Gereğinden Çok Serbestlik Dereceli Robotların (Redundant Robots) Bir Ürün Olarak Geliştirilmesi

Proje No: 104M260

Doç. Dr. E. Şahin ÇONKUR

Yrd. Doç. Dr. Abdullah T. TOLA

MAYIS 2008

DENİZLİ

Önsöz

Gereğinden çok serbestlik dereceli robotlar, sahip oldukları fazla serbestlik derecelerini kullanarak standart robotlar için çok zor olan hareketleri yapabilen robotlardır. Bu tür robotlar, insanların ulaşması zor veya imkansız olduğu bölgelere girerek, el becerisi ve zeka gerektiren fakat insanların yapması zor ve tehlikeli olan birçok işi otomatik olarak yapabileceklerdir. Mekanikte ve kontrolde sahip oldukları problemler bu tür robotların uygulama alanına geçmesini engellemektedir. Bu projede gereğinden çok serbestlik dereceli robotlar için yazılım, kontrol ve mekanik dizayn alanlarında çalışmalar yapılmıştır. Serbest ve engellerle dolu bölgelerde verilen hedefe gidebilecek şekilde gerçek zamanda uzuv açılarını ayarlayabilen algoritmalar geliştirilmiş ve dizayn ve imal edilen gerçek robot üzerinde denenmiştir. Bu projede yaptığımız çalışmalar çok heyecanlı ve verimli olmuştur. Bize çok şey kazandırdığını söyleyebiliriz. Proje ekibi olarak projeye destek verdiği için TÜBİTAK'a çok teşekkür ediyoruz.

İçindekiler

Tablo ve Şekiller

Özet

Gereğinden çok serbestlik dereceli robotlar, sahip oldukları fazla serbestlik derecelerini kullanarak standart robotlar için çok zor olan hareketleri yapabilen robotlardır. Bu tür robotlar, insanların ulaşması zor veya imkansız olduğu bölgelere girerek, el becerisi ve zeka gerektiren fakat insanların yapması zor ve tehlikeli olan birçok işi otomatik olarak yapabileceklerdir. Bu gibi işleri başaran bir robot kolu, hem zaman ve para tasarrufu sağlayacak hem de insanları bu işleri yaparken karşılaşacakları tehlikelerden koruyacaktır. Fakat mekanikte ve kontrolde ortaya çıkan problemler bu tür robotların uygulama alanına geçmesini engellemektedir. Bu projede 2 boyutlu düzlemde çalışan 4 uzuvlu bir gereğinden çok serbestlik dereceli robotun dizaynı ve imalatı gerçekleştirilmiştir. Robotun kontrolü için RoboKol isimli Visual C# dilinde 3 boyutlu görüntüleme özelliğini de içeren kapsamlı bir program geliştirilmiştir. Bilgisayar programı engelli veya engelsiz bölgede potansiyel alanı kullanarak robotun uç noktası için gerekli yörüngeyi hesaplamaktadır. Robotun diğer uzuvları için gerekli açı değerleri geliştirdiğimiz algoritmalarla gerçek zamanlı olarak hesaplanmakta ve gerçek robota yollanarak robotun çalışması sağlanmaktadır. Engelsiz bölgelerde çalışan potansiyel alandan bağımsız orijinal bir hareket planlama algoritması geliştirilmiştir. Ayrıca, robotun fiziksel olarak geçebileceği en dar alanlardan geçmesini başararak robotun hedefine ulaşmasını sağlayan bir engellerden kaçınma algoritması geliştirilmiştir. Proje çalışmaları 2 boyutlu uzayda yoğunlaşmakla beraber, kontrol algoritmasının engelsiz ortamda çalışan 3 boyutlu versiyonu da geliştirilmiş ve bundan sonraki çalışmaların altyapısı hazırlanmıştır.

Anahtar Sözcükler: Gereğinden çok serbestlik dereceli robotlar, hareket planlaması, potansiyel alanlar

Abstract

Redundant robots are the robots that are able to perform the motion that is quite difficult for standard robots using extra degrees of freedom they have. This kind of robots will be able to achieve various tasks automatically that require man-equivalent capabilities by entering areas that are difficult or impossible to enter for human beings. A robotic arm that is able to achieve such tasks not only saves time and money but also protects human beings from the dangers they face while performing such tasks. However, difficulties encountered in mechanics and control prevent redundant robots from being applied. In this project, a redundant robot design and production were carried out in 2 dimensions with 4 links. For the robot control, a comprehensive computer program called RoboKOL written in Visual C# including 3D capabilities was developed. The computer program calculates the trajectory for the tip of the robot using potential fields. All the angle values required for the other robot links are calculated by the algorithms we have newly developed and the real robot is made work by sending these values to the robot. An original motion planning algorithms that works in the free space and is independent from the potential field has been developed. In addition, an obstacle avoiding algorithm has been developed, which let the robot reach its goal by directing the robot through the narrowest space which is the robot can pass physically. Although the project work was mainly focused on 2D space, the 3D version of the control algorithms that work in free space was also developed and a framework for future studies was established.

Keywords: Redundant robots, path planning, potential fields

1 Giriş

1.1 Literatür Özeti

Gereğinden çok serbestlik dereceli robot kolları, kendi uzuv değişkenlerine sonsuz sayıda çözüm üretebilen robot kolları olarak tanımlanır. Bu robot kolları iç ve dış engeller ortaya çıktığında değişik konfigürasyonlar seçerek bu engeller arasından geçebilirler[1]. Başlıca kullanım alanları şu şekilde sıralanabilir[2,3]:

- Büyük makinelerin içlerine tamir ve bakım için girebileceklerdir.
- Serbestlik dereceleri sınırlı olan ve çalışma ortamları çok itinayla hazırlanması gereken günümüzdeki endüstriyel robotların hareket kabiliyetlerini çok fazla arttırabileceklerdir.
- Uzay istasyonu inşası gibi el becerisi, mikro-elektronik imalatı gibi vakum ortamı gerektiren çok çeşitli işleri yapabileceklerdir.
- Bazı beyin ameliyatlarında, cerrahın elle ulaşmasının çok zor olduğu beyin kısımlarına ulaşmada kullanılabileceklerdir.
- Depremde yıkık altında kalmış canlıların yer tespitini yapabilecek ve yıkıklar arasında kendi yolunu bulup ilerleyerek onlara ilk müdahaleyi yapabilecektir.

Bu alanlar çok daha fazla genişletilebilir. Bu tür robotlar, insanların ulaşması zor veya imkansız olduğu bölgelere girerek, el becerisi ve zeka gerektiren fakat insanların yapması zor ve tehlikeli olan birçok işi otomatik olarak yapabileceklerdir. Bu gibi işleri başaran bir robot kolu, hem zaman ve para tasarrufu sağlayacak hem de insanları bu işleri yaparken karşılaşacakları tehlikelerden koruyacaktır.

Çalışma alanında robotun uç noktasının yörüngesi verildiğinde geçerli bir mafsal yörüngesinin hesabına *gereğinden çok eklemli çözümleme* denir[4]. Bu çözümleme sınıfında, *gradyan izdüşümü tekniği* boş uzay ile çeşitli performans kriterleri uygulayarak robot uzuvlarının kendi iç hareketini belirler[5]. *Genişletilmiş Jacobian tekniği*, mafsal uzayı ile görev uzayı arasındaki ilişkiyi tanımlanan ek sınırlamaları kullanarak tam belirli olmayan bir sistemi belirli bir sistem haline dönüştürür[6]. Ek kinematik sınırlamalar engelden kaçınma için de tanımlanabilir[7]. Gereğinden çok eklemli çözümleme çevredeki değişikliklere çok çabuk tepki verebilir, fakat yörünge planlaması bakımından yeterliliği tartışılır. Bu teknikler esas olarak yerel tekniklerdir, yani üretilen çözümler istenen hareket alanı dar olduğunda geçerli olur.

Fakat az sayıda olmakla beraber gereğinden çok eklemli çözümlemeyi global olarak kullanan teknikler de vardır[8].

Eğer görev uç noktanın belirli bir hedef noktaya ulaşması olarak verilirse, robot mafsallarının yörünge hesaplaması *yörünge planlama problemi* olarak isimlendirilir ve *hareket planlaması* içinde değerlendirilir[9]. Geometrik hareket planlama algoritmaları robotun tamamı için engellerle çarpışmayan yörüngeler hesaplayabilir. Birçok hareket planlama algoritması arasında öne çıkan genel yaklaşımlar *yol haritaları, hücre ayrıştırma ve potansiyel alan* metotlarıdır. Bu yaklaşımlar hem çalışma uzayında hem de konfigürasyon uzayında uygulanabilir. Çalışma uzayı robotun içinde hareket ettiği üç boyutlu uzayı temsil ederken, konfigürasyon uzayı robotun mümkün olan bütün konfigürasyonlarını temsil eder. Robotun çalışma uzayındaki yörünge planlaması konfigürasyon uzayında bir noktanın yörünge planlamasına indirgenir[10]. Yol haritaları çalışma alanının serbest bölgeleri arasındaki bağlantıları tek boyutlu doğrular setine indirger[11]. Yol haritaları, görünürlük grafikleri, Voronoi diyagramları ve serbest yol ağları ile oluşturulur[12]. Bu yöntemin önemli bir dezavantajı verimsizliğe sebep olan çok sayıda düğüm içerebilmesidir. Hücre ayrıştırma metodu serbest bölgeleri hücrelere ayırır ve hücreler arasındaki bitişiklikleri temsil eden bağlantı grafiğini oluşturur. Bu grafik daha sonra hedef noktası ile başlangıç noktasını bağlayan birbirine bitişik bir hücre grubu bulmak için taranır[13].

Potansiyel alan metodunda, çalışma alanı suni bir potansiyel alanın etkisi altında tutulur. Engeller itme etkisi verirken hedef noktası çekme etkisi oluşturur. Bu iki etkinin negatif gradyanının toplamı, robot uzuvlarındaki kontrol noktaları vasıtasıyla robot hareketinin kontrolünde kullanılır[14]. Potansiyel alan metodunda en büyük problem, robot hedefe varmadan önce yerel minimumlardan birinde takılıp kalmasıdır[15]. Bu soruna değişik çözümler düşünülmüştür. Bunlardan biri yerel minimumları arayıp bularak devre dışı bırakmaktır[16]. Bir diğeri de yerel minimumları olmayan potansiyel alanlar oluşturmaktır[17]. Potansiyel alan metodu gerçek zamanlı yerel uygulamalarda kullanılabilir[18]. Fakat engellerin sayısı arttığında engele çok yaklaşmak imkansız hale gelebilmektedir[19]. Potansiyel alan metodu global olarak da uygulanabilir. Bu, sayısal potansiyel alanların yerel minimumsuz olarak bir ızgara üzerinde tanımlanması ile olur[18].

Global bir yörünge için iyi bilinen yöntemleri bir şekilde kullanan bazı algoritmalar vardır. Fakat bu algoritmalar uç nokta yerine robot uzuvlarının yörünge planlamasını yapar ve bunun için farklı yöntemler kullanır. Örneğin, sensör verisi kullanarak ve robot uzuvlarının engellere hafif dokunmasına izin vererek robotun kinematik kontrolü başarılmıştır[20]. Robot konfigürasyonunu bir omurga eğrisine uygun hale getiren bir kontrol modeli

oluşturulmuştur[21]. Belirli bir eğriye robot konfigürasyonunu uygun hale getirmek için yeni kinematik denklemler geliştirilmiştir[22]. Yörünge planlaması penaltı fonksiyonlarını içeren bir dizi minimizasyon problemi olarak incelenmiştir[23]. Sonsuz derecede esnek robot kontrolü, Catmull-Rom eğrileri[24] veya elipsoidler[25] kullanılarak kontrol edilmiştir. Konfigürasyon uzayının hesap zorluklarını hafifletmek için boyutu konfigürasyon uzayından daha az olan duruş uzayı tasarlanmıştır[26]. Gereğinden çok serbestlik dereceli bir robot dizayn edilmiş[27] ve duruş uzayı kullanılarak kontrol edilmiştir[28].

Gereğinden çok serbestlik dereceli robot kollarının mekanik dizaynı ile ilgili literatürde az sayıda yayın vardır. Mekanik dizayn genelde üç kategoride incelenir. Birincisi uzuvları birbirine eklenerek oluşturulan seri uzuv kollardır. İkincisi uzuvlar yerine örneğin hidrolik silindirler kullanılarak şekli değiştirilebilen yapılardan oluşan robotlardır. Üçüncüsü ise ardı ardına eklenen modüllerden oluşan robotlardır[29]. Ayrıca, endoskop gibi tamamen esnek yapılar da vardır[24].

1.2 Projede Yapılan Çalışmalar

Bu projede gereğinden çok serbestlik dereceli robotlar için teorik ve pratik çalışmalar yapılmıştır.

Bölüm 2'de 2 boyutlu düzlemde çalışan 4 serbestlik dereceli bir gereğinden çok serbestlik dereceli robot dizaynı ve imalatı anlatılmıştır. Aynı bölümde 3 boyutta çalışan bir robotun dizayn ve imalatı da anlatılmıştır.

Bölüm 3'de robotun kontrol sisteminin kurulması ve geliştirilmesi ile ilgili konular tartışılmıştır. Servo motor kontrolü için geliştirilen program açıklanmıştır. Sensör bilgilerinin bilgisayara alınıp işletilmesi konusu da bu bölümde işlenmiştir.

Bölüm 4'de robotun kontrolünde ve algoritma geliştirilmesinde kullanılan detaylı ve yoğun çalışmalar sonucu geliştirilmiş olan RoboKol programına yer verilmiştir. RoboKol Programı Windows tabanlı bir programda olması gereken hemen her şeye sahip bir programdır. İstenilen sayıda ve uzuv uzunluğuna sahip robotlar ekrana çizilebilir. Daha sonra istendiğinde robotun bazı özellikleri değiştirilebilir. Çalışma alanına engeller çizilebilir. Alanda zoom'lama yapılabilir. Robot hardisk'e kaydedilebilir. Bunun gibi daha birçok özelliğe sahiptir.

Bölüm 5'de RoboKol programına direct X kullanılarak eklenen 3 boyutlu görüntülemesi anlatılmıştır. İstenildiği sayıda pencerede mouse ile istenildiği açıdan bakılabilen çok gelişmiş

bir 3 boyutlu görüntülemeye sahip olunmuştur. Ayrıca bu bölümde 2 ve 3 boyutlu uzayda çalışan potansiyel alan ile ilgili çalışmalar özetlenmiştir.

Bölüm 6'da "serbest bölgede" geliştirilen algoritmalar anlatılmıştır. Bu proje kapsamında geliştirdiğimiz orijinal ve son derece verimli olan bir ters kinematik algoritması sunulmuştur. Algoritmanın özellikleri arasında basitlik, gerçek zamanlı çalışabilme, eklenebilirlik, engeller olma durumunda engellerden kaçınmaya yardımcı olma, robotu oluşturan uzuv sayısından bağımsız çalışma gibi özellikler sayılabilir.

Bölüm 7'de "engellerden kaçınma" algoritmasının geliştirme aşamaları verilmiştir. Bu algoritma fiziksel olarak mümkün olduğu müddetçe robota engeller arasında manevra yaptırmakta ve engellere çarpmadan ilerlemesini sağlamaktadır.

Bölüm 8'de robotların çalıştırılması ve deneylerin yapılması anlatılmıştır. Yazılım, kontrol sistemi ve robotu içeren tüm sistem başarılı bir şekilde çalıştırılmış ve bilgisayarın ürettiği sonuçlar fiziksel ortama aktarılmıştır. Örnek bir uygulama olarak robota gazaltı kaynağının yaptırılması da bu bölümde anlatılmıştır.

Bölüm 9'da projenin başarı kriterleri ile ilgili olarak genel bir değerlendirme yapılırken, Bölüm 10'da ise sonuç kısmı yer almış ve gelecekle ilgili çalışmalar tartışılmıştır.

2 Gereğinden Çok Serbestlik Dereceli Robot Dizaynı ve İmalatı

2.1 İki Boyutlu Uzayda Dört Serbestlik Dereceli Seri Gereğinden Çok Serbestlik Dereceli Bir Robot Dizaynı

Mekanik dizayn ile ilgili literatür araştırması yapılmış ve konu ile ilgili internette bulunan bütün yayın, animasyon, resim, video vb. kaynakları bir araya getirilmiştir. Bu araştırmalardan çıkarılan sonuçları sıralarsak, bunlar;

- Mekanik dizaynla ilgili müstakil yayın sayısı çok azdır. Yayınlar genel olarak kontrol algoritmalarını içermekte ve mekanik dizayn ikincil derecede yer almaktadır.
- Hareket iletiminde birçok durumda motorlar mafsalların üzerine monte edilmiştir. İletimin kablolarla ve kayış-kasnak mekanizmalarıyla yapıldığı dizaynlar da vardır.
- Kablolarla ve kayış-kasnak mekanizmalarıyla olan dizaynlarda çözümünü bilmediğimiz problemlerle karşılaştık. En pratik olarak tasarlanabilecek mekanizmaların kayış-kasnak mekanizmaları olduğunu gördük. Fakat ne yazık ki bu tür mekanizmalar üç boyutlu dizayna daha az imkan vermektedirler.
- İmkanlar nispetinde bir tane robot değil birden çok robot tasarlamamız ve bunları deneyerek sonuçları görmemiz gerektiği sonucuna vardık.

Yapılacak tasarım için tarafımızdan aşağıda verilen bazı kriterler belirlenmiştir.

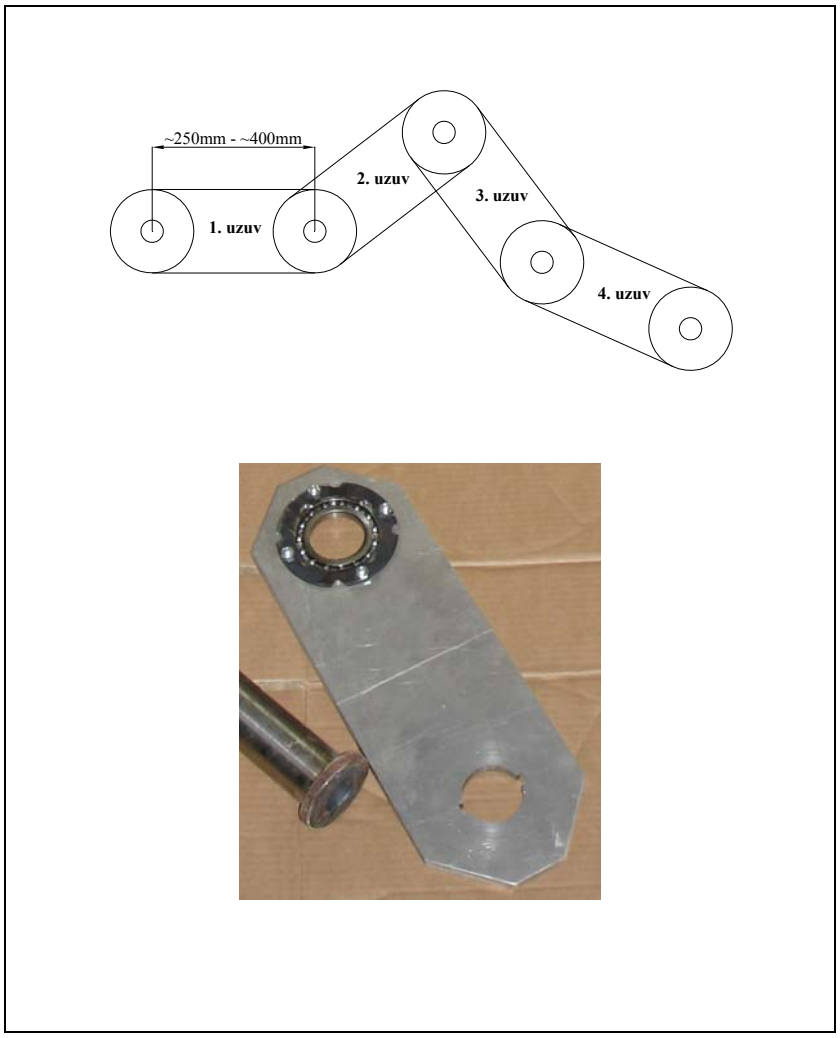
- Robotun serbestlik derecesi
- Robot uzuvlarının boyu
- Tahrik sistemi
- Parçaların üretim şekli
- Parçaların üretim yeri
- Robotun zeminde duruşu
- Kolların kaldırabileceği ağırlık
- Emniyetin sağlanması
- Robotun bilgisayar animasyonunun yapılması

2.1.1 Robotun Serbestlik Derecesi

Tasarlanacak robotun serbestlik derecesinin 2,4 veya 8 olması tartışılmıştır. Bilgisayara takılan hareket kontrol kartı ile 8 adet motor aynı anda kontrol edilebilmektedir. Böylelikle ilk aşamada yapılabilecek maksimum serbestlik derece sayısı yani robot uzuv sayısı 8 olabilmektedir. İlk dizaynda 8 kollu bir imalat yapmanın dizayn hataları, çıkabilecek sorunlar ve maliyeti bakımından tercih edilmemesi hususunda öne çıkmıştır. Daha sonra 2 ve 4 kollu robot üzerinde durulmuştur. Çıkabilecek problemlere karşı aşırı güvenli çalışma düşünülmüş ve robotun ilk denemesinin 2 kollu olması önerilmiştir. Fakat 2 kollu bir robotta gereğinden çok serbestlik dereceli özelliği bulunmadığından sonuç olarak ilk tasarımın 4 kollu yapılmasına karar verilmiştir. 4 kollu bir robot hem gereğinden çok serbestlik dereceli özelliği taşımaktadır hem de karşılaşılabilecek zorlukların görülüp fazla zaman ve para kaybetmeden düzeltilebileceği kadar az eklemlidir. Mekanizma X-Y düzleminde çalışacaktır.

2.1.2 Robotun Uzuvlarının Boyu

Kullanacağımız servo motorlar 2.2 Nm tork verebilmekte ve 3000 rpm hızda dönebilmektedir. Bundan dolayı herhangi bir redüksiyon yapılmadan mevcut tork çok kısa kollara izin verecektir. Ayrıca her ne kadar motor hızları ayarlanabiliyor olsa da herhangi bir hata anında ya da bilgisayarın kilitlenmesi karşısında motorlar kontrolsüz bir şekilde ve çok hızlı dönebilmektedirler. Eğer mekanizma kurulmuşken motorlar bir turdan daha fazla dönerse kollar birbirinin içinden geçmeğe çalışacaktır ve geçemeyeceğinden dolayı hasar oluşturacaktır. Bundan dolayı mekanizmada mutlaka redüktör kullanılması gerekmektedir. Robotun toplam boyunun 1000 mm ile 1600 mm arasında bir değer olması kararlaştırılmıştır. Her bir kol için de yaklaşık olarak 250 Nm ila 350 Nm'lik tork düşünülmüştür. Böylelikle her kol yaklaşık 25 kg ila 30 kg yükü kaldırabilecek şekilde olabilecektir. Düşünülen uzuv boyları ve kullanılabilecek triger kayış boyları göz önünde bulundurularak 360H tipindeki triger kayışının kullanılmasına karar verilmiştir. Böylelikle uzuvların mil delik merkezleri arası 254 mm olmuştur. Değerin 250 mm gibi yuvarlak bir değer alınamamasının nedeni kullanılan kayışın standart olarak 10 inç boyunda olmasıdır. (10 inç = 254 mm). Şekil 1'de 4 uzuvlu robot kolunun üstten görünüşü ve imal edilen uzvun bir parçasının resmi yer almaktadır. Resimde alüminyum uzvun kalınlığı 10 mm'dir; bir tarafına 45 mm'lik delik ve kama kanalı açılmıştır. Diğer tarafına ise iç çapı 45 mm dışı 75 mm olan ve kalınlığı 10 mm olan rulman çakılmıştır ve rulmanın sabit kalabilmesi için iki taraftan levhalarla tutturulmuştur.



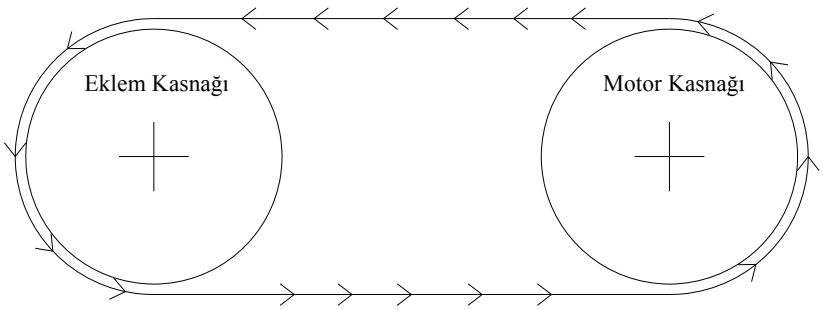
Şekil 1. Dört uzuvlu robot kolunun üstten görünüşü ve imal edilen uzvun bir parçası

2.1.3 Tahrik Sistemi

Tahrik sistemi için üç ana fikir ortaya çıkmıştır. Bu fikirler şöyle sıralanabilir:

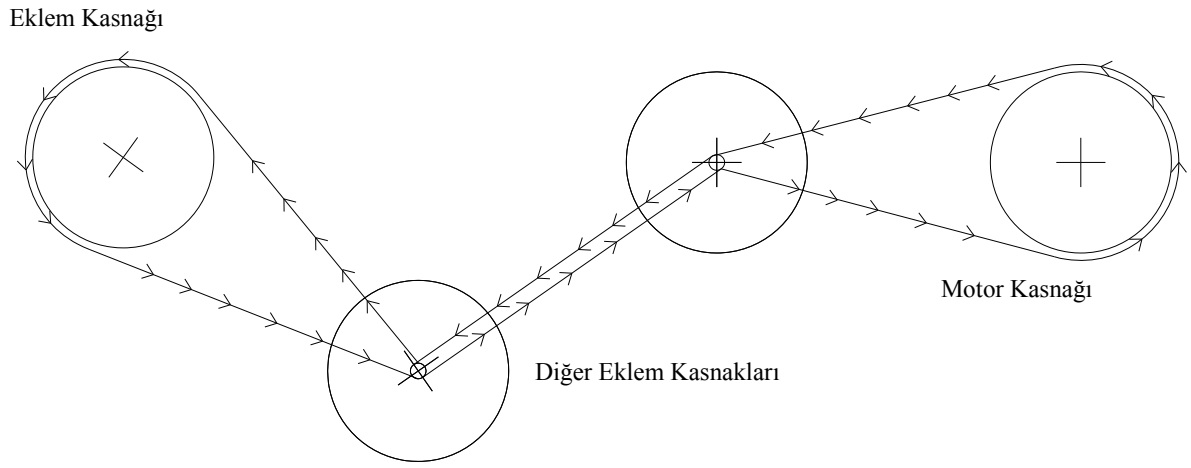
***Motorların uzuvların üzerine montaj yapılması**:* Motorların uzuvlara 90 dereceli redüktörler aracılığıyla bağlanması düşünülmüştür. Redüktörler motorlardan aldığı hareketi 90 derece döndürüp uzvu çevirecektir. Motorların her birinin ağırlığı yaklaşık olarak 3 kg olduğu için 4 motor mekanizmaya fazladan 12 kg getirecek ve kullanılacak redüktörlerin de kolların üzerinde yer alacağı için sistemi oldukça fazla ağırlaştıracaktır. Servo motorların eklemlerin üzerine konulabilmesi için uzuv uzunlukları motorların ağırlığına ve boyutlarına göre daha büyük olmalıdır. Ayrıca her motor için enerji ve enkoder kablosu olmak üzere 2 adet kablo bulunmaktadır. Böylelikle 4 motor için 8 adet kablo bütün mekanizmanın içinden geçirilmeli ve eklemler kıvrıldığında kablolarda meydana gelen kasılmalar sorun çıkarmamalıdır. Bu dizayn mecbur kalınmadıkça ilk tasarım için oldukça güç bulunmuştur.

Motorların kolların gerisine montajın yapılıp hareketin çelik halatla aktarılması: Çelik halatlar araştırıldı ve 2 mm, 3 mm gibi ince ve dayanıklı halatların piyasada kolayca ve ucuz bir şekilde bulunduğu görülmüştür. Prensip olarak motor bir şaseye bağlanıp miline de bir halat kasnağı takılacaktır. Robot uzuvlarının herhangi biri ele alındığında, bir kolda iki adet eklem yeri bulunmaktadır. Bu eklem yerlerinden birine yine halat kasnağı takılacaktır. Çelik halat motordaki kasnağa geçirilecek ve diğer tarafı da eklemdeki kasnağa geçirilecektir (Şekil 2). Böylelikle motor döndüğü zaman halat eklemdeki kasnağı da döndürecektir. Sonuç olarak hareket iletilmiş olacaktır.



Şekil 2. Kayış kasnaklı sistem

İlk eklem için halatın kasnaklara bağlanmasında bir problem oluşmayacağı tahmin edilmektedir. Fakat ikinci ve diğer eklemlere hareket iletilirken halat oldukça uzun bir yol kat edecektir. Burada çıkabilecek problemlerden biri halatın uzunluğundan dolayı oluşabilecek esnemelerdir. Bir diğer problem ise halatın kasnaklarda patinaj yapıp hareketi birebir iletememesidir. Ayrıca kollar kıvrılmaya başladığında halatlarda oluşan gerilmeler kıvrılma açısına göre değişmektedir. Kasıntıların teorik olarak hiç olmaması için halatların bir eklemi aşıp diğer ekleme hareket taşıması için takip edeceği yol, her eklemin merkezi olmalıdır. Eğer böyle olursa kasıntı minimum düzeyde oluşacaktır. Halatın izlediği yol Şekil 3'de görülmektedir.

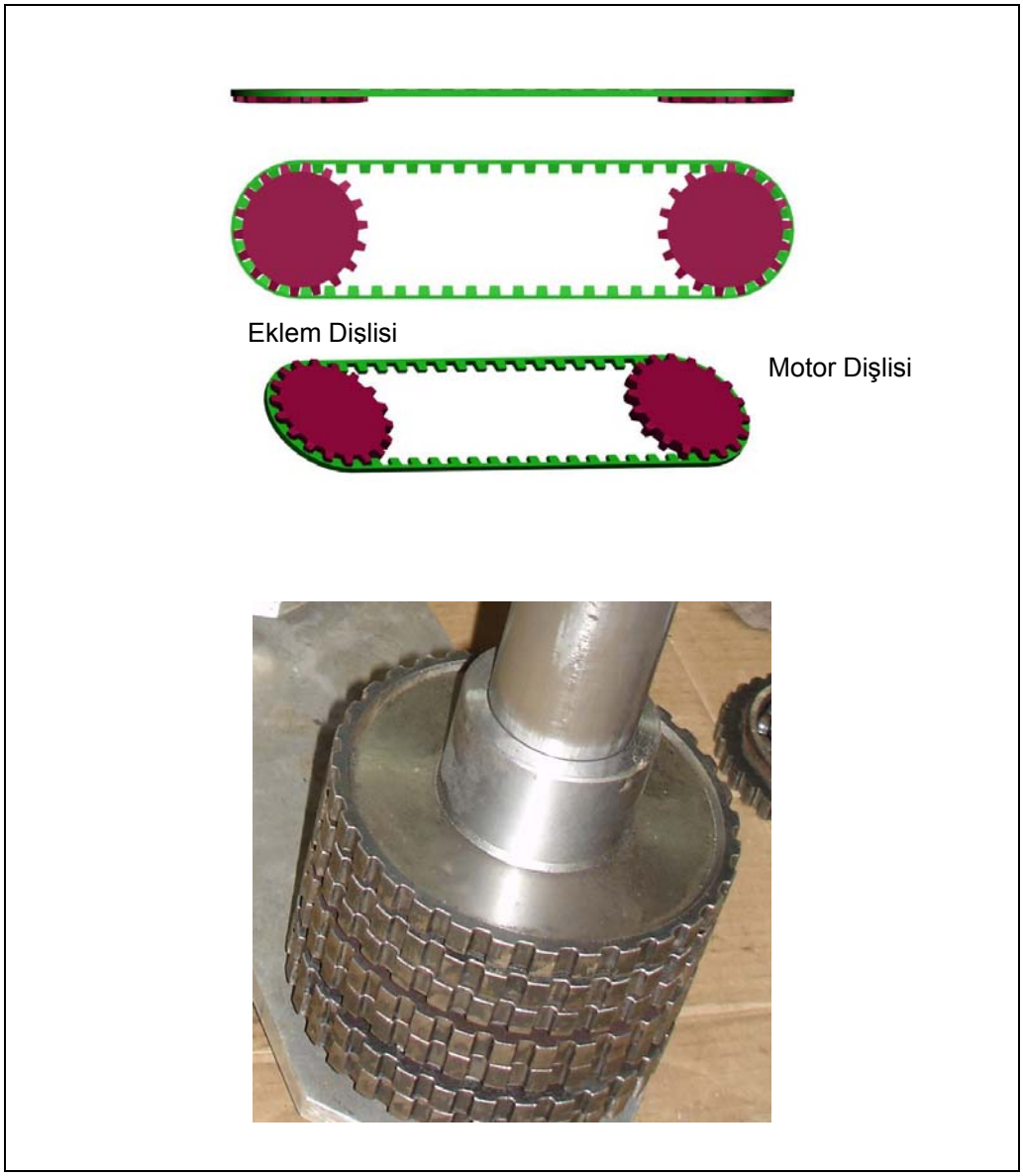


Şekil 3. Halatlı sistem

Şekildeki gibi halat merkezden geçirilebilmektedir, fakat halatın geçeceği yerin halatın boşa çıkmaması için çok dikkatlice dizayn edilmesi gereklidir. 4 eklemli bir dizaynda motorlara en yakın olan eklemden 8 adet çelik halat geçecektir. Bu 8 halatın en az kasıntıya yer verilerek geçirilmesi zorunludur. Yukarıdaki şekilde görülen en soldaki eklem kasnağından bir önceki kasnak yukarıya doğru döndüğünde altta kalan halat fazladan merkezdeki küçük dairenin etrafından dolanacaktır ve üstte kalan halat ise bir o kadar boşa çıkacaktır. Daireye sarılan halatta gerilme oluşacağı için bağlı bulunduğu kolu döndürecektir. Bu dönme istenmeyen bir dönmedir ve büyük bir problemdir. Bu dönmenin ne kadar olacağı hesaplanmalı ve bilgisayardaki program buna göre düzenlenmelidir.

Motorların kolların gerisine montajın yapılıp hareketin triger kayışıyla aktarılması: Triger kayışları ve standartları araştırılmış ve piyasada ucuz ve kolayca bulunabildikleri görülmüştür. Bunların halatlara göre avantajları ve dezavantajları vardır. İlk olarak halat patinaj yapabilmekte fakat triger kayışları dişli kayışlar olduğundan patinaj yapmamakta, hareketi birebir iletebilmektedirler. Diğer olumlu bir yönü ise halatta eklem yapmak zorunluluğu olduğu halde kayışta eklem yeri bulunmamasıdır. Triger kayışlarında kullanılan triger dişlileri sanayide azdırma tezgâhında çok kaliteli bir şekilde imal edilebilmektedir. Fakat az sayıda dişli sayısı için azdırma tezgâhında dişli açtırmak oldukça pahalıdır. Buna karşın lazer kesim tezgahı ile levha şeklindeki malzemeden dişli kesilmektedir. Lazer kesim tezgâhlarında parça işlettirmek çok ucuzdur. Fakat ne yazık ki lazer kesimle yapılan dişlilerin

yüzey kalitesi azdırma tezgahında açılan dişlilerin yüzey kalitelerine göre çok kötüdür. 1/10 mm hassasiyette kesim yapabilen lazer tezgahında deneme amaçlı 2 adet dişli kestirilmiştir. İncelemelerimizde ilk prototip robot için dişlilerin yüzey pürüzlülük değeri uygun görülmüş ve dişlilerin lazer kesim ile üretilebileceği kanısına varılmıştır. Burada hareketin aktarılması halattaki aktarmaya benzemektedir. Motorun ucuna ve ekleme birer adet dişli takılır. Böylelikle motor dişlisi döndüğünde kayış hareketlenecek ve eklemi de döndürecektir. Şekil 4'de bilgisayarda çizilmiş triger dişlileri ve lazer kesim tezgahında kesilmiş triger dişlileri görülmektedir. Dişlilerin alüminyum yapılması düşünülmüştür. Fakat Denizli'de bulunan tezgahlar en fazla 6 mm kalınlığındaki alüminyumu kesebildiğinden, başka bir şehirde tezgah araştırması yapmak yerine dişliler S235 kalite çelikten kestirilmiştir.



Şekil 4. Bilgisayarda çizilmiş triger dişlileri ve lazer kesim tezgahında kesilmiş triger dişlileri

Burada hareket ilk uzuv için problemsiz bir şekilde iletilebilmektedir. Hareketin ikinci ve diğer uzuvlara iletilebilmesi için de oldukça basit ve zahmetsiz bir yöntem bulunmuştur. Hareket dişliden dişliye ve tekrar dişliden dişliye olmak üzere birinden diğerine taşınarak ulaşması gereken uzva kadar iletilmektedir (Şekil 5). Motor dişlisi motor miline sabitlenecektir. Aradaki dişliler rulmanlar üzerinde avare olarak dönecek ve sadece hareketi ileteceklerdir. Son olarak da eklemi döndürecek dişli o eklemin miline sabitlenecektir. Aradaki hareketi taşıyan dişliler iki kayışın da takılabileceği kadar geniş imal edilmelidir.

Bu üç tahrik sistemi göz önüne alındığında proje için en verimli sistemin “triger kayışlarıyla yapılan tahrik sistemi” olduğu kararına varılmıştır.

Şekil 5. Hareketin bir eksenden diğer eksene kayışla geçmesi

2.1.4 Parçaların Üretim Şekli

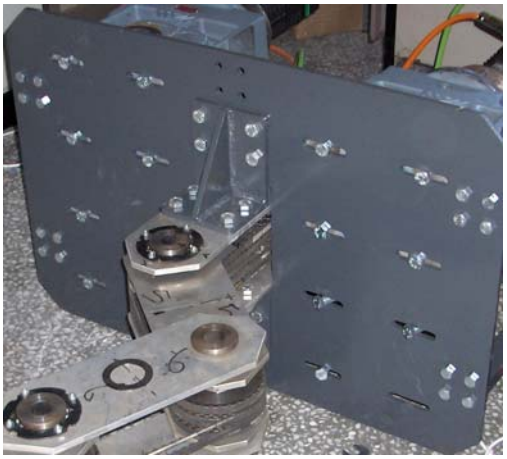
Triger kayışlarının boy standartları ve kalınlık standartları mevcuttur. Uzuvların boylarına göre uygun uzunlukta kayış seçimi yapıldı. Fakat kalınlık seçimi standart yelpazesinin dar olmasından dolayı yapılamamaktadır. Uygun kalınlıkta kayış elde etmek için triger kayışı boydan boya istenilen kalınlıkta kesilmiştir. Yapılan araştırma sonucunda bir triger kayışı boydan boya 5 mm kalınlığında bölünebileceği görülmüştür. Yapılan bir deney sonrası 5 mm’lik bir triger kayışının 160 kg yükü emniyetli bir şekilde kaldırabildiği görülmüştür. Fakat robotu çalıştırırken sistem hatası yüzünden uzuvlardan birisi kendi gövdesine dayandığı halde motor dönmeye devam etmiştir ve kayış kopmamış fakat bizim aklımıza gelmeyen bir olay olmuştur; kayışın dişleri sıyrılmıştır. Bu da yapılan deneyin gerçek hasar tipini tam olarak canlandırmadığı anlamına gelmektedir. Triger dişlileri yukarıda da açıklandığı gibi lazer kesim ile üretilmiştir. Kolların gövdeleri için alüminyum levhalar kullanılmıştır. Levhalar

dikdörtgen şeklinde kestirilerek satın alınmıştır. Daha sonra planya tezgahında kenarları, freze tezgahında delikleri ve tekrar planya tezgahında kama kanalları açılmıştır. Miller imalat çeliğinden belli ölçüde kestirilmiş, torna tezgahında çapı 45 mm ye tornalanmış, taşlama tezgahında toleranslı şekilde işlenmiş ve freze tezgahında üzerine boydan boya kama kanalı açılmıştır. Şekil 6'da sistemde kullanılan taşlanmış bir mil görülmektedir.

Şekil 6. Taşlanmış mil

Şase olarak S235 10 mm'lik saçtan alevli kesim ile üretilmiştir (Şekil 7). Şase levhasının ortasındaki dikdörtgen şeklindeki delikten robot kolunun ilk eklemi geçmekte ve tutturma parçasıyla tutturulmaktadır. Ayrıca üzerindeki delikler yardımıyla arka tarafına redüktörler tutturulmaktadır. Yine delikler sayesinde profil ayak şaseye bağlanıp şasenin dik bir şekilde ayakta durmasını sağlamaktadır.

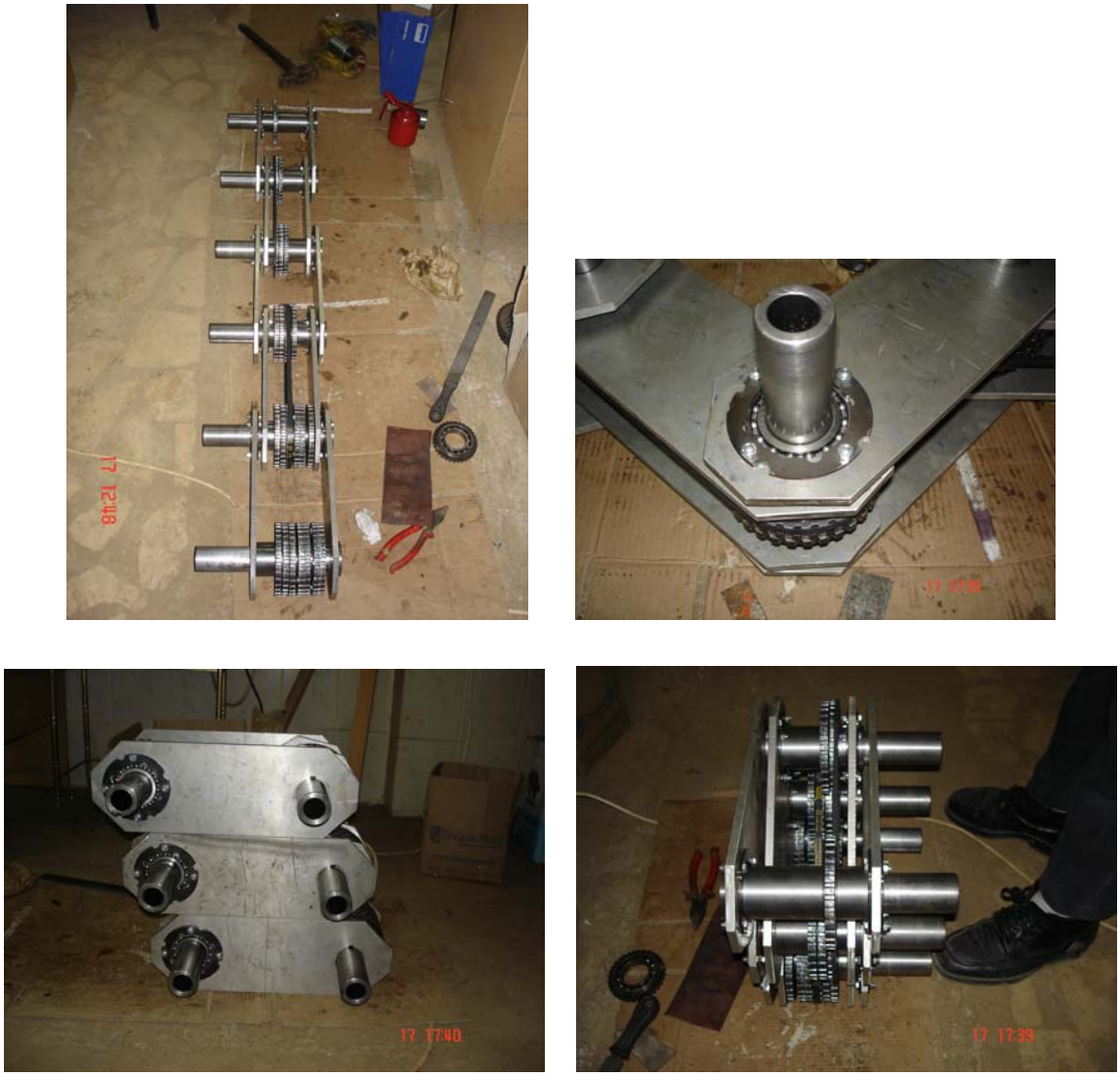
Şase ayakları ise profil çubukların kaynaklı konstrüksiyonu ile yapılmıştır. Diğer malzemelerden olan rulmanlar, cıvatalar, somunlar, segmanlar vb. gereçler standart olduğundan dolayı üretilmeyip hazır olarak alınıp kullanılmıştır. Redüktörler de hazır olarak alınmıştır. Redüktörlerin çevrim oranları i=142 dir.



Şekil 7. Şase levhası ve kolun tutturulması

2.1.5 Parçaların Üretim Yeri

Parçaların tamamının tek bir yerde üretilmesi zordur. Çünkü üretilecek parçaların üretim tezgâhlarının hepsi bir firmada genelde bulunmamaktadır. Bulunan firmalar ise seri üretim yaptıkları için prototip gibi az kazanç sağlayan ve çok zahmetli bir üretim işini kabul etmemektedirler. Bundan dolayı sanayide lazer kesimi bir firmaya, gövdeyi başka bir firmaya ve diğer parçalar için de başka yerler bulmak zorunda kalınmıştır. Ayrıca bütün bu parçalar tek tek başka yerlerde imal edildiğinde bunları bir araya getirip montaj yapılacak bir yer gerekmektedir. Montaj yapılacak yerin de atölye olması zorunludur. Çünkü montajda çıkabilecek sorunlar atölyede giderilmeli, işlenen parçalar gerekirse tekrar ikinci bir işleme tabi tutulmalıdır. Şase imalatı ve montaj atölyesi olarak Doğan İş Makineleri firması kullanılmıştır. Diğer parçalar ise kalite ve fiyat araştırması yapılarak sanayideki diğer firmalara yaptırılmıştır. Tüm imalat süreci yaklaşık olarak 3 ay sürmüştür. Üretimde çıkan sorunların başında üretimi yapılacak parçaların az sayıda olması ve az kazanç sağlaması nedeniyle işletmecinin ilgisizliği ve işin yavaş yürümesidir. İşletmeciler küçük bir parçayla uğraşırken gelen diğer büyük kazançlı iş karşısında küçük parçanın imalatını bırakmakta ve önce büyük işi bitirip küçük işe tekrar dönmektedir. Bu imalatın yavaş yürümesine, moral bozukluğuna, konsantrenin dağılmasına ve zaman kaybına yol açmıştır. Tüm bu olumsuzluklara karşın yapılan robotun bir ilk olacağı fikri her gün sabah yepyeni bir hevesle imalatın başına geçmeye yetmiştir. Aşağıda Şekil 8'de robotun üretim aşamalarında çekilen resimlerinden örnekler yer almaktadır.



Şekil 8. Robotun üretim aşamalarında çekilen resimlerinden örnekler

2.1.6 Robotun Zeminde Duruşu

Robot kolu X-Y düzleminde iki boyutlu olarak çalışacaktır. Robot, yer düzlemine paralel oluşuna veya dik oluşuna göre iki çeşit düzlemde çalıştırılabilir. Paralel olarak çalıştırıldığında kayışlara yer çekiminden dolayı hiç yük gelmeyecektir. Kol dik olarak çalıştırıldığında kayışlara yer çekiminden dolayı oldukça fazla yük binecektir. Bu da ilk deneme için fazladan problem demektir. Bundan dolayı Şekil 11'de görüldüğü gibi robotun yer düzlemine paralel olarak çalıştırılması kararlaştırılmıştır.

2.1.7 Kolların Kaldırabileceği Ağırlık

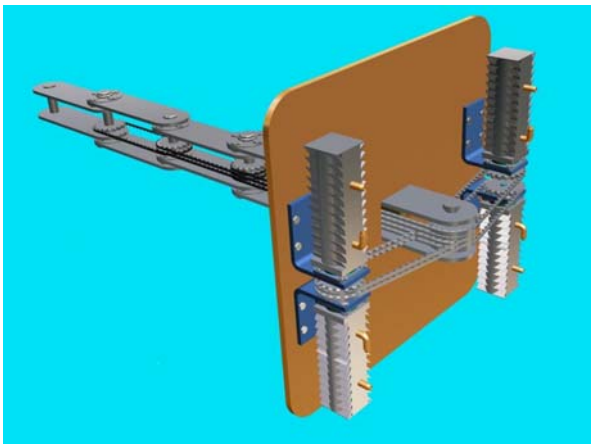
Paralel çalışmada kayışlara yer çekiminden dolayı yük binmeyeceği için kaldırılabilecek ağırlık mekanik dizayna bağlı olmaktadır ve yapılacak tasarım deney amaçlı 25 kg ila 35 kg'lık yükleri kolaylıkla kaldırabilmektedir. Fakat kol dik bir şekilde çalıştırıldığında yukarıdaki ağırlıklarda çalışılabilmesi için kayışlar yeterli dayanırlıkta olmalıdır, dişli çapları moment hesapları yapılarak optimum çap bulunmalıdır ve 2.2 Nm'lik servo motorların torkları redüksiyon ile yeterli derecede artırılmalıdır.

2.1.8 Emniyetin Sağlanması

Mekanizmanın istenmeyen bir hareket yapıp insanlara, kendine, motorlara ya da çevresindeki cihazlara zarar vermemesi gerekir. Bundan kaçınmak için her halükarda robot acil stop edilebilmelidir. Bunun için elektrik panosuna acil stop butonu yerleştirilmiştir. Ayrıca redüktör konularak kolların hareket hızının mekanik olarak da yavaşlatılması konusunda ortak karara varılmıştır.

2.1.9 Robotun Bilgisayar Animasyonunun Yapılması

İmalattaki hataları en aza indirmek ve yapılacak işlerin tam bir planlamasını yapabilmek için robotun komple dizaynı bilgisayar üzerinde tasarlanmış ve çıkabilecek birçok problem önceden giderilmiştir. Ayrıca bilgisayarda çizilmiş bir robotun imal edilip gerçek hayata geçişi, bizler için eşsiz bir deneyim olmuştur. Bu rapora ek olarak verilen CD'de Şekil 9'da bir resmi verilen animasyonun **104M260_ESahin_CONKUR_2DRobotAnim.wmv** isimli videosu yer almaktadır.



Şekil 9. Robotun animasyonundan bir görüntü

2.1.10 Üretim Planlaması

Planlama yapılırken ilk önce yapılan tasarımdan yola çıkılarak hangi parçaların üretileceği, hangilerinin satın alınacağı, hangi parçadan başlanacağı gibi bazı ana başlıklar belirlenmiştir. Buna göre hangi parçaların üretileceği ortaya çıkmıştır.

Üretilecek parçalar: Triger dişlileri
Kol levhaları
Miller
Rulman kapakları
Şase levhası
Şase ayağı
Burçlar
Kol destek parçası
Kamalar

Satın alınacak parçalar:
Cıvatalar
Rulmanlar
Redüktörler
Triger kayışları
Mil somunları

Robotu çalışır hale getirebilmek için küçük parçalar halinde çok sayıda çalışma yapılmış ve proje ekibi hep beraber bu çalışmaları gerçekleştirmiştir. Sonuç olarak ortaya çıkan robotun resimleri Şekil 10 ve Şekil 11'de görülmektedir.

Şekil 10. RoboKol'un üstten görünüşü

Şekil 11. RoboKol'un yandan görünüşü

2.1.11 Robotun Hassasiyeti

Robotta aktarma sistemi olarak triger kayışları kullanılmıştır. Triger kayışları hemen hemen bütün otomobillerde kullanılmaktadır ve hareketi birebir aktarmaktadır. İmal ettiğimiz robotta da hareketi birebir aktarmıştır ancak kayışlarda gerdirme kasnakları kullanılmadığı için kol zorlandığı zaman esnemeler meydana gelmektedir. Hassasiyeti ölçmek için ilk başta robot boşta çalıştırılmış ve hassasiyetin 4 mm olduğu gözlenmiştir. Fakat kol bir yöne gidip tekrar

geldiğinde 4 mm olmaktadır. Önce bir birim sağa sonra iki birim sola daha sonra bir birim sağa döndürüldüğü zaman hassasiyet 0,1mm olmaktadır. Tek yönde giderken hassasiyet hesaplanırsa, servo motor puls ve redüktör çevrim oranı çarpımı,

16384*142=2326528

olur.

Bu hesapla robot kolunun her bir uzvu 360 derecenin 1/2326528'i kadar parçalar halinde dönebilmektedir. Bu hassasiyet çıplak gözle gözlenememektedir, ancak çok yavaş ve uzun süre döndürme işlemlerinde motorun döndüğü görülmektedir.

Bu ölçüm farklarının nedeni hemen tespit edilmiştir. Bu da redüktörlerdeki boşluklardır. Geçici bir çözüm olarak her redüktörün üzerine bir kasnak imal edilmiş ve her kasnağa uzun yaylar takılmıştır. Böylelikle redüktördeki boşluk bir yana dayandırılmıştır. Yaylar takıldıktan sonraki hassasiyet 0,1 mm olmuştur. Ancak kolun hareketi elle zorla engellendiğinde yaylar esnemekte ve hassasiyet yine 4 mm olmaktadır. Bunun yanında kol elle biraz daha zorlandığında kayışlarda da bir esneme meydana gelmekte ve hassasiyet 6-10mm ye çıkabilmektedir. Kayışların ve yayların esnemesi robot kolunun hızlı hareket etmesine de engel olmuştur. Çünkü ani kalkışlarda ve duruşlarda esnemelerden dolayı hassasiyet bozulmaktadır. 6. Bölümde bahsedileceği gibi, robot kolu kararlı olarak çalıştırıldıktan sonra kaynak makinesiyle kaynak yapma fikri ortaya çıkmıştır. Kola kaynak torcunun bağlanması ve torç kablosunun 5 cm ye yakın kalınlıkta olması kolun hareketlerine fazladan yük getirmiş ve bir miktar esnemeler meydana gelmesine sebep olmuştur. Kaynak yaparken çalışma hassasiyeti 1-2 mm arasında değişmektedir.

2.2 Üç Boyutlu Dizayn Ve İmalat

3 boyutlu bir robotta en büyük problem robotun ve iş parçasının ağırlığıdır. Çünkü ağırlık arttıkça motorlara fazladan yük binmektedir. Yatay düzlemde hareket eden robot kolu ile dikey düzlemde hareket eden bir robot kolunun motorlarına gelen yük arasında büyük bir fark vardır. Hareketin sağlanabilmesi için burada da redüktörler kullanılmalıdır. Her eklem iki eksende hareket edebildiğinden motorların uzuvların gerisine konulması ve hareketin kayışla aktarılması oldukça güçtür. Bundan dolayı en iyi çözüm motorlar ve redüktörler uzuvların üzerine konulmalıdır. Ancak motorların ve redüktörlerin de kendi ağırlığı vardır ve bu ağırlıklar uzuvların ağırlığı yanında küçümsenmeyecek kadar fazladır. Piyasada bulunan hassas redüktörler ile çalışabilecek bir dizayn oluşturulamamıştır. Robot kolunun her

uzvunun çalışabilmesi için bize i=250 çevrim oranlı redüktörler gerekmektedir. Arka arkaya redüktör bağlantısı düşünüldüğünde ise hem ağırlık artmaktadır hem de redüktörlerin tork kapasiteleri aşılmaktadır. Sanayideki robot uygulamalarında özel üretim yüksek tork dayanımlı, yüksek çevrim oranlı hassas redüktörler kullanılmaktadır. Ayrıca büyük uzuvların arkasına dengeleyici ağırlık koyulmakta ya da hidrolik veya pnömatik olarak dengeleyici sistemler konulmaktadır. Özel yapım redüktörler Türkiye'de satılmamaktadır.

Örnek olarak 4 uzuvlu ve 8 serbestlik dereceli bir robot düşünelim. 8 adet motor ve redüktör ile hareket sağlanacaktır. Her motor 30N ve her redüktör 55N ağırlığındadır. İki motor ve redüktör şaseye bağlanacak ve ilk uzvu hareket ettirecektir. Robot kolunun üzerinde toplam 6 adet redüktör ve 6 adet motor bulunmaktadır. Ayrıca robot kollarının ağırlığı da olacaktır. Her uzuv için 50N'luk ağırlık öngörülebilir. Bunların toplam ağırlığı (6*(30+55))+(4*50)=710N dur. Robot kolunun toplam uzunluğunu 1400mm alınırsa ve ilk uzvun redüktörüne gelen tork hesaplanırsa:

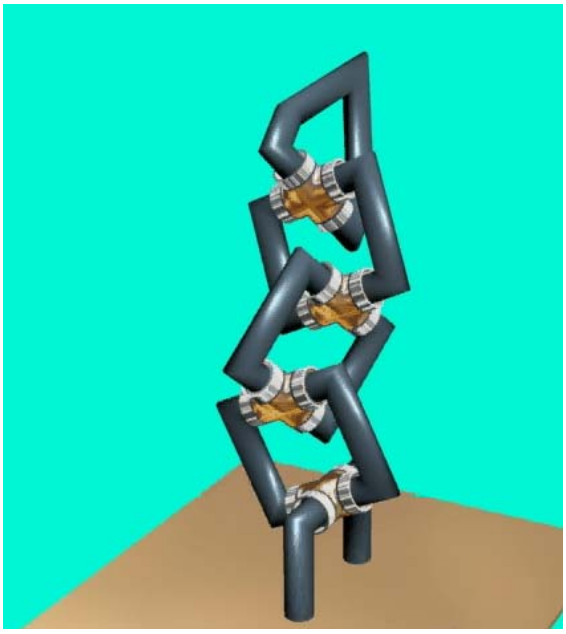
Tork=F*L'den

Tork=710*700=497000Nmm=497Nm olur.

Piyasada i=125 çevrim oranlı hassas redüktörler bulunmaktadır. 497Nm/125=3,976Nm'lik servo motorlara ihtiyaç vardır. Elimizde bulunan motorlar 2,2Nm'liktir. Aşağıdaki sebeplerden dolayı bu dizayndan vazgeçilmiştir. Elimizde bulunan malzemeyi kullanamadığımızdan, bütçemizde olmayan ve maliyeti yüksek olan redüktör ihtiyacı olduğundan ve Yüksek Lisans öğrencimiz İsmail Boztay'ın Yüksek Lisansını bitirdiğinden, bu proje kapsamında sanayi tipi bir robot çalışmasını bu noktada bitirmiş bulunuyoruz.

Şekil 12'de sanayi tipi robotta motorların nasıl yerleştirileceği düşünülürken yaptığımız ön animasyondan bir sahne görülmektedir. Robotun şekli çok gerçekçi değildir. Burada, motorların uzuvların içine yerleştirilmesi ve 90 derece redüktörlerle hareketin iletilmesi görülmek istenmiştir.

Ekte verilen CD'de **104M260_ESahin_CONKUR_3DRobotSimulasyonu.wmv** videosunda robotun görüntüsü seyredilebilir.



Şekil 12. Üç boyutlu dizayn için düşünülen animasyon

Fakat yukarıda bahsedilen problemler bu çalışmanın yapılmasını engelledi. Proje imkanları çerçevesinde, alternatif olarak “kablolarla” tahrik edilen 3 boyutlu bir robot yapılmıştır.

Bu tasarımda Şekil 13’de görüldüğü gibi 4 uzuvlu 3 boyutlu bir robot kolu imal edilmiştir. 4 adet servo ile her uzuv ayrı ayrı hareket edebilmektedir. Mekanik zorluklardan dolayı tek eklemde iki serbestlik derecesi kullanılmamıştır. İlk eklem x ekseninde sonraki y ekseninde, üçüncüsü yine x ekseninde ve sonuncusu da y ekseninde hareket yapmaktadır. Eksenler arası 150mm dir.

Robotun olabildiğince hafif olabilmesi için eksenler arası küçük seçilmiştir ve malzeme olarak polyamit ve derlin kullanılmıştır. Yataklama sistemi olarak rulman kullanılmamıştır. Uzvun ortasından geçen delik yatak olarak kullanılmıştır. Her uzvun milinin sonuna iki oluklu bir kasnak takılmıştır. Kansak döndüğünde bağlı uzuv da dönmektedir.

Motorlardaki hareket çelik halatlarla iletilmiştir. Her bir motorun milime M5 gijon takılmış gijona da pirinç malzemeden bir gezdirge takılmıştır. Lazer kesim ve kaynakla bir motor tablası imal edilmiştir. Motor döndüğünde gijon dönmekte ve gezdirge motor mili yönüne bağlı olarak ileri ve geri hareket etmektedir. Gezdirgeye bağlı olan çelik halatlar çekilmekte ve bırakılmaktadır.

Her motordan iki adet çelik halat çıkmaktadır. Her iki grup çelik halat her uzvun kasnağına sağına ve soluna vidalarla bağlanmaktadır. Böylelikle motorlar hareket ettiğinde çelik halatlar

yöne bağlı olarak çekilmekte ve uzuvlar hareket etmektedir. Hassasiyeti 5-10 mm arasındadır. Hassasiyetin arttırılması için yataklamalar kullanılmalıdır. Ancak çelik halatın kullanılmasında her zaman boşluklar olacaktır. Bu gibi bir sistem hassasiyetin çok gerekmediği yerlerde çok fazla kullanılabilir.

Ekte verilen CD'de **104M260_ESahin_CONKUR_3DRobotKablolu.wmv** videosunda robotun görüntüsü seyredilebilir.

Ekte verilen CD'de **104M260_ESahin_CONKUR_3DRobotKabloluMotorlar.wmv** videosunda Şekil 14'de bir görüntüsü verilen robotun motor bağlantı detayları seyredilebilir.

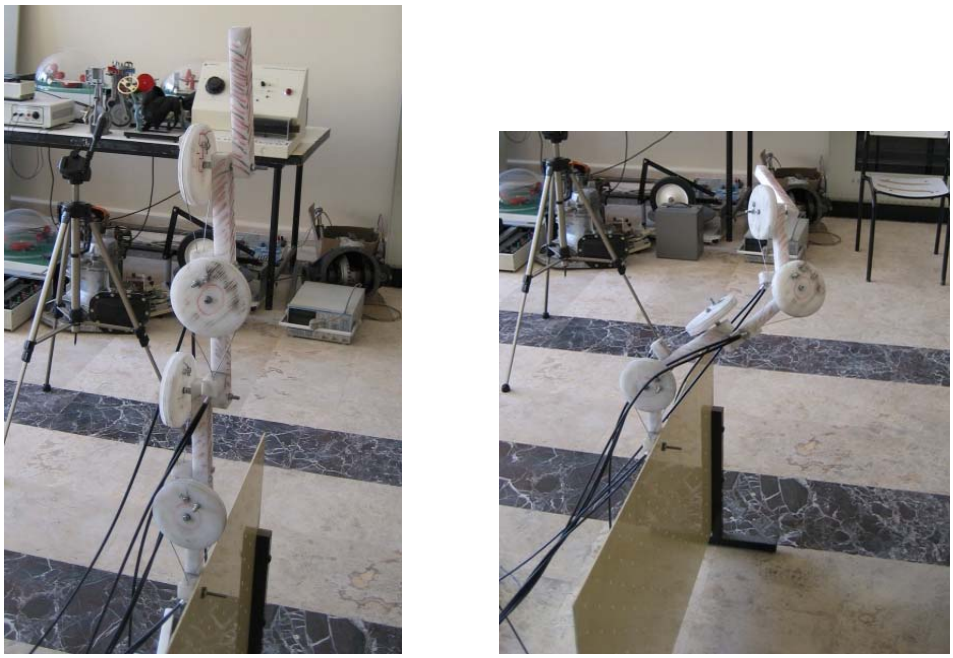
2.2.1 Üç Boyutlu Robotun Hassasiyeti

Yukarıda da bahsedildiği gibi, mevcut olan malzeme kullanılamadığından, bütçemizde olmayan ve maliyeti yüksek olan redüktör ihtiyacı olduğundan ve Yüksek Lisans öğrencimiz İsmail Boztay'ın Yüksek Lisansını bitirdiğinden, ortaya çıkan 3 boyutlu robot mekanik olarak eksikliklere sahiptir.

Fakat bu eksiklikler olmasa dahi kablo ile hareket aktarımının özelliğinden dolayı hassasiyette problemler olacaktır. Örneğin, robot başlangıç pozisyonunda dururken, kabloları elle, eğip-büktüğümüzde, kabloya bağlı uzvun hareket ettiği gözlemlenmektedir. Aynı olay, günlük hayatta bisiklet frenlerinde de gözlemlenebilir. Arka fren kablosu büküldüğünde fren pabuçları bir miktar hareket edecektir. Ayrıca bükülen kablolarda sürtünme aşırı miktarda artmaktadır. Bu da hareketi engelleyebilecek bir etkendir.

Bu tür kablo hareketi belli işler için gereklidir ve hassasiyet zor olmasına rağmen, kullanıcıdan alınan geri beslemeyle hareket edilir. Örneğin endeskop kullanımında, doktor oluşacak boşluklardan, aşırı yük ihtiyacından dolayı hedefine ilk planda ulaşamasa da, gözle endeskopun uç noktasının nerede olduğunu gördüğü için ayar yapabilir. Otomobilin yan aynalarında da aynı durum söz konusudur. Elle ayarlamalarda, ayna bazı konumlarda çok güç istediği için önce dönmez, sonra birden hızlı bir şekilde ve fazlaca döner. Fakat biraz uğraştırsa da sonunda ayar yapılır.

Robotun hassasiyeti, gitmesi gereken noktayı vererek ve bu noktayı ne kadar gerçekleştirdiğini gözlemleyerek çıkartılmıştır.



Şekil 13. 4 serbestlik dereceli ve üç boyutlu dizayn

Şekil 14. Yukarıdaki dizaynda motor bağlantı detayları

2.3 Direct Drive Motorlar

Motorları robot kolunun uzuvları üzerine yerleştirmenin yollarından biri “direct drive” motorlar kullanmaktır. Bu tür motorlar redüksiyon olmadan 200 Nm mertebelerinde torklar üretebilmektedir ve küçük boyutlara sahiplerdir. Bildiğimiz kadarıyla Türkiye’de direct drive servo uygulaması yapan bir firma yoktur. Fakat en azından bir tane direct drive motor üreten firmanın Türkiye temsilcisi vardır. Bundan sonraki projede dizayn için direct drive seçeneği incelenecektir.

3 Kontrol Sisteminin Geliştirilmesi

3.1 Servo Motor, Sürücü ve Bilgisayar Sisteminin Kurulması

Servo motorlar bilgisayara bağlanarak kendi geliştirdiğimiz programımız içinden çalıştırılmaktadır. Öncelikle sistemimizi kısaca tanıtmak istiyoruz.

- 0 ile 3000 dev/dk arasında sabit 2.2 Nm tork üretebilen 8 adet servo motor
- Her biri bir servo motoru kontrol eden 8 adet sürücü
- Sürücülerle bilgisayar arasındaki bağlantıyı sağlayan PCI 208 hareket koordinatör kartı (PCI 208 *Motion Coordinator)*
- Hareket koordinatör kartına hareket emirlerini yollayan üretici firmanın sağladığı Motion Perfect 2 yazılımı
- Kendimiz tarafından geliştirilmiş olan hareket koordinatör kartına doğrudan hareket emirlerini yollayan bilgisayar yazılımı

Şekil 15'da servo motorlar, PCI 208 kartının takıldığı ve yazılımların yüklendiği bilgisayar ve içinde sürücülerin olduğu pano görülmektedir. Şunu belirtmekte yarar var. Pano hariç diğer malzemeleri TÜBİTAK projemiz başlamadan önce diğer bir projeden almıştık.

PCI 208 kartının çalıştırılmasıyla ilgili sorunlar ortaya çıktı. Kart normalde 2 tane servo kontrol edebilmektedir. Diğer 6 servo için ek modül satın alınması, karta eklenmesi ve aktif hale getirilmesi gerekiyordu. Alınan bu ek modül paketinden eksik çıktığı için İngiltere'ye üretici firmaya geri yollandı ve uzun bir zaman kartın geri gelmesi beklendi.

Projemiz başlamadan hemen önce kart çalıştırılmış ve 2 tane servo motor bilgisayara bağlanmıştır. Projeyle beraber denemelere başlandı. İlk önce üretici firmanın sağladığı Motion Perfect 2 yazılımıyla motorlar çalıştırıldı. Bu program içinde Visual Basic diline çok benzeyen bir dil içermektedir. Bu dilin amacı endüstriyel otomasyon problemlerine pratik çözümler üretecek yazılımlar geliştirmektir. Ek olarak, Motion Perfect 2, PCI 208 kartı üzerinden sürücülere ulaşarak motor parametreleri okumak ve yazmak gibi çeşitli görevleri yapabilmektedir.

Bütün bu avantajlara rağmen Motion Perfect 2 ihtiyaçlarımıza cevap vermemektedir. Çünkü ihtiyaç duyduğumuz robot uzuvlarıyla ilgili hesaplamaları bu program yapamamaktadır.

Ayrıca, bizim yazdığımız programın çıktılarını Motion Perfect 2'ye aktarsak bile bunu gerçek zamanlı olarak yapamıyoruz.

Motion Perfect 2 dışındaki bir seçeneğimiz de üretici firmanın geliştirdiği Active X'dir (TrioPC ActiveX component). Active X bir bilgisayar programı içine gömülerek çalışabilen başka bir bilgisayar programı parçacığıdır. Bilgisayar programcılığında önemli bir yeri vardır. Birçok büyük firma kendi geliştirdikleri yazılımların içine satın aldıkları Active X'leri yerleştirerek programlarını daha kısa zamanda ve daha çok özellikle sunabilmektedirler.

Üretici firmanın Active X'ini kullanmada da ciddi sorunlar yaşandı. Active X parçacığı Visual Basic'de yazılmıştı ve Visual C++ içinde doğrudan kullanılamadı. Üretici firmanın Türkiye temsilcisi bu konularda tecrübesiz olduğundan sorunların çözümü olması gerekenden daha fazla zaman aldı.

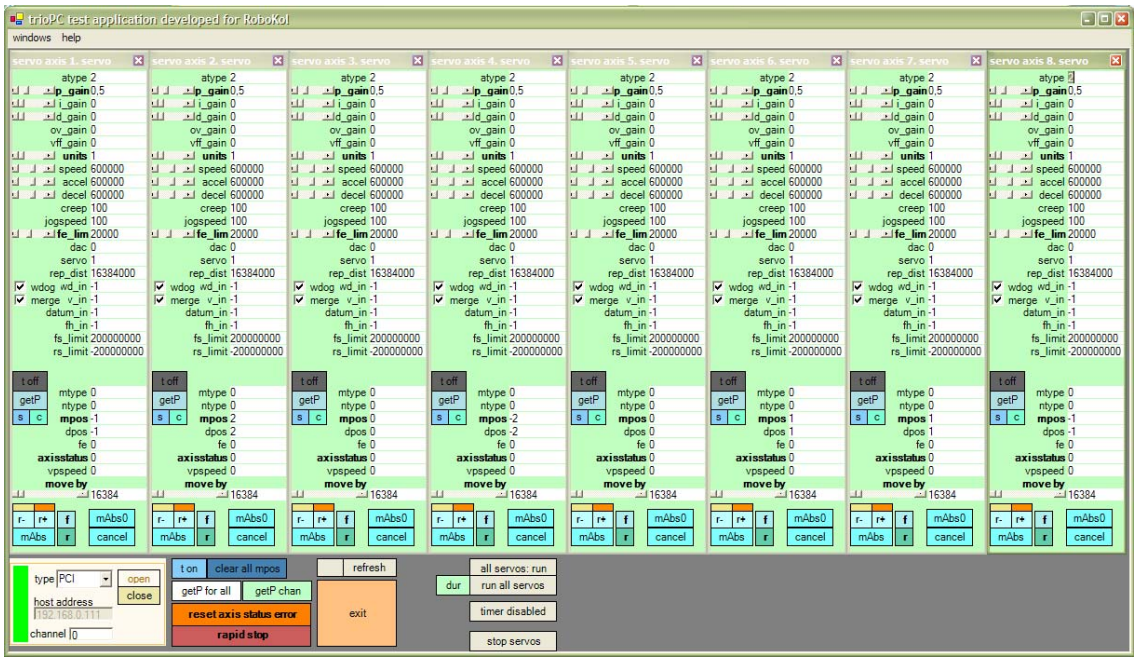
Sorunlarımız tam çözülmüştü ki bu sefer yeni çıkan Visual Studio.NET her şeyi değiştirdi. Biz Visual Studio.NET programını kullanmak istiyorduk ve Visual Basic de Visual C++ da bir önceki versiyona göre tamamen değişmişti. Bu iki dil şimdi çok daha fazla kolaylaşmış, güçlenmişti. Fakat bizim bu değişikliklere adapte olmamız belli bir zaman alacaktı. Ayrıca firmanın dillerin yeni versiyonları için örnekleri henüz yoktu. Belli bir zaman sonra projemizde görevli olan Yüksek Lisans Öğrencimiz İlker Eren'in firmadan yeni örnekleri getirmesiyle Visual Basic.NET dilinde Şekil 16'da arayüzü görülen programı geliştirmiş bulunuyoruz.

Bu programı geliştirmek için sürücüleri, PCI 208 hareket koordinatör kartını ve Active X parçacığını tanımak gerekli olduğundan bu programı geliştirmek oldukça fazla zaman almıştır.

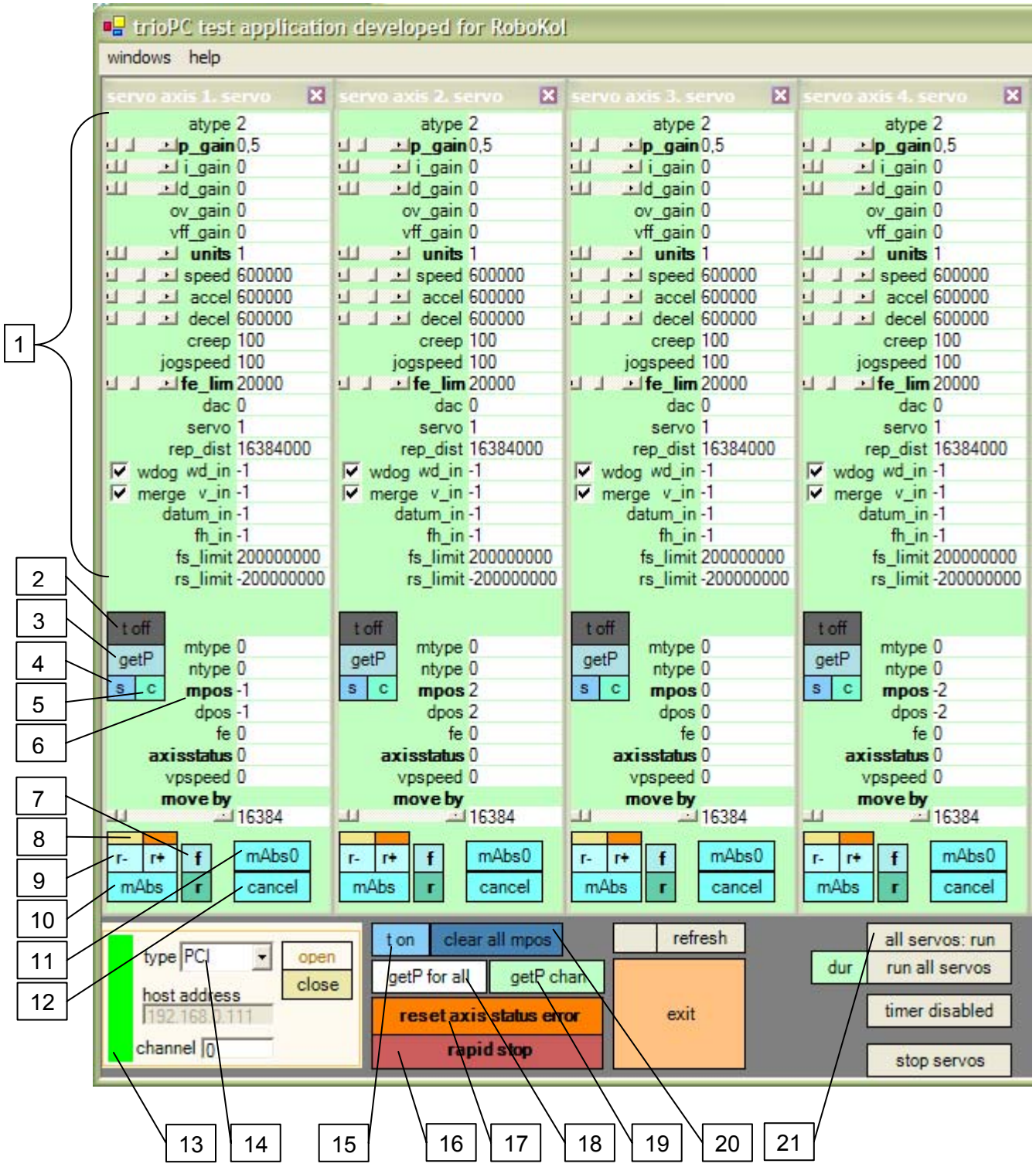
Şekil 16'da ara yüzü görülen program, Active X ile uyumlu olması için Visual Basic'de geliştirilmiştir. Fakat bu programı kontrol algoritmaları için kullandığımız Visual C++ Managed Extension'ları içinde kullanabiliyoruz. Yani, Active X'de olduğu gibi Visual Basic'de geliştirdiğimiz içinde Active X gömülü programı *.dll haline getirip Visual C++ içine gömüyoruz.

Geliştirdiğimiz bu program en yeni programlama teknolojisini temsil eden Visual Studio.NET ile geliştirilmiş "nesne temelli programlama" olarak isimlendirilen programlamadaki en son trendi kullanan bir yapıya sahiptir. Sadece bu tarafıyla bile öğrencilerimiz için iyi bir uygulama özelliği taşımaktadır.

Şekil 15. 8 tane servo motor, bilgisayar ve servoları kontrol eden sürücülerin bulunduğu pano



Şekil 16. 8 tane servo motoru test etmek ve ayarlarını yapmak için geliştirilmiş programın ara yüzü



Şekil 17. 8 tane servo motoru test etmek ve ayarlarını yapmak için geliştirdiğimiz programın işlevlerinin açıklanması

Şekil 17'de programın ara yüzü ile ilgili açıklamalar yer almaktadır. Ana form içinde her bir motor için ayrı bir form açılmıştır. Motor parametresi için ayrı ayrı metin kutuları ve düğmeler yerleştirmek yerine tek bir form hazırlayıp bu formdan 8 tane oluşturmak çok pratik olduğu için bu yol tercih edilmiştir. Formlar birbirinin aynıdır, sadece formlar oluşturulurken o formun hangi motorla ilişkilendirildiğini gösteren bir değişkenin değeri farklıdır. Formun bu değişkeni ile örneğin 0 ise 1. motor, 3 ise 4. motor kontrol edilir. Ayrıca, form üzerinde yapılan bir değişiklik aynen diğer örneklere yansıdığından değişiklik yapma ve hata bulma sekiz kat kolaylaşmaktadır.

1 numara ile gösterilen kısımda metin kutuları görülmektedir. Bu kutularda okunabilen ve yazılan eksen parametre değerleri vardır. Örneğin "speed" değişkenine istenilen bir değer girilerek gerçek zamanlı olarak motor hızı değiştirilebilir. Bazen girilen değerin PCI 208 kartına yazıldığından emin olmak gerekmektedir. Bu durumda 3 numara ile gösterilen düğmeye tıklamak, bütün parametre değerlerini PCI 208 kartından okuyarak ilgili metin kutusu içine almak için yeterli olur. Çok değiştirilen parametre değerlerinin soluna yerleştirilen kayma çubukları, değişiklik yapmayı daha kolay hale getirir.

6 numara ile gösterilen "mpos" parametresinin içinde olduğu grup "sadece okunabilen parametreler" olarak isimlendirilir. Bu parametreler belli aralıklarla okunur. Bu okuma işlemi 100 milisaniyede bir yapıldığında pratik olmaktadır. Daha sık aralıklarda yapılmak istendiğinde 8 tane motorun parametresi okunduğundan dolayı sistem yavaşlamakta ve bazı komutlara cevap vermemektedir. Gerektiğinde 15 nolu düğme ile okuma işlemi kapatılıp-açılabilir. Ek olarak, her motor için o motorun formunda bulunan 2 nolu düğme ile bu okuma işlemi ayrı ayrı açılıp-kapatılabilir. Fakat aynı anda dörtten fazla zamanlatıcı düzgün çalışmamaktadır. Bunu da bu vesile ile öğrenmiş bulunuyoruz. İstendiğinde 3 nolu düğme bütün eksen parametre değerlerini o motor için, 18 nolu düğme ise bütün motorlar için PCI 208 kartından okur.

Motorun iki durumu söz konusudur: "stop" ve "run". Motorun çok uzun süre hareketsiz kaldığı durumlarda aşırı ısınmayı engellemek için "stop" durumunda durması tercih edilir. 4 nolu düğme Motorun durumunu o motor için, 21 nolu düğme ise bütün motorlar için "stop" ve "run" arasında değiştirir.

6 numara ile gösterilen "mpos" değişkeni o eksenin enkoder pulsu olarak mutlak konumunu verir. Bir devir 4096x4=16384 enkoder pulsuna karşılık gelir. Bu da motor milini 16384'de bir döndürebileceğimiz anlamına gelir. Bazen "mpos" değerini sıfırlamak gerekir. 5 nolu düğme bunu o motor için, 20 nolu düğme ise bütün motorlar için yapar.

7 nolu düğme motoru sürekli olarak saat yönünde döndüren “forward” düğmesidir. Bu düğmenin hemen altında “r” ile gösterilen “reverse” ise motoru sürekli olarak saat tersi yönünde döndüren düğmedir. 12 nolu “cancel” düğmesi “forward” veya “reverse” düğmeleriyle devamlı olarak dönen motoru durdurma işlevini görür.

8 nolu düğme motoru yaklaşık 2 derece saatin tersi yönünde döndürür. Bu düğmenin hemen sağındaki düğme ise motoru yaklaşık 2 derece saat yönünde döndürür.

9 nolu düğme motoru saatin tersi yönünde hemen üstte sağda görülen metin kutusuna girilen değer kadar döndürür. Bu düğmenin hemen sağındaki düğme ise motoru saat yönünde metin kutusuna girilen değer kadar döndürür. Şekilde metin kutusunda motoru bir devir döndürecek olan 16384 değeri vardır. Belli bir değer kadar dönme yapan hareketler “bağıl hareket” olarak isimlendirilir.

10 nolu düğme motoru hemen üstte sağda görülen metin kutusuna girilen değere getirecek kadar saat yönünde veya saat tersi yönde döndürür. Bu hareket “mutlak hareket” olarak isimlendirilir. 11 nolu düğme ise motoru, pozisyonu sıfır olana kadar saat yönünde veya saat tersi yönde döndürür.

Program, çalıştırıldığında ilk iş olarak PCI 208 kartıyla bağlantı sağlamaya çalışır. Bağlantı başarılı olursa 13 nolu kutu yeşil renkli olur. Bağlantı başarısız olursa kırmızı renkli olur. 14 nolu kutu bağlantı tipini gösterir. Şekilde bizim bağlantı tipimiz olan PCI 208 kartı ile olan bağlantı görülmektedir.

Burada iki önemli düğme vardır: Bunlar, 16 nolu herhangi bir hata durumunda “acil duruş düğmesi” ve 17 nolu herhangi bir hata olduğunda hatayı giderip motorları tekrar çalışır hale getiren “sıfırlama” düğmesidir.

Ana formun en sağ alt köşesinde bulunan düğmeler ise 21 nolu düğme hariç motorları belli aralıklarla çalıştıran durduran deneme amaçlı düğmelerdir.

Programdaki hatalar giderilmiş ve program belli bir olgunluğa ulaşmıştır. Fakat gerektikçe programa eklemeler yapılmakta ve program iyileştirilmektedir.

3.2 Enkoder Bilgisini Bilgisayara Almak

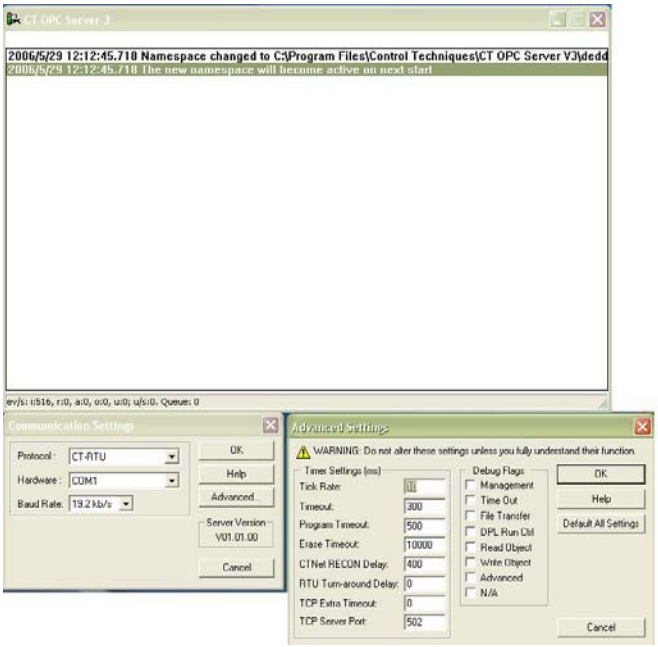
Robotun mafsalları üzerine enkoder yerleştirildiği takdirde buradan da geri besleme alınabileceği ve bunun faydalı olacağı düşünülmüştür. Gerçi motorların kendi enkoderleri vardı ve kapalı çevrim servo kontrollü çalışan motor ve redüktör sisteminde motor hareket çözünürlüğü 1/16384 hassasiyetinde sağlanmıştı ama mekanikte olabilecek kayıplardan dolayı robot uzuvlarının gerçek yerini verecek olan bu enkoderlerin gerekli olduğunu düşünüyoruz. Böylelikle uzvun gerçekte ne kadar döndüğü programımızdan okunabilecek ve eğer hatalı bir dönme varsa gerekli düzeltme program tarafından otomatik olarak yapılabilecektir. Konuyla ilgili çalışmalar aşağıda özetlenmiştir.

Her servo motorun sürücüsüne ikinci bir enkoder bağlayabilmek için 2 adet sürücü kullanarak Şekil 18'de görülen deney düzeneği hazırlanmıştır. Mevcut sistemde kullanılan her CT (Control Techniques) sürücüsünün 3 adet opsiyonel kart takma slotu vardır. Sürücünün kendi üzerinde hali hazırda bulunan enkoder girişine motor üzerindeki enkoderi bağlı bulunmaktadır. Bu yüzden opsiyonel kart olan "Encoder Plus" modülü temin edilmiş ve buna ikinci bir enkoder bağlantısı yapılmıştır. Encoder Plus modülü de sürücünün 1. opsiyon modülüne takılmıştır.

Karşımıza çıkan ilk problem bu enkoder bilgisinin hızlı ve doğru bir şekilde bir bilgisayar programına aktarılması oldu. Mevcut sürücülerde RS485 modbus RTU protokollü bir haberleşme protokolü bulunmaktadır. Araştırmalarımız sonucunda CT firmasına ait ve otomasyon dünyasının endüstriyel bir haberleşme standardı olan bir OPC server (OLE for Proses Control) yazılımını bulduk. Bu tür yazılımlar genellikle üreticiler tarafından ücretsiz verilmektedir. Şekil 19'da bu yazılımın arayüzü görülmektedir.

Bir sürücüde sistemi çalıştırdık. Daha sonra bu sistemi 8 veya 16 sürücü için nasıl çalıştırılabileceğimizi düşündük. Deneme amacıyla iki sürücünün her birine iki enkoder (bir tanesi motor için diğeri uzuv için) bağladık. Sürücüler RS485 portlarından uygun bir şekilde bağlanılarak bir network oluşturuldu. Sürücülerin network node adresleri değiştirilip ayarlanarak veri aktarımı 2 sürücü ile başarılı olarak gerçekleştirildi.

Şekil 18. Sistemin genel görünüşü

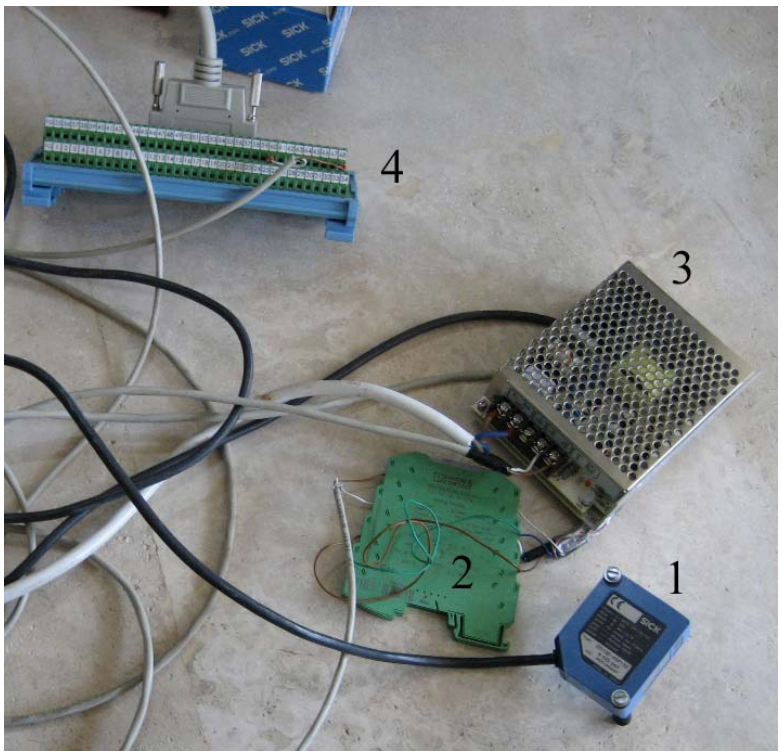


Şekil 19. CT OPC server yazılımının arayüzü

3.3 Mesafe Ölçen Lazer Sensörünün Montajı ve Çalıştırılması

Çevreden bilgi almak için proje başlamadan önce düşündüğümüz sensörler ultrasonik sensörler idi. Ultrasonik sensörleri tercih etmemizin sebebi hemen her tür cismi tanıyabilmesiydi. Çok sayıda ultrasonik sensörü sırayla uzuvlar üzerine yerleştirmeyi düşünmüştük. Fakat bizim uygulamamıza benzer pek uygulama olmadığından teknik yönden biraz yanıldık. Bu sensörleri belli bir mesafeden daha yakın yan yana veya ard arda yerleştiremiyoruz, çünkü algılamalar karışıyor. Daha sonra öğrencilerimizden birinin bitirme

projesinde web-cam kullanarak uzaktan kumandası bilgisayara bağlı bir oyuncak arabaya hedefi buldurduğumuzda, kamera görüntüsünü alarak işlemenin işimize yarayacağına karar verdik. Ayrıca lazer sensörü kullanmanın da faydalı olacağını düşündük. Aşağıda lazer sensörüyle ilgili yaptığımız çalışmalar anlatılacaktır.



1 - Lazer sensörü
2 - Lazer sensörünün çıktısı olan akımı voltaja çeviren dönüştürücü
3 - 24 volt DC çıkışı olan güç kaynağı
4 - Lazer sensörünün bilgisayar PCI DAC kartına bağlantısı

Şekil 20. Lazer sensörü ve bağlantı elemanları

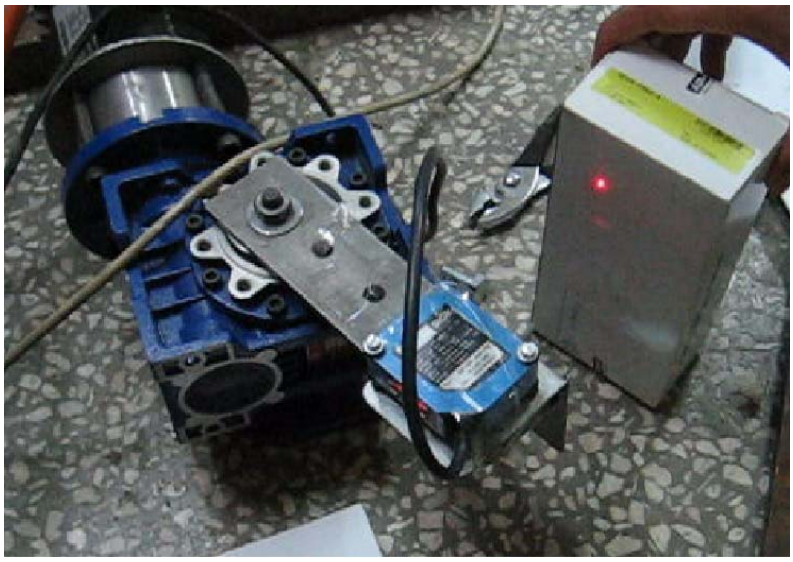
Şekil 20'de lazer sensörü ve bağlantı elemanları görülmektedir. Sensör oldukça başarılı bir şekilde çalışmakta, çok parlak olmamak şartıyla hemen her yüzeyde başarılı olmaktadır. Sensör 4-20 mA arası akım çıktısı verir. 4 mA 60 mm mesafeye 20 mA ise 140 mm mesafeye karşılık gelir. Bu değerlerin arası lineer olarak değiştiğinden 60 mm ile 140 mm arasının ölçümü elde edilir. Hassasiyeti 20 μm'dir. Fakat PCI DAC kartı gerilim aldığından bir dönüştürücüyle amper çıkışı giriş olarak alınıp 0-10 Volt aralığında gerilim çıktısı elde edilir. Tablo 1 bu değerleri özetlemektedir.

	Minimum Ölçüm	Maksimum Ölçüm
Sensör Çıktısı	4 mA	20 mA
Amper-Voltaj Dönüştürücü Çıktısı	0 Volt	10 Volt
Voltaj-mm Dönüşümü	60 mm	140 mm

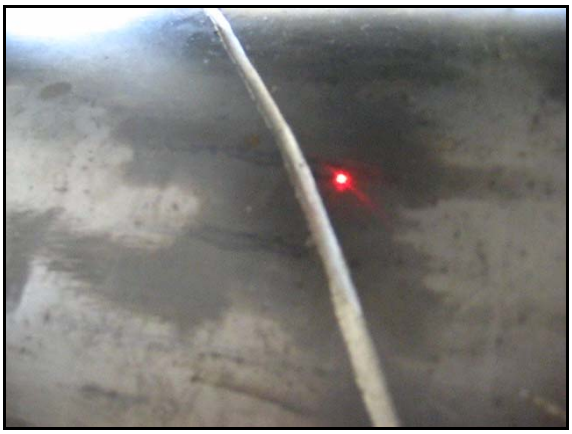
Tablo 1. Sensör ve dönüştürücü çıktıları

Lazer sensörüyle yaptığımız bir uygulama mesafeyi sabit tutma uygulamasıdır. Şekil 21'de görüldüğü gibi lazer sensörü servo motora bağlı bir kolun üzerine takılmıştır. Sensörün önünde hareket ettirilen karton kutu sensörden uzaklaştığında sensör verisini kullanan basit bir algoritmayla servo motor bu mesafeyi azaltacak yönde dönmektedir. Kutu sensöre yaklaştığında ise servo motor bu mesafeyi arttıracak yönde dönmektedir. Böylece mesafe daima sabit kalmaktadır.

Ekte verilen CD'de **104M260_ESahin_CONKUR_lazerMesafeOlcme.wmv** videosunda elde edilen hareket seyredilebilir.



Şekil 21. Lazer sensörünün mesafe sabitleme videosundan alınan bir görüntü

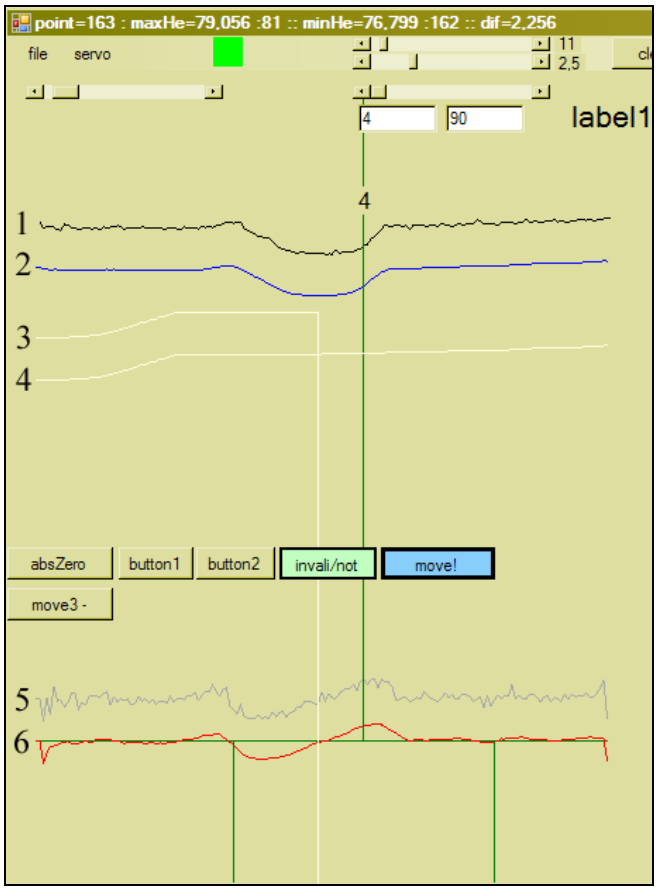


Şekil 22. Lazer sensörünün mesafe ölçerken çekilmiş videosundan alınan bir görüntü

Lazer sensörüyle yaptığımız diğer bir uygulama verilen bir bölgenin çizgisel olarak profilini çıkarmak ve en derin noktasını bulmaktır. Şekil 22'de görüldüğü gibi lazer sensörü yine servo motorla ilerletilmekte ve bu esnada hem sensör verisi hem de bu sensör verisi alındığı andaki servo motor pozisyonu kaydedilmektedir. Şekil 23'de sol tarafta "1" ile gösterilen eğri lazer verisi kullanılarak çizilmiştir. Diğer eğriler şeklin altında açıklanmıştır. Ekte verilen CD'de **104M260_ESahin_CONKUR_lazerYuzeyTarama.wmv** videosunda lazerin hareketi seyredilebilir.

3.4 Servo Motorların Güç ve Enkoder Kablolarının ve Lazer Sensörü Kablolarının Uzatılması

Daha rahat bir çalışma sağlamak için servo motorların güç ve enkoder kablolarını uzatmamız gerekti. Bu kablolar çok pahalıya mal olmaktadır. Piyasayı araştırarak manyetik korumalı çok daha ucuz kabloları kullanabileceğimizi gördük. Daha sonra bu kabloları alarak ara bağlantı lehimlerini kendimiz yaparak kabloları uzaktık. Motorlar sorunsuz çalışmaktadır. Aynı olayı lazer sensörü için de yaptık. Sensör bir problem çıkarmadan çalışmaktadır.



1 – Sensörden alınan yükseklik verisi

2 – Bu verinin “moving average filter” tekniği ile yumuşatılmış hali

3 – Verilerin düzenlenerek en uzun mesafenin bulunması

4 – Düzenlenmiş verilerin tamamının çizilmesi

5 – Verilerin açılarının grafiği

6 – Verilerin açılarının grafiğinin yumuşatılmış hali

Şekil 23. Lazer verisinin alınması ve en uzun mesafenin bulunması için işlenmesi

4 Gereğinden Çok Serbestlik Dereceli Robotlar İçin RoboKOL Programının Geliştirilmesi

Projemizin iki ana bölümü vardır. Birincisi mekanik dizaynın gerçekleştirilmesidir. Bundan önceki bölümde mekanik dizayn için gerekli olan servo sistemi ile ilgili gelişmeler aktarılmıştır. İkinci ana bölüm ise kinematik kontrol algoritmalarının geliştirilmesidir.

Bu projeden önce *RoboKol* ismini verdiğimiz, hareket planlaması algoritmaları geliştirebilmek ve simülasyonlarını bilgisayar ortamında görebilmek için Windows tabanlı Microsoft Visual C++ dilinde bir program yazmıştık. Bu programın en önemli özellikleri, engellerin ve robotların ekrana çizilmesi, potansiyel alanın iki ve üç boyutlu görüntülerinin elde edilmesi ve robotların hedefe varmasının gözlemlenmesidir. Program değişik hareket planlama algoritmaları içermektedir ve mobil robotlar için yörünge planlaması yapabilmektedir.

RoboKol, Microsoft Visual C++'ın 6. versiyonunda yazılmıştı ve Microsoft'un C++ için hazırladığı MFC (Microsoft Foundation Classes) kütüphanelerini kullanmaktaydı. Bilgisayar için geliştirilen büyük programlar çoğunlukla MFC'li Visual C++'ı kullanmıştır. MFC'nin avantajı bilgisayar ile yapılabilecek her şeyin bu versiyonla yapılabilmesidir. Dezavantajı ise öğrenme ve verimli olma zamanının oldukça uzun olmasıdır.

Genel olarak bir bilgisayar programını bir versiyondan diğer versiyona aktarmak çok zor bir iş değildir. Fakat Microsoft, Visual Studio.NET ile programlama olayını tamamen değiştirmiş ve eski versiyonlarla uyum diye bir şey kalmamıştır. Özellikle MFC'li Visual C++'dan "Framework"ü kullanan Visual C++ Managed Extension"a kolaylıkla geçmek mümkün değildir. Programın hemen hepsini yeniden yazmak gerekmesine rağmen, bir kez yazıldığında, elimizde artık çok güçlü bir araç olmuş olacaktır. Çünkü eskiye nazaran, daha az kodla daha pratik ve daha güçlü programlar yazmak artık mümkündür.

Algoritma geliştirebilmek yeni RoboKol programdaki temel kısımlar aşağıdaki gibi sıralanabilir:

- Robotu temsil eden sınıfı yazmak
- Engelleri temsil eden sınıfı yazmak
- Engelleri çizecek kodu yazmak

- Potansiyel alanı hesaplayıp ekranda görüntüleyecek kodu yazmak
- Menüler, ikonlar, dosya açma vb. kısımları yazmak
- Robotları ve engelleri içeren çalışma alanını kaydedecek kodu yazmak

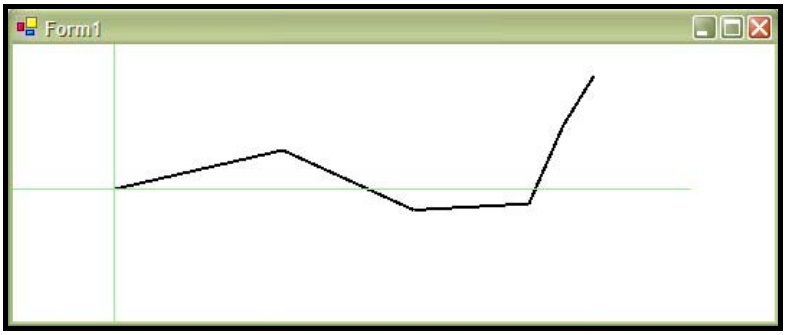
Algoritma geliştirme çalışmalarına hemen başlayabilmek için acil olarak yukarıda bahsettiğimiz kısımlardan robotu ekranda çizmeyi ve kaydetmeyi içeren kod öncelikle yazılmıştır. Yazılan kodla, istediğimiz sayıda uzvu olan robot kolu ekrana çizilebiliyor ve farenin koordinatlarını yörünge olarak alınıp fareyi takip edilebiliyor. Ayrıca robot bir dosyaya kaydedilerek daha sonra çağrılıp kullanılabiliyor.

Proje Öneri Formunda "Yöntem ve Kapsam" bölümlerinde "Hareket Planlaması" başlıkları altında anlatılan ve başarı ölçütlerinde belirtilen 2 numaralı problemin yani "robot uzuvlarının bir kısmının potansiyel alanın yönlendirmesine ters olarak hareket etmesi" büyük oranda çözülmüştür. Çözüm, aşağıda algoritma geliştirme kısmında detaylı olarak anlatılacaktır.

4.1 Nesneler ve Nesne Dizileri (Objects and Object Arrays)

Nesne temelli yaklaşım, geliştirdiğimiz *RoboKol* programının her kısmında kullanılmaktadır. Microsoft Visual Studio.NET dillerinden olan Visual C++.NET ile geliştirilmiş "nesne temelli programlamayı" esas alan bir yapıya sahiptir. Robot uzuvları, robotun kendisi, engeller, robotun uç noktasının çizimindeki çizgi parçacıkları gibi birçok işlem nesneleri tanımlayan sınıflarla başarılır. Her nesne de kendisiyle ilgili *ArrayList* isimli bir nesne dizisinde saklanarak kullanılabilir ve birçok manipülasyona maruz bırakılabilirler.

Nesnelerin ve nesne dizilerinin bizim programımızdaki öneminden dolayı geliştirdiğimiz basit bir programla nesneleri ve nesne dizilerini kısaca anlatmak istiyoruz. Bu program, çok sayıda uzva sahip bir seri manipülatörü oluşturup, manipülatörün uzuv açılarına keyfi değerler vererek ekranda manipülatörün hareketini sağlamaktadır. Arayüzü Şekil 24'de görülen ObjectArraysRobotDemo isimli programın tamamının kodu "http://sconkur.pamukkale.edu.tr/english/robotsoftware.htm" adresinden yüklenebilir.



Şekil 24. Beş uzuvlu manipülatör

Öncelikle aşağıdaki gibi bir uzuv (link) sınıfı yazılır:

```
public __gc class  link
{
        public:
        double l, teta;
};
```

Görüldüğü gibi "link" sınıfı son derece basittir, sadece iki tane üyesi vardır. "l" uzuv uzunluğu değişkeni ve "teta" uzuv açısı değişkenidir. Şimdi bir nesne dizisi (ArrayList) oluşturalım ve bu dizinin adresini "ar" isimli ArrayList tipinde bir işaretçiye atayalım:

```
ArrayList *ar;
ar = new Arraylist();
```

Daha sonra bir *link* nesnesi oluşturalım ve bu nesnenin adresini "le" isimli *link* tipindeki işaretçiye atayalım. Daha sonra, uzuv değişkenlerine keyfi değerler verelim:

```
link *le=new link();

le->l = 200;

le->teta = 0.2;
```

Sonra, *link* nesnesini nesne dizisine ekleyelim.

```
ar->Add(le);
```

Bu şekilde istediğimiz kadar *link* nesnesi oluşturup *ArrayList'e* ekleyebiliriz. *Link* ekleme bittiğinde artık manipülatör ekranda devamlı hareket edecek şekilde çizilmeye hazırdır.

Bunun için bir *timer* olayı kullanılabilir. *Timer'ın* içinde bir *link* nesnesi oluşturalım. *ArrayList*'te önceden kaydedilmiş *link* nesnesi bu *link*'e atanır. Böylece bu *link'in* üyelerine erişim

mümkün hale gelir. Aşağıda gösterildiği gibi her *link* açıları *timer*'ın her *tick* olayında belli bir miktar arttırıldığında ve *paint* olayında çizimler yapıldığında robot ekranda hareket eder.

```
link *le=new link();

le = dynamic_cast < link* >(ar->Item[0]);

le->teta+=-0.01;

le= dynamic_cast <link* >(ar->Item[1]);

le->teta+=0.02;
```

Link açılarına farklı arttırım değerleri verilerek değişik hareketler elde edilmektedir. Ayrıca robotun uç noktasının hareketi çizildiğinde çok ilginç eğriler elde edilmektedir. Belli bir zaman sonra robot ekranda çizdiği eğrilerin üzerinden giderek, hareketlerini tekrarladığı görülmektedir.

Eğer yukarıdaki sınıfları kullanmadan bu programı yazmaya çalışsaydık birçok ciddi zorlukla karşılaşacaktık. Örneğin her uzvun sadece iki değişkeni olmasına rağmen 5 uzuvlu robotumuza 10 tane değişken tanımlamamız gerekecekti. Uzuvlar için yeni bir değişken düşündüğümüzde her uzuv için ayrı ayrı değişiklik yapmamız gerekecekti. Robot kol sayısını arttırmaya kalktığımızda da kodun hemen her yanında zor olan değişiklikler yapmamız gerekecekti. Ayrıca döngüleri kullanarak kodu kısaltamayacaktık.

Aşağıdaki kod, yukarıda Visual C++'da yazılmış olan kodun Visual C#'da yazılmış halini göstermektedir. Kolayca görüleceği gibi Visual C#'da yazılmış kod daha basit ve anlaşılırdır.

```
public class  link
{
        public:
        double l, teta;
}
ArrayList ar;
ar = new Arraylist();

link le=new link();
le.l = 200;
le.teta = 0.2;
ar.Add(le);
link le=new link();
```

```
le = (link) ar[0];
le.teta+=-0.01;
le = (link) ar[1];
le.teta+=0.02;
```

4.2 RoboKol Programının Temel Yapısı ve Sınıfları

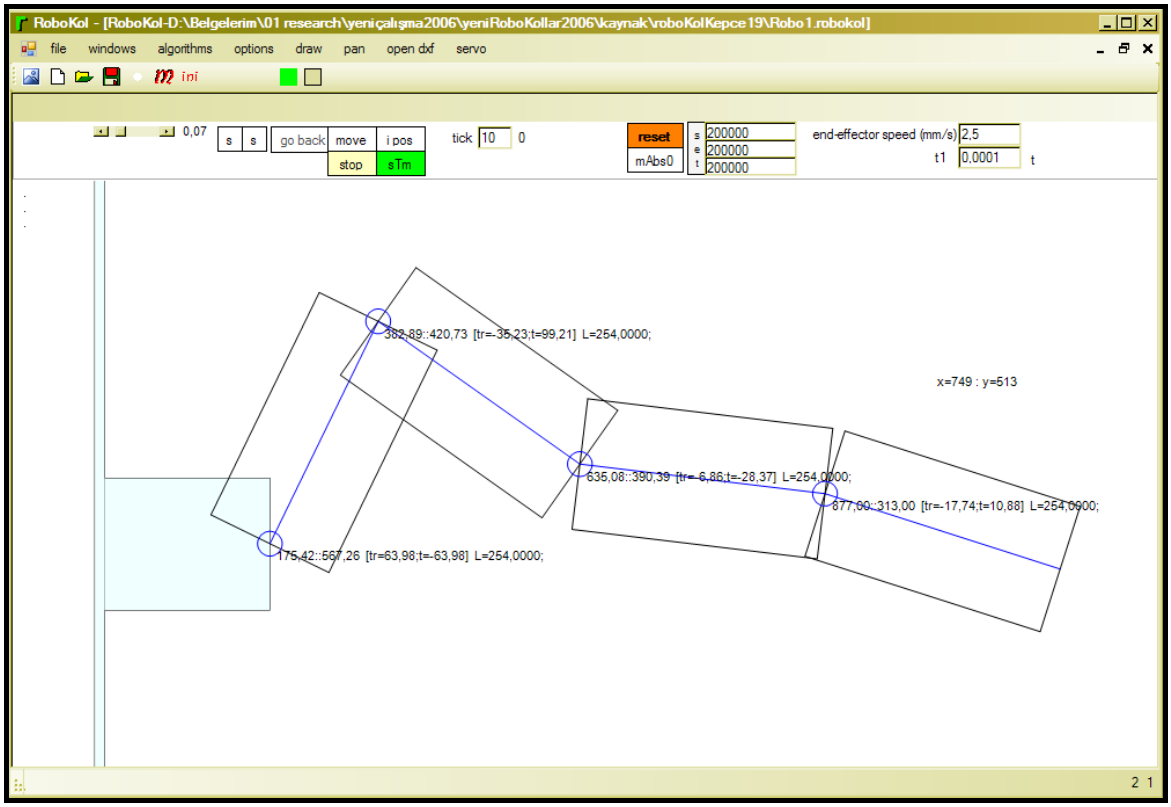
RoboKol programında fonksiyon ve değişken isimleri İngilizce olarak ve mümkün olduğunca yaptığı işe uygun olarak ve yeteri kadar uzunlukta isimlendirilmiştir.

Bundan önceki kısımlarda *RoboKol* isimli programımızın, Microsoft Visual C++'ın 6. versiyonunda yazılmış olduğunu ve Microsoft'un C++ için hazırladığı MFC (Microsoft Foundation Classes) kütüphanelerini kullandığını belirtmiştik. Daha sonra da genel olarak bir bilgisayar programını bir versiyondan diğer versiyona aktarmanın çok zor bir iş olmadığını MFC'li Visual C++'dan "Framework"ü kullanan Visual C++ Managed Extension"a geçmek için programı yeniden yazmak gerektiğini söylemiştik. Visual Studio 2005 çıktığında gördük ki Visual C++'ın hem ismi hem de içeriği değişmiş. "Managed Extention" ismi "CLR" olmuş. O ana kadar yazdığımız kodu elle düzelterek yeni versiyona adapte ettik. Bu da bir hafta gibi yoğun bir çalışma zamanı aldı. Bir örnek vermek gerekirse pointer işareti "*" kaldırılmış ve bunun yerini "^" işareti almıştır.

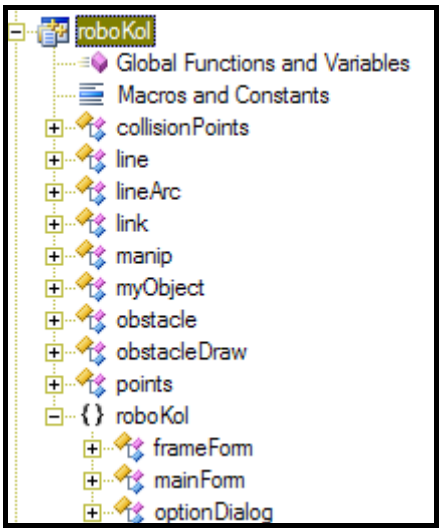
Şekil 25'de görülen *RoboKol* programının arayüzünde dört uzuvlu, uzuv genişlikleri ve uzuvların bazı parametreleri de görülen bir robot çizilmiştir. Algoritma geliştirirken uzuv genişlikleri ve parametre değerleri bazen gerekli olmakta bazen de bunların ekranda görüntülenmesi istenmemektedir. Bu yüzden uzuv genişlikleri ve parametre değerleri "options" diyalog kutusundan gerektiğinde iptal edilebilir.

Şekil 26'da yazmış olduğumuz kodda bulunan sınıflardan bir kısmı görülmektedir. Bu sınıflar aşağıda kısaca açıklanacaktır.

Şekil 27'de görülen *line* sınıfı robotun uç noktasının gittiği yolu ekranda çizmek için kullanılır. Bu yol ard arda eklenmiş çizgilerden oluşur. Şekildeki dört değişken her çizgi için başlangıç ve bitiş koordinatlarını saklar. Bu koordinatlara erişmek için ise dört tane değişken alan "yapıcı" tasarlanmıştır.

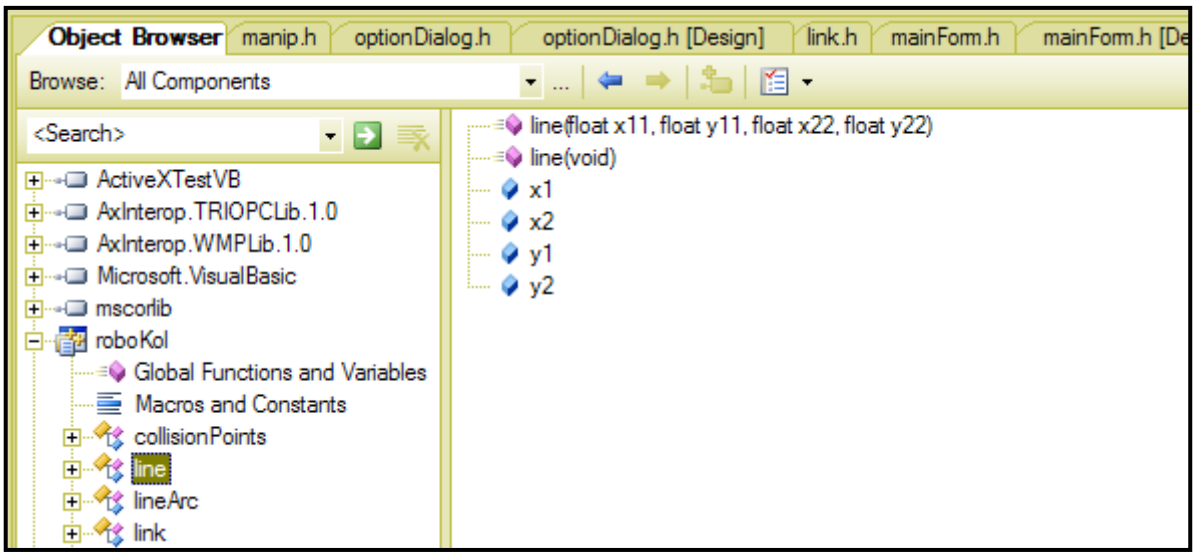


Şekil 25. *RoboKol* programının arayüzü

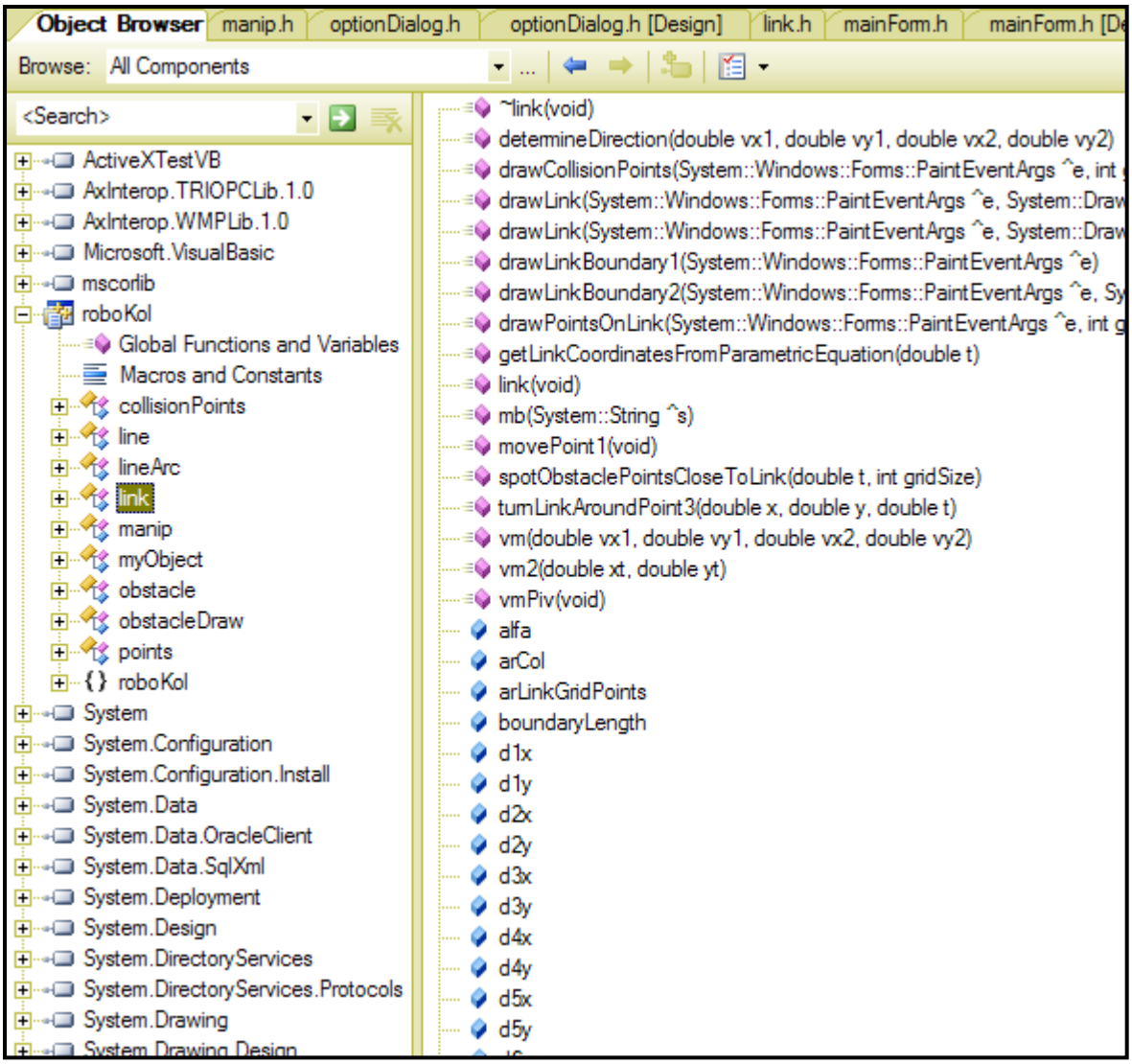


Şekil 26. RoboKol programının sınıfları

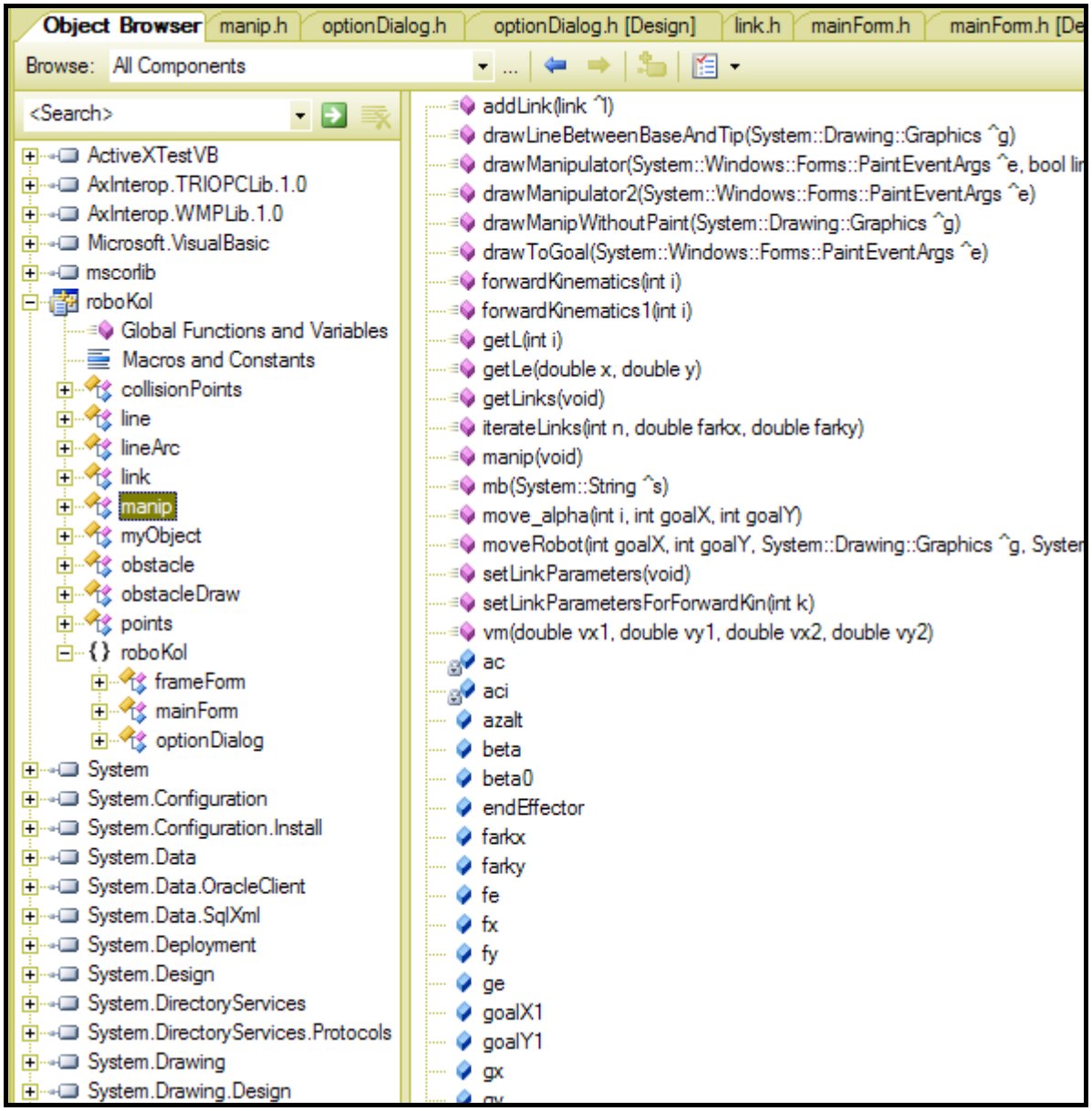
Şekil 28'de görülen *link* sınıfı uzuvlarla ilgili işlemleri bir araya getiren bir sınıftır. Her *link* ile ilgili başlangıç ve bitiş koordinatları, *link*'in rengi, *link*'in mutlak ve bağıl açıları gibi çok sayıda değişken bu sınıftan oluşturulan nesnelerce saklanır. Ayrıca *link*'in çizimini bu sınıfı kullanarak yapmak oldukça pratik olmaktadır. Bu sebepten Şekil 28'de görüldüğü gibi bu sınıf içinde *link* çizimiyle ilgili çok sayıda fonksiyon geliştirilmiştir.



Şekil 27. *line* sınıfının değişkenleri ve fonksiyonları

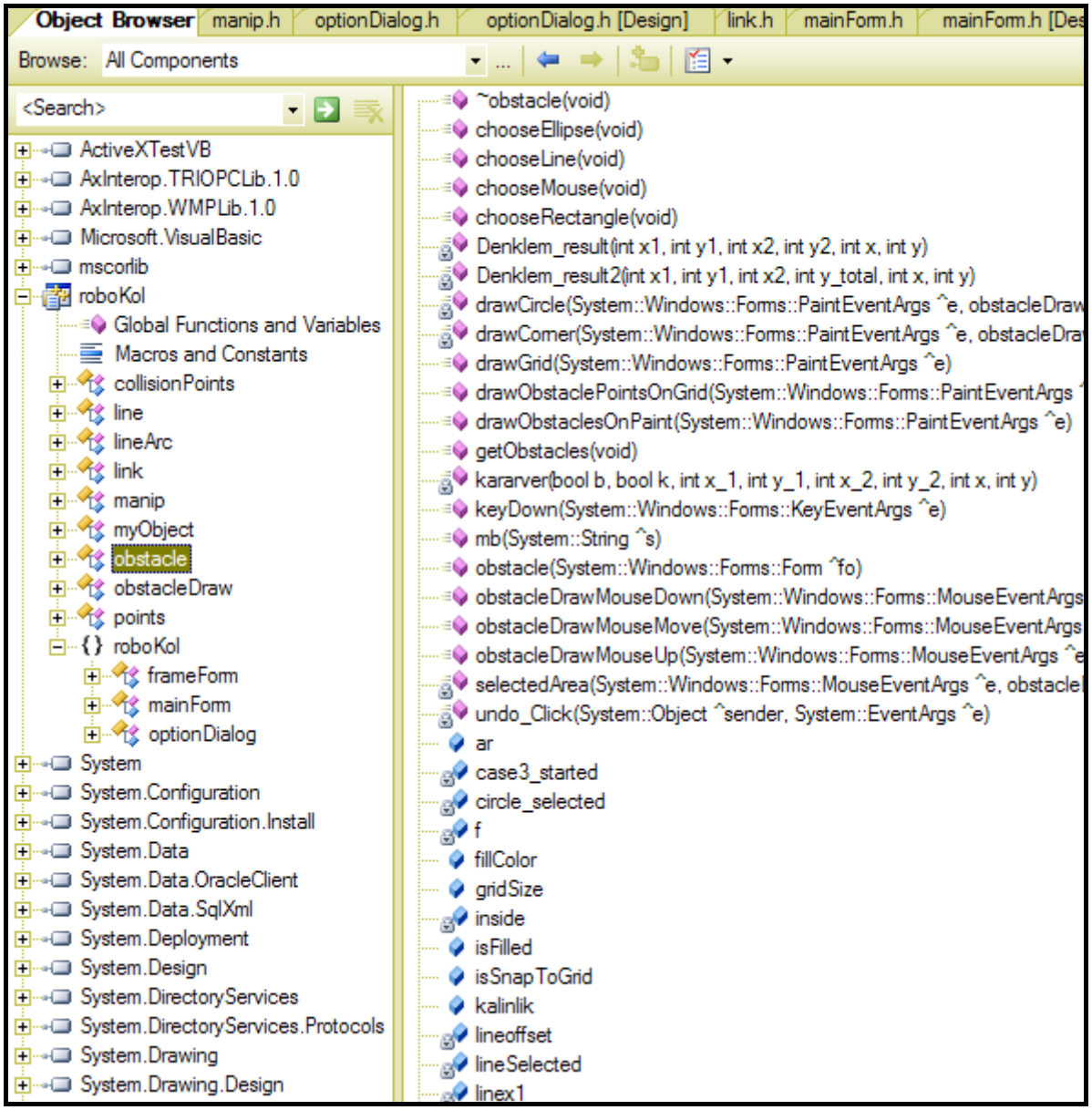


Şekil 28. *link* sınıfının değişkenleri ve fonksiyonları



Şekil 29. *manip* sınıfının değişkenleri ve fonksiyonları

Şekil 29'de görülen *manip* sınıfı manipülatörle ilgili işlemleri gerçekleştirir. Çok sayıda fonksiyon ve değişken içerir. Bu sınıfı kullanarak aynı anda birden çok manipülatör oluşturulabilir. Böylece program daha esnek bir yapıya sahip olmaktadır. *Manip* sınıfı yukarıda bahsedilen *link* sınıfını kullanarak uzuvlarını oluşturur ve uzuvlar arasındaki ilişkileri kurar. Örneğin bir *link* hareket ettirildiğinde diğer linklerin koordinatları değişecektir. Bu yüzden *link* nesnelerinin üyelerine yeni uygun değerler atanmalıdır. Bunu *manip* sınıfı "forwardKinematics" fonksiyonuyla yapar. Ekranda manipülatörü çizmek için *manip* sınıfında "drawManipulator" fonksiyonu geliştirilmiştir. Esasında bu fonksiyonda *link*'leri çizecek kodun kendisi yoktur. Bu fonksiyon sadece her *link* için *link* sınıfında bulunan ve esas çizimi yapan "drawLink" fonksiyonunu çağırır.



Şekil 30. *obstacle* sınıfının değişkenleri ve fonksiyonları

Şekil 30'da görülen *obstacle* sınıfı bu projede burslu olarak çalışan Yüksek Lisans öğrencimiz İsmail Boztay tarafından ayrı bir program olarak geliştirilmiş ve tarafımızdan *RoboKol* programına entegre edilmiştir. Bu bizim iki ayrı kişinin yazdığı kodu birleştirip tek parça haline getirmede ilk ciddi deneyimimiz oldu diyebiliriz. Burada ortaya çıkan ilginç bir nokta, "mousemove" gibi olaylara yazılmış kodu adapte etmekte ortaya çıktı. Bir programda tek bir "mousemove olayı olduğundan diğer programda olan "mousemove" olayına yazılmış kodu kopyalayıp, bu program içine yazmak yerine, "mousemove" olayının ismini değiştirerek basit bir fonksiyon haline getirip esas programın "mousemove"unda kullandık. Böylece karışıklığı önlemiş olduk.

Obstacle sınıfı, dikdörtgen, daire, çizgi gibi nesneleri ekrana çizmekte ve gerektiğinde büyültme, küçültme, uzatma, kısaltma ve silme gibi edit işlemlerine izin vermektedir. Burada ekrana bir dikdörtgen çizmekle onu fare ile değiştirmek için gerekli kodun oldukça farklı ve zor olduğunu vurgulamak gerekir.

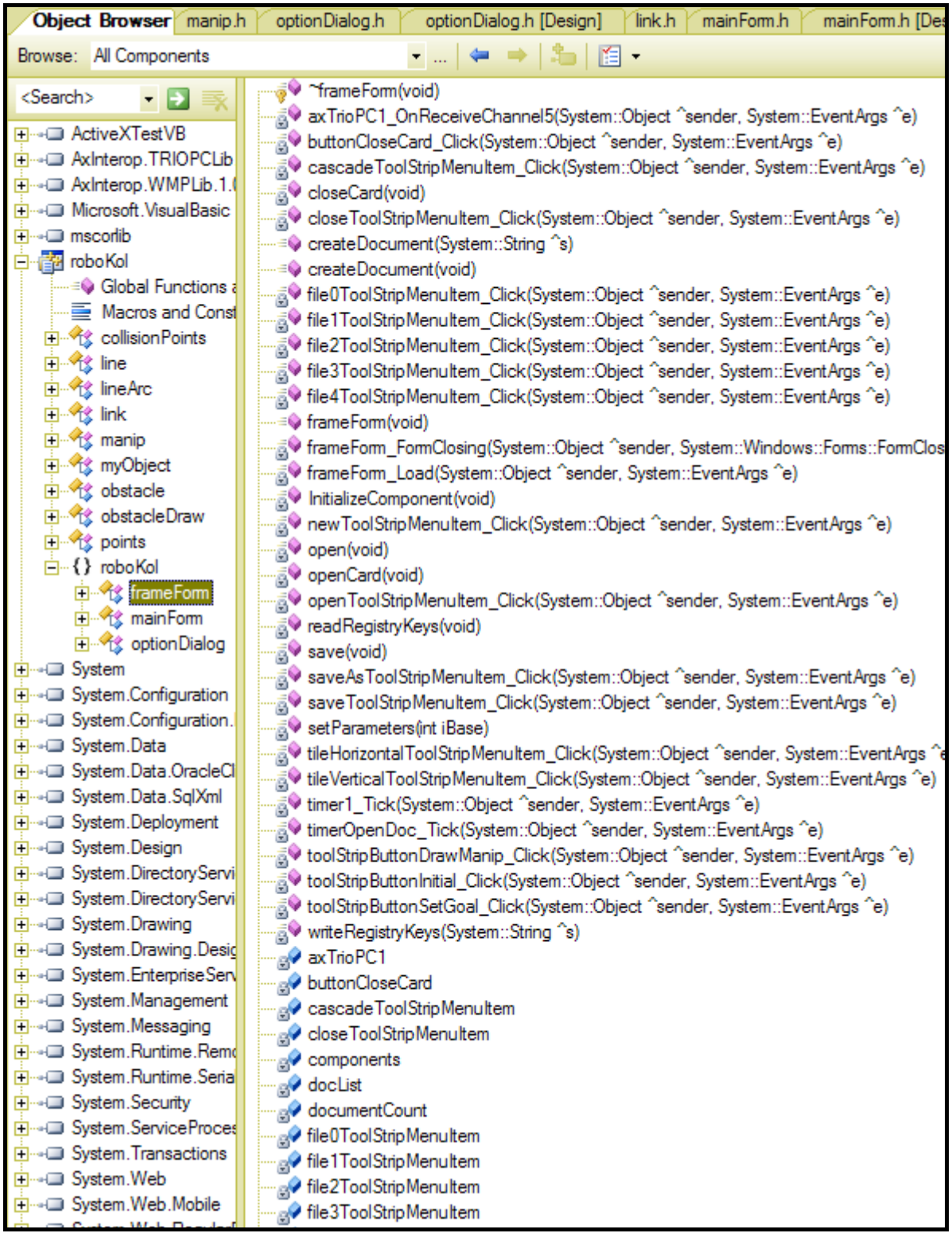
Bu sınıfın esas görevi ise ekrana çizdiği temel geometrik şekilleri, genişliği ve yüksekliği kullanıcı tarafından ızgarayı kullanarak "engel noktalarına" dönüştürmesidir. İstendiğinde ekrana bir ızgara çizebilir ve çizimleri bu ızgaraya uyumlu hale getirebilir.

RoboKol sınıfı MDI (Multiple Document Interface) özelliğine sahip bir programdır. Yani program içinde aynı anda birden fazla doküman açılabilir (Daha sonra gelecek olan Şekil 36'da bu özellik görülebilir). Şekil 31'de görülen *frameForm* sınıfının "parent" özelliği aktif hale getirilerek MDI oluşturulur. Bu sınıf kayıt yapmak, menüleri kullanmak, ikonları oluşturmak ve Windows kayıt defterine kayıt yapmak gibi işlemlerden sorumludur.

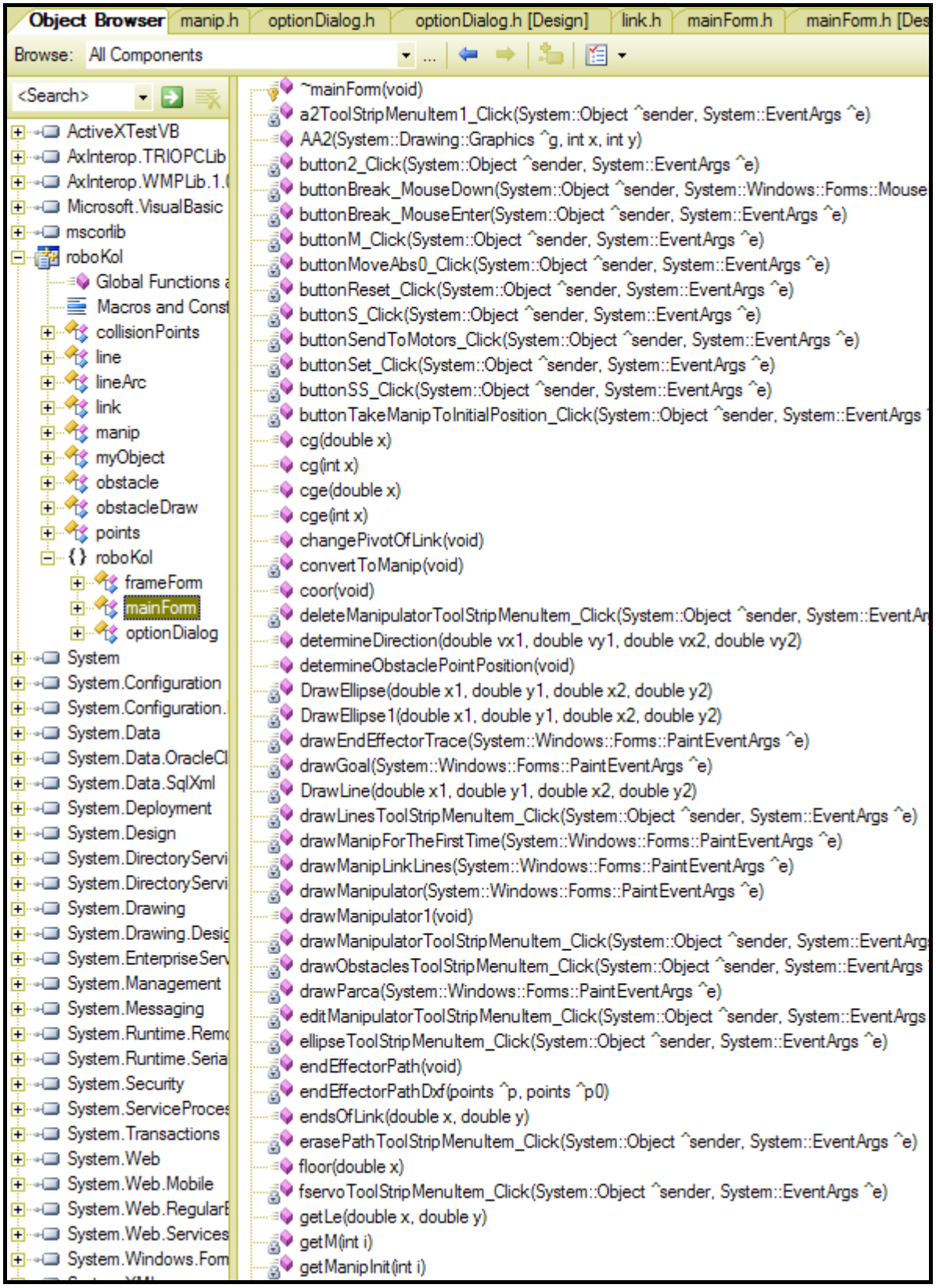
Şekil 32'de görülen *mainForm* sınıfı adı üzerinde asıl işi gören sınıftır. Diğer sınıflarda tanımlanmış fonksiyonlar, o sınıflar vasıtasıyla çağırılır ve bu sınıf içinde kullanılır. Şekilde de görüldüğü gibi çok sayıda fonksiyon ve değişken içermektedir. Bu yüzden bu raporun diğer kısımlarında da olduğu gibi sadece bazı önemli gördüğümüz noktaları açıklayacağız.

Öncelikle ekrana bir manipülatör çizilir. Fakat çizim tamamlanmadan hemen önce bu çizim sadece uç uca eklenmiş doğrulardan başka bir şey değildir. Çizim tamamlanıp farenin ortadaki tuşuna basıldığında "convertToManip" fonksiyonu bu çizgileri manipülatöre çevirir. Ayrıca, gerektiğinde manipülatörün ilk haline dönebilmek için manipülatörün ilk durumu ayrı bir *ArrayList*'de saklanır.

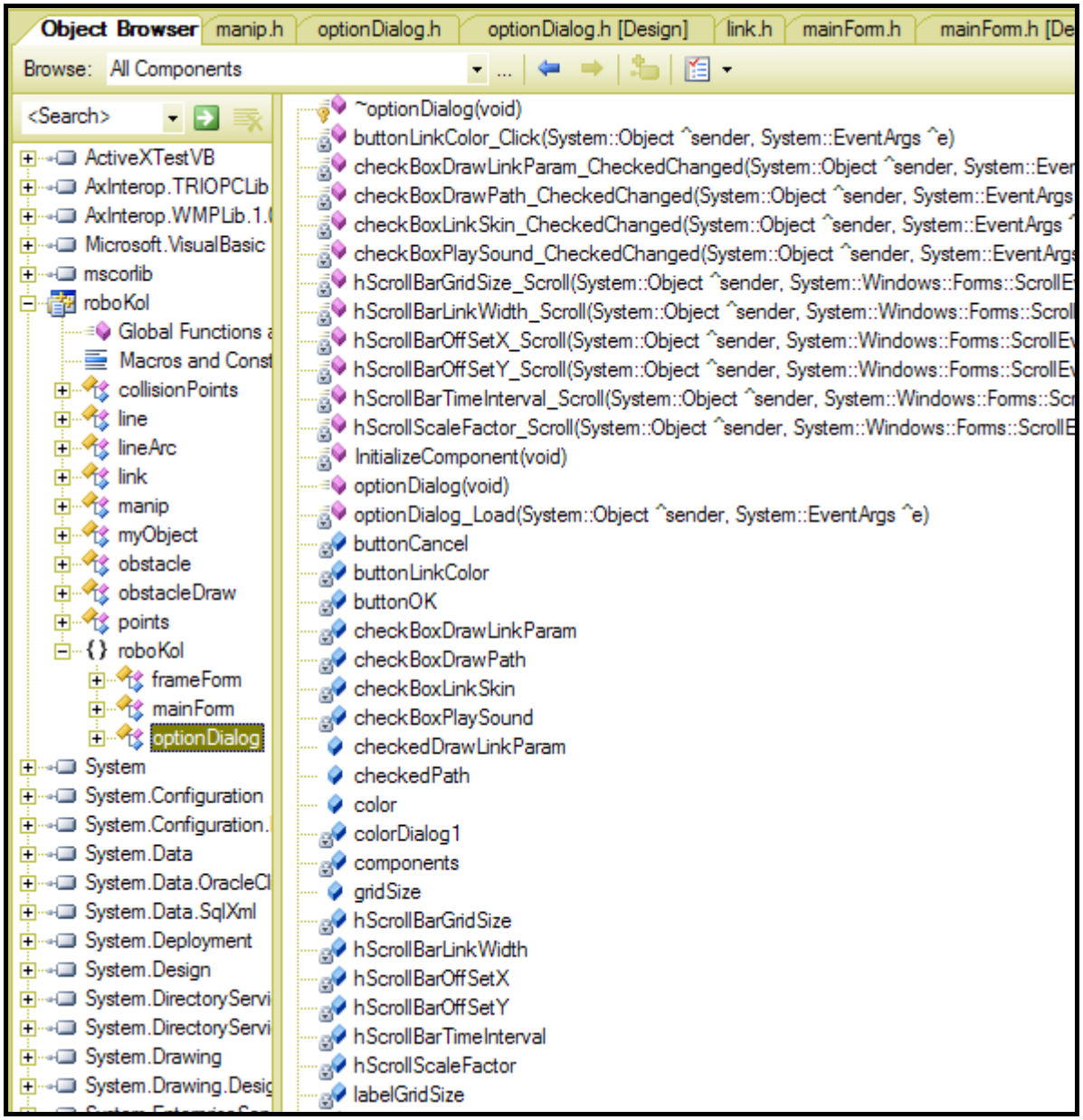
Program mümkün olduğunca değişik denemelere izin verecek şekilde esnek tasarlanmıştır. Birden çok manipülatör aynı çalışma alanında oluşturulabilir. Bu manipülatörler temel (ilk) uzuvlarındaki mafsala tıklanarak istenilen bir yere sürüklenebilir. Diğer mafsallara fare ile tıklanarak uzuvlar elle oynatılarak düz kinematik denenebilir ve uzuv uzunlukları fare ile değiştirilebilir. Bunlara ek olarak, bütün çalışma alanı fare ile sağa-sola ve yukarı-aşağı serbestçe kaydırılabilir, küçültülebilir ve büyültülebilir.



Şekil 31. *frameForm* sınıfının değişkenleri ve fonksiyonları



Şekil 32. *mainForm* sınıfının değişkenleri ve fonksiyonları

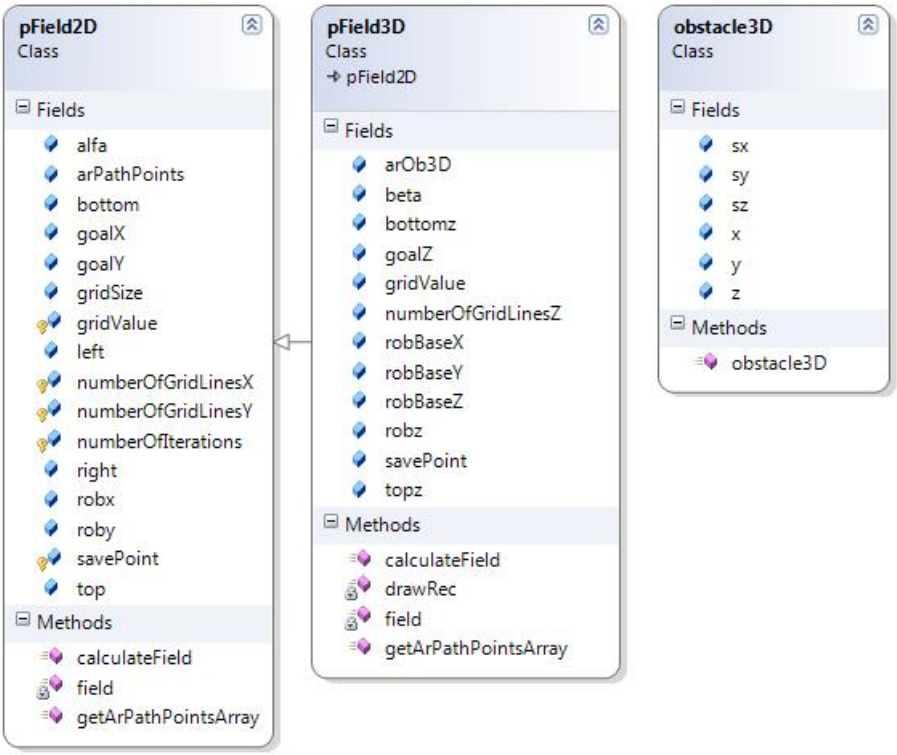


Şekil 33. *optionDialog* sınıfının değişkenleri ve fonksiyonları

Diğer bir yazması zor olan kısım da koordinat sistemini değiştirmek oldu. Bilindiği gibi Windows'un varsayılan koordinat sisteminin orijini formun sol üst köşesidir. *x* ekseni sağa doğru, *y* ekseni ise aşağı doğru pozitiftir. Bu sistemi bizim her zaman kullandığımız orijini sol alt köşede istenilen bir yerde olan, *x* ekseni sağa doğru, *y* ekseni ise yukarı doğru pozitif olan sisteme dönüştürdük.

Şekil 32'deki fonksiyonlar incelendiğinde, bunlar arasında servo motorlarla ilgili fonksiyonların olduğu görülecektir.

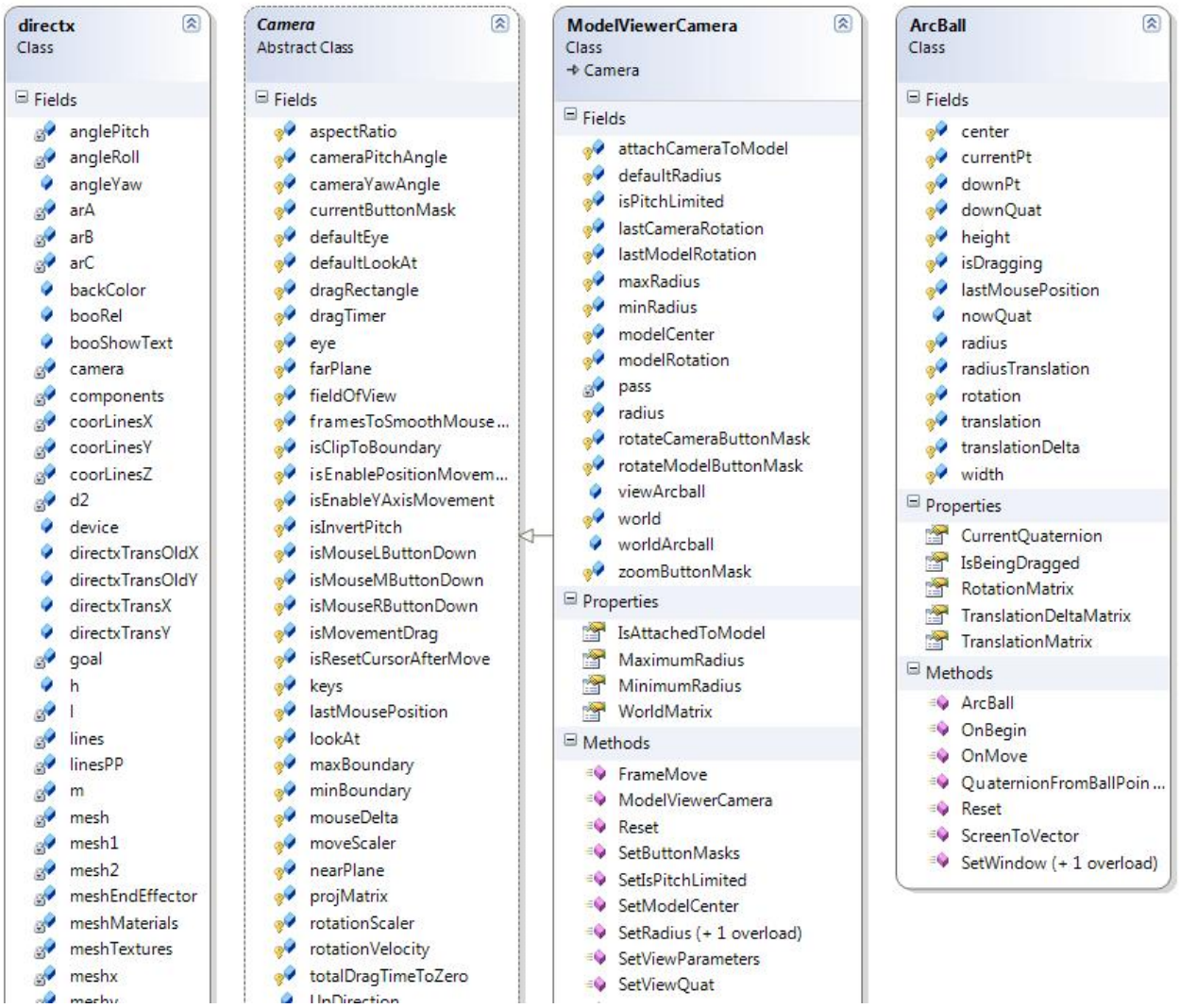
Şekil 33'de görülen *optionDialog* sınıfı (Şekil 40'da bu sınıfın arayüzü görülmektedir) *RoboKol* programında kullanıcının bazı çok kullanılan değerleri ayarlayabilmesini sağlar. Ekrandaki robot veya robotlar, engeller ve bu sınıf değişkenlerine verilen değerler kullanıcı dokümanı kaydettiğinde, doküman ile saklanır. Kullanıcı dokümanı açtığında değiştirdiği değerlerin saklandığını görecektir.



Şekil 34. Potansiyel alanla ilgili sınıflar

Şekil 34'de RoboKol için geliştirilmiş 2 ve 3 boyular için geliştirilmiş sınıflar ve 3 boyutta engelleri çizmek için geliştirilmiş engel sınıfı görülmektedir. Robokol'un bu sınıflarının çalışmasına ilişkin örnekler ilgili bölümlerde verilmiştir.

Şekil 35'de RoboKol için geliştirilmiş "direct X" ile ilgili sınıflar görülmektedir. Robokol'un bu sınıflarının çalışmasına ilişkin örnekler ilgili bölümlerde verilmiştir.

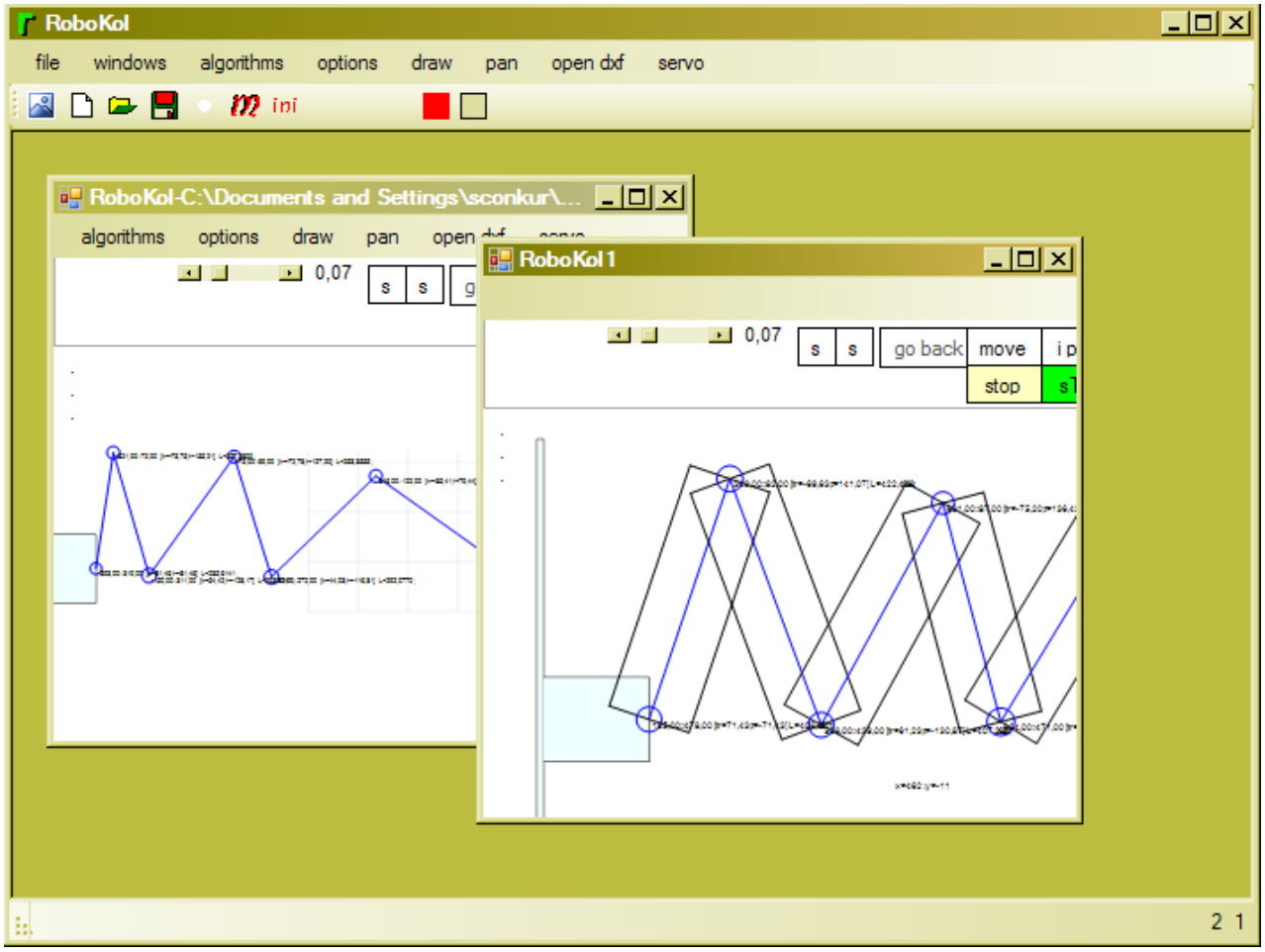


Şekil 35. Direct X ile ilgili sınıflar

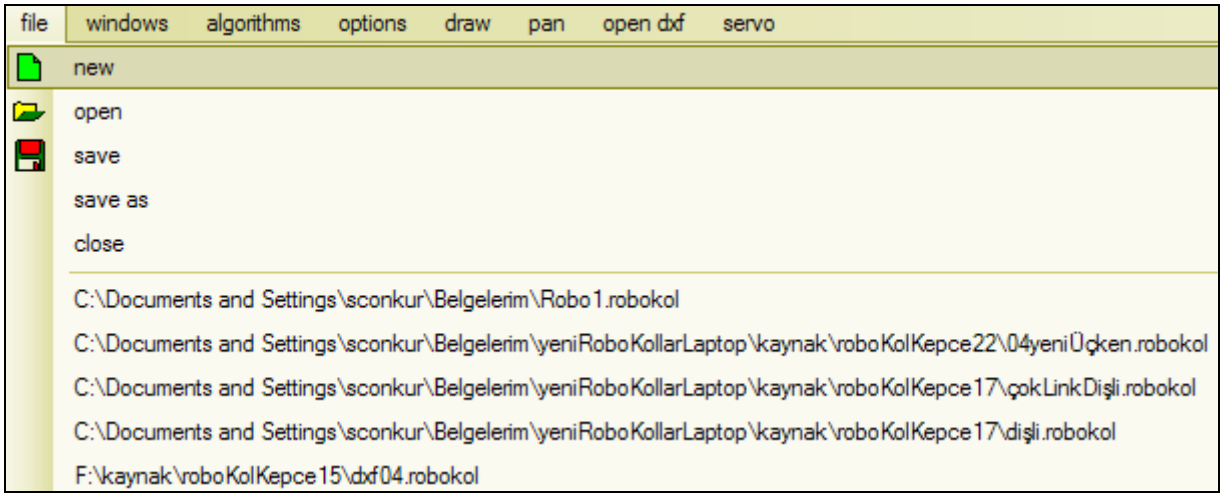
4.3 RoboKol Programının Menüleri

RoboKol sınıfının MDI özelliğine sahip bir program olduğu yukarıda bahsedilmişti. Şekil 36'da *RoboKol* programının ana çerçevesi içinde iki tane açık doküman görülmektedir. Bu dokümanlardan istenilen dokümanın başlık kısmına çift tıklanarak, bu dokümanın tüm formu kaplaması sağlanabilir. Ayrıca yine şekilde görülen "Windows" menüsü bu dokümanlara "tile horizontal", "tile vertical" ve "cascade" düzenlerinin verilmesini sağlar.

Diğer menülerden bir kısmı aşağıda kısaca açıklanmıştır.



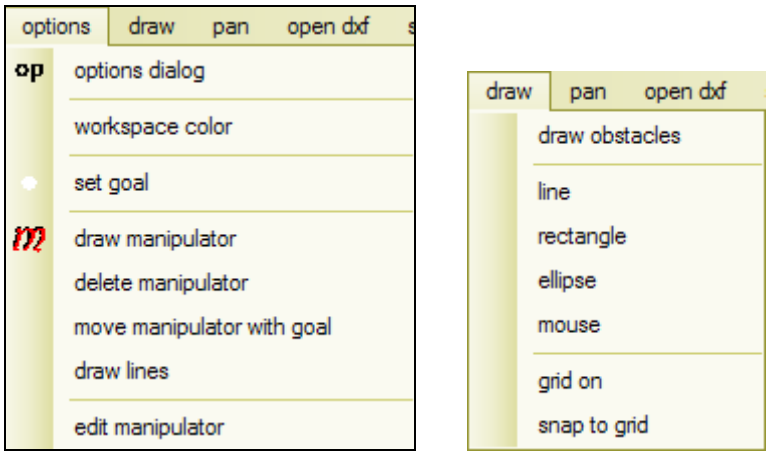
Şekil 36. *RoboKol* programının MDI özelliğini gösteren açık iki dokümanlı arayüzü



Şekil 37. *file* menüsü

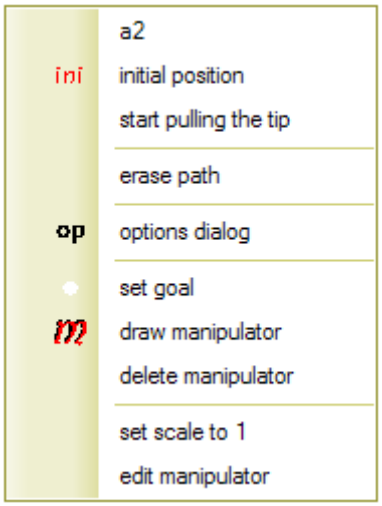
Şekil 37'de görülen *file* menüsü standart doküman işlemlerini gerçekleştirir. Şekilde görülen son açılan dokümanlar listesi MFC'de standart olarak gelmekteydi. Fakat yeni versiyonda hazır olmadığından bunu bizim yazmamız gerekti. Bu listeyi dokümanla kaydetmek mümkün

olmadığından “Windows kayıt defterine” yazmamız gerekti. En büyük zorluk da sıralamada ortaya çıktı. En son açılan dokümanın adının en üstte olması zorunluluğu bizi biraz uğraştırdı. Fakat sonunda sorunu çözdük. Bu listenin önemi doküman açmayı çok kolaylaştırmasıdır. Biraz daha ileri gidilip program açılırken en son dokümanın da otomatik olarak açılması sağlanmıştır. Böylece bir dosya üzerinde çalışırken devamlı olarak program açılıp kapandığından çok zaman kazanılmaktadır.



Şekil 38. *option* ve *draw* menüleri

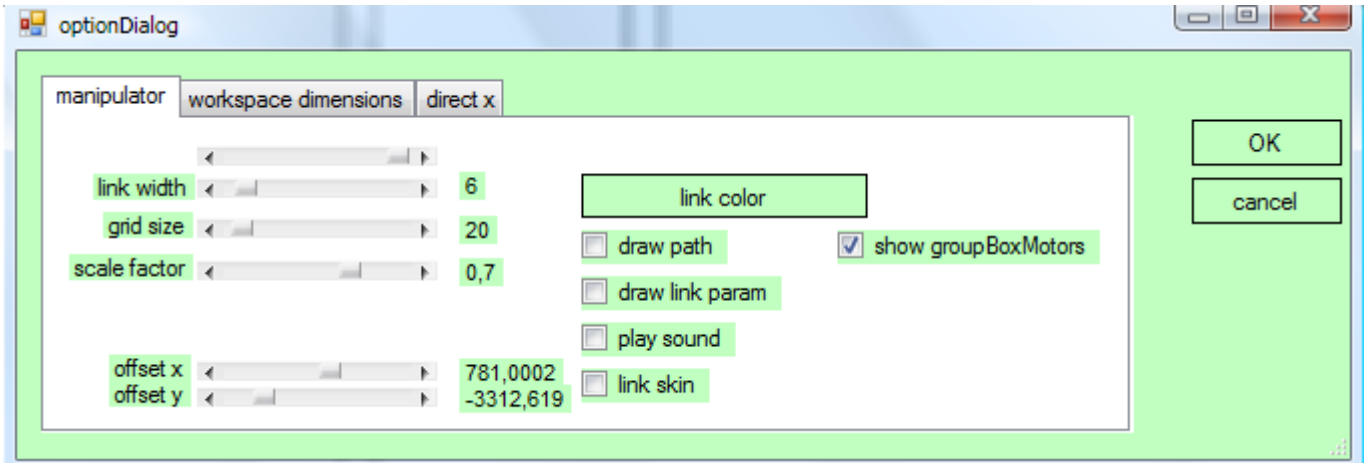
Şekil 38’de görülen menülerdeki komut isimleri kendilerini açıklamaktadırlar.



Şekil 39. *context* menüsü

Şekil 39’da fare form üzerinde hareket ederken sağ tuşa basıldığında farenin olduğu yerde ortaya çıkan ve en çok kullanılan komutları içeren “context” menüsü görülmektedir.

Şekil 40'da, *options* sınıfı incelenirken bahsedilen *options* dialoğu görülmektedir. Burada da yine diyalog içindeki seçenekler kendilerini anlatmaktadırlar.



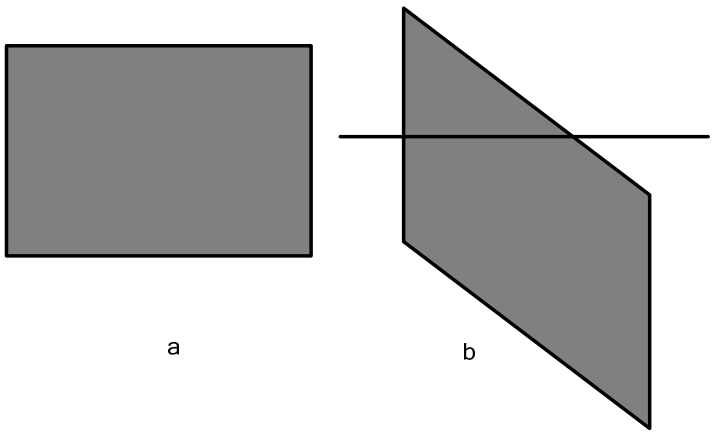
Şekil 40. *options* dialoğu

5 Robotun ve Çalışma Uzayının 3 Boyutlu Görüntülenmesi İçin Direct X Kodu Geliştirilmesi, RoboKol Programına Entegrasyonu ve Potansiyel Alan

5.1 Direct X Kodu Geliştirilmesi

TÜBİTAK projemiz kapsamında geliştirdiğimiz RoboKol programımız iki boyutta gelişmiş bir ara yüze sahiptir. İstenildiği kadar farklı uzunlukta uzvu olan robotlar aynı anda program çizim alanına çizilebilir. Daha sonra robotlar edit edilip uzuv uzunlukları ve açıları değiştirilebilir. Bütün çalışma alanı sağa veya sola kaydırılabilir, çalışma alanı içindeki robotlarla birlikte küçültülüp büyültülebilir. Bütün bunlar Visual Studio'da bulunan NET Framework 2.0 ile gelen "drawing" ile ilgili sınıflar kullanılarak geliştirilmiştir. NET Framework çok gelişmiş olmasına rağmen sonuçta, ilginçtir, sadece iki boyutlu çizim ile sınırlıdır. Bu sebepten üç boyuta geçilmek istendiğinde fazla bir faydası olmamaktadır.

Gerçekte üç boyut, bilindiği gibi yine iki boyutta çalışmaktır. Fakat göz yanılgısı olmakta, iki boyutlu ekranda nesneler üç boyutlu olarak canlandırılmaktadır. Bu canlandırmayı ilk baştan kod yazarak yapmak tabii ki mümkündür. Fakat çok fazla çalışma yapmayı gerektirmektedir. Örneğin Şekil 41'de görülen dikdörtgeni iki ve üç boyutta görüntülemeyi ele alalım. İki boyutta dikdörtgen "e.Graphics.Drawline" fonksiyonuyla dikdörtgenin sol üst köşesinin koordinatları, genişliği ve yüksekliği verilerek çizilir.

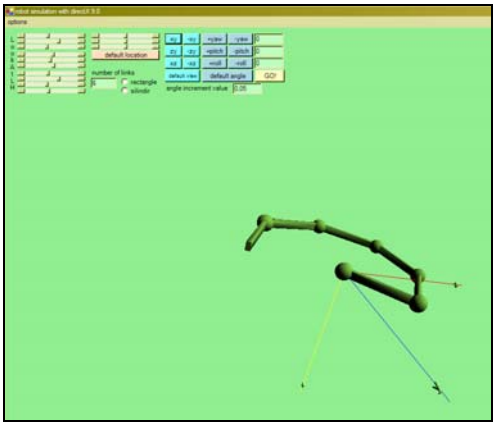


Şekil 41. 2 ve 3 boyutlu dikdörtgenler

Şekil 41'de sağda bu dikdörtgenin üç boyutta çizilmiş hali de görülmektedir. Tam karşıdan değil de yandan açılı olarak bakıldığında dikdörtgenin uzun kenarlarının artık yataya paralel değil de bakış açısına bağlı olarak daha kısa ve açılı çizilmesi gerekmektedir. Ayrıca örneğin dikdörtgenin içinin doldurulması gerektiğinde şekilde görülen doğru parçasının dikdörtgenin içinde kalan kısmının kapladığı piksellerin tespit edilip doldurulması gerekecektir. Bu da basit bir işlem olmayıp, bir algoritma geliştirmeyi gerektirmektedir. Bu şekilde geliştirilmesi gereken çok sayıda kod parçacığı vardır. Bu sebepten, nesneleri üç boyutlu görüntülemeyle ilgili basit bir yazılım geliştirmiş bulunmamıza rağmen, kendi kodumuzu geliştirerek üç boyutlu görüntüleme yapmak bize pratik gelmemiştir.

Esasında yukarıda bahsettiğimiz zorluklardan dolayı üç boyutlu çizim programlarında ve bilgisayar oyunlarında olduğu gibi üç boyutlu yazılımlar "Open GL" veya "Direct X" gibi hazır kütüphaneleri kullanılarak hazırlanırlar. Böylece hem estetik yönünden çok ileri hem de daha iyi bir simülasyon imkanı verebilen görüntüleme elde edilebilir.

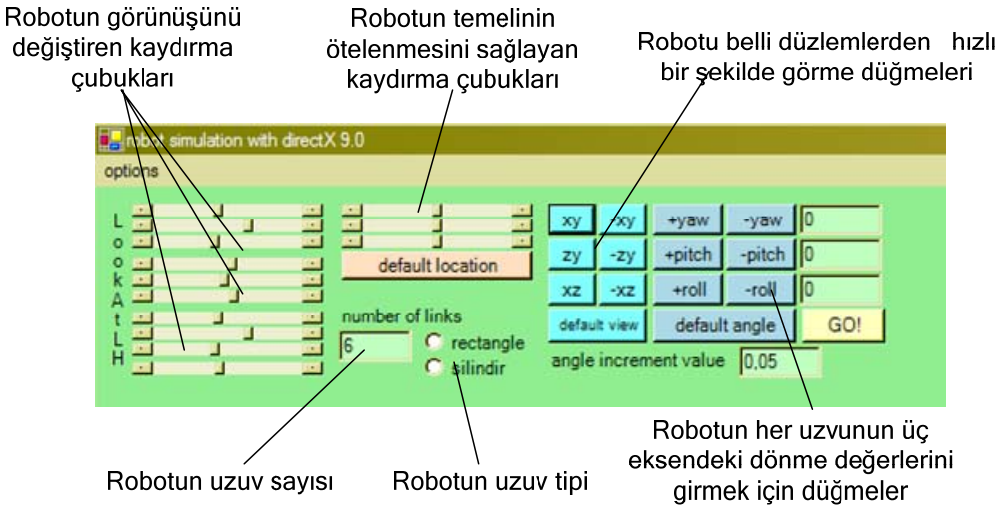
Open GL ve Direct X'in C++ için yazılmış kütüphaneleri oldukça gelişmiş ve olgunlaşmıştır. Fakat bizim kullandığımız Visual C++ Managed Extension'da bunları kullanmakta zorluk çekeceğimiz açıktı. Tam nasıl bir yol izleyelim diye düşünürken Direct X'in Managed Extension versiyonunun piyasaya çıkmış olduğunu öğrendik. Direct X'in Managed Extension örnekleri verilirken Visual C# dili kullanılmaktaydı. Yeni geliştirmeye başlandığı için de fazla bilgi ve örnek yoktu. Fakat Microsoft'da programı geliştiren ekibin başkanının yazdığı kitabı bulduktan sonra işimiz kolaylaştı. Direct X'in Managed Extension oyun yazmak için belki çok yeni, fakat bizim amaçlarımız için çok uygundu. Gelecekte de Microsoft artık bu kütüphaneleri geliştireceğini söylemektedir. Kitapta verilen örnekleri modifiye ederek 3 boyutta nesneleri nasıl oluşturacağımızı, onları nasıl döndüreceğimizi ve öteleyeceğimizi öğrendik.



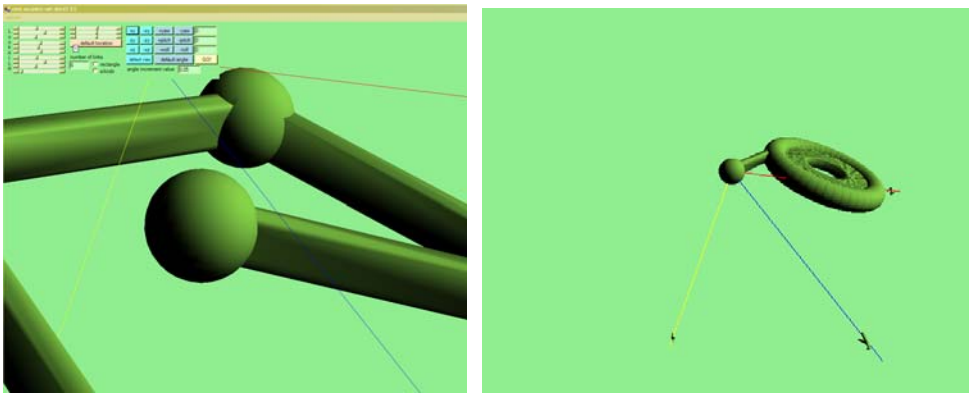
Şekil 42. Robotun 3 boyutlu görüntüsü için geliştirdiğimiz programın arayüzü

Verilen bir cismi döndürmekle beraber robotu görüntüleyebilmek için robotun ileri kinematiğini geliştirmemiz gerekiyordu. Yoksa buradaki bir hata her bir uzvun başka bir yöne gitmesine sebep olmaktadır. Robot kinematiğini çözdükten sonra her uzva devamlı artan açılar vererek robotun ileri kinematiğini gerçekleştirdik (Şekil 42). Geliştirilen programda istenildiği kadar çok uzuv sayısı girilebilir. Robot uzayda Şekil 43'de açıklanan kontroller kullanılarak istenildiği gibi döndürülebilir. Direct X normal bir bilgisayarda bile 2000 tane uzvu çalıştırabilmektedir. Bu hız gerçekten çok etkileyicidir.

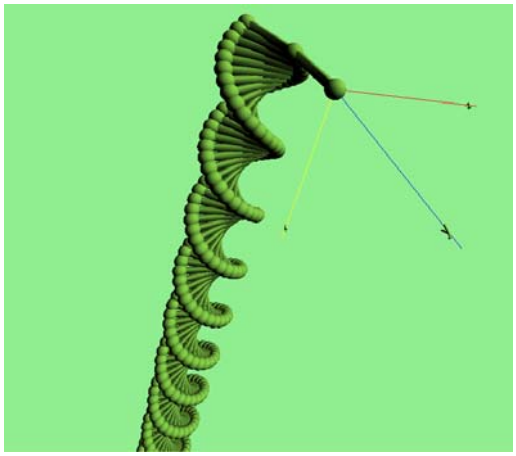
Ekte verilen CD'de **104M260_ESahin_CONKUR_directXRobotSimulasyonu.wmv** videosunda robotun görüntüsü seyredilebilir. Bu videoda robot çalışırken uzuv sayısı değiştirilmekte ve robota değişik açılardan bakılmaktadır. Videodaki robot hareketinde bazen kesiklikler olmaktadır. Bunun sebebi robot çalışırken, bu videoyu alabilmek için çalıştırdığımız "video capture" programının işlemciye aşırı bir yük getirmesidir.



Şekil 43. Robotun 3 boyutlu görüntüsü için geliştirdiğimiz programın arayüzündeki kontrollerin açıklanması



a b



c

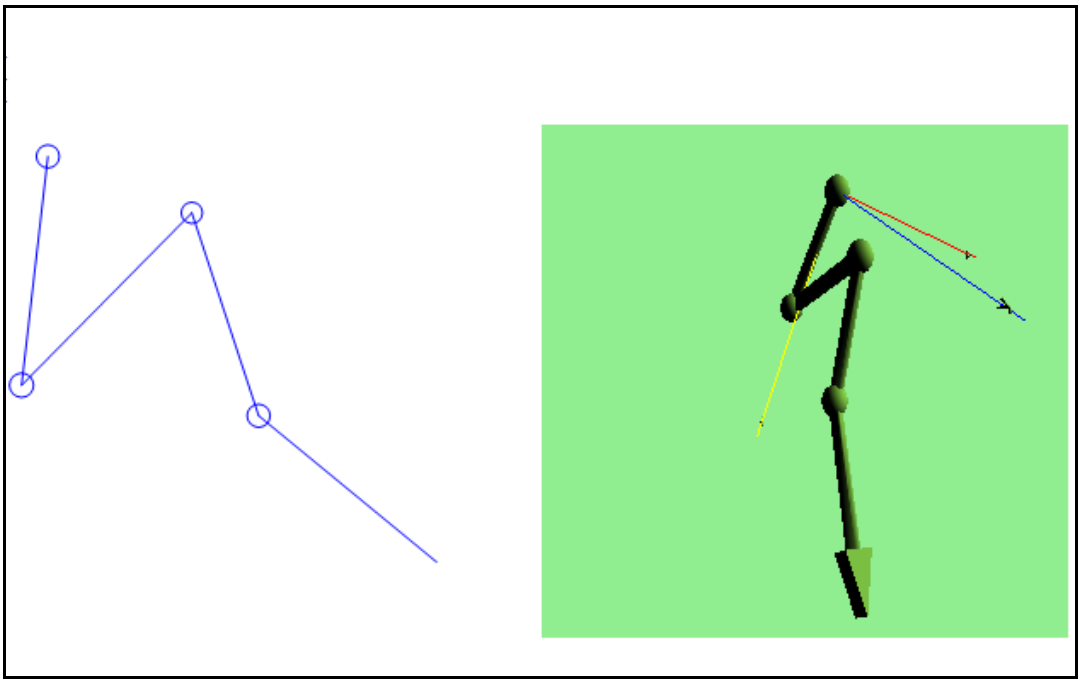
Şekil 44 a-c. Robotun 3 boyutlu görüntüsü için geliştirdiğimiz programdan görüntüler

Şekil 44 a-c'de robotun 3 boyutlu görüntüsü için geliştirdiğimiz programdan görüntüler görülmektedir.

5.2 RoboKol Programına Entegrasyonu

Yukarıda geliştirdiğimiz Direct X'i kullanan robot simülasyonu programı belli bir olgunluğa geldikten sonra sıra RoboKol programına entegre edilmesine gelmişti. Bunun için geliştirdiğimiz programı bir sınıf haline getirerek RoboKol'a entegre etmek istedik. Fakat Managed C++ için Managed Direct X desteği yoktu. Internet'te olan desteklerin çoğu Visual C# içindi. Bir süredir düşündüğünüz RoboKol'u çoğu Visual C#'a aktarma işine karar verdik ve bütün RoboKol kodunu çevirdik. Visual C# daha yavaş ama daha hızlı program geliştirme olanağı sunuyor.

Sonuç olarak robot simülasyonu için yukarıdaki programdan bir Direct X sınıfı hazırladık ve RoboKol'a entegre ettik. Şekil 45'de görüldüğü gibi 2 boyutlu robot 2 boyutlu uzayda hareket ederken aynı robot yine 2 boyutlu düzlemde fakat 3 boyutlu uzayda hareket etmektedir.



Şekil 45. Robotu 2 ve 3 boyutlu uzayda görüntüleme

Ekte verilen CD'de **104M260_ESahin_CONKUR_roboKol3dEntegrasyonu.wmv** videosunda robotun AutoCAD'dan aktarılan bir doğru parçasını takip ederken elde edilen 2 ve 3 boyutlu görüntüsü seyredilebilir.

5.3 Direct X İle İlgili Kodun Yeniden Düzenlenmesi

Yukarıda Şekil 44'de görülen direct X'de çizilen bir alanın değişik açılardan görünümü ile ilgili kontroller verilmişti. Fakat bu kontroller çok detaylı olmalarına rağmen istenilen görünümleri vermede yetersiz kaldı. Bizim istediğimiz çalışma alanını hızlı bir şekilde her yönden görebilmektir. Bunu yapabilmek için Microsoft'un direct X için oyun geliştiricilerine yönelik hazırladığı örnek kodlara başvurduk. Sonuç olarak mouse hareketlerine bağlı olarak istenildiği yönden görünümü veren kodu geliştirdik.

Bölüm 4.22'de Şekil 35'de kendi geliştirdiğimiz ve yeniden organize ettiğimiz **direct X** sınıfı ve bakış açısını ayarlamada kullandığımız Microsoft'tan adapte edilmiş olan **Camera**, **ModelViewerCamera** ve **ArcBall** sınıfları görülmektedir.

Ekte verilen CD'de **104M260_ESahin_CONKUR_potansiyelAlan3D.wmv** videosu yeni geliştirdiğimiz çalışma alanını değişik açılardan görüntüleme ile ilgili olarak fikir vermesi için seyredilebilir.

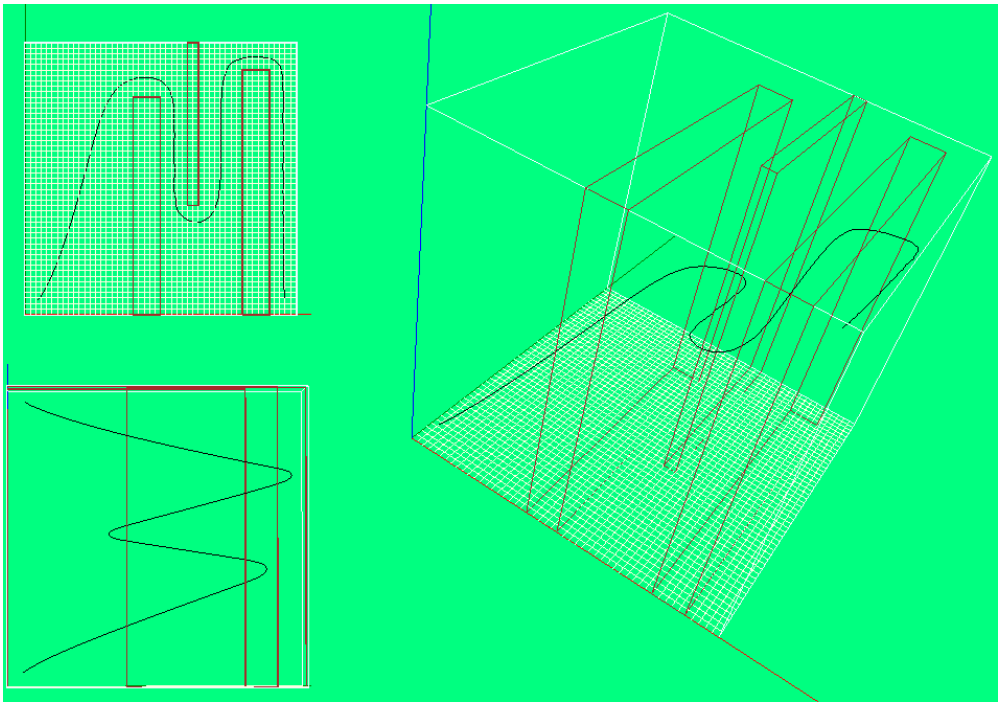
5.3.1 Direct X Kullanılarak Görüntülenen 3 Boyutlu Çalışma Alanında Potansiyel Alanın Hesaplanıp Hedefin Bulunması

Burada 1000*1000*1000 birim^3 boyutlarında içinde 3 tane dikdörtgenler prizması şeklinde engel içeren, grid büyüklüğü 20 birim bir çalışma alanı kullanan, başlangıç noktası (40, 40, 40) ve bitiş noktası (950, 50, 950) olan ve engeller arasından hedefe yol çizen bir örnek verilmektedir.

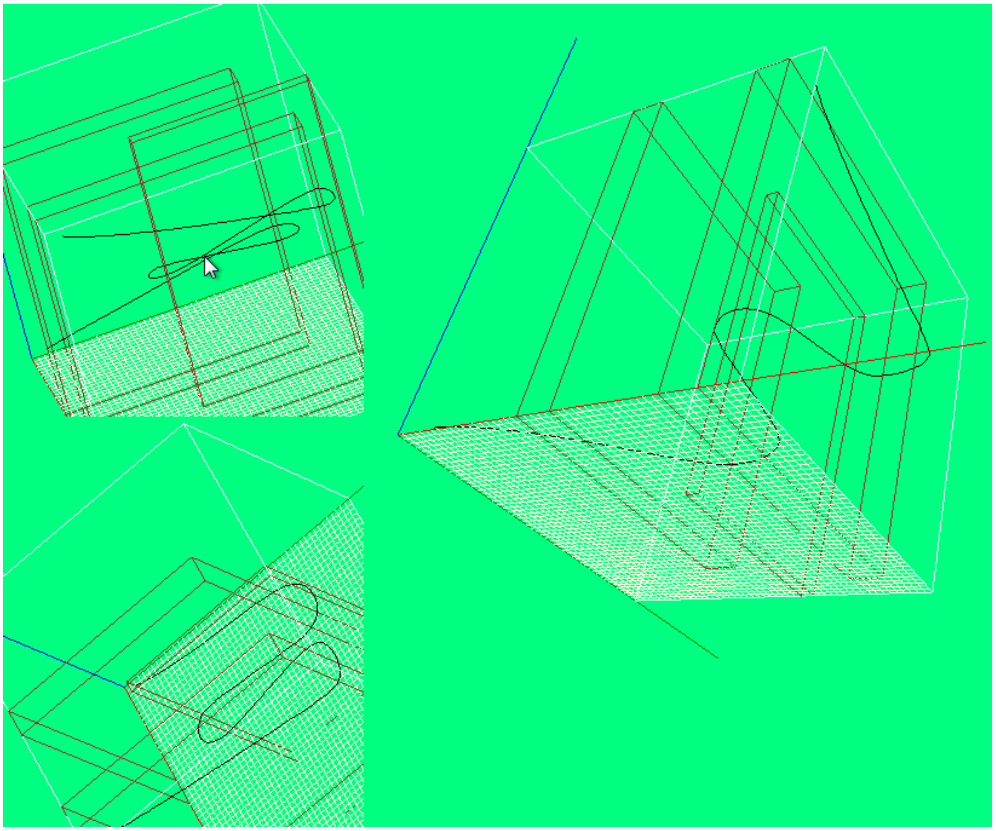
Bu özelliklere sahip çalışma alanında 400 iterasyon yapıldığında hesap zamanı 2.0 GHz bir bilgisayarda yaklaşık 3 s olmaktadır. Kenar uzunluğu grid büyüklüğüne bölündüğünde grid noktası sayısı her kenar için 50 olmaktadır. Böylece, toplam grid noktası sayısı 50*50*50=125000 dir. Bu durumda hesaplamalar 400*125000=50000000 kez yapılmaktadır.

Şekil 46'da sağ tarafta büyük görüntüde başlangıç noktasından hedefe varan yol çizilmiştir. Sol üst taraftaki daha küçük görüntüde ise çalışma alanının üstten görünüşü verilmektedir. Bu görüntüden yolun engeller arasından geçtiği bariz olarak anlaşılmaktadır. Sol alt taraftaki küçük görüntüde ise çalışma alanının yandan ve tam karşıdan görünüşü verilmektedir. Burada ise yolun z ekseni boyunca yükselmesi rahalıkla gözlemlenebilmektedir.

Nesne temelli programlamadan dolayı kolaylıkla aynı özelliklere sahip istenildiği kadar ek görünüş elde edilebilir. Fakat bizce üç tane yeterlidir. Örneğin, herbir görünüş mouse hareketlerine tepki verebilir. Şekil 47'de aynı örnek için mouse ile **değiştirilmiş** bakış açıları görülmektedir.



Şekil 46. 3 boyutlu örnekteki çalışma alanı ve görünüşler



Şekil 47. 3 boyutlu örnekteki çalışma alanı ve **değiştirilmiş** görünüşler

5.4 Potansiyel Alan

Robotun uç noktasının takip edeceği yörünge potansiyel alan metodu ile bulunmaktadır. RoboKol'da potansiyel alan ile ilgili kod geliştirilene kadar robotun "uç" noktasının takip edeceği yörünge mouse hareketlerine bağlı olarak verilmiştir. Potansiyel alan geliştirildikten sonra 2 boyutta hem potansiyel alan ile hem de mouse ile robotun uç noktasının koordinatları verilebilmektedir. Aşağıda hem 2 hem de 3 boyutta potansiyel alanın geliştirilmesi anlatılmaktadır.

5.4.1 Potansiyel Alan Kodunun Geliştirilmesi: 2 Boyut

5.4.1.1 Kısmi Türev Tanımı

x ve y'ye bağlı bir H fonksiyonunun, herhangi bir (x, y) noktasındaki x'e göre kısmi türevi;

$$\frac{\partial H}{\partial x} = \lim_{\Delta x \to 0} \frac{H(x + \Delta x, y) - H(x, y)}{\Delta x}$$

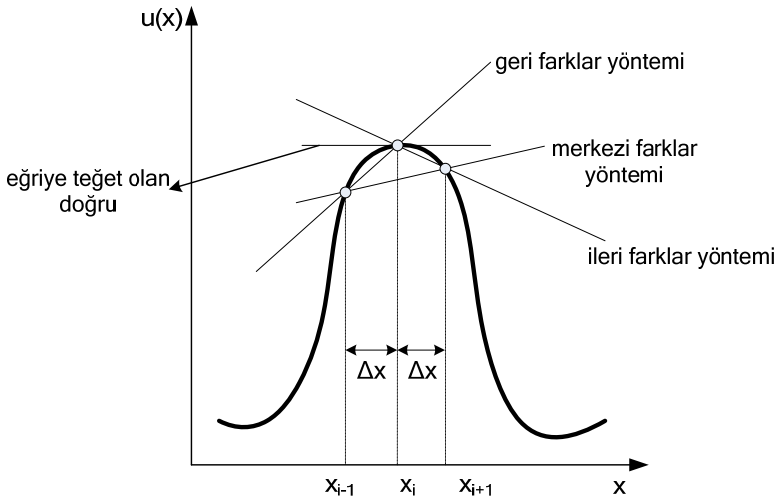
Benzer şekilde H fonksiyonunun y'ye göre kısmi türevi;

$$\frac{\partial H}{\partial y} = \lim_{\Delta y \to 0} \frac{H(x, y + \Delta y) - H(x, y)}{\Delta y}$$

5.4.1.2 Sonlu Farklar Yöntemi

Sonlu farklar yöntemi kısmi diferansiyel denklemlerinin nümerik olarak çözülmesi için kullanılmaktadır. Sonlu farklar yöntemi üçe ayrılmaktadır: *geri farklar yöntemi, merkezi farklar yöntemi ve ileri farklar yöntemi*. Sonlu farklar, geometrik yorumla Şekil 48'deki gibi izah edilebilir.

Şekil 48'de görüldüğü gibi x_i noktasındaki eğrinin tanjantı üç farklı şekilde izah edilebilir. Ayrıca bu tanjantın değeri x_i noktasındaki 1. dereceden türevi vermektedir. Kısmi türev tanımından yola çıkılarak elde edilen geri farklar, merkezi farklar ve ileri farklar yöntemleri aşağıdaki gibi tanımlanmaktadır.



Şekil 48. Sonlu farklar yönteminin geometrik yorumu

$$\left(\frac{\partial u}{\partial x}\right)_i \approx \frac{u_{i+1} - u_i}{\Delta x} \qquad \text{İleri Farklar}$$

$$\left(\frac{\partial u}{\partial x}\right)_i \approx \frac{u_i - u_{i-1}}{\Delta x} \qquad \text{Geri Farklar}$$

$$\left(\frac{\partial u}{\partial x}\right)_i \approx \frac{u_{i+1} - u_{i-1}}{2\Delta x} \qquad \text{Merkezi Farklar}$$

Yukarıdaki formüller 1. mertebeden türevler için yazılmıştır. Şimdi de bu üç yöntemin 2. mertebeden türev için çıkarılışını yazalım($\Delta x = h$). İleri farklar için;

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_i = \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}\right)_i$$

$$\approx \frac{1}{\Delta x}\left[\left(\frac{\Delta u}{\Delta x}\right)_{i+1} - \left(\frac{\Delta u}{\Delta x}\right)_i\right]$$

$$= \frac{1}{h}\left[\frac{u_{i+2} - u_{i+1}}{h} - \frac{u_{i+1} - u_i}{h}\right]$$

$$= \frac{u_{i+2} - 2u_{i+1} - u_i}{h^2} \qquad \text{İleri Farklar}$$

1. mertebeden geri farklar formülünden yararlanarak aşağıdaki gibi 2. mertebeden geri farklar formülünü çıkarabiliriz.

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_i \approx \frac{1}{\Delta x}\left[\left(\frac{\Delta u}{\Delta x}\right)_i - \left(\frac{\Delta u}{\Delta x}\right)_{i-1}\right]$$

$$= \frac{1}{h}\left[\frac{u_i - u_{i-1}}{h} - \frac{u_{i-1} - u_{i-2}}{h}\right]$$

$$= \frac{u_i - 2u_{i-1} + u_{i-2}}{h^2} \qquad \text{Geri Farklar}$$

1. mertebeden merkezi farklar formülünden yararlanarak aşağıdaki gibi 2. mertebeden merkezi farklar formülünü çıkarabiliriz.

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_i \approx \frac{1}{2\Delta x}\left[\left(\frac{\Delta u}{\Delta x}\right)_{i+1} - \left(\frac{\Delta u}{\Delta x}\right)_{i-1}\right]$$

$$= \frac{1}{\Delta x}\left\{\frac{1}{2}\left[\left(\frac{\Delta u}{\Delta x}\right)_{i+1} + \left(\frac{\Delta u}{\Delta x}\right)_i\right] - \frac{1}{2}\left[\left(\frac{\Delta u}{\Delta x}\right)_i + \left(\frac{\Delta u}{\Delta x}\right)_{i-1}\right]\right\}$$

$$= \frac{1}{\Delta x}\left[\left(\frac{\Delta u}{\Delta x}\right)_{i+1/2} - \left(\frac{\Delta u}{\Delta x}\right)_{i-1/2}\right]$$

$$= \frac{1}{h}\left[\frac{u_{i+1} - u_i}{h} - \frac{u_i - u_{i-1}}{h}\right]$$

$$= \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} \qquad \text{Merkezi Farklar}$$

Burada Δ fark anlamında kullanılmaktadır. Geometrik yorumdan bulunan kısmi türevler belli bir hata ihmal edilerek bulunmaktadır. Bu hatalara istenirse Taylor Serisinden bakılabilir. Fakat bizim için belli bir miktar hata olması problem oluşturmamakta, tam tersine hatayı azaltmak için yapılacak olan fazla iterasyonlar gerçek zamanlı çalışmayı engelleyebildiğinden tercih edilmemektedir.

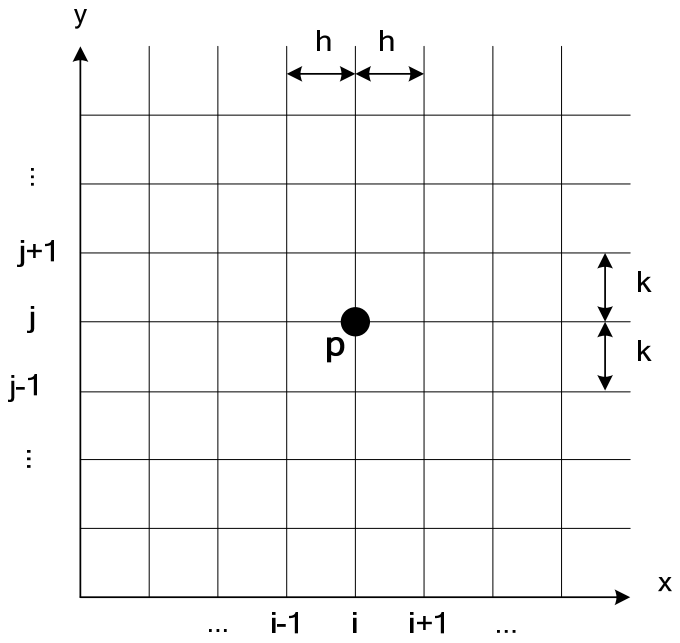
5.4.1.3 Sonlu Farklar Yönteminin Uygulanması

2. mertebeden kısmi diferansiyel denklemlerinin sonlu farklar metodu ile çözümünde, ilk önce çözüm uzayı bir ızgara sistemiyle (grid system) ifade edilir. Izgara sistemi olarak genellikle dikdörtgensel bir ızgara sistemi kullanılmaktadır (Şekil 49).

P noktasındaki 2. dereceden merkezi farkları x ve y'ye göre aşağıdaki gibi yazabiliriz($\Delta x = h$, $\Delta y = k$).

$$\frac{\partial^2 u_{i,j}}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}$$

$$\frac{\partial^2 u_{i,j}}{\partial y^2} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{k^2}$$



Şekil 49. P noktasının 2 boyutlu bir alanda gösterimi

2 Boyutlu Laplace Denklemini aşağıdaki gibi yazabiliriz.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

Merkezi farkları Laplace Denkleminde yerine koyarsak;

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{k^2} = 0$$

denklemini elde ederiz.

Eğer $h = k$ olursa denklemimiz aşağıdaki hali alır.

$$u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} = 0$$

veya

$$u_{i,j} = \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}}{4}$$

Yukarıdaki denkleme potansiyel alan denklemi de denilmektedir. Eğer alandaki sınır değerleri biliniyorsa buna Dirichlet sınır koşulları altında potansiyel alan denklemi denilmektedir.

5.4.1.4 Potansiyel Alanın Hesaplanması

Çalışma alanını çevreleyen grid noktaları ve engelleri temsil eden grid notalarına **sıfır,** hedef noktasına ise çok küçük bir değer, **-2^{124}**, verilerek hesaplamalar yapılır. Hesaplamalar, çalışma alanını çevreleyen grid noktaları ve engelleri temsil eden grid notalarına dışındaki noktalar için yapılır.

$$u^{m+1}{}_{i,j=\frac{1}{4}}(u^{m}{}_{i-1,j}+u^{m}{}_{i+1,j}+u^{m}{}_{i,j-1}+u^{m}{}_{i,j+1})\,;\quad i,j\in[0,N]$$

Burada N çalışma alanının x ve y yönlerindeki grid sayısıdır. m iterasyonu göstermektedir. İterasyon sayısı 200 ile 400 arasında herhangi bir değer alındığında iyi sonuçlar elde edilmektedir.

5.4.1.5 Başlangıç Noktasıyla Hedef Arasında Yolun Bulunması

Potansiyel alan grid üzerinde hesaplandıktan sonra, alan değerlerini kullanarak, başlangıç noktasından hedefe doğru parçacıkları çizilerek gidilir. Bu doğru parçacıklarının uzunlukları ne kadar küçükse o kadar yumuşak bir yol elde edilir.

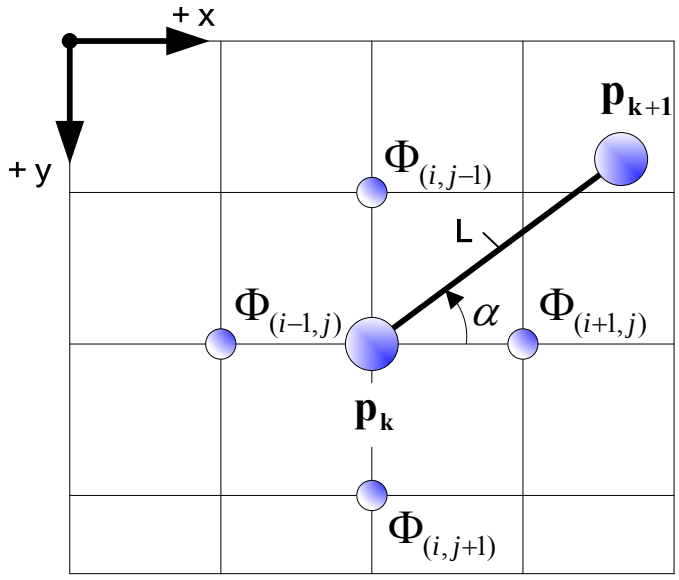
Şekil 50'de p_k noktasından p_{k+1} noktasına gidilmek istenmektedir. Bunun için noktanın etrafındaki grid değerlerini kullanarak

$$\alpha = a\tan 2\left(\frac{\Phi_{(i,j-1)}-\Phi_{(i,j+1)}}{\Phi_{(i-1,j)}-\Phi_{(i+1,j)}}\right)$$

açısı bulunur. Bu doğrultu kullanılarak, istenilen $\mathbf{L}$ vektörünün bileşenleri

$$L_x = \cos(\alpha).|\mathbf{L}|$$
$$L_y = \sin(\alpha).|\mathbf{L}|$$

ile elde edilir. Bulunan p_{k+1} noktasından yeni doğru parçasını bulmak için hedefe ulaşana kadar aynı işlem tekrarlanır.



Şekil 50. En küçük alan değeri doğrultusunu bulma

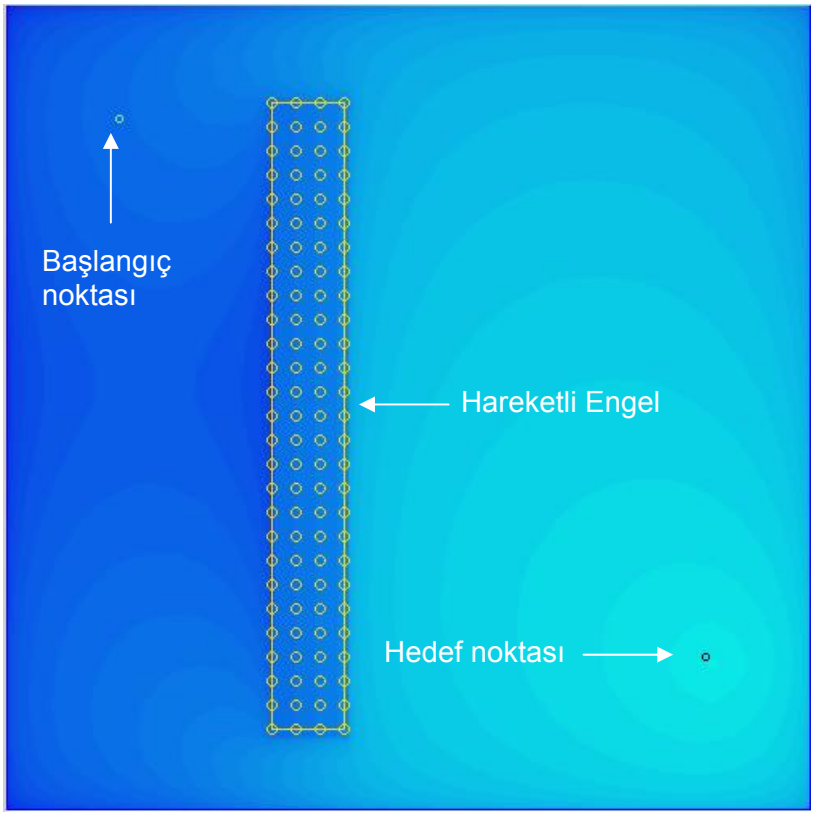
Burada dikkat edilirse doğrultuya ve $\mathbf{L}$ vektörüne bağlı olarak bulunan p_{k+1} noktası genellikle tam bir grid noktasına denk gelmeyecektir. Bu sebepten grid noktaları arasındaki potansiyel değerlerine de ihtiyaç vardır. Bu değerler lineer interpolasyon ile kolaylıkla bulunabilir. Böylece daha düzgün yollar elde edilebilir. Lineer interpolasyon formülü Şekil 57'de verilen 3 boyutlu formülde *z* yerine sıfır yazılmasıyla elde edilen formüldür.

5.4.1.6 Sonlu Farklar Yönteminin Uygulanması İçin Geliştirilmiş Küçük Bir Program

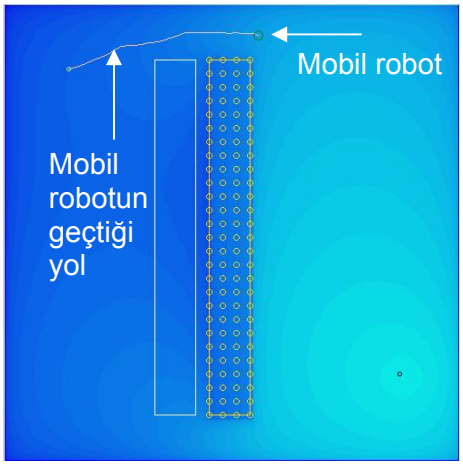
Dirichlet sınır koşulları altında potansiyel alan denklemi kullanılarak mobil robotun verilen başlangıç noktasından hedef noktasına ulaşmasını sağlayan küçük bir program geliştirdik. Bu program yazdığımız kodu denemek için iyi bir platform oluşturmaktadır. Daha sonra bu kodu bir sınıfa dönüştürerek RoboKol'a entegre ettik. Yazılan programın özellikleri aşağıdaki gibi sıralanabilir:

- Robotun hareket edeceği alan belirlenir.
- Robotun hareketi için başlangıç noktası ve bitiş noktası verilir.
- Engeller çizilir.
- Hareketli engeller için potansiyel alan gerçek zamanlı olarak hesaplanabilir.
- Program çalışırken yeni engel çizilebilir.
- Engellerin bitmap olarak renkli görüntüsü elde edilebilir.

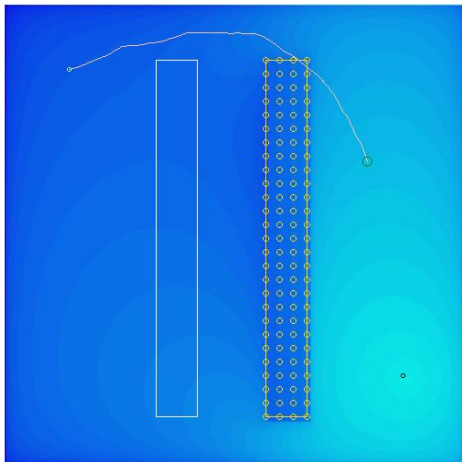
Örnek bir alan Şekil 51'de kare bir çalışma alanında başlangıç ve hedef noktası ve bir hareketli engel görülmektedir. Ayrıca potansiyel alanın bitmap görüntüsü çizilmiştir. Fakat ne yazık ki siyah-beyaz lazer çıktısında bu görüntü yeteri kadar net görünmemektedir.



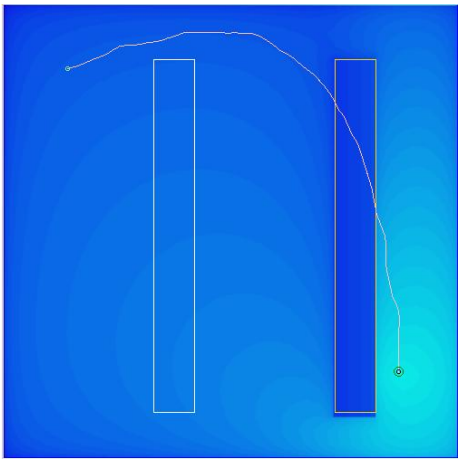
Şekil 51. Başlangıç durumu



a



b

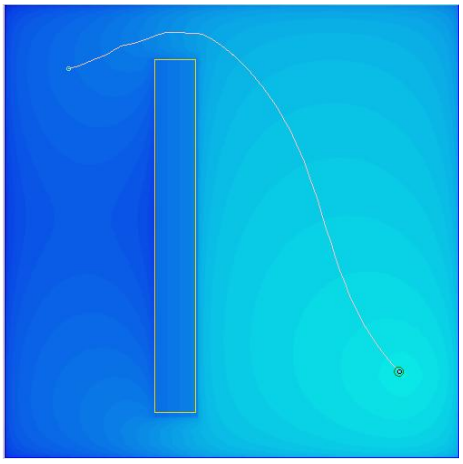


c

Şekil 52 a-c. Yukarıda bahsedilen hareketli engel olan alanda mobil robotun hareketi

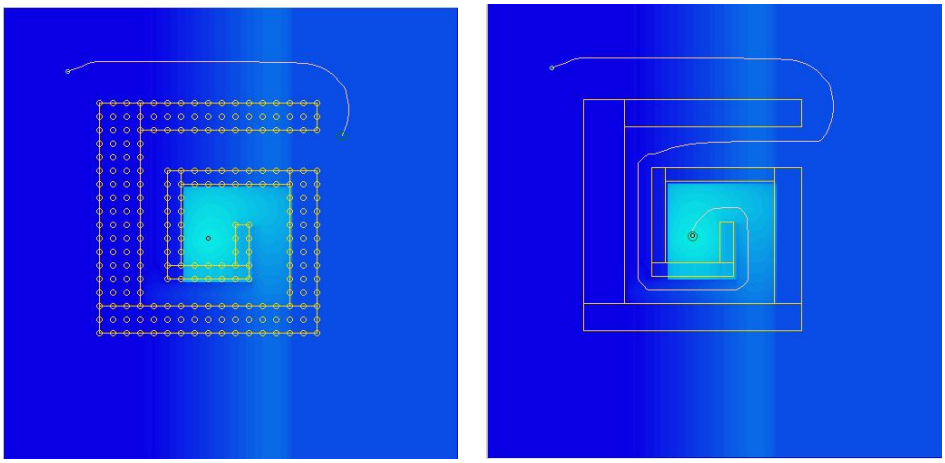
Şekil 52 a-c'de mobil robotun (şekilde bir küçük daire ile temsil edilmektedir) hareketli engel arasından hedefi bulması görülmektedir. Burada potansiyel alan devamlı olarak hesaplanmaktadır. Herhangi bir optimizasyon olmamasına rağmen gerçek zamanlı olarak çalışmaktadır. Şekil 52c'ye bakıldığında robotun gittiği yolun engelin içinden geçtiği zannedilebilir. Fakat engel sonradan, robot oradan geçtikten sonra oraya gelmiştir.

Şekil 53'de aynı engelin hareketsiz olduğu durumda mobil robotun hedefi bulması görülmektedir. Elde edilen bu iki yol karşılaştırıldığında aradaki fark bariz biçimde görülmektedir ki bu da hareketli engel durumunda yolun daha fazla uzamasıdır.



Şekil 53. Bir adet hareketsiz engel için mobil robotun hareketi

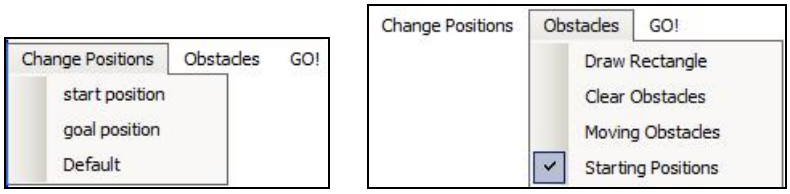
Diğer bir örnek olarak 8 adet hareketsiz engelden oluşan labirente benzer bir alanda mobil robotun hedefi bulması Şekil 54 a-b'de görülmektedir.



(a) (b)

Şekil 54 a-b. Mobil robotun 8 adet hareketsiz engel olduğu durumdaki hareketi

Programda üç adet menü vardır(Şekil 55). Bunlar **Change Positions**, **Obstacles**, **GO** olarak isimlendirilmiştir. **Change Positions** menüsünde 3 adet alt menü bulunmaktadır. Bu alt menülerden start position ile mobil robotun başlangıç noktası değiştirilebilmektedir. **Goal position** ile hedef noktasının yeri değiştirilebilmektedir. Default ile mobil robotun geçtiği yol silinmektedir. Çizilmiş hareketli ve hareketsiz engeller ilk çizildikleri yere geri dönmektedirler.



Şekil 55 Program menüleri

Obstacles menüsünde 4 adet alt menü bulunmaktadır. Bu alt menülerden Draw Rectangle ile engel çizilebilmektedir. Clear Obstacles ile hem engeller hem de mobil robotun geçtiği yol silinmektedir. Moving Obstacles ile engellerin hareket edip etmeyeceği belirlenmektedir, eğer Moving Obstacles işaretli ise engeller hareket etmekte aksi halde hareket etmemektedir. Starting Positions ile Hareketli engellerin ilk başlangıç yerlerinin gösterilip gösterilmeyeceği belirlenmektedir. **GO!** Menüsü ile de mobil robotun harekete başlaması sağlanmaktadır.

Bu programda geliştirilen 2 boyutlu potansiyel alan kodunu, Şekil 34'de görülen **pField2D** sınıfına dönüştürerek RoboKol'a entegre ettik.

5.4.2 Potansiyel Alan Kodunun Geliştirilmesi: 3 Boyut

5.4.2.1 Potansiyel Alanın Hesaplanması

3 boyutta potansiyel alanın hesaplanması 2 boyutlu alanın hesaplanmasına benzemektedir. 3 boyutlu denklem aşağıda çıkarılmıştır.

$$dx = dy = dz = h = L / N.$$

Burada L çalışma alanı olan küpün kenar uzunluğu, N ise bu uzunluğun bölüm sayısıdır. Düğüm potansiyelleri;

$$u_{i,j,k} = u(x_i, x_j, x_k); \quad i, j, k \in [0, N],$$

$$u_{i-1,j,k} + u_{i+1,j,k} + u_{i,j-1,k} + u_{i,j+1,k} + u_{i,j,k-1} + u_{i,j,k+1} - 6u_{i,j,k} = 0; \quad i, j, k \in [0, N],$$

$$\frac{\partial^2}{\partial x^2} u(x_i, y_j, z_k) \approx \frac{u_{i-1,j,k} - 2u_{i,j,k} + u_{i+1,j,k}}{h^2}$$

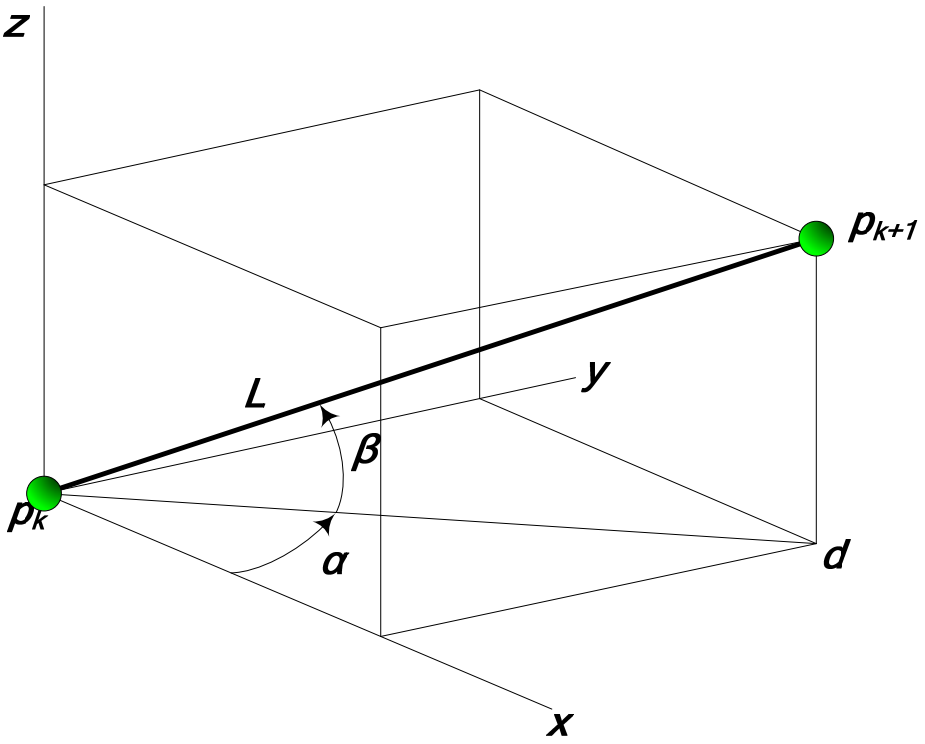
Gauss-Seidel metodunun kullanılmasıyla:

$$u^{m+1}{}_{i,j,k} = \frac{1}{6}\left(u^{m}{}_{i-1,j,k} + u^{m}{}_{i+1,j,k} + u^{m}{}_{i,j-1,k} + u^{m}{}_{i,j+1,k} + u^{m}{}_{i,j,k-1} + u^{m}{}_{i,j,k+1}\right); \quad i, j, k \in [0, N]$$

Burada N çalışma alanının x ve y yönlerindeki grid sayısıdır. m iterasyonu göstermektedir.

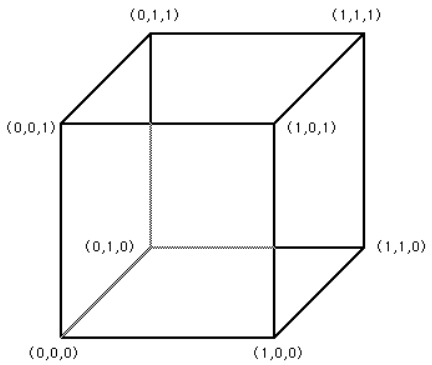
5.4.2.2 Başlangıç Noktasıyla Hedef Arasında Yolun Bulunması

Bölüm 5.4.1.5'de verilen doğrultu bulma burada da geçerlidir. Fakat burada bulunan α açısına ek olarak, Şekil 56'da görüldüğü gibi, yeni bulunan $p_k d$ doğrusunu ve z eksenindeki potansiyel alan farkı olan $p_{k+1} d$ doğrusunu kullanarak elde edilen β açısı gereklidir. Böylece **L** vektörüne bağlı olarak, p_k noktasından p_{k+1} noktasına 3 boyutta gidilebilir.



Şekil 56. 3 boyutlu uzayda yolun bir çizgisinin bulunması için gerekli parametreler

3 boyutta da doğrultuya ve $\mathbf{L}$ vektörüne bağlı olarak bulunan p_{k+1} noktası genellikle tam bir grid noktasına denk gelmeyecektir. Grid noktaları arasındaki herhangi bir noktadaki potansiyel değeri Şekil 57'de verilen 3 boyutlu lineer interpolasyon formülü ile kolaylıkla bulunabilir. Böylece daha düzgün yollar elde edilmektedir.



$$
\begin{aligned}
V_{xyz} = {} & V_{000}\,(1-x)\,(1-y)\,(1-z) + \\
& V_{100}\,x\,(1-y)\,(1-z) + \\
& V_{010}\,(1-x)\,y\,(1-z) + \\
& V_{001}\,(1-x)\,(1-y)\,z + \\
& V_{101}\,x\,(1-y)\,z + \\
& V_{011}\,(1-x)\,y\,z + \\
& V_{110}\,x\,y\,(1-z) + \\
& V_{111}\,x\,y\,z
\end{aligned}
$$

Şekil 57. 3 boyutlu uzayda interpolasyon [30]

3 boyut hesaplaması için Bölüm 4.2'de Şekil 34'de görülen **pField3D** sınıfı geliştirilmiştir. Bu sınıfın bir özelliği **pField2D** sınıfından **miras** (inheritance) alınmasıdır. Böylece bu özelliği de kullanmış olduk. Faydası ise Şekil 34'de görüldüğü gibi hem 2 hem de 3 boyutta kullanılan ortak değişkenleri tekrar tanımlanmasına gerek kalmamasıdır. Kullanılan metot isimleri aynı olabilir. İsmi aynı olan metotlarda hangi sınıfın nesnesi kullanılıyorsa otomatikman o sınıfın metodu seçilir.

Şekil 34'de ayrıca küçük bir sınıf olan **obstacle3D** sınıfı görülmektedir. Bu sınıf 3 boyutlu uzayda dikdörtgenler prizması şeklinde engeller oluşturmak için kullanılmaktadır.

6 Gereğinden Çok Serbestlik Dereceli Robotlar İçin “Serbest Bölgede” Kontrol Algoritmaları Geliştirilmesi

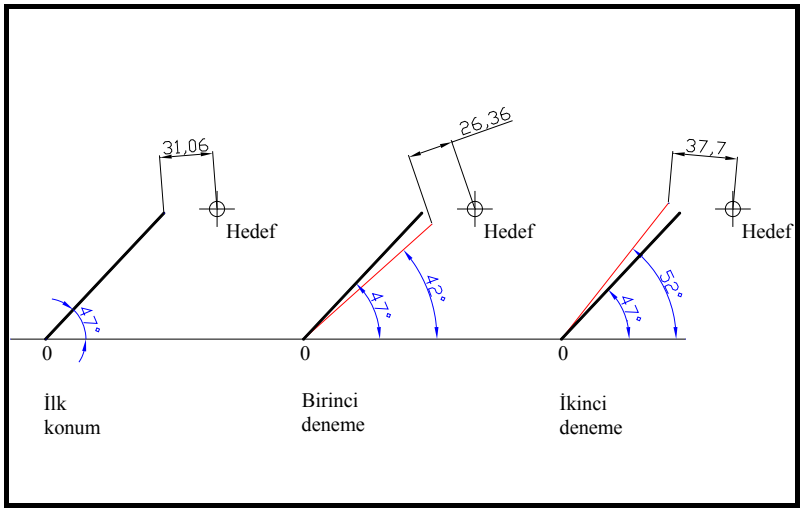
6.1 Serbest Çalışma Bölgesi İçin Geliştirilen Basit Bir Algoritma

Bu kısımda ilk olarak yüksek lisans öğrencimizin çoğunlukla üzerinde çalıştığı bir hareket planlama algoritması tanıtılacaktır. Algoritma belli noktalarda başarılı olmakla beraber, aşağıda da açıklanacağı gibi birçok noktada eksikliklere sahip olduğundan kullanılması pratik olarak pek faydalı değildir.

6.1.1 Algoritmanın Çalışması

Algoritmada robot uzvunun uç kısmının hedefe ulaşması sayısal bir yöntemle gerçekleşmektedir. Robot bütün uzuvlarını sırayla ve tek tek küçük bir açı değeri ile sağa ve sola çevirir. Her denemede uç kısmın hedefe olan uzaklığı hesaplanır ve ilgili diziye bu değer kaydedilir. Çevirme işlemlerinden sonra uzuv açıları eski durumuna getirilir. Bu deneme çevirmeleri ekranda gösterilmez. Ayrıca dizide, her değerle birlikte hangi uzvun hangi tarafa çevrildiği de kaydedilir. Daha sonra bir döngü ile dizideki robotun uç kısmının hedefe olan uzaklık değerleri arasından en küçük değer bulunur. Ek olarak en küçük değeri hangi uzvun aldığı ve o uzvun ne tarafa dönmesi gerektiği bilgisi de alınır.

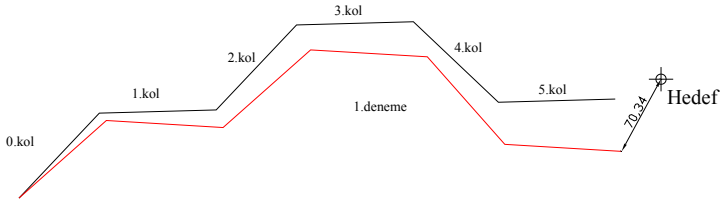
Tespit edilen uzuv tespit edilen yöne küçük bir açı değeri ile döndürülür ve bu dönüş ekrana yansıtılır. Robotun uç kısmı artık hedefe daha önceki durumuna göre biraz daha yakındır. Bu çizimin ardından tekrar aynı işlemler yapılır ve robot uzvu hedefe biraz daha yaklaşır. Robotun uzunluğu hedefe ulaşılması için kısa kaldığında ise robot düz bir çubuk şeklini almaktadır. Bu metot aşağıda adım adım anlatılacaktır.

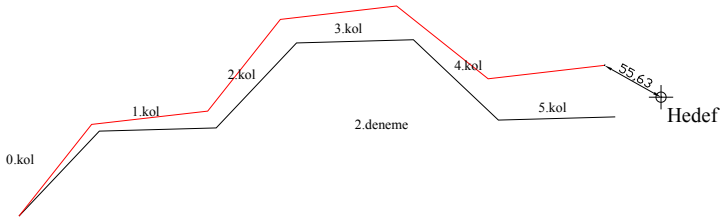


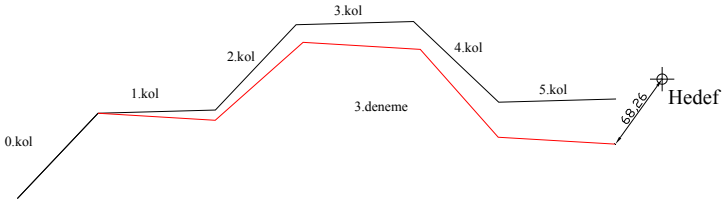
Şekil 58. Hedefe varmak için yapılan denemeler

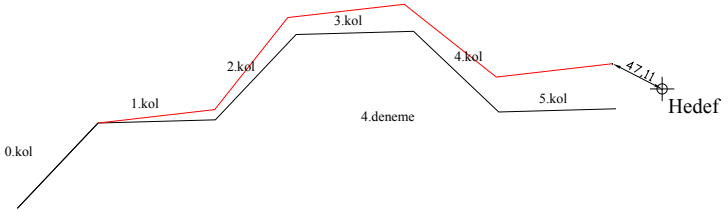
Şekil 58'de tek uzuvlu bir robot ve robotun hedefe olan uzaklığı görülmektedir. Uzuv ilk konumda *x* eksenine göre 47° açıyla durmaktadır. İlk konumda robotun hedefe uzaklığı 31,06 mm'dir. Program, birinci denemede uzvu (-) yönde 5° döndürmektedir ve hedefe olan uzaklık diziye 26,36 mm olarak kaydedilir. Program yine ikinci bir deneme yapar ve uzvu (+) yönde 5° döndürür ve diziye hedefe kalan uzaklık olan 37,7 mm'yi kaydeder. Sonra bir döngü ile kayıtta bulunan iki değerden en küçük olanını seçer. Uzvun (–) yönde döndürülmesi gerekmektedir. Uzuv (-) yönde 0,1° döndürülür ve döndürme işleminden hemen sonra uzuv ekrana çizdirilir. Denemenin 5° ile yapılması fakat döndürülmenin 0,1° ile yapılmasının sebebi, küçük dönme açıları ile ekranda akıcılığın sağlanmak istenmesindendir.

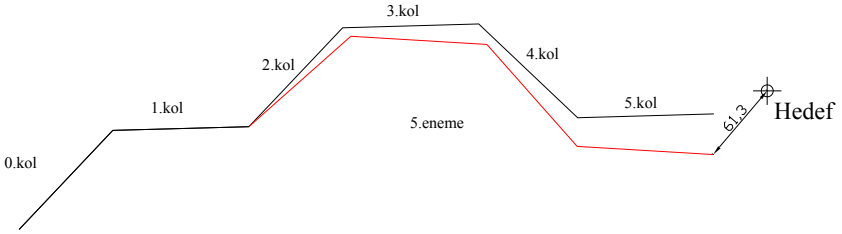
Şimdi aşağıda Şekil 59'da görülen 6 uzuvlu bir robotun denemeleri inceleyelim. 0. uzuv (-) ve (+) yönde deneme amaçlı döndürüldüğünde diğer bütün uzuvlar da aynı açı değerinde hep beraber döneceklerdir. 0. uzvun denemesi bittiğinde diğer uzuvların her biri (+) ve (-) yönde döndürülür ve uç kısmın hedefe olan uzaklıkları diziye kayıt yapılır. Deneme yapıldığındaki uzuvların alacağı şekil kırmızı renkle çizilmiştir. Aşağıdaki denemeye göre 10. denemede uç kısım hedefe en yakın konumdadır. 10. deneme 4. uzuv (+) yönde döndürülürse robotun tek hamlede hedefe en çok yaklaşacağını göstermektedir. Daha önceden de belirtildiği gibi gerçek dönme miktarı çok küçük alınır böylelikle robotun hareketi akıcı olur.

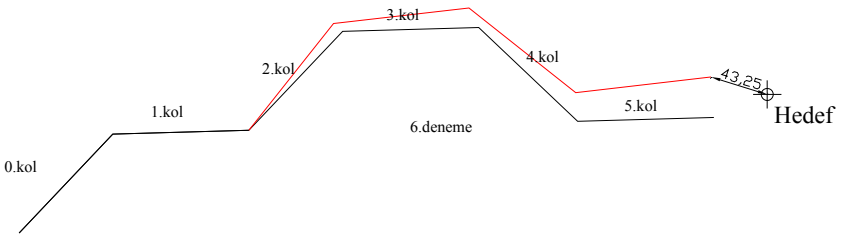

3.kol
2.kol
4.kol
1.kol
5.kol
1.deneme
70,34
Hedef
0.kol

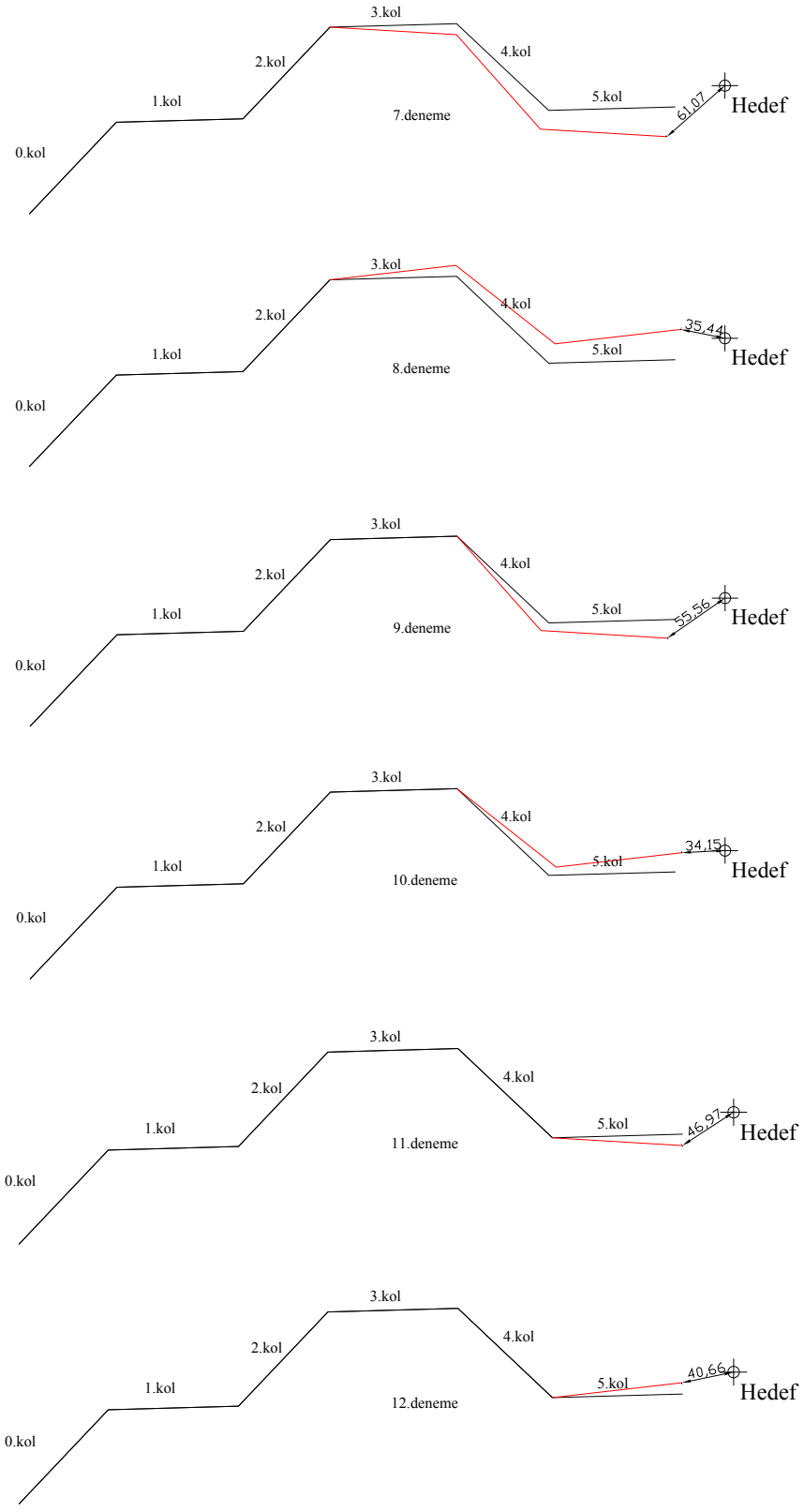

3.kol
2.kol
4.kol
55,63
1.kol
5.kol
2.deneme
Hedef
0.kol


3.kol
2.kol
4.kol
1.kol
5.kol
3.deneme
68,26
Hedef
0.kol

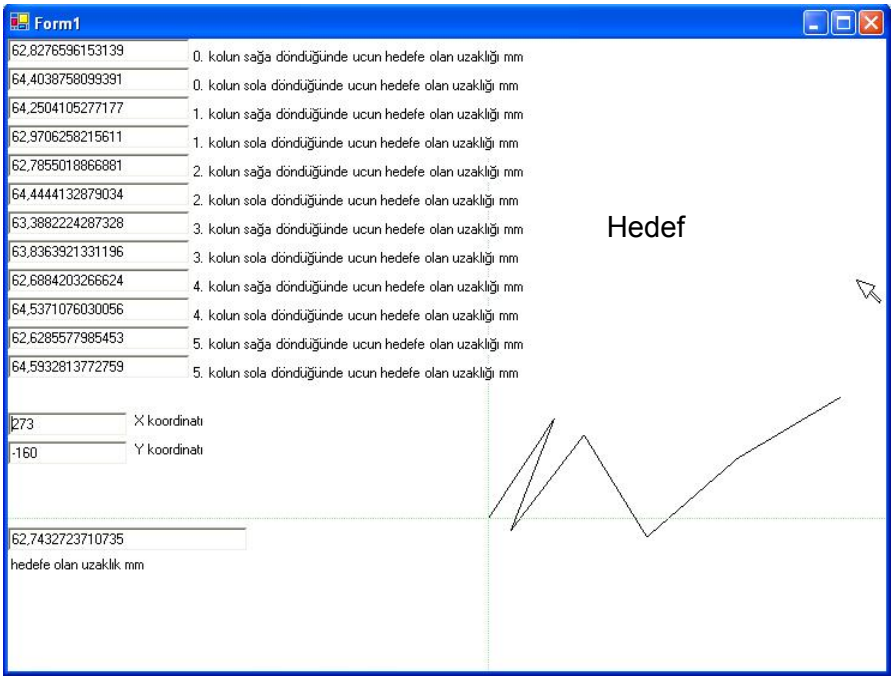

3.kol
2.kol
4.kol
47,11
1.kol
5.kol
4.deneme
Hedef
0.kol


3.kol
2.kol
4.kol
1.kol
5.kol
5.eneme
51,3
Hedef
0.kol


3.kol
2.kol
4.kol
43,25
1.kol
5.kol
6.deneme
Hedef
0.kol



Şekil 59 Uzuvların hangisinin robotu hedefe yaklaştırdığıyla ilgili denemeler



Şekil 60. Serbest çalışma bölgesi için geliştirilen programın arayüzü

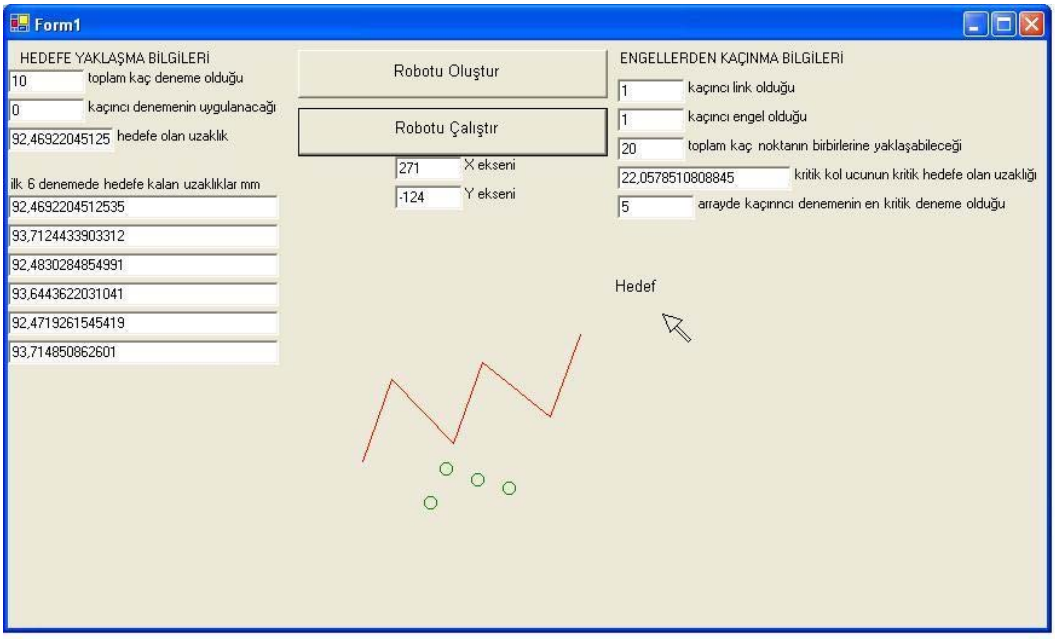
Şekil 60'da programımızın arayüzü görülmektedir. Hedef olarak fare imleci seçilmiştir. Fareyi ekranda gezdirdikçe farenin *x* ve *y* koordinatı yazı kutuları aracılığı ile görüntülenmektedir. Hedefin yeri farenin koordinatları olarak tanımlanmıştır. Hedefe olan uzaklık da görüntülenmektedir. Ayrıca programlama yapılırken yardımcı olması için her denemede her uzvun hedefe olan uzaklıkları da görüntülenmektedir. Bu programda uzuvlar sıfırdan başlanarak sayılmaktadır. Program her açıldığında sabit olarak 5 uzuvlu bir robot ekrana gelmektedir. Uzuv uzunlukları sabittir.

Yukarıda bahsedilen programdan sonra yazılan programda robot ilk önce fare yardımı ile çizilmektedir. Robot istenildiği kadar çok uzva sahip olabilir ve her uzvun uzunluğu değişik değerler alabilir. Çizilen her uzuv başka bir dizi kullanılarak kayıt yapılır.

Bu programda daha önceki programda bulunan hedefe gitme fonksiyonunun yanında bir de engellerden kaçınma fonksiyonu vardır. Program yine her uzuv için deneme yapar ve uygun olan uzvu uygun olan yöne doğru çevirir.

Şekil 61'de görüldüğü gibi pencerenin sol tarafında "hedefe yaklaşma bilgileri" sağ tarafında ise "engellerden kaçınma bilgileri" yer almaktadır. Programdan alınan anlık görüntüde fare ile çizilmiş 5 uzuvlu bir robot ve 4 adet engel bulunmaktadır. İlk çalıştırıldığında farenin sol tuşuna basılı tutulur ve sürüklenir istenilen uzunlukta uzuv oluştuğunda tuş bırakılır ve bundan sonrada her uzuv için aynı işlem yapılır. Yılan şeklindeki robot çizildikten sonra

“robotu oluştur” düğmesine basılır. Fare engel koyulmak istenen yere getirilir ve sol tuşa basılır. Tuşa basılan yerin koordinatında küçük yeşil bir daire çizilir. Bu dairenin merkez koordinatı engel olarak diziye kaydedilir. “Robotu çalıştır” düğmesine basılarak robotun hareketi başlatılır. Artık robot fareyi takip etmektedir.



Şekil 61. Engelleri içeren çalışma bölgesi için geliştirilen programın arayüzü

Programda yer alan bilgi alanlarının ne olduğu aşağıda verilmiştir.

Hedefe Yaklaşma Bilgileri:

- *Toplam kaç deneme olduğu:* Hedefe yaklaşma denemelerinin toplam kaç defa yapıldığı bilgisidir. Her uzuv ile iki deneme yapılır. Burada toplam 5 uzuv olduğu için toplam 10 deneme yapılmıştır.
- *Kaçıncı denemenin uygulanacağı:* Hedefe en çok yaklaşma sağlayan denemenin kaç numaralı deneme olduğunu gösterir. Şekil 4’de görüldüğü gibi 0. yani ilk uzuv hedefe en çok yaklaşma sağlamaktadır.
- *Hedefe olan uzaklık:* Robotun ucunun hedefe olan uzaklığını verir. Fare bir yere sürüklendiğinde robot hedefe yaklaşırken hedefe olan uzaklığın azaldığı ve hedefe ulaşıldığında sıfıra yaklaştığı görülmektedir.
- *İlk 6 denemede hedefe kalan uzaklıklar:* Programlama esnasında yardımcı bilgi olması ve olası hataların görülmesi amacıyla ilk 6 denemenin robotu hedefe ne kadar yaklaştırdığını gösterir.

- *x ve y ekseni:* Farenin o andaki koordinatlarını verir. Sıfır koordinatı her zaman robotun başlangıç noktasıdır. Robot nereden çizilmeğe başlanırsa orijin orası olmaktadır.

Engellerden kaçınma algoritması mantık olarak bundan önce bahsedilen algoritmaya benzemektedir. Bir uzuv bir engelden kaçınacaksa program bütün uzuvları teker teker sağa ve sola hareket ettirir ve hangi uzvun hangi tarafa döndürüldüğünde engelden en çok uzaklaşma sağladığını diziye kaydeder ve uzvu bulunan yönde çevirir. Böylelikle uzuv engelden uzaklaşmış olur. Burada önemli bir nokta hangi uzvun hangi engelden kaçınması gerektiğidir. Engele çok uzak bir uzvun engelden kaçınması gereksiz olduğundan ilk önce uzuvların her birinin ucunun her bir engele olan uzaklığı hesaplanır.

Örnek olarak iki uzuvlu bir robot ve iki engel olduğunu farz edelim. Her uzvun bir adet uç kısmı bulunmaktadır. İki uzvun iki adet ucu vardır. Birinci uzvun ucunun birinci ve ikinci engele uzaklığı bulunur ve kayıt edilir. İkinci uzvun ucunun da birinci ve ikinci engele olan uzaklığı bulunur. Bir döngü ile en küçük değer bulunur. Diyelim ki ikinci uzvun ucu birinci engele diğerlerinden daha yakın, o zaman bütün uzuvların denemesi yapıldığında her seferinde ikinci uzvun ucunun birinci engelde uzaklaşması dikkate alınacaktır. Anlık görüntüde 0,1,2,3 ve 4. uzuvlar vardır ve soldan sağa doğru 0,1,2 ve 3. engeller vardır. Robotun engellere en yakın olan yeri 1. uzvun ucudur ve 1. engele yakındır. Bu bilgi edinildikten sonra artık program uzuvları sağa ve sola çevirme denemeleri yapabilir. Engelden uzaklaşma için en uygun uzuv bulunur ve çevirme gerçekleştirilir.

Engellerden Kaçınma Bilgileri:

- Kaçıncı uzuv olduğu: En kritik yakınlaşmanın hangi uzuvda meydana geldiğini belirtmektedir.
- Kaçıncı engel olduğu: En kritik uzvun yaklaştığı engel numarasını vermektedir.
- Toplam kaç noktanın birbirine yaklaşabileceği: Uzuv sayısı ile engel sayısının çarpımını verir. Programlamaya ve çalışması sırasında gözlemlemeye yardımcı bir bilgidir.
- Kritik uzuv ucunun kritik engele olan uzaklığı: İki nokta arasındaki mesafeyi verir.
- Dizide kaçıncı denemenin en kritik deneme olduğu: Programlamaya yardımcı bir bilgidir. Denemeler diziye bir döngü ile kayıt yapıldığı için dizi numarası hangi uzvun denemesi olduğu bilgisi için çok önemlidir.

6.1.2 Algoritmayla İlgili Genel Değerlendirme

Robot hedefe çok güzel bir şekilde ulaşabiliyor. Uzuvlar hedefe doğru ilerlerken küçük titreşimler meydana geliyor. Hedefe 10mm kala ilerleme daha da azaltılıyor ve hedefe daha az titreşimle ulaşılma sağlanıyor. Uzuvlar ileri ve geri açılma ve kapanma işlemlerini yapabiliyor. Fakat uzuvlara açı sınırlaması getirilmediği için uzuvlar birbirinin üzerinden geçebiliyor. Farklı uzunluklarda uzuvlar çizilirse uzun uzuvlar öncelikli olarak hareket ediyor. Bunun sebebi deneme gerçekleşirken aynı açı değeri için uzun uzvun daha çok yaklaşma sağlamasıdır.

Engellerden kaçınma da ise kısmen başarı ve büyük ölçüde başarısızlık gözlemleniyor. Uzuvlar kendisini engelden uzaklaştırabiliyor. Fakat engelden uzaklaşırken genelde hedeften de uzaklaşma meydana geliyor. Tekrar hedefe yönelen uzuvlar engelden tekrar kaçınmaya çalışıyor ve bir müddet sonra bütün denemeler hem hedefe ulaşma için hem de engelden kaçmak için aynı uzvun hareket ettirilmesi gerektiğini gösteriyor. Yani bir tek uzuv bir tarafa dönerek hedefe ulaşmaya çalışırken, diğer tarafa dönerek de engelden kaçmaya çalışıyor. Böylelikle bir kilitlenme meydana geliyor.

Deneme yanılma ile robot yolunu bulabilmektedir fakat engellerden kaçınamamaktadır. Ayrıca hedefe giderken istenmeyen titreşimler oluşmaktadır. Bu dezavantajlar göz önüne alınarak bu algoritma üzerinde daha fazla çalışılmaması gerektiğine karar verilmiştir.

6.2 2 Boyutlu Serbest Çalışma Bölgesi İçin Geliştirilen Verilen Yörüngeyi Takip Etme Algoritması

RoboKol programının hedefi bilindiği gibi gereğinden çok serbestlik dereceli robot kolları için kinematik kontrol algoritmaları geliştirmektir. Program belli bir olgunluğa eriştikten sonra algoritma geliştirmeye başlanmıştır. Program sürekli olarak modifiye edilmekte ve algoritmalar geliştirilmektedir.

Özet olarak hemen belirtmek gerekirse engellerin olmadığı alanda robot uç noktasının verilen yörüngesini, robotun kaç tane uzvu olursa olsun, takip edebilecek algoritmayı geliştirmiş bulunuyoruz. Bu yörünge sadece ileri doğru değil geriye doğru da olabilir. Bundan önceki

çalışmalarımızdan elde ettiğimiz tecrübelere de dayanarak bu algoritmanın avantajlarını şöyle sıralayabiliriz:

- Basitlik
- Gerçek zamanlı çalışabilme
- Eklenebilirlik: Algoritmaya daha sonradan istenen bazı ek özellikler kolayca monte edilebilir. Bu bir anlamda lineer sistemlerdeki süperpozisyon prensibine benzetilebilir.
- Engeller olma durumunda engellerden kaçınmaya yardımcı olma
- Robotu oluşturan uzuv sayısından bağımsız çalışma

Şekil 62'de dört uzuvlu seri bir manipülatör görülmektedir. Buradaki dört uzuv, anlatımda basitlik için seçilmiştir. Yukarıda bahsettiğimiz gibi algoritmada uzuv sınırlaması yoktur. Bilindiği gibi, robotun uç noktasının ulaşması gereken koordinatları verildiğinde bu koordinatlara ulaşmak için gereken uzuv açılarını hesaplamaya "ters kinematik" denmektedir. Bizim yapmaya çalıştığımız da gereğinden çok serbestlik dereceli robotlar için sayısal ters kinematik çözümler geliştirmektir.

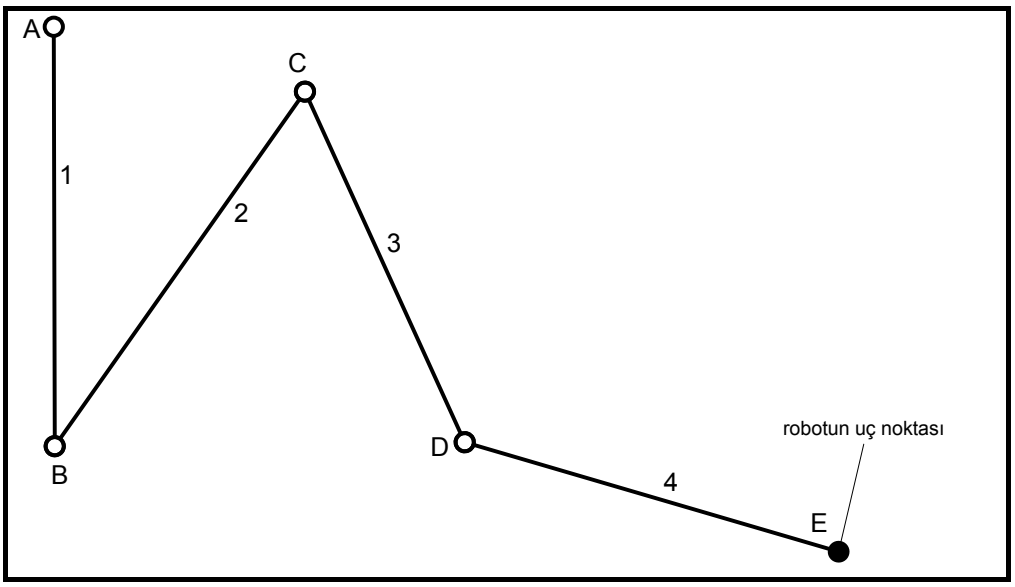
Şekil 63'de aynı manipülatörün iki konfigürasyonu çizilmiştir. Manipülatör birinci konfigürasyonda iken ikinci konfigürasyona ulaşması istenmektedir. Başka bir deyişle, manipülatörün uç noktası olan E noktasının E' noktasına gelmesi istenmektedir. Bu işlem gerçekleşirken de uzuvların mümkün olduğunca az hareket etmesi tercih edilmektedir.

Şekil 63'e tekrar bakarak, E' noktasından D noktasına bir E'D vektörü çizelim. Bu vektör E' noktasında başlayıp D noktasında bitmektedir. Bu vektörü kendi büyüklüğüne bölersek E' noktasında başlayan ve E'D doğrultusunda olan bir birim vektör elde ederiz. Şimdi bu birim vektörü 4 nolu uzvun uzunluğuyla yani DE ile çarparsak D' noktasını elde ederiz. Böylece 4 nolu uzuv DE konumundan D'E' olan yeni konumuna gelmiş olur.

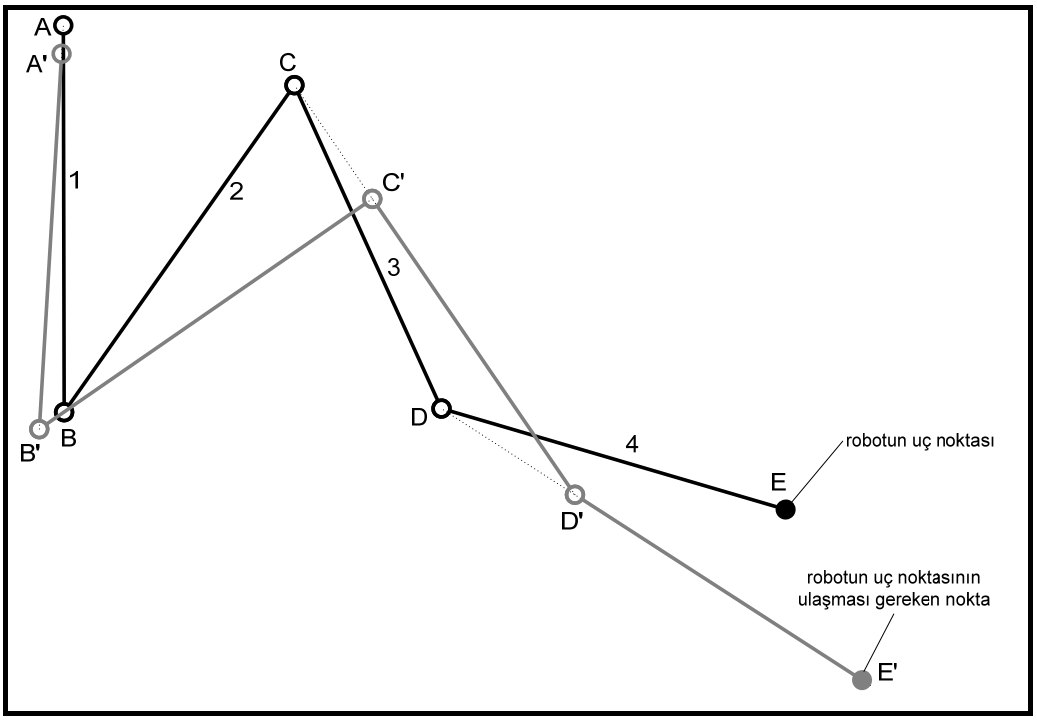
Benzer şekilde D'C arası çizilip D'C birim vektörü kullanılarak C' noktası, C'B arası çizilip C'B birim vektörü kullanılarak B' noktası ve B'A arası çizilip B'A birim vektörü kullanılarak A' noktası belirlendiğinde manipülatörün yeni konfigürasyonu ortaya çıkmış olur.

Uzuvlar birbirine dönel mafsallarla bağlı rijit cisimler olduğundan ve manipülatörün temeli yani A noktası sabit olduğundan henüz işimiz bitmiş değildir. Çünkü Şekil 63'de rahatlıkla görülebileceği gibi A noktası A' noktasına kaymıştır. Bu kaymayı geçici bir süre için dikkate

almaz ve manipülatörü fare ile hareket ettirirsek çok yumuşak hareketler elde ettiğimizi görürüz.



Şekil 62. Dört uzuvlu seri manipülatör



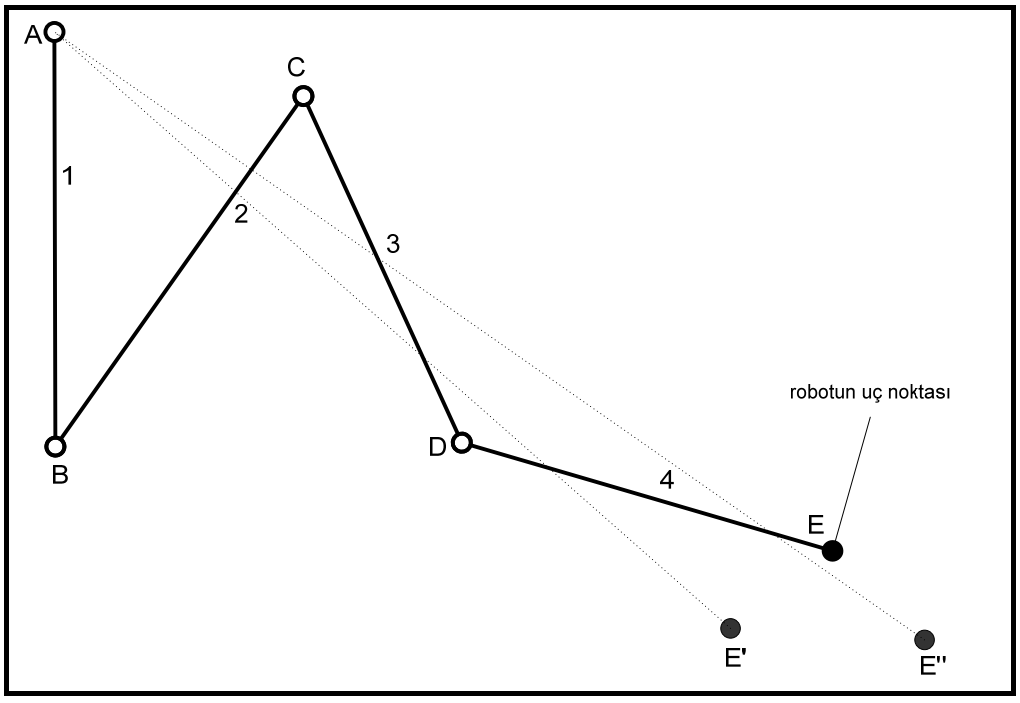
Şekil 63. Dört uzuvlu seri manipülatörün iki ayrı konfigürasyonu

Biz uzun bir zaman manipülatörü bu halde kullandık. Gerçi ilk baştan beri bu problemi bir şekilde çözeceğimizi düşünüyorduk, fakat bunu uygulayıp sonucu görmek hemen olmadı. Bulduğumuz çözüm şu şekilde açıklanabilir: Manipülatörün uç noktasıyla temel noktasının

yerini değiştiriyoruz. Yani A' noktasını manipülatörün uç noktası ve E' noktasını manipülatörün temel noktası kabul ediyoruz. Şimdiki amacımız bu ters dönmüş manipülatör ile A noktasına ulaşmak. Biraz önce açıkladığımız metot kullanarak A noktasına ulaşıyoruz. Bu sefer de manipülatörün şimdiki temel noktası olan E' noktası kayıyor. Manipülatörü eski haline getirip algoritmayı tekrar uyguluyoruz.

Manipülatörün uç ve temel noktalarını sırayla değiştirmeyi hata sıfır olana kadar, yani robotun temeli A noktasına ucu da E' noktasına gelene kadar devam ediyoruz. Buradaki can alıcı soru şu: Acaba kaç iterasyondan sonra bu yerleşme gerçekleşiyor? Şanslı olduğumuzu söyleyebiliriz. Her defasında hata gittikçe azalarak genelde 3-5 en çok da 10-15 defada yerleşme gerçekleşiyor.

Yukarıda her ne kadar yumuşak hareketler elde ettiğimizi söylesek de önemli bir sorun henüz çözüm beklemektedir. Manipülatörün uzuvları bazı durumlarda iç içe geçerek manipülatör uzuvları karmakarışık bir hal almaktadır. Hedef noktası manipülatörün temelinden uzaklaşırken hiçbir problem ortaya çıkmamaktadır. Gözlemlerimiz sonucu bu problemin, manipülatörün ucu manipülatörün temeline doğru giderken ortaya çıktığını fark ettik. Bu durumda başka bir strateji belirlememiz gerekmektedir.



Şekil 64. "İleri" ve "geri" kavramlarının belirlenmesi

Şekil 64'e bakarak robotun uç noktasının E' ve E'' noktalarına gelmesinin istendiğini farz edelim. "İleri" ve "geri" kavramlarının tanımlanmasında E'A ve EA doğrularını kullanıyoruz.

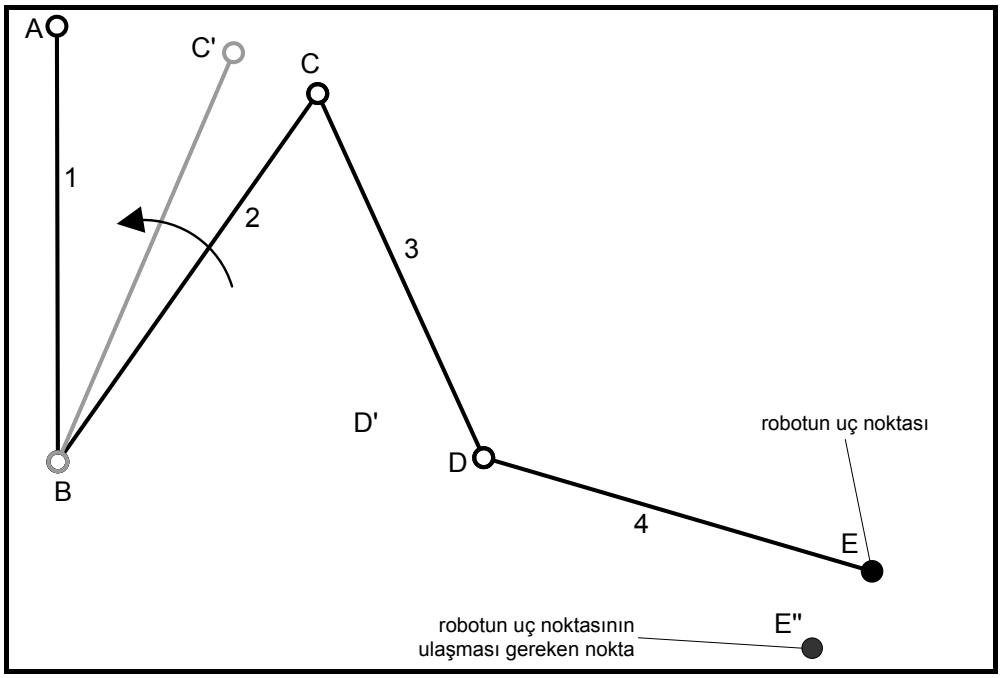
- E’A doğrusunun uzunluğu, EA doğrusunun uzunluğundan daha kısa olduğu için robotun E’ noktasına olan hareketini “geriye doğru” olarak tanımlıyoruz.

- E’’A doğrusunun uzunluğu, EA doğrusunun uzunluğundan daha uzun olduğu için robotun E’’ noktasına olan hareketini “ileriye doğru” olarak tanımlıyoruz.

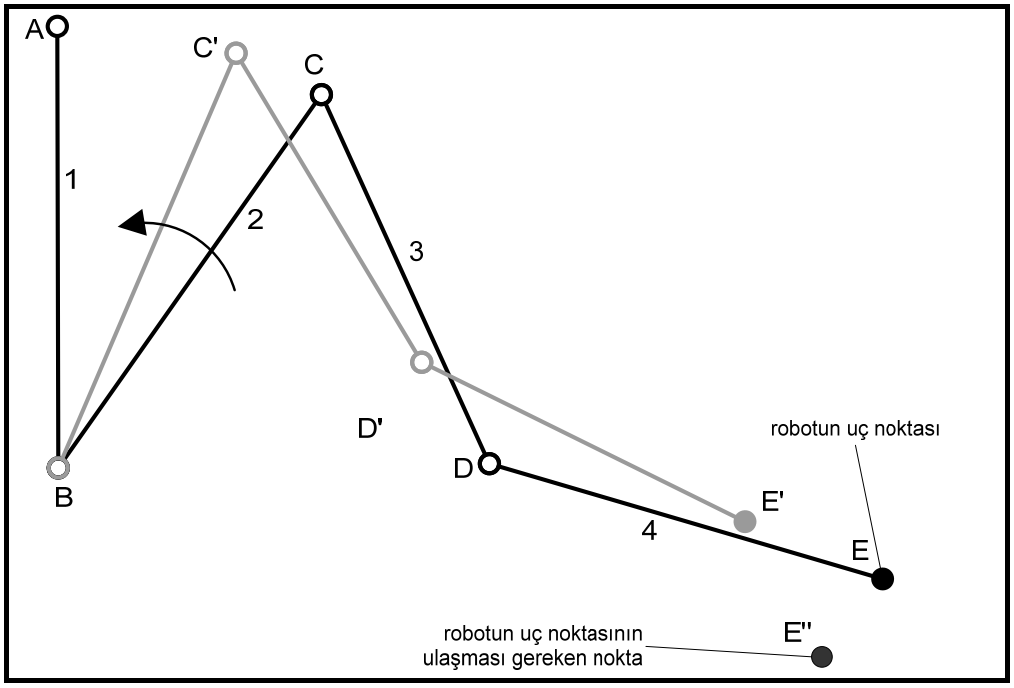
Şekil 65’de görülen manipülatörün “geriye doğru” hareketini inceleyelim. 1 nolu uzvun hareket sınırlarına eriştiğini ve artık harekete dahil edilmediğini düşünelim. Bu durumda bir sonraki uzuv, yani 2 nolu uzuv, saat tersi yönde önceden belirlenen bir miktar döndürülür ve 2 nolu uzuv üzerindeki C noktası C’ noktasına gelir. Manipülatörün 3 nolu uzvu hala C noktası üzerindedir.

Şimdi bu C noktasını robotun uç noktası, E noktasını da robotun temeli olarak alıp “ileri doğru hareket” algoritması “bir” defa uygulanır. Bu durumda Şekil 66’da görüldüğü gibi 3 nolu uzvun C noktası da C’ noktasına gelir. Fakat önceden de bildiğimiz gibi bir defa uygulanan “ileri doğru hareket” algoritmasında temel noktası kayacaktır ve E noktası E’ noktasına gelecektir.

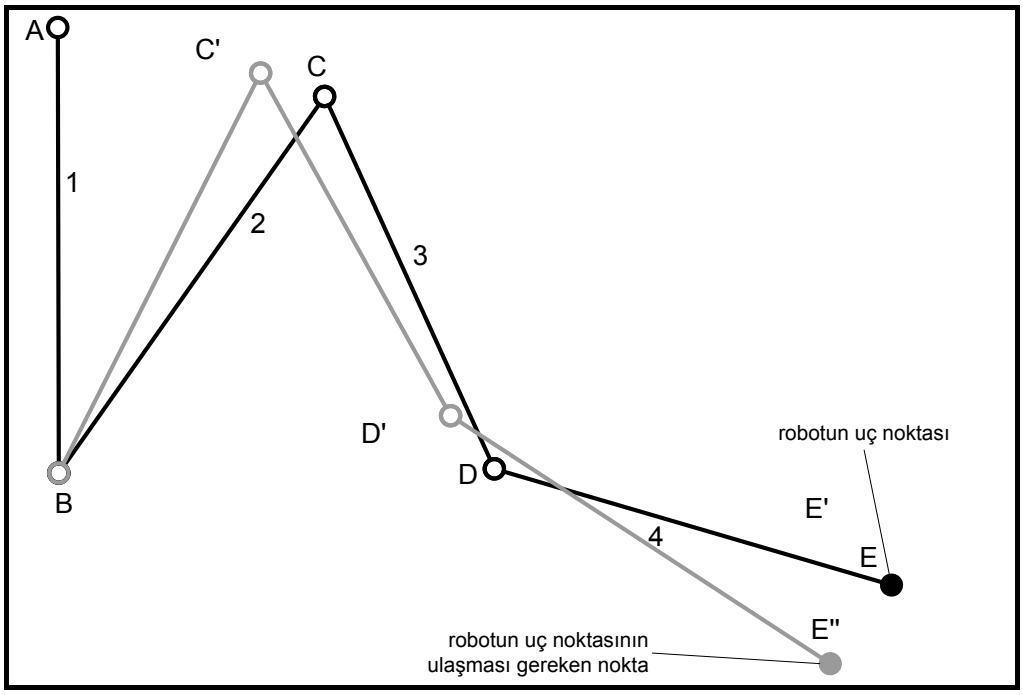
Şekil 66’ya tekrar baktığımızda manipülatör artık öyle bir pozisyona gelmiştir ki bu pozisyon bizim çok iyi bildiğimiz “ileri gitme” pozisyonudur. “İleri gitme” algoritması uygulanarak manipülatör Şekil 67’de görülen E’’ konumuna gelir. Böylece “geriye gitmede” 3 ve 4 nolu uzuvlar aşırı derecede birbirine yakınlaşmaz ve mafsal limitleri ihlal edilmez.



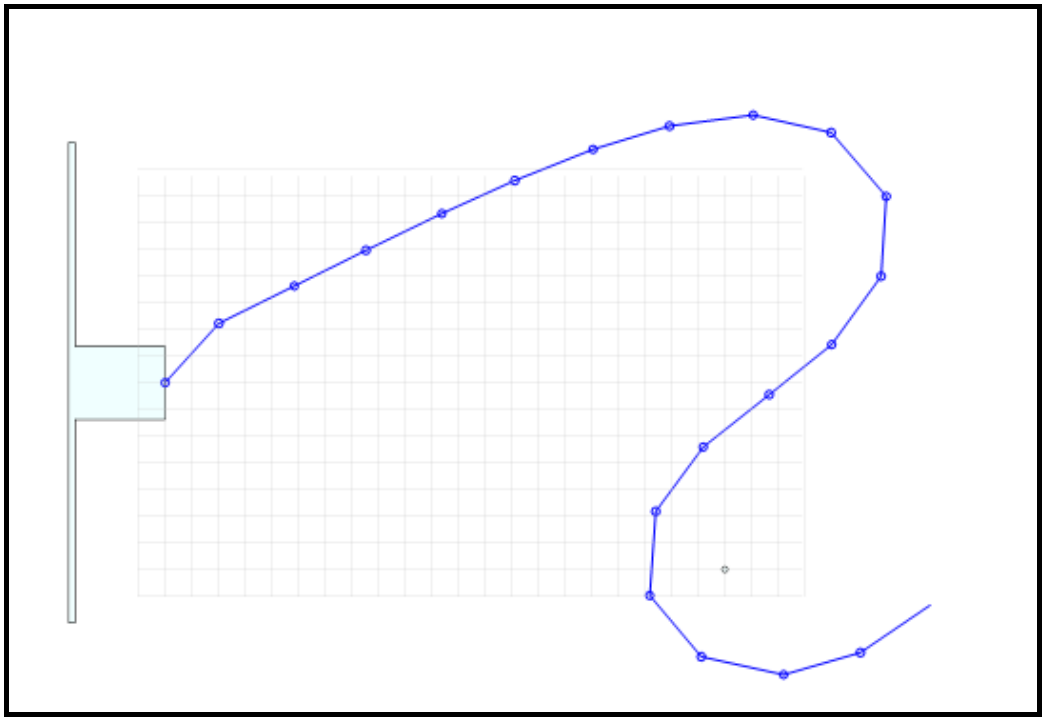
Şekil 65. 2 nolu uzvun geriye alınması



Şekil 66. 2 nolu uzvun geriye alınması



Şekil 67. "İleriye doğru" hareketle robotun istenilen konuma varması

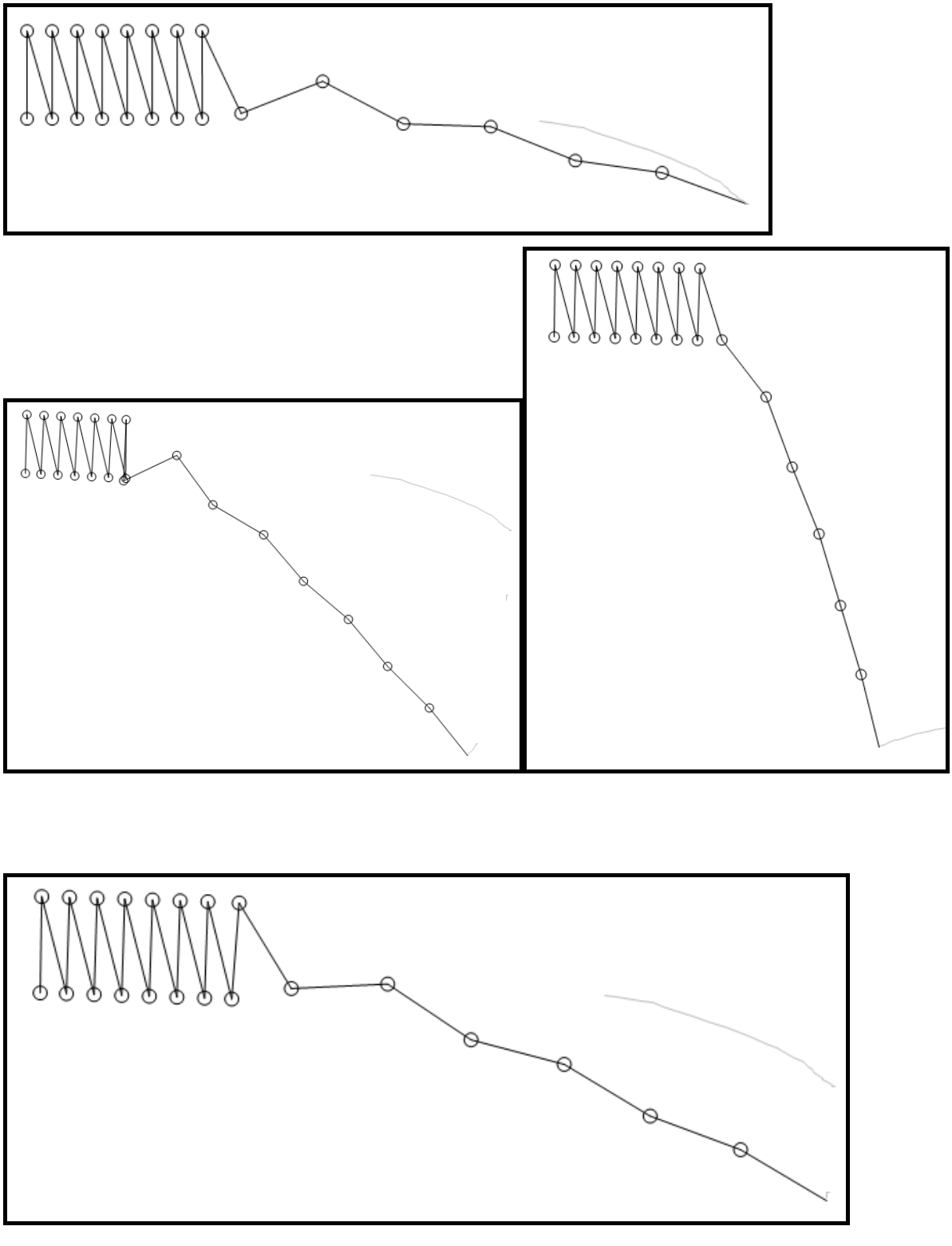


Şekil 68. Manipülatörün ideal hareketi

Bu metotta hareketler ileriye ve geriye doğru yumuşak bir şekilde gerçekleşmekte, Şekil 68'de görülen manipülatörün "tabii kıvrımları" korunmakta ve "ani değişmelere" izin verilmemektedir.

Bu anlattığımız "geriye gitme" algoritması hemen ortaya çıkmamıştır. En az 6-7 değişik metot denenmiş ve bu denemelerden elde edilen tecrübelerle geliştirilmiştir. Bu kadar çok deneme yapacağımızı ve her defasında ilginç fakat bizi tatmin etmeyen sonuçlar alacağımızı hiç düşünmemiştik. Denediğimiz metotlarda birinden örnek olarak bahsedelim.

Şekil 69'da fare hareketine bağlı olarak manipülatörün aldığı şekiller görülmektedir. Buradaki manipülatör ileri giderken "ileri gitme" algoritmasını kullanmakta, "geriye gitmede" ise manipülatör önce akordeon şeklindeki sıkışmış ilk haline gelmekte ve oradan ileriye gitmektedir. Tabii kullanıcı bu geriye gitme hareketini görmemektedir. Bu metodun dezavantajı, Şekil 69'dan rahatça çıkarılabileceği gibi, manipülatörün daima ileri gitme durumunda oluşan kıvrımlarının geriye gitme anında aniden kaybolup manipülatörün daima dik bir şekilde hareket etmesidir.

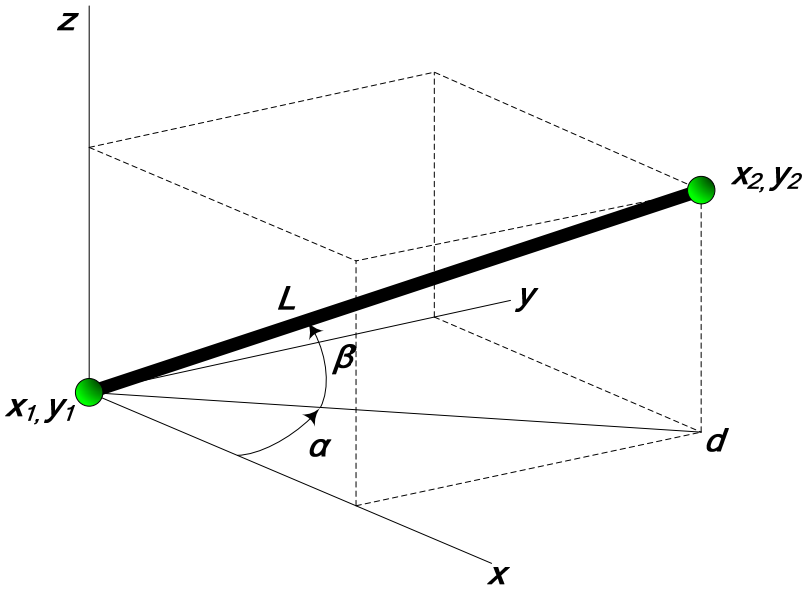


Şekil 69. Başka bir “geriye doğru” gitme algoritmasının uygulaması

6.2.1 2 Boyutta Serbest Çalışma Bölgesi İçin Geliştirilen Algoritma İçin Örnek

Yukarıda verilen algoritma için örnek Bölüm 8'de Robotların Çalıştırılması ve Deneylerin Yapılması anlatılırken verilecektir.

6.3 3 Boyutlu Serbest Çalışma Bölgesi İçin Geliştirilen Verilen Yörüngeyi Takip Etme Algoritması



Şekil 70. 3 boyutta uzuv açılarının tanımı

Şekil 70'de robot uzuvlarının tanımı görülmektedir. Burada $x_1 y_1$ uzvun başlangıç noktasının koordinatlarını, $x_2 y_2$ uzvun bitiş noktasının koordinatlarını, $\mathbf{L}$ uzvun boy uzunluğunu, α uzvun *z* ekseni etrafındaki dönmesini ve β uzvun *y* ekseni etrafındaki dönmesini gösterir.

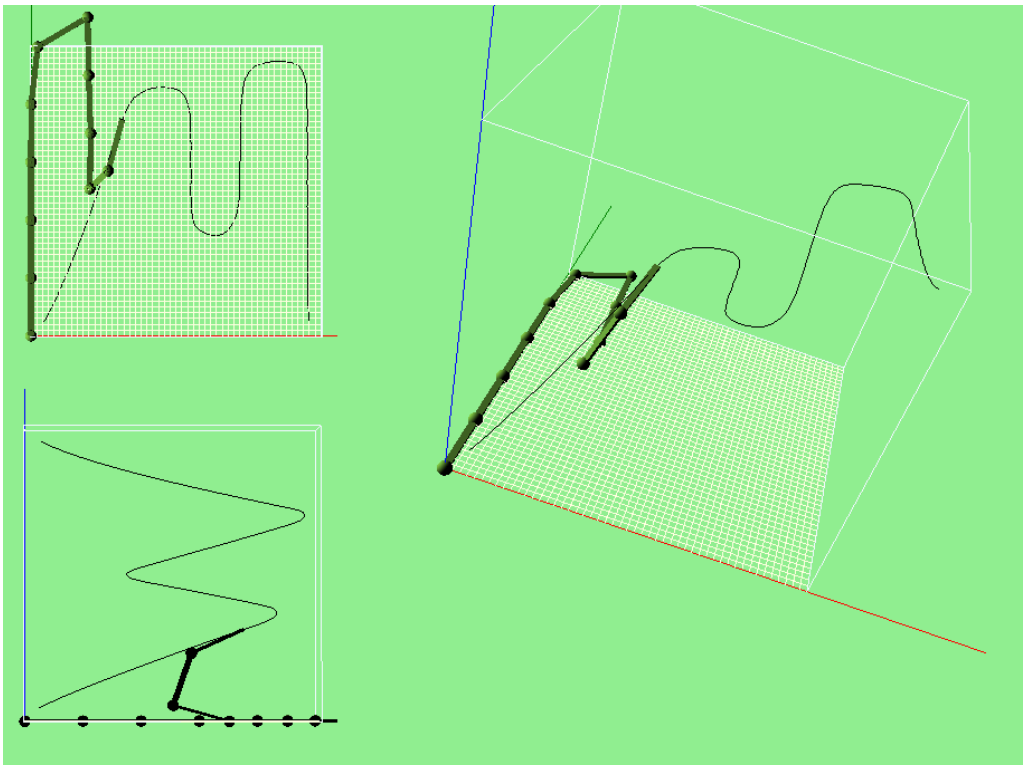
3 boyutta geliştirilen algoritma, yukarıda açıklanan 2 boyutta serbest çalışma bölgesi için geliştirilen algoritmaya 3. boyut eklenerek elde edilmiştir. Bunun için bütün kod gözden geçirilmiş ve çalışmanın hemen hemen hepsi kodda değişiklikler yapmakla ortaya çıkmıştır. Burada ilginç olan sadece bu değişikliklerle algoritmanın 3 boyutta oluşturulması ve hemen hiç problem çıkarmadan çalışmış olmasıdır.

6.3.1 3 Boyutlu Serbest Çalışma Bölgesi İçin Geliştirilen Algoritma İçin Örnek

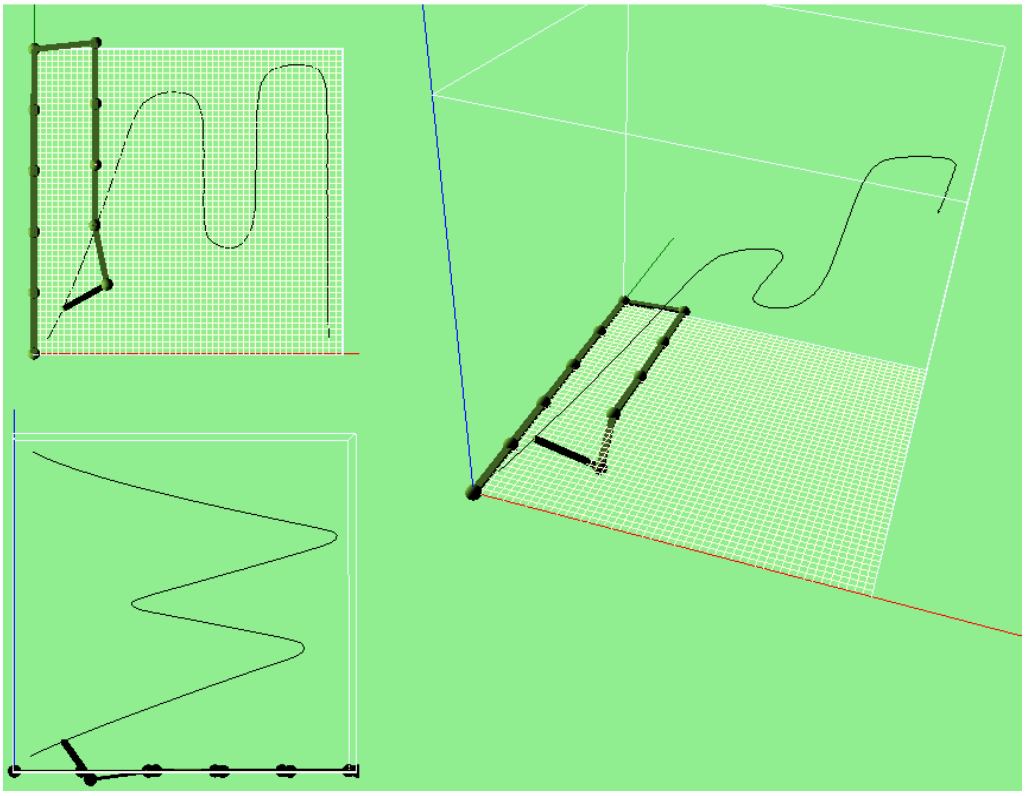
Bölüm 5.3.1'de verilen çalışma alanına benzer bir çalışma alanında 11 uzuvlu, her uzvunda 2 serbestlik derecesi olmak üzere toplan 22 serbestlik derecesine sahip bir seri robotun verilen noktalardan oluşmuş bir yörüngeyi takip etmesi istenmektedir.

Şekil 71 a-g'de robot yörüngeyi takip ederken, robotun 3 tane 3 boyutlu görüntüsü aynı anda robotu değişik açılarda göstermektedir.

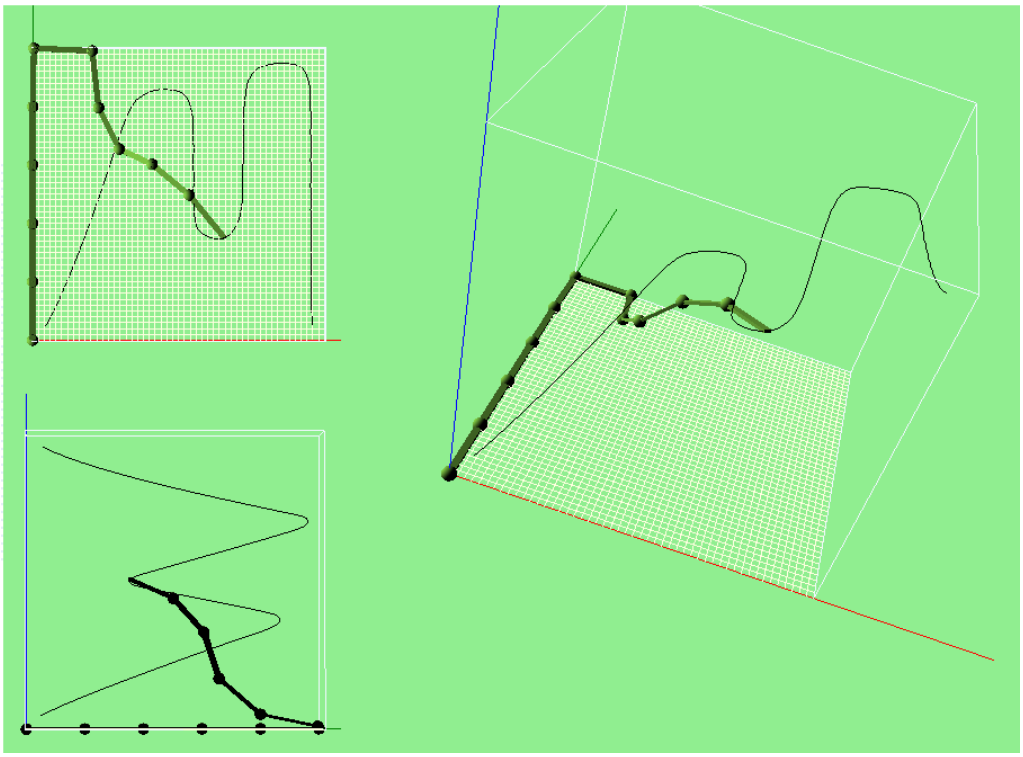
Ekte verilen CD'de **104M260_ESahin_CONKUR_3DSerbestBolge.wmv** isimli videoda algoritmanın nasıl çalıştığı gözlemlenebilir.



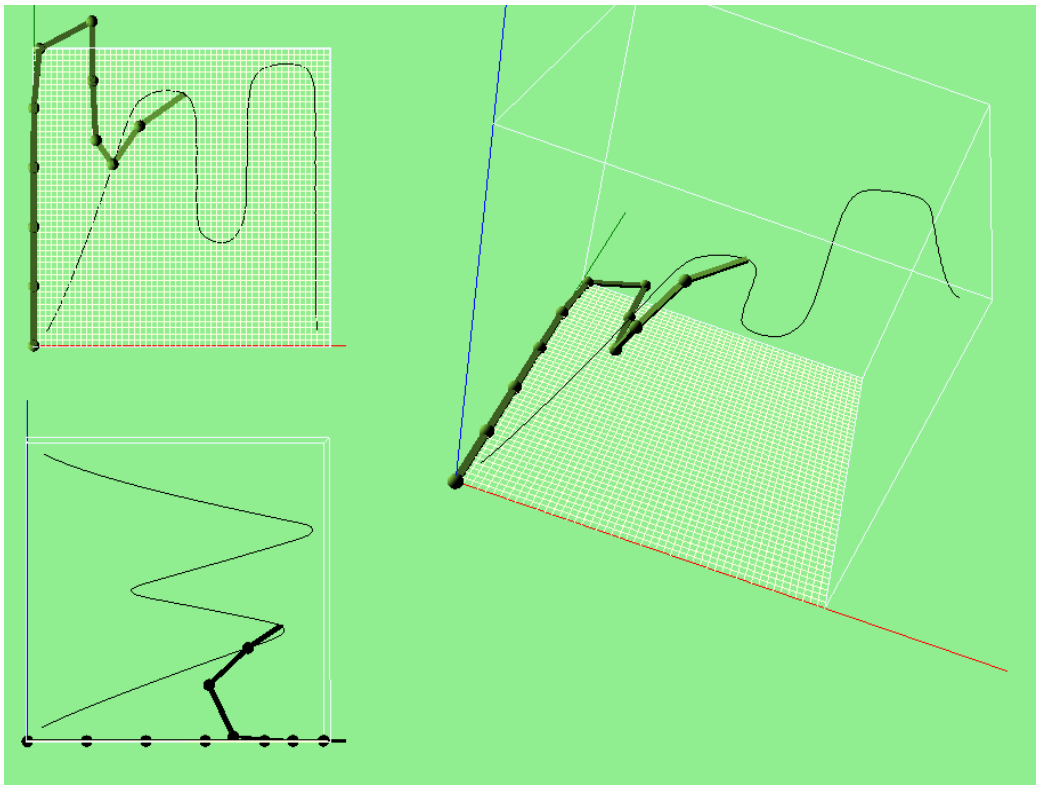
a



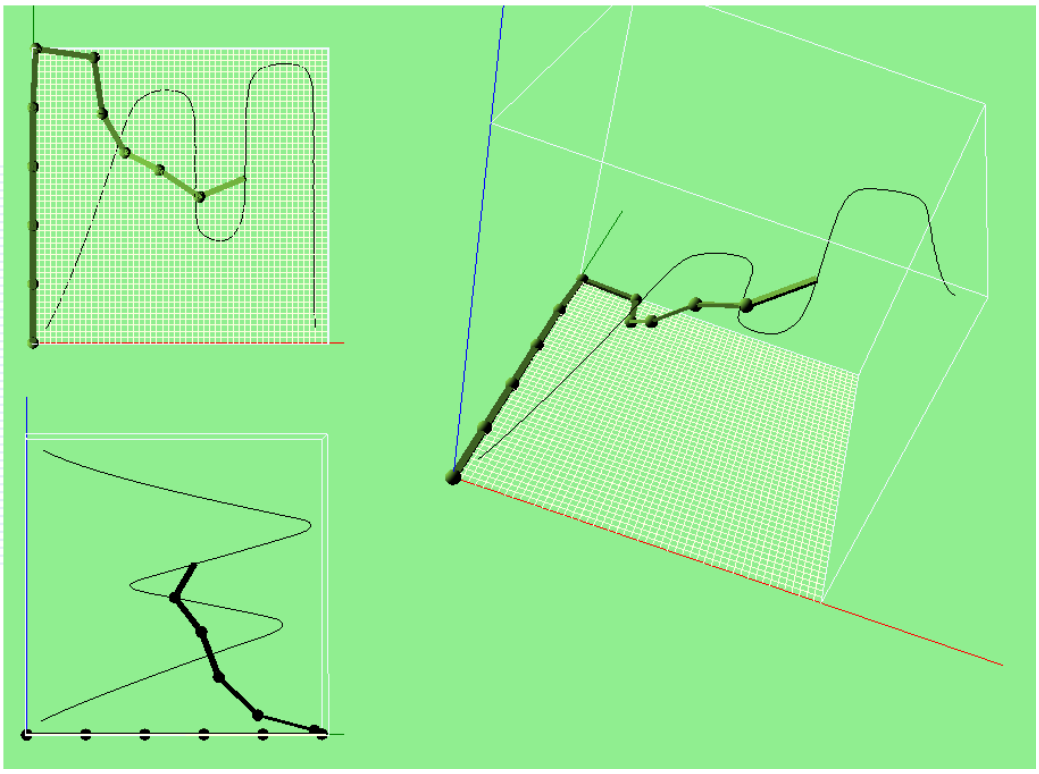
b



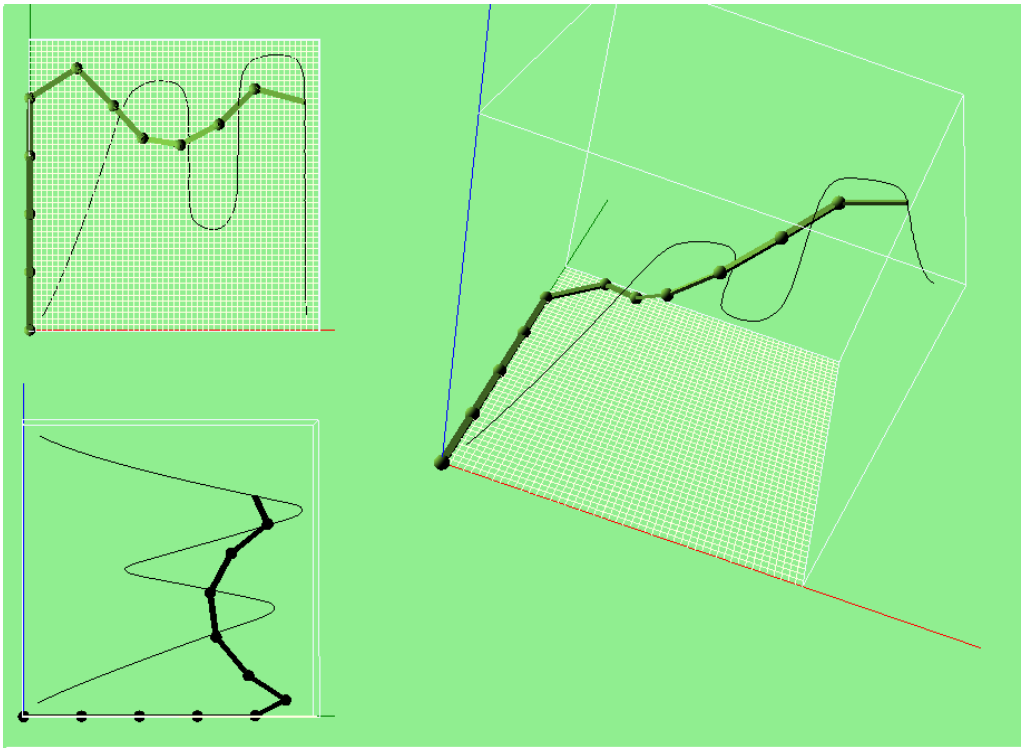
c



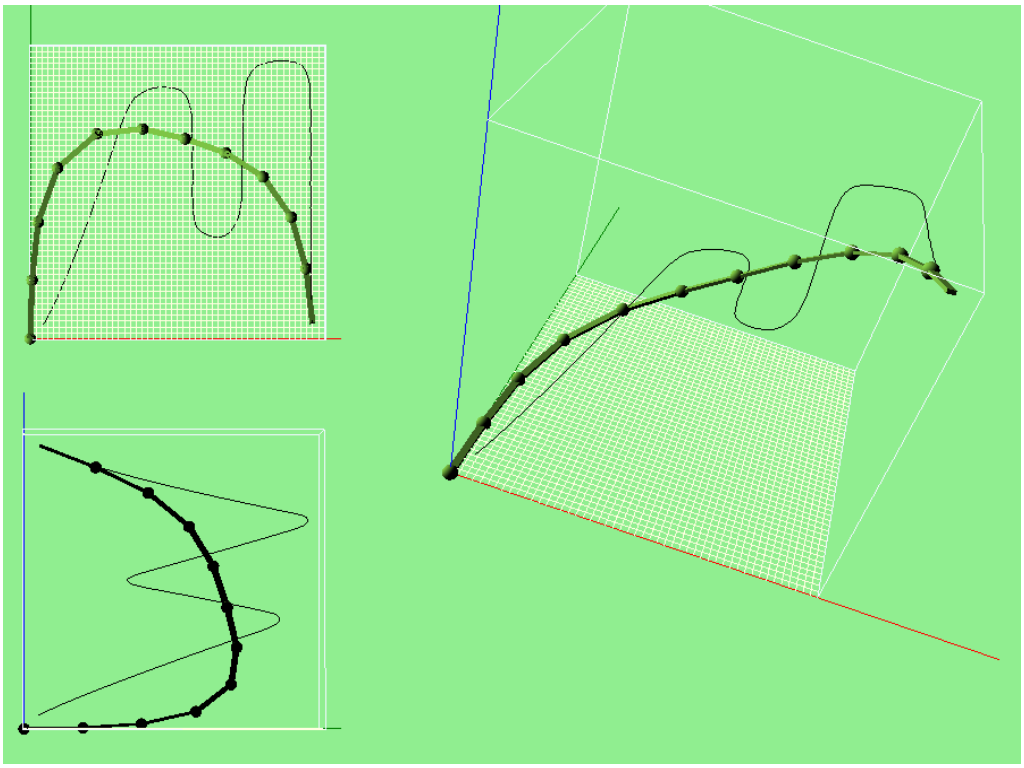
d



e



f



g

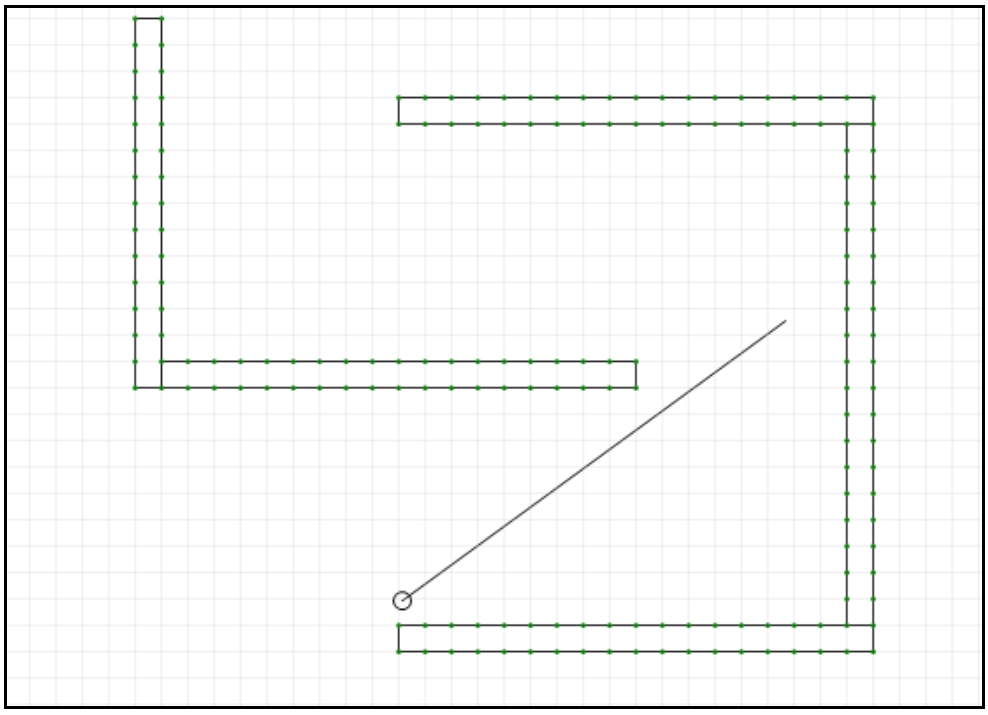
Şekil 71 a-g 3boyutta uzayda 11 uzuvlu robotun hedefini bulması

7 “Engellerden Kaçınma Algoritması” Geliştirilmesi

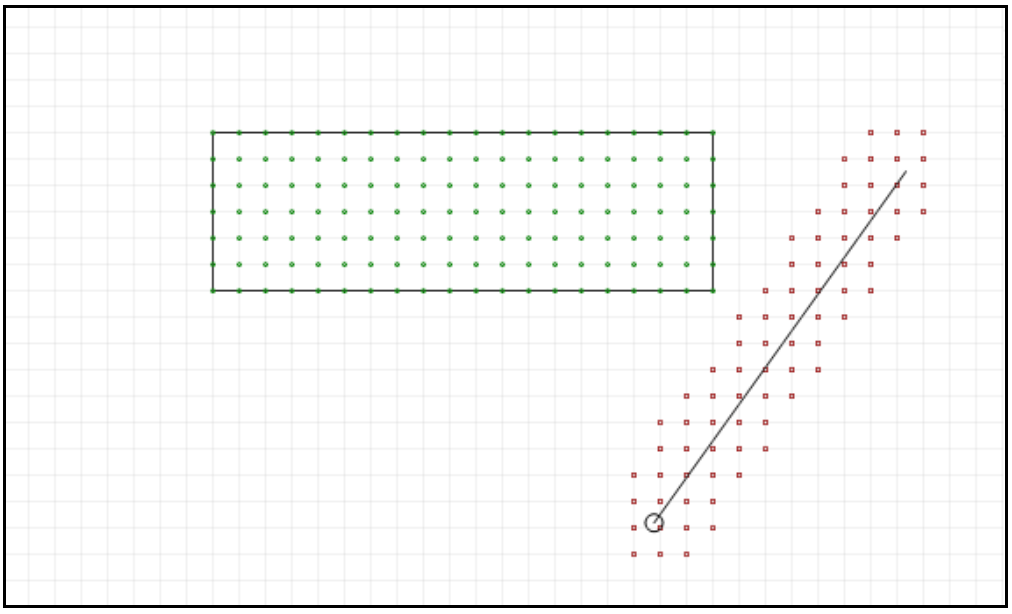
Projemizin yazılım kısmındaki önemli hedeflerinden birisi “gereğinden çok serbestlik dereceli seri robot” için “engellerden kaçınarak hedefine ulaşma algoritması” geliştirmektir.

Fiziksel olarak mümkün olduğu müddetçe robot engeller arasında manevra yaparak ve engellere çarpmadan ilerlemesi gerekmektedir. Bunun oldukça zor bir iş olduğunu fakat burada açıklanan algoritmayla bunu gerçekleştirdiğimizi söyleyebiliriz. Aşağıda detaylar açıklanacaktır.

Şekil 72’de engellerle dolu bir çalışma alanı verilmiştir. Bu alanda bir uzuvlu bir robot görülmektedir. Engeller şekilde görüldüğü gibi grid noktalarıyla temsil edilmektedir. Buradaki amaç, dediğimiz gibi, bu tek uzuvlu robotun engellere çarpmadan bu engeller arasından geçmesidir. Daha sonra bu uzva diğer uzuvlar da eklenecek ve manipülatörün hiçbir uzvunun engellere çarpmadan hareket edebilmesine çalışılacaktır.



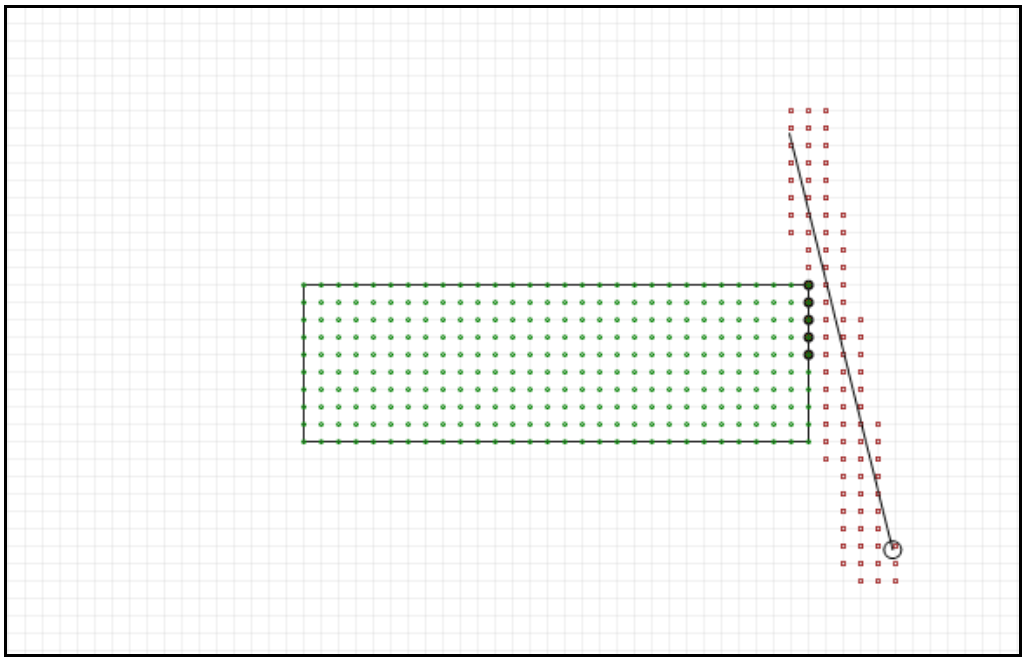
Şekil 72. Tek uzuvlu robotu ve engelleri içeren çalışma alanı



Şekil 73. Uzvu çevreleyen güvenli alan

Burada bizim için önemli bir nokta da robotun hareketinin yumuşak geçişlere sahip olmasıdır. Yoksa robot hareketi istenmeyen şekilde çok titrek olabilmektedir. Bu titreklikleri gidermek için algoritmanın üzerinde detaylı çalışma yapılmıştır.

Şekil 73'de görülen çalışma alanında bir engelle yine bir uzuvlu bir robot görülmektedir. Robot engellerden yeteri kadar uzak olduğunda, bu bölgede robot bundan önceki raporda detaylarını verdiğimiz algoritmayı kullanarak hareket etmektedir. Uzvun şekilde görüldüğü gibi çevresini kaplayan bir alan vardır. Bu alan uzuv hareket ettiğinde onunla birlikte hareket etmekte ve uzuv engellere yaklaştığında engellerin grid noktaları bu alan içine girmesi durumunda uzuv bundan haberdar olmaktadır. Şekil 73'de görülen pozisyonda hiçbir engel grid noktası bu alan içinde değildir. Uzuv serbest bölgede hareket etmektedir.



Şekil 74. Uzvu çevreleyen güvenli alana giren engel noktaları (koyu renkli)

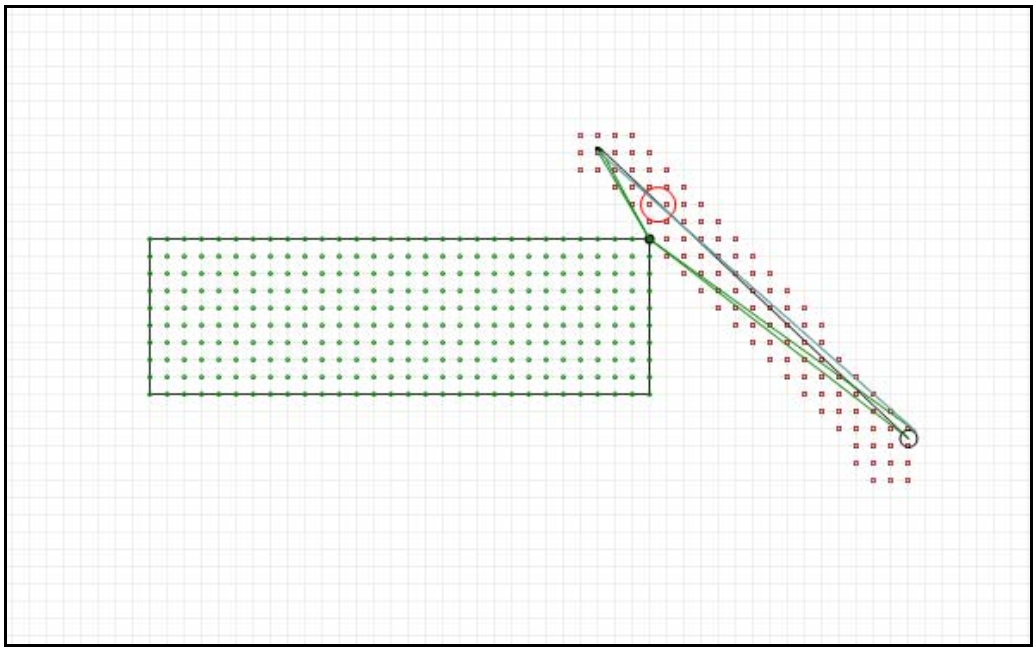
Şekil 74'de ise uzuv engele yeterince yaklaşmıştır ve 5 tane engel noktası bu alan içine girmiştir. Şekilde görüldüğü gibi bu kritik engel noktaları daha koyu olarak çizilmiştir.

Buradaki en kolay engelden kaçınma yöntemi "uzvu engelden uzaklaştıracak şekilde geriye doğru itme" hareketi yapmaktır. Fakat bu yöntemin yukarıda bahsedilen titreme de dahil birçok dezavantajı vardır.

Bunun yerine, geliştirdiğimiz bir algoritma ile, en yakın engel noktası etrafında önce dönme sonra öteleme ile uzvu hareket ettirdik. Şekil 75'de uzvun dönme merkezi olarak seçilmiş olan köşedeki koyu renkli engel noktası görülmektedir. Uzuv döndükten sonra öteleme yapmaktadır.

Fakat burada bir soru "bir serbestlik dereceli birden fazla uzuv için bu hareketi gerçekleştirmenin ne derece mümkün olduğu" sorusu olabilir. Uzuvların sadece dönme hareketi olmasına rağmen bir uzuvdan önceki uzuvların dönme hareketlerinden o uzvun hem dönme hem de öteleme hareketi çıkarılmıştır.

Ekte verilen CD'de **104M260_ESahin_CONKUR_roboKolObstacle.wmv** isimli videoda algoritmanın nasıl çalıştığı gözlemlenebilir.



Şekil 75. En yakın engel noktası etrafında önce dönme

7.1 Sabit Bir Nokta Etrafında Dönme Olayının Geliştirilmesi

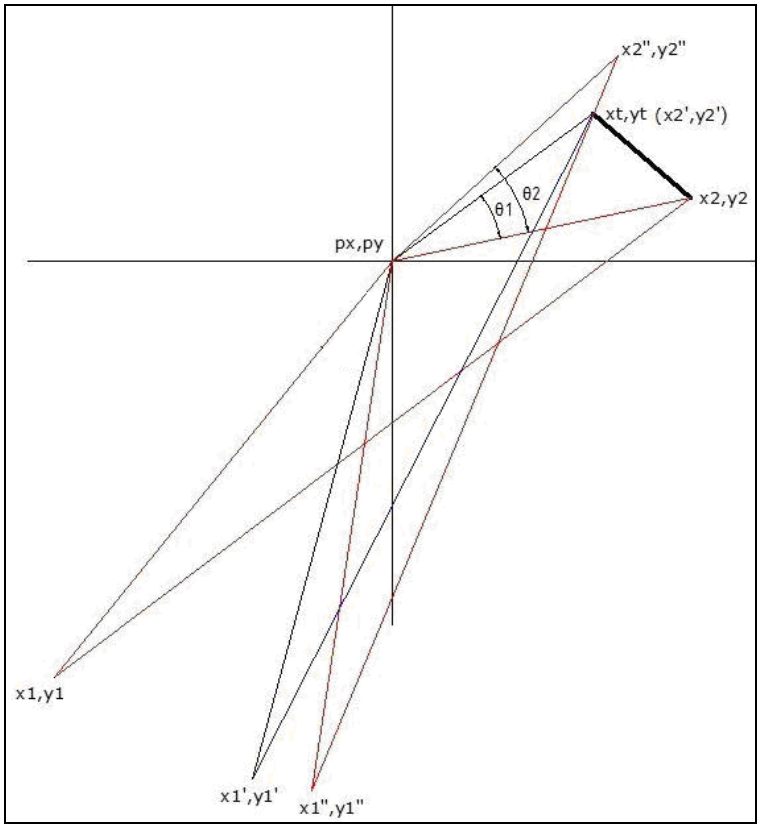
Uzuv engellerle karşılaştığında kendisine en uygun yakın engel noktası etrafında önce **dönme** sonra **öteleme** hareketi yapmaktadır. Şekil 76 dikkate alınarak **dönme**+**öteleme** hareketi şu şekilde açıklanabilir:

(x_1, y_1) noktasından başlayıp (x_2, y_2) noktasında biten uzvun (p_x, p_y) noktası etrafında, uzvun uç noktası (x_2, y_2)'nin, gitmesi gereken yeni nokta (x_t, y_t)'yi dikkate alarak dönmesi istenmektedir. (p_x, p_y), (x_2, y_2) ve (x_t, y_t) noktalarının değerleri bilindiğinden θ_1 açısı kolaylıkla bulunur ve (x_1, y_1) noktası bu açı miktarı kadar döndürülür. Sonuçta uzvun ucu (x_t, y_t)'de veya (x_2', y_2')'de, başlangıç noktası da (x_1', y_1') olacak şekilde yeni doğrultuya yerleştirilir. Burada uzvun başlangıç noktası genellikle tam olarak (x_1', y_1') noktasında olmaz. Uzuv, uzvun boyu sabit kalacak ve uzvun ucu (x_t, y_t)'de olacak şekilde (x_t, y_t) noktası ile (x_1', y_1') noktasının oluşturduğu doğrultu üzerine yerleştirilir.

Bu şekilde yapılan **dönme**+**öteleme** hareketi normalde çalışmaktadır. Fakat bazı durumlarda uzuv, etrafında döndüğü engel noktasına gerektiğinden fazla yaklaşmaya başlamakta ve o

nokta ile çarpışmaktadır. Uzuv engel noktası etrafında döndüğünden bunun olmaması gerekir. Daha sonraki çalışmalarımızda tam olarak dönmenin niçin gerçekleşmediğini bulduk. Şekil 76'ya tekrar dönersek, esasında uzuv ancak θ_2 kadar döndüğünde olması gereken doğrultuda olacaktır. Bu doğrultu aynı zamanda (x_t, y_t) noktasından geçmektedir. Şekilde, uzuv θ_2'ye göre döndürülmüş ve uzvun ucu (x_2'', y_2'')'de, başlangıç noktası da (x_1'', y_1'') olacak şekilde yeni doğrultuya yerleştirilmiştir. Uzuv bu doğrultu üzerinde ileri veya geriye istenildiği gibi hareket edebilir.

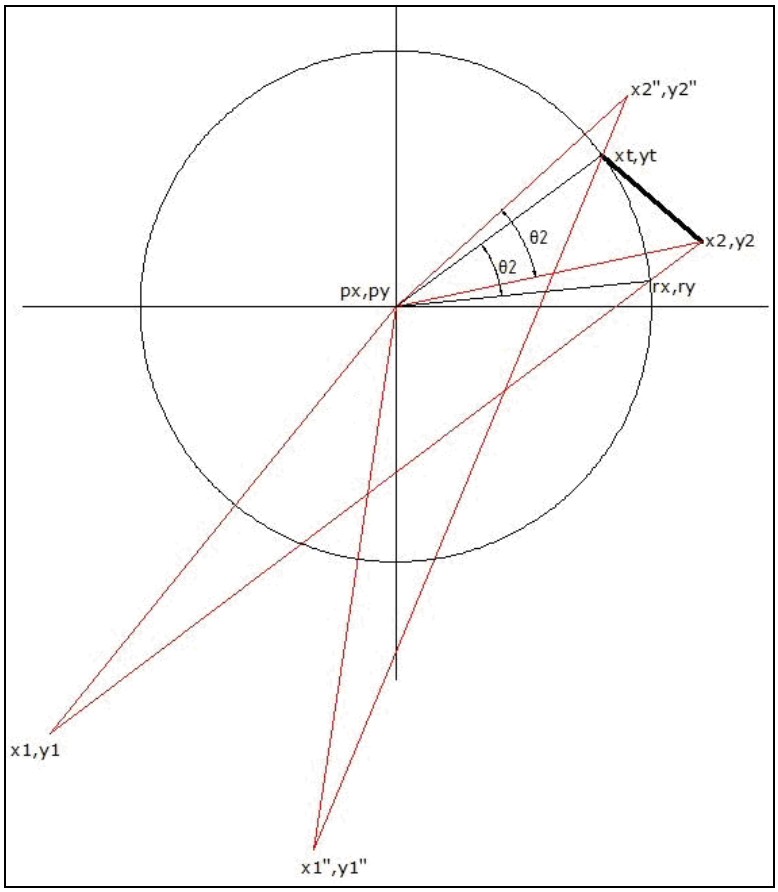
Şekil 76'da her iki metotla döndürülmüş uzuv şekilleri görülmektedir. Görüldüğü gibi uzvun son durumundaki doğrultuları arasında oluşan fark uzvu dönme noktasına yaklaştırmasını açıklamaktadır.



Şekil 76. Uzvun sabit bir nokta etrafında dönmesi

Burada θ_2 açısını bulmak, θ_1 açısını bulmak kadar kolay olmamıştır. Şekil 77'ye bakıldığında, (p_x, p_y) noktası ile (x_t, y_t) noktası arasındaki uzaklığa r diyelim. (r_x, r_y) noktası ile (p_x, p_y) noktası arasındaki uzaklık da r olacak şekilde uzuv üzerinde bir (r_x, r_y) noktası

seçelim. Uzuv döndüğünde, (r_x, r_y) noktası (x_t, y_t) üzerine gelecektir. Aradaki açı θ_2'dir. Burada (p_x, p_y), (x_t, y_t) ve r bilinmektedir. Fakat (r_x, r_y) değerleri bilinmemektedir.



Şekil 77. θ_2 açısını bulmak

Şekil 78' e bakarsak $\mathbf{a}, \mathbf{n}$ ve $\mathbf{r}$ vektörleri arasındaki ilişki

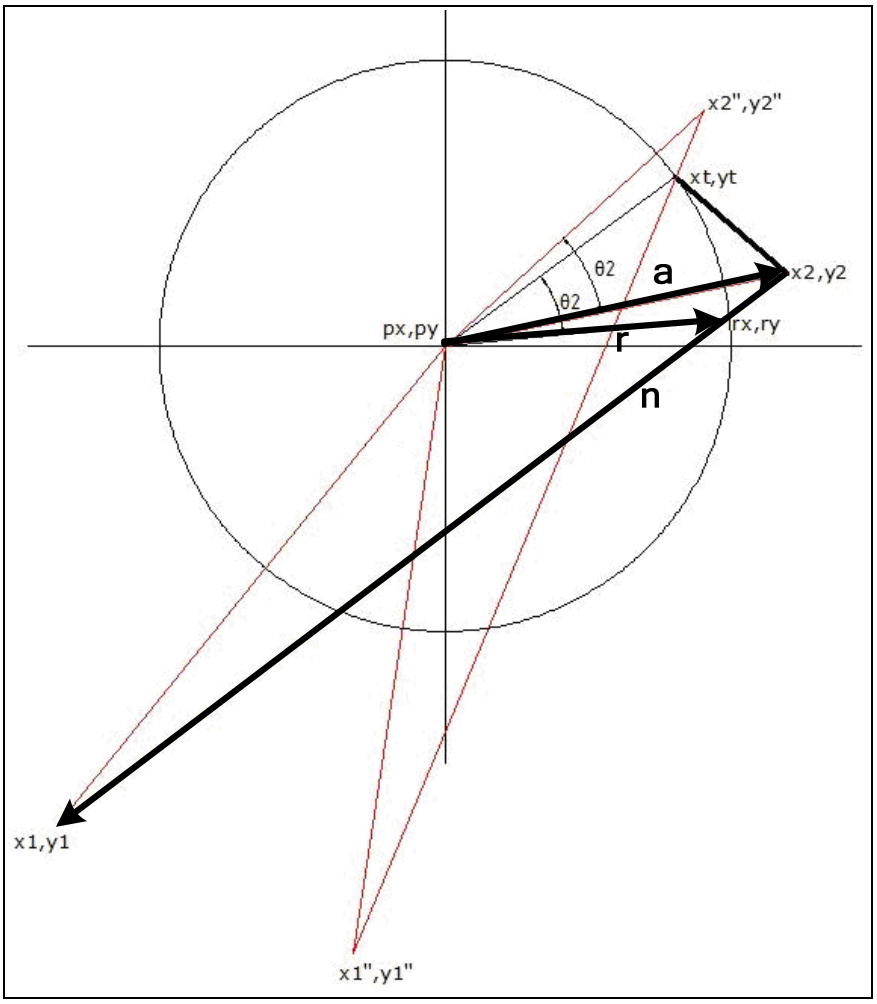
$$\mathbf{r} = \mathbf{a} + t.\mathbf{n}$$

dir. Burada $t.\mathbf{n}$ vektör parçacığının büyüklüğü, (x_2, y_2) ile (r_x, r_y) arasındaki mesafeye karşılık gelmektedir. Ayrıca,

$$|\mathbf{r}|^2 = |\mathbf{r}_x|^2 + |\mathbf{r}_y|^2$$

olduğundan bu iki denklem ortak olarak çözülürse önce t parametresinin değeri, sonra da buna bağlı (r_x, r_y) değerleri bulunur. Çözüm Şekil 79'da verildiği gibi MathCad'de yapılmıştır.

Yalnız, t parametresinin iki değeri bulunmaktadır. Çözümün çalışıp çalışmadığını ve hangi değerin, nasıl kullanılacağını görmek için Şekil 80'de arayüzü görülen küçük bir program parçacığı geliştirilmiştir. Bu program çalıştırıldığında Şekil 79'da verilen kökün çözümü için her durumda gerekli ve yeterli olduğu gözlemlenmiştir.



Şekil 78. θ_2 açısını bulmak

Given

$r_x - a_x = t \cdot n_x$

$r_y - a_y = t \cdot n_y$

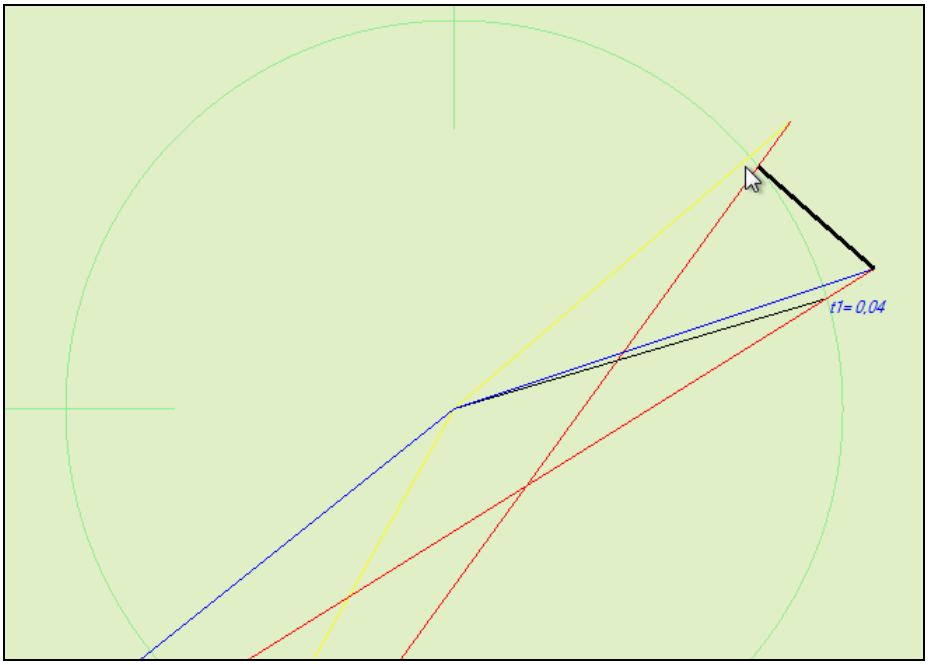
$r_x^2 + r_y^2 = r^2$

$$\text{Find}(r_x, r_y, t) \rightarrow \begin{bmatrix} a_x + \frac{1}{\left[2 \cdot \left(n_y^2 + n_x^2\right)\right]} \cdot \left[-2 \cdot a_x \cdot n_x - 2 \cdot a_y \cdot n_y + 2 \cdot \left(2 \cdot a_x \cdot n_x \cdot a_y \cdot n_y - n_y^2 \cdot a_x^2 + n_y^2 \cdot r^2 + n_x^2 \cdot \right.\right. \\ a_y + \frac{1}{\left[2 \cdot \left(n_y^2 + n_x^2\right)\right]} \cdot \left[-2 \cdot a_x \cdot n_x - 2 \cdot a_y \cdot n_y + 2 \cdot \left(2 \cdot a_x \cdot n_x \cdot a_y \cdot n_y - n_y^2 \cdot a_x^2 + n_y^2 \cdot r^2 + n_x^2 \cdot \right.\right. \\ \frac{1}{\left[2 \cdot \left(n_y^2 + n_x^2\right)\right]} \cdot \left[-2 \cdot a_x \cdot n_x - 2 \cdot a_y \cdot n_y + 2 \cdot \left(2 \cdot a_x \cdot n_x \cdot a_y \cdot n_y - n_y^2 \cdot a_x^2 + n_y^2 \cdot r^2 + n_x^2 \cdot \right.\right. \end{bmatrix}$$

$r_x := 431 \quad r_y := 45 \quad a_x := 516 \quad a_y := 109 \quad n_x := -1097 \quad n_y := -827 \quad r := 423$

$$t := \frac{1}{\left(n_x^2 + n_y^2\right)} \cdot \left[-a_x \cdot n_x - a_y \cdot n_y + \left(2 \cdot a_x \cdot n_x \cdot a_y \cdot n_y + n_x^2 \cdot r^2 - n_x^2 \cdot a_y^2 - n_y^2 \cdot a_x^2 + n_y^2 \cdot r^2\right)^{\left(\frac{1}{2}\right)}\right]$$

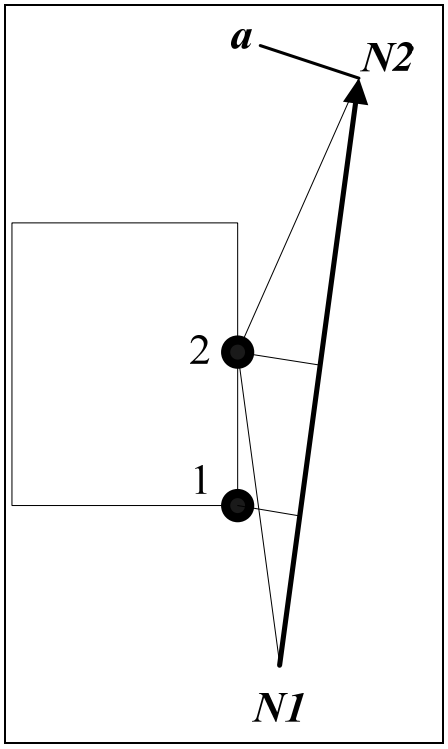
Şekil 79. MathCad'de yapılan çözüm



Şekil 80. θ_2 açısını bulmak için geliştirilen çözümün doğruluğunu denemek için yazılmış programın arayüzü

7.2 Dönme Olayının Dışındaki Diğer Noktalar

Şekil 81'de örnek olarak verilen durumda, uzvun **N2**'den **a** noktasına gitmesi için gerekli dönme noktasının seçiminde, iki tane muhtemel engel noktası görülmektedir. Bundan önce, **uzva dik uzaklığı en kısa olan noktayı** seçmiştik. Fakat şekle dikkatle bakıldığında uzvun 1 numaralı nokta etrafında dönmesi, uzvun engele daha da yakınlaşmasına sebep olmaktadır. En iyi tercih, uzva dik uzaklığı en kısa olan nokta değil, **uzvun uç noktasına en yakın olan noktadır**. Şekil 21'de verilen örnekte bu nokta 2 numaralı nokta olduğundan, dönme noktası **2 numaralı nokta** olarak seçilmesi uygundur.



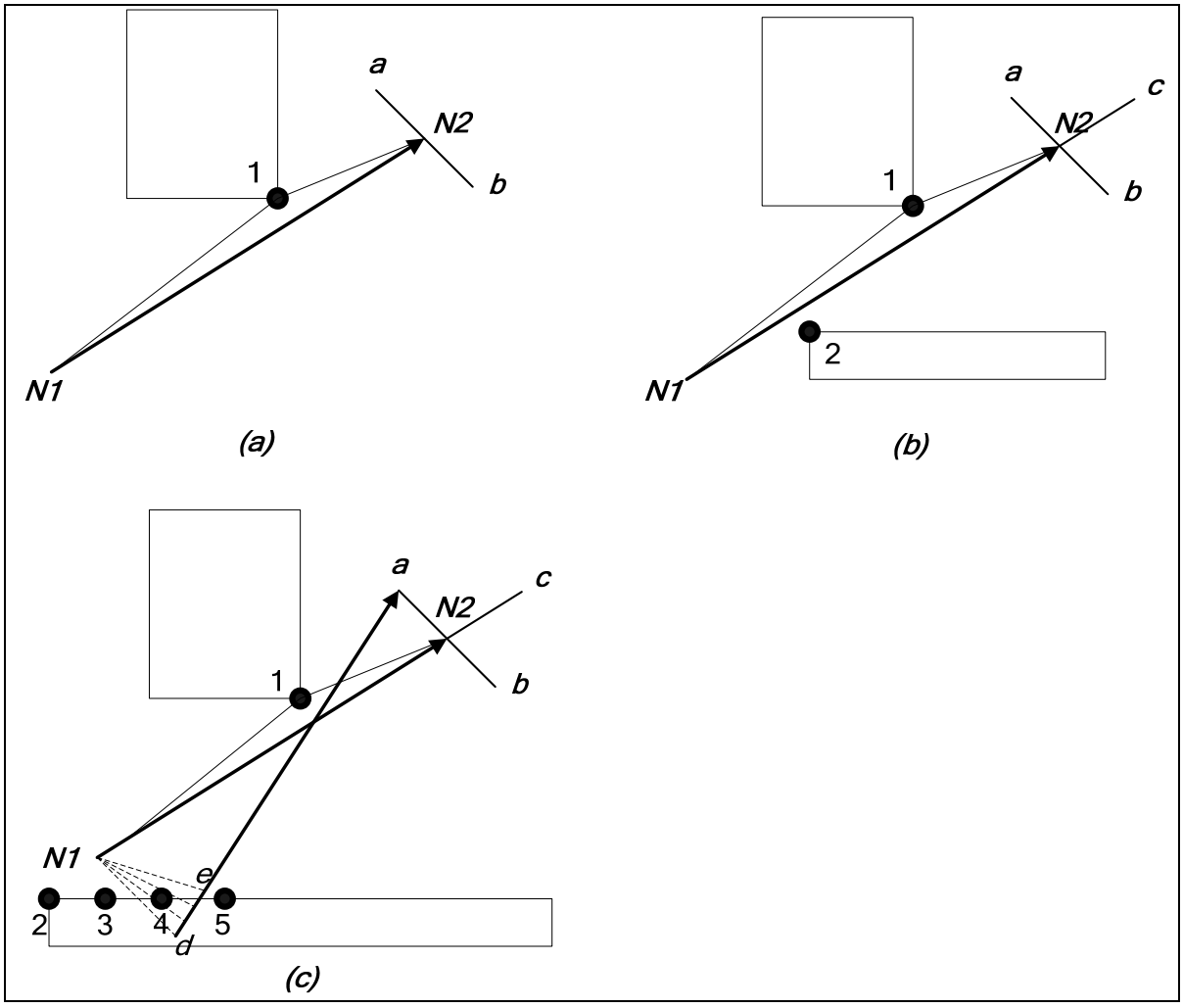
Şekil 81. Dönme noktasının belirlenmesi

Şekil 82a'da 1 numaralı engel noktası etrafında dönme olayı görülmektedir. Eğer uzuv **a** noktası tarafına yönlenmişse, dönme yukarıda açıklandığı gibi gerçekleşecektir. Eğer uzuv **b** noktası tarafına yönlenmişse, engele çarpma durumu olmadığından algoritma 1 numaralı engel noktasını dikkate almadan uzvu **serbest bölge modunda** çalıştıracaktır.

Şekil 82b'de 1 ve 2 numaralı engel noktaları arasında kalma olayı görülmektedir. Eğer uzuv **a** noktası tarafına yönlenmişse, herhangi bir dönme olmayacak uzuv sahip olduğu doğrultu boyunca **ileriye** veya **geriye doğru** hareket edecektir. Çünkü başka türlü bir hareket

çarpışmaya sebep olur. Eğer uzuv **b** noktası tarafına yönlenmişse, algoritma uzvu **2 numaralı engel noktası** etrafında döndürecektir.

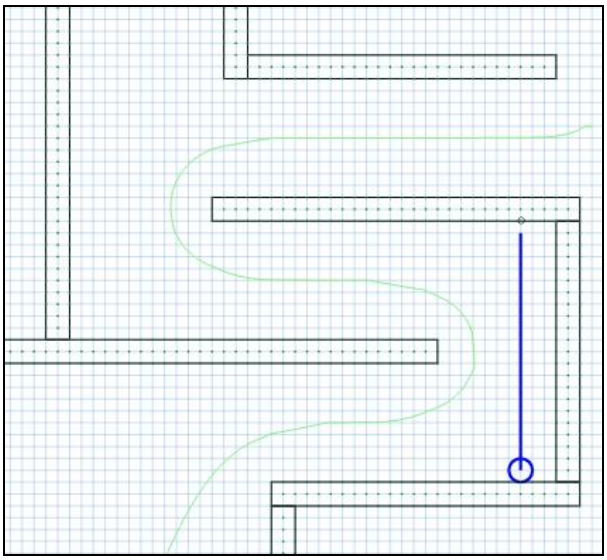
Şekil 82c'de ise en zor durum görülmektedir. Eğer uzuv **a** noktası tarafına yönlenmişse, algoritma uzvu 1 numaralı engel noktası etrafında döndürmek isteyecektir. Fakat uzvun uç noktası **N1**, **d** noktasına gelmek istediğinde 3 ve 4 numaralı engeller arasından geçmesi gerektiğinden ve bu da çarpışmaya sebep olduğundan, uzvun bu şekilde dönmesi mümkün değildir. Bu durumda **d** noktasından **a** noktasına yönlenen şekilde görüldüğü gibi doğrular alınır ve engellerin arasından geçip geçmediği kontrol edilir. Şekilde **N1** ile **e** arasındaki doğru engeller arasından geçmemektedir, serbest bölgededir. Böylece uzvun yeni uç noktası, **d** noktası yerine **e** noktası alınır. Buna göre uzuv döndürülürken ötelenir. Eğer uzuv **b** noktası tarafına yönlenmişse, algoritma uzvu **5 numaralı engel noktası** etrafında döndürecektir.



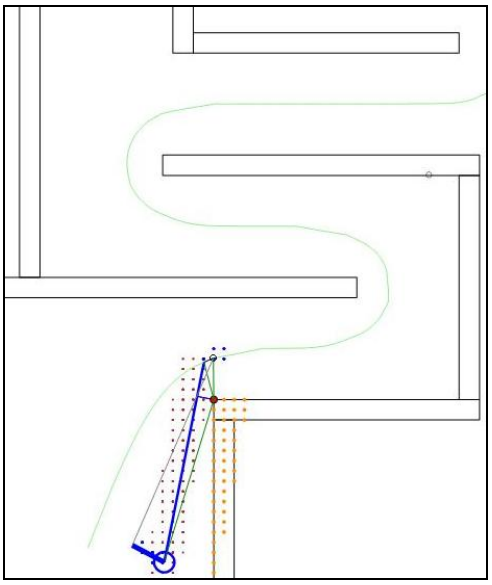
Şekil 82. a-c. Uzvun karşılaştığı diğer durumlardan örnekler

7.3 Uzvun Engellerden Kaçınma Algoritması İçin Örnek

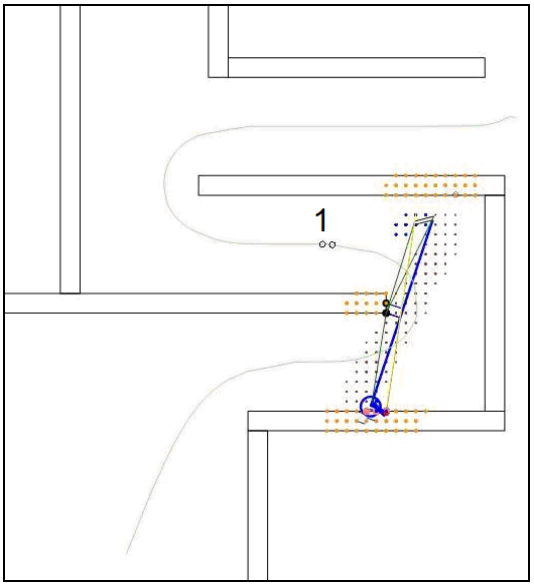
Engellerden kaçınma algoritmasının **bütün altyapısı** hazırlanmış ve bu bölümde çalışır bir örneği gösterilmiştir. Şekil 83'de engellerle dolu bir çalışma alanı ve bir uzuv görülmektedir. Uzvun manevra yapacağı bölgede, **uzvun boyunun yaklaşık olarak manevra alanının genişliğiyle aynı olduğu** görülmektedir. Uzvun genişliği manevra alanının genişliğinden fazla olduğunda uzuv fiziksel olarak manevra yapamaz. Yani bu senaryo, manevra için en zor durumlardan birisini temsil etmektedir.



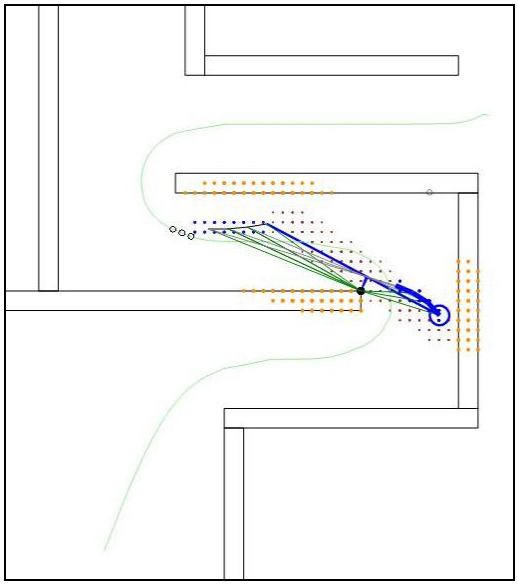
Şekil 83. Engellerle dolu çalışma alanı ve uzuv



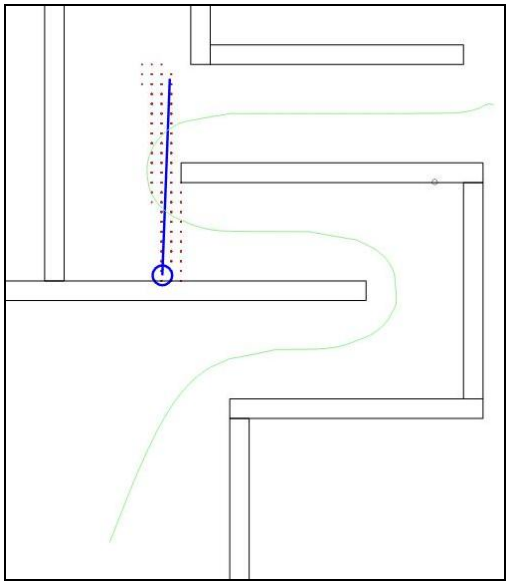
(a)



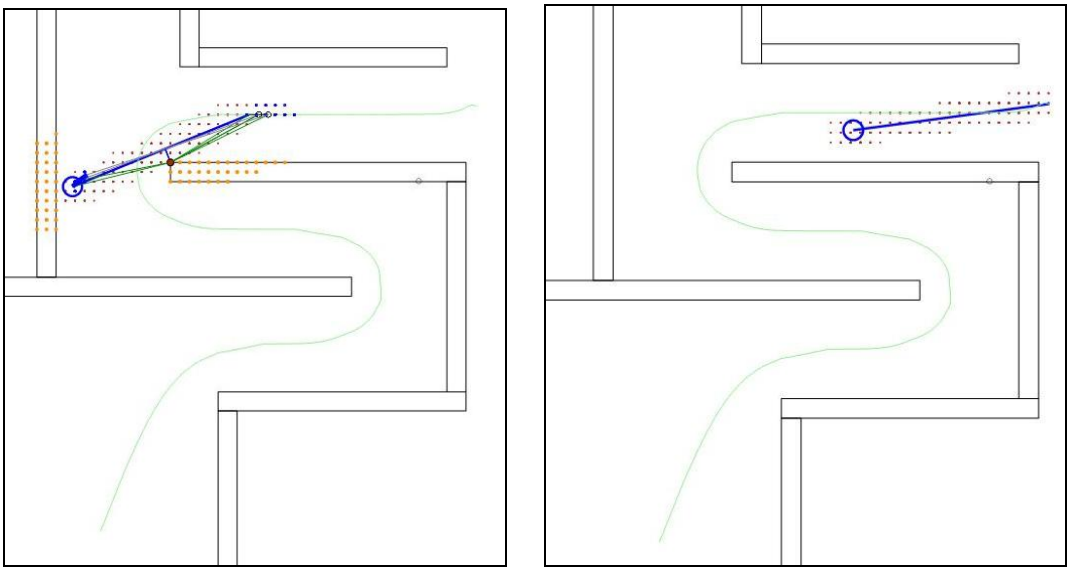
(b)



(c)



(d)



(e) (f)

Şekil 84 a-f. Uzvun engellerden kaçınarak hedefine varması

Şekil 84 a-f'de uzvun engellerden kaçınarak hedefine varması sırasında çekilmiş resimler görülmektedir. Uzuv, bundan önceki bölümlerde anlatılan **2 boyutlu potansiyel alan tarafından üretilen yolu** takip etmektedir.

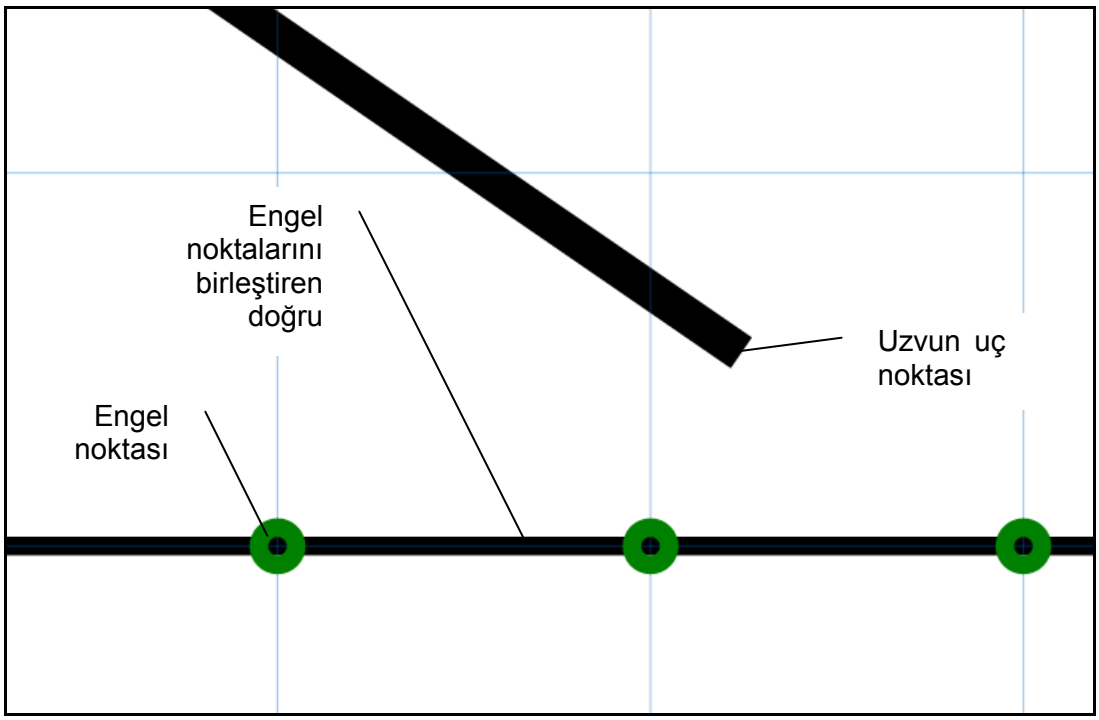
Uzvun uç noktası fiziksel olarak mümkün olduğu durumlarda yolu birebir takip etmektedir. Fakat keskin manevra gereken durumlarda fiziksel olarak yolu takip etmek imkansız olduğundan yoldan geçici olarak ayrılmakta daha sonra tekrar yolu birebir takibe başlamaktadır. Örneğin Şekil 84b'de **1** ile gösterilen yola ait olan iki küçük daire o anki ulaşılması gereken hedef noktalarıdır. Uzvun engellerden kaçınabilmesi için uç noktasının şekilde gösterilen konumda kalması gerekir. Uç noktası yoldan ayrılsa da hareketinin doğrultusu bu küçük dairelere doğru yönlenmiştir.

Ekte verilen CD'de bulunan **104M260_ESahin_CONKUR_engellerdenKacinma.wmv** videosu bu örneğin daha net olarak görünmesini sağlamaktadır.

7.4 Uzuvların Uç Noktalarının Engellerden Kaçınması

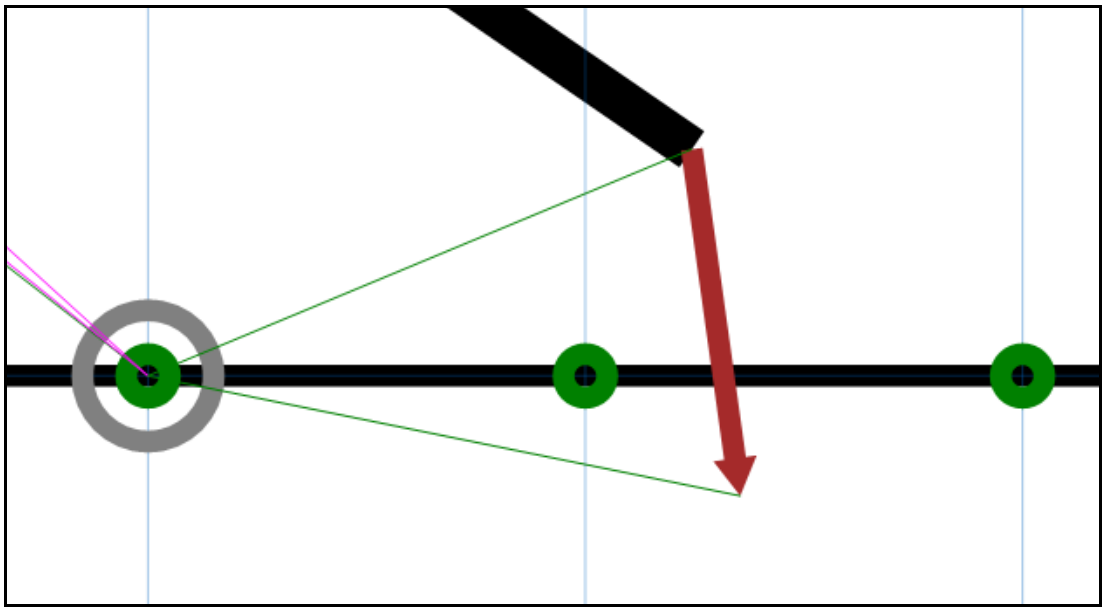
Şekil 82c'de gösterilen uzvun **uç** noktasının engellerden kaçınması için geliştirilmiş teknik, geliştirildiği konum için çalışmakla beraber birçok yan etkiyi de beraberinde getirmiştir. Bu yan etkileri gidermek yerine köşe uç noktalarının engellerden kaçınması yeniden düzenlenmiştir. Şekil 82c'de gösterilen teknik tamamen iptal edilmiştir.

Uç noktasının engellerden kaçınması için geliştirilen teknik bir grid karesinin köşe noktalarının engel noktaları içerip içermemesi üzerine geliştirilmiştir.



Şekil 85. Uzvun uç noktası ve grid üzerinde engel noktaları

Şekil 85'de grid'e bölünmüş çalışma alanı içinde grid kareleri üzerinde engel noktaları, engel noktalarını birleştiren doğru parçaları ve engel noktalarına yakın olan bir uzvun uç noktası görülmektedir.

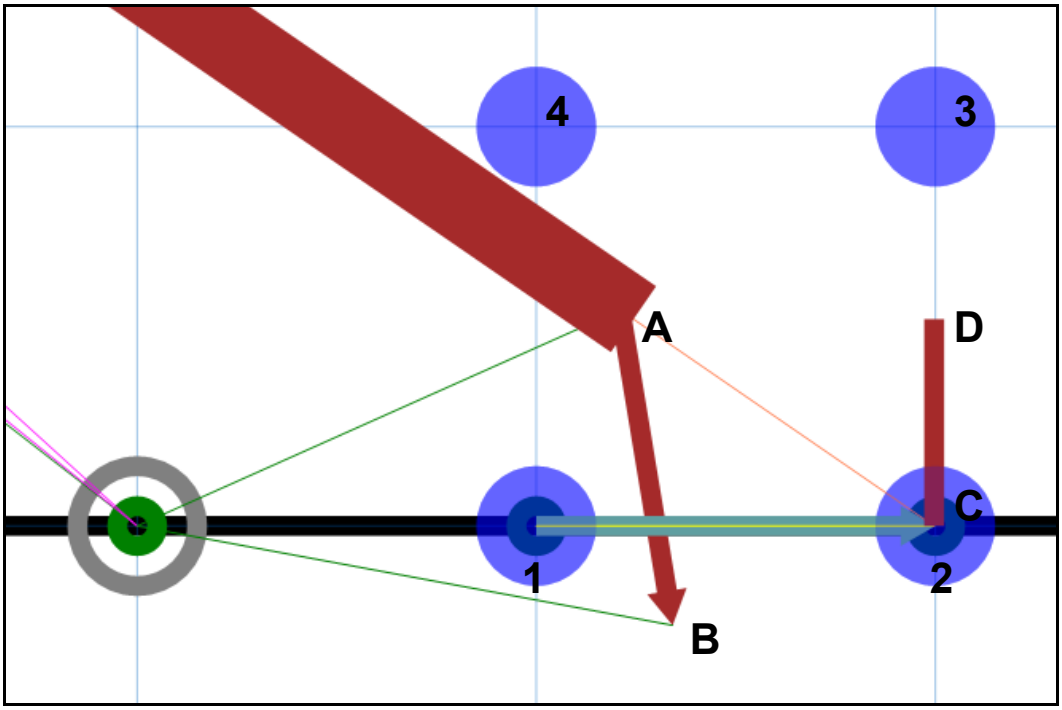


Şekil 86. Robotun uç noktasının hareketi

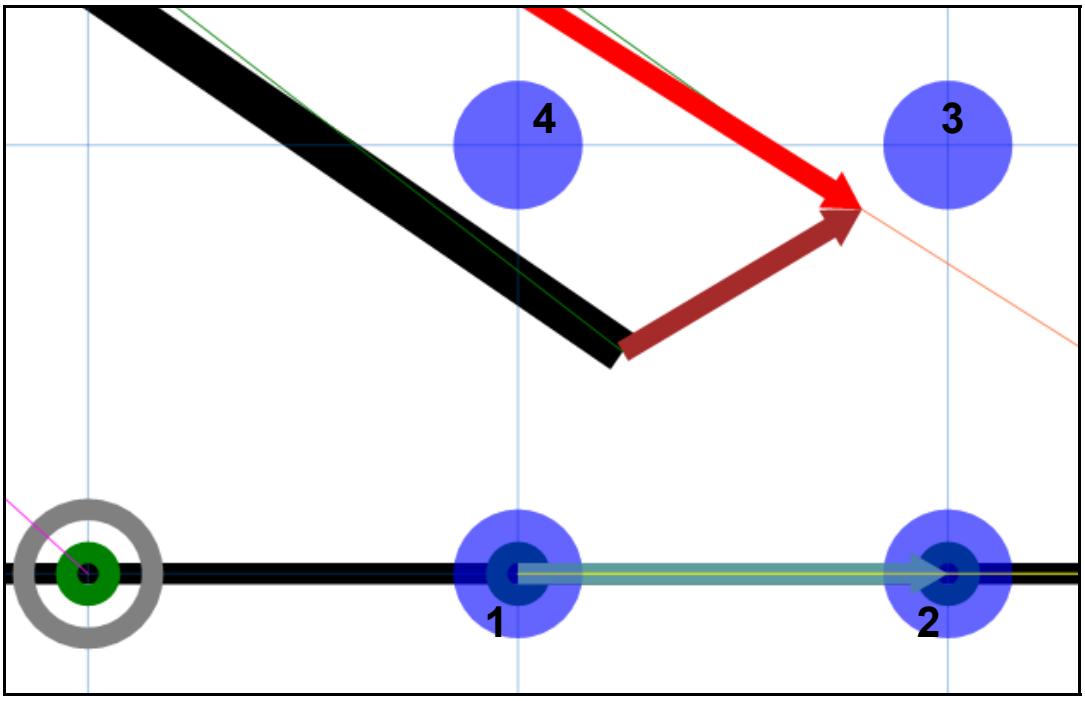
Şekil 86'da uzvun uç noktasının A pozisyonundan B posizyonuna hareket etmesi istenmektedir. Şekilden de görüldüğü gibi bu hareket uzvun engelle çarpışması anlamına gelmektedir. Bu sebepten her hareket öncesinde uzvun uç noktalarının engel içine girip girmediği kontrol edilir.

Şekil 87'de uzvun uç noktasının içinde kaldığı grid karesinin köşe noktaları gösterilmiştir. Algoritma ikişer ikişer bu grid noktalarını ayırır ve her iki grid noktalarında da engel noktası olup olmadığını inceler. Eğer her iki nokta da engel noktası ise uç noktasının hareket vektörü bu iki nokta arasından geçmemeli hatta uç noktasının bu iki nokta arasındaki doğruya dik olan mesafesi belli bir değerden daha kısa olmamalıdır. Yoksa uzuv ya linke çarpar ya da aşırı yaklaşmış olur.

Şekil 87'ye tekrar bakarsak 1 ve 2 numaralı grid noktalarının her ikisinde de engel noktası vardır ve uç noktasının hareket vektörü bu iki nokta arasından geçmektedir. Diğer noktalar 2-3, 3-4, 4-1 ikişer engel noktası içermez, bu sebepten onların aralarından geçmek serbesttir. CD doğru parçasıyla gösterilen mesafe uç noktasının bu iki nokta arasındaki doğruya dik olan mesafedir. Bu mesafe önceden belirlenen bir mesafe örneğin ½ grid genişliği kadar alınır ve CD mesafesi bu kriterden daha kısaysa uzuv ters yönde hareket ettirilerek engel noktalarından uzaklaştırılır.



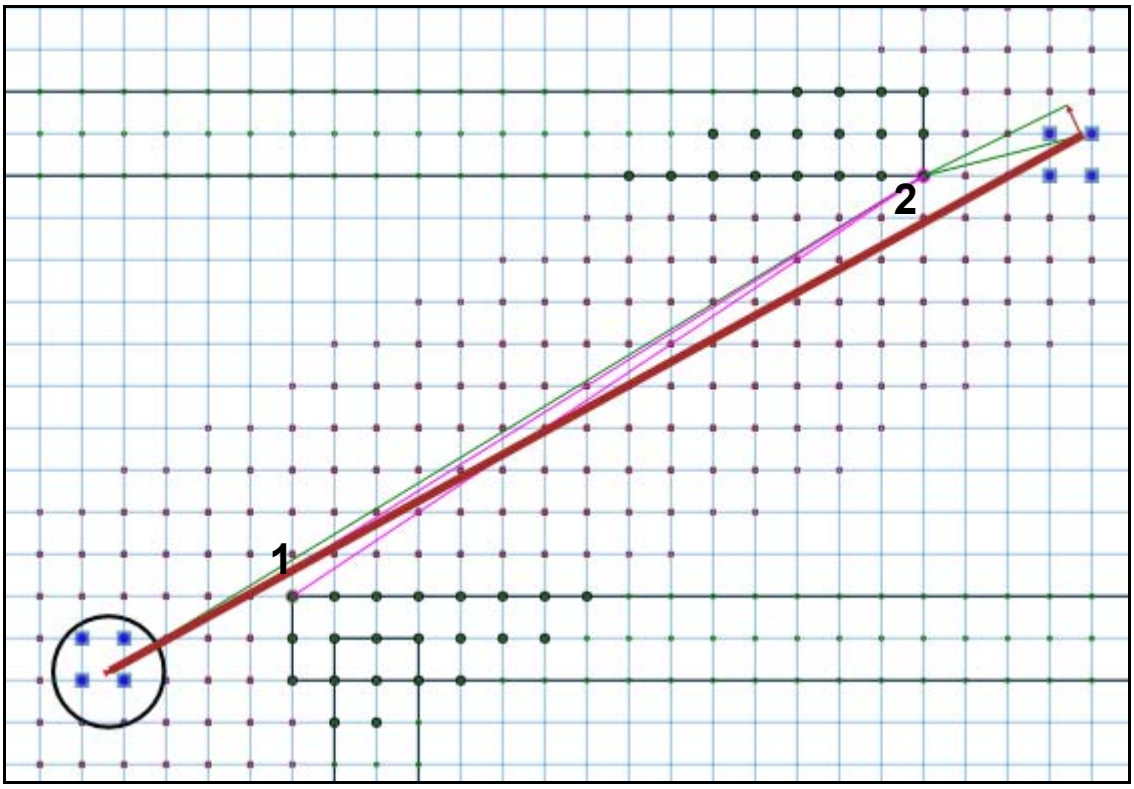
Şekil 87. Robotun uç noktasının içinde kaldığı grid karesinin köşe noktaları



Şekil 88. Robotun uç noktasının başka bir yönde hareketi

Şekil 88'de uzvun uç noktasının başka bir yönde hareketi görülmektedir. Bu örnekte uç noktasının hareket vektörü uç noktasının içinde kaldığı grid karesinin köşe noktalarıyla kesişmemekte ve 1 ve 2 nolu engel noktalarından yeteri kadar uzakta bulunmaktadır. Bu durumda robot serbest bölgede hareketini sürdürmektedir.

Şekil 82b'de verilen iki engel noktası arasında kalan durumda uzuv iki nokta arasında kayarak ilerlemektedir. Bu durumda iyileştirmeler yapılmıştır (Şekil 89). Öncelikle şekilde 2 noktası etrafında saat tersine dönme sonucu 1 noktasına çok yaklaşılıp yaklaşılmadığı kontrol edilmektedir. Eğer yaklaşım miktarı limitlerin dışarısında ise dönmeye izin verilmektedir. Dönme miktarı da ayrıca her hareket için sınırlanmaktadır. Eğer herhangi bir hareket sonucu limit içine girme durumu varsa hareket yine sınırlanmakta ve limit ihlal edilmeyecek şekilde dönme miktarı belirlenmektedir. Eğer uzuv şekildeki gibi dönmeye izin verilmeyecek bir durumdaysa ileri doğru kayma yapılmaktadır.



Şekil 89. İki engel arasında kayma durumu

Yukarıda belirttiğimiz durumlar dışında bir de köşe noktalarında özel bir durum vardır. Köşe durumu bir grid karesinde sadece bir engel noktası olmasına rağmen diğer o engel noktasının hemen yatay grid noktasında ve yan grid noktasında birer engel noktası olması durumunda ortaya çıkmaktadır. Algoritma bunu anladığında bu duruma özel olarak uzvun uç noktasını engelden belli bir mesafede tutmaktadır.

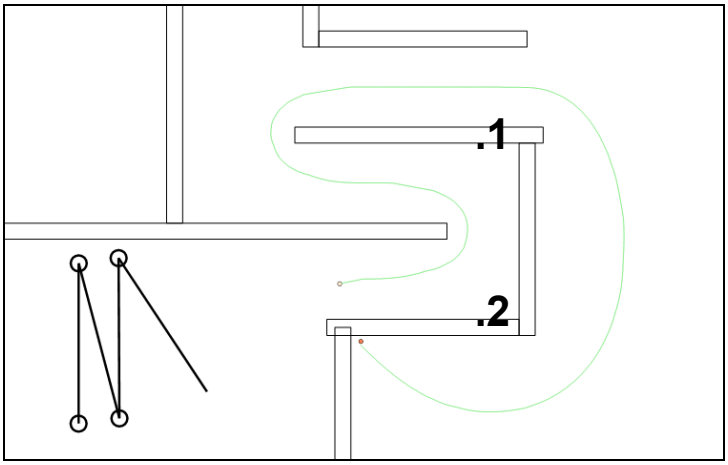
7.5 Uzuvların Ortak Çalışarak Engellerden Kaçınması

Yukarıda anlatılan teknikler kullanılarak tek bir uzvun engellerden kaçınarak hedefe ulaşması sağlanmaktadır. Fakat esas problem olan robotun bütün uzuvlarının engellerden kaçınarak hedefine ulaşması için uzuvlar arasında bir koordinasyonun sağlanması gerekmektedir. Koordinasyon olmadığı takdirde uzuvlar birbirinden kopuk vaziyette hedefe ulaşacaktır ki bu tabii olarak istenmeyen bir durumdur.

Kısaca ifade edilirse koordinasyon algoritması, bir uzuv hareket ettikten sonra bu uzvun bir önündeki ve bir arkasındaki uzuvları da hareket ettirerek, bahsedilen uzvun bir sonraki hareketine geçmesi olarak ifade edilebilir.

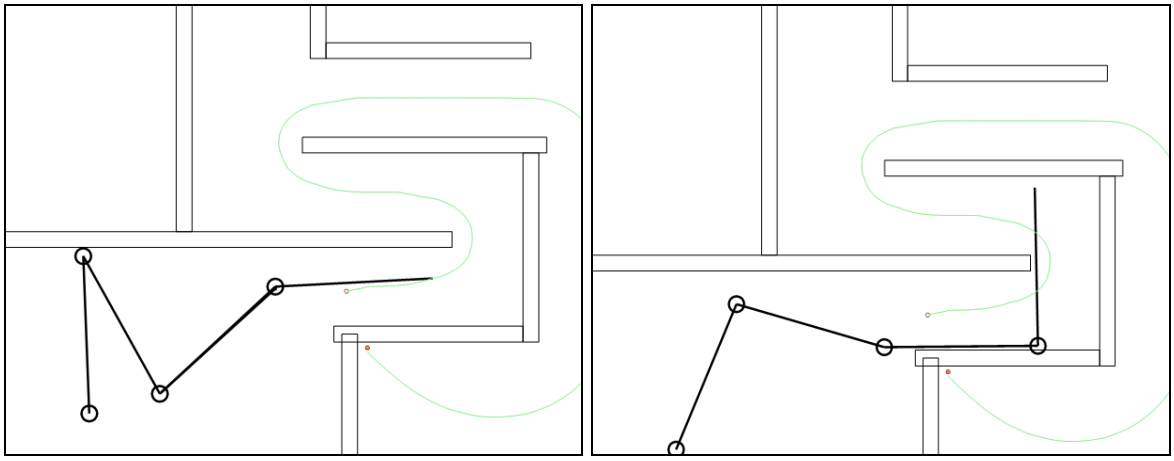
7.6 Tüm Robotun Engellerden Kaçınması İçin 1. Örnek

Aşağıda 4 uzuvlu bir robotun engellerden kaçınarak hedefe ulaşırken çekilmiş resimleri görülmektedir. Robot uzuvlarının uzunlukları, robotun fiziksel olarak geçebileceği en kısa uzunluk olan Şekil 90'da gösterilen 1 ve 2 numaralı noktaların arasındaki uzaklıktan sadece bir grid uzunluğu kadar kısadır. Başka bir değişle uzuvlar içinden geçmek için hemen hemen en zor durumdadırlar.

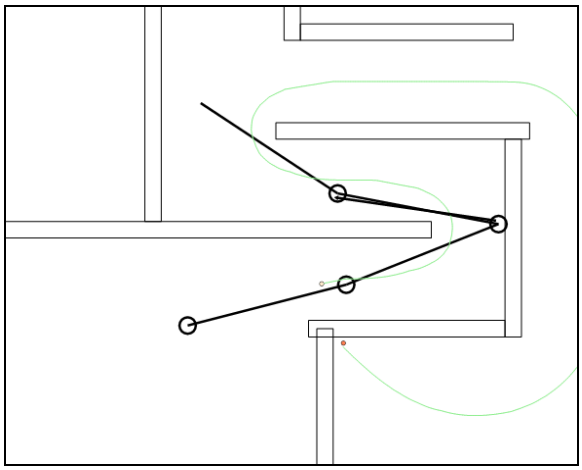


Şekil 90. 1. örnek için robotun başlangıç pozisyonu ve engellerin yerleşimi

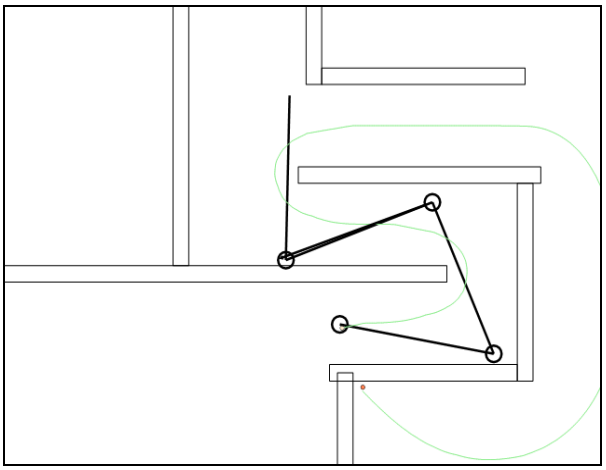
Şekil 91 a-k incelendiğinde 4 uzuvlu robot engeller arasından engellere çarpmadan ilerleyip hedefini bulmaktadır.



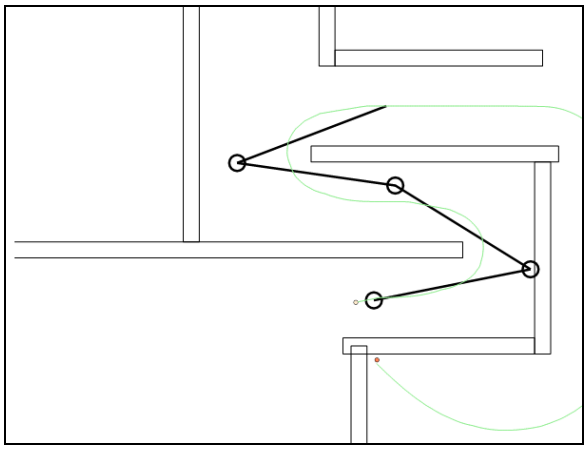
a b



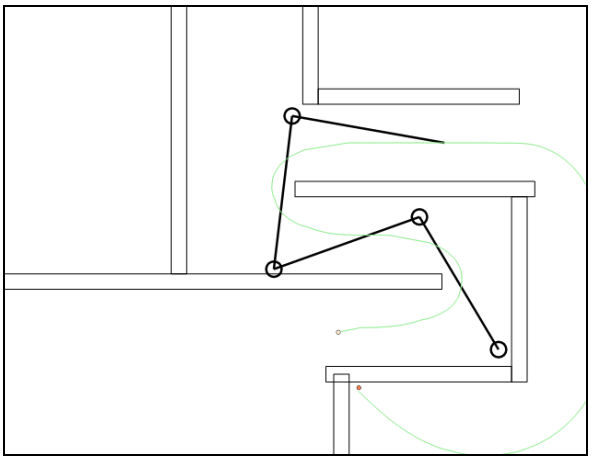
c



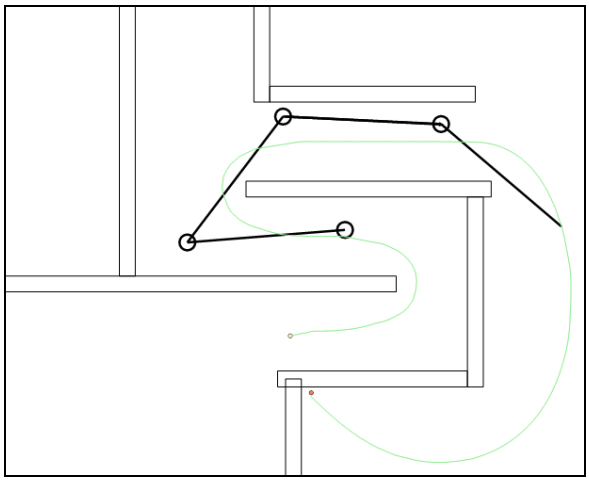
d



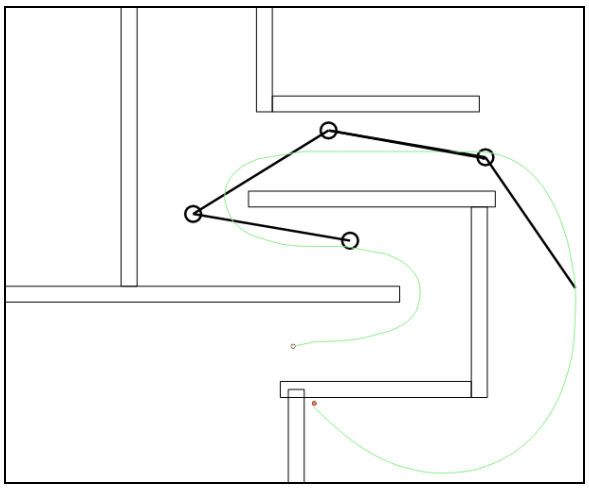
e



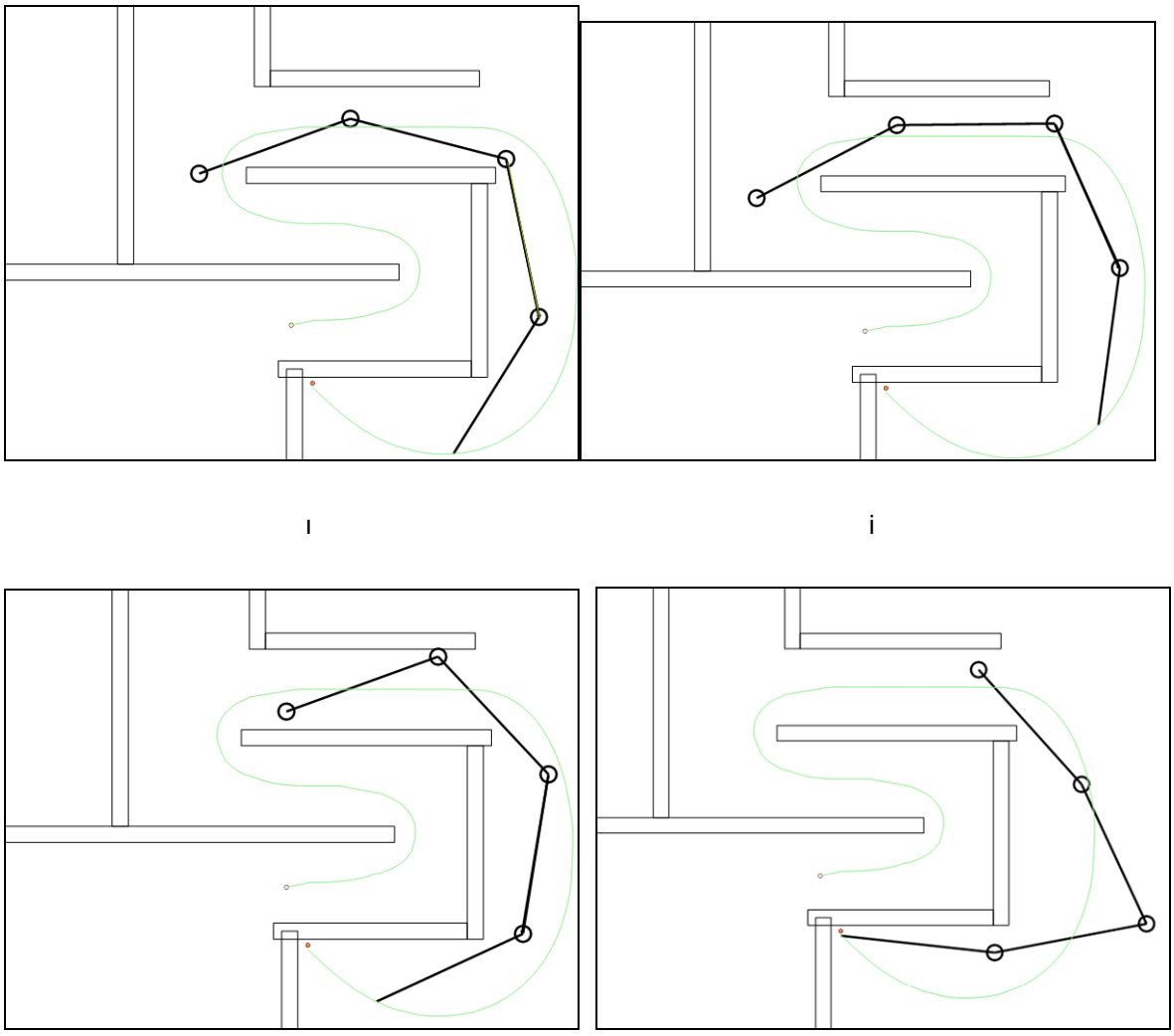
f



g



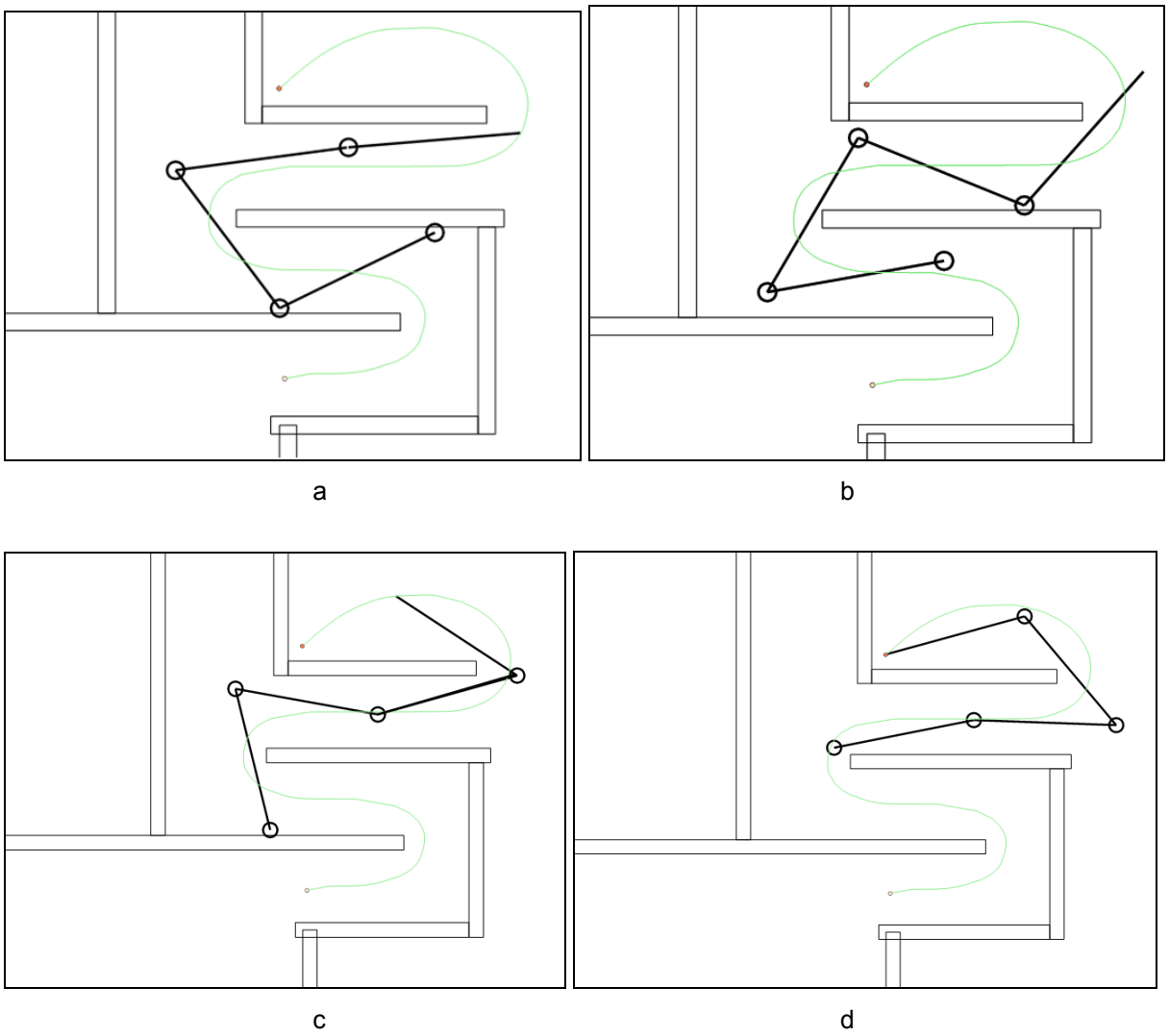
h



Şekil 91 a-k. 1. örnek için robotun hedefine ulaşırken çekilmiş resimleri

7.7 Tüm Robotun Engellerden Kaçınması İçin 2. Örnek

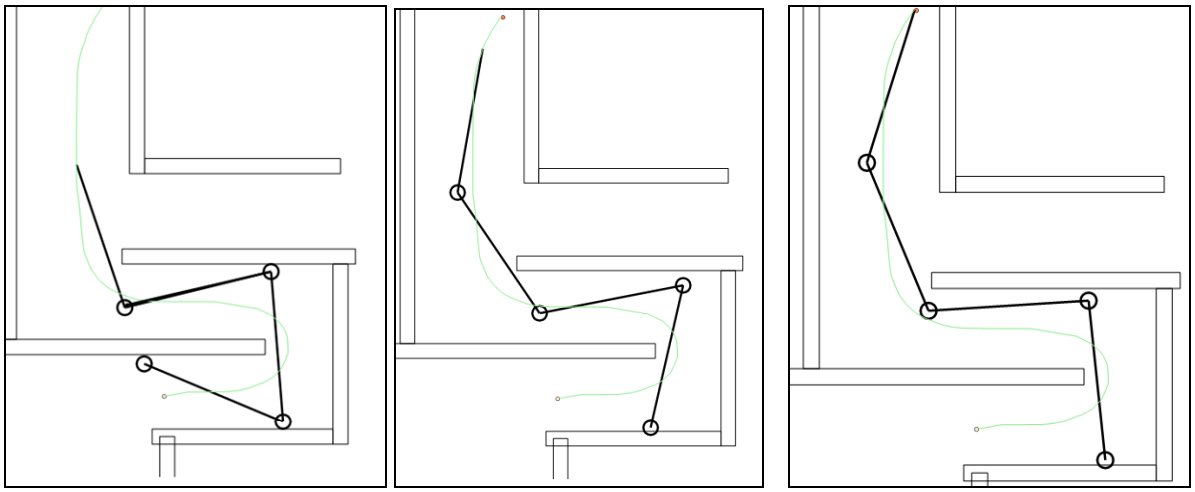
Bu örnek de bundan önceki örneğe benzemektedir. Robotun başlangıç noktası aynı olmakla beraber hedef noktası farklıdır. Robot aynı noktadan başladığı için robotun yukarıdaki örnekle benzer olan resimleri verilmemiştir. Şekil 92 a-d'de robotun hedefe ulaşması görülmektedir.



Şekil 92 a-d. 2. örnek için robotun hedefine ulaşırken çekilmiş resimleri

7.8 Tüm Robotun Engellerden Kaçınması İçin 3. Örnek

Bu örnekte de robot aynı başlangıç noktasından yeni bir hedef noktasına ulaşmaktadır (Şekil 93 a-c).

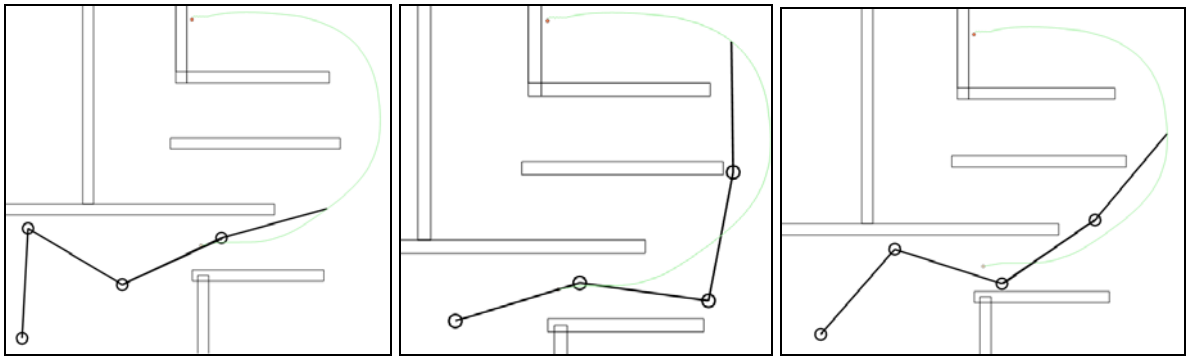


a b c

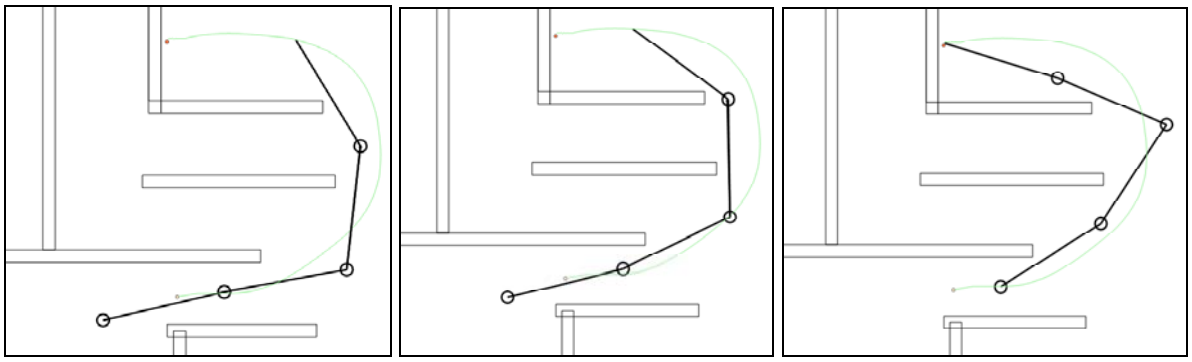
Şekil 93 a-c. 3. örnek için robotun hedefine ulaşırken çekilmiş resimleri

7.9 Tüm Robotun Engellerden Kaçınması İçin 4. Örnek

Bu örnekte robot yine diğer örneklerde bulunan başlangıç noktasında bulunmaktadır. Fakat bu sefer çok dar bir alandan geçmesi gerekmemektedir. Bunu sağlamak için alanı darlaştıran yatay bir engel kaldırılmıştır(Şekil 94 a-f).



a b c



d e f

Şekil 94 a-f. 4. örnek için robotun hedefine ulaşırken çekilmiş resimleri

7.10 Engellerden Kaçınma Algoritması İçin Yorum

Engellerden kaçınma algoritmasının birinci hedefi “gerçek zamanda engellerden kaçınabilmesi”, diğeri de “tamam (complete) bir algoritma” olmasıdır. Tamam bir algoritmayla ifade edilmek istenen şey fiziksel olarak mümkün olmak şartıyla her durumda verilen işi başarabilmesidir. Bugüne kadar geliştirilmiş algoritmalar bildiğimiz kadarıyla “tamam” değildir, bazı durumlarda çalışırlar fakat bazı durumlarda başarısız olurlar.

Burada geliştirdiğimiz algoritma gerçek zamanda engellerden kaçınabilmektedir ve böylece projede belirlediğimiz hedefleri tutturmuştur. Fakat bu haliyle iki eksikliği vardır. Birincisi yukarıdaki örneklerde görüldüğü gibi sadece 4 uzuvlu robotlar için çalışabilmektedir. Bunun sebebi algoritmayı genelleştirmek için yeteri kadar zamanımızın olmaması ve öncelikle çalışmasını yeterli görmemizdir. İkinci eksiklik ise henüz algoritmanın “tamam” bir algoritma olmamasıdır. Fakat “tamam” bir algoritmaya çok yakın sonuçlar vermiştir: yaptığımız çok sayıda denemede %90 oranında başarılı olmuştur. Belli bir miktar daha çalışıldığında son derece gelişmiş bir algoritmaya sahip olacağımızı düşünüyoruz.

Ekte verilen CD’de **104M260_ESahin_CONKUR_robotunEngellerdenKacinmasi01.wmv, 104M260_ESahin_CONKUR_robotunEngellerdenKacinmasi02.wmv** ve **104M260_ESahin_CONKUR_robotunEngellerdenKacinmasi03.wmv** isimli videoda algoritmanın değişik hedeflere ulaşması gözlemlenebilir.

Not: Bu videolarda bazen uzuvlar arası kopmalar ve çift çizimler gözükmektedir. Bunun sebebi uzuvlar arası ilişkileri kurarken, robotun ara hesaplamaları sırasında robotun ekrana istenmeden çizilmesi sebebiyledir. Düzeltmedeki zorluklardan dolayı bunun çözümü ertelenmiştir.

8 Robotların Çalıştırılması ve Deneylerin Yapılması

Robotumuzu imal ettikten ve çalışma odamıza kurduktan sonra ilk yapmak istediğimiz şey, motorları hareket ettirerek robotun hareket ettiğini gözlemek oldu. Geliştirdiğimiz yazılım ile motorları çalıştırmak zor olmadı ve robotu belli bir zaman bu şekilde çalıştırarak bilgisayardan elle kontrol ettik. Robotun her bir uzvu maksimum ±300 derece civarlarında dönebilmektedir. Tabii bu bir uzvun diğer herhangi bir uzuvla çarpışma durumu sözkonusu olmadığı durumlarda geçerlidir.

Uzuvlara bağlı motorların redüktörlerinin çevirme oranı 1/142.374999999999999 gibi yüksek bir değer olmasına karşın, motorların maksimum hızı 3000 dev/dak olduğundan uzuvlar gerektiğinde çok hızlı hareket edebilmektedirler.

Esas yapmak istediğimiz iş geliştirdiğimiz kontrol algoritmasını robot üzerinde denemekti. Fakat bunu yapmak için bazı küçük işleri bitirmemiz gerekiyordu. Bunlardan ilki motorlara gönderilecek hareket mutlak veya relatif hareket komutları arasında tercih yapmaktı. Biz aşağıda bahsedeceğimiz gibi relatif hareket komutunu tercih ettik. İkincisi, algoritma tarafından radyan olarak üretilen uzuv açısını redüksiyon oranını dikkate alarak motor pulse'una dönüştürmekti. Üretilen açı değeri redüksiyon oranından dolayı 142.374999999999999'a bölüneceğinden, üretilen açı değerini bu sayı ile çarptık. Ayrıca üretilen açı değerinin ne kadar pulse ettiğini hesaplamak gerekiyordu. 2π radyan 16384 pulse ettiğinden doğru orantı ile 1 radyanın 2607.59458761761 ettiğini kolayca hesaplanabilir. Sonuç olarak algoritmanın ürettiği açıya θ dersek

Motora yollanan dönme değeri= θ*142.374999999999999*2607.59458761761

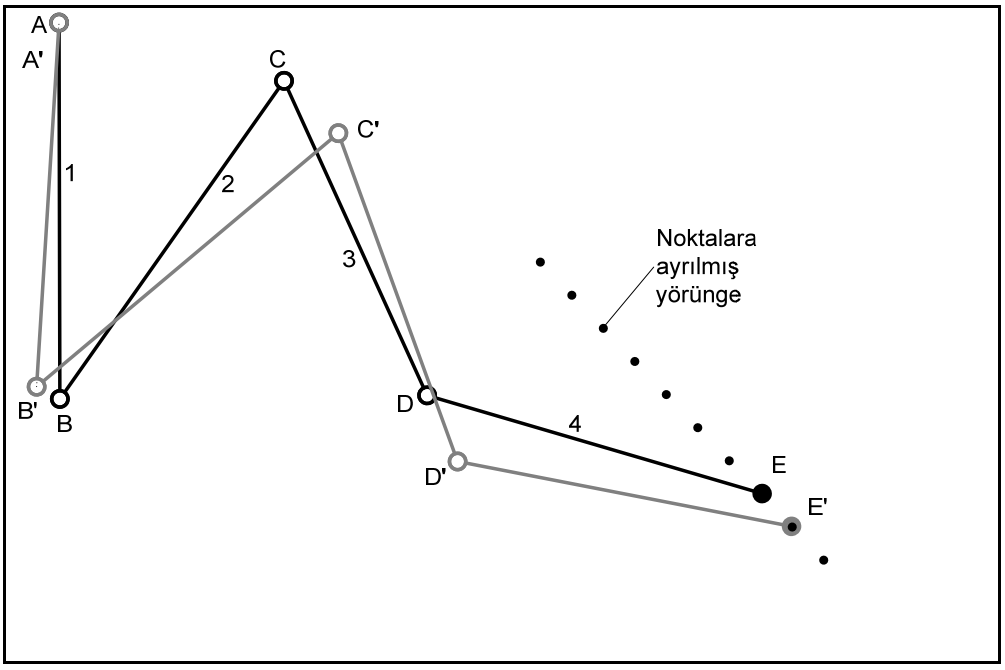
olarak çıkar. Fakat iş burada bitmemektedir. Uzuv için üretilen açı birim çemberin ikinci bölgesinde, uzvun bir önce bulunduğu konum üçüncü bölgede ise pulsa dönüştürülen açı değeri ulaşması gereken değere ters yönden diğer uzvu keserek gitmek istemektedir ki bu da fiziksel olarak imkansızdır. Bu problem bizi oldukça uğraştırdı. Uzun zaman yarım yamalak çalışan bir algoritmayla idare ettikten sonra algoritmamızı geliştirdik. Geliştirdiğimiz kod aşağıdadır.

```
for(int i=0;i<4;i++){
        //üretilen açı
       curA[i]=getM(manipForEdit)->getL(i)->thetaReal;
       //üretilen açı π/2'den büyükse ve bir önce üretilen açı -π/2'den küçükse
      if(curA[i]>(Math::PI/2.0) && previousA[i]<-(Math::PI/2.0)){
         // dönüştürülmüş açı
          curAConverted[i]=2.0*Math::PI+(previousA[i]-curA[i]);
       }
       //üretilen açı π/2'den küçükse ve bir önce üretilen açı π/2'den büyükse
       else if(curA[i]<-(Math::PI/2.0) && previousA[i]>(Math::PI/2.0)){
           // dönüştürülmüş açı
           curAConverted[i]=-2.0*Math::PI+(previousA[i]-curA[i]);}
      else curAConverted[i]=previousA[i]-curA[i]; //yukarıdaki durumlar yoksa
dönüştürülmüş açı
       previousA[i]=curA[i]; //bir önce üretilen açıyı güncelleştir
 }
```

8.1 Robotun Uç Noktasının Fare İle Hareketi

Bundan önceki raporda, robotun uç noktasını istenilen koordinatlara getirmek için gerekli uzuv açılarını hesaplaması için geliştirdiğimiz algoritmayı detaylı olarak açıklamıştık. Algoritmaya bilgisayar faresinin koordinatları robotun uç noktasının koordinatları olarak verildiğinde, robot bilgisayar ekranında fareyi takip edebilmektedir. Kullanıcı fareyi çok hızlı hareket ettirilirse noktalar arası uzaklık çok fazla olacaktır. Fakat algoritma için yeni noktayı bulmak zor olmamaktadır. Ayrıca iki nokta arasındaki mesafe fazla olursa noktalar arasındaki yol eğrisel olmaktadır. Şekil 95'de RoboKol ve RoboKol'un uç noktasının takip edeceği noktalardan oluşan örnek bir yol görülmektedir. Bilgisayarda simülasyonunu gerçekleştirdiğimiz bu ters kinematik algoritmayı gerçek robot üzerinde denedik ve gerçek robotun fare ile verdiğimiz hareketleri birebir gerçekleştirdiğini gözlemledik. Gerçekten RoboKol verdiğimiz emirleri yerine getiriyordu.

Ekte verilen CD'de **104M260_ESahin_CONKUR_2DGercekRobot.wmv** isimli videoda robotun çalışması görülebilir.



Şekil 95. RoboKol'un iki ayrı konfigürasyonu ve takip ettiği noktalar

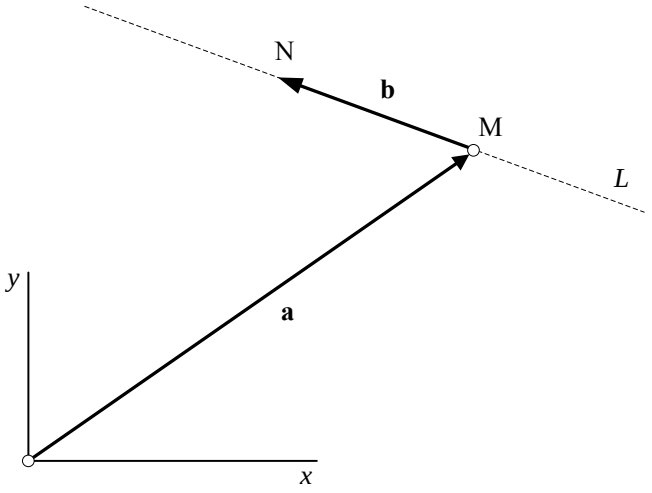
8.2 Robotun Uç Noktasının AutoCAD Programında Çizilen Şekilleri Takip Etmesi

Yukarıda bahsettiğimiz gibi robot kontrol algoritması robotun uç noktasını koordinatları bilinen bir noktadan yine koordinatları bilinen başka bir noktaya götürebilmektedir. Verilen bir yol eğer ard arda gelen ve birbirine çok yakın noktalara bölünürse robotun uç noktası bu noktaları takip ederek verilen yolu takip etmiş olacaktır.

RoboKol programına AutoCAD programında çizilen bir şekli alıp noktalara bölerek takip etmesini sağlayan bilgisayar kodunu geliştirip ekledik. Bunu kısaca şu şekilde açıklayabiliriz.

Bilindiği gibi AutoCAD programı vektör tabanlı bir programdır. Yani çizilen şekillerin geometrik özellikleri saklanır ve bu özellikler kullanılarak şekiller tekrar çizilir. Örneğin ekranda çizilmiş bir doğru parçası kaydedildiğinde esasında bu parçanın uç noktasının koordinatları, çizgi kalınlığı ve çizgi rengi gibi özellikleri kaydedilmiş olur. Bu dosya açıldığında bu bilgiler kullanılarak doğru parçası tekrar ekrana çizilir. Bitmap temelli programlarda ise şekil kaydedilirken çok küçük karecikler olan piksellerdeki renk bilgisi kaydedilir. Bu yüzden AutoCAD'de çizilen doğru paçası sabit diskte çok az yer kaplarken aynı doğru parçası bitmap olarak kaydedildiğinde çok daha fazla yer kaplar.

AutoCAD'de yapılan bir çizimin bütün özellikleri *.dxf sonekli standart bir text dosyasına kaydedilebilmektedir. AutoCAD'in yardımından bu özelliklerin nasıl kullanılacağı kolaylıkla öğrenilebilir.



Şekil 96. Doğrunun parametrik gösterimi

AutoCAD'de bir çizim yapıp bunu *.dxf dosyası olarak kaydettikten sonra RoboKol'da bu dosyayı açıp içini okuyarak içindeki nesneleri RoboKol'da çizdirecek kodu geliştirdik. Nesne olarak sadece "line" ve "arc" şimdilik yeterli oldu. Fakat istenirse daha çok nesne için kod geliştirilebilir.

Nesneleri ekrana çizdik fakat bu yeterli değildir. RoboKol'un bu nesnelerden oluşan yolu takip etmesi için yolun noktalara bölünmesi gerekir, yukarıda bahsetmiştik. Noktalara bölme için line" ve "arc" nesnelerinin parametrik denklemini kullanmak bize uygun geldi. Şekil 96'da görüldüğü gibi M noktasında başlayan ve N noktasında biten bir doğrumuzun olduğunu farzedelim. Bu doğru orijinde başlayan ve doğrunun başlangıcında biten **a** ve M de başlayan ve N'de biten **b** vektörleriyle temsil edilebilir. Ayrıca **b** vektörü bir *t* parametresiyle çarpılır. Sonuç olarak "doğru" aşağıdaki parametrik denklemle ifade edilir.

$$\mathbf{r}(t) = \mathbf{a} + t\mathbf{b}$$

Bu denklem **i** ve **j** birim vektörleri cinsinden şu şekilde yazılabilir.

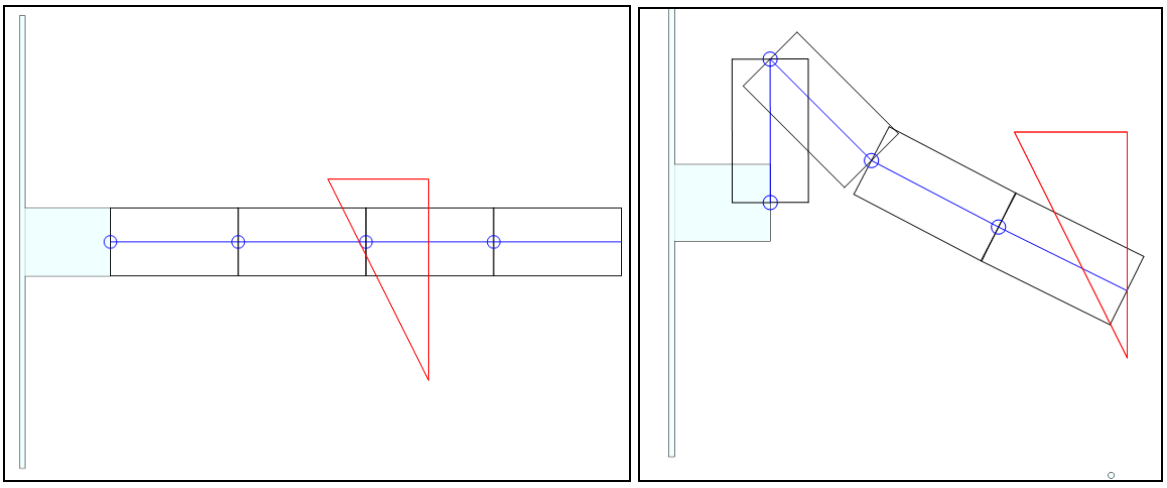
$$\mathbf{r}(t) = a_x\mathbf{i} + a_y\mathbf{j} + t\left(b_x\mathbf{i} + b_y\mathbf{j}\right)$$

Burada *t*'nin değeri sıfır ile bir arasında değişir. *t=0* olduğunda doğrunun başlangıç koordinatları yani M noktasının koordinatları elde edilir. *t=0.*5 olduğunda doğrunun

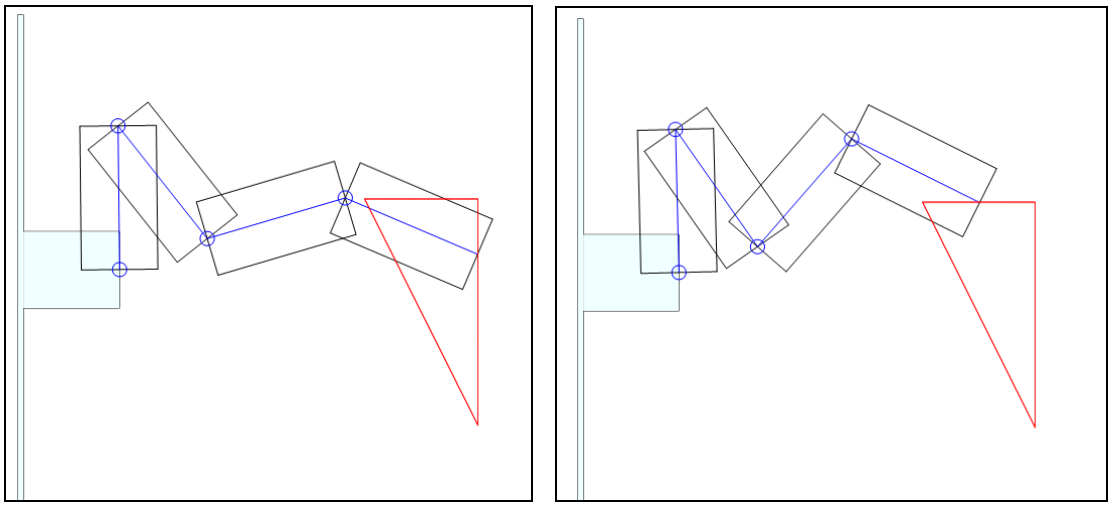
merkezinin koordinatları elde edilir. *t=1* olduğunda doğrunun uç noktasının yani N koordinatları noktasının koordinatları elde edilir. T sıfırdan başlanarak bire kadar örneğin 0.1'er arttırılarak götürülürse doğru 10 parçaya bölünmüş olur. 0.01'er arttırarak götürülürse doğru 100 parçaya bölünmüş olur. Arttırma miktarı veya hassasiyet istenildiği gibi ayarlanabilir. Benzer şekilde "daire parçası" da parametrik olarak aşağıdaki denklemle ifade edilebilir.

$$\mathbf{r}(t) = c\cos(t)\mathbf{i} + c\sin(t)\mathbf{j}$$

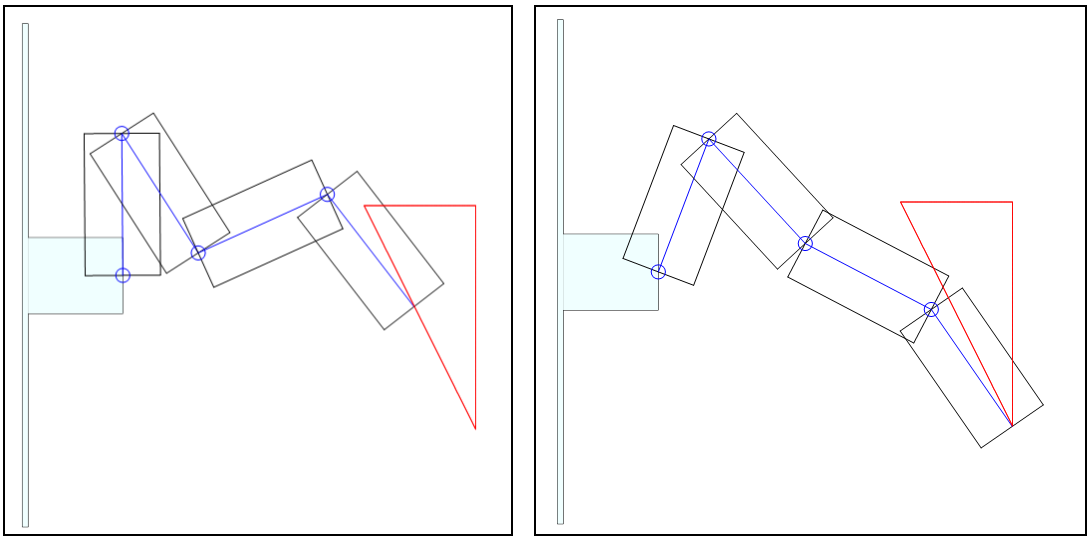
Burada *c* daire parçasının yarıçapıdır. Sonuç olarak Robokol'da bir yolun çizimini içeren *.dxf dosyası açılıp RoboKol'un bu yolu takip etmesi sağlanmıştır. Şekil 97 a-f'de RoboKol'un başlangıç konfigirasyonu ve AutoCAD'den alınmış bir üçgeni takip ederken çekilmiş görüntüleri yeralmaktadır.



a b



c d



e f

Şekil 97 a-f. RoboKol'un başlangıç konfigirasyonu ve AutoCAD'den atılmış bir üçgeni takip ederken çekilmiş görüntüleri

8.3 Robotun Kaynak Yapması

Daha önceden düşünmediğimiz fakat proje yürürken ortaya çıkan bir gelişme, robotumuza örnek bir uygulama olarak gazaltı kaynağı yaptırmaya karar vermemiz olmuştur. Bunun için gerekli olan gazaltı kaynak cihazını bir firmadan ödünç aldık ve yedi kişiyle üçüncü kattaki odamıza çıkarttık. Kaynak torçunu robotun uç noktasına bağlamak için bir aparat geliştirdik.

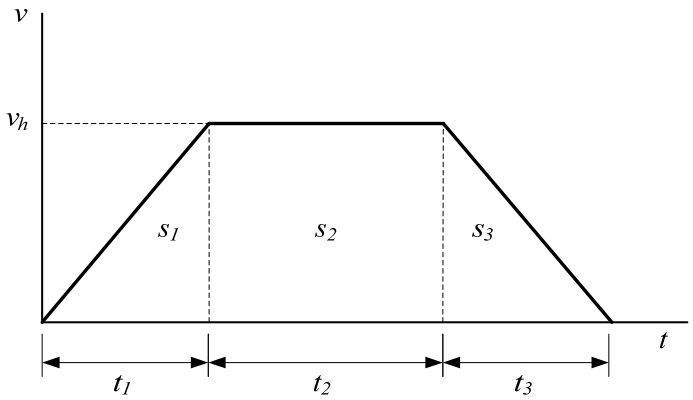
Kaynağın iyi yapılabilmesinde birçok faktör rol oynamaktadır. Bunlar arasında kaynak tel hızı, torç ile malzeme arasındaki mesafe ve akım sayılabilir. Bizim için önemli olan ise robotun uç noktasının hızıdır. Robotun uç noktası istenilen konuma gitmektedir. Fakat bütün motorların hız ve ivmelenme değerleri her zaman aynı olduğundan robotun uç noktasının hızını kontrol edemedik. Hızı sabitlemek için her noktada her bir motor için hız ve ivme değerlerini yeniden belirlememiz gerektiği sonucuna vardık.

Hız ve ivme ile ilgili değerlendirmeler basit gibi görünmesine karşılık, motorların nasıl çalıştığını deneyerek görmek gerektiğinden umduğumuzdan fazla zamanımızı aldığını söyleyebiliriz.

8.4 Hız ve İvme İncelemeleri

8.4.1 Hız Maksimum Hıza Ulaşabildiği Durum

Motora belli bir dönme değeri verildiğinde, motor verilmiş olan sabit ivme ile hızlanarak yine verilmiş olan maksimum hızına ulaşır, maksimum hızda belli bir zaman devam eder ve sabit negatif ivmelenmeyle durur. Hız-zaman grafiğinin altında kalan alan *pulse* olarak dönme miktarını verir. Burada bizim motorlar için 16384 *pulse* bir devir dönmeye karşılık gelmektedir.



Şekil 98. Motorun hız-zaman grafiği

s dönme miktarının, v_h hızının ve *a* ivmesinin verildiğini ve *t* toplam zamanın istendiğini farzedelim. Bu durumda ivmelenme için gereken t_1 zamanı

$$t_1 = \frac{v_h}{a}$$

ile bulunur. Toplam dönme miktarı s

$$s = s_1 + s_2 + s_3$$

ile verilir. Alanlar hesaplanırsa

$$s_1 = \frac{v_h \cdot t_1}{2}, s_2 = v_h \cdot t_2, s_3 = \frac{v_h \cdot t_3}{2}$$

bulunur. Pozitif ve negatif ivme değerleri aynı alınırsa

$$t_1 = t_3$$

olur. Bu durumda s

$$s = v_h \cdot t_1 + v_h \cdot t_2$$

yazılabilir. Buradan t_2

$$t_2 = \frac{s}{v_h} - t_1$$

çıkar.

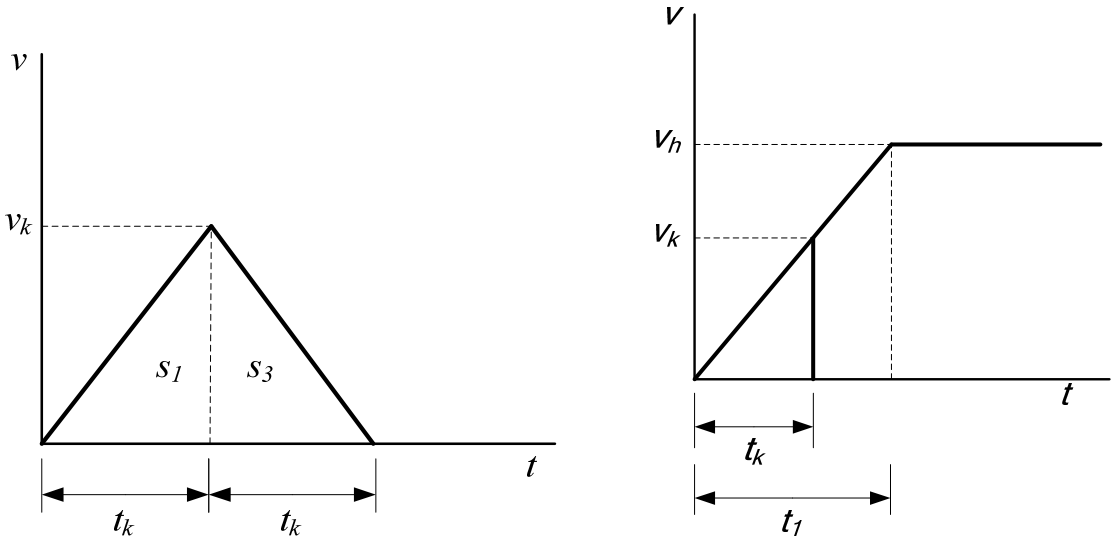
$$t = 2t_1 + t_2$$

olduğundan

$$t = t_1 + \frac{s}{v_h}$$

ile bulunur.

8.4.2 Hız Maksimum Hıza Ulaşamadığı Durum



Şekil 99. Hız maksimuma ulaşmadığı durumdaki hız-zaman grafiği

Hız miktarı çok fazla ve dönme miktarı çok az verildiğinde, hız maksimuma ulaşmadan hareket bitmektedir. Burada sadece pozitif ve negatif ivmelenme olmaktadır. Hızın ne kadar artacağı motor sürücüsü tarafından hesaplanmaktadır.

Yine yukarıdaki gibi s dönme miktarının, v_h hızının ve *a* ivmesinin verildiğini ve *t* toplam zamanın istendiğini farzedelim. Şekil 99'a bakarak ivmelenme için gereken v_k hızı üçgenlerin benzerliğinden

$$\frac{v_h}{v_k} = \frac{t_1}{t_k}$$

yazılarak

$$v_k = \frac{v_h \cdot t_k}{t_1}$$

bulunur. Pozitif ve negatif ivme değerleri aynı alınırsa

$$s_1 = s_3$$

olur. Bu durumda s

$$s = s_1 + s_3 = 2s_1 = 2\left(\frac{v_k \cdot t_k}{2}\right)$$

ile yazılır. Buradan t_k

$$t_k = \frac{s}{v_k} = \frac{s}{\frac{v_h \cdot t_k}{t_1}}$$

$$t_k = \sqrt{\frac{s \cdot t_1}{v_h}}$$

şeklinde elde edilir. Toplam zaman *t* ise

$$t = 2t_k = 2\sqrt{\frac{s \cdot t_1}{v_h}}$$

ile bulunur.

Örnek 1

$s = 4 \cdot 16384\ pulse$ (4 devir), $v = 16384\ pulse/s$, $a = 16384\ pulse/s^2$ veriliyor. t_1, t_2 ve t'yi bulunuz.

$$v = v_h$$

$$t_1 = \frac{v}{a} = \frac{16384}{16384} = 1s$$

$$t_2 = \frac{s}{v_h} - t_1 = \frac{4 \cdot 16384}{16384} - 1 = 3\ s$$

$$t = t_1 + \frac{s}{v_h} = 1 + \frac{4 \cdot 16384}{16384} = 5\,s$$

Örnek 2

Örnek 1'de her şey aynı kalmak şartıyla sadece hız $v = 2 \cdot 16384\ pulse/s$ şeklinde arttırılırsa sonuç ne olur?

$$v = v_h$$

$$t_1 = \frac{v}{a} = \frac{2 \cdot 16384}{16384} = 2\,s$$

$$t_2 = \frac{s}{v_h} - t_1 = \frac{4 \cdot 16384}{2 \cdot 16384} - 2 = 0$$

$$t = t_1 + \frac{s}{v_h} = 2 + \frac{4 \cdot 16384}{2 \cdot 16384} = 4\,s$$

Buradaki ilginç nokta hızın maksimum olması fakat maksimum olduğu anda negatif ivmelenmenin başlamasıdır. Bu sebepten s_2 alanı sıfır çıkmaktadır.

Örnek 3

Örnek 1'de her şey aynı kalmak şartıyla sadece hız $v = 4 \cdot 16384\ pulse/s$ şeklinde arttırılırsa sonuç ne olur?

Bu durumda hız maksimuma ulaşamadığından b şıkkında bulduğumuz denklemleri kullanacağız. Hız maksimuma ulaşabilseydi bu sırada geçen zaman 4 s olacaktı.

$$t_1 = \frac{v}{a} = \frac{4 \cdot 16384}{16384} = 4\,s$$

Aşağıda görüldüğü gibi hareket için toplam süre 4 s olmaktadır.

$$t = 2\sqrt{\frac{s \cdot t_1}{v_h}} = 2\sqrt{\frac{4 \cdot 16384 \cdot 4}{4 \cdot 16384}} = 4\,s$$

Hız $v = 4 \cdot 16384\ pulse/s$ verilmesine rağmen hızın ulaşabildiği en yüksek değer Örnek 2'de bulunan değerle aynı olan

$$v_k = a \cdot t_k = 2 \cdot 16384\ pulse/s$$

olmaktadır.

8.4.3 Verilen Zaman İçinde Hareketin Başlayıp Bitmesini Sağlayacak Hız ve İvme Değerlerinin Bulunması

Şekil 98'e tekrar gidersek, dönme miktarı s, t_1, t_2 ve t verilip hız ve ivme değerleri istenirse bu değerler aşağıdaki gibi bulunabilir.

$$s = 2s_1 + s_2$$

$$s = 2\left(\frac{v_h \cdot t_1}{2}\right) + v_h \cdot t_2 = (t_1 + t_2)v_h$$

$$v_h = \frac{s}{t_1 + t_2}$$

$$a = \frac{v_h}{t_1}$$

Örnek 4

$s = 16384\,pulse$, $t = 1s$ ve $t_1 = t_3 = \frac{1}{8}s$, $t_2 = \frac{3}{4}s$ olarak veriliyor. Buna göre bu değerleri gerçekleştirecek hız ve ivmeyi bulunuz.

c şıkkında çıkarılan hız denklemi kullanılarak

$$v_h = \frac{s}{t_1 + t_2} = \frac{16384}{\frac{1}{8} + \frac{3}{4}} = \frac{8}{7}16384\ pulse/s$$

olarak bulunur. İvme ise

$$a = \frac{v_h}{t_1} = \frac{16384}{\frac{1}{8}} = 8 \cdot 16384\ pulse/s^2$$

elde edilir.

Bulduğumuz değerleri kullanarak problemin sağlamasını yapabiliriz. Dönme miktarı s

$$s = (t_1 + t_2)v_h = \left(\frac{1}{8} + \frac{3}{4}\right)\frac{8}{7}16384 = \frac{8}{7}\frac{7}{8}16384 = 16284\ pulse$$

çıkar ki bu doğru değerdir.

s_1, s_2 ve s_3 değerleri de aşağıdaki gibi bulunur.

$$s_1 = s_3 = \frac{v_h \cdot t_1}{2} = \frac{\frac{8}{7}16384 \cdot \frac{1}{8}}{2} = \frac{1}{14}16384\,pulse$$

$$s_2 = v_h \cdot t_2 = \frac{8}{7}16384 \cdot \frac{3}{4} = \frac{6}{7}16384\ pulse$$

8.4.4 Yukarıda Bulunan Hız ve İvme Değerlerinin Robotun Uç Noktasının Hızının Sabit Tutulması İçin Kullanılması

Önceden de belirttiğimiz gibi, Şekil 95'de görülen robotun E noktasından E' noktasına gelmesi için gerekli 4 uzuv açısı algoritmamız tarafından üretilmektedir. Normalde her uzvun açısı birbirinden farklı olacaktır. Bu yüzden bütün motorlar için aynı hız ve ivme değerlerini kullanılırsa, bir uzuv hareketini bitirdiğinde diğeri hala hareketini sürdürüyor olabilecektir. Ayrıca uç noktasından uzaktaki uzuvların hareketi büyük olduğunda uç noktasının hızı artacaktır. Genel olarak robotun uç noktasının hızı değişken olacaktır.

Burada robotun uç noktasının hızını sabit tutmak için şöyle bir strateji geliştirdik. Yine Şekil 95'e dönersek, robotun E noktasından E' noktasına gelmesi için bütün motorlar için üretilen birbirinden farklı açıları Δt zamanı içinde gerçekleştirmek istediğimizi düşünelim. Bu durumda bütün motorların harekete aynı anda başlayıp aynı anda bitirmesi gerekir. Başka bir değişle Δt'yi belirlersek ve bütün motorların sadece bu zaman içinde çalışmalarının sağlarsak uç noktanın hızı sabit kalacaktır. Her motor için hareket miktarı ve zamanı belli olduğuna göre, bu değerleri gerçekleştirecek hız ve ivmeler her motor için ayrı ayrı c şıkkında gösterildiği gibi bulunarak motorlar hareket ettirilirse uç noktasının hızı sabit kalacaktır.

Şekil 100'de hızın kontrol edilmediği ve edildiği durumlarda yapılan kaynaklar görülmektedir. Şekilde1 ve 2 numara ile gösterilen kaynaklarda hız kontrolü yoktur. Fakat 3 numara ile gösterilen kaynakta hız kontrolü yapılmıştır. (Düz kaynak çekilmek istenmiştir. Fakat özellikle baş kısımlarda olan eğrilikler kaynak torcunun hortumunun takılması sonucu oluşmuştur.)

Şekil 100. Hız kontrol edildiği ve edilmediği durum

Şekil 101'de RoboKol'un yaptığı kaynaklardan örnekler verilmiştir. Şekilde görülen kaynaklardan kalın olanlar AutoCAD'de zigzag şeklinde yapılan çizimler kullanılarak gerçekleştirilmiştir.

Şekil 101. RoboKol'un yaptığı kaynaklardan örnekler

9 Projenin Başarı Kriterleri İle İlgili Olarak Genel bir Değerlendirme

Proje başvurusunda belirttiğimiz başarı kriterleri ve proje bitimindeki gelişmeler aşağıda değerlendirilmiştir.

1. ***Yöntem*** **ve** ***Kapsam*** **bölümlerinde** ***Hareket Planlaması*** **başlıkları altında sıralanan 1 ve 2 numaralı problemlerin çözümü için çalışılacaktır. Bunlardan birinin çözüme kavuşturulması planlanmıştır. Ya 1 numaralı problemin yani "keskin virajlarda engellerle çarpışma" ya da 2 numaralı problemin yani "robot uzuvlarının bir kısmının potansiyel alanın yönlendirmesine ters olarak hareket etmesi" çözülmesi düşünülmektedir.**

   2 numaralı problemin yani "robot uzuvlarının bir kısmının potansiyel alanın yönlendirmesine ters olarak hareket etmesi" Bölüm 6.2.2'de verilen algoritmayla çözülmüştür. 1 numaralı problemin yani "keskin virajlarda engellerle çarpışma" ise Bölüm 7'de verilen algoritmayla çözülmüştür.

2. ***Yöntem*** **ve** ***Kapsam*** **bölümlerinde** ***Hareket Planlaması*** **başlıkları altında sıralanan 3 numaralı problemin çözümü yani "potansiyel alanı üç boyutta temsil edecek kodu yazmak ve cisimleri (en azından robotu) üç boyutlu olarak görüntüleme" düşünülmektedir.**

   Bölüm 5.4'de 2 ve 3 boyutlu potansiyel alan kodu yazılmıştır. Hem robotu hem de engelleri görüntüleyebilen ve istenilen açıdan görülmesini sağlayan direct X tabanlı kod geliştirilmiştir.

3. ***Yöntem*** **ve** ***Kapsam*** **bölümlerinde** ***Hareket Planlaması*** **başlıkları altında sıralanan 4 numaralı problem olan "hesaplamaların gerçek zamanlı olup olmaması" probleminin en azından Robot Kolu hareket planlaması için çözülmesi düşünülmektedir.**

   Yazmış olduğumuz algoritmalar optimize edilmemesine rağmen "gerçek zamanlı olarak çalışmaktadır. Burada herhangi bir problem çıkmamıştır. Buna bilgisayarların hızlanmasının da katkısı vardır. Sadece 3 boyutlu potansiyel alan hesaplaması birkaç saniye almaktadır. Bu hesap robotun hareketi başlamadan bir defalığına yapıldığından pratikte gerçek zaman olarak kabul edilebilir.

4. ***Yöntem* ve *Kapsam* bölümlerinde *Mekanik Dizayn* başlıkları altında bahsedilen "birbirine mafsallarla bağlı uzun ve ince seri uzuv robotun" dizaynı yapılacaktır.**

   Bölüm 2'de detayları verilen dört kollu "birbirine mafsallarla bağlı uzun ve ince seri uzuv robot"un dizaynı ve imalatı yapılmıştır.

5. **Dizayn yapıldıktan sonra, RoboKol'da hazır olan iki algoritma "settling ve point settling" algoritmaları ve yeni geliştirilecek algoritmalar denenecektir.**

   Robotun dizaynı ve imalatı yapıldıktan sonra yeni geliştirdiğimiz hareket algoritmasıyla çok başarılı denemeler yapılmıştır. Ortaya çıkan gelişmeler sonucu, "settling ve point settling" algoritmalarını denemek yerine, robotun AutoCad'de çizilen yolları alarak takip etmesinin ve gazaltı kaynağı yapmasının daha verimli olduğu düşünülmüştür.

6. **En azından "settling" algoritmasının üç boyutlu hali ve yeni bir üç boyutlu *mekanik* dizayn gerçekleştirilecektir.**

   Sonuç raporuna kadar "settling" algoritması yerine, 2 boyut için bu proje kapsamında yeni geliştirdiğimiz hareket algoritmasını kullanan 3 boyutlu bir algoritma tamamlanmıştır. Biz "settling" algoritması demişiz, fakat yukarıda bahsettiğimiz gelişmeler sonucu yeni algoritmanın 3 boyutlu hale getirmenin daha mantıklı olacağını düşünüyoruz.

   Üç boyutlu mekanik dizayn eldeki imkanlar nisbetinde Bölüm 2.2'de belirtildiği gibi gerçekleşmiştir.

7. **Uzuvlara sersörler eklenerek, sensörlerden gelen bilgi bilgisayara aktarılacaktır.**

   Bölüm 3'de gösterildiği gibi belli bir aralıkta hassas bir şekilde mesafe ölçen lazer sensörü çalıştırılmış ve mesafe bilgisi bilgisayara alınmıştır.

10 Sonuç ve Gelecekle İlgili Çalışmalar

Projemizin başlığı “Gereğinden Çok Serbestlik Dereceli Robotların Bir Ürün Olarak Geliştirilmesi” özellikle seçilmiş bir başlıktır. Buradaki temel motivasyon, araştırma çalışmalarımızı belli bir disiplin içinde birbirine ekleyerek sonuçta ortaya bir ürün çıkarmaktır. Bu projede bu konuyla ilgili teorik ve pratik olarak çok sayıda çalışma yapılmış ve bu hedefe yönelik oldukça fazla mesafe kat edilmiştir. Proje teklifini sunarken tasarladığımız hedefler tutturulmuş, bunlara ek olarak endüstriyel otomasyonla ilgili önemli miktarda bir bilgi birikimi elde edilmiştir.

Projede yapılan çalışmalardan bir bölümü, robotun mekanik dizaynını ve üretimini yapmakta yoğunlaşmıştır. 2 boyutlu ve 4 serbestlik dereceli ve 3 boyutlu yine 4 serbestlik dereceli robotlar dizayn ve imalat edilmiştir.

Diğer bir çalışma alanımız, robotu bilgisayardan kontrolü için gerekli donanımın kurulması, çalıştırılması ve servo motorların tüm parametreleri kontrol edilebilecek şekilde yazılımların geliştirilmesi şeklinde gelişmiştir.

RoboKol isimli tüm yapılan yazılım çalışmalarının entegre edildiği kapsamlı bir bilgisayar programı geliştirilmiştir. Bu program hem algoritma geliştirilmesi için kullanılmakta hem de geliştirilen bu algoritmaları kullanarak robot kontrolünü gerçekleştirmektedir. RoboKol programı, istenildiği sayıda pencerede mouse ile istenildiği açıdan bakılabilen çok gelişmiş bir 3 boyutlu görüntülemeye sahiptir. Ayrıca program 2 ve 3 boyutlu uzayda çalışan potansiyel alan üretme özelliğine sahiptir.

Gereğinden çok serbestlik dereceli robotlar için “serbest bölgede” çalışan orijinal ve son derece verimli olan bir ters kinematik algoritması geliştirilmiştir. Algoritmanın özellikleri arasında basitlik, gerçek zamanlı çalışabilme, eklenebilirlik, engeller olma durumunda engellerden kaçınmaya yardımcı olma, robotu oluşturan uzuv sayısından bağımsız çalışma sayılabilir.

Gereğinden çok serbestlik dereceli robotlar için fiziksel olarak mümkün olduğu müddetçe robot engeller arasında manevra yaparak ve engellere çarpmadan ilerlemesini sağlayan “engellerden kaçınma” algoritması geliştirilmiştir. Böylece fazla serbestlik derecelerinin verimli bir şekilde kullanımı başarılmıştır.

Yazılım, kontrol sistemi ve robotu içeren tüm sistem başarılı bir şekilde çalıştırılmış ve bilgisayarın ürettiği sonuçlar fiziksel ortama aktarılmıştır. Örnek bir uygulama olarak robota gazaltı kaynağının yaptırılmıştır.

Projede çalışmaları çoğunlukla 2 boyutta olmuştur. Bunun basitçe sebebi, 2 boyutta çalışmanın daha kolay olmasıdır. Bununla beraber, 3 boyut için serbest bölgede çalışan 2 boyutlu algoritmanın devamı niteliğinde bir algoritma geliştirilmiştir. 3 boyutlu görüntüleme çalışmaları da yapılmış ve böylece daha sonraki çalışmalar için altyapı hazırlanmıştır.

Bundan sonraki çalışmada birinci hedef esas hedef 3 boyutta çalışan ve engellerden kaçınan gerçek zamanlı bir algoritma geliştirmektir. Şu anda geliştirdiğimiz 2 boyut için olan algoritmadaki temel fikir çalıştığı için, aynı algoritma 3 boyuta da uygulanabilecektir. İkinci hedef ise motorların uzuvlar üzerine yerleştirildiği ve her uzvun 2 serbestlik derecesine sahip olduğu toplam 8 serbestlik dereceli bir robot dizaynıdır.

Bu çalışmalar dışında dışarıdan veri almada ve işlemede çalışmalar yapılması gerekmektedir. Veri alma ile ilgili olarak bu projemizde çok verimli çalışmalar yapılmıştır. Fakat bu çalışmalara ek olarak görüntü alma ve işleme konusunda çalışılmalar yapılması gerekmektedir. Böylece komple bir sisteme sahip olunabilecektir.

Projede henüz SCI'de yayın yapılmamıştır. Fakat şu anda "engellerden kaçınma algoritması" yayına hazırlanmaktadır. "Serbest Bölge"de çalışan algoritmayı da yayın olarak hazırlıyoruz. Bunlara ek olarak, "engellerden kaçınma algoritması" gerçek zamanlı olarak kolaylıkla çalıştığı ve algoritmanın yapısı müsait olduğu için "hareketli engeller" durumunda da çalışacağını düşünüyor ve kısa bir çalışma ile buradan da bir yayın planlıyoruz.

Referanslar

[1]. Conkur E. S., Buckingham R., Clarifying the definition of redundancy as used in robotics, *Robotica* 15 (5), 583-586, (1997).

[2]. Ma S., Hirose S., Yoshinada H., Development of a hyper-redundant multijoint manipulator for maintenance of nuclear reactors, *Advanced Robotics*, 9 (3), 281-300, (1995).

[3]. Buckingham R., Towards safe active robotic devices for surgery, *Industrial Robot*, 20 (2), 8-11, (1993).

[4]. Maciejewski A. A., Klein C. A., Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments, *The International Journal of Robotics Research,* 4 (3), 109-117, (1985).

[5]. Chen J. L., Liu J. S., Lee W. C., Liang T. C., On-line multi-criteria based collision-free posture generation of redundant manipulator in constrained workspace, *Robotica,* 20 (6), 625–636, (2002).

[6]. Nenchev D. N., Redundancy resolution through local optimisation: a review, *Journal of Robotic Systems,* 6 (6), 769-798 (1989).

[7]. Boddy C. L., Taylor J. D., "Whole arm reactive collision avoidance control of kinematically redundant manipulators", *IEEE International Conference on Robotics and Automation* (1993), pp. 382-387.

[8]. Nakamura Y., *Advanced robotics*, *redundancy and optimisation* (Addison-Wesley Pub. Company, Reading, (1991).

[9]. Latombe J., *Robot motion planning* (Kluwer Academic Publishers, USA, (1991).

[10]. Hsu D., Latombe J., Motwani R., Path planning in expensive C-spaces, *IEEE International Conference on Robotics and Automation* (1997), pp. 2719-2726.

[11]. Amato N., Wu Y., A randomised roadmap method for path and manipulation planning, *IEEE International Conference on Robotics and Automation* (1996), pp. 113-120.

[12]. K. Gupta K., Fast collision avoidance for manipulator arms: a sequential search strategy, *IEEE Transactions on Robotics and Automation* 6 (5), 522-532 (1990).

[13]. Hayashi A., Geometric motion planning for highly redundant manipulators using a continuous model (PhD Thesis), The University of Texas at Austin, (1994).

[14]. Khatib O., Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research,* 5 (1), 90-98, (1986).

[15]. Janabi-Sharifi F., Vinke D., Robot path planning by integration the artificial potential field approach with simulated annealing, *IEEE International Conference on Robotics and Automation* (1993), pp. 282-287.

[16] Kim S. W., Boley D., Building and navigating a network of local minima, *Journal of Robotic Systems,* 18 (8),405–419, (2001).

[17]. Connolly C., Grupen R., "The application of harmonic functions to robotics", *Journal of Robotic Systems* 10 (7), 931–946 (1993).

[18]. Rimon E., Koditschek D. E., Exact robot navigation using artificial potential functions*, IEEE Transactions on Robotics and Automation,* 8 (5), 501-517, (1992).

[19]. Faverjon B., Tournassoud P., A local based approach for path planning of manipulators with a high number of degrees of freedom, *IEEE International Conference on Robotics and Automation* (1987), pp. 1152-1159.

[20]. Reznik D., Lumelsky V., Sensor-based motion planning in three dimensions for a highly redundant snake robot, *Advanced Robotics,* 9 (3), 255-280, (1995).

[21]. Fahimi F., Ashrafiuon H., Nataraj C., Obstacle avoidance for spatial hyper-redundant manipulators using harmonic potential functions and the mode shape technique, *Journal of Robotic Systems,* 20 (1), 23-33, (2003).

[22]. Mochiyama H., Kinematics of the whole arm of a serial-chain manipulator, *Advanced Robotics,* 15 (2), 255-275, (2001).

[23]. Li J. Z. , Trabia M. B., Adaptive path planning and obstacle avoidance for a robot with large number of redundancy, *Journal of Robotic Systems,* 13 (3), 163-176, (1996).

[24]. Choi P. J., Rice J. A., Cesarone J. C., Kinematics of an indefinitely flexible robot arm, *Journal of Robotics Systems,* 10 (4), 407-425, (1993).

[25]. Hannan M. W., Walker I. D., Kinematics and the implementation of an elephant's trunk manipulator and other continuum style robots, *Journal of Robotic Systems,* 20 (2), 45–63, (2003).

[26]. Ma S., Konno M., An obstacle avoidance scheme for hyper-redundant manipulators-global motion planning in posture space, *IEEE International Conference on Robotics and Automation* (1997), pp. 161-166.

[27]. S. Ma, I. Kobayashi, S. Hirose and K. Yokoshima, Control of a multijoint manipulator moray arm, *IEEE/ASME Transactions on Mechatronics*, 7 (3), 1-14 (2002).

[28]. S. Ma and I. Kobayashi, An obstacle avoidance control scheme for the Moray arm on the basis of posture space analysis, *Robotics and Autonomous Systems,* 32 (2-3), 163–172 (2000).

[29]. Chirikjian G.S., Burdick J.W., Parallel formulation of the inverse kinematics of modular hyper-redundant manipulators, in: Proceedings of the IEEE International Conference on Robotics and Automation (1991), pp. 708–713.

[30].www.cs.siu.edu/~mengxia/Courses%20PPT/591/07_VolumeVisualizationIntro.ppt

[31]. Colbaugh R., Seraji H., Glass K.L., Obstacle avoidance for Redundant robots using configuration control, journal of robotic systems, 6(6), 721-744, (1989).

[32]. Rahmanian-Shahri N., Troch I., A new on-line method to avoid collisions with links of redundant articulated robots, robotica, 14, 11-619,(1996).

TÜBİTAK
PROJE ÖZET BİLGİ FORMU

Proje No: 104M260

Proje Başlığı:

Gereğinden Çok Serbestlik Dereceli Robotların (Redundant Robots) Bir Ürün Olarak Geliştirilmesi

Proje Yürütücüsü ve Araştırmacılar:

Doç. Dr. E. Şahin ÇONKUR

Yrd. Doç. Dr. Abdullah T. TOLA

Projenin Yürütüldüğü Kuruluş ve Adresi:

Pamukkale Üniversitesi, Mühendislik Fakültesi,
Makine Mühendisliği Bölümü,
Makine Teorisi ve Dinamiği Anabilim Dalı,
Kınıklı Kampusu, 20070 Denizli, Türkiye

Destekleyen Kuruluş(ların) Adı ve Adresi: -

Projenin Başlangıç ve Bitiş Tarihleri:

01.04.2005 01.04.2008

Öz (en çok 70 kelime)

Gereğinden çok serbestlik dereceli robotlar, sahip oldukları fazla serbestlik derecelerini kullanarak standart robotlar için çok zor olan hareketleri yapabilen robotlardır. Mekanikte ve kontrolde sahip oldukları problemler bu tür robotların uygulama alanına geçmesini engellemektedir. Bu projede gereğinden çok serbestlik dereceli robotlar için yazılım, kontrol ve mekanik dizayn alanlarında çalışmalar yapılmıştır. Serbest ve engellerle dolu bölgelerde verilen hedefe gidebilecek şekilde gerçek zamanda uzuv açılarını ayarlayabilen algoritmalar geliştirilmiş ve dizayn ve imal edilen gerçek robot üzerinde denenmiştir.

Anahtar Kelimeler: Gereğinden çok serbestlik dereceli robotlar, hareket planlaması, potansiyel alanlar

Projeden Yapılan Yayınlar:

E. Şahin Çonkur, Abdullah T. Tola, "Yılan Benzeri robotlar için bir yörünge planlama algoritması" III. Otomasyon Sempozyumu, Pamukkale Üniversitesi, Kasım 2005, Denizli, Sayfa 166-169.

TÜBİTAK
PROJE ÖZET BİLGİ FORMU

Proje No: 104M260

Proje Başlığı:

Gereğinden Çok Serbestlik Dereceli Robotların (Redundant Robots) Bir Ürün Olarak Geliştirilmesi

Proje Yürütücüsü ve Araştırmacılar:

Doç. Dr. E. Şahin ÇONKUR

Yrd. Doç. Dr. Abdullah T. TOLA

Projenin Yürütüldüğü Kuruluş ve Adresi:

Pamukkale Üniversitesi, Mühendislik Fakültesi,
Makine Mühendisliği Bölümü,
Makine Teorisi ve Dinamiği Anabilim Dalı,
Kınıklı Kampusu, 20070 Denizli, Türkiye

Destekleyen Kuruluş(ların) Adı ve Adresi: -

Projenin Başlangıç ve Bitiş Tarihleri:

01.04.2005 01.04.2008

Öz (en çok 70 kelime)

Gereğinden çok serbestlik dereceli robotlar, sahip oldukları fazla serbestlik derecelerini kullanarak standart robotlar için çok zor olan hareketleri yapabilen robotlardır. Mekanikte ve kontrolde sahip oldukları problemler bu tür robotların uygulama alanına geçmesini engellemektedir. Bu projede gereğinden çok serbestlik dereceli robotlar için yazılım, kontrol ve mekanik dizayn alanlarında çalışmalar yapılmıştır. Serbest ve engellerle dolu bölgelerde verilen hedefe gidebilecek şekilde gerçek zamanda uzuv açılarını ayarlayabilen algoritmalar geliştirilmiş ve dizayn ve imal edilen gerçek robot üzerinde denenmiştir.

Anahtar Kelimeler: Gereğinden çok serbestlik dereceli robotlar, hareket planlaması, potansiyel alanlar

Projeden Yapılan Yayınlar:

E. Şahin Çonkur, Abdullah T. Tola, "Yılan Benzeri robotlar için bir yörünge planlama algoritması" III. Otomasyon Sempozyumu, Pamukkale Üniversitesi, Kasım 2005, Denizli, Sayfa 166-169.