli bir dil ve etkileşimli bir ortamdır. Yerleşik matematik fonksiyonlar ve araçlarla Matlab, C/C++ veya Java gibi geleneksel programlama dillerinden daha hızlı sonuca ulaşmaktadır [76]. Uygulamaların Matlab ile geliştirilmesi, ARIA'nın C++ kütüphanesindeki fonksiyon karmaşıklıklarını önlemekte, gerçek zamanlı dinamik çalışmalar için uygulama programlarını sık sık yeniden derleme gereksinimini ortadan kaldırmaktadır.

Robotla haberleşme için, Matlab'da yazdığımız bir istemci program, robota doğrudan komut paketlerini yollamaktadır. Robot ile haberleşme için RS-232 portu kullanılmaktadır. RS-232 portu vasıtasıyla haberleşme başlar başlamaz, robot sürekli olarak, her 100 ms'de bir SIP yollamaktadır. Her bir SIP içerisinde, o an için mevcut koordinat ve açı (x, y, α) değerleri ve sonar bilgileri gibi robota ait birçok bilgi yer almaktadır. Bu şekilde robot ile ilgili veriler elde edilebilmektedir. Robota herhangi bir komut gönderildiğinde, robotun hareket motorları çalışır ve hareket tamamlanana kadar robot başka bir komutu çalıştırmaz. Bu durumda, her gönderilen komuttan sonra motorların durumu, ancak hareket tamamlandıktan sonra kontrol edilebilmektedir. Bu problemin üstesinden gelebilmek için, SIP paketleri vasıtasıyla, sürekli olarak robotun dinlenmesi yeterlidir.

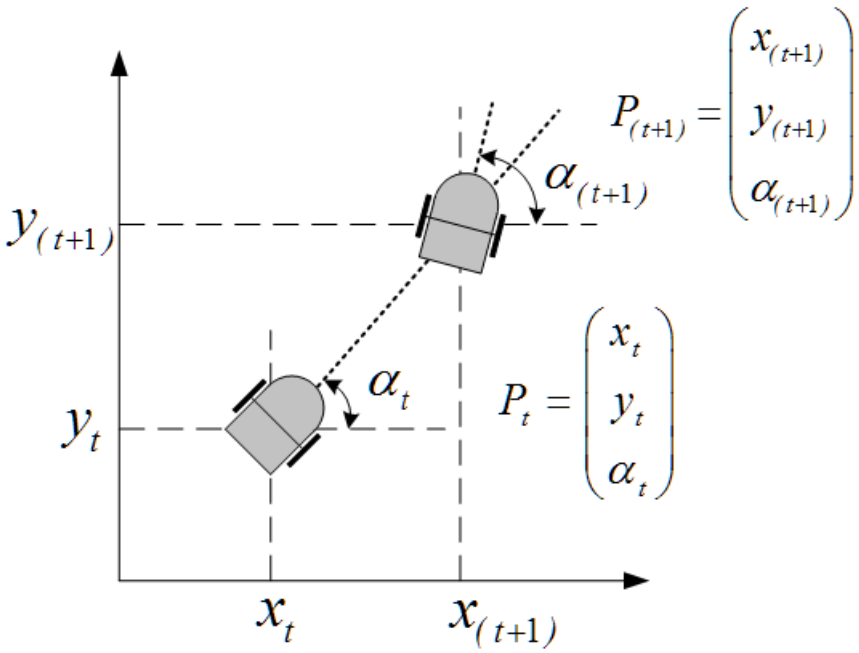
4.3. Gezgin Robotun Hareket Modeli

Robotun ortamdaki duruşu x_t, y_t, α_t parametreleriyle tanımlanır. x_t ve y_t parametreleri robotun t anında bulunduğu koordinatları, α_t ise robotun x düzlemine göre açısal konumunu ifade etmektedir. P_t robotun koordinat ve açı değişkenlerine bağlı olarak konumunu göstermektedir. Şekil 4.6'da robotun konum ve açısı gösterilmektedir.



Şekil 4.6. Robotun konum ve açısı

Şekil 4.7'de görüldüğü gibi, gönderilen komut içerisindeki parametrelere bağlı olarak, robot (x_t, y_t) noktasından $(x_{(t+1)}, y_{(t+1)})$ noktasına hareket etmiştir. İlk durumdaki α_t açısı yatay düzleme göre $\alpha_{(t+1)}$ olarak değişmiştir.



Şekil 4.7. Hareket eden robotun yeni konum ve açısı

Tez çalışmasında, gezgin robotu bir noktadan diğer bir noktaya hareket ettirmek için, iki nokta arasındaki açı ve mesafe bilgileri kullanıldı. Hesaplanan mesafe ve açı bilgileri, istemci uygulama programından RS-232 portu vasıtasıyla robot sunucusuna gönderilmekte ve robotun hareketi sağlanmaktadır.

5. YOL BULMA SİSTEMİNİN FPGA ÜZERİNDE GERÇEKLENMESİ

Bu bölümde, tez çalışmasında geliştirilen yol bulma sisteminin gerçekleştirilmesi yer almaktadır. Geliştirilen sistem, bütün olarak bir blok diyagramı ile gösterilmektedir. Genel olarak, sistemin bileşenleri olan; bir kameradan alınan görüntüler aracılığıyla çalışma ortamının haritasının oluşturulması, elde edilen haritaya göre başlangıç ve hedef nokta arasında en kısa yolun FPGA üzerinde bulunması, bulunan yolun gezgin robota gönderilerek robotun bu rota üzerinde hareketinin sağlanması anlatılmaktadır. Yol bulma görevini yerine getiren, özgün bir GA IP çekirdeği oluşturmak için gerekli olan işlem basamakları verilmektedir.

FPGA ile ilgili çalışmalar; Xilinx ISE, XPS ve SDK programları kullanılarak geliştirildi. Tasarımda, benzetim gerektiren çalışmalar için Modelsim programı kullanıldı. Sistemin gerçekleştirilmesi için; öncelikle yazılımsal mikroişlemci tabanlı bir uygulama, daha sonra hem yazılımsal mikroişlemci tabanlı hem de donanımsal IP çekirdek içeren hibrit bir uygulama, en sonra da tamamı donanımsal IP çekirdek içeren bir uygulama geliştirildi. GA için zaruri olan rastgele sayı üretimi FPGA üzerinde gerçekleştirildi. Geliştirilen GA IP çekirdeğinin FPGA üzerinde başarılı bir şekilde çalıştığı görüldükten sonra, sistem gerçek bir laboratuvar ortamında robot ve kamera kullanılarak gerçekleştirildi. Ayrıca, benzetimde ve gerçeklenen sistemde kullanmak üzere Matlab tabanlı kullanıcı arayüzü hazırlandı.

5.1. Tez Çalışmasında Gerçekleştirilen Yol Bulma Sistemi

Laboratuvar ortamında gerçekleştirilen çalışma için, laboratuvar tavanına bir adet kamera yerleştirildi. Kameranın görüş açısı dahilinde, robotun hareket edebileceği sınırlar tespit edilerek işaretlendi. Buna göre, yerdeki zemin kaplama parkelerinin her biri birer hücre kabul edilerek, 10x8'lik bir alan çalışma ortamı olarak düzenlendi. Kameranın yüksekliğine ve görüş açısına bağlı olarak bu alan genişletilebilmektedir. Şekil 5.1'de, laboratuvar ortamında hazırlanmış olan engellerin olduğu bir çalışma alanı ve Pioneer 3-DX gezgin robot farklı açılardan görülmektedir.

Şekil 5.1. Çalışma alanı görüntüsü

Laboratuvar ortamında hazırlanan kameralı sistemde, çalışma alanı üzerindeki yeşil renkli hücreler engel bölgelerini, mavi renkli hücreler çalışma alanı sınırlarını ifade etmektedir. Robot üzerinde yer alan sarı renk tek başına robotun konumunu, sarı ve kırmızı renk birlikte robotun *x* eksenine göre doğrultusunu belirlemek için kullanılmaktadır. Geliştirilen yol bulma çalışmasında, çalışma alanı içerisinde kalan ve engel olmayan hücrelerde robotun hareket etmesine izin verilmektedir.

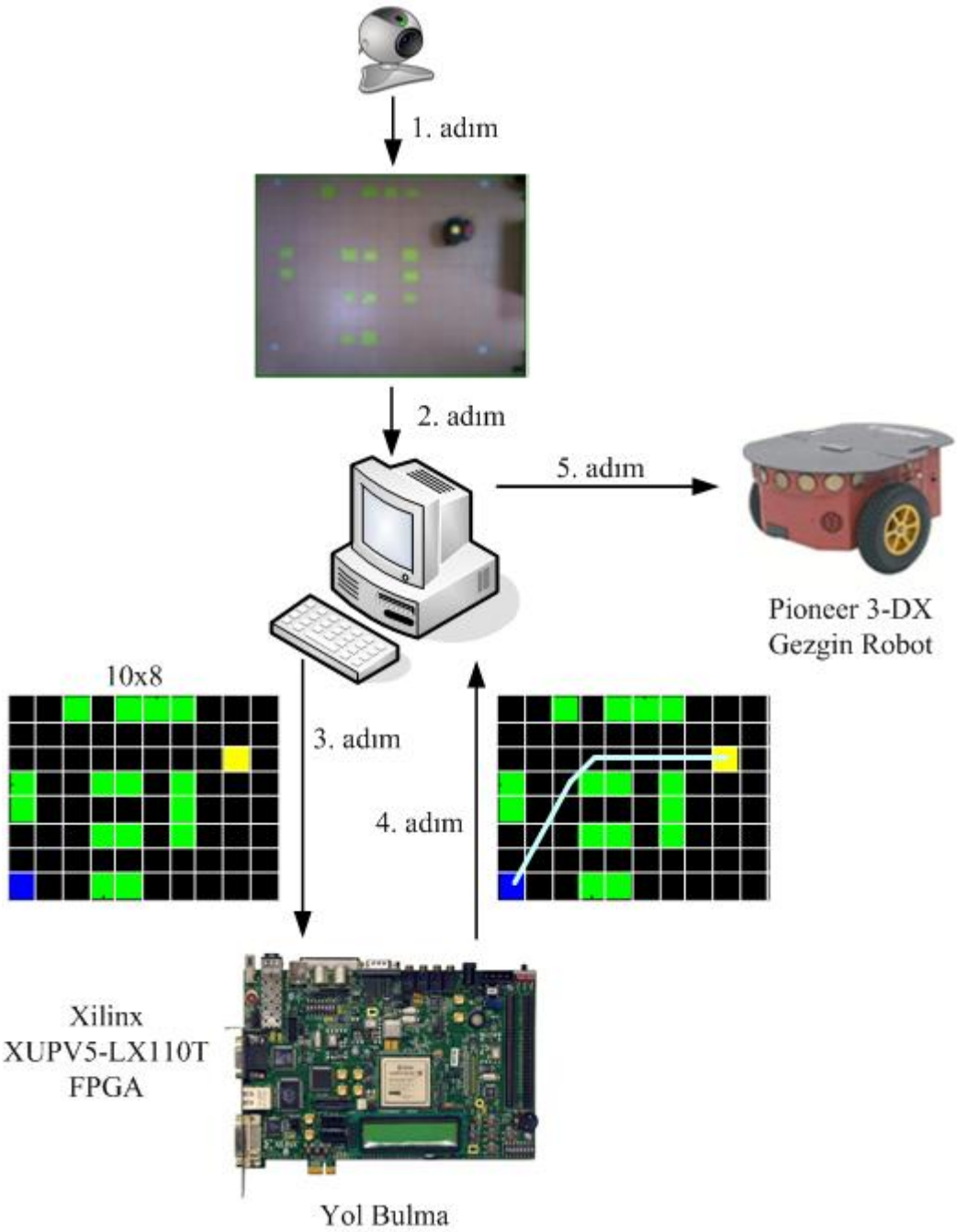
Geliştirilen yol bulma sisteminin işlem basamakları, çok genel bir gösterimle, Şekil 5.2'de belirtilen adımları yerine getirmektedir;

Birinci adımda, laboratuvar tavanına yerleştirilmiş olan bir kamera ile ortamın görüntüsü elde edilmektedir. Elde edilen görüntü, sadece bir fotoğraf olup mavi, yeşil, sarı, kırmızı ve zemin renklerini içermektedir.

İkinci adımda, bu fotoğraf, kameranın bağlı olduğu bir bilgisayarda görüntü işleme yöntemleri ile işlenerek, matris şeklinde bir harita bilgisine dönüştürülmektedir. Fotoğraftaki renklerin yoğunlaştığı pikseller tespit edilip, bu piksellerin çalışma alanında hangi hücreye ait olduğu hesaplanmaktadır. Matris olarak elde edilen haritada, '0' robotun hareket edebileceği hücreleri, '1' engel ihtiva eden hücreleri, '2' robotun konumunu ve '3' ise hedefi göstermektedir.

Üçüncü adımda, oluşturulan harita bilgisayardan FPGA'deki Microblaze sanal işlemciye gönderilmektedir. Microblaze de, aldığı harita bilgisine göre en kısa yolun hesaplanması için, haritayı FPGA'de oluşturulan GA IP çekirdeğine vermektedir ve burada GA ile yol bulma işlemi gerçekleştirilmektedir.

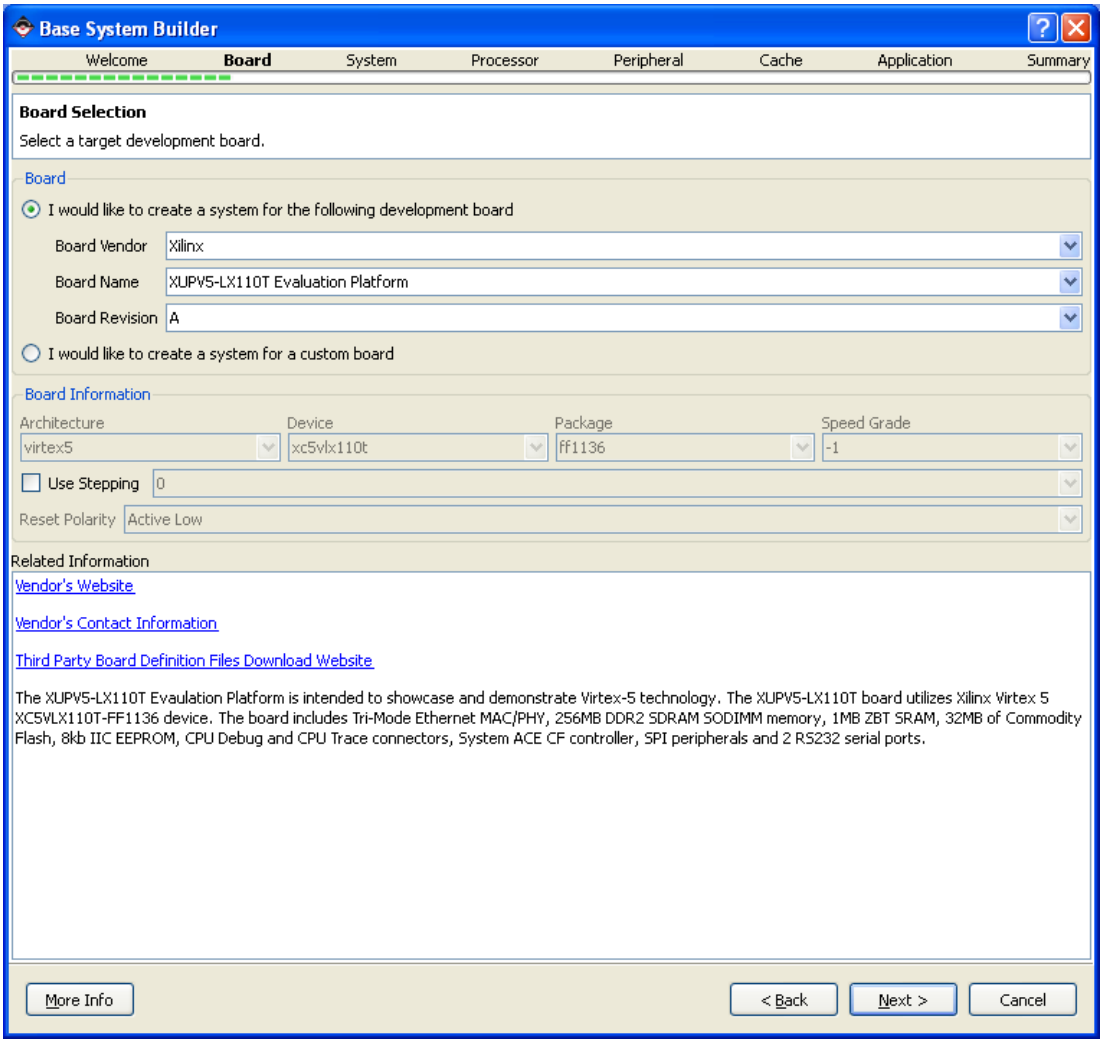
Dördüncü adımda, yine Microblaze vasıtasıyla, bulunan yol bilgisi bilgisayara gönderilmektedir. En son adımda ise, bulunan yol koordinatları hareket komutlarına dönüştürülüp RS-232 üzerinden Pioneer 3-DX gezgin robota gönderilmekte, robotun yol koordinatları boyunca hareket edip hedefe ulaşması sağlanmaktadır.



Şekil 5.2. Gerçekleştirilen laboratuvar uygulamasın işlem basamakları

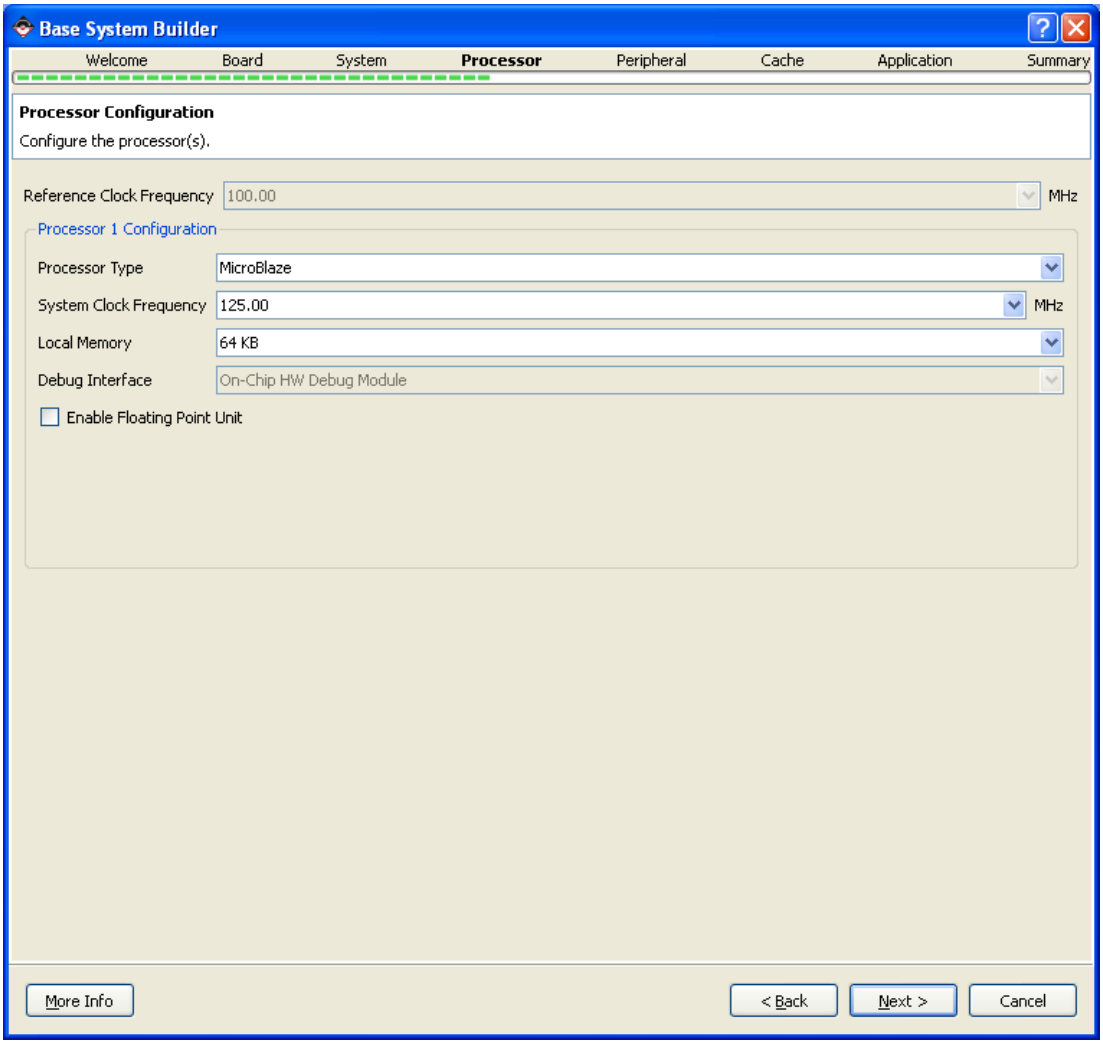
5.2. FPGA Üzerinde Mikroişlemci Tabanlı Gömülü Sistem Oluşturma

FPGA üzerinde mikroişlemci tabanlı gömülü sistem oluşturmak için, Xilinx Platform Studio (XPS)'da bulunan Base System Builder (BSB) sihirbazı kullanılmaktadır. Bu sihirbaz, tasarım aşamasını basitleştirmekte ve birkaç adıma düşürmektedir. BSB sihirbazını kullanarak, yeni bir tasarım oluştur seçeneği seçildikten sonra, Şekil 5.3'de gösterilen ekranda, kullanılan FPGA geliştirme kartının üreticisi ve modeli seçilmektedir.



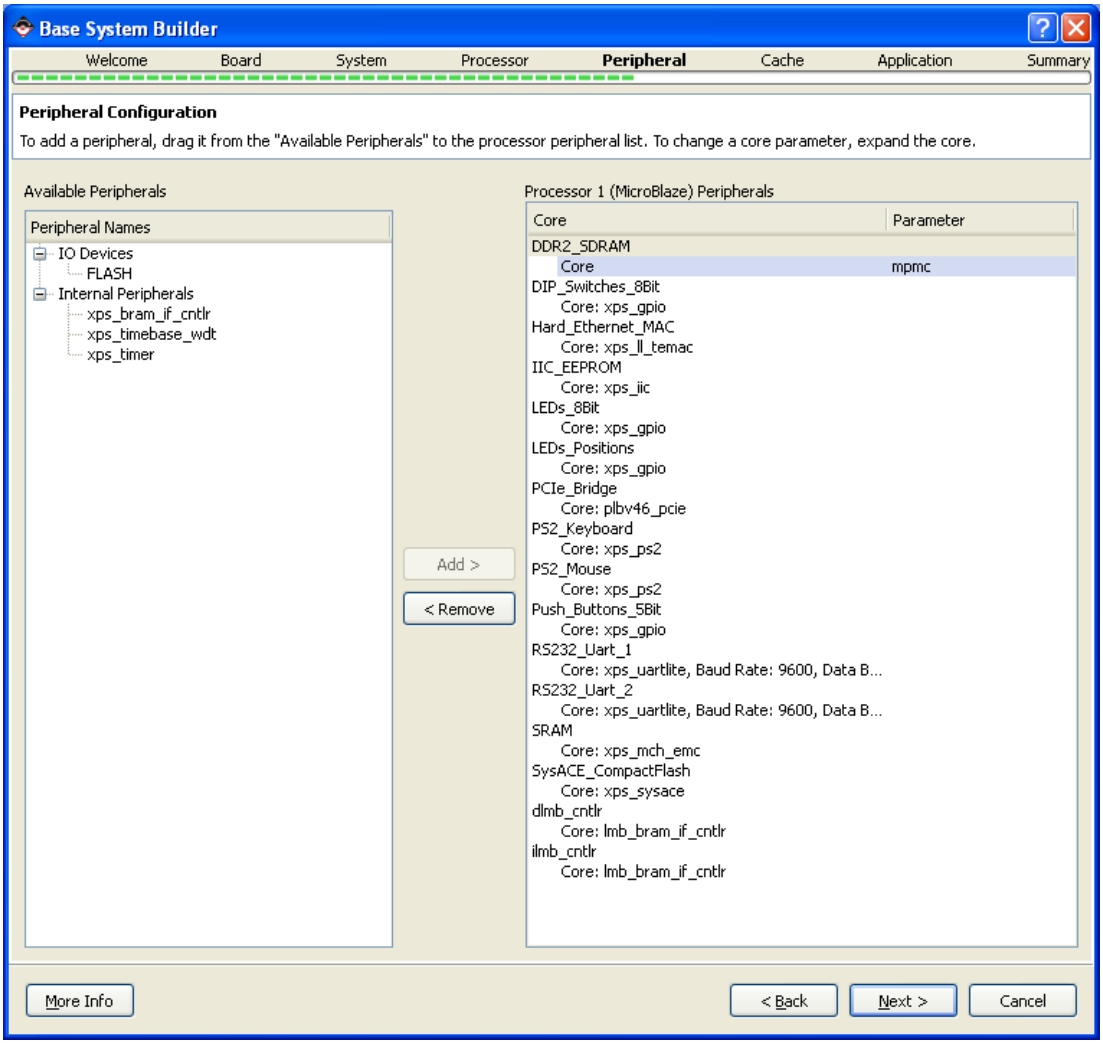
Şekil 5.3. BSB sihirbazı ile FPGA geliştirme kartının seçimi

FPGA geliştirme kartı tanıtım işlemi bittikten sonra; işlemci tipi, sistem saat frekansı ve dahili bellek boyutu belirlenmektedir. Şekil 5.4'te de gösterildiği gibi, tasarlamış olduğumuz çalışmada; işlemci tipi olarak Microblaze işlemcisi, saat frekansı olarak 125 MHz ve yerel bellek boyutu olarak da 64 KB dahili bellek seçilmiştir.

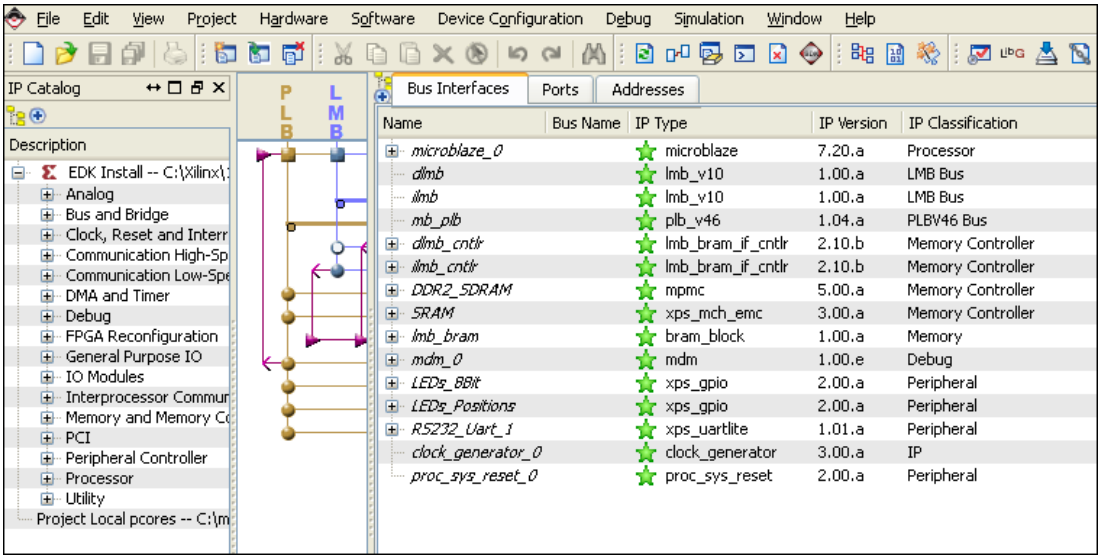


Şekil 5.4. BSB sihirbazında işlemci, frekans ve bellek seçimi

Bir sonraki adımda, seçilmiş olan işlemciye, istediğimiz çevresel donanım birimlerini ekleyebileceğimiz bir ekran gelmektedir. Şekil 5.5 ile gösterilen bu ekranın sol tarafında, eklenmesi mümkün olan tüm donanımların listesi verilmektedir. Ekranın sağ tarafında ise eklenmek istenen donanımların listesi yer almaktadır. Bu ekranda, kullanılmak istenen çevresel birimler eklenip, istenmeyenler çıkartılabilir. Tasarım aşamasından sonra da sisteme çevresel birimler ve IP çekirdekleri eklenip, çıkartılabilmektedir. Sisteme eklenen çevresel birimlerin parametrelerinde de değişiklikler yapılabilmektedir.



Şekil 5.5. BSB sihirbazı ile işlemciye çevresel birimlerin eklenmesi



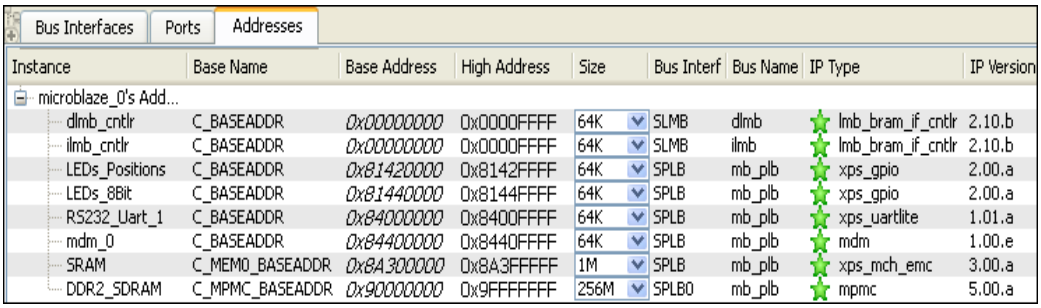
Şekil 5.6. Oluşturulan Microblaze işlemcisisi, çevresel birimleri ve bağlantı arayüzünün XPS'de görüntülenmesi

Sistemin sihirbaz aracılığıyla tasarım aşaması tamamlandıktan sonra, sisteme ait çevresel birimler XPS programında, Şekil 5.6'da gösterildiği gibi, hazır hale gelmektedir. Programda, oluşturulan gömülü sisteme ait çevresel birimler, IP çekirdekleri, bağlantılar, hazır IP çekirdekleri listesi, portlar ve çevresel birimlerin adres bilgileri gibi pek çok bilgi görüntülenebilmektedir.

Tablo 5.1, Microblaze tabanlı olarak tasarlanan sistemdeki çevresel birimlerin bağlantı açıklamalarını, Şekil 5.7'de oluşturulan sistemdeki birimlerin adres haritasını göstermektedir.

Tablo 5.1. Microblaze tabanlı sistemin bağlantıları

Blok adı	Görevi
Microblaze_0	Microblaze işlemcisi
dlmb	İşlemci belleği için veri arayüzü (sadece BRAM)
ilmb	İşlemci belleği için komut arayüzü (sadece BRAM)
mb_pld	Çevresel birimlerin bağlantısı için veri yolu
dlmb_cntrl	İşlemci belleği veri yolu için kontrol bloğu
ilmb_cntrl	İşlemci belleği komut yolu için kontrol bloğu
LEDs_8Bit	Yazılım aşamalarını test için kullanılan LED kontrol bloğu



Şekil 5.7. Sistemin adres haritası

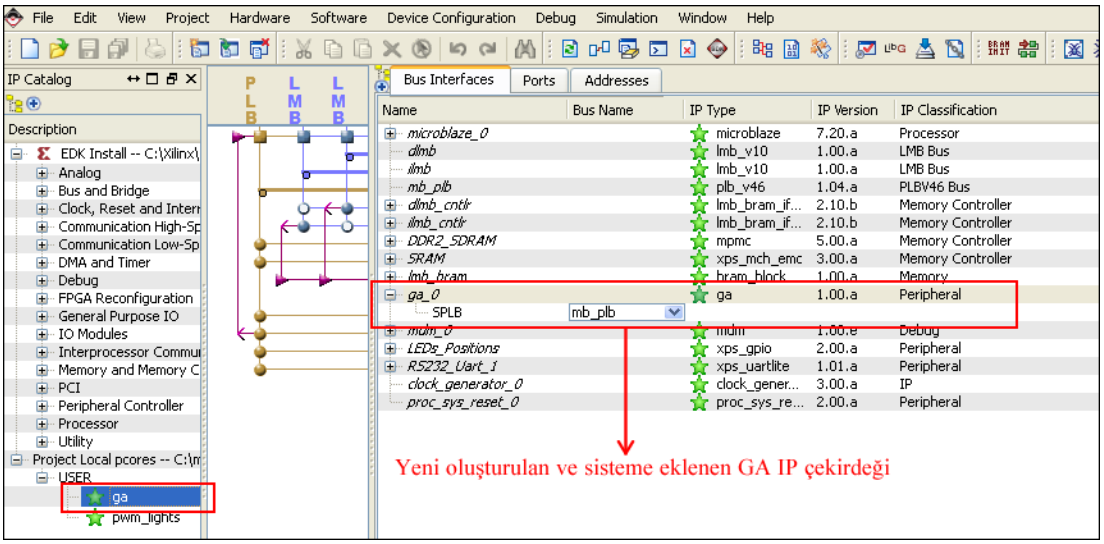
5.3. Özgün Bir GA IP Çekirdeği Oluşturma

Özgün bir IP çekirdeği oluşturmak için, XPS programında bulunan Create or Import Peripheral (CIP) sihirbazı kullanılmaktadır. IP çekirdeği oluştururken dikkat edilmesi gereken durumlardan birisi, çekirdeğin hangi veri yolu ile sisteme bağlanacağını

belirlemektir. Çalışmada, GA IP çekirdeğinin diğer çevresel birimlerle haberleşmesini sağlamak için PLB veri yolu seçilmiştir.

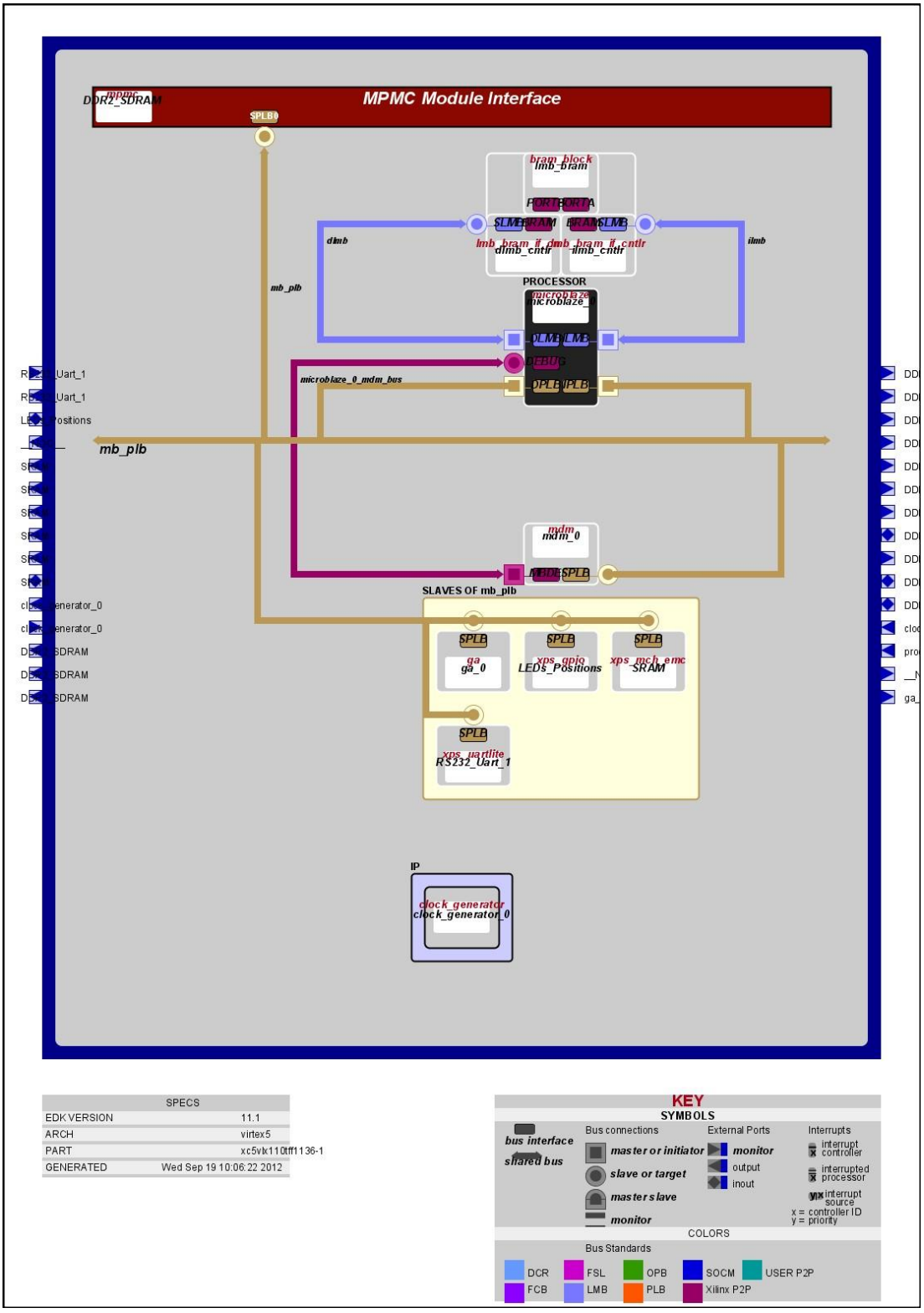
IP çekirdeği oluşturma işlemi tamamlandıktan sonra, program tarafından otomatik olarak iki adet VHDL dosyası üretilmektedir. Dosyalardan bir tanesi, IP çekirdek için belirlenen isimde "IP_çekirdek_adı.vhd", diğeri de "user_logic.vhd" ismindedir. "IP_çekirdek_adı.vhd" dosyası veri yolu arayüzü mantık birimini içerir. IP çekirdeğinin, portlar aracılığıyla dışarıya bağlantısı sağlanacaksa, port isimleri bu dosyaya eklenmektedir. "user_logic.vhd" dosyası, gerekli mantık yapıları ve kodlamalar eklenerek, özgün IP çekirdeğinin fonksiyonlarının tanımlandığı şablon dosyadır.

Çalışmada özgün bir GA IP çekirdeği oluşturuldu ve oluşturulan GA IP çekirdeği Microblaze tabanlı sisteme eklendi. Tasarlanan GA IP çekirdeği, IP Arayüzü (Intellectual Property Interface, IPIF) üzerinden PLBv46 veri yoluna bağlanmaktadır. IP çekirdeği oluşturma işlemi tamamlandıktan sonra, yeni IP çekirdeğinin sisteme eklenmesi için tekrar CIP sihirbazı kullanılmaktadır. Şekil 5.8'de, oluşturulan ve sisteme eklenen GA IP çekirdeği ve bağlantı türü gösterilmektedir.



Şekil 5.8. Sisteme eklenen GA IP çekirdeği ve bağlantısı

Xilinx XPS programında tasarlanan sistemin blok diyagramı Şekil 5.9'da gösterilmektedir. Burada, sistem tasarımı yapılırken, sisteme eklenen çevresel birimler ve bağlantıları gösterilmektedir. Bu blok diyagramı, tasarımdan sonra XPS tarafından otomatik olarak oluşturulmaktadır.



Şekil 5.9. XPS'de tasarlanan sistemin blok diyagramı

5.4. FPGA Üzerinde Rastgele Sayı Üretimi

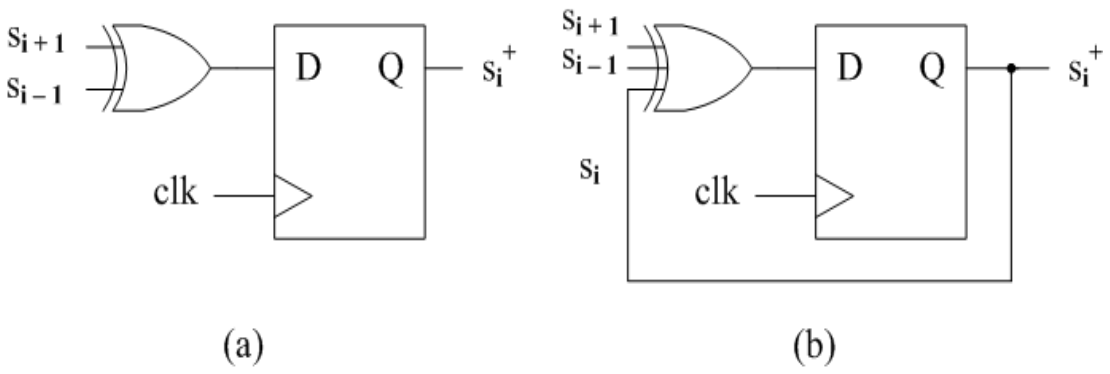
GA'da; başlangıç nüfus oluşturma, seçim, çaprazlama ve mutasyon gibi operatörlerde rastgele sayılar yoğun olarak kullanılmaktadır. Donanım olarak gerçekleştirilecek olan GA'da kullanmak üzere, rastgele sayıların yine donanım olarak üretilmesi gerekmektedir.

Rastgele sayı üreteci, herhangi bir düzenli forma sahip olmayan, seri olarak rastgele sayılar veya semboller üreten araçlar olarak tanımlanır. Literatürde, rastgele sayıları üretmek için pek çok yöntem geliştirilmiştir. Bunlardan bazıları; Blum Blum Shub, Inverse Congruential Generator, Lagged Fibonacci Generator, Linear Congruential Generator, Linear Feedback Shift Register (LFSR) yöntemleridir.

Genellikle rastgele sayı üretiminde iki tip algoritma kullanılmaktadır [77]; LFSR ve doğrusal hücresel otomat (linear cellular automata). Hücresel otomat'ın LFSR'den daha iyi rastgele sayılar ürettiği belirtilmektedir [78]. FPGA üzerinde rastgele sayı üreteci için pek çok çalışmada hücresel otomat yapısı kullanılmaktadır [7, 77, 78]. Bu bakımdan tez çalışmasında, rastgele sayı üretimi için doğrusal hücresel otomat yapısı kullanıldı. Hücresel otomat'ta pek çok kural mevcuttur. Rastgele sayı üreteci için hücresel otomat yapısındaki 90 ve 150 kuralları kullanıldı. Bu kurallar referans [79, 80]'de açıklanmıştır. Kural-90 ve 150 yapıları Denklem (5.1) ve (5.2)'de gösterilmektedir.

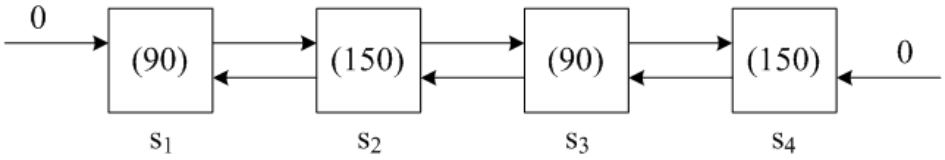
Kural-90: $s_i^+ = s_{i-1} \oplus s_{i+1}$ (5.1)

Kural-150: $s_i^+ = s_{i-1} \oplus s_i \oplus s_{i+1}$ (5.2)



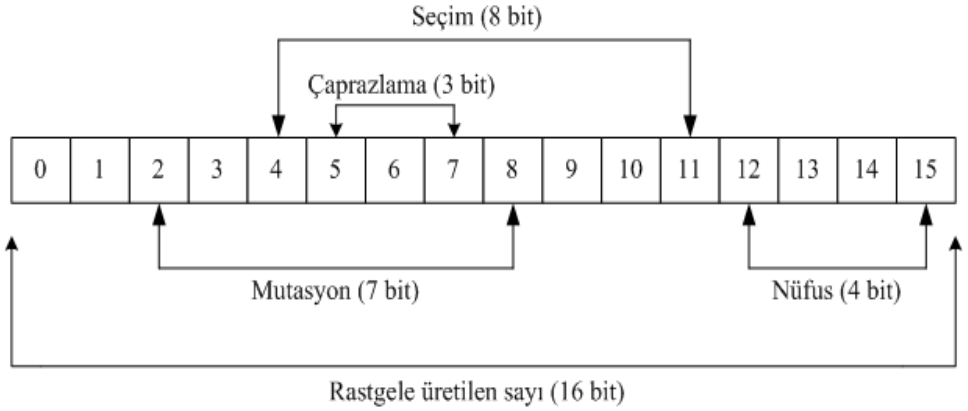
Şekil 5.10. Kural-90 (a) ve kural-150 (b) [77]

Şekil 5.10 kural-90 ve kural-150'nin mantıksal kapılarla gerçekleştirilmesini göstermektedir. Burada; s_i doğrusal dizideki i hücresinin mevcut durumu, s_i^+ s_i için sonraki durum ve $\oplus$ özel veya operatörüdür. Kural-150, önceki ve sonraki durumla beraber mevcut durumu da kullanırken, kural-90 yalnızca önceki ve sonraki durumu kullanarak güncelleme yapmaktadır. Şekil 5.11'de kural-90 ve 150 yapılarını kullanan hibrit bir hücresel otomat gösterilmektedir.



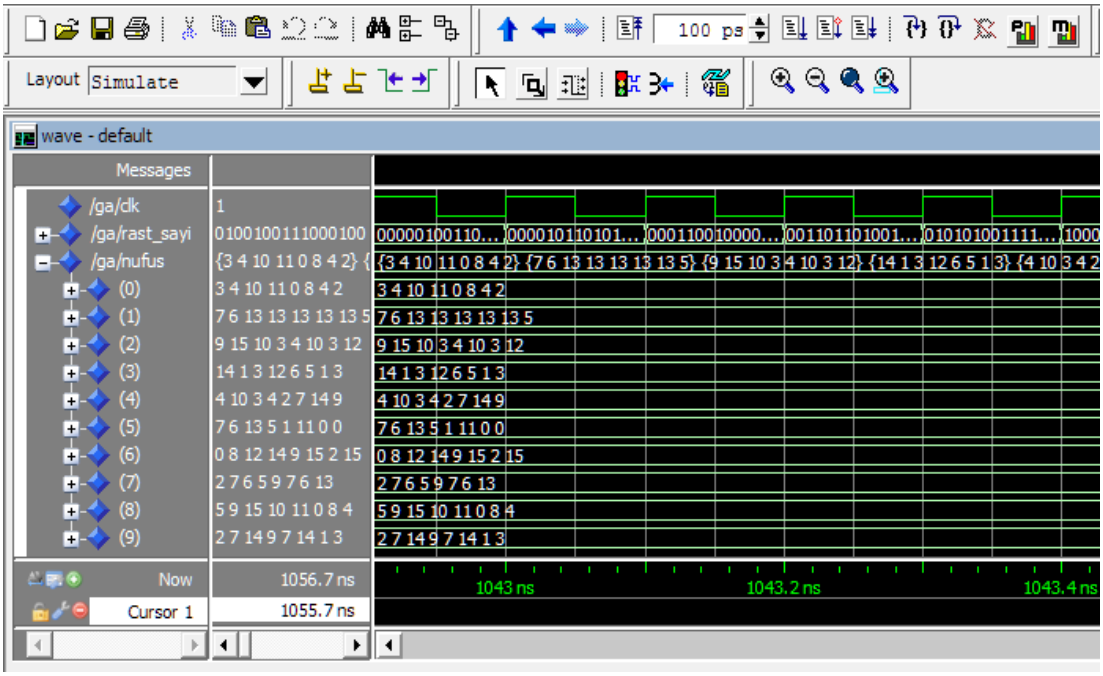
Şekil 5.11. Hücresel otomat ile rastgele sayı üretim örneği [78]

Tezde, rastgele sayı üretimi için 16 bitlik ikili sayılar kullanıldı. Bu 16 bitlik rastgele sayı dizisi içerisindeki farklı bitler GA'nın farklı operatörleri için tahsis edildi. Şekil 5.12, rastgele 16 bitlik bir sayının operatörlere göre dağılımını göstermektedir. Örneğin, üretilen 16 bitin 2-8 arası bitleri mutasyon operatörünün ihtiyaç duyduğu rastgele sayılar için, 5-7 arası bitleri çaprazlama operatörünün ihtiyaç duyduğu rastgele sayılar için kullanılmaktadır.



Şekil 5.12. 16 bitlik rastgele sayının GA operatörlerine göre dağılımı

Şekil 5.13, Modelsim benzetim ekranında, rastgele sayı üreticinin ürettiği 16 bitlik sayıları ve bu rastgele sayıların başlangıç nüfusu için kullanılan 4 bitlik kısımlarını göstermektedir. Benzetim için örnek olarak, nüfus büyüklüğü 10, her bir kromozomun uzunluğu 8 olarak alındı ve nüfus için toplam 80 adet rastgele sayı kullanıldı.

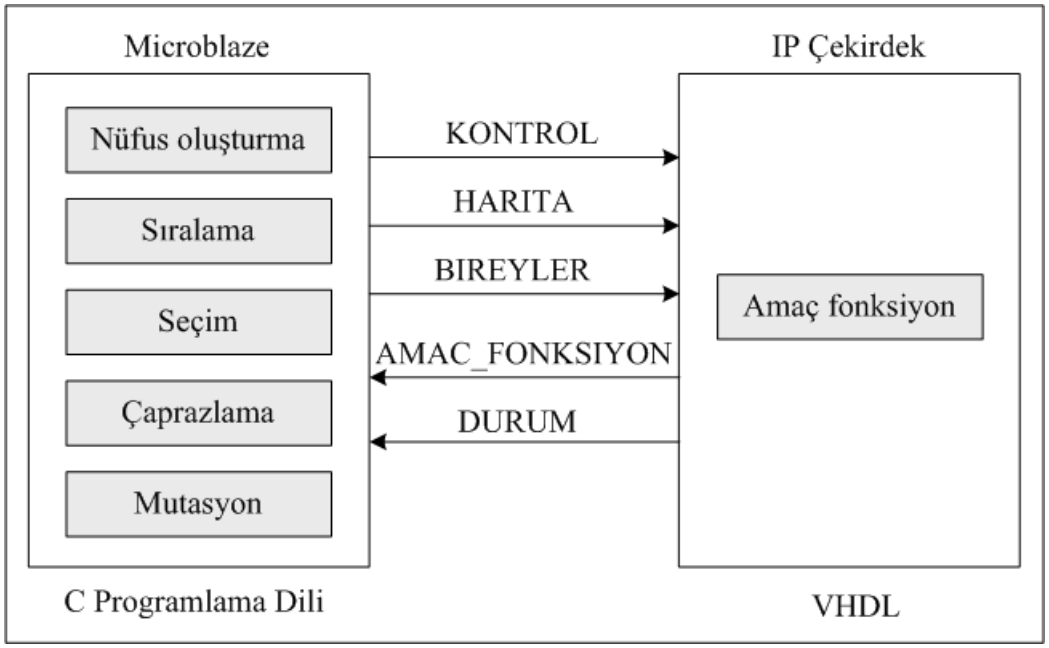


Şekil 5.13. Rastgele başlangıç nüfusu üretiminin Modelsim ekranında görünümü

5.5. Genetik Algoritmanın Hibrit Yapıda Gerçekleştirilmesi

Tez çalışmasında öncelikle, GA'nın tamamı C programlama dili kodlarıyla yazılarak, FPGA'de Microblaze işlemci üzerinde çalıştırıldı ve GA'nın çalışma süresi belirlendi. Toplam çalışma süresine bakıldığında, FPGA ile gerçekleştirilen yol bulma işleminin normal bir bilgisayarda çalıştırılan aynı GA'dan daha yavaş olduğu tespit edildi. GA'nın her bir operatörü için çalışma süreleri ayrı ayrı incelendiğinde, amaç fonksiyonun hesaplama zamanının diğer operatörlerden çok daha fazla sürdüğü görüldü. GA'da en fazla hesaplama zamanı gerektiren bölümün, genellikle amaç fonksiyon hesaplaması olduğu [51, 81, 82] belirtilmektedir.

Amaç fonksiyonu hızlandırmak için, ilgili operatörün donanımsal olarak gerçekleştirilip hızlandırılmasına karar verildi. Böylece, GA'nın diğer operatörleri Microblaze yazılımsal işlemci üzerinde çalışırken, sadece amaç fonksiyon için IP donanımsal çekirdek gerçekleştirildi. IP donanımsal çekirdek için VHDL, geri kalan diğer operatörler için de C programlama dili kullanıldı. Şekil 5.14, tasarlanan hibrit yol bulma sisteminin genel mimarisini göstermektedir.



Şekil 5.14. Yazılım-donanım hibrit yol bulma sisteminin genel yapısı

Sistem yazılımsal ve donanımsal çekirdek olmak üzere iki ana bölüm halinde oluşturuldu. IP donanımsal çekirdeğini, Microblaze yazılımsal işlemciye entegre etmek için PLB veri yolu kullanıldı. Sistemde, Microblaze işlemci ve IP donanımsal çekirdek arasında veri iletişimini sağlamak için kontrol ve durum sinyalleri kullanıldı. Kontrol sinyalinin mümkün olan değerleri "git" ve "bekle"dir. Kontrol sinyali "bekle" olduğunda, Microblaze başlangıç, hedef ve engel koordinatlarının bulunduğu ortamın haritasını ve nüfusun bireylerini IP donanımsal çekirdeğe göndermektedir. Kontrol sinyali "git" olduğunda, IP donanımsal çekirdek bireylerin amaç fonksiyon değerlerini hesaplamaya başlamaktadır. Durum sinyalinin mümkün olan değerleri ise "hazır" ve "meşgul"dür. IP çekirdek amaç fonksiyonu hesaplarken, durum sinyalinin değeri "meşgul", hesaplama bittikten sonraki değeri ise "hazır"dır. Durum sinyali "hazır" değerini aldığında, IP çekirdek, amaç fonksiyon değerlerini Microblaze'e göndermektedir. Tablo 5.2'de, Microblaze ve tasarlanan hibrit sistemin bir jenerasyon için çalışma süreleri ve hız artışı gösterilmektedir.

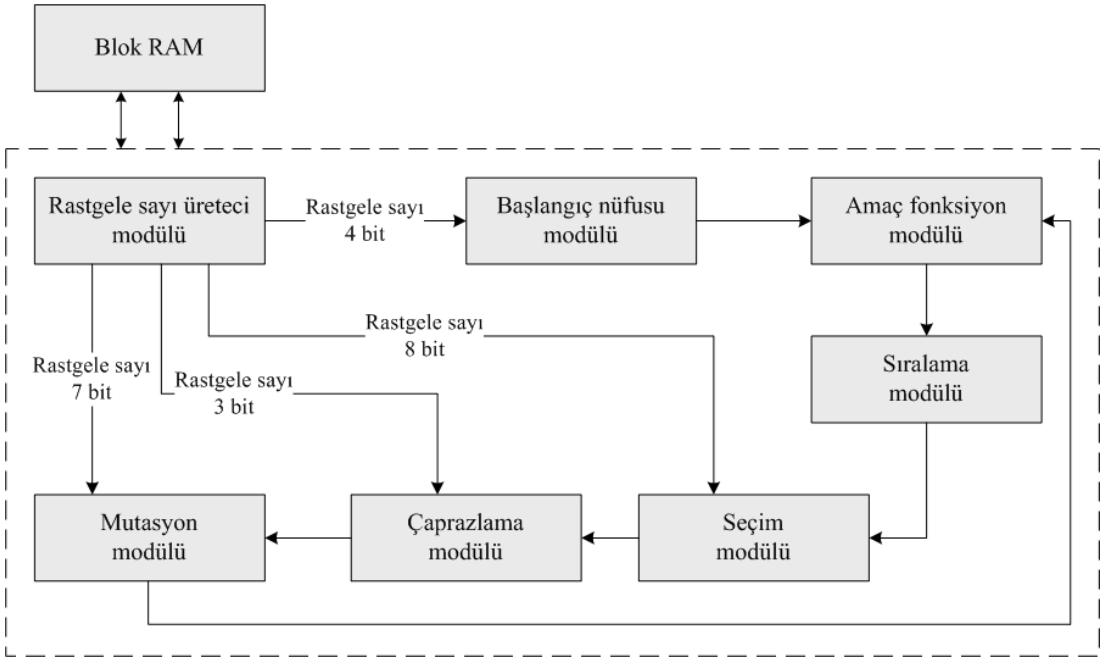
Tablo 5.2. Yazılımsal ve hibrit sistemin çalışma süreleri karşılaştırması

	Microblaze	Hibrit tasarım	İyileşme
Amaç fonksiyon	840 µs	8,8 µs	98,95 kat
Tüm GA	0,16 s	0,1 s	% 37,5

Tablodaki verilere göre, donanımsal IP çekirdek olarak tasarlanan amaç fonksiyon modülü, Microblaze sanal işlemcide çalışan yazılımsal amaç fonksiyonundan 98,95 kat daha hızlı çalışmaktadır. Sadece amaç fonksiyonun donanımla gerçekleştirilmesi GA'nın tamamının performansında %37,5'lik bir iyileşme sağlamıştır. Amaç fonksiyonda elde edilen bu iyileşme, diğer operatörlerin de donanım olarak gerçekleştirilmesiyle daha büyük katkı sağlanacağı konusunda cesaret vermiştir.

5.6. Genetik Algoritmanın IP Çekirdek Olarak Gerçekleştirilmesi

Amaç fonksiyonun donanımsal olarak gerçekleştirilmesiyle elde edilen başarı sonrasında, GA'nın tamamı özgün bir IP çekirdeği olarak tasarlandı ve çalıştırıldı. FPGA haricindeki dış ortamla bağlantı Microblaze işlemci üzerinden gerçekleştirildi. Harita bilgisi, kameranın bağlı olduğu bilgisayardan FPGA'deki GA IP çekirdeğine gönderilirken Microblaze işlemcisi üzerinden aktarılmakta, IP çekirdeğinde hesaplanan amaç fonksiyon değeri ve yol koordinatları yine Microblaze aracılığıyla dış ortama veya bilgisayara gönderilmektedir. Kamerayı ve gezgin robotu doğrudan FPGA'e bağlamak en mantıklı çözüm gibi görünmesine rağmen, FPGA'in hem GA'yı donanımsal olarak gerçekleştirmeye hem de bu bağlantıları sağlamaya kapasitesi yetmemektedir. Bu nedenle bağlantıları sağlamak amacıyla bir bilgisayar kullanılmıştır.



Şekil 5.15. GA IP çekirdeğinin genel yapısı

Çalışmada tasarlanan GA IP çekirdeğinin genel yapısı şekil 5.15'de gösterilmektedir. Rastgele sayı üreteci, GA'nın diğer operatörlerine paralel olarak çalışmaktadır. GA'nın diğer operatörlerinin çalışmasından bağımsız olarak, her saat darbesinin yükselen kenarında bir rastgele sayı üretmektedir. GA'nın herhangi bir operatörü, ihtiyaç duyduğu anda, o an üretilmiş olan rastgele sayıyı alıp kullanabilmektedir. Sistemde iteratif sıralı olarak işleyen modüller arası geçişlerde sonlu durum makineleri kullanıldı.

Tablo 5.3. GA IP çekirdeğinde kullanılan yazmaçlar ve adresleri

Adres	Yazmaç	Erişim türü	Genişlik (bit)
Base + 0x00	KONTROL	yazma	1
Base + 0x04	DURUM	okuma	1
Base + 0x08	BASLA_HEDEF	yazma	16
Base + 0x0C	YOL_KOORDINAT	okuma	32
Base + 0x10	HARITA_0	yazma	32
Base + 0x14	HARITA_1	yazma	32
Base + 0x18	HARITA_2	yazma	32
Base + 0x1C	HARITA_3	yazma	32
Base + 0x20	HARITA_4	yazma	32
Base + 0x24	HARITA_5	yazma	32
Base + 0x28	HARITA_6	yazma	32
Base + 0x2C	HARITA_7	yazma	32
Base + 0x30	HARITA_8	yazma	32
Base + 0x34	HARITA_9	yazma	32
Base + 0x38	HARITA_10	yazma	32
Base + 0x3C	HARITA_11	yazma	32
Base + 0x40	HARITA_12	yazma	32
Base + 0x44	HARITA_13	yazma	32
Base + 0x48	HARITA_14	yazma	32
Base + 0x4C	HARITA_15	yazma	32
Base + 0x50	LED_AC	yazma	8
Base + 0x54	LED_KAPAT	yazma	8

Tablo 5.3, tasarlanan özgün GA IP çekirdeğinde kullanılan yazmaçları ve adres bilgilerini göstermektedir. Yazmaçlar, Microblaze işlemci ile IP çekirdek arasındaki veri aktarımı için kullanılmaktadır. Toplam 22 adet yazmaç bulunmaktadır. Yazmaçların adresleri, ilk yazmaç olan Base+0x00 adresinden başlamaktadır. IP çekirdeğinde yer alan yazmaçların kullanım amaçları aşağıda belirtilmektedir;

- KONTROL yazmacı: Sistem çalışırken, IP çekirdeğindeki GA bölümü, görevine başlamadan hazır durumda beklemektedir. Görevine başlayabilmesi için, öncelikle Microblaze'den harita bilgisinin alınması gerekmektedir. Microblaze dış ortamdan harita bilgisini aldıktan sonra, bu yazmacı aktif ederek IP çekirdeğine başla komutu göndermektedir (Şekil 5.16). Kontrol yazmacının değeri "0" olduğunda IP çekirdek için "bekle", "1" olduğunda "başla" komutu geçerli olmaktadır.

- DURUM yazmacı: GA IP çekirdeğinin çalışması sonucunda elde edilen en uygun yol bilgisi ve amaç fonksiyon değerinin Microblaze'e ve oradan da dış ortama aktarılabilmesi için, Microblaze, IP çekirdek bölümündeki işlemlerin tamamlanmasını beklemektedir. IP çekirdek hesaplama işlemlerini bitirdiğinde, Microblaze'e bu yazmacı kullanarak durum bilgisi gönderir. IP çekirdek işlemlere devam ederken durum yazmacının değeri "0", işlemler bittiğinde "1" olmaktadır.

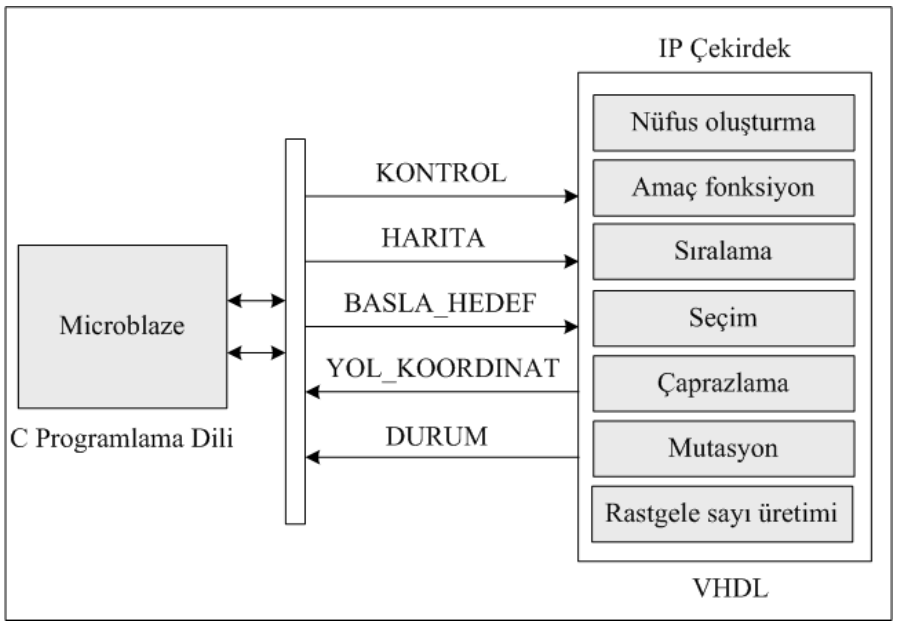
- BASLA_HEDEF yazmacı: Bu yazmaç, Microblaze'den IP çekirdeğe haritadaki başlangıç ve hedef koordinatlarının aktarılması için kullanılmaktadır.

- YOL_KOORDINAT yazmacı: IP çekirdekteki işlemler tamamlandıktan sonra, robot için en uygun yol koordinatları Microblaze'e bu yazmaç ile aktarılmaktadır.

- HARITA(0-15) yazmaçları: Çalışmada, gezgin robotun üzerinde hareket edeceği 16x16' lık bir çalışma alanı kullanıldı. Bu yüzden, haritanın her bir satırı için bir tane olmak üzere toplam 16 adet yazmaç kullanılmaktadır. Bu yazmaçlar vasıtasıyla, Microblaze'den IP çekirdeğe harita bilgisi yollanabilmektedir.

- LED_AC ve LED_KAPAT yazmaçları: IP çekirdekteki hesaplama işlemi süresince, FPGA üzerindeki LED'leri yakıp hesaplama bittiğinde de söndürmek

için kullanıldı. Bu yazmaçların programa herhangi bir etkisi olmayıp, sadece kontrol amaçlı olarak kullanılmışlardır.



Şekil 5.16. Donanımsal yol bulma sisteminin genel yapısı

GA.c

```
#include "xparameters.h"
#include "mb_interface.h"
#include "stdio.h"
#include "xutil.h"
#include "stdlib.h"

#define KONTROL           XPAR_GA_0_BASEADDR + 0
#define DURUM             XPAR_GA_0_BASEADDR + 4
#define BASLA_HEDEF       XPAR_GA_0_BASEADDR + 8
#define AMAC_FONKSIYON    XPAR_GA_0_BASEADDR + 12
#define HARITA_0          XPAR_GA_0_BASEADDR + 16
#define LED_AC            XPAR_GA_0_BASEADDR + 80
#define LED_KAPAT         XPAR_GA_0_BASEADDR + 84
```

Şekil 5.17. Yazmaçların Microblaze'de adreslenmesi

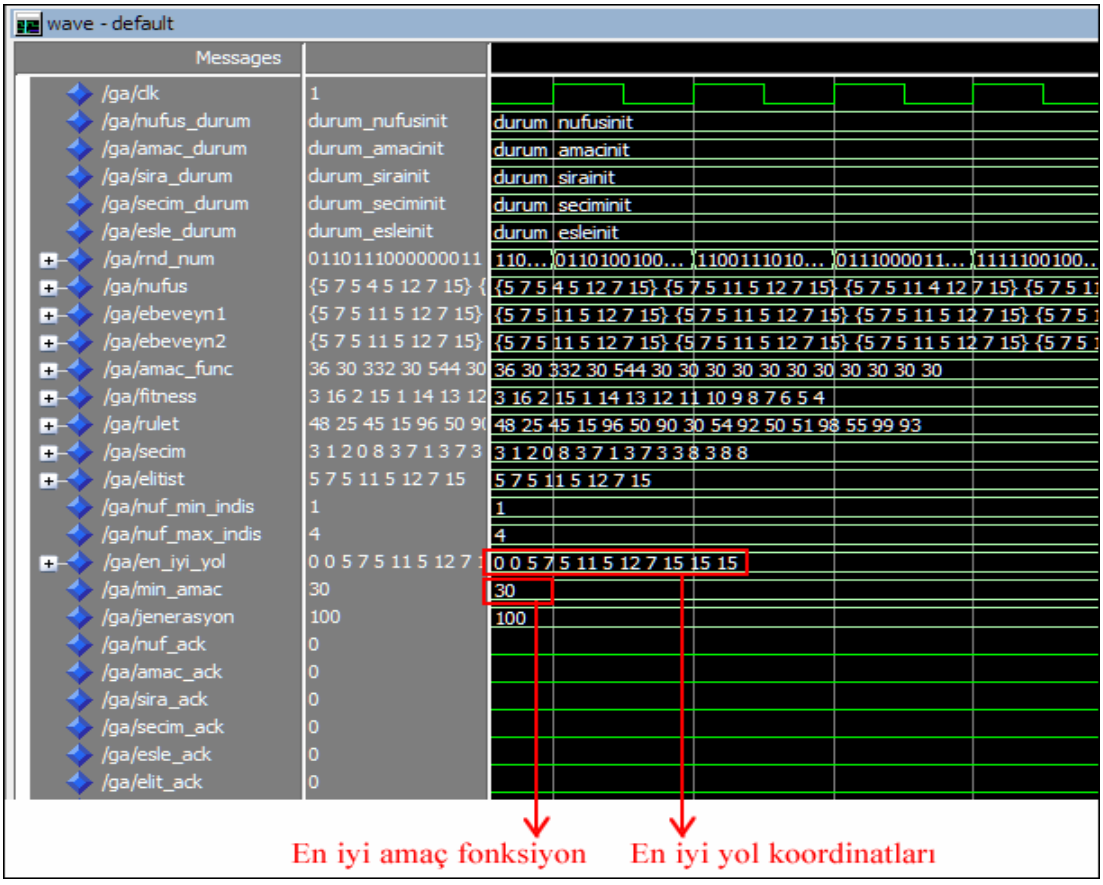
Şekil 5.17, IP çekirdekte kullanılan yazmaç adreslerinin Microblaze'de tanımlanmasını göstermektedir. Şekilde görüldüğü gibi IP çekirdeğin ilk yazmaç adresi XPAR_GA_0_BASEADDR+0 şeklinde ifade edilmektedir. Diğer yazmaçların adresleri de 4 artırılarak devam etmektedir.

Şekil 5.18, Microblaze ve IP çekirdek bağlantısındaki kontrol ve durum yazmaçlarına değer yüklemek için kullanılan örnek VHDL kodlarını göstermektedir. Şekildeki "when" ifadesinin yanında toplam 22 adet sayı değeri bulunmaktadır. Bu 22 adet değer, IP çekirdekte kullanılan 22 adet yazmacın içeriğini ifade etmektedir. KONTROL yazmacının içeriğini birinci sayı, DURUM yazmacının değerini ikinci sayı göstermektedir. Microblaze'de hangi yazmaca değer yüklenecek ise, o yazmacın adresi belirtilir ve değer ilgili yazmaca şekilde görüldüğü gibi yüklenir.

```
when "10000000000000000000000" => -- KONTROL
 for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
  if ( Bus2IP_BE(byte_index) = '1' ) then
   KONTROL(byte_index*8 to byte_index*8+7)<=Bus2IP_Data(byte_index*8 to byte_index*8+7);
  end if;
 end loop;

when "01000000000000000000000" => -- DURUM
 for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
  if ( Bus2IP_BE(byte_index) = '1' ) then
   DURUM(byte_index*8 to byte_index*8+7)<=Bus2IP_Data(byte_index*8 to byte_index*8+7);
  end if;
 end loop;
```

Şekil 5.18. IP çekirdekteki kontrol ve durum yazmaçları



Şekil 5.19. Modelsim ekranında GA parametrelerinin görüntülenmesi

Şekil 5.19, Modelsim programında, IP çekirdekte kullanılan GA parametrelerinin değerlerini göstermektedir. Şekilde, 100 jenerasyon sonunda tüm operatörlerin aldığı değerler ve en iyi amaç fonksiyon değeri ile bu amaç fonksiyon değerine sahip en iyi yol koordinatları gösterilmektedir. En iyi yol ve amaç fonksiyon değeri, aynı zamanda Microblaze'e ve oradan da bilgisayara gönderilmektedir.

Tablo 5.4, gerçekleştirilen GA IP çekirdeğinin, Virtex-5 FPGA üzerinde kullandığı donanım kaynaklarının istatistiksel verilerini göstermektedir.

Tablo 5.4. FPGA' da mevcut ve kullanılan donanım kaynakları

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	7,997	69,120	11%
Number used as Flip Flops	7,996		
Number used as Latch-thrus	1		
Number of Slice LUTs	67,669	69,120	97%
Number used as logic	66,729	69,120	96%
Number using O6 output only	65,312		
Number using O5 output only	112		
Number using O5 and O6	1,305		
Number used as Memory	197	17,920	1%
Number used as Dual Port RAM	64		
Number using O5 and O6	64		
Number used as Shift Register	133		
Number using O6 output only	132		
Number using O5 output only	1		
Number used as exclusive route-thru	743		
Number of route-thrus	944	138,240	1%
Number using O6 output only	805		
Number using O5 output only	91		
Number using O5 and O6	48		
Number of occupied Slices	17,275	17,280	99%
Number of occupied SLICEMs	108	4,480	2%
Number of LUT Flip Flop pairs used	68,161		
Number with an unused Flip Flop	60,164	68,161	88%
Number with an unused LUT	492	68,161	1%
Number of fully used LUT-FF pairs	7,505	68,161	11%
Number of unique control sets	514		
Number of slice register sites lost to control set restrictions	963	69,120	1%
Number of bonded IOBs	200	640	31%
IOB Flip Flops	403		
Number of BlockRAM/FIFO	25	148	16%
Number using BlockRAM only	25		
Number of 36k BlockRAM used	25		
Total Memory used (KB)	900	5,328	16%
Number of BUFG/BUFGCTRLs	8	32	25%
Number used as BUFGs	8		
Number of IDELAYCTRLs	3	22	13%
Number of BSCANs	1	4	25%
Number of BUFIOs	8	80	10%
Number of DCM_ADVs	1	12	8%
Number of DSP48Es	5	64	7%
Number of PLL_ADVs	1	6	16%
Average Fanout of Non-Clock Nets	5.62		

Tablodaki verilere göre, FPGA'de yer alan LUT kaynaklarının %97'si kullanıldığı görülmektedir. Bu, tasarlanan sistemin FPGA içerisine ancak sığdığı anlamına gelmektedir. Kapasite sorunu nedeni ile, nüfus içerisinde yer alan birey sayısı benzetim çalışmalarında 40 veya 50 gibi alınırken, donanımsal gerçeklemede 16 ile sınırlandırıldı.

GA IP çekirdeği gerçeklendikten sonra, sistem çalıştırılıp performansı değerlendirilmeye tabi tutuldu ve donanımsal GA'nın bütün operatörlerine ait çalışma süreleri tek tek belirlendi. Karşılaştırma yapabilmek için; bir bilgisayar üzerinde çalışan ve C kodları ile yazılmış GA (EK-A), Microblaze üzerinde çalışan ve C kodları ile yazılmış yazılımsal GA ve Microblaze ile koordineli çalışan donanımsal IP çekirdek olarak yazılmış GA'ya (EK-B) ait çalışma süreleri belirlendi ve bu süreler Tablo 5.5'de verilmektedir. Tablodaki verilerin belirlenmesinde kullanılan nüfus büyüklüğü, her üç uygulamada da 16 olarak alındı. Bilgisayarda çalışan yazılımsal GA için kullanılan bilgisayarın özellikleri; Intel i5 işlemci, 4 GB RAM, 2.53 GHz'dir. Tasarlanan GA IP çekirdeği için FPGA'in çalışma frekansı 78 MHz'dir.

Tablo 5.5. GA operatörlerinin farklı platformlarda çalışma süreleri

	Yazılımsal GA (µs)	Microblaze (µs)	IP Çekirdeği (µs)	İyileşme (%)
Başlangıç nüfusu	2,33	18110	0,82	64,8
Amaç fonksiyon	30,72	336000	4,10	86,6
Sıralama	1,33	1860	0,05	96,2
Seçim	1,64	5320	0,92	43,9
Eşleştirme	0,65	1910	0,05	92,3
Çaprazlama	1,13	3040	0,10	91,1
Mutasyon	3,47	19640	0,20	94,2
100 Jenerasyon	$6{,}167 \times 10^3$	35.000×10^3	$0{,}594 \times 10^3$	90,3

Tablodaki verilere göre, en yavaş çalışan uygulama Microblaze üzerinde çalışan GA'dır ve diğerlerine göre oldukça yavaştır. Bu sonuç oldukça doğaldır, çünkü 125 MHz'lik FPGA üzerinde çalışan aynı kod 2.53 GHz'lik bilgisayardan daha yavaş çalışacaktır. VHDL ile yazılmış ve IP çekirdek olarak gerçekleştirilen GA, hem bilgisayar hem de Microblaze'de çalıştırılan yazılıma göre daha hızlı çalışmaktadır.

IP çekirdekte gerçekleştirilen GA operatörlerinin hız iyileştirmeleri, bilgisayarda çalıştırılan yazılımla karşılaştırıldığında %43,9 ile %96,2 arasındadır. 16 birey ve 100 jenerasyon kullanıldığında, en kısa yol bulma süresi IP çekirdekte yaklaşık 0,6 milisaniye iken, bu süre bilgisayarda 6 milisaniyenin üzerindedir.

Tablodaki veriler, kullanılan FPGA özelliklerine göre değişebilmektedir. Tez çalışmasında gerçekleştirilen tasarımın daha üst model bir FPGA'de gerçekleştirilebilmiş olması durumunda çözüm zamanının daha da düşeceği öngörülmektedir. Bunun sebebi, gelişmiş FPGA'lerde mevcut kapı sayısının çok daha fazla olmasıdır. Örneğin; çalışmada kullanılan Virtex-5 FPGA için mevcut LUT sayısı 69.120 iken, bu sayı en gelişmiş Virtex-7 FPGA için 1.221.600'dür.

5.7. Literatürdeki Benzer FPGA Uygulamaları ile Tez Çalışması Sonuçlarının Karşılaştırılması

Allaire ve diğ. [2] tarafından gerçekleştirilen çalışmada, GA tabanlı yol bulma algoritmasının FPGA üzerinde gerçekleştirilmesi amaçlanmıştır. Uygulama alanı olarak insansız hava araçlarının gerçek zamanlı yol planlaması seçilmiştir. Yazarların çalışma ile elde ettikleri sonuçlar Tablo 5.6'da verilmektedir. Çalışmada kullandıkları bilgisayarın işlemcisi Pentium 4, 2.8 GHz'dir. Kullandıkları FPGA ise 100MHz'lik Virtex-II Pro (2vp30ff896)'tür. Çalışmada, GA'nın tamamı FPGA üzerinde gerçekleştirilememiş, amaç fonksiyon ve mutasyon benzetim seviyesinde kalmıştır. Bunun sebebi olarak da, kullandıkları FPGA'in donanım kaynaklarının yetersiz kalmasını göstermişlerdir.

Tablo 5.6. Literatür çalışmasındaki zaman sonuçları [2]

GA	PC tabanlı GA	FPGA tabanlı değiştirilmiş GA	Hız artışı
Seçim ve çaprazlama	94 ms	8,85 μs	10.000x
Mutasyon	2 ms	18 μs	111x
Nüfus güncelleme	30 ms	600 ns	50.000x

Tez çalışmasında elde edilen ve Tablo 5.5'te verilen sonuçlarla, Allaire ve diğ.'lerin elde ettikleri ve Tablo 5.6'da verilen sonuçlar karşılaştırıldığında;

- Karşılaştırılan çalışmada kullanılan bilgisayar modeli tez çalışmasında kullanılan bilgisayar modelinden daha düşük, fakat işlemci hızı daha yüksektir.

- Karşılaştırılan çalışmada kullanılan FPGA modeli tez çalışmasında kullanılan FPGA modelinden daha düşük, çalışma frekansı 100 MHz'dir. Tez çalışmasında tasarlanan IP çekirdek sistemi, FPGA kaynaklarının tamamına yakınını kullandığı için çalışma frekansı 78 MHz'in üzerine çıkamamıştır.

- Hız artışları, kullanılan bilgisayar hızına bağlı olarak hesaplandığı için, her iki çalışmanın sonuçlarını, hız artışları sütunundaki değerlere göre yapmak sağlıklı olmayacaktır. Bu nedenle, doğrudan FPGA ile elde edilen süreleri karşılaştırmak daha gerçekçi olacaktır.

- Karşılaştırılan çalışmada, seçim ve çaprazlama 8,85 μs sürerken, tez çalışmasında seçim, eşleştirme ve çaprazlama toplamı 1,07 μs sürmüştür.

- Karşılaştırılan çalışmada, mutasyon 18 μs sürerken, tez çalışmasında 0,205 μs sürmüştür. Ayrıca, karşılaştırılan çalışmada mutasyon benzetimle gerçekleştirilmiş olup, FPGA ile gerçekleştirilmesi halinde süre daha da uzayacaktır.

- Karşılaştırılan çalışmada, nüfus güncellemesi 600 ns sürerken, tez çalışmasında çocuklar için yeni bir nüfus kullanılmayıp, ebeveyn nüfus üzerinde güncellemeler yapılmıştır. Bunun için geçen süreler, çaprazlama ve mutasyon sürelerine dahildir.

Karşılaştırmalar sonucunda, tez çalışması ile, yol bulma problemi için GA'nın tamamen FPGA üzerinde gerçekleştirildiği ve karşılaştırılan çalışmadan daha hızlı sonuçlar alındığı görülmüştür.

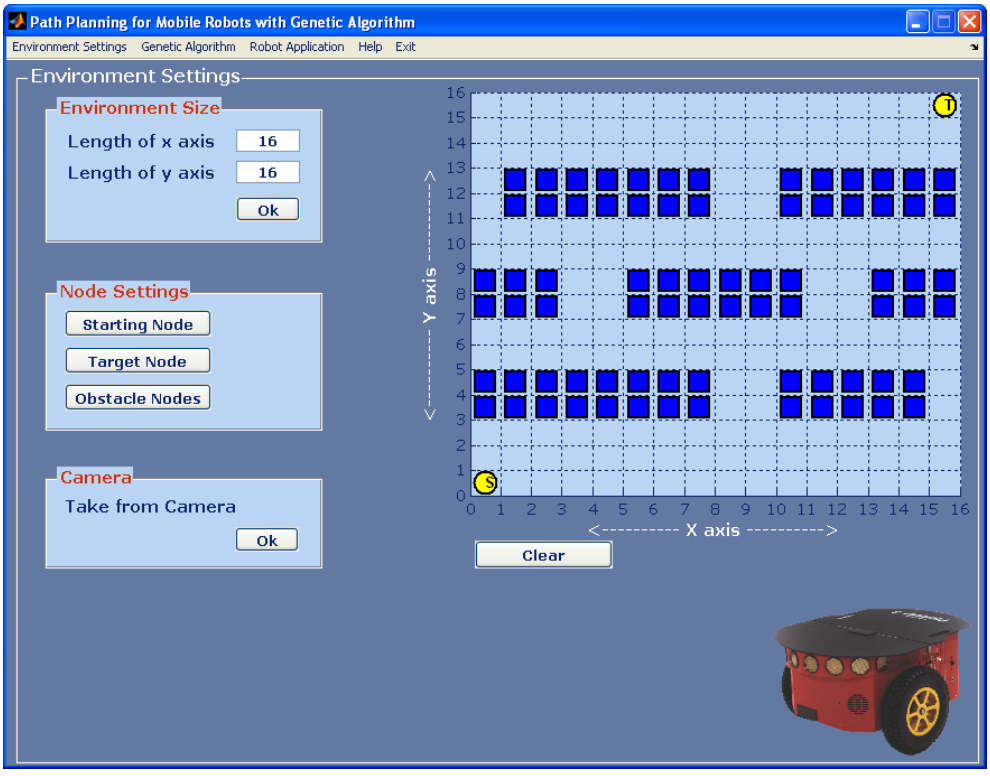
GA ile yol bulma probleminin FPGA üzerinde gerçekleştirilmesine yönelik literatürde çok fazla çalışma bulunmamaktadır. Konu ile ilgili olarak; Kok ve diğ. [83], Hachour [84], Tsai ve diğ. [85], yapmış oldukları çalışmalarda, GA ile yol bulma problemlerini FPGA üzerinde gerçekleştirmişlerdir. Ancak, çalışmaların sonucunda herhangi bir somut sayısal netice vermedikleri için tez çalışmasındaki sonuçlarla karşılaştırılamamıştır.

5.8. Robot ve Genetik Algoritmalar için Kullanıcı Arayüzü Tasarımı

Tez çalışmasında, sistemin tamamını kolay bir şekilde kontrol edebilmek amacıyla Matlab-GUI tabanlı bir kullanıcı arayüzü tasarlandı. Arayüz üç ana menüden oluşmaktadır. Birinci menüde, kameradan alınan görüntü yardımıyla veya kameradan bağımsız olarak kullanıcı tarafından oluşturulan çalışma ortamı ve bu çalışma ortamının içeriğinin tanımlandığı ekran yer almaktadır. İkinci menüde, GA'nın parametreleri tanımlanmakta ve bilgisayar üzerinde GA ile yol bulma işlemi gerçekleştirilebilmektedir. Üçüncü menüde ise bilgisayarla ya da FPGA ile belirlenmiş yol koordinatlarına göre robotun hareketinin sağlandığı ekran yer almaktadır. Bu ekranda, yol bilgisi olmadan, kullanıcının vereceği anlık komutlarla da robotun hareketi gerçekleştirilebilmektedir.

5.8.1. Arayüz yardımıyla çalışma ortamı ayarlarının yapılması

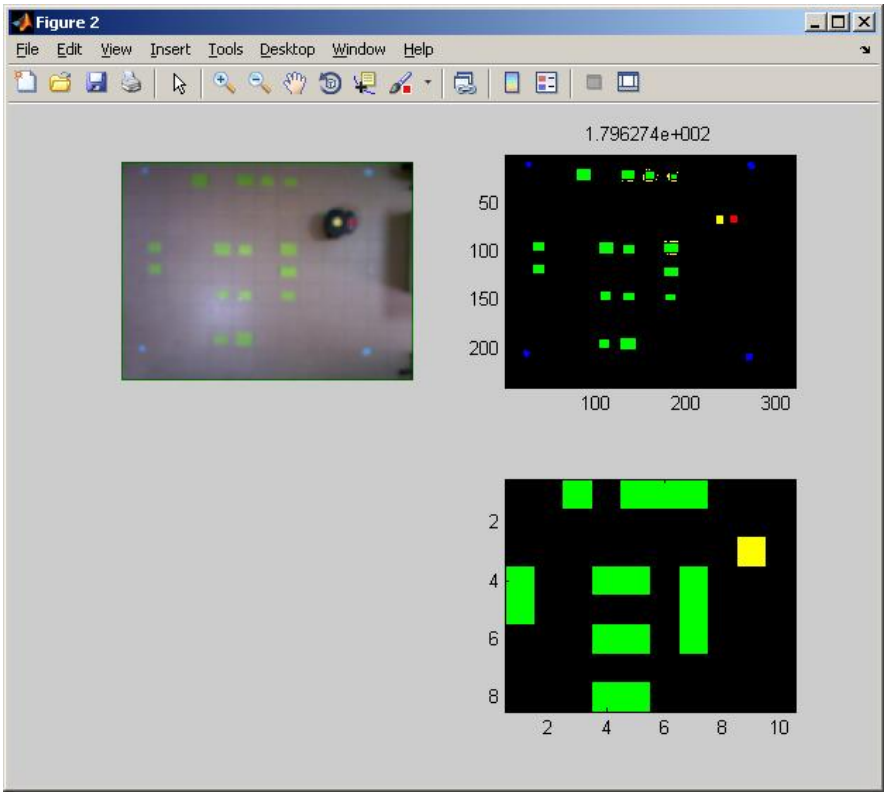
"Environment Settings (Çevresel Ayarlar)" menüsünü ile, çalışma alanı boyutu, başlangıç, hedef ve engel noktaları kullanıcı tarafından belirlenebilmektedir. Kullanıcı bu noktaları, ekranda bulunan butonlar yardımıyla, çalışma alanındaki hücreler üzerinde fare ile tıklayarak belirleyebilmektedir.



Şekil 5.20. Çalışma ortamının belirlendiği Çevresel Ayarlar ekranı

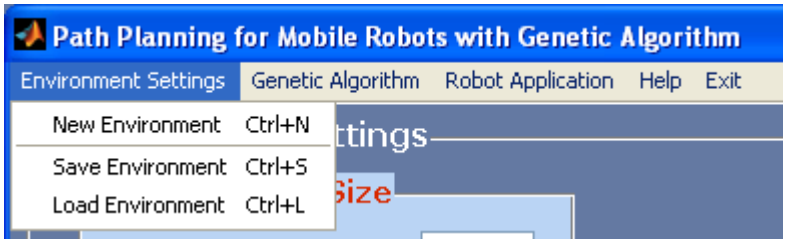
Şekil 5.20'de, 16x16'lık bir çalışma alanı, başlangıç ve hedef noktaları ile engellerin belirlendiği "Çevresel Ayarlar" ekranı görüntülenmektedir. Şekilde görüldüğü gibi, başlangıç ve hedef noktaları sarı daire, engeller ise mavi kareler ile ifade edilmektedir.

Çalışma ortamı laboratuvar tavanına yerleştirilen bir kamera yardımı ile otomatik olarak da elde edilebilmektedir. Bunun için, yine ekranda bulunan "Take from Camera (Kameradan Al)" butonu kullanılmaktadır. Kamera ile alınan ortamın gerçek görüntüsü, Şekil 5.21'de gösterildiği gibi, görüntü işleme aşamalarından geçirilerek haritaya dönüştürülmektedir.



Şekil 5.21. Kameradan alınan çalışma alanı görüntüsü ve haritaya dönüştürülmesi

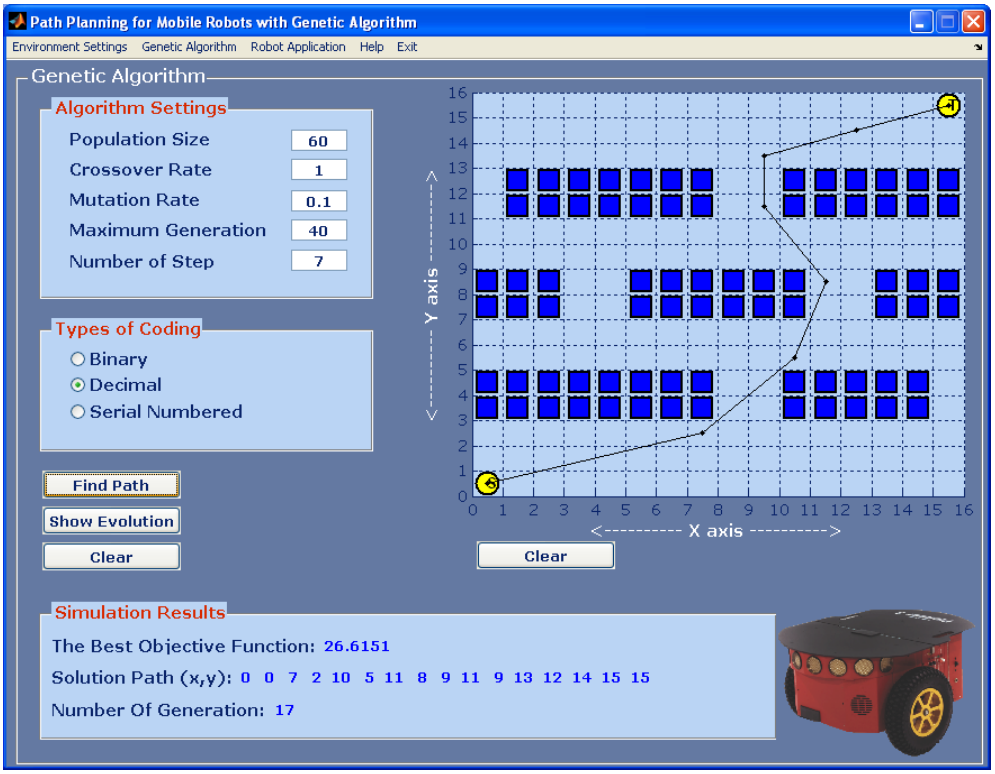
Şekil 5.22'de gösterildiği gibi, arayüzde bulunan "Çevresel Ayarlar" menüsündeki "Save Environment (Çevreyi Kaydet)" ve "Load Enviroment (Çevreyi Yükle)" komutları ile kullanılmakta olan çalışma ortamı kaydedilmekte ve daha sonra istenildiğinde tekrar geri yüklenebilmektedir. Böylece, kullanıcının her defasında, çalışma ortamını yeniden düzenleme gerekliliği ortadan kalkmaktadır.



Şekil 5.22. Yeni bir çalışma ortamı belirleme, kaydetme ve yükleme

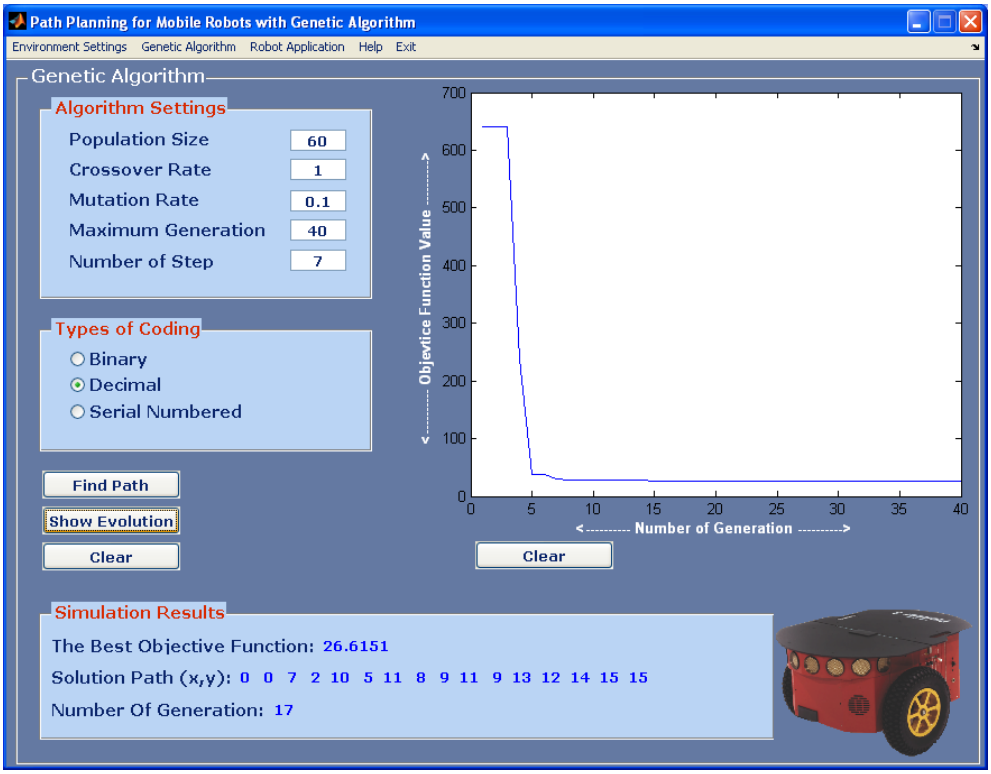
5.8.2. Arayüz yardımıyla GA parametrelerinin ayarlanması ve yol bulma

Çalışma ortamı kullanıcı tarafından sanal olarak ya da kameradan alınan gerçek görüntü ile belirlendikten sonra, "Genetic Algorithm (Genetik Algoritma)" menüsü kullanılarak genetik algoritma parametreleri ayarlanır. Parametreler içerisinde; nüfus büyüklüğü, çaprazlama oranı, mutasyon oranı, maksimum jenerasyon sayısı ve adım sayısı yer almaktadır. Adım sayısı, başlangıç noktası ile hedef noktası arasında üzerinden geçilen noktaların sayısıdır. Her nokta *x* ve *y* koordinatlarını içermektedir. Bu nedenle, GA'da her bir bireyin uzunluğu adım sayısının iki katıdır. Ayrıca, "Types of Coding (Kodlama Tipi)" ekranı üzerindeki bölümden; ikili, desimal ve seri numaralı kodlama metotlarından hangisinin kullanılacağı seçilmektedir.



Şekil 5.23. Arayüz üzerinde GA ile en uygun yolun bulunması

"Find Path (Yol Bul)" butonu tıklandığında, kullanıcı tarafından belirlenen parametreler kullanılarak, GA en uygun yolu bulur ve bulunan yol çalışma ortamı üzerinde çizdirilir. Ayrıca, bulunan yolun koordinatları, toplam mesafesi (amaç fonksiyon değeri) ve yolun kaçıncı jenerasyonda bulunduğu, ekranın "Simulation Results (Benzetim Sonuçları)" bölümünde gösterilmektedir (Şekil 5.23).

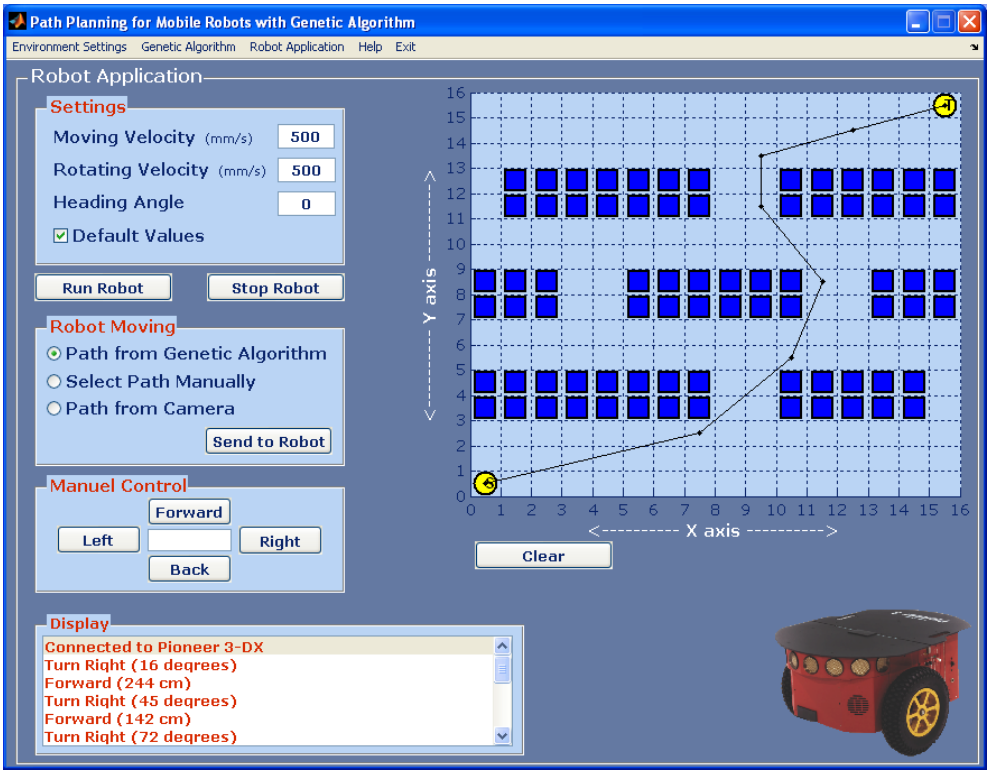


Şekil 5.24. Amaç fonksiyon-jenerasyon sayısı grafiği

Şekil 5.24, Genetik Algoritma ekranında bulunan "Show Evolution (Gelişimi Göster)" butonu tıklandığında, her bir jenerasyonda elde edilen en uygun yolun amaç fonksiyon değerlerini grafik olarak göstermektedir.

5.8.3. Robot ile ilgili uygulamalar

"Robot Application (Robot Uygulaması)" menüsünde, robotun temel hareket parametreleri ayarlanmakta ve belirlenen yol boyunca robotun hareketi sağlanmaktadır. Temel hareket parametreleri; ilerleme hızı, dönme hızı ve doğrultu açısını kapsamaktadır. Robot hareketine başlamadan önce bu değerler ayarlanabilir veya varsayılan değerler kullanılabilir (Şekil 5.25).

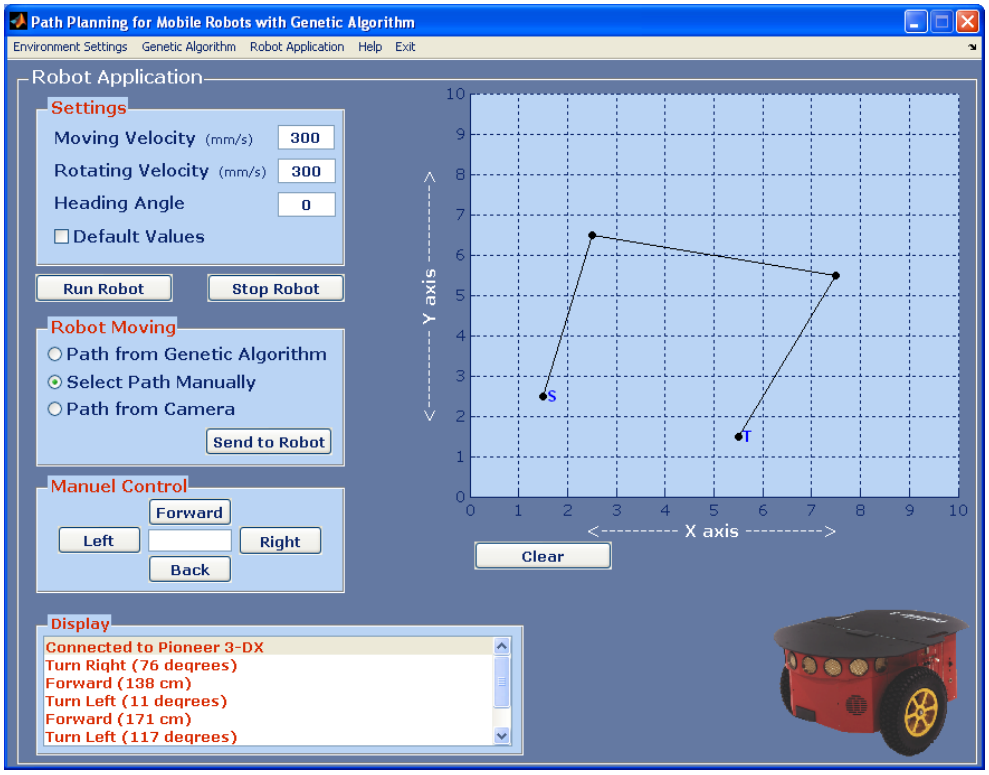


Şekil 5.25. Arayüz üzerinde GA ile bulunan yola göre robotun hareketi

Robotun hareketi üç farklı kaynaktan sağlanabilmektedir. Birincisi seçenekte, robot daha önceden tanımlanmış olan çalışma ortamında ve GA ile bulunmuş olan en uygun yol boyunca ilerleyebilmektedir. Şekil 5.25'de görüldüğü gibi, yol koordinatları robota gönderildikten sonra, ekranın en altında bulunan "Display (Gösterge)" bölümünde, robotun ne kadar ilerlediği veya hangi yöne kaç derece açıyla döndüğü bilgileri kullanıcıya gösterilmektedir. İkinci seçenekte robot, kullanıcının o an ekran üzerinde manuel olarak belirleyeceği bir yolda ilerleyebilmektedir. Üçüncü seçenekte ise, "Kameradan Al" seçeneği kullanılarak, kameradan alınan çalışma ortamı bilgisi ve FPGA ile bulunan yol koordinatlarına göre robot ilerleyebilmektedir.

Şekil 5.26, kullanıcı tarafından manuel olarak belirlenen yolda robotun ilerlemesini göstermektedir. Buradaki fark, yolun GA ile değil kullanıcı tarafından belirlenmiş olmasıdır. Bunun dışında kullanıcı, robotun ilerleyeceği yolu çalışma ortamı üzerinde baştan sona belirlemek yerine, yolun her bir parçasını oluşturan hareket komutlarını robota tek tek göndererek robotun hareketini sağlayabilmektedir. Örneğin, ekranın "Manuel Kontrol" bölümünde, yön butonlarının ortasındaki veri giriş kutucuğuna 50

değeri girilerek ve ileri/geri butonuna basılarak robotun 50cm ilerlemesi veya sağ/sol butonuna basılarak 50° sağa/sola dönmesi sağlanabilmektedir.



Şekil 5.26. Manuel olarak belirlenen yola göre robotun ilerlemesi

6. SONUÇ VE ÖNERİLER

Gezgin robotlar günümüzde askeri alanlardan günlük yaşama kadar pek çok alanda kullanılmaktadır. Belirli görevleri yerine getirmesi beklenen robotlar, tamamen insanlar tarafından kontrol edilebileceği gibi, kendi başlarına otonom olarak da hareket edebilmektedirler. Bir robotun otonom olabilmesi için, kendi kendine karar verebilme ve verdiği kararı uygulayabilme yeteneğine sahip olması gerekir. Otonom bir gezgin robotun özellikleri; yol planlaması yapabilmek ve planlanan yol boyunca hareket edebilmek, engelleri bulma, tanıma ve işleme kabiliyetine sahip olmak, algılayıcılardan gelen verileri okuyabilmek ve değerlendirebilmek, görüntü işleme, keşif ve gözetleme yapabilmek olarak sıralanabilir.

Yol bulma, otonom gezgin robot uygulamalarında en önemli problemlerden biridir. Geçmişte, yol bulma problemlerine çözüm üretmek için pek çok algoritma ve yöntem geliştirilmiştir. Bu tez çalışmasında, otonom bir gezgin robot için yol bulma problemi GA kullanılarak çözüldü. Her biri bir aday yolu oluşturan kromozomların kodlama metodunu belirlemek amacıyla; ikili, tamsayı ve seri numaralı kodlama yöntemleri için örnek uygulamalar yapıldı. Bu uygulamalar ile kodlama metotlarının birbirlerine göre avantaj ve dezavantajları belirlenmeye çalışıldı. Eşit çalışma ortamı ve GA parametreleri ile yapılan çalışmalar sonucunda; tamsayı kodlama yönteminin en kısa yolu bulmak için daha uygun olduğu ve seri numaraları kodlama yönteminin de diğer kodlama yöntemlerinden daha kısa sürede uygun bir çözüm bulabildiği sonucuna varıldı.

GA'da mutasyon operatörünün en yaygın kullanımı, mutasyona uğrayacak genin değerini rastgele üretmektir. Fakat engelli bir ortamda, bu durum optimizasyonu yavaşlatır ve jenerasyon sayısının artmasına sebep olur. Mutasyon operatörünü geliştirmeye yönelik literatürde farklı çalışmalar yapılmıştır. Bu tez çalışmasında, literatürde yer alan diğer mutasyon operatörlerinden daha etkili yeni bir mutasyon operatörü önerildi ve geçmiş çalışmalarla eşit koşullar altında karşılaştırıldı.

Önerdiğimiz metodun ortalama amaç fonksiyon değerleri ve ortalama jenerasyon sayılarının, diğer metotlarda bulunan sonuçlardan daha başarılı olduğu görüldü.

Tez çalışması ile geliştirilen yol bulma sisteminin işlem basamakları şunlardır: Birinci adımda, laboratuvar tavanına yerleştirilmiş olan bir kamera ile çalışma ortamının görüntüsü elde edilmektedir. İkinci adımda, bu görüntü bir bilgisayarda görüntü işleme yöntemleri ile işlenerek, matris şeklinde bir harita bilgisine dönüştürülmektedir. Üçüncü adımda, oluşturulan harita bilgisayardan FPGA'deki GA IP çekirdeğine verilmekte ve burada GA ile yol bulma işlemi gerçekleştirilmektedir. Dördüncü adımda, bulunan yol bilgisi Microblaze üzerinden bilgisayara gönderilmektedir. En son adımda ise, bulunan yol koordinatları hareket komutlarına dönüştürülüp RS-232 üzerinden Pioneer 3-DX gezgin robota gönderilmekte, robotun yol koordinatları boyunca hareket edip hedefe ulaşması sağlanmaktadır. Kullanılan FPGA'in hem GA'yı donanımsal olarak gerçekleştirmeye hem de kamera ve robot bağlantılarını sağlamaya kapasitesinin yetmemesi nedeniyle, bir bilgisayar kullanmak zorunda kalınmış, kamera ve gezgin robot doğrudan FPGA'e bağlanamamıştır.

FPGA üzerinde mikroişlemci tabanlı gömülü sistem oluşturmak için Xilinx ISE, XPS ve SDK programları kullanıldı. FPGA cihazı olarak Xilinx Virtex-5 XUPV5LX110T geliştirme kartı kullanıldı. FPGA geliştirme kartı tanıtım işlemi tamamlandıktan sonra; işlemci tipi olarak Microblaze, işlemci saat frekansı olarak 125 MHz ve bellek boyutu olarak da 64 KB dahili bellek seçildi. GA IP çekirdeğini oluşturmak için, XPS programında bulunan CIP sihirbazı kullanıldı. IP çekirdeği oluştururken, çekirdeğinin diğer çevresel birimlerle haberleşmesini sağlamak için PLB veri yolu seçildi. Yeni IP çekirdeği tekrar CIP sihirbazı kullanılarak Microblaze tabanlı sisteme eklendi. Rastgele sayı üretimi için, doğrusal hücresel otomat sisteminde kullanılan kural-90 ve 150 yapıları kullanıldı. Rastgele sayı üretimi için 16 bitlik ikili sayılar kullanıldı ve bu sayı dizisi içerisindeki farklı bitler GA'nın farklı operatörleri için tahsis edildi.

Sistem, Microblaze tabanlı ve gömülü olarak tasarlanmadan, sadece HDL ile tasarlanıp FPGA üzerinde de çalıştırılabilirdi. Fakat, işlemci kullanmadan gerçekleştirilen tasarımlarla karşılaştırıldığında, işlemci kullanarak gerçekleştirilen

gömülü sistemler tasarıma esneklik kazandırmaktadır. Bu esneklik sayesinde, tasarım aşamasından sonra bile tasarımın istenen parametreleri üzerinde değişiklik yapabilmek mümkün olmaktadır. Bu nedenle, daha esnek ve işlevsel olabilmesi için sistem Microblaze tabanlı olarak gerçekleştirildi ve IP çekirdek olarak tasarlandı.

Tez çalışmasında öncelikle, GA'nın tamamı C programlama dili kodlarıyla yazılarak, FPGA'de Microblaze işlemci üzerinde çalıştırıldı ve GA'nın çalışma süresi belirlendi. Toplam çalışma süresine bakıldığında, FPGA ile gerçekleştirilen yol bulma işleminin normal bir bilgisayarda çalıştırılan aynı GA'dan daha yavaş olduğu tespit edildi. GA'nın her bir operatörü için çalışma süreleri ayrı ayrı incelendiğinde, amaç fonksiyonun hesaplama zamanının diğer operatörlerden çok daha fazla sürdüğü görüldü. Amaç fonksiyonu hızlandırmak için, GA'nın diğer operatörlerinin Microblaze yazılımsal işlemci üzerinde çalıştığı, sadece amaç fonksiyon için IP donanımsal çekirdek gerçekleştirildiği hibrit bir sistem tasarlandı. Yazılımsal ve hibrit sistemin çalışma süreleri karşılaştırıldığında, donanımsal IP çekirdek olarak tasarlanan amaç fonksiyon modülünün, Microblaze sanal işlemcide çalışan yazılımsal amaç fonksiyonundan kat kat daha hızlı çalıştığı görüldü.

Hibrit sistemle elde edilen başarı sonrasında, GA'nın tamamı özgün bir IP çekirdeği olarak tasarlandı ve sistem çalıştırılıp performansı değerlendirilmeye tabi tutuldu. Bir bilgisayarda çalışan GA, Microblaze üzerinde çalışan GA ve IP çekirdek olarak yazılmış GA'ya ait çalışma süreleri eşit koşullarda karşılaştırıldı. IP çekirdek olarak gerçekleştirilen GA'nın, hem bilgisayar hem de Microblaze'de çalıştırılan yazılıma göre daha hızlı çalıştığı görüldü.

Tez çalışmasında, sistemin tamamını kolay bir şekilde kontrol edebilmek amacıyla Matlab-GUI tabanlı bir kullanıcı arayüzü tasarlandı. Arayüzde, yeni bir çalışma ortamı oluşturulabilmekte veya çalışma ortamı laboratuvar tavanına yerleştirilen bir kamera yardımı ile otomatik olarak da elde edilebilmektedir. GA'da kullanılan tüm parametreler arayüzde seçilebilmekte ve bu belirlenen parametrelere göre GA ile en uygun yol bulunabilmektedir. Arayüzde robotun temel hareket parametreleri ayarlanabilmekte ve belirlenmiş olan yol boyunca robotun hareketi sağlanabilmektedir. Ayrıca, kullanıcı robotun ilerleyeceği yolu çalışma ortamı

üzerinde baştan sona belirleyebileceği gibi, yolun her bir parçasını oluşturan hareket komutlarını robota tek tek göndererek robotun hareketini sağlayabilmektedir.

Sonuç olarak, tez çalışması kapsamında; GA operatörleri, gezgin robotların yol bulma problemi için uyarlandı, yeni bir mutasyon operatörü geliştirildi, GA ile yol bulma FPGA üzerinde özgün bir donanımsal IP çekirdek olarak gerçekleştirildi, sisteminin tamamı gerçek bir laboratuvar ortamında uygulanmış ve uygulama için bir arayüz yazıldı.

Gerçekleştirilen sistem başarılı bir şekilde çalışmasına rağmen, robot üzerinde yapılacak bazı eklemeler ile sistemin gelecekte daha etkin bir hale gelmesi sağlanabilir. Örneğin, robot üzerine yerleştirilecek 360° görüntü alabilen hareketli bir kamera ve daha yüksek kapasiteli bir gömülü sistem ile, robotun ortamın haritasını kendisinin çıkarması ve işlemesi sağlanabilir. Böylece, robot kısıtlı bir laboratuvar ortamında değil istenen herhangi bir ortamda da görevini yapabilir.

FPGA için gerekli olan enerjinin, robotun güç kaynakları üzerinden alınabilmesi halinde, FPGA doğrudan robot üzerine yerleştirilip çalıştırılabilir, böylece sistem tam olarak otonom hale getirilmiş olur.

KAYNAKLAR

[1] Hu Y., Yang S. X., A Knowledge Based Genetic Algorithm for Path Planning of a Mobile Robot, *Proceedings of the 2004 IEEE, International Conference on Robotics & Automation*, New Orleans, LA, 26 April-1 May 2004.

[2] Allaire F. C. J., Tarbouchi M., Labonté G., Fusina G., FPGA Implementation of Genetic Algorithm for UAV Real-Time Path Planning, *Journal of Intelligent and Robotic Systems*, 2009, **54**, 495-510.

[3] Tu J., Yang S. X., Genetic Algorithm Based Path Planning for a Mobile Robot, *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, 14-19 September 2003.

[4] Elshamli A., Abdullah H. A., Areibi S., Genetic Algorithm for Dynamic Path Planning, *Canadian Conference on Electrical and Computer Engineering*, Canada, 2-5 May 2004.

[5] Swarnalatha A., Shanthi A. P., Optimization of single variable functions using complete hardware evolution, *Applied Soft Computing*, 2012, **12**(4), 1322-1329.

[6] Mostafa H. E., Khadragi A. I., Hanafi Y. Y., Hardware implementation of genetic algorithm on FPGA, *21th National Radio Science Conference*, Cairo, Egypt, 16-18 March 2004.

[7] Scott S. D., Samal A., Seth S., HGA: A hardware-based genetic algorithm, *Proceedings of the Third Int. ACM Symposium on Field-Programmable Gate Arrays (FPGA'95)*, California, USA, 12-14 February 1995.

[8] Lysecky R. L., Vahid F., A Study of the Speedups and Competitiveness of FPGA Soft Processor Cores using Dynamic Hardware/Software Partitioning, *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)*, Munich, Germany, 7-11 March 2005.

[9] Al-Taharwa I., Sheta A., Al-Weshah M., A Mobile Robot Path Plannig Using Genetic Algorithm in Static Environment, *Journal of Computer Science*, 2008, **4**(4), 341-344.

[10] Nagib G., Gharieb W., Path Planning for a Mobile Robot Using Genetic Algorithms, *2004 International Conference on Electrical, Electronic and Computer Engineering, ICEEC'04*, Cairo, Egypt, 5-7 September 2004.

[11] Guo J., Liu L., Liu Q., Qu Y., An Improved of D* Algorithm for Mobile Robot Path Planning in Partial Unknown Environment, *2009 Second International Conference on Intelligent Computation Technology and Automation*, Changsa, Hunan, China, 10-11 October 2009.

[12] Çınar E., Parlaktuna O., Yazıcı A., Robot Navigasyonunda Potansiyel Alan Metodlarının Karşılaştırılması ve İç Ortamlarda Uygulanması, *Elektrik-Elektronik-Bilgisayar Mühendisliği 12. Ulusal Kongresi ve Fuarı*, Eskişehir, Türkiye, 14 Kasım 2007.

[13] Pakkan B., Ermiş M., İnsansız Hava Araçlarının Genetik Algoritma Yöntemiyle Çoklu Hedeflere Planlanması, *Havacılık ve Uzay Teknolojileri Dergisi*, 2010, **4**(3), 77-84.

[14] Chen W., Qin H., Path Planning of Mobile Robot Based On Hybrid Cascaded Genetic Algorithm, *9th World Congress on Intelligent Control and Automation (WCICA),* Taipei, Taiwan, 21-25 June 2011.

[15] Changan L., Xiaohu Y., Chunyang L., Guodong L., Dynamic Path Planning for Mobile Robot Based on Improved Genetic Algorithm, *Chinese Journal of Electronics*, 2010, **19**(2), 245-248.

[16] Kala R., Multi-robot path planning using co-evolutionary genetic programming, *Expert Systems with Applications*, 2012, **39**(3), 3817-3831.

[17] Fernando P. R., Katkoori S., Keymeulen D., Zebulum R., Stoica A., Customizable FPGA IP Core Implementation of a General-Puspose Genetic Algorithm Engine, *IEEE Transactions on Evolutionary Computation*, 2010, **14**(1), 133-149.

[18] Yan-cong Z., Jun-hua G., Yong-feng D., Huan-ping H., Implementation of Genetic Algorithm for TSP Based on FPGA, *Control and Decision Conference (CCDC)*, Mianyang, China, 23-25 May 2011.

[19] Kim D., Lee S., Improved mutation method for providing high genetic diversity of genetic algorithm processor, *IEICE Electronics Express*, 2012, **9**(9), 822-827.

[20] Deliparaschos K. M., Doyamis G. C., Tzafestas S. G., A parameterised genetic algorithm IP core: FPGA design, implementation and performance evaluation, *International Journal of Electronics*, 2008, **95**(11), 1149-1166.

[21] Dijkstra E. W., A Note on Two Problems in Connexion with Graphs, *Numerische Mathematik*, 1959, **1**(1), 269-271.

[22] Hart P. E., Nilsson N. J., Raphael B., A formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. Syst. Science and Cybern.*, 1968, **4**(2), 100-107.

[23] Stentz A., Optimal and efficient path planning for partially-known environments, *In Proceedings IEEE International Conference on Robotics and Automation*, San Diego, California, USA, 8-13 May 1994.

[24] Rimon E., Koditschek D. E., Exact robot navigation using artificial potential functions, *IEEE Trans. Robot. Autom.*, 1992, **8**(5), 501-518.

[25] Smierzchalski R., Michalewicz Z., Path Planning in Dynamic Environments, *Innovations in Robot Mobility and Control*, 2005, **8**, 135-153.

[26] Ündeğer Ç., Single and Multi Agent Real-Time Path Search in Dynamic and Partially Observable Environments, Doktora Tezi, Orta Doğu Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Ankara, 2007, 198835.

[27] Undeger C., Polat F., RTTES: Real-time search in dynamic environments, *Applied Intelligence*, 2007, **27**(3), 113-129.

[28] Koenig S., Likhachev M., Liu Y., Furcy D., Incremental Heuristic Search in Artificial Intelligence, *Artificial Intelligence Magazine*, 2004, **25**(2), 99-112.

[29] Stentz A., The focussed D* algorithm for real-time replanning, *In Proceedings of the International Joint Conference on Artificial Intelligence*, Canada, 20-25 August 1995.

[30] Koenig S., Likhachev M., D* Lite, *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*, Aberta, Canada, 28 July-1 August 2002.

[31] Korf R. E., Real-Time Heuristic Search, *Artificial Intelligence*, 1990, **42**, 189-211.

[32] Sedighi K. H., Ashenayi K., Manikas T. W., Wainwright R. L., Tai H. M., Autonomous Local Path Planning for a Mobile Robot Using a Genetic Algorithm, *Congress on Evolutionary Computation, CEC2004*, Portland, Oregon, USA, 19-23 June 2004.

[33] Gao M., Xu J., Tian J., Wu H., Path Planning for Mobile Robot Based on Chaos Genetic Algorithm, *Fourth International Conference on Natural Computation*, Jinan, China, 18-20 October 2008.

[34] Wai R., Liu C., Lin Y., Design of switching path-planning control for obstacle avoidance of mobile robot, *Journal of the Franklin Institute*, 2011, **348**(4), 718-737.

[35] Zhang Y., Zhang L., Zhang X., Mobile Robot Path Planning base on the Hybrid Genetic Algorithm in Unknown Environment, *ISDA '08. Eighth International Conference on Intelligent Systems Design and Applications*, Kaohsiung, China, 26-28 November 2008.

[36] Mahjoubi H., Bahrami F., Lucas C., Path Planning in an Environment with Static and Dynamic Obstacles Using Genetic Algorithm: A Simplified Search Space Approach, *2006 IEEE Congress on Evolutionary Computation*, Vancouver, Canada, 16-21 July 2006.

[37] Yao Z., Ma L., A Static Environment-Based Path Planning Method by Using Genetic Algorithm, *2010 International Conference on Computing, Control and Industrial Engineering (CCIE)*, Wuhan, China, 5-6 June 2010.

[38] Li Q., Zhang W., Yin Y., Wang Z., Liu G., An Improved Genetic Algorithm of Optimum Path Planning for Mobile Robots, *ISDA '06. Sixth International Conference on Intelligent Systems Design and Applications*, Jinan, China, 16-18 October 2006.

[39] Noto M., Sato H., A Method for the Shortest Path Search by Extended Dijkstra Algorithm, *2000 IEEE International Conference on Systems, Man, and Cybernetics*, Nashville, TN, USA, 8-11 October 2000.

[40] Dijkstra Algorithm, http://en.wikipedia.org/wiki/Dijkstra's_algorithm, (Ziyaret tarihi: 11 Ocak 2013).

[41] A* Algorithm, http://www.policyalmanac.org/games/aStarTutorial.htm, (Ziyaret tarihi: 16 Ocak 2013).

[42] Holland J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan, 1975.

[43] Goldberg D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed., Addison-Wesley Publishing Company Inc., Boston, MA, USA, 1989.

[44] Marra M. A., Walcott B. L., Stability and Optimality in Genetic Algorithm Controllers, *Proceedings of the 1996 IEEE International Symposium on Intelligent Control*, Dearborn, MI, USA, 15-18 September 1996.

[45] Angelov P. P., Wright J. A., A Center-of-Gravity-based Recombination Operator for Genetic Algorithms, *Proceedings of the 26th Annual Conference of the IEEE Industrial Electronics Society*, Nagoya, Japan, 22-28 October 2000.

[46] Manikas T. W., Ashenayi K., Wainwright R. L., Genetic Algorithms for Autonomous Robot Navigation, *IEEE Instrumentation & Measurement Magazine*, 2007, **10**(6)**,** 26-31.

[47] Fries T. P., Evolutionary Path Planning for Robot Navigation Under Varying Terrain Conditions, Editor: Mo H., *Handbook of Research on Artificial Immune Systems and Natural Computing: Applying Complex Adaptive Technologies*, 1st ed., IGI Global, USA, 361-382, 2009.

[48] Yıldırım M., Genetik Algoritmalar ve Benzetilmiş Tavlama ile Uzun Dönem Üretim Genişletme Planlaması, Doktora Tezi, Kocaeli Üniversitesi, Fen Bilimleri Enstitüsü, Kocaeli, 2003, 135971.

[49] Bresenham J. E., Algorithm for computer control of a digital plotter, *IBM Systems Journal*, 1965, **4**(1), 25-30.

[50] Yıldırım M., Erkan K., Determination of acceptable operating cost level of nuclear energy for Turkey's power system, *Energy*, 2007, **32**(2), 128-136.

[51] Gomez-Pulido J. A., Vega-Rodriguez M. A., Sanchez-Perez J. M., Priem-Mendes S., Carreira V., Accelerating floating-point fitness functions in

evolutionary algorithms a FPGA-CPU-GPU performance comparison, *Genetic Programming and Evolvable Machines*, 2011, **12**, 403-427.

[52] Corno F., Reorda M. S., Squillero G., Violante M., A Genetic Algorithm-based System for Generating Test Programs for Microprocessor IP Cores, *Proceedings of the 12th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2000,* Vancouver, BC, Canada, 13-15 November 2000.

[53] Xilinx Inc., Connecting Customized IP to the MicroBlaze Soft Processor Using the Fast Simplex Link (FSL) Channel, *Xilinx XAPP (v1.3)*, 2004.

[54] Maxfield C., *The Design Warrior's Guide to FPGAs: Devices, Tools and flows*, 1st ed., Newnes Publishers, Elsevier, 2004.

[55] http://www.xilinx.com (Ziyaret tarihi: 14 Mart 2011).

[56] http://www.altera.com (Ziyaret tarihi: 14 Mart 2011).

[57] http://www.atmel.com (Ziyaret tarihi: 14 Mart 2011).

[58] http://www.xilinx.com/univ/xupv5-lx110t.htm (Ziyaret tarihi: 21 Kasım 2011).

[59] Xilinx Inc., ML505/ML506/ML507 Evaluation Platform User Guide, *Xilinx UG347 (v3.1.2)*, 2011.

[60] Wang J., Loo S. M., Case Study of Finite Resource Optimization in FPGA using Genetic Algorithm, *International Journal of Computers and Their Applications*, 2010, **17**, 95-101.

[61] Xilinx Inc., Embedded System Tools Reference Guide EDK 11.3.1, *Xilinx, UG111*, 2009.

[62] Tuncer T., Bilgisayar Ağları için Saldırı Tespit Sistemi Tasarımları ve FPGA Ortamında Gerçekleştirilmesi, Doktora Tezi, Fırat Üniversitesi, Fen Bilimleri Enstitüsü, Elazığ, 2010, 259289.

[63] Xilinx Inc., MicroBlaze Processor Reference Guide EDK 11.4, *Xilinx, UG081 (v10.3)*, 2009.

[64] http://www.xilinx.com/support/documentation/ipembedprocess_memoryinterface_lmb.htm (Ziyaret tarihi: 8 Ekim 2012).

[65] Xilinx Inc., EDK Concepts,Tools,and Techniques - A Hands-On Guide to Effective Embedded System Design, *Xilinx, UG683 EDK 11*, 2009.

[66] Pedroni V. A., *Circuit Design with VHDL*, 1st ed., MIT Press Cambridge, Massachusetts London, England, 2004.

[67] http://www.mobilerobots.com/researchrobots/researchrobots/pioneerp3dx.aspx (Ziyaret tarihi: 18 Ekim 2010).

[68] Chang H. J., Lee C. S. G., Lu Y., Hu Y. C., P-SLAM: Simultaneous Localization and Mapping With Environmental-Structure Prediction, *IEEE Transactions on Robotics*, 2007, **23**(2), 281-293.

[69] Zhuang Y., Gu M., Wang W., Yu H., Multi-robot cooperative localization based on autonomous motion state estimation and laser data interaction, *Science China Information Sciences*, 2010, **53(**11), 2240-2250.

[70] Teimoori H., Savkin A. V., Equiangular navigation and guidance of a wheeled mobile robot based on range-only measurements, *Robotics and Autonomous Systems*, 2010, **58**(2), 203-215.

[71] Parlaktuna O., Sipahioglu A., Kirlik G., Yazici A., Multi-robot sensor-based coverage path planning using capacitated arc routing approach, *2009 IEEE Control Applications, (CCA) & Intelligent Control (ISIC) 2009*, St. Petersburg, Russia, 8-10 July 2009.

[72] MobileRobots Inc., *Pioneer 3 Operations Manual*, Version 6, September 2010.

[73] Susnea I., Filipescu A., Vasiliu G., Filipescu S., Path following, real-time, embedded fuzzy control of a mobile platform wheeled mobile robot, *Proceedings of the IEEE International Conference on Automation and Logistics, ICAL 2008*, Qingdao, China, 1-3 September 2008.

[74] http://www.ai.rug.nl/vakinformatie/pas/content/Ariamanual/main.html# (Ziyaret tarihi: 9 Aralık 2010).

[75] http://robots.mobilerobots.com/wiki/ARIA#Documentation (Ziyaret tarihi: 9 Aralık 2010)

[76] http://mathworks.com/products/matlab/ (Ziyaret tarihi: 12 Şubat 2011).

[77] Chen P., Chen R., Chang Y., Shieh L., Malki H. A., Hardware implementation for a genetic algorithm, *IEEE Transactions on Instrumentation and Measurement*, 2008, **57**(4), 699-705.

[78] Serra M., Slater T., Muzio J. C., Miller D. M., The Analysis of One-Dimensional Linear Cellular Automata and Their Aliasing Properties, *IEEE Transactions on Computer-Aided Design*, 1990, **9**(7), 767-778.

[79] Wolfram S., Statistical mechanics of cellular automata, *Reviews Modern Phys*, 1983, **55**, 601-644.

[80] Wolfram S., Universality and Complexity in Cellular Automata, *Physica 10D*, 1984, **10**, 1-35.

[81] Hidalgo J. I., Baraglia R., Perego R., Lanchares J., Tirade F., A parallel compact genetic algorithm for Multi-FPGA partitioning, *Proceedings of the Ninth Euromicro Workshop on Parallel and Distributed Processing*, Mantova, Italy, 7-9 February 2001.

[82] Glette K., Torresen J., A Flexible On-Chip Evolution System Implemented on a Xilinx Virtex-II Pro Device, *6th International Conference on Evolvable Systems*, Sitges, Spain, 12-14 September 2005.

[83] Kok J., Gonzalez L. F., Kelson N., FPGA Implementation of an Evolutionary Algorithm for Autonomous Unmanned Aerial Vehicle On-Board Path Planning, *IEEE Transactions on Evolutionary Computation*, 2013, **17**(2), 272-281.

[84] Hachour O., The Proposed Genetic FPGA Implementation For Path Planning of Autonomous Mobile Robot, *International Journal of Circuits, Systems and Signal Processing*, 2008, **2**(2), 151-167.

[85] Tsai C., Huang H., Chan C.. Parallel Elite Genetic Algorithm and Its Application to Global Path Planning for Autonomous Robot Navigation, *IEEE Transactions on Industrial Electronics*, 2011, **58**(10), 4813-4821.

EKLER

EK - A

Yol bulma için GA tabanlı C Kodu

```
#include <stdio.h>
#include <stdlib.h>
int adim_sayisi=8, nuf_buy=16, boyut = 16, ceza = 100;
float mutasyon_orani = 0.1, capraz_orani = 1;
float fitness_toplam=0;
float amac_fonksiyon(int [], int[][boyut]);
int fitness_fonksiyon(float [], float [],int);
int basla[2],hedef[2];
int main()
{
   int i,j,max,min, nufus[nuf_buy][adim_sayisi],jenerasyon;
   int ebeveyn1[nuf_buy/2][adim_sayisi],ebeveyn2[nuf_buy/2][adim_sayisi];
   int cocuk1[nuf_buy/2][adim_sayisi],cocuk2[nuf_buy/2][adim_sayisi],
   cocuk[nuf_buy][adim_sayisi];
   int elitist[adim_sayisi], nokta;
   float dilim,temp, min_amac, max_amac;
   float fitness[nuf_buy],rulet[nuf_buy],capraz[nuf_buy];
   float amac_fonk[nuf_buy];
   int secim[nuf_buy],d;
   int harita[16][16] = {
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,
      0,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,1,1,1,0,0,0,1,1,0,
      0,0,0,0,0,0,0,1,1,1,0,0,0,1,1,0,
      0,0,0,0,0,0,0,1,1,1,0,0,0,1,1,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,
      0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

  basla[0] = 0; basla[1] = 0;
  hedef[0] = 15; hedef[1] = 15;

 // Başlangıç Nüfusu Oluşturuluyor
// ----------------------------------------
 for(i=0;i<nuf_buy;i++)
    {
        for(j=0;j<adim_sayisi;j++)
```

```
          nufus[i][j] = rand()%boyut;
        amac_fonk[i] = amac_fonksiyon(nufus[i],harita);
  }
// Fitness Fonksiyonu Çağrılıyor.
// ------------------------------------------------------
min = fitness_fonksiyon(amac_fonk,fitness,1);
min_amac = amac_fonk[min];
for(jenerasyon=0;jenerasyon<100;jenerasyon++)
 {
    // Elitist Bulunuyor
    // -------------------------
    for(i=0;i<adim_sayisi;i++)
       elitist[i] = nufus[min][i];

    // Seçim İşlemi Yapılıyor
    // ------------------------------
   for(i=0;i<nuf_buy;i++)
    {
       rulet[i]=fitness_toplam*(float)rand()/RAND_MAX;
       capraz[i] = (float)rand()/RAND_MAX;
    }
    for(i=0;i<nuf_buy;i++)
    {
       if(capraz[i]<capraz_orani)
       {
          dilim = 0; d = 0;
          while(dilim<rulet[i])
          {
             dilim = dilim + fitness[d];
             d = d + 1;
          }
         secim[i] = d-1;
        }
       else
       {
         while(capraz[i]>capraz_orani)
         {
            rulet[i] = fitness_toplam*(float)rand()/RAND_MAX;
            capraz[i] = (float)rand()/RAND_MAX;
          }
          dilim = 0; d = 0;
          while(dilim<rulet[i])
          {
             dilim = dilim + fitness[d];
             d = d + 1;
           }
           secim[i] = d-1;
         }
       }
```

```
// Eşleştirme Havuzu
// ----------------------------
for(i=0;i<nuf_buy/2;i++)
{
   for(j=0;j<adim_sayisi;j++)
   {
        ebeveyn1[i][j]=nufus[secim[i]][j];
        ebeveyn2[i][j]=nufus[secim[i+nuf_buy/2]][j];
   }
}

 // Çaprazlama İşlemi
// ----------------------------------
 nokta=rand()%adim_sayisi;
 for(i=0;i<nuf_buy/2;i++)
 {
    for(j=0;j<nokta;j++)
    {
         cocuk1[i][j] = ebeveyn1[i][j];
         cocuk2[i][j] = ebeveyn2[i][j];
       }
       for(j=nokta;j<adim_sayisi;j++)
       {
          cocuk1[i][j] = ebeveyn2[i][j];
         cocuk2[i][j] = ebeveyn1[i][j];
      }
  }
  for(i=0;i<nuf_buy/2;i++)
       for(j=0;j<adim_sayisi;j++)
       {
                  cocuk[i][j] = cocuk1[i][j];
          cocuk[i+nuf_buy/2][j] = cocuk2[i][j];
       }

   // Mutasyon İşlemi
   // -------------------------------------
   for(i=0;i<nuf_buy;i++)
      for(j=0;j<adim_sayisi;j++)
      {
         temp = (float)rand()/RAND_MAX;
         if(temp<=mutasyon_orani)
            cocuk[i][j] = rand()%boyut;
      }
   // Yeni Nüfus
   // ------------------------------
   for(i=0;i<nuf_buy;i++)
   {
        for(j=0;j<adim_sayisi;j++)
            nufus[i][j] = cocuk[i][j];
```

```
                amac_fonk[i] = amac_fonksiyon(nufus[i],harita);
            }

            // Fitness Fonksiyonu Çağrılıyor.
            // ------------------------------------------
            max = fitness_fonksiyon(amac_fonk,fitness,0);
            max_amac = amac_fonk[max];
            for(i=0;i<adim_sayisi;i++)
                nufus[max][i] = elitist[i];
            for(i=0;i<nuf_buy;i++)
                 amac_fonk[i] = amac_fonksiyon(nufus[i],harita);
            min = fitness_fonksiyon(amac_fonk,fitness,1);
            min_amac = amac_fonk[min];
        }

        // En İyi Yol Koordinatları ve Amaç Fonksiyon Yazdırılıyor
        // -----------------------------------------------
        printf("En iyi yol: \n");
        printf("%d %d ",basla[0], basla[1]);
        for(i=0;i<adim_sayisi;i++)
           printf("%d ",nufus[min][i]);
        printf("%d %d\n",hedef[0], hedef[1]);
        printf("\nmesafe: %f\n",min_amac);
}

// Sıralama İşlemi için Fitness Fonksiyon
// ------------------------------------------------------------
int fitness_fonksiyon(float amac_fonk_fit[], float fitness[],int maxmin)
{
  int b[nuf_buy], i, j, indis, temp;
  fitness_toplam = 0;
  for (i = 0; i < nuf_buy; i++)
     b[i] = i;

  for (i = 0; i < nuf_buy - 1; i++)
    for (j = 0; j < nuf_buy - 1 - i; j++)
      if (amac_fonk_fit[b[j]] > amac_fonk_fit[b[j + 1]])
      {
        temp = b[j];
        b[j] = b[j + 1];
        b[j + 1] = temp;
      }

  for(i=0;i<nuf_buy;i++)
  {
    fitness[b[i]] = ((float)nuf_buy+1-((float)i+1))/(float)nuf_buy;
    fitness_toplam = fitness_toplam + fitness[b[i]];
  }
  if(maxmin==1)
```

```
  return b[0];
  else
  return b[nuf_buy-1];
}

// Amaç Fonksiyon Hesaplama
// -----------------------------------------
float amac_fonksiyon(int nufus[], int harita[][boyut])
{
   int i,j,x1,x2,y1,y2,n,egim,dx,dy,amac_dizi[adim_sayisi+4];
   float amac = 0;
   int ix,iy,fx,fy, m,m1,k;
   int d[16] = {1023,1023,512,341,256,205,171,146,128,114,102,93,85,79,73,68};

   amac_dizi[0] = basla[0];
   amac_dizi[1] = basla[1];
   for(i=0;i<adim_sayisi;i++)
      amac_dizi[i+2] = nufus[i];
   amac_dizi[adim_sayisi+2] = hedef[0];
   amac_dizi[adim_sayisi+3] = hedef[1];

   for(i=0;i<adim_sayisi+1;i=i+2)
   {
     x1 = amac_dizi[i]; y1 = amac_dizi[i+1];
     x2 = amac_dizi[i+2]; y2 = amac_dizi[i+3];
     amac = amac + sqrt(pow((x2-x1),2)+pow((y2-y1),2));
     dx = abs(x2-x1);
     dy = abs(y2-y1);
     if (dx>dy)
       m=dx;
    else
       m=dy;
    m1 = d[m];
    ix= (x2-x1)*m1;
    iy= (y2-y1)*m1;
    while (m>0)
    {
      fx+=ix;
      fy+=iy;
      m--;
      if(harita[px][py]==1)
        amac = amac + ceza;
    }
   }
   return amac;
}
```

EK - B

GA IP Çekirdeği için VHDL Kodu

```
------------------------------------------------------------------------------
-- Filename:     user_logic.vhd
-- Version:      1.00.a
-- Description: User logic.
-- Date:         Wed Feb 15 13:06:22 2012 (by Create and Import Peripheral Wizard)
-- VHDL Standard:    VHDL'93
------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library proc_common_v3_00_a;
use proc_common_v3_00_a.proc_common_pkg.all;

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
--   C_SLV_DWIDTH          -- Slave interface data bus width
--   C_NUM_REG             -- Number of software accessible registers

--   Bus2IP_Clk            -- Bus to IP clock
--   Bus2IP_Reset          -- Bus to IP reset
--   Bus2IP_Addr           -- Bus to IP address bus
--   Bus2IP_Data           -- Bus to IP data bus
--   Bus2IP_BE             -- Bus to IP byte enables
--   Bus2IP_RdCE           -- Bus to IP read chip enable
--   Bus2IP_WrCE           -- Bus to IP write chip enable
--   IP2Bus_Data           -- IP to Bus data bus
--   IP2Bus_RdAck          -- IP to Bus read transfer acknowledgement
--   IP2Bus_WrAck          -- IP to Bus write transfer acknowledgement
--   IP2Bus_Error          -- IP to Bus error response
------------------------------------------------------------------------------
entity user_logic is
  generic
  (
    C_SLV_DWIDTH          : integer             := 32;
    C_NUM_REG             : integer             := 22  );
  port
  (
    LEDs                : out std_logic_vector (0 to 7);
   Bus2IP_Clk     : in  std_logic;
   Bus2IP_Reset    : in  std_logic;
   Bus2IP_Addr     : in  std_logic_vector(0 to 31);
```

```
  Bus2IP_Data      : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
  Bus2IP_BE        : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
  Bus2IP_RdCE      : in  std_logic_vector(0 to C_NUM_REG-1);
  Bus2IP_WrCE      : in  std_logic_vector(0 to C_NUM_REG-1);
  IP2Bus_Data      : out std_logic_vector(0 to C_SLV_DWIDTH-1);
  IP2Bus_RdAck     : out std_logic;
  IP2Bus_WrAck     : out std_logic;
  IP2Bus_Error     : out std_logic
 );
 attribute SIGIS : string;
 attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
 attribute SIGIS of Bus2IP_Reset  : signal is "RST";
end entity user_logic;

------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of user_logic is

signal KONTROL              : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal DURUM                : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal HARITA_0             : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal HARITA_1             : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal HARITA_2             : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal HARITA_3             : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal HARITA_4             : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal HARITA_5             : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal HARITA_6             : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal HARITA_7             : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal HARITA_8             : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal HARITA_9             : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal HARITA_10            : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal HARITA_11            : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal HARITA_12            : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal HARITA_13            : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal HARITA_14            : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal HARITA_15            : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal LED_AC               : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal LED_KAPAT            : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal BASLA_HEDEF          : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal YOL_KOORDINAT  : std_logic_vector(0 to C_SLV_DWIDTH-1);

signal slv_reg_write_sel   : std_logic_vector(0 to 21);
signal slv_reg_read_sel    : std_logic_vector(0 to 21);
signal slv_ip2bus_data     : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_read_ack        : std_logic;
signal slv_write_ack       : std_logic;
constant rng_size          : integer := 16;
```

```
constant pop_size           : integer := 16;
constant pop_width          : integer := 8;
constant fitness_toplam     : integer := 136;
constant mutasyon_orani     : integer := 12;

TYPE state1 IS (st_NUFUSINIT, st_NUFUS, st_CAPRAZ, st_MUTASYON);
TYPE state2 IS (st_AMACINIT, st_AMACHESAP);
TYPE state3 IS (st_SIRAINIT, st_SIRALAMA);
TYPE state4 IS (st_SECIMINIT, st_SECIM);
TYPE state5 IS (st_ESLEINIT, st_ESLE);

SIGNAL nufus_state: state1;
SIGNAL amac_state: state2;
SIGNAL sira_state: state3;
SIGNAL secim_state: state4;
SIGNAL esle_state: state5;

TYPE nufus_tip IS ARRAY (0 to pop_size-1,0 to pop_width-1) of integer range 0 to
15;
TYPE ebeveyn_tip IS ARRAY (0 to pop_size/2-1,0 to pop_width-1) of integer range
0 to 15;
TYPE dizi_tip IS ARRAY (0 to 15) of integer range 0 to 1023;
TYPE rulet_tip IS ARRAY (0 to pop_size-1) of INTEGER RANGE 0 to
fitness_toplam;
TYPE min_max_tip IS ARRAY (0 to pop_size-1) of INTEGER RANGE 0 to
pop_size-1;
TYPE amac_func_tip IS ARRAY (0 to pop_size-1) of INTEGER RANGE 0 to
50000;
TYPE elitist_tip IS ARRAY (0 to pop_width-1) of INTEGER RANGE 0 to 15;
TYPE harita_tip IS ARRAY (0 to 15,0 to 15) of integer range 0 to 3;

SIGNAL rnd_num: std_logic_vector(0 to rng_size-1);
SIGNAL nufus: nufus_tip;
SIGNAL ebeveyn1,ebeveyn2: ebeveyn_tip;
SIGNAL amac_func: amac_func_tip;
SIGNAL fitness,rulet,secim: rulet_tip;
SIGNAL elitist: elitist_tip;
SIGNAL min_amac: integer range 0 to 50000;
SIGNAL nuf_min_indis,nuf_max_indis: integer range 0 to pop_size;
SIGNAL nuf_bitti, nuf_ack, amac_ack, sira_ack, secim_ack, esle_ack, elitist_ata,
elit_ack, nuf_to_amac: std_logic := '0';
SIGNAL kontrol: std_logic := '0';
SIGNAL harita: harita_tip;

begin
  slv_reg_write_sel <= Bus2IP_WrCE(0 to 21);
  slv_reg_read_sel  <= Bus2IP_RdCE(0 to 21);
  slv_write_ack     <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or
  Bus2IP_WrCE(2) or Bus2IP_WrCE(3) or Bus2IP_WrCE(4) or
```

```
  Bus2IP_WrCE(5) or Bus2IP_WrCE(6) or Bus2IP_WrCE(7) or
  Bus2IP_WrCE(8) or Bus2IP_WrCE(9) or Bus2IP_WrCE(10) or
  Bus2IP_WrCE(11) or Bus2IP_WrCE(12) or Bus2IP_WrCE(13) or
  Bus2IP_WrCE(14) or Bus2IP_WrCE(15) or Bus2IP_WrCE(16) or
  Bus2IP_WrCE(17) or Bus2IP_WrCE(18) or Bus2IP_WrCE(19) or
  Bus2IP_WrCE(20) or Bus2IP_WrCE(21);

  slv_read_ack      <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or
  Bus2IP_RdCE(2) or Bus2IP_RdCE(3) or Bus2IP_RdCE(4) or
  Bus2IP_RdCE(5) or Bus2IP_RdCE(6) or Bus2IP_RdCE(7) or
  Bus2IP_RdCE(8) or Bus2IP_RdCE(9) or Bus2IP_RdCE(10) or
  Bus2IP_RdCE(11) or Bus2IP_RdCE(12) or Bus2IP_RdCE(13) or
  Bus2IP_RdCE(14) or Bus2IP_RdCE(15) or Bus2IP_RdCE(16) or
  Bus2IP_RdCE(17) or Bus2IP_RdCE(18) or Bus2IP_RdCE(19) or
  Bus2IP_RdCE(20) or Bus2IP_RdCE(21);

SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
  begin
    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
          if Bus2IP_Reset = '1' then
              KONTROL                 <= (others => '0');
              DURUM                   <= (others => '0');
              HARITA_0                <= (others => '0');
              HARITA_1                <= (others => '0');
              HARITA_2                <= (others => '0');
              HARITA_3                <= (others => '0');
              HARITA_4                <= (others => '0');
              HARITA_5                <= (others => '0');
              HARITA_6                <= (others => '0');
              HARITA_7                <= (others => '0');
              HARITA_8                <= (others => '0');
              HARITA_9                <= (others => '0');
              HARITA_10               <= (others => '0');
              HARITA_11               <= (others => '0');
              HARITA_12               <= (others => '0');
              HARITA_13               <= (others => '0');
              HARITA_14               <= (others => '0');
              HARITA_15               <= (others => '0');
              LED_AC                  <= (others => '0');
              LED_KAPAT               <= (others => '0');
              BASLA_HEDEF             <= (others => '0');
              YOL_KOORDINAT           <= (others => '0');

      else
        case slv_reg_write_sel is
          when "10000000000000000000" =>
            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
```

```
            KONTROL(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;
        when "01000000000000000000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            KONTROL(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;
        when "00100000000000000000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            HARITA_0(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;
        when "00010000000000000000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            HARITA_1(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;
        when "00001000000000000000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            HARITA_2(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;
        when "00000100000000000000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            HARITA_3(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;
        when "00000010000000000000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            HARITA_4(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;
        when "00000001000000000000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
```

```
            HARITA_5(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;
        when "00000000100000000000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            HARITA_6(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;
        when "00000000010000000000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            HARITA_7(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;
        when "00000000001000000000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            HARITA_8(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;
        when "00000000000100000000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            HARITA_9(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;
        when "00000000000010000000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            HARITA_10(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;
        when "00000000000001000000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            HARITA_11(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;
        when "00000000000000100000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
```

```
            HARITA_12(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;
        when "00000000000000001000000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            HARITA_13(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;

        when "00000000000000000100000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            HARITA_14(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;

        when "00000000000000000010000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            HARITA_15(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;

        when "00000000000000000001000" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            LEDs(0 to 7) <= "11111111";
          end if;
         end loop;
        when "00000000000000000000100" =>
         for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            LEDs(0 to 7) <= "00000000";
          end if;
         end loop;

         when "00000000000000000000010" =>
          for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
           if ( Bus2IP_BE(byte_index) = '1' ) then
            BAS_HED(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
         end loop;
```

```
          when "00000000000000000001" =>
          for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
           if ( Bus2IP_BE(byte_index) = '1' ) then
             YOL_KOORDINAT(byte_index*8 to byte_index*8+7) <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
           end if;
          end loop;
  STAT(31) <= '0';
    if Kontrol = '1' then
      YOL_KOORDINAT(0 to 3) <=
conv_std_logic_vector(nufus(nuf_min_indis,7),4);
       YOL_KOORDINAT(4 to 7) <=
conv_std_logic_vector(nufus(nuf_min_indis,6),4);
       YOL_KOORDINAT(8 to 11) <=
conv_std_logic_vector(nufus(nuf_min_indis,5),4);
       YOL_KOORDINAT(12 to 15) <=
conv_std_logic_vector(nufus(nuf_min_indis,4),4);
       YOL_KOORDINAT(16 to 19) <=
conv_std_logic_vector(nufus(nuf_min_indis,3),4);
       YOL_KOORDINAT(20 to 23) <=
conv_std_logic_vector(nufus(nuf_min_indis,2),4);
       YOL_KOORDINAT(24 to 27) <=
conv_std_logic_vector(nufus(nuf_min_indis,1),4);
       YOL_KOORDINAT(28 to 31) <=
conv_std_logic_vector(nufus(nuf_min_indis,0),4);
      STAT(31) <= '1';
    end if;
        when others => null;
       end case;
     end if;
   end if;

end process SLAVE_REG_WRITE_PROC;

SLAVE_REG_READ_PROC: process( slv_reg_read_sel, KONTROL, DURUM,
HARITA_0, HARITA_1, HARITA_2, HARITA_3, HARITA_4, HARITA_5,
HARITA_6, HARITA_7, HARITA_8, HARITA_9, HARITA_10, HARITA_11,
HARITA_12, HARITA_13, HARITA_14, HARITA_15, LED_ON, LED_OFF,
BASLA_HEDEF, YOL_KOORDINAT ) is
 begin
  case slv_reg_read_sel is
    when "10000000000000000000" => slv_ip2bus_data <= KONTROL;
    when "01000000000000000000" => slv_ip2bus_data <= DURUM;
    when "00100000000000000000" => slv_ip2bus_data <= HARITA_0;
    when "00010000000000000000" => slv_ip2bus_data <= HARITA_1;
    when "00001000000000000000" => slv_ip2bus_data <= HARITA_2;
    when "00000100000000000000" => slv_ip2bus_data <= HARITA_3;
    when "00000010000000000000" => slv_ip2bus_data <= HARITA_4;
    when "00000001000000000000" => slv_ip2bus_data <= HARITA_5;
```

```
    when "00000000100000000000" => slv_ip2bus_data <= HARITA_6;
    when "00000000010000000000" => slv_ip2bus_data <= HARITA_7;
    when "00000000001000000000" => slv_ip2bus_data <= HARITA_8;
    when "00000000000100000000" => slv_ip2bus_data <= HARITA_9;
    when "00000000000010000000" => slv_ip2bus_data <= HARITA_10;
    when "00000000000001000000" => slv_ip2bus_data <= HARITA_11;
    when "00000000000000100000" => slv_ip2bus_data <= HARITA_12;
    when "00000000000000010000" => slv_ip2bus_data <= HARITA_13;
    when "00000000000000001000" => slv_ip2bus_data <= HARITA_14;
    when "00000000000000000100" => slv_ip2bus_data <= HARITA_15;
    when "00000000000000000010" => slv_ip2bus_data <= LED_AC;
    when "00000000000000000001" => slv_ip2bus_data <= LED_KAPAT;
    when "00000000000000000000" => slv_ip2bus_data <= BASLA_HEDEF;
    when "00000000000000000000" => slv_ip2bus_data <= YOL_KOORDINAT;
    when others => slv_ip2bus_data <= (others => '0');
   end case;
  end process SLAVE_REG_READ_PROC;
  IP2Bus_Data      <= slv_ip2bus_data when slv_read_ack = '1' else (others => '0');
  IP2Bus_WrAck   <= slv_write_ack;
  IP2Bus_RdAck   <= slv_read_ack;
  IP2Bus_Error     <= '0';

---- NÜFUS OLUŞTURMA ----------------------------------------
PROCESS (Bus2IP_Clk, Bus2IP_Reset)
    variable i,j,k: integer range 0 to 50 := 0;
BEGIN
  IF (Bus2IP_Reset='1') THEN
    i := 0; j:= 0; k:= 0;
    nufus_state <= st_NUFUSINIT;
  ELSIF (Bus2IP_Clk'EVENT AND Bus2IP_Clk='1') THEN
  CASE nufus_state IS
    WHEN st_NUFUSINIT =>
       nuf_ack <= '0';
       nuf_to_amac <= '0';
       if elitist_ata = '1' then
         for k in 0 to pop_width-1 loop
            nufus(nuf_max_indis,k) <= elitist(k); -- elitist atama işlemi yapıldı
         end loop;
         nuf_to_amac <= '1'
       end if;
       if nuf_bitti = '0' then
         nufus_state <= st_NUFUS;
       else
          nufus_state <= st_NUFUSINIT;
         end if;
         if esle_ack = '1' then --Eşleştirme işlemi bittiyse çaprazlama-mutasyon başla
          i := 0;
          nufus_state <= st_CAPRAZ;
         end if;
```

```
 WHEN st_NUFUS =>
   if KONTROL(31) = '1' then
     nufus(i,j) <= conv_integer('0' & rnd_num(12 to 15));
     j := j + 1;
     if (j=pop_width) then
       i := i + 1; j := 0;
    end if;
    if(i=pop_size) then
      nuf_bitti <= '1';
      nuf_ack <= '1'; -- Nüfus oluşturma işlemi bitti
      nufus_state <= st_NUFUSINIT;
    else
      nufus_state <= st_NUFUS;
    end if;
  end if;

WHEN st_CAPRAZ =>
   for j in 0 to pop_width-1 loop
     if(j < conv_integer(rnd_num(11 to 13))) then
        nufus(i,j) <= ebeveyn1(i,j);
        nufus(i+pop_size/2,j) <= ebeveyn2(i,j);
     end if;
     if(j >= conv_integer(rnd_num(11 to 13))) then
        nufus(i,j) <= ebeveyn2(i,j);
        nufus(i+pop_size/2,j) <= ebeveyn1(i,j);
     end if;
   end loop;
   i := i + 1;
   if i=pop_size/2 then
      i := 0; j:= 0;
      nufus_state <= st_MUTASYON;
   else
      nufus_state <= st_CAPRAZ;
   end if;

WHEN st_MUTASYON =>
   if (conv_integer('0' & rnd_num(2 to 8)) < mutasyon_orani) then
      nufus(i,j) <= conv_integer('0' & rnd_num(12 to 15));
  end if;
   j := j + 1;
   if (j=pop_width) then
      i := i + 1; j := 0;
   end if;
   if(i=pop_size) then
      nuf_ack <= '1'; -- Çaprazlama ve mutasyon işlemi bitti, Yeni nüfus oluştu
      nufus_state <= st_NUFUSINIT;
   else
        nufus_state <= st_MUTASYON;
    end if;
```

```
      END CASE;
   END IF;
END PROCESS;

---- AMAÇ FONKSİYON HESAPLAMA ----------------------------------------
PROCESS (Bus2IP_Clk, Bus2IP_Reset)
  variable i,n,j: integer range 0 to 50 := 0;
  variable toplam: integer range 0 to 50000 := 0;
  variable x1,x2,y1,y2,dx,dy,m: integer range 0 to 20 := 0;
  variable m1: integer range 0 to 1023 := 0;
  variable ix,iy: integer range -15500 to 15500 := 0;
  variable fx,fy: integer range -300000 to 300000 := 0;
  variable px,py: integer range -40000 to 40000 := 0;
  variable temp1,temp2: std_logic_vector(0 to 13);
  variable dizi: dizi_tip:=
(1023,1023,512,341,256,205,171,146,128,114,102,93,85,79,73,68);
BEGIN
   IF (Bus2IP_Reset='1') THEN
      toplam := 0; cap := 0; n := 0; i := 0; j := 0;
      amac_state <= st_AMACINIT;
   ELSIF (Bus2IP_Clk'EVENT AND Bus2IP_Clk='1') THEN
      CASE amac_state IS
         WHEN st_AMACINIT =>
            if nuf_ack = '0' then -- Nüfus işlemi oluşmadıysa bekle
              amac_state <= st_AMACINIT;
            else
              toplam := 0; cap := 0; n := 0; i := 0; j := 0;
              amac_state <= st_AMACHESAP;
            end if;
         WHEN st_AMACHESAP =>
            if n=0 then
               x1 := conv_integer('0' & BASLA_HEDEF(0 to 3));
               y1 := conv_integer('0' & BASLA_HEDEF(4 to 7));
               x2 := nufus(i,0);
               y2 := nufus(i,1);
            end if;
            if n=1 then
               x1 := conv_integer('0' & BASLA_HEDEF(8 to 11));
               y1 := conv_integer('0' & BASLA_HEDEF(12 to 15));
               x2 := nufus(i,pop_width-2);
               y2 := nufus(i,pop_width-1);
            end if;
            if n>1 then
               x1 := nufus(i,j*2);
               y1 := nufus(i,j*2+1);
               x2 := nufus(i,j*2+2);
               y2 := nufus(i,j*2+3);
            end if;
            dx := abs(x2-x1);
```

```
            dy := abs(y2-y1);
            toplam := toplam + dx + dy;
            if(dx>dy) then
               m := dx;
            else
               m := dy;
            end if;
            m1 := dizi(m);
            ix := (x2-x1)*m1;
            iy := (y2-y1)*m1;
            for nokta in 0 to 15 loop
              px := fx + 512;
              py := fy + 512;
              fx := fx+ix;
              fy := fy+iy;
              x1 := conv_integer(temp1(0 to 3));
              y1 := conv_integer(temp2(0 to 3));
              if nokta<=m then
                if(harita(x1,y1)=1) then
                   toplam := toplam + 100;
                end if;
              end if;
            end loop;
            n := n + 1;
            if n>2 then
               j := j + 1;
            end if;
            if (j*2>pop_width-4) then
               j := 0; n := 0;
               amac_func(i) <= toplam;
               i := i + 1;
               toplam := 0;
            end if;
            if i = pop_size then
              amac_state <= st_AMACINIT;
            else
              amac_state <= st_AMACHESAP;
            end if;
         END CASE;
      END IF;
END PROCESS;

---- SIRALAMA ------------------------------------------------
PROCESS (Bus2IP_Clk, Bus2IP_Reset)
  variable i,indis,temp_sira: integer range 0 to 50 := 0;
  variable sira_sayac: std_logic := '0';
  variable min_max: min_max_tip;
BEGIN
  IF (Bus2IP_Reset='1') THEN
```

```
      kontrol <= '0';
      sira_sayac := '0';
      sira_state <= st_SIRAINIT;
    ELSIF (Bus2IP_Clk'EVENT AND Bus2IP_Clk='1') THEN
      CASE sira_state is
        WHEN st_SIRAINIT =>
          sira_ack <= '0';
          elitist_ata <= '0';
          if amac_ack = '0' then -- Amac hesaplama bitmediyse bekle
            sira_state <= st_SIRAINIT;
          else
            sira_state <= st_SIRALAMA;
          end if;
        WHEN st_SIRALAMA =>
          sira_sayac := not sira_sayac; -- sayac her defasında sıfırlanabilir
          for i in 0 to pop_size-1 loop
            min_max(i) := i;
          end loop;
          for i in 0 to pop_size-2 loop
            for j in 0 to pop_size-2-i loop
              if(amac_func(min_max(j)) > amac_func(min_max(j+1))) then
                temp_sira := min_max(j);
                min_max(j) := min_max(j+1);
                min_max(j+1) := temp_sira;
              end if;
            end loop;
          end loop;
          for i in 0 to pop_size-1 loop
            fitness(min_max(i)) <= pop_size-i;
          end loop;
          if sira_sayac = '0' then
            sira_ack <= '0';
            elitist_ata <= '1';
          else
            min_amac <= amac_func(min_max(0));
            nuf_min_indis <= min_max(0);
            nuf_max_indis <= min_max(pop_size-1);
            if gener > 99 then
              sira_ack <= '0';
              kontrol <= '1';
            else
              sira_ack <= '1';
            end if;
          end if;
          sira_state <= st_SIRAINIT;
      END CASE;
    END IF;
END PROCESS;
```

```
---- SEÇİM --------------------------------------------------
PROCESS (Bus2IP_Clk, Bus2IP_Reset)
  variable i,j,d: integer range 0 to 50 := 0;
  variable durum: integer range 0 to 5 := 0;
  variable dilim: integer range 0 to fitness_toplam := 0;
  variable temp: integer range 0 to 512 := 0;
BEGIN
  IF (Bus2IP_Reset='1') THEN
     gener <= 0;
     secim_state <= st_SECIMINIT;
   ELSIF (Bus2IP_Clk'EVENT AND Bus2IP_Clk='1') THEN
      CASE secim_state is
         WHEN st_SECIMINIT =>
            secim_ack <= '0';
            if sira_ack = '0' then -- Sıralama işlemi bittiyse Seçim işlemine başla.
              durum := 0;
              secim_state <= st_SECIMINIT;
            else
              secim_state <= st_SECIM;
            end if;
         WHEN st_SECIM =>
            if durum = 0 then
               for i in 0 to pop_width-1 loop
                  elitist(i) <= nufus(nuf_min_indis,i);
               end loop;
               gener <= gener + 1;
               durum := 1;i := 0;
               secim_state <= st_SECIM;
            elsif durum = 1 then
               rulet(i) <= conv_integer(rnd_num(9 to 15) + rnd_num(12 to 14)) + 2;
               i := i + 1;
               if i = pop_size then
                  durum := 2;
                  i := 0;
               end if;
               secim_state <= st_SECIM;
             else
               dilim := 0; d := 0;
               for j in 0 to pop_size-1 loop
                  if (dilim < rulet(i)) then
                    dilim := dilim + fitness(d);
                    d := d + 1;
                  end if;
               end loop;
               secim(i) <= abs(d - 1);
               i := i + 1;
               if i = pop_size then
                  secim_ack <= '1'; -- Seçim işlemi bitti
                  secim_state <= st_SECIMINIT;
```

```
                else
                   secim_state <= st_SECIM;
                end if;
              end if;
          END CASE;
     END IF;
END PROCESS;

---- EŞLEŞTİRME ------------------------------------------------
PROCESS (Bus2IP_Clk, Bus2IP_Reset)
   variable i,j: integer range 0 to 50 := 0;
BEGIN
   IF (Bus2IP_Reset='1') THEN
      esle_state <= st_ESLEINIT;
   ELSIF (Bus2IP_Clk'EVENT AND Bus2IP_Clk='1') THEN
     CASE esle_state is
        WHEN st_ESLEINIT =>
          esle_ack <= '0';
          if secim_ack = '0' then -- Seçim işlemi bitmediyse bekle
            esle_state <= st_ESLEINIT;
          else
            i := 0; j := 0;
            esle_state <= st_ESLE;
          end if;
       WHEN st_ESLE =>
          for j in 0 to pop_width-1 loop
             ebeveyn1(i,j) <= nufus(secim(i),j);
             ebeveyn2(i,j) <= nufus(secim(i+pop_size/2),j);
          end loop;
          i := i + 1;
          if i=pop_size/2 then
             esle_ack <= '1'; -- Eşleştirme işlemi bitti, Sonraki adım çaprazlama-
mutasyon
             esle_state <= st_ESLEINIT;
          else
             esle_state <= st_ESLE;
          end if;
      END CASE;
   END IF;
END PROCESS;

---- RASTGELE SAYI ÜRETİCİ ---------------------------------
PROCESS (Bus2IP_Clk,Bus2IP_Reset)
   variable rast1  : std_logic_vector(0 to rng_size-1) := "1010101000010101";
   variable rast2 : std_logic_vector(0 to rng_size-1);
   variable evenodd : std_logic;
BEGIN
IF (Bus2IP_Clk'EVENT AND Bus2IP_Clk='1') THEN
   rast2(rng_size-1) := '0' xor rast1(rng_size-1) xor rn1(rng_size-2);
```

```
   rast2(rng_size-2) := rast1(rng_size-1) xor rast2(rng_size-2) xor rast1(rng_size-3);
   evenodd := '0';
   for i in 1 to rng_size-3 loop
     rast2(i) := rast1(i+1) xor rast1(i-1);
     if evenodd = '1' then
       rast2(i) := rast2(i) xor rast1(i);
     end if;
     evenodd := not evenodd;
   end loop;
   rast2(0) := rast1(1) xor '0';
   if evenodd = '1' then
     rast2(0) := rast2(0) xor rast1(0);
   end if;
   rast1 := rast2;
         rnd_num <= rast1;
END IF;
END PROCESS;
-------------------------------------------------------------------------------

READ_HARITA_PROC : PROCESS( HARITA_0, HARITA_1, HARITA_2,
HARITA_3, HARITA_4,HARITA_5, HARITA_6, HARITA_7, HARITA_8,
HARITA_9, HARITA_10, HARITA_11, HARITA_12, HARITA_13, HARITA_14,
HARITA_15 ) is
BEGIN
FOR j in 15 downto 0 LOOP
  harita(0,j) <= conv_integer('0' &HARITA_0(2*j to 2*j+1));
  harita(1,j) <= conv_integer('0' &HARITA_1(2*j to 2*j+1));
  harita(2,j) <= conv_integer('0' &HARITA_2(2*j to 2*j+1));
  harita(3,j) <= conv_integer('0' &HARITA_3(2*j to 2*j+1));
  harita(4,j) <= conv_integer('0' &HARITA_4(2*j to 2*j+1));
  harita(5,j) <= conv_integer('0' &HARITA_5(2*j to 2*j+1));
  harita(6,j) <= conv_integer('0' &HARITA_6(2*j to 2*j+1));
  harita(7,j) <= conv_integer('0' &HARITA_7(2*j to 2*j+1));
  harita(8,j) <= conv_integer('0' &HARITA_8(2*j to 2*j+1));
  harita(9,j) <= conv_integer('0' &HARITA_9(2*j to 2*j+1));
  harita(10,j) <= conv_integer('0' &HARITA_10(2*j to 2*j+1));
  harita(11,j) <= conv_integer('0' &HARITA_11(2*j to 2*j+1));
  harita(12,j) <= conv_integer('0' &HARITA_12(2*j to 2*j+1));
  harita(13,j) <= conv_integer('0' &HARITA_13(2*j to 2*j+1));
  harita(14,j) <= conv_integer('0' &HARITA_14(2*j to 2*j+1));
  harita(15,j) <= conv_integer('0' &HARITA_15(2*j to 2*j+1));
END LOOP;
END PROCESS READ_HARITA_PROC;
end IMP;
```

GA_IP ÇEkirdeği için Microblaze Kodu

```
#include "xparameters.h"
#include "mb_interface.h"
#include "stdio.h"
#include "xutil.h"
#include "stdlib.h"

#define ADDR_KONTROL            XPAR_GA_0_BASEADDR + 0
#define ADDR_DURUM              XPAR_GA_0_BASEADDR + 4
#define ADDR_BAS_HEDEF          XPAR_GA_0_BASEADDR + 8
#define ADDR_YOL_KOORDINAT      XPAR_GA_0_BASEADDR + 12
#define ADDR_HARITA_0           XPAR_GA_0_BASEADDR + 16
#define ADDR_LED_AC             XPAR_GA_0_BASEADDR + 80
#define ADDR_LED_KAPAT          XPAR_GA_0_BASEADDR + 84

led_ac();
led_kapat();

int main()
{
    int *addr_ptr;
    addr_ptr = ADDR_KONTROL;
    *addr_ptr = 0x0;

    while (1) // Harita RS-232 portundan okunuyor
    {
        harita[i][j] = XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR);
        j = j + 1;
        if (j == 10)
        {
           j = 0;
           i = i + 1;
        }
        if (i == 8)
           break;
    }

    led_on();
    for(i=0;i<16;i++)
    {
         temp = harita[i][0];
         for(j=1;j<16;j++)
           temp = temp*4 + harita[i][j];
           addr_ptr = ADDR_HARITA_0;
           *addr_ptr = temp;
           adres = adres + 4;
    }
```

```
  addr_ptr = ADDR_KONTROL;
  *addr_ptr = 0x1;

  addr_ptr = ADDR_DURUM;
  while(*addr_ptr!=0x1)
      addr_ptr = ADDR_DURUM;
   addr_prt =   ADDR_YOL_KOORDINAT;
   // IP Çekirdeğinden gelen Yol Koordinatları RS_232 Portuna Yollanıyor
   // ---------------------------------------------------------------------------------
   xil_printf("Yol Koordinatları = %d", addr_ptr);

}

led_ac()
{   /* LED'leri yak*/
  int *addr_ptr;
  addr_ptr = ADDR_LED_AC;
  *addr_ptr = 0x00000000;
}

led_kapat()
{    /* LED'leri söndür */
  int *addr_ptr;
  addr_ptr = ADDR_LED_KAPAT;
  *addr_ptr = 0x00000000;
}
```

KİŞİSEL YAYIN VE ESERLER

[1] **Tuncer A.**, Yildirim M., Dynamic path planning of mobile robots with improved genetic algorithm, *Computers & Electrical Engineering*, 2012, **38**(6), 1564-1572.

[2] **Tuncer A.**, Yildirim M., Erkan K., A Motion Planning System for Mobile Robots, *Advances in Electrical and Computer Engineering*, 2012, **12**(1), 57-62.

[3] **Tuncer A.**, Yildirim M., Web-Based Virtual Laboratory for Genetic Algorithms, *Selçuk Üniversitesi, Teknik-Online Dergi*, 2012, **11**(1), 19-31.

[4] **Tuncer A.**, Yildirim M., Erkan K., A Hybrid Implementation of Genetic Algorithm for Path Planning of Mobile Robots on FPGA, *The 27th International Symposium on Computer and Information Sciences (ISCIS 2012)*, Paris, France, 3-5 October 2012.

[5] **Tuncer A.**, Yildirim M., Chromosome Coding Methods in Genetic Algorithm for Path Planning of Mobile Robots, *The 26th International Symposium on Computer and Information Sciences (ISCIS 2011)*, London, England, 26-28 September 2011.

[6] **Tuncer A.**, Yıldırm M., Genetik Algoritmalar İçin Uzak Sanal Laboratuar, *Ulusal Teknik Eğitim, Mühendislik ve Eğitim Bilimleri Genç Araştırmacılar Sempozyumu*, Kocaeli, Türkiye, 20-22 Haziran 2007.

ÖZGEÇMİŞ

1980 yılında İstanbul'da dünyaya geldi. İlk ve orta öğrenimini İstanbul'da tamamladı. 1998 yılında Kocaeli Üniversitesi, Teknik Eğitim Fakültesi, Elektronik ve Bilgisayar Eğitimi, Bilgisayar Öğretmenliği Bölümü'nde eğitime başladı ve 2003 yılında mezun oldu. 2004-2007 yılları arasında, Kocaeli Üniversitesi, Fen Bilimleri Enstitüsü, Elektronik ve Bilgisayar Eğitimi Bölümü'nde yüksek lisans öğrenimini tamamladı. 2008 yılında aynı üniversite ve aynı bölümde doktora eğitimine başladı. 2009 yılından beri Kocaeli Üniversitesi, Elektronik ve Bilgisayar Eğitimi Bölümü'nde araştırma görevlisi olarak görev yapmaktadır. Genetik algoritmalar, robotlar için yol planlama, FPGA çalışma alanlarından bazılarıdır.