İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

BİR SERVİS ROBOTUNDA TEMEL VERİ İLETİŞİMİ VE KONTROL ALTYAPISININ TASARIMI

YÜKSEK LİSANS TEZİ

Aydın Çağatay SARI

Mekatronik Mühendisliği Anabilim Dalı

Mekatronik Mühendisliği Programı

OCAK 2013

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

BİR SERVİS ROBOTUNDA TEMEL VERİ İLETİŞİMİ VE KONTROL ALTYAPISININ TASARIMI

YÜKSEK LİSANS TEZİ

Aydın Çağatay SARI
518091036

Mekatronik Mühendisliği Anabilim Dalı

Mekatronik Mühendisliği Programı

Tez Danışmanı: Yrd. Doç. Dr. Pınar BOYRAZ

OCAK 2013

İTÜ, Fen Bilimleri Enstitüsü'nün 518091036 numaralı Yüksek Lisans Öğrencisi **Aydın Çağatay Sarı**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"BİR SERVİS ROBOTUNDA TEMEL VERİ İLETİŞİMİ VE KONTROL ALTYAPISININ TASARIMI"** başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Yrd. Doç. Dr. Pınar BOYRAZ** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Prof. Dr. Ata MUĞAN** İstanbul Teknik Üniversitesi	.....
	Doç. Dr. Müştak Erhan YALÇIN İstanbul Teknik Üniversitesi	.....

Teslim Tarihi : **17 Aralık 2012**
Savunma Tarihi : **24 Ocak 2013**

Anneme, babama ve ağabeylerime,

ÖNSÖZ

Tez danışmanım Yrd.Doç.Dr Pınar BOYRAZ'a bana gösterdiği sabır, değerli yardımları ve zorluk çektiğim konularda bana yol gösterdiği için kendisine en içten dileklerimle teşekkür ederim. Bıkmadan ve usanmadan sorduğum her soruya yanıt veren arkadaşlarım Cihat Bora YİĞİT 'e ve Gökçe Burak TAĞLIOĞLU 'na bilgilerini benle paylaştıkları için teşekkür ederim. Ne kadar karamsar bir durumda olsam bile bana inancını ve güvenini yitirmeyen, bu tezi tamamlamda manevi olarak en büyük desteği sağlayan sevgili anneme teşekkür ederim.

Ocak 2013

Aydın Çağatay SARI
Elektrik Mühendisi

İÇİNDEKİLER

Sayfa

KISALTMALAR

COLLADA : Collabrative Desing Activity
DNS : Domain Name Service
IMU : Inertial Measurement Unit
IR : Infra Red
KDL : Kinematics And Dynamics Library
KPIECE : Kinodynamix Planning By Interior-Exterior Cell Exploration
OMPL : Open Motion Planning Library
OpenNI : Open Natural Interaction
OROCOS : Open Robot Control Software
PID : Proportional Integral Derivative
PRM : Probablistic Road Map
RGB : Red,Green,Blue
RGBD : Red,Green,Blue, Depth
ROS : Robot Operating System
RRT : Rapidly Expanding Random Trees
SBL : Single Query Bi-directional PRM with Lazy Collision Checking
TOF : Time of Flight
URDF : Unified Robot Description Format

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

BİR SERVİS ROBOTUNDA TEMEL VERİ İLETİŞİMİ VE KONTROL ALTYAPISININ TASARIMI

ÖZET

İnsansı robotların gerek ev gerek çalışma ortamlarında insanlarla birlikte çalışacağı bir gelecek öngörüsü ile robotik alanında sosyal robotik ismi verilen yeni bir alan doğmuş ve son yıllarda bu alanda yapılan araştırmaların sayısı hızla artmıştır. İnsanlarla yan yana ve işbirliği içinde robotların çalışabilmesi için vazgeçilmez üç ana unsuru birleştirmeleri gerekmektedir: (1) İnsanlar için tasarlanmış bir ortamda gerekli olan hareket esnekliğine sahip olma, (2) kararlı ve uzun süre çalışabilecek bir iletişim altyapısı ve (3) çalıştığı ortamda verilen görevleri yerine getirirken karşılaştığı engellerden sakınma.

Bu çalışmada yukarıda sıralanan üç temel özellik, kararlı bir robot işletim sistemi altyapısı kullanılarak gerçekleştirilmiş. Bu sayede donanım ile yazılımlar arasında kararlı bir bağlantı oluşturulup sistemin modüler olması sağlanmıştır.

Geliştirilen uygulamalar sayesinde insansı robot için servis robotiğinde kullanılmak üzere bazı kontrol ve veri alış-verişi alt-yapısı hazırlanmıştır. Örneğin robot, ofis ve ev ortamlarını tanılayıp içerde bulunan insanları ve hareketli nesneleri takip edebilecektir. Verilen görevleri yerine getirirken engellerden sakınacaktır ve kararlı bir işletim sistemi altyapısıyla daha önceden planlanmış görevleri hata olmadan yerine getirecektir. Robotun öğrenmeyi belli bir başarımla gerçekleştirebilmesi için ilk öncül olan taklit etme yeteneği (mimesis) bulunmaktadır. Bu uygulamalar ile robot, servis amaçlı robotik uygulamalarına da hazır bir alt yapı içermektedir ve az bir güncelleme ve uyarlama ile yaşlı-hasta bakımı vb. uygulamalarda kullanılabilir.

BASIC COMMUNICATION AND CONTROL INFRASTRUCTURE DESIGN OF A SERVICE ROBOT

SUMMARY

A new research area called social robotics has been established since it was understood that in near future humans and robots will have to share the same working environment as in the office scenarios and the robots will be of service at our homes. In order to realize this visionary future, the robots need to be capable of (1) Flexibility to move around in the environments originally designed for humans and the compliance to handle dynamic force changes, (2) stable and long-term erro-free communication infrastructure, (3) mapping and navigating through non-collision roadmaps.

In this work, all the three targets have been deployed on a robot specific operating system called ROS which is supported by several academical and professional institutions all over the world. By using such an operating system, robot's high level software is made independent from hardware infrastucture. Standartization of a robot operating system help developers with using several libraries without software dependcy problems. Robot Operating System is modular so developers can change any algorithm with another causing no harm to software infrastructure.

Using these applications and implementations, the service robot can navigate and perform tasks without causing any damage to humans. By using 3d sensors, robot can detect and follow people's motions and imitate them. While performing pre-defined tasks such as daily house tasks (unsetting table, emptying dishwasher), robot can avoid dynamic or static obstacles. These applications contains three fundemantel operations that a service robot must have in an enviroment which is placed by humans. By applying small modifications, developers can adapt this system to various social applications such as elderly-care or child education.

This work consists of five chapters. First one is the introduction chapter which gives detailed research on robots using robot operating system and its several benefits

which fastens the design process for developers. The research is extending from academical works to industrial applications of robot operating system.

The second chapter focuses on the hardware and software infrastructure and gives detailed description of tools that are used while developing the infrastructure. In this work, robot uses a Microsoft Kinect sensor for 3D perception, MINI-ITX board as the computer, ROS as the operating system. Robot uses several packages in order to fasten the software design of the robot such as transformation packages, robot description packages, motion planning packages, services and nodes which enables robot programs communicate with each others.

Chapter 3 gives detailed theoratical information about algorithms which are used by packages,nodes and services and 3D design of the robot. 3D design of the robot is created by a 3D CAD program. Robot uses trajectory filters for smoothing the planned motion. Motion planning generates paths that are not in collision with enviroment. Inverse kinematics is used for solving the target position and orientation. Kinect helps robot with imitating humans' motion

Trajectory planner is using cubic and quintic polinomial splines which smooth the paths created by the motion planning algorithm. Smoothed trajectories prevent robot exceed its joints' maximum velocities and accelerations. Smoothed trajectories create more human like motions for a robot.

Motion planners are the main structure which creates collision free paths by using Single Query Bi-directional PRM with Lazy Collision Checking algorithm based on probabilistic road maps.

Basically PRM based motion planners, randomly generates points on robot's workspace and eliminates the points which are in collision with obstacles or robot itself. The points that are not in collision are connected together in order to create collision free paths which robot can perform several tasks without damaging the enviroment or itself.

Robot uses newton raphson method which is a numerical method and a recursive algorithm. Given a maximum number of iterations and a maximum error value, the solver tries to converge the tip position and orientation of the robot to target position and orientation within joint limits.

Kinect is used to imitate humans' motion by calculating humans' arm and elbow joint angles. The angles are normalized and applied to robot's arm joints considering the fact that robot's arm joints' limits are not exceeded.

Chapter 4 is the implementation part. In the implementation part, a very powerful simulator called Gazebo is used in order to analyze the robots' joint motions and torques by integrating the 3D desing and the robot operating system. By using such a power simulator, errors that are made during the design can be detected and fixed easily. Modeling the service robot in a powerful simulation enviorement is very important because the real service robot may not be ready to use every time it is needed. But in a simulation enviroment, developers can work on robot any time they want to.

It is very important to check results of the algorithms and fix the bugs in a simulation enviroment before deploying it on the actual robot.

Considering the modularity, a simulation enviroment gives possibilty to developers and researches maintain their contributions regardless of locality, this means robot can be developed by several contributors from all over the world.

In chapter 4, joint torques and dynamics of the robot are analyzed and in conclusion it is observed that the mechanical design of the robot meets the requirements that are set before desing process.

Chapter 5 is the conclusion part which evaluates all the software infstracture used in order to build a modular service robot.

1. GİRİŞ

1.1 Tezin Amacı

Bu çalışmada, iç ortamlarda çalışmak üzere bir insansı robotun veya servis robotunun içinde çalışacak olan programların birbirleriyle anlaşması için gerekli temel veri iletişimi altyapısının ve eyleyicileri kontrol etmek için kontrol altyapısının yazılımsal olarak modüler bir şekilde oluşturulması hedeflenmektedir. Çalışmanın ana hedefi, servis robotlarının çalıştırdığı programların arasındaki mesaj haberleşmesinde oluşucak senkronizasyon problemlerini robotik için özelleşmiş bir işletim sistemiyle çözüp, alt seviye işlerle üst seviye işler arasındaki bağı modülerleştirmeye çalışmaktır. Böylece robotun herhangi bir donanım değişikliğinde üst seviyenin donanıma göre değiştirilmesine engel olunucaktır. İletişim ve kontrol altyapısının bir işletim sistemiyle çözülmesi, robotun gelişim sürecine hız kazandırmış olacaktır. Yazılımsal olan modüler yapıdan dolayı, farklı bir çok kontrol, görüntü işleme, lokalizasyon vb. metodları aynı robot üzerinde test edilebilecektir. Servis robotiği çok karmaşık olduğu için kurulacak bu altyapı farklı disiplinlerdeki geliştiricilerin paralel olarak sistemi geliştirmesine imkan tanıyacaktır. Buna ek olarak geliştiricilerin sadece labrotuardan değil, dünya üzerinde herhangi bir yerden global olarak projeye katkıda bulunmaları sağlanacaktır.

1.2 Problem Tanımı

İnsan ve robotlar arasındaki etkileşim her geçen gün artmaktadır ve teknoloji geliştikçe robotlar insanların yaşam alanlarına daha çok girmeye başlamıştır. Bu etkileşim ne kadar artarsa bir insanın robotla uyumlu bir şekilde çalışması da o kadar önem kazanmaktadır. Bu uyumu sağlamak için insan ve robotun iletişiminin uyumlu bir şekilde gerçekleşmesi kritik önem taşımaktadır. Ayrıca eğer insansı robotlar yakın gelecekte çalışma ortamlarında ve evlerde insanlara yardımcı olmak üzere beraber çalışacaklarsa robotun kararlı bir işletim sistemine, gecikmesiz bir mesaj iletişim altyapısına ve kontrol altyapısına sahip olması gerekmektedir. Servis

robotiği, tüm robotik alanının bir çok alt yapısını içinde bulundurmaktadır. İnsanın doğal olarak yaptığı günlük işleri değerlendirmeye aldığımızda bu işlerin hepsinin servis robotiğinin içinde araştırılan alt konular olduğu görülür. Bu sebebten dolayı servis robotiğinin geliştirilmesinde birden çok araştırmacının çalışması gerekmektedir. Birden çok araştırmacının çalışması demek, onlarca, belki de yüzlerce yazılımın aynı anda bir robot üstünde çalışması demektir. Kullanılan farklı yazılımların ve yazılım kütüphanelerinin ortak bir işletim sistemi altında toplanması ve robot üzerinde uyum içinde çalışmasını sağlamak çok zorlu bir problemdir ve robot üzerinde bir çok algoritma deneneceğinden dolayı bu işletim sisteminin modüler bir yapıya sahip olması gerekmektedir. Bu çalışmada, Yrd. Doç. Dr. Pınar Boyraz denetiminde gerçekleştirilecek olan UMAY rehabilitasyon ve eğitim amaçlı servis robotu projesinin, üst seviye yazılım entegrasyon problemine ve kontrol altyapısı problemine sunulan çözüm, Robot İşletim Sistemi olarak isimlendirilen ve Willow Garage firması tarafından açık yazılım olarak çıkarılan ROS'dur ve bu tez kapsamında ROS'un UMAY'a entegrasyonu tasarlanmıştır.

1.3 Literatür Araştırması

Servis robotlarının kullandığı işletim sistemlerinin hangi ideal özelliklere sahip olması gerektiği çalışma [1]'de tartışılmış ve şu özellikler sunulmuştur: (i) Kurulumunun kolay olması, (ii) geliştirmeye açık olması, (iii) kullanımının kolay olması, adapte edilebilir olması. Çalışma [1] de göz önünde bulundurular diğer bir kıstas ise sınırlı programlama bilgisine sahip olan öğrenciler tarafından bu işletim sistemlerine program yazılmasının istenmesidir. İşletim sistemlerine yüz üzerinden puan verilmiştir. ROS 'a 71 puan verilmiştir ve dördüncü sıraya yerleştirilmiştir. ROS 'un bu sıralamada dördüncü olmasının sebebi ise kullanımının kolay olmaması ve kullanıcının adaptasyon sürecinin uzun olmasıdır, fakat bir servis robotu geliştirilirken geliştiricilerin sınırlı programlama bilgisinden daha üst seviyede olması gerektiği ve birden çok geliştiricinin bu robot üzerinde çalıştığı göz önünde bulundurulmalıdır. Servis robotlarının kompleks yapılar olduğu ve verimli iletişim yapısına ve kontrol altyapısına sahip olması gerektiği [2]'de vurgulanmıştır. Buna ek olarak [2]'de, ROS 'un akademinin ve endüstrinin ortak çalışmalarıyla büyüdüğüne dikkat çekilmiştir, dikkat çekilen endüstri geliştiricilerinin başında Bosch firması ve Intel firması gösterilmiştir. Servis robotiği kompleks bir yapıya sahip olduğundan

eğer standartlaşmış bir işletim sistemi yapısı oturturulmazsa çözülmüş problemlerin sürekli tekrardan yapılması gerekeceği [3]'te belirtilmiştir. Bu standartlaştırma, hızlı robot geliştirme basamağının bir adımı olacaktır ve geliştiricilere donanım ve yazılım uyuşmazlıkları ile vakit kaybettirmeyecektir. Ayrıca böyle bir standartlaşmış yapı, akademideki geliştirici ile profesyonel geliştirici arasındaki bir köprü olacak ve bilim ile birlikte ilerleyen endüstirinin robotik bilimine katkıları hız kazanacaktır. ROS'un standartlaşma sürecinde bir adım olduğu, bir çok robot yarışmaları düzenleyen DARPA'nın simülasyon yarışmalarında ROS'un bir projesi olan ve bu çalışmada da simülatör olarak kullanılan Gazebo'yu seçmesinden ve altı milyon dolar yatırım yapmasından anlaşılmaktadır [4]. Bulut robotiğinde *(cloud robotics)* önemli adımlar atan Google firması, ROS tabanlı robotların, Android tabanlı gömülü sistemlerde uyumlu çalışabilmesi için ROSJava altyapısını geliştirmiştir [5]. MIT, UC Berkeley, Standford gibi bir çok önemli üniversitenin ROS yazılım depoları halka açık olarak bulunmaktadır. NASA ve General Motors firmasının geliştiridiği Robonaut projesinin altyapısı tamamiyle ROS'a adapte edilmeye çalışılmaktadır [6]. ROS'un ve OROCOS'un sağladığı faydaların yüksek performans gerektiren çalışmalarda kullanılabileceği vurgulanmıştır [6]. OROCOS bu çalışmada gerçek zamanlı işletim sistemi özelliği ile değil, sağladığı kinematik ve dinamik kütüphaneler için kulllanılmıştır. ROS'la çalışan farklı tip robot sayısının 90 olduğu fakat 28 tip farklı robotun resmi olarak ROS kullandığı ve 175 farklı tip organizasyonun ROS deposuna sahip olduğu [7]'de belirtilmiştir. ROS'un sağladığı faydaların başında manipülasyon ve lokalizasyon gelmektedir. Bunu daha iyi anlamak için Willow Garage firmasının ticari olarak sattığı iki adet robotu değerlendirebiliriz. Bunlardan birincisi, yüksek derecede gelişmiş olan PR2 ve ucuz bir fiyata satılan Turtlebot'tur. PR2 robotu, insansı işleri, insanların bulunduğu çevrelerde yapmak için geliştirilmiş bir servis robotudur. Freiburg, GATech, MIT, Stanford, TUM, Berkeley gibi üniversitelerin çalışmalarıyla PR2'ya evde yapılan masa temizleme, masa kurma, bulaşık makinasını boşaltma, çamaşır yıkama ve genel olarak insanların ev işlerinde yardım etmesi öğretilmiştir. Bosch firması, akademide çalışan bilimadamları için PR2'ya uzaktan erişim hakkı tanımıştır[8]. Bu örnekler, PR2 robotunun ROS desteği ve donanım desteği sayesinde yapılmıştır. Diğer robot ise daha düşük maliyetli ve açık kaynak lisansına sahip olan Turtlebot'tur. Bileşen olarak iRobot create, Microsoft Kinect ve bir bilgisayar ile donatılmış bir robottur. Düşük maliyeti sayesinde robotlara erişme maliyeti yelpazesini eğitimcilerden hobicilere kadar

genişletilmiştir. Açık kaynak lisansı sayesinde bir çok türevi üretilmiş ve satışa sunulmuştur [9]. ROS desteği sayesinde Turtlebot, kompleks robotik algoritmalarının denenebileceği güçlü bir robot haline gelmiştir. ROS'un kuvvetli üst seviye yazılım altyapısı sadece servis robotlarında degil endüstriyel robotlarda da kullanılması için yapılandırılmaya başlanmıştır. Southwest Araştırma Enstitüsü, ROS-Industrial isminde bir program başlatmıştır. Bu programın amacı ticari endüstriyel otomasyonla akademinin arasındaki boşluğu doldurmak olarak belirtilmiştir. Bu programın öncü faydaları, ticari olarak bulunmayan çarpışmasız hareket planlama ve otonom al yerleştir uygulamaları olarak belirtilmiştir [10]. Şuanda tam olarak ROS'un çalıştırılabildiği endüstriyel robot Motoman firmasının ürettiği SIA10D robotudur. ROS-Industrial hakkında geniş bilgi [11,12]'de bulunabilir. Çalışma [13]'te ROS tabanlı PIONEER-3DX robotu otonom navigasyon, haritalama ve lokalizasyon amaçıyla kullanılmıştır. ROS'un sağladığı navigasyon uygulamaları üç farklı çevrede test edilmiştir ve başarılı sonuçlar alındığı belirtilmiştir. Çalışma [14]'te otonom sualtı robotu sunulmuştur. Sualtı robotunun üst seviye yazılımında ROS kullanılmıştır ve ROS'un kullanılma amacının otonom sualtı robotları arasındaki iletişimi ve birlikte çalışmayı sağlaması olarak gösterilmiştir. Simülasyon, test ve görev gerçekleştirme aşamalarında ROS'un önemi vurgulanmış ve çalışmanın başarılı bir uygulamasının olduğu belirtilmiştir. Çalışma [15] 'te ROS'un navigasyon paketi, RFID ile birleştirilerek nesne lokalizasyonu uygulaması yapılmıştır ve çalışma [15]'in sonucuna göre birleşme, hata payı az, güvenilir ve güç verilimliliği olan bir sistem ortaya çıkarmıştır. UMAY projesinin, gelecekte ROBOCUP@HOME yarışmalarına katılması düşünüldüğü için katılan robotlar arasında da karşılaştırma amaçlı bir araştırma yapılmıştır. CAMBADA robotu Averio Üniversitesi'nin ROBOCUP@HOME yarışması için hazırladığı bir robottur. Takım 2012 yılında tüm robotun üst seviye yazılımını ROS'a geçirmiştir. Takım tanıtım makalesinde, ROS'un robotikte bir standart haline geldiğini ve modüler yapısından dolayı büyük kolaylık sağladığını vurgulanmıştır [16]. Delft Teknoloji Üniversites'nin robotu DRP1, lokalizasyon için ROS'un navigasyon paketini, üç boyutlu görüntüleme yapısı için Microsoft Kinect kullanmaktadır [17]. Donaxi@HOME robotu, Meksika'da bulunan UPAEP Üniversitesi'nin robotu olup, manipülasyon için UMAY'da da kullanılan KDL Kütüphanesini ve navigasyon için ROS'u kullanmaktadır [18]. Almanya'daki Koblenz ve Landau Üniversitesi'nin takımı homer@UniKoblenz ROS'un sürekli büyüyen algoritma deposundan faydalanmak ve modüler bir yapıya

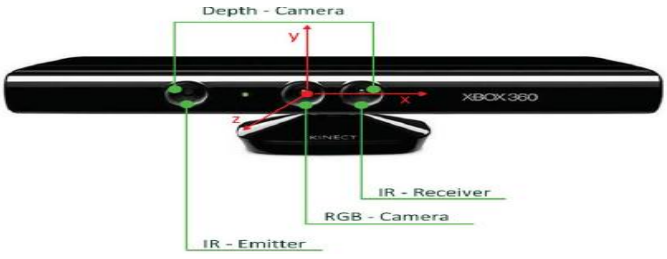
sahip olmak için ROBOCUP@HOME yarışmasında işletim sistemi olarak ROS'u kullanmışlardır [19]. Bonn Üniversitesi'nin takımı NimbRO, iletişim altyapısı problemini ve navigasyon problemini ROS ile çözmüştür [20].

2. YAZILIM VE DONANIM ALTYAPISI

Bu bölümde UMAY'ın robot kolunun gerçek uygulamasında ve yapay görü sisteminin oluşturulmasında kullanılacak olan donanımsal ve yazılımsal araçlar hakkında detaylı bilgiler verilecektir. Donanımsal olarak Kinect'in yapısından ve bilgisayar sisteminin yapısıdan bahsedilecektir. Yazılımsal olarak ise donanımla uyumlu şekilde çalışabilen, robot kolun temel hareket planlamasında, yörünge oluşturulmasında, yörünge takibinde ve yörünge süzgeçlerinde kullanılacak olan ROS, Gazebo, KDL ve OMPL yazılımları ve temel görme ile hareket taklidinde kullanılacak olan OpenNI yazılımı hakkında detaylı bilgi verilecektir.

2.1 Kinect Algılayıcısı

Kinect, Microsoft ve PrimeSense tarafından geliştirilmiştir. Bu donanım aslen Microsoft'un XBOX isimli oyun konsolunu kontrol etmek için kullanılmaktadır. Bu kontrol aşamasında kullanıcının başka herhangi bir araç kullanmasına gerek yoktur [21]. Kinect'in içinde bir adet RGB kamera , IR algılayıcı, çoklu dizi mikrofon sistemi ve algılayıcıyı yukarı aşağı hareket ettirebilecek bir mekanizma bulunmaktadır. UMAY'da Kinect algılayıcısı nesne tanıma, basit insan hareket taklidi ve hareket tanımada kullanılacaktır.



Şekil 2.1 : Kinect algılayıcısı.

Kinect'in özellikleri hakkında resmi bir açıklama olmadığı için kaynak olarak alınan [21] 'de belirtilen özellikler aşağıdaki gibidir.

- 8-bit VGA çözünürlüğü: 640x480 piksel
- 11-bitlik derinlik çözünürlüğü
- Yatay alan görüşü: 57 derece
- Düşey alan görüşü: 43 derece
- Derinlik kamerası: 30 çerçeve/saniye ‘de 640x480 piksel
- RGB kamerası: 30 çerçeve/saniye ‘de 640x480 piksel
- Güç tüketimi: 2,25 Watt

Kinect, gördüğü cisimlerin derinlik haritalarını algılamak için “Light Coding” sistemini kullanmaktadır fakat bu sistem hakkında herhangi bir resmi açıklama bulunmamaktadır [21]. “Light Coding” sisteminin “Time of Flight” (uçuş zamanı) metodunu kullandığı düşünülmektedir fakat PrimeSense firması bunu reddetmektedir. Uçuş zamanı metodu, çevreye IR ışınlarının yollanmasıyla kamera içindeki bir yakalama algılayıcısının ışınların dalgaboyunu ve uçuş zamanını hesaplayarak çevrenin nasıl göründüğünü göstermesidir [22]. Kinect’lerin tipik perspektif hatasının 0.35 ile 0.17 piksel arasında olduğu düşünülmektedir [23]. Kinect, IR algılayıcı kullandığı için güneş ışığının fazla olduğu dış ortamlarda kullanılınca görevini tam anlamıyla yerine getirememektedir. IR algılayıcıların ışınları camlardan da geçebildikleri için fazla cam olan bir iç ortamda Kinect algılayıcı görevini yine tam anlamıyla yerine getiremez, satüre olur. Tek kameralı bir sistemde aynı renkli cisimleri ayırt etmek imkansızken üç boyutlu segmentasyon haritası sunan Kinect algılayıcı ile uzaklık eşik değeri çıkarıp bunu başarmak oldukça kolaydır. Uzaklık eşik değeri çıkararak görüntüdeki istenmeyen kısımlar ve gürültü kolayca çıkarılabilir [24].

2.2 Bilgisayar

UMAY’da Mini-Itx olarak adlandırılan gömülü bilgisayar sistemi kullanılacaktır. Mini-itx anakartlar az yer kapladığı (17 cm x 17 cm veya 15 cm x 15 cm) [25], endüstriyel uygulamalar için uygun olduğundan böyle bir çözüme gidilmiştir. Mini-Itx anakartlar düşük enerji tüketimlerinden dolayı robotik çalışmalarında sıkça kullanırlar. Mini-Itx anakartlar PowerPC, Intel ve AMD gibi mikroişlemci mimarilerini desteklemektedirler. Bu tür bir sistemden 3 Boyutlu görüntü işlemesi

beklenmesi durumunda çok fazla performans verememektir bunun sebebi ise bu tür sistemlerin ekran kartının bulunmamasıdır.Bu sistemler Windows, Linux gibi işletim sistemlerine uyumlu olduğu için tasarım aşamasında performans ve hız açısından aynı fakat daha ucuz bir çözüm sunmaktadır Fiyat olarak ortalama bir dizüstü bilgisayardan daha hesaplıdır bu da tasarım aşamasında ucuz yollu bir çözüm sunmaktadır. Test aşamasında kullanılan sistemin özellikleri aşağıdaki gibidir.

- Intel® ATOM Mini-ITX dual core D525 1.8 GHz işlemci
- 2 adet SODIMM yuvası ile 2 GB/4 GB a kadar DDR3 800 MHz SDRAM desteği
- 18/24 bit tek kanal LVDS çıkışı
- DVI+VGA Ekran Çıkışları
- Çift Gigabit LAN
- 1 adet PCI, 1 adet mini PCIe ve 1 adet Type I/II CF
- 6 adet COM, 8 adet USB 2.0
- 12 Volt DC besleme

Şekil 2.2 : MINI-ITX anakart.

2.3 ROS

ROS, robot yazılımları geliştirmek için tasarlanmış bir yazılım iskeletidir. Dağınık (distributed) bilgisayar kümelerinde çalışabilir ve işletim sistemi benzeri bir işlevsellik sunar. ROS'un ilk tasarım çalışmaları 2007'de Standford AI Robot desteğinde Standford Üniversitesi'nde başlamıştır. Daha sonra, tasarım Willow Garage firması tarafından devam ettirilmiştir. ROS bir yandan işletim sistemi gibi bir işlevsellik sağlarken diğer yandan paket yönetimi, diğer bilgisayar işlemleri arasında mesaj alış-verişi, aşağı seviye aygıt kontrolü gibi işlevsellikler de sağlamaktadır [28]. En büyük özelliği açık kaynak kodlu olmasıdır ve geliştirmeye açık olmasıdır. Linux'un en önemli dağıtımları Ubuntu, Fedora gibi işletim sistemlerinde çalışabilmektedir; Apple'ın işletim sistemi olan MacOSX ve Microsoft'un işletim sistemi olan Windows'a da bu işletim sistemi kurulabilir. Android işletim sistemi kullanan gömülü sistemlere de ROS desteği bulunmaktadır. Robotta kullanılacak herhangi bir algoritma basit birkaç işlemden geçirilerek ROS 'a uygun hale getirilebilir. ROS'da program yazmak gayet kolaydır ve belgelendirilmesi çok geniş kapsamlıdır. Bir çok insansı robot veya servis robotu çalışmalarında ROS kullanılmaktadır, bu robotlara örnekler Bölüm 1'de verilmiştir.ROS 'un bazı uygulama alanları aşağıdaki gibidir:

- Hareket Tanıma
- Nesne tanıma
- Yer saptama ve haritalama
- Hareket takibi
- Stereo görüntü sağlama
- Bulut robotik
- Robot ağları

Temel yazılım dili olarak C++ kullanmaktadır fakat Python ve Java gibi çok kullanılan yazılım dilleriyle de uyumludur. ROS'un bazı ana üstünlükleri aşağıda sıralanmıştır.

- Noktadan noktaya topoloji uygulaması
- Bilgisayar hafızasıdan fazla yer kaplamaması

- İşletim sistemi çekirdeğini yormayıp diğer programların çalışmasına da izin vermesi
- Ücretsiz ve açık yazılım olması

ROS ile ilgili temel kavramlar dosya sistemi ve hesaplama şemaları olarak ikiye ayrılır. Dosya sistemi ile ilgili kavramlar, bilgisayar belleğinizde bulunan ROS'a ait kaynaklarla ilgilidir. Hesaplama şemalarının yapısı, düğümler arasında noktadan noktaya iletişimi sağlayan ağlar oluşturmaktadır [26]. ROS herhangi bir grafik ara yüze gerek duymaz yani ROS'da çalışan tüm program ve tüm veriler komut satırından analiz edilebilir. İstenildiği taktirde tüm mesajlar ve başlıklar ***'rviz'*** isimli bir paket sayesinde göreselleşitirebilir. ROS 'un iletişim ağları ve dosya sistemi sağlamasına ilave olarak, modüler biçimde yazılmış her robota uygulanabilecek bir çok uygulama paketleri mevcuttur. ROS ile ilgili temel kavramlar ve UMAY'da kullanılan uygulamalar alt başlıklar olarak aşağıda detaylandırılmıştır.

2.3.1 Dosya sistemi

ROS'un dosya sisteminin ana birimi paketlerdir. Bir paket temel olarak ROS'un hesaplama diyagramlarını içinde bulunduran bir klasördür. Paketlerin içinde ROS'da çalışması için derlenen düğümler, ROS'a bağlı kütüphaneler, veri kümeleri, yapılandırma dosyaları ve paketin işlevselliği ile ilgili diğer dosyalar tutulur. Paketlerin içinde paketin ne iş yaptığını tanımlayan, hangi kütüphanelere bağımlılığı olduğunu gösteren ve derleyici bayraklarına (*flag*) sahip *manifest.xml* isimli bir beyan (*configuration*) dosyası vardır. Bir çok paketin birleşiminden yığın (*stack*) dosyaları oluşur. Yığın dosyalarının içeriğini belirten de bir beyan dosyası bulunmaktadır. ROS 'un bilgisayarda yayınladığı başlık olarak isimlendirilmiş mesajlaşma sisteminin yapılandırma dosyaları ve iki düğümün talep ve cevap şeklinde anlaşmasında kullanılan servis sisteminin yapılandırma dosyaları bulunmaktadır [6].

2.3.2 Hesaplama şemaları

ROS'da temel hesaplama şema yapılarını, düğümler, servisler, mesajlar, başlıklar, ana sunucu, değişkenler sunucusu ve içinde veri kümelerini bulunduran ***bag*** çuval dosyaları oluşturur [26]. Düğümler ROS için derlenmiş kaynak dosyalarıdır. Genelde bir robot içinde onlarca düğüm dosyası bulunur. Bazı düğümler

algılayıcılardan gelen verileri işlerken, bazı düğümler daha üst seviyede hareket planlaması veya davranış planlaması yapabilir. Servis yapıları, düğümler arasındaki talep ve cevap ilişkisini sunmaktadır. Servisler eğer robotun dağılmış (*distributed*) bir ağ yapısı varsa, örnek olarak bir bilgisayar görüntü işlemek için özelleşmiş ve diğer bir bilgisayar algılayıcı verilerini işlemek üzere özelleşmişse bu iki bilgisayarda çalışan farklı düğümlerin birbirleri arasında talep ve cevap ilişkisine dayalı iletişimini sağlamaktadır. Servislerin sunduğu iletişimin alyapısında ROS'un mesaj sistemi bulunmaktadır. İki düğüm birbirleri arasında mesaj sistemiyle anlaşır. Servislerden farklı olarak bu iletişim, bir düğümün bir başlık yayınlamasıyla diğer bir düğümün o başlığa kaydolması şeklindedir[26]. Mesaj yapıları, basit C programlama dili veri yapılarından (*integer*, *float* gibi) ve bu veri yapılarının C benzeri dizilerinden oluşmaktadır. Lazer algılayıcısına özel olarak hazırlanmış mesaj tipini [27]'de görebilirsiniz. Kullanıcı kendi mesaj tipini oluşturabilir veya ROS tarafından hazır olan standartlaşmış mesaj tiplerini kullanabilir. ROS'da standart haline gelmiş bir çok mesaj tipi vardır örnek olarak algılıyıcı mesaj tiplerinden IMU ve lazer algılıyıcısının mesaj tipleri gösterilebilir. ROS'da bir mesaj, düğüm tarafından yayınlanan bir başlık ile ağa taşınır. Başlıklar belirli bir frekansta yayınlanır, eğer bir düğümün belirli bir mesaj tipinde verilere ihtiyacı varsa uygun başlığa kaydolur. Her başlığın düğümde belirlenen ayırt edici bir ismi vardır. Bir başlığa birden fazla düğüm bağlanabilmektedir. Bir mesaj tipi, farklı bir başlık ismi altında birden çok yayınlanabilir. Bir düğüm bir başlıktan istediği zaman kaydını siler ve isterse tekrar o başlığa kaydolabilir. İçinde mesajlaşmayı, başlığı ve servis yapısını sunan temel bir şema Şekil 2.3 'de verilmiştir.

Şekil 2.3 : ROS'un temel kavram şeması.

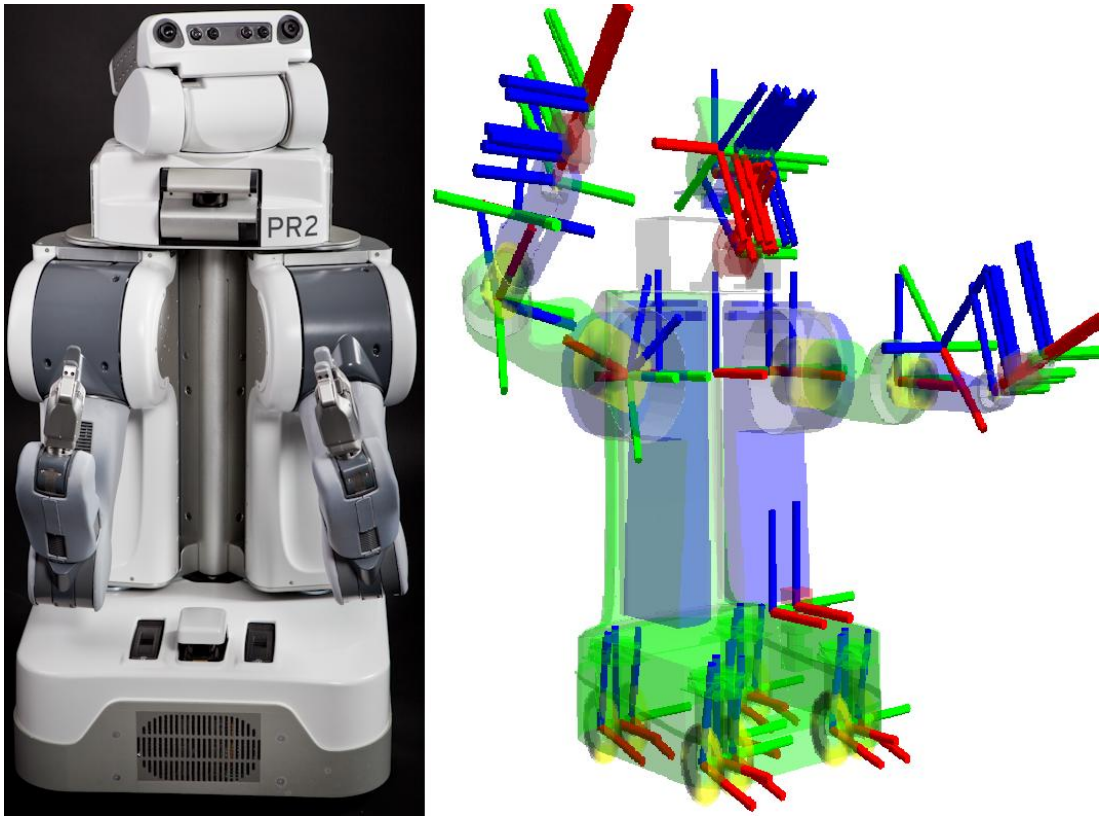
Değişken sunucusu, değişkenlerin ayarlanmasında kullanılmaktadır. Değişken sunucusu yardımıyla bir düğümün değişkenleri değiştirilebilir. Bu değişkenler bir dosyada veya direk olarak ROS'un ana sunucusunda tutulabilir. Değişken sunucusu, bir düğümün çalışmaya başlamadan önce kendisini nasıl yapılandırması gerektiğini

gösterir. Örnek olarak bir düğüm konum kontrolü yapmak için PID kontrol uygulamak istediğini düşünelim, bu durumda bu kontrollör için gerekli olan PID katsayıları direk bir dosya içinden *Değişken sunucusu'*na atılabilir veya bir başlat dosyası içinden bunu düğüm kendisi okuyabilir. Çuval dosyaları içinde bir çok veri kümesi tutulur, bu veri kümeleri mesajlar yayan başlıklardan oluşur. Bir çuval dosyası içinde birden çok başlık kaydedilebilir. Robot gerçek bir uygulamadayken yayınlanan başlıklar çuval dosyaları içine kaydedilebilir ve daha sonra tekrar oynatılabilir. Bu sistem, algoritma geliştirirken sürekli aynı senaryoları robot üzerinde uygulamaktan geliştiriciyi kurtarır ; buna ek olarak, robot üzerinden alınmış ve bir çuval dosyasına kaydedilmiş olan veri kümeleri başka bir bilgisayar üzerinde işlenebilir. Ana sunucu ROS'da isim servisi görevini yapmaktadır. Ana sunucu, başlık, servis ve düğüm isim kayıtlarını tutumaktadır. Ana sunucu olmasaydı hiç bir düğüm diğer düğümle iletişime geçemez veya bir servis çağrılamazdı[26]. Düğümler, kayıt bilgilerinin durumlarını raporlamak için ana sunucuyla haberleşirler ve diğer kayıtlı düğümler hakkında bilgi alırlar. Düğümler birbirlerine direk olarak bağlanırlar, ana sunucu burda sadece arama bilgileri sağlamaktadır. Ana sunucunun görevi DNS sunucusuna benzemektedir. Ana sunucu, düğümleri yönlendirdikten sonra başlığa kaydolmak isteyen düğüm başlığı yayınlayan düğüme bağlanma isteği yollar. Bu istek TCP/IP protokolünde yapılmaktadır. ROS'da en çok kullanılan iletişim protokolü TCP/IP'dir.

2.3.3 Dönüşüm paketi

Bir robotta birden çok kordinat sistemi bulunmaktadır. Bu kordinat sistemleri arasındaki dönüşümler robotikte önemli bir yer tutmaktadır. Eğer bir kordinat sisteminin diğerine göre nerede bulunduğunun izi tutulmazsa, hareket planlamada, engellerden sakınımda ve görevleri yerine getirmede birçok sıkıntı çıkabilir. ROS'da bu problemi çözen ***'tf'*** [32] isminde bir paket bulunmaktadır. Bu paket, çok modüler olup herhangi bir robota kolayca uygulanabilir. Her bir robot uzuvuna ve/veya algılıyıcısına, referans kordinat sistemine göre nerde olduğunu belirtilen konum, yönelim bilgisi ve bir eksen takımı atanır. Robot hareket ettiğinde bu algıyıcının ve/veya uzuvun nerede ana kordinat sistemine göre hangi zamanda nerede bulunduğu kolayca takip edilebilir. Bu paket belki de ROS'un sunduğu en güçlü pakettir. ROS kullanan nerdeyse her robot bu paketten yararlanır. Bu veriler ROS'un ana sunucusunda tutulduğu için geçmişte bir kordinat sisteminin ana kordinat sistemine

göre nerde olduğunun da takibi yapılabilmektedir. ROS dağınık bir ağ yapısında da çalışabildiği için eğer ana kordinat sistemimizi bir oda veya bir fabrikann hangarı olarak seçersek bir çok robot binada veya odada birbirlerine göre nerede olduğunun takibi yapılabilir. Atanan eksenler ya programla bir başlık altında ya da eksen sabitse statik olarak ROS'da yayınlanabilir. Yayınlama ve bu yayınlanan başlıklara kaydolma ile ilgili geniş bilgi [32,33]'de detaylıca işlenmiştir. Şekil 2.4'de Willow Garage firmasının ürettiği PR2 isimli robotun atanmış kordinat sistemlerinin ***tf***'de nasıl gözüktüğü gösterilmiştir.



Şekil 2.4 : PR2'nin kordinat sistemleri.

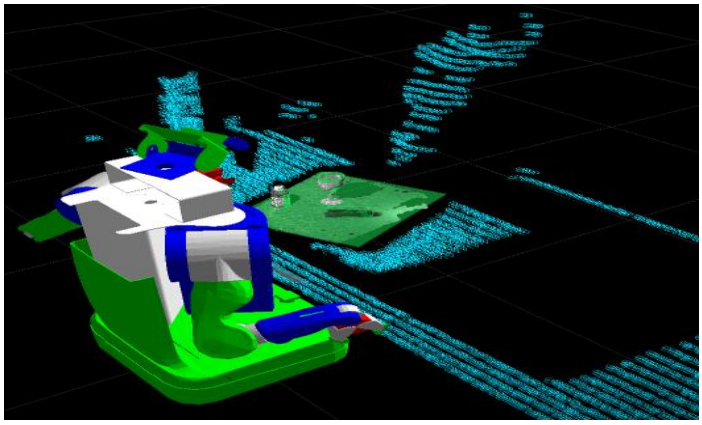
2.3.4 Görüntüleme paketi

ROS'da yayınlanan tüm başlıklar ve bu başlıkları oluşturan paketler ***rviz*** isimli 3 boyutlu görüntüleme paketiyle görselleştirilir. ROS'un içinde standartlaşmış tüm paket görüntülenebilir fakat geliştircinin yarattığı özel bir mesaj yapısı varsa bunun ***rviz*** 'de görüntülenebilmesi için geliştiricinin ***rviz*** için bir yazılım eklentisi programlaması gerekmektedir. ROS 'un görüntüleyebildiği bir kaç mesaj tiplerine bir kaç örnek aşağıdakiler gibidir

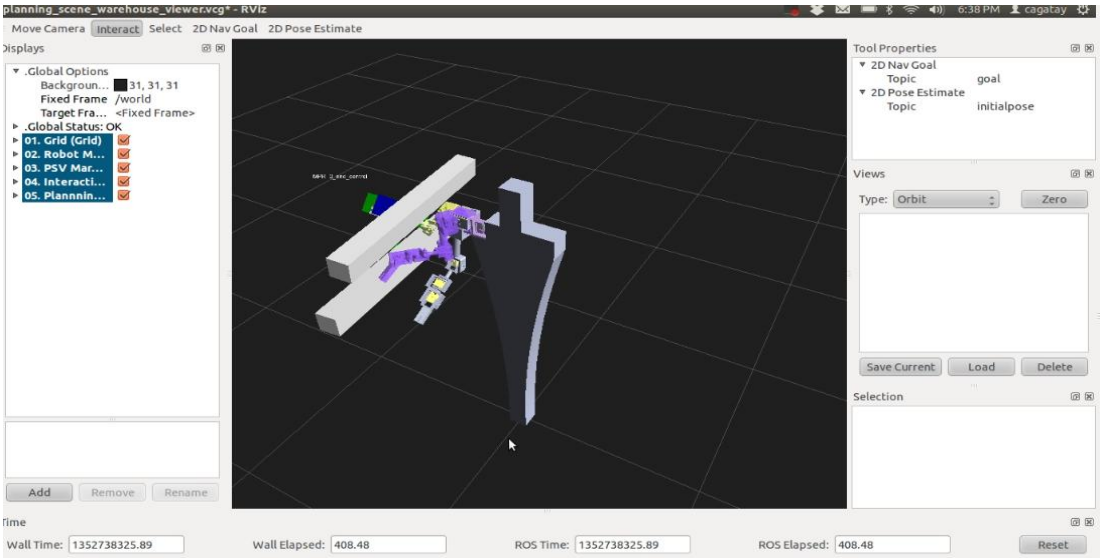
- Algılayıcı mesajları

- 2 boyutlu ve 3 boyutlu Harita mesajları
- Nokta bulutları
- Konum, yönelim mesajları
- Kordinat sistemleri mesajları
- Kamera mesajları

ROS dağınık bir yapıya sahip olduğu için bir robotta yayımlanan mesajlar başka bir operatörün kullandığı bilgisayarda ***rviz***'de görüntülenebilir. 3 boyutlu görüntüleme paketi olan ***rviz***, ROS'un çalışabildiği her işletim sistemi platformunda çalışabilir, ***rviz'***le ilgili detaylı bilgi [34] 'de verilmiştir. Şekil 2.5'de ve Şekil 2.6 'da ***rviz***'e ait farklı mesajların görüntülenmesi ve UMAY'ın ***rviz***'deki görüntüsü verilmiştir.



Şekil 2.5 : PR2 için Nokta bulutu görüntüleme.



Şekil 2.6 : UMAY'ın rviz'deki görüntüsü.

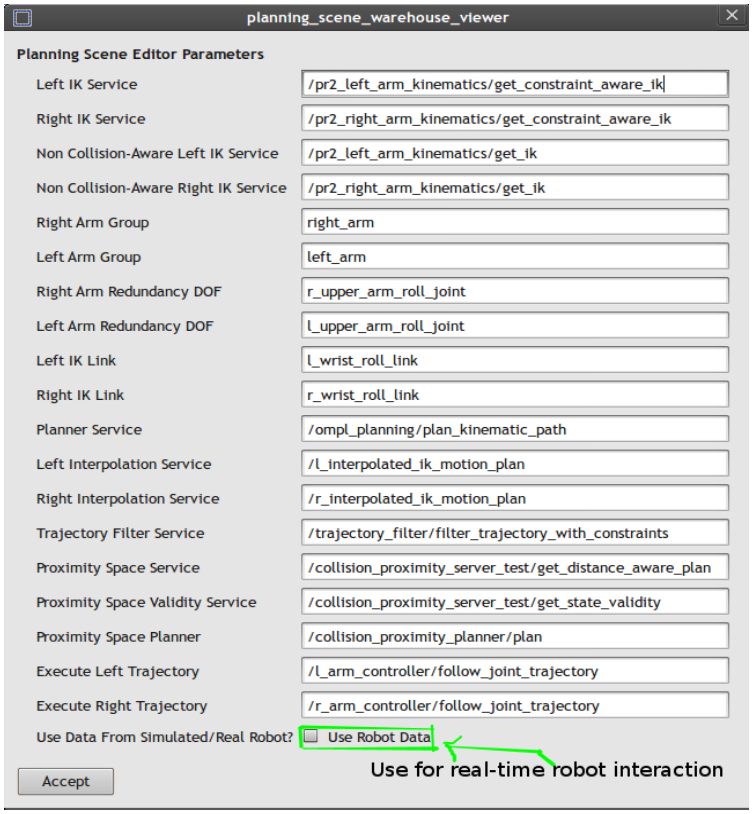
2.3.5 URDF paketi

URDF (universal robot description file) paketi, XML dosya yapısında olan URDF *robot tanımı dosya yapısı* için geliştirilmiş bir ayrıştırıcıdır. Robot tanımı dosyası ağaç yapısına sahiptir. Ağaç yapısından örnek alınarak isimlendirilen sistemde bir bağın bağlandığı sadece bir ana bağ olabilir ve böylece bir zincir oluşturulur. Parallel yapıya sahip robotları desteklemez. Robot tanımı dosya yapısında robotu oluşturan tüm altyapıların tanımları bulunmaktadır. Bu tanımlar içinde uzuvların boyutları, uzuvların görüntü dosyaları ve konum yönelimleri, eklemler, eklemlerin dönme ve ötelenme sınırları, eklemlerin dönme eksenleri, algılayıcıların konum ve yönelimleri, eyleyicilerin konum ve yönelimleri bulunmaktadır. Bunlara ek olarak kütle, atalet gibi fiziksel özellikleri, çarpışma özellikleri de robotu oluşturan bağlara ve eklemlere atanabilir. Algılıyıcılar, uzuvlar eğer durağanlarsa robot tanıtım dosyasında bağ olarak, eyleyiciler ise eklem olarak tanıtılırlar. Robot tanım dosyasının uzantısı ***.urdf*** **'**dir ve robot tanım dosyası değişken sunucusuna yüklenebilir. Değişken sunucusuna yüklenen robot tanımı düğümler, hareket planlayıcılar, dönüşüm paketi, 3 boyutlu görüntüleyici ***rviz*** ve Gazebo tarafından ulaşılır hale gelir. ROS kullanan nerdeyse her robotun bir robot tanım dosyası bulunmaktadır. Bu dosyalar zorunlu olmasa bile tasarım aşamasında çok işe yaramaktadır. URDF ile ilgili detaylı bilgi [35,36] 'de verilmektedir.

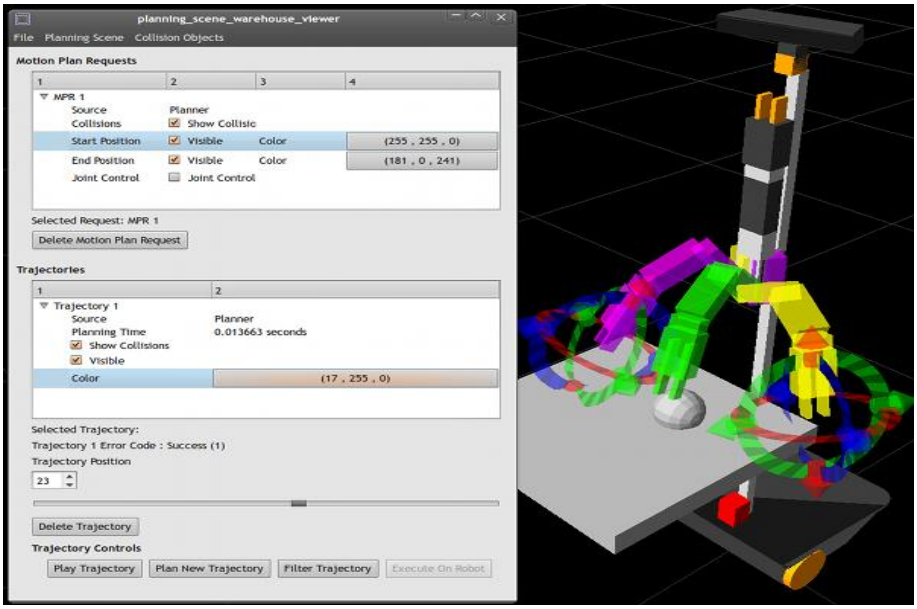
2.3.6 Hareket planlayıcı paketler

Robot kol için hareket planlayıcı kısmı iki ayrı pakete ayrılmaktadır. Bunlardan biri uzuvların ve eklemlerin birbirlerine çarpıp çarpmadığına kontrol eden paket *Planning Description Configuration Wizard* ve diğeri ise ilkinin sonuçlarını göz önünde bulundurup hareket planlamasını ve yörünge planlamasını yapan ve bunu gerçek robot üzerinde veya Gazebo simulasyonu üzerinde çalıştırabilen *Warehouse Viewer* 'dır. İlk paket, verilen robot tanım dosyasına dayanarak bir kinematik zincir oluşturmakta, bu zincirin eklemlerinin sınırlarını da göz önünde bulundurarak farklı birçok eklem konumları denemektedir. Eğer denenen eklem konularında, robotik kol kendi uzuvlarıyla çarpışma içindeyse o yapılandırma hareket planlamasının dışında kalır. Tüm tekrarlamalı denemeler bittikten sonra robotun çalışma uzayı çıkarılır ve bir ROS paketi olarak kaydedilir. İlk pakete ilişkin detaylı bilgi [37]'de verilmiştir. Diğer paket ***Warehouse Viewer*** olarak isimlendirilmiştir. Bu pakette tüm hareket

planlama, ters kinematik, yörünge paketleri denenebilir ve ***rviz*** ‘de görüntülenebilir. Şekil 2.7’de ***Warehouse Viewer***‘ın yapılandırma pencereleri gözükmektedir, Şekil 2.8’de ise ***Warehouse Viewer*** ve ***rviz***‘in etkileşimi gözükmektedir.



Şekil 2.7 : Hareket planlıyıcının yapılandırma penceresi.



Şekil 2.8 : Hareket planlayıcının 3 boyutlu görüntüleyici ile etkileşimi.

Bu paketin kullanımıyla ilgili daha detaylı bilgi [38]‘de bulunmaktadır.

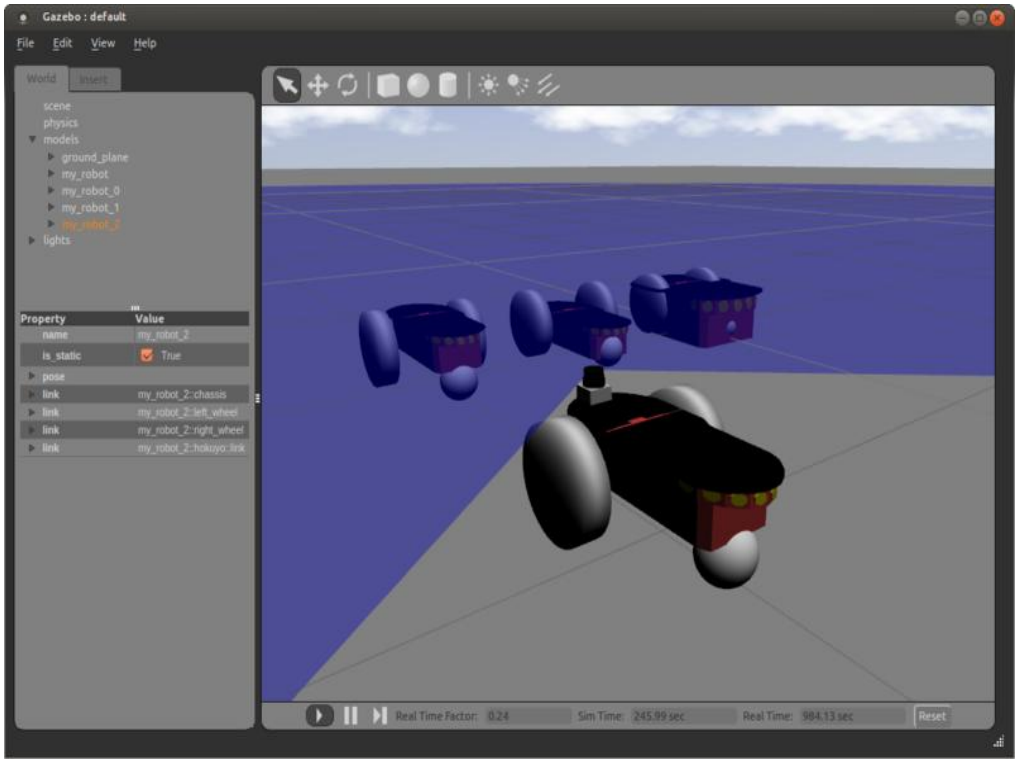
2.4 Gazebo

Gazebo, 2002 yılında Southern California Üniversitesi’nde geliştirilmeye başlanmış bir açık yazılım projesidir. Şu an projenin geliştirilmesini finansal olarak Willow Garage firması üstlenmiştir. Gazebo, dış mekanlarda ve iç mekanlarda çalışan çoklu robotlar için geliştirilmiş bir üç boyutlu simülasyon programıdır. Bir çok nesneyi, algılayıcıyı ve nesne ile robot arasındaki etkileşimi simüle etme yeteneğine sahiptir [39]. Gazebo’nun içinde katı-gövde dinamiğini simüle eden bir fizik motoru da bulunmaktadır. Gazebo tek başına kullanıbilinir veya ROS ile birlikte kullanılabilir. Gazebonun sunduğu bazı araçlar aşağıdaki gibidir.

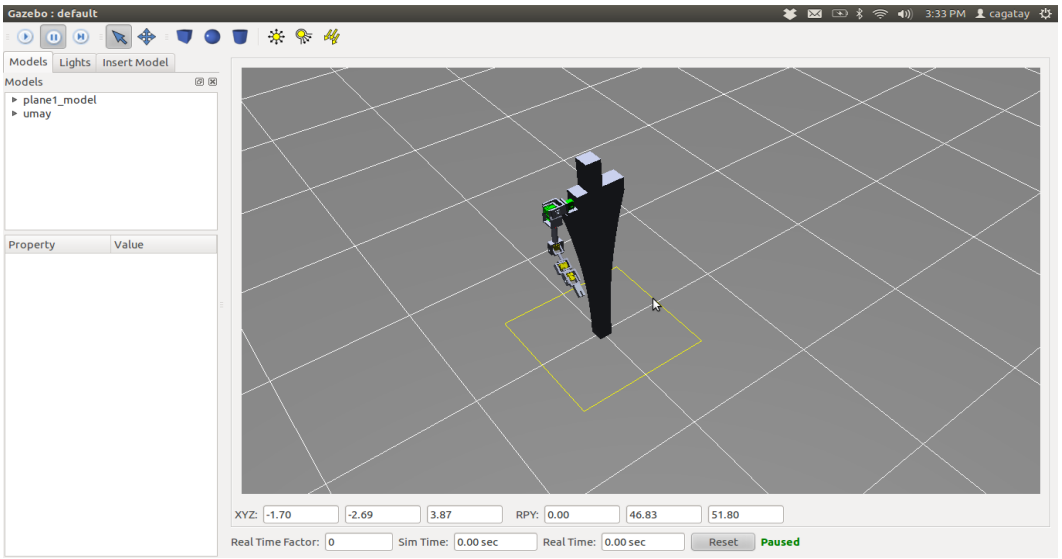
- IMU algılayıcısı
- Kuvvet-moment algılayıcısı
- Lazer algılayıcısı
- Sonar algılayıcısı
- Kinect algılayıcısı
- Kamera algılayıcısı
- DC motor
- Gerçek zamanlı kontrolör altyapısı

Yukardaki algılayıcılar ve eyleyiciler Gazebo ile birlikte standart olarak gelmektedir fakat Gazebo bize istediğimiz tarzda bir eyleyici, algılayıcı ve kontrolör tasarlama imkanı sunmaktadır. Tasarlanan yapıların Gazebo’yla birlikte çalışabilmesi için Gazebo’nun yorumlayabileceği bir yazılım eklentisi haline getirilmelidir. Yazılım eklentisi ile ilgili rehber bilgiler [39]‘da detaylı olarak verilmiştir. Gazebo ile ROS‘u birlikte kullanabilmek için robotun, Bölüm 2.3.5‘de açıklanan bir tanım dosyası olması gerekmektedir. Robot tanım dosyasındaki algılayıcı bağları, eklem eyleyicileri sadece bir bağ ve eklemdir. Bu algılayıcı bağlarının ve eklem eyleyicilerinin simülasyonda çalışması için o bağların ve eklemlerin Gazebo yazılım eklentileriyle birleştirilmesi gerekmektedir. Kısacası robotun ROS ve Gazebo’yla birlikte çalışması için Gazebo için yapılandırma dosyalarına ve robot tanım

dosyalarına ihtiyacı vardır. Ek olarak, uzuvlara ve eklemlere etki edecek sürtünme kuvvetleri ve sönümleme oranları da Gazebo'nun yapılandırma dosyalarına eklenir. Gazebo bu yapılandırma dosyalarının içeriğine bakarak ROS'a adapte edilmiş gerçeğe yakın bir simülasyon ortamı yaratmaktadır. Gazebo'da çalışan bir robotun örneği Şekil 2.9'da verilmiştir. UMAY'ın, Gazebo ortamındaki görüntüsü ise Şekil 2.10'da verilmiştir.



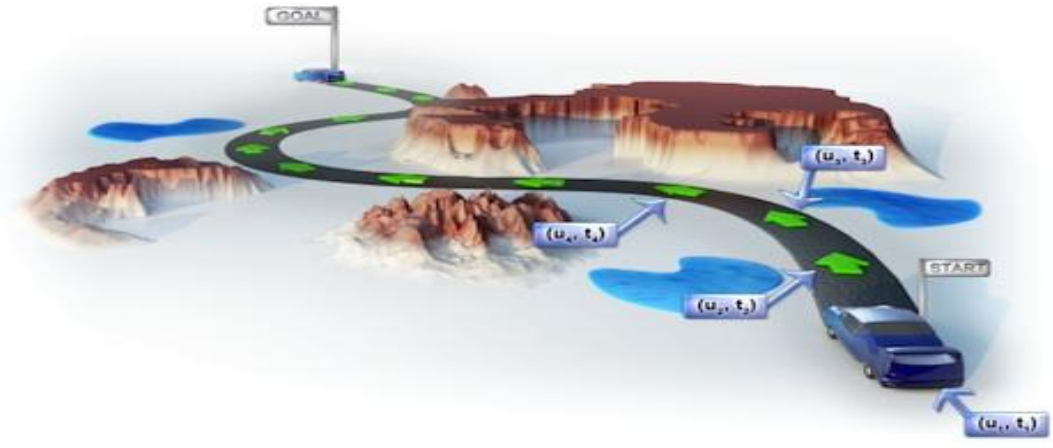
Şekil 2.9 : Gazebo'da çalışan bir robot.



Şekil 2.10 : UMAY'ın Gazebo'daki görüntüsü.

2.5 OMPL Kütüphanesi

OMPL, örnekleme tabanlı ve açık yazılım olan üç boyutlu hareket planlama kütüphanesidir. OMPL'nin içinde bir çok hareket planlama algoritmalarının uygulamaları bulunmaktadır. OMPL Kütüphanesi, araştırma, eğitim ve endüstriyel uygulamalarda kullanılması için geliştirilmiştir. Tasarım olarak geliştiricinin asgari veri girişi yaparak, karmaşık hareket planlama algoritmalarını robotta uygulamasına imkan tanımaktadır. OMPL için algoritmaların çalışma verimlerini kıyaslayabileceğiniz bir performans kıyaslama yazılımı bulunmaktadır. OMPL, ROS'la birlikte çalışabilmektedir. Servis robotlarının çalıştıkları ortamda verimli olarak yol bulmaları çok önem taşımaktadır. Hareket planlamadaki temel problem, eklemlerdeki ve uzuvlardaki kısıtlamaları göz önünde bulundurarak robotun bulunduğu konumdan hedef konuma götürebilecek devamlı bir yol bulmaktır. Örnekleme tabanlı algoritmaların geneli olasılıksaldır. OMPL'nin kullandığı örnekleme tabanlı algoritmalardan bazıları, PRM, RRT, KPIECE'dır. UMAY'da kullanılan hareket planla algoritması *Single Query Bi-directional PRM with Lazy Collision Checking* (SBL)'dir ve Bölüm 3'de detaylı olarak bu algoritmadan bahsedilmektedir. OMPL, ROS'da bulunan robot kol hareket planlama yığınlarının arka ucunu oluşturmaktadır. Arka uç olarak kastedilen kavram, kullanıcının birebir olarak paketi kullanmadan yardımcı yazılımlarla hareket planlamayı gerçekleştirmesidir [40]. Bölüm 2.3.6'da anlatılan hareket planlayıcı paketlerinin çekirdeğinde OMPL Kütüphanesi bulunmaktadır. OMPL Kütüphanesi'nin grafik arayüzü bulunmaktadır ve örnek bir yol bulma uygulaması Şekil 2.11'de gösterilmiştir.



Şekil 2.11 : OMPL Kütüphanesi'nin grafik arayüzü.

2.6 KDL Kütüphanesi

KDL Kütüphanesi, OROCOS Projesi'nin tamamını oluşturan 4 yazılım kütüphanesinden biridir. OROCOS Projesi, bir açık yazılım projesidir ve amacı robotlar için gerçek zamanlı olarak kullanılabilecek işletim sistemlerinden bağımsız bir uygulama iskeleti yapısı sunmaktır. KDL Kütüphanesi, OROCOS Projesi'nin gerçek zamanlı olarak kinematik zincirleri hesaplama imkanı sunan yazılım altyapısıdır [28]. KDL Kütüphanesi paralel, humanoid, seri, mobil kinematik zincir yapılarıyla birlikte kullanılabilmektedir. Kütüphane, C++ yazılım dili tabanlı bir kütüphanedir ve geliştiriciye farklı C++ nesne grupları sağlamaktadır. Bunlar aşağıdaki gibi sıralanabilir.

- Çizgi, nokta, çerçeve gibi geometrik nesneler
- Dönüşüm matrisleri
- Kinematik zincirler
- Kinematik ağaçlar
- Kinematik çözücüler

Kinematik zincirler, geliştirciye eklemler tarafından bağlanmış seri gövde zincirleri tanımlamasını sağlamaktadır. Kinematik ağaçlar, tanımlı gövdelerden bir ağaç yapısı oluşturma imkanı sunmaktadır. Kinematik çözücüler, ters kinematik ve ileri kinematik problemlerini çözmemizi sağlamaktadır. KDL'nin diğer bir avantajı ise geliştiriciye bir kinematik zincirin Jakobiyen'ini ve bu Jakobiyen'in tersini ve sözde tersini (pseudo-inverse) hesaplama imkanı sunmaktadır. KDL Kütüphanesi, ROS 'un kurulumunda standart olarak gelmektedir. Bölüm 2.3.5'te bahseliden robot tanım dosya sistemi kullanılarak KDL'ye uygun kinematik ağaç ve zincir yapıları çıkarılabilir. UMAY'da KDL'nin kinematik çözücüleri ve kinematik zincir yapısı kullanılmıştır. KDL, uygulama olarak geliştiriciye çok büyük rahatlık sunmaktadır ; sadece 10-15 satır kod yazılarak bir seri zincirin kinematik çözümlemesi yapılabilir. KDL Kütüphanesi hakkında detaylı bilgi [28,41]'de verilmiştir.

2.7 OpenNI Kütüphanesi

OpenNI, bir çok yazılım dilini destekleyen ve farklı işletim sistemlerinde çalışabilen bir kütüphanedir, Kinect algılayıcısıyla uyumlu olarak çalışabilmektedir ve

PrimeSense firması tarafından geliştirilmiştir. Burdaki NI doğal etkileşimi yani insan ve makina arasındaki etkileşimi simgelemektedir [2]. OpenNI iki tip API sunmaktadır bunlardan birincisi algılıyıcılarla iletişim kurabildiğimiz aygıt sürücüleri diğeri ise ara yazılım için kullanılabilecek API'ler. Ara yazılımdan kast edilen işlevsellikler aşağıdaki gibidir

- Tam vücut analizi
- El noktası analizi
- Hareket tanıma

Aygıt sürücüleri yazılımcının 3 boyutlu algılıyıcıyla RGB kameraya, IR algılayıcıya ve ses aygıtına erişimini sağlar. Yazılımcı istediği taktirde ara yazılım API lerini kullanmadan direkt saf işlenmemiş veriye erişebilir. OpenNI ücretsiz bir yazılımdır fakat ara yazılımı olan NITE açık kaynak kodlu değildir. Farklı platformlarda çalışabildiği için bir işletim sistemi için yazılmış kaynak kodlar diğer bir işletim sisteminde de kullanılabilir. ROS için paket haline getirilmiştir. Genelde Kinect ve ROS kullanan sistemler bu kütüphaneyi kullanılır. Bu kütüphane ile ilgili ayrıntılı bilgi [31]'den edinilebilir.

3. TASARIM VE KONTROL ALT YAPISI

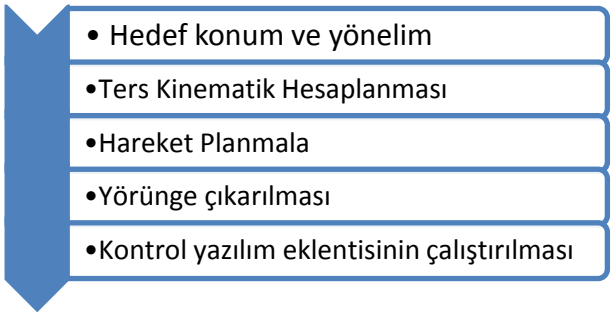
UMAY'ın gerçek hayatta başarılı olarak tüm işlevlerini gerçekleştirmesi için yapılan yazılım altyapısı ile ilgili çalışmalar bu bölümde detaylı olarak incelenmiştir. Bölüm, dört ana başlık altında toplanmıştır. İlk bölüm UMAY'ın üç boyutlu görsel tasarımını anlatmaktadır. Kalan bölümler ise UMAY'a hareket yeteneklerini kazandıran yazılımsal ve matematiksel bölümleri kapsamaktadır.

UMAY'ın eklem açılarının kontrolünü simüle edebilmek için ROS 'un içinde bulunan mekanizma kontrol altyapısı kullanılmıştır. Mekanizma kontrol altyapısı gerçek zamanlı işletim sistemlerinde kullanılmak üzere tasarlanmış[42]. Altyapı aslen Willow Garage firmasının ürettiği PR2 isimli robotta kullanmak için yapılmış olsa bile tüm robotlarda basitçe uygulanabilecek şekilde ayarlanabilmektedir. UMAY'da uygulanmak üzere bu altyapı bir eklenti şekline getirilmiştir. Yazılım eklentisi 3 aşamadan oluşmaktadır:

- İlk kullanıma hazırlanma aşaması
- Başlangıç
- Güncelleme
- Durma

İlk kullanıma hazırlanmada, UMAY'ın robot tanımı dosyasından kinematik zinciri alınır ve KDL'ye içerik olarak aktarılır. Başlangıç kısmında eklemlerin ilk konumları, sistem zamanı ve kontrolcü ile eşzamanlandırılır. Günceleme bölümünde istenilen yörüngelerin takibi için eklem açılarının PID kontrolü ile sağlanmaktadır. Bu yazılım eklentisinin çalışabilmesi için gerekli olan koşul, takip edilmek istenilen yörügenin UMAY'ın çalışma uzayında tanımlı olmasıdır. UMAY'ın kolunun uç noktasının bir hedefe gitmesi için KDL'nin kütüphaneleri kullanılarak ters kinematik hesaplama programı yazılmıştır ve bu program bir ROS servisi haline getirilmiştir. Eğer bu servis başarı geri-beslemesi yollarsa hareket planlaması ve yörünge

çıkarılmasına başlanmaktadır. Hareket planlama OMPL ile yapılmaktadır [43]. Hareket planlama yapıldıktan sonra yörünge çıkarılır ve kübik polinom ile bu yörünge yumuşaklaştırılır. Yumuşaklaştırmanın amacı robotun eyleyicilerinin korunması ve düzgün, ani ivmeleri olmayan bir hareket kazandırmadır. Kol sistemi için akış diyagramı Şekil 3.1'de verilmiştir.



Şekil 3.1 : Kol kontrolü için akış diyagramı.

Mekanizma kontrolü altyapısı için daha detaylı bilgi [42,44,45]'de bulunmaktadır.

3.1 Üç Boyutlu Mekanik Tasarım

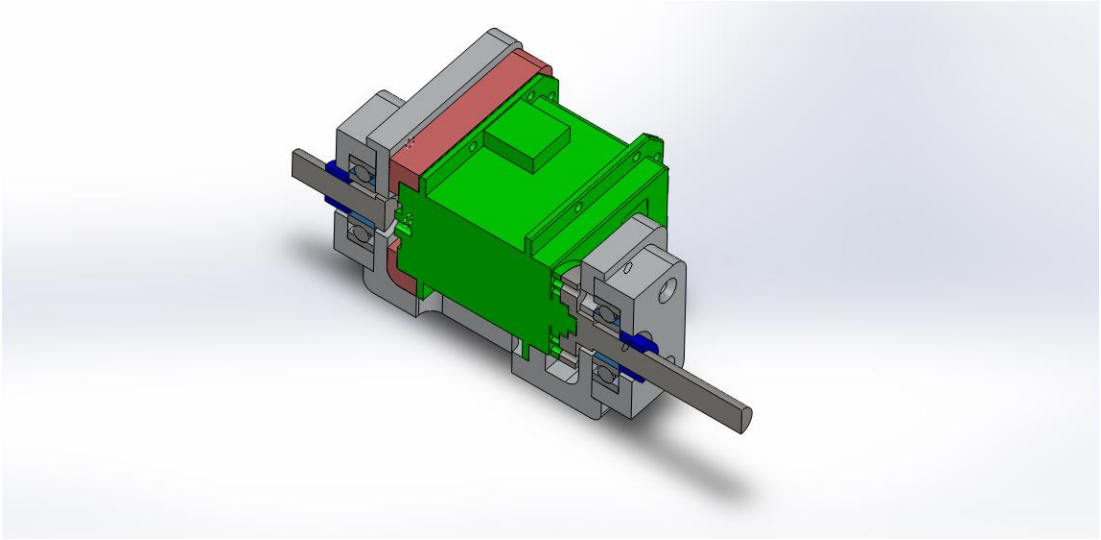
UMAY'ın kolu 6 serbestlik derecesine sahiptir. UMAY'a 6 serbestlik derecesine kazandıran toplam 6 adet servo motor kullanılmaktadır. Eyleyici olarak servo motorların seçilmesinin sebebi, servo motorların konum referanslarına göre hareket kazanmalarıdır ve bu konum referanslarında sabit kalmalarıdır. Eyleyicilerin kol üzerindeki dağılımları aşağıdaki gibidir,

- 3 adet omuzda
- 1 adet dirsekte
- 2 adet bilekte

Servo motorların üreticisi Dynamixel firmasıdır. Dynamixel firmasının motorlarının seçilmesinin sebebi, servo motorların çoğunda bulunan konum referansı aşımı problemi en az düzeyde bulunmasıdır ve konum, akım geri-beslemesinin alınmasıdır. UMAY'ın kolunda Dynamixel servo motorlarından 2 model kullanılmıştır. Omuzlarda bulunan motorların modeli EX 106, dirsek ve bileklerde bulunan motorların modeli RX 28'dir. EX 106 modeli 14.8 Volt ile 18.5 Volt gerilimlerinde beslenmekte, çıkış olarak ise 107 ile 87 kg-cm [46] torka sahiptir. RX 28 modeli, 12

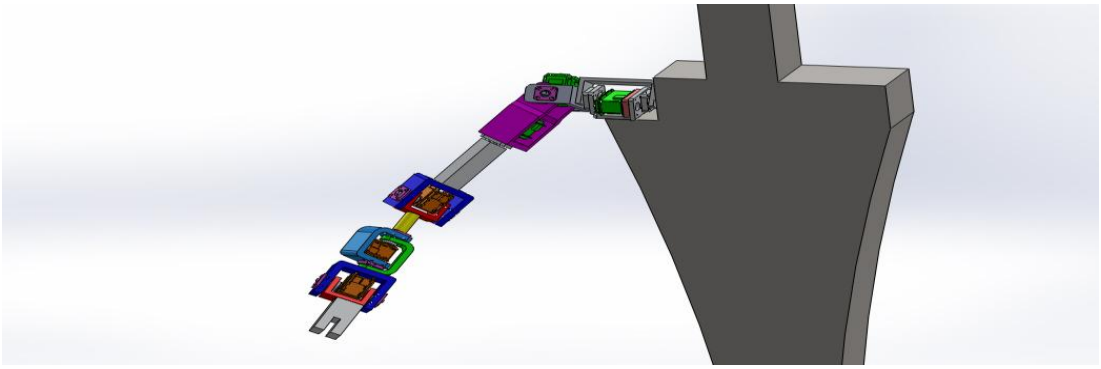
Volt – 16 Volt gerilimleri arasında çalışmakta, 28.3 kKg-cm – 37.7 Kg-cm çıkış vermektedir [47]. UMAY'ın kaldırması istenilen azami yükünün 500 gr olmasından dolayı, bu motorların tork çıkışları tasarım için uygun olarak öngörülmüştür. Azami yükün 500 gr olarak hesaplanması, UMAY'ın çocukların eğitiminde ve insan etkileşiminde kullanılacağından dolayı çok ağır yükler kaldırmasına gerek olmamasından kaynaklanmaktadır. UMAY'ın kaldıracağı yüklere örnek olarak bir su bardağı veya çeşitli oyuncaklar verilebilir.

UMAY'ın kolunun katı modellenmesi ve mukavemet hesaplamaları SolidWorks programında yapılmıştır. Birlikte hareket eden malzemeler, SolidWorks'te alt montaj şeması olarak birleştirilmektedir bu Gazebo simülasyon programında hem basitleştirme hem de işlem hızı kazandırmaktadır. Şekil 3.2'de alt montaj olarak düzenlenmiş bir kol malzemesinin şekli bulunmaktadır.

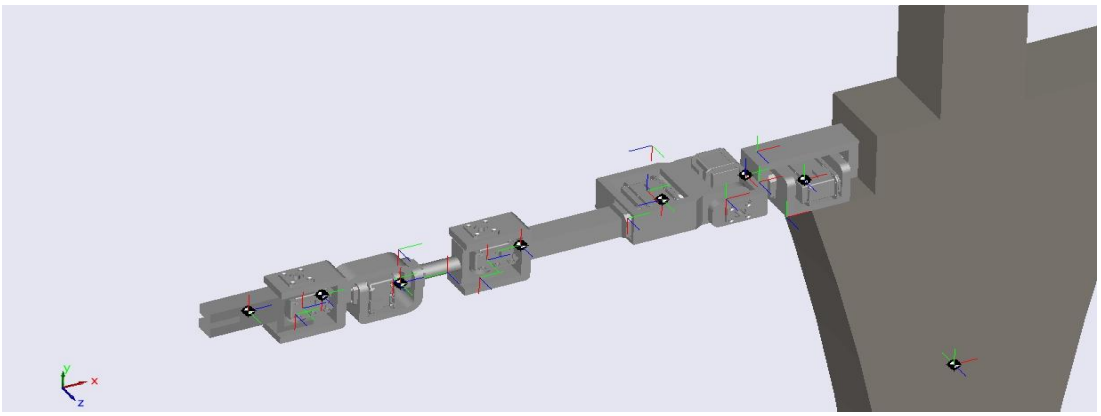


Şekil 3.2 : Alt montaj şeması.

Şekil 3.3'de ve Şekil 3.4'de UMAY'ın katı model kol yapısı gösterilmiştir.



Şekil 3.3 : Kol yapsı 1.



Şekil 3.4 : Kol yapısı 2.

Katı modelleme programından çıkış olarak alınan dosya SimMechanics'e geçirilmekte ve SimMechanics'ten alınan çıkış, ROS'da bulunan bir paket yardımıyla [48] URDF formatına çevirilmiştir. Bu aşamadan sonra UMAY'ın kinematik zinciri oluşturabilir. Şekil 3.4'de URDF dosya sisteminden çıktı olarak alınan kinematik zincir bulunmaktadır.

Şekil 3.5 : Kinematik zincir

Şekil 3.5'de sırayla isimlendirilmiş robot kolun uzuvların çerçevelerini temsil etmektedir. *World* ile isimlendirilen çerçeve, Gazebo'nun referans çerçevesidir. *Body* olarak isimlendirilen çerçeve, Şekil 3.3'de ve Şekil 3.4'de görülen temsili insan figürünün *World*'e bağlandığı sabit eklemin çerçevesidir. *Bir, Iki, Uc, Dort, Bes, Alti* ve *Yedi* olarak isimlendirilen çerçeveler robot kolun eklemlerini temsil etmektedir. Her bir uzuv birlerilerine bir adet eklemle bağlanmaktadır ve eklemler sadece dönme hareketi yapabilmektedir; öteleme hareketi yapabilen bir eklem UMAY'ın kol tasarımında kullanılmamıştır. Eklemlerin dönme sınıları, simülasyonda -180 derece ile +180 derece olarak atanmıştır. Eklemler, eyleyecilerin sınırlarından dolayı gerçek uygulamada bu sınılara ulaşamamaktadır, fakat, simülasyon sınırlarının bu aralıkta olmaları, farklı düzenlenme biçimlerinin denenmesinde yardımcı olur.

Kolun ilk simülasyon testleri Matlab'in bir eklentisi olan SimMechanics programında yapılmıştır. Bu programda eklemlere binen torklarla Gazebo'da alınan

ileri dinamik tork değerlerinin eşleştiği görülmüştür, bu veriler Bölüm 4'te detaylı olarak verilmiştir.

Tüm kolu oluşturan uzuvları, omuz ile dirsek arası yaklaşık 290 mm ve diresek ile bilek arası yaklaşık 220 mm olmak üzere toplam 510 mm boyutlarındandır. Uzuvların malzemeleri, bağlantı elemanlarının ve yataklarının malzemeleri alüminyum olarak seçilmiştir. Alüminyum malzemenin seçilmesinin sebebi, kol eklemlerindeki motorların kaldırabileceği bir hafif kol tasarımına olanak tanımasıdır. UMAY'ın bir kolunun toplam ağırlığı yaklaşık olarak 2.5 kg'dır.

Kolun bütün tasarımında ön planda tutulan tasarım ilkesi, kolun bütünüyle modüler bir yapıya dayanmaktadır. Bu modüler yapının getireceği kolaylık, kolun herhangi bir parçası esnediğinde veya kırıldığında kırılan bölgeyi çıkararak yerine yeni parçayı koyup, kolun hızlı bir şekilde tekrar işlevlerini yerine getirmesini sağlamasıdır. Kolun ilk tasarımında uç uzuva eklenecek bir tutucu el tasarlanmamıştır. Tüm kol sistemi gerçek uygulamada tutarlı bir biçimde işlevlerini yerine getirdikten sonra tutucu tasarımına başlanacaktır.

3.2 Kinematik Ve Dinamik Tasarım

Robot kol kinematiği, robotun eklemlerinin hareketi ile tüm robotu oluşturan katı gövdenin hareketini tanımalamaktadır. Çoğu robotik kolun eklemleri hidrolik, elektrik motoru ve pnömatik eyleyiciler tarafından sürülmektedir. Robot kol dinamiği, eklemlere uygulanan torklar veya kuvvetler etkisiyle robot kolun nasıl bir hareket izleyeceğini tanımlamaktadır [49].

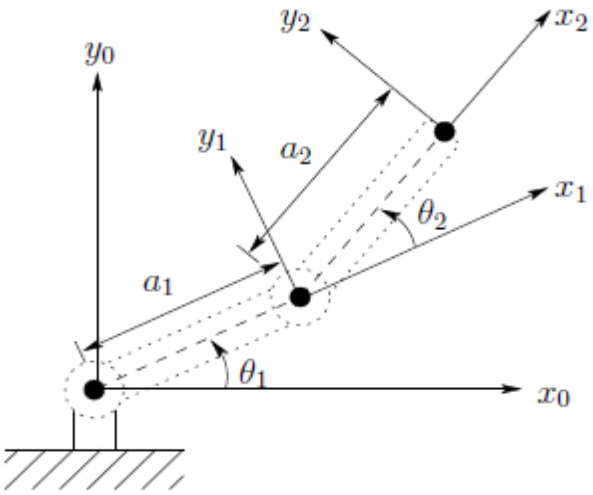
3.2.1 İleri kinematik

Matematiksel olarak ileri kinematik denklemleri, kartezyen kordinatlardaki konum ve yönelim ile eklem konum uzayını açıklamaktadır [50]. Seri kinematik bir zincirin ileri kinematiği en genel haliyle **(3.1)**'de gösterildiği gibi yazılır.

$$T_n^0 = \prod_{i=1}^{n} T_i^{i-1}(\theta_i) \qquad \textbf{(3.1)}$$

Burada θ_i eklem açı parametresi, T_i^{i-1} ise i. kol eklemi ile i-1. kol eklemi arasındaki dönüşüm matrisidir. Robotikte dönüşüm matrisi, robotun eklemlerine sanal olarak

eklenmiş olan kordinat eksenleri arasındaki dönüşümleri ifade etmektedir. T_j^i dönüşüm matrisi, DH (Denavit-Hartenberg) metoduna uygun olarak yazılabilir. DH metodu bu çalışmanın dışında kaldığından dolayı bu metod hakkında detaylı bilgi [50]'de bulunabilir.



Şekil 3.6 : İki serbestlik derecesine sahip örnek bir robot kol.

Örnek olarak Şekil 3.6'da iki serbestlik derecesine sahip kolun ileri kinematik denklemleri **(3.2)**'de ve **(3.6)**'de verilmiştir. A_1 ve A_2 matrisleri, homojen transformasyon matrisi olmak üzere,

$$A_1 = \begin{bmatrix} c_1 & -s_1 & 0 & a_1c_1 \\ s_1 & c_1 & 0 & a_1s_1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(3.2)}$$

$$A_2 = \begin{bmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2s_2 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(3.3)}$$

Bu yüzden, T_1^0 ve T_2^0 matrisleri:

$$T_1^0 = A_1 \quad \textbf{(3.4)}$$

$$T_2^0 = A_1A_2 = \begin{bmatrix} c_{12} - s_{12} & 0 & a_1c_1 + a_{12}c_{12} \\ s_{12} & c_{12} & 0 & a_1s_1 + a_{12}s_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(3.5)}$$

Uç noktanın konumu x_o, y_o ‘ya göre **(3.6)**’daki gibi olacaktır.

$$x_o = a_1 c_1 + a_{12} c_{12}$$
$$y_o = a_1 s_1 + a_{12} s_{12} \quad \textbf{(3.6)}$$

Burada c_{12} ve s_{12} birinci eklem açısı $\theta_1$ve ikinci eklem açısı θ_2’nin toplamının sinus ve kosinüs değelerini simgelemektedir. UMAY 6 serbestlik derecesine sahip olduğu için burada tüm ileri kinematik denklemlerinin yazılması hem zordur hemde çok karmaşık olacağı için gereksizdir. Denklem **(3.5)**, bize referans noktayla n’inci eklemiyle arasındaki konum dönüşümünü vermektedir; bu konum dönüşümünün Jakobiyen’i bize eklemler ve uç nokta arasındaki hız ilişkisini gösterir. Jakobiyen matrisi ters kinematik bölümünde çıkarılacaktır.

UMAY’da ileri kinematik problemini hızlı bir şekilde KDL ve URDF yazılımsal araçlarını kullanarak çözülmüştür. KDL ile URDF arasındaki bağlantı ROS uygulama iskeleti tarafından kurulmuştur. UMAY’ın sahip olduğu tüm eklemler ve bu eklemlerin mekanik sınırları (eklemin dönemebileceği azami açı değeri vb.) URDF dosya sisteminde tutulmaktadır Bölüm 2 ‘de URDF dosya sistemi hakkında daha detaylı bilgi verilmiştir. Bu dosya sistemi ROS ‘un parametre sunucusuna atıldıktan sonra KDL tarafından seri kinematik bir zincir haline getirilir. Bir ROS servisi çağırarakUMAY’ın uç noktasının konumu ve yönelimi geri-besleme olarak ROS sunucusundan alınabilir. KDL, ileri kinematik problemini tekrarlamalı(recursive) olarak çözmektedir. KDL’nin ileri kinematik problemini çözmekte kullandığı kütüphane [51]’de daha detaylı olarak incelenebilir. İleri kinematik hesaplama programı, UMAY’da ters kinematik yazılımını doğrulamak ve kullanıcının eklem açılarını elle vererek, uç konumu ve yönelimini gözlemlemesi için yazılmıştır.

3.2.2 Ters kinematik

Kartezyen konumlar, yönelimler ve eklem uzayının konumları arasındaki ilişki ileri kinematik denklemlerinin jakobiyeniyle hesaplanabilir.

Jakobiyen, robot hareket kontrolü ve analizinde, hareket planlamada, yumuşak yörünge takibinde, tekil konumların tespitinde, robot kol hareketinin dinamik denklemlerinin çıkarılmasında, kuvvetlerin ve torkların uç noktaydan eklemlere

dönüşümünde kullanılan önemli bir matristir. Bir n uzuvlu robot kol için, Jakobiyen n vektörlü eklem hızlarıyla içinde hem doğrusal hızları hem de açısal hızları tutan 6 vektörlü uç nokta hızları arasındaki anlık dönüşümü sağlamaktadır [50]. Bu yüzden Jakobiyen 6 x n boyutunda bir matristir.

Uç noktadan referans noktaya kadar olan tüm uzuvların açısal hızları referans noktaya göre ω_n^0 şekilde ifade edilebilir. Denklem 3.7'de ω_n^0 'nin hesaplanması gösterilmiştir.

$$\omega_n^0 = \sum_{i-1}^{n} p_i \ \dot{q}_\imath z_{i-1}^0 \tag{3.7}$$

(3.7)'de gösterilen $\dot{q}_\imath$, i. eklemin konumunu türevini göstermektedir. Diğer bir değişken olan p_i eğer i. eklem dönme hareketi yapiyorsa 1'e, öteleme hareketi yapiyorsa 0'a eşittir. Değişken z_{i-1}^0 dönme eksenini temsil etmedir ve k birim vektör olmak üzere Denklem 3.8'deki gibi ifade edilir.

$$z_{i-1}^0 = R_{i-1}^0 k \tag{3.8}$$

Robot kolun uç noktasının hızı $\dot{o}_n^0$, türev almakta kullanılan zincilerleme kuralıyla Denklem **(3.9)**'da hesaplanır.

$$\dot{o}_n^0 = \sum_{i=1}^{n} \frac{\partial o_n^o}{\partial q_i} \ \dot{q}_\imath \tag{3.9}$$

Bir robot kolunJakobiyeni hem dönme hem de öteleme hızlarından oluşmaktyasa Jakobiyen Denklem **(3.12)**'deki gibi ifade edilebilir

$$J_v = [J_{v_1} \cdots \ J_{v_n}] \tag{3.10}$$

$$J_w = [J_{w_1} \cdots \ J_{w_n}] \tag{3.11}$$

Böylece,

$$J = \begin{bmatrix} J_v \\ J_w \end{bmatrix} \tag{3.12}$$

Uç noktanın referans noktaya göre hızları ve bunların Jakobiyen ile birleşmesi **(3.14)**'de verilmiştir

$$\xi = \begin{bmatrix} v_n^0 \\ \omega_n^0 \end{bmatrix} \tag{3.13}$$

$$\xi = J\dot{q} \tag{3.14}$$

Jakobiyen ilişkisi sağlandıktan sonra, bir robot kolun ters kinematik denklemi eğer Jakobiyen kare bir matris ve tekil noktalara sahip değilse, **(3.15)**'deki gibi ifade edilebilir.

$$\dot{q} = J^{-1}\xi \tag{3.15}$$

UMAY 6 serbestlik derecesine sahip olduğu için, Jakobiyen'in tersi, tekil olmayan noktalarda bir matris tersi alınır gibi alınabilir, fakat genelde her robot kolun 6 serbestlik derecesi yoktur; bu tür robot kolların Jakobiyen'nin tersi alınamamaktadır. Bu sorunu çözmek için Gauss eleme metodu gibi bir çok algoritma ve yöntemler bulunmaktadır [51]. Diğer bir durumda ise robot kol 6 serbestlik derecesinden fazla serbestlik derecesine sahiptir. Bu tür durumlarda **(3.15)** için çözüm Jakobiyen'in sağ sözde tersini (right pseudo inverse) alarak sağlanabilir. Serbestlik derecesinin 6'dan yüksek olduğu durumlar için $n > 6$, sağ sözde ters denklemi Denklem 3.16'da gösterilmiştir.

$$J^{+} = J^{T}(JJ^{T})^{-1} \tag{3.16}$$

(3.14)'ün çözümü için **(3.16)** ve **(3.14)** ilişkilendirilirse,

$$\dot{q} = J^{+}\xi + (I - J^{+}J)b \tag{3.17}$$

Şeklinde ifade edilir. **(3.17)**'deki değişken b, $b \in \mathbb{R}^n$ ile ifade edilen keyfi bir vektördür. $(I - J^{+}J)b$ ifadesi 0'a eşit olmadığından dolayı, eğer bir eklem hızı $\dot{q}'$ bu ifadeye eşitse, bu eklem hızı Jakobiyenin sıfır uzayındadır. Böyle bir durumda uç noktanın konumu ve yönelimi sabit kalırken, eklemler $\dot{q}'$ hızında hareket edecektir çünkü $J\dot{q}'$ sıfıra eşittir [51]. KDL, ters kinematik ile uç noktanın konum ve yönelim hesabını yapmak için Newton-Raphson olarak isimlendiren tekrarlama metodunu

kullanmaktadır. Bu algoritma KDL'de gömülü olarak bulunmaktadır. Newton-Raphson Metodu bir fonksiyonun köklerini bulmak için kullanılan nümerik bir metoddur; bu metod doğrusal olmayan denklemlerin çözümünde sıkça kullanılmaktadır. Bu hesabı, ilk bir tahmin noktasının etrafında Jakobiyen doğrusallaştırmasını hesaplayarak yapar ve bu doğrusallaştırmayı en yakın sıfıra gitmek için kullanır [52]. Doğrusal olmayan denklemler tekrarlamalı olarak Newton-Raphson Metoduyla çözüldüğünde $(k+1)$'nci tekrarlama **(3.18)**'deki gibi ifade edilir [53].

$$q^{k+1} = q^k - J^{-1}(q^k)\delta_k \tag{3.18}$$

Burada $J^{-1}(q^k)$, q^k hesaplanan robot kolun Jakobiyen'in tersidir. Değişken δ_k ise uç noktanın referans noktaya göre diferansiyel konumu ve yönelimini göstermektedir. Referans noktaya göre hedeflenen uç noktanın konum ve yönelimi T_n^* ise, Newton Raphson metodu, **(3.19)** gerçeklendiğinde ya da azami tekrarlama sayısına ulaşıldığında son bulur [52].

$$T_n(q^{k+1}) - T_n^* = 0 \tag{3.19}$$

olduğunda Newton-Raphson metoduyla ilgili dikkat edilmesi gereken bir kaç hussus vardır, bunlar aşağıdaki gibi sıralanabilir

- İlk tahmini konumun seçilmesi
- Yakınsama sınırı
- Tekilliğin olup olmadığı
- Yakınsama hızı
- Yerel minimumdan kaçınma

UMAY için KDL kullanılarak yazılan konum ve yönelim hesaplama programı, ters kinematik ve Newton-Raphson metoduyla tekrarlamalı kartezyen kordinatlarda uç noktanın istenilen konuma ve yönelime gitmesini sağlamaktadır. Tekrarlama sayısı 100, yakınsama sınır $1e-6$ olarak tanımlanmıştır. Bu kontrol yapılırken eklemlerin azami sınırlarına ve asgari sınırlarına dikkat edilmektedir.

3.2.3 Dinamik

Kinematik denklemler, kuvvet ve torkları göz önünde bulundurmadan robotun hareketlerini tanımlamaktadır [50]. Dinamik denklemler ise, robot kolun kinematik ve atalet tensörü özelliklerine dayanan, doğrusal olmayan ikinici dereceden diferansiyel denklemlerle ifade edilmektedirler [49]. Dinamik denklemler, simülasyon oluşturmakta, kontrol algoritmaları geliştirmekte ve robotun hareketlerinin eyleyicilerle birlikte analiz edilmesinde fayda sağlamaktadır. Robot kolun dinamik denklemlerinin çıkarılmasında sıkça kullanılan iki adet yöntem bulunmaktadır. Bunlardan ilki Lagrange-Euler denklemleri diğeri ise Newton-Euler denklemleridir. Bu bölümde, Newton-Euler yöntemine kıyasla daha kolay matris işlemleri [55] olan Langrange-Euler denklerimden bahsedilecektir. UMAY'da, eklemlere yansıyan kuvvet ve tork değerleri Gazebo Simülasyon Ortamından elde edilmiştir. KDL Kütüphanesi'nin ters dinamik problemini çözen bir sınıf yapısı olsa da UMAY'ın dinamik analizi açışından Gazebo'nun kullanılması daha uygun görülmüştür.

Bir sistemin Langrange fonksiyonu, sistemin toplam kinetik enerjisi ile toplam potansiyel enerjisi arasındaki farktır [55]. K sistemin kinetik enerjisi, P ise sistemin potansiyel enerjisi olmak üzere, Sistemin Lagrange fonksiyonu **(3.20)**'deki gibi ifade edilir.

$$L(q, \dot{q}) = K(q, \dot{q}) - P(q) \tag{3.20}$$

Yukaridaki denklemde q eklemlerin konumlarını temsil etmektedir. Denklem **(3.20)**'de gözüktüğü üzere, K fonksiyonu eklem konum ve hızına, P fonksiyonu ise sadece eklem konumlarına bağlıdır. Sistemin kinetik enerji fonksiyonunu açarsak Denklem **(3.21)**'de verildiği gibi gözükür.

$$K(q, \dot{q}) = \frac{1}{2}\sum_{i=1}^{n} v_i^T \, m_i v_i + \omega_i^T I_i \omega_i \tag{3.21}$$

Denklem **(3.21)**'de olduğu gibi toplam kinetik enerji, herbir eklemin toplam kinetik enerjisine eşittir, n toplam eklem sayısını temsil etmektedir . Burada v_i ve ω_i sırasıyla, i. bağın kütle merkezinin ana kordinat sistemine göre doğrusal ve açısal hızlarını temsil etmektedir. Diğer değişkenler olan m_i , i. bağın kütlesini ve I_i ise i.

bağın kütle merkezinin ana kordinat sistemine göre atalet tensörünü temsil etmektedir [14]. ${}_{i}^{0}R$, i. bağa ait kordinat sisteminin ana kodinat sistemine göre yönelimini gösteren bir dönüşüm matrisi olmak üzere, I_i, **(3.22)**'deki gibi yazılır.

$$I_i = {}_{i}^{0}R I_m {}_{i}^{0}R^T \tag{3.22}$$

(3.22)'deki I_m değişkeni, katı bir nesne olan uzuvun kendi kütle merkezine göre olan atalet tensörünü simgelemektedir. Jakobiyen matrisi olan J_i, i. bağa ait Jakobiyeni matrisi olmak üzere

$$J_i = \begin{bmatrix} J_{v_i} \\ J_{\omega_i} \end{bmatrix} \tag{3.23}$$

$$v_i = J_{v_i}\,\dot{q} \tag{3.24}$$

$$\omega_i = J_{\omega_i}\dot{q} \tag{3.25}$$

Değişkenler v_i'nin ve ω_i'nin açılmış halini **(3.21)**'de yerine yazarsak, kinetik enerji fonksiyonu **(3.26)**'daki hali alıcaktır.

$$K(q,\dot{q}) = \frac{1}{2}\dot{q}^T \sum_{i=1}^{n} \left[{J_{v_i}}^T m_i \; J_{v_i} + \; {J_{\omega_i}}^T I_i \, J_{\omega_i} \right] \dot{q} \tag{3.26}$$

Potansiyel enerji, yer çekimi ivmesinin bulunduğu bir ortamda bağ kütle merkezlerinin yer değiştirmesiyle oluşan iş miktarıdır [55]. Sistemin toplam potansiyel enerjisi, **(3.27)**'de verildiği gibi ifade edilmektedir.

$$P(q) = \sum_{i=1}^{n} m_i \, g^T h_i \tag{3.27}$$

Yukaridaki denklemdeki değişken h_i , i. bağın kütle merkezinin ana kordinat sistemine göre konumu, g ise yer çekimi ivmesinin vektörünü temsil etmektedir. Lagrange fonksiyonu çıkarıldıktan sonra, sistemde robotun hareketinden dolayı eyleyicilerde oluşan torkları ve kuvvetleri veren ifade **(3.28)**'de verilmektedir.

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = \tau \quad \textbf{(3.28)}$$

(3.28)'de L yerine, K ve P fonksiyonları koyulursa denklemin yeni hali aşağıdaki gibi olacaktır.

$$\frac{d}{dt}\frac{\partial K}{\partial \dot{q}} - \frac{\partial K}{\partial q} + \frac{\partial P}{\partial q} = \tau \quad \textbf{(3.29)}$$

Böylece Denklem **(3.29)**'un çıkarılmasıyla eklemlere yansıyan torklar ve kuvvetler bulunmuş olur.

3.3 Hareket Ve Yörünge Planlama

UMAY'ın kolunun gerçek hayatta verim olarak çalışabilmesi için çalışma uzayında başlangıç konumundan ve yöneliminden karşısına çıkıcak olan engellerden sakınarak hedef bir konuma ve yönelime gitmesi için hareket planlama yapılması gerekmektedir. Hareket planlamasının sonucunda ortaya çıkan engelsiz yol düğümlerinde verilen yörüngelere göre eklemlerin hareket edilmesi için ise yörünge planlama yapılması gerekmektedir. UMAY'da kullanılan hareket planlama ve yörünge planlama metodlarının teorisi bu bölümde detaylı olarak incelenecektir.

3.3.1 Hareket planlama

Hareket planlama algoritmları, robotun çalışma uzayında bulunan engellerden sakınarak hedef konuma ve yönelime gitmesini sağlamaktadır. Algoritmalar bunu yaparken sadece robotun geometrik tanımını göz önünde bulundururlar, dinamik denklemleri işin içine katmamaktadırlar. Eklem değişkenlerinin mümkün her konumdaki yapılandırmalarının toplamına yapılandırma uzayı denilmektedir [50]. Çarpışma, robot çalışma uzayında bir engelle etkileşime geçtiğinde olur. Çalışma uzayı, robotun hareket ettiği kartezyen bir uzaydır. Robotun yapılandırma uzayı Q, eklem değişkenleri q , robot tarafından kaplanan çalışma uzayanın alt kümesini $A(q)$ olarak ifade edilirse, yapılandırma uzayında bulunan engellerin hepsini Denklem **(3.30)**'daki gibi yazabiliriz [50].

$$QO = \{\, q \in Q \mid A(q) \cap O \neq 0 \,\} \quad \textbf{(3.30)}$$

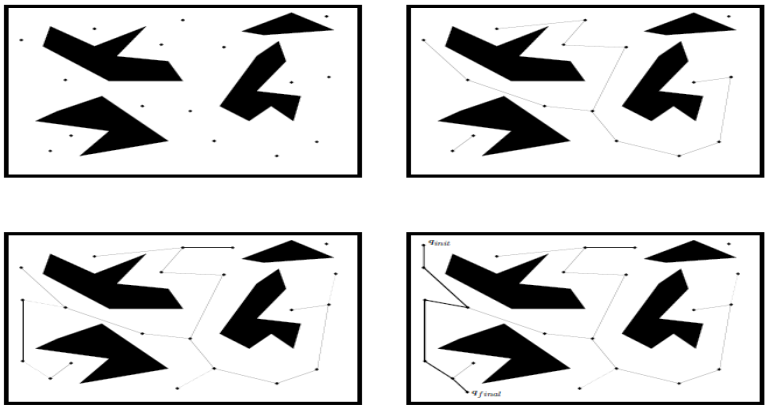
Yukarıdaki denklemde O, yapılandırma uzayindaki tüm engelleri ifade etmektedir. Çarpışma bulunmayan tüm yapılandırmalar ise **(3.31)**'deki gibi yazılır.

$$Q_{free} = Q \setminus QO \tag{3.31}$$

UMAY'da çarpışma bulunmayan yapılandırmalar seçerek yol çıkarma problemini PRM (*Probablistic Road Map*) tabanlı SBL (*Single Query Bi-directional PRM with Lazy Collision Checking*) algortiması çözmektedir. Bu algoritmanın yazılımsal olarak uygulanma yöntemi OMPL ile standart olarak gelmektedir. PRM algoritması, temel olarak çalışma uzayında çarpışma bulunmayan ve rastgele konuma sahip düğümlerden ve bu düğümlerin birbirlerine bağlayan yollardan oluşmaktadır [50]. Düğümlerin hespi birer robot eklem yapılandırmasını temsil etmektedir. PRM basit olarak dört aşamadan oluştmaktadır. Bu aşamalar aşağıdaki gibi sıralanabilir:

- Yol haritasınıın içinde rastgele düğümler oluştururlur
- Bu düğümler birbirlerine bağlanır
- Aralarında bağlanma bulunmayan düğümlerin bağlanması için yeni düğümler ve bağlantılar kurulur
- Bir lokal yol planlayıcı ilk yapılandırma ile son yapılandırmayı birbirine bağlar

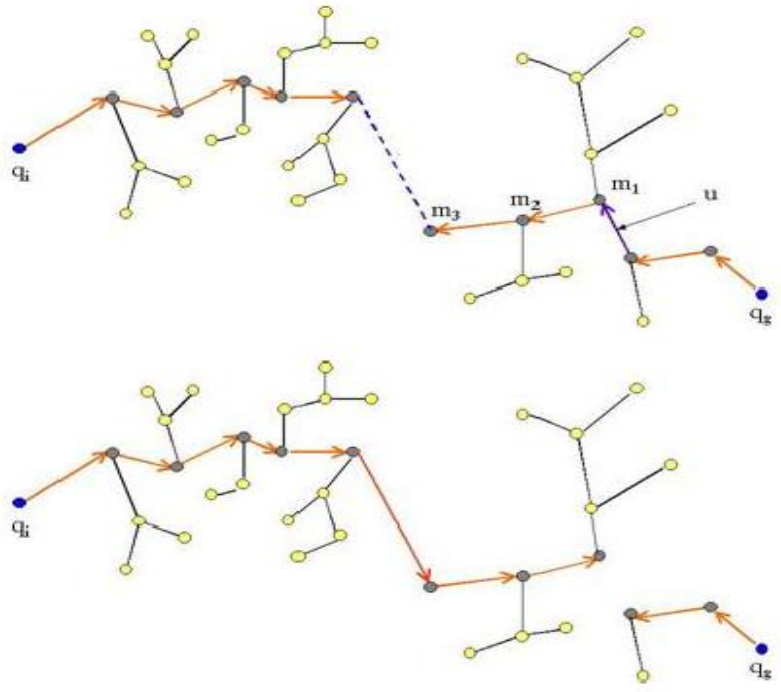
Bu aşamalar sırasıyla Şekil 3.7'de verilmiştir.



Şekil 3.7 : PRM 'yi oluşturma basamakları.

PRM tabanlı algoritmaların dezavantajı, çalışma zamanlarının çoğunu çarpışmaları kontrol etmek için ayırmalarıdır. UMAY'da, bu zaman harcamasını azaltan ve PRM

tabanlı olan SBL algortiması kullanılmıştır. SBL algoritması, tüm boş alanları keşfederek yol haritası çıkarmak yerine, birbirlerine çok yakın olan iki düğüm arasında bu işlemi yapmaktadır [54]. Boş alanları bulma kısmı, başlangıç ve hedef konumlarından 2 ayrı ağaç yapılandırması çıkararak gerçekleştirilmektedir. SBL algoritması çarpışma kontrollerini kesinlikle gerekmedikçe yapmamaktadır. Deneysel olarak bu işlemlerin planlama zamanını kısalttığı idda edilmektedir [54]. SBL algoritmasında, eklem değişkenleri olan başlangıç konumu q_i'den ve hedef konumu q_g'den köklenen iki ayrı ağaç yapılandırması bulunmaktadır. Bu ağaçlara rastgele olarak düğümler eklenmektedir. Ağaçlar rastgele eklenen düğümler ile dallanmaktadır. Algoritmanın diğer bölümündeyse bu ağaçların birleştirilme işlemi yapılmaktadır. Belirlenen tekrarlanma sayısından sonra q_i ve q_g arasında çarpışmasız bir yol haritası bulunamazsa, çarpışmasız bir yol haritası yoktur veya algoritma bu yol haritasını bulmakta başarız olmuştur [54]. SBL'nin ağaçlanan yapısı Şekil 3.8 'de gösterilmiştir.



Şekil 3.8 : SBL'nin ağaçlanma yapısı

Yukaridaki şekilde u ile gösterilenler çarpışma bulunmayan yolları, m ile gösterilenler çarpışma bulunmayan düğümleri simgelemektedir.

3.3.2 Yörünge planlama

Bir eklem için yörünge, eklem yapılandırmasının zamana göre değişmesidir yani q, eklem konumunu temsil ederse, eklem konumu zaman göre değişen bir fonksiyon olmaktadır. Eklemin konumu zamana bağlı bir fonksiyon olduğu için eklemin hızını ve ivmesini hesaplayabiliriz [9]. UMAY'da yörünge planlaması, bir başlangıç noktasından bitiş noktasına kadar olan hareketi temsil etmektedir. Kısacası yörünge planlama, yalnızca bir eklem için ; ilk an t_i'yi ve ilk konumu $q(t_i)$' yi, son anı t_s'yi ve son konumu $q(t_s)$'yi kapsamaktadır [50]. UMAY'ın eklemlerinin yörünge planlayıcısı olarak kübik polinom yörünge planlayıcısı kullanılmıştır. Kübik polinom yörünge planlamasının çıkartılmasında dört kısıtlama bulunmaktadır bunlardan ilk ikisi, **(3.32)**'de verilen başlıngıç anındaki konum ve **(3.33)**'te verilen başlangıç anındaki hızdır.

$$q(t_i) = q_i \tag{3.32}$$

$$\dot{q}(t_i) = \dot{q}_\iota \tag{3.33}$$

Kalan iki kısıt ise **(3.34)**'te verilen son andaki konum ve **(3.35)**'te verilen son andaki hızdır.

$$q(t_s) = q_s \tag{3.34}$$

$$\dot{q}(t_s) = \dot{q}_s \tag{3.35}$$

Dört kısıtı olan yörünge planlayıcıyı çözüp hızları ve konumları elde etmek için en az dört adet birbirinden bağımsız katsayıya ihtiyaç bulunmaktadır. Bu sebebten dolayı kübik polinom yörünge planlayıcısının yapsı Denklem **(3.36)**'daki gibi olacaktır.

$$q(t) = a_o + a_1 t + a_2 t^2 + a_3 t^3 \tag{3.36}$$

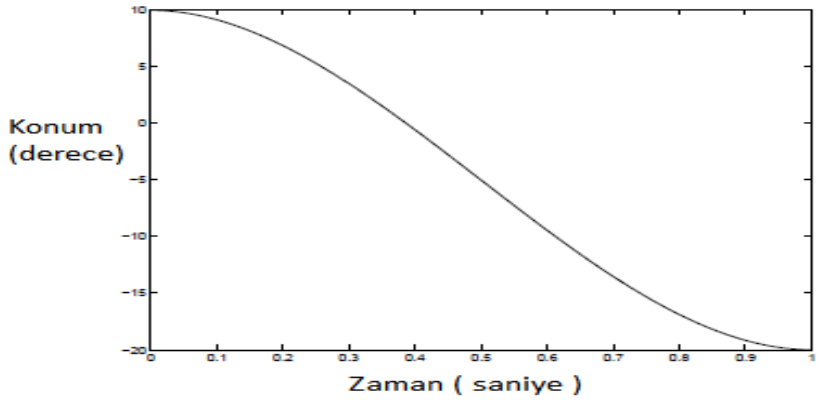
Yukarıdaki denklem eklemin konumunun zamana göre değişimini ifade etmektedir. Denklem **(3.37)**'de ise eklemin hızının zaman göre değişiminin ifadesi verilmiştir.

$$\dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2 \tag{3.37}$$

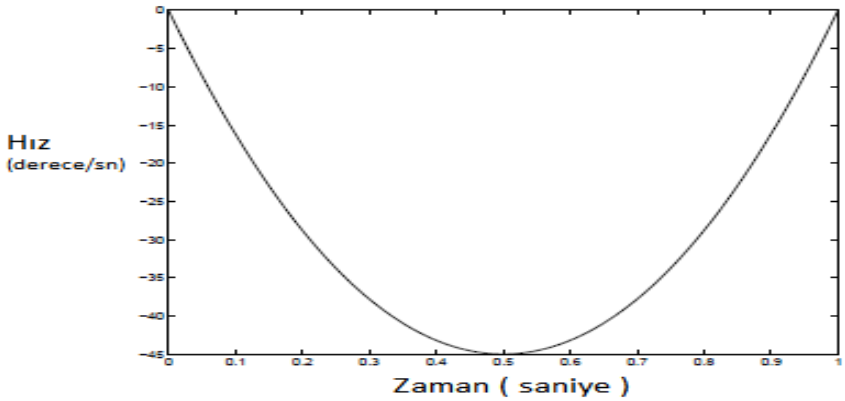
Denklem **(3.36)** ve **(3.37)** ilk konum, ilk hız ve son konum, son hız için birleştirilirse dört bilinmeyenli dört denklem elde ederiz.

$$
\begin{aligned}
q_i &= a_o + a_1 t_i + a_2 {t_i}^2 + a_3 {t_i}^3 \\
v_i &= a_1 + 2\, a_2 t_i + 3 a_3 {t_i}^2 \\
q_s &= a_o + a_1 t_s + a_2 {t_s}^2 + a_3 {t_s}^3 \\
v_s &= a_1 + 2\, a_2 t_s + 3 a_3 {t_s}^2
\end{aligned}
\qquad \textbf{(3.38)}
$$

Bu denklemler, belirlenen bir başlangıç değeri zaman değeri, eklem konumu ve bitiş zaman değeri, eklem konumu için çözülerek katsayılar bulunur. Zaman artış değeri de işin içine katılarak hesaplanan hızlar ve konumlar eklemlere uygulanıp yörünge planlama yapılmaktadır. Kübik polinom yörünge planlayıcının ortaya çıkardığı konum ve hız grafikleri Şekil 3.9 ve Şekil 3.10'da verilmiştir.



Şekil 3.9 : Eklemin konum grafiği.

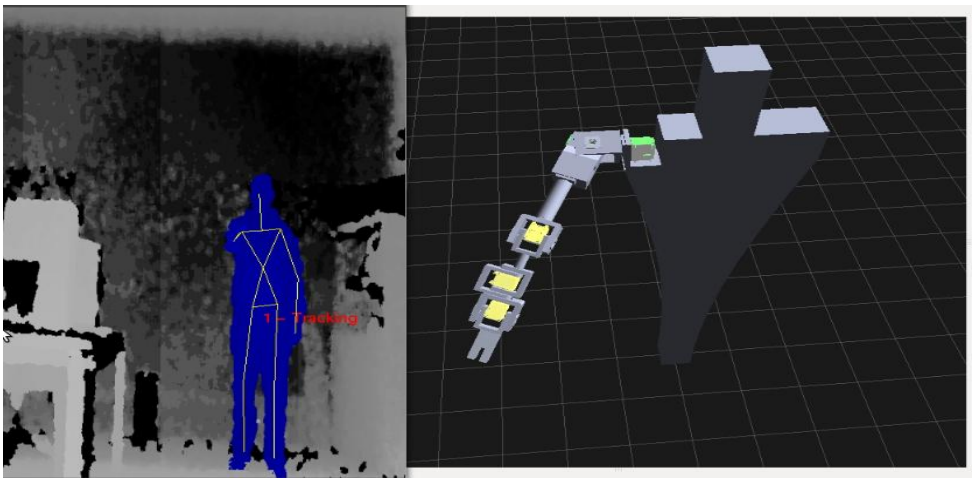


Şekil 3.10 : Eklemin hız grafiği.

Kübik polinom yörünge planlamayla detaylı olarak [50,55]'de işlenmiştir.

3.4 Kinect İle Etkileşim

UMAY'da, basit insan hareketlerinin takliti Kinect algılayıcısı kullanılarak yapılmaktadır. Basit insan hareket takliti yapabilmek için Bölüm 2.7'de anlatılan OpenNI kütüphanesi ve Bölüm 2.3.3'te anlatılan Dönüşüm paketi kullanılmaktadır. OpenNI kütüphanesi bize Kinect ile iletişimi sağlama imkanı veren bir sürücüdür. Kullanıcı, OpenNI tarafından algılanılır ve üzerine bir iskelet yerleştirilir böylece kullanıcının takibi başlamış olur. Kullanıcı üzerine oturtturulan iskelet, ***tf*** başlığı altında ROS'da yayınlanmaktadır ve basit hareket takliti bu başlığa kayıt olunarak yapılmaktadır. UMAY'ın tasarımın aşamasında sadece bir kolu olduğu için, ***tf*** başlığı altında sadece sol kolun eklem hareketlerine bakılmaktadır. Analiz edilen eklemler sol kolun omuzunun, dirseğe kadar olan kısmı ve dirsekle ele kadar olan kısımdır. Kullanıcının dirsek ile omuz arasında oluşan 3 boyutlu açılar, UMAY'ın omuz eklemlerini hareket ettirmektedir. Dirsek ve el arasındaki 3 boyutlu açılar, UMAY'ın dirsek eklemini hareket ettirmektedir. Taklit hareketleri yapıtırılırken dikkat edilmesi gereken husus, kullanıcı OpenNI Kütüphanesi tarafından takibe başlandığında kullanıcının ilk konumunun referans olarak alınmasıdır. Referans alınan konum üzerinde bazı değişiklikler yaparak doğal taklit işlemi başlatılabilir. Düzeltme işleminde kullanıcının dirsek ile omuz arasındaki uzuvun birim vektör haline getirilmesi daha sonra bu uzuvun kartezyen uzaydaki hareketlerinin açılarının ters sinüsü ve ters kosinüsü alınarak UMAY'ın eklem açılarının bulunmasından ibarettir. Aynı işlem omuz ile el arasındaki uzuvun için tekrarlanır. Şekil 3.11'de kullanıcı ve UMAY arasındaki etkileşim gösterilmiştir. Daha detaylı etkileşim şekilleri Bölüm 4.2'de bulunmaktadır.



Şekil 3.11 : UMAY'ın kullanıcıyı takip etmesi.

4. UYGULAMA

Bu bölümde, Bölüm 3'te anlatılan tüm teorik çalışmaların ROS'da uygulaması detaylandırılacaktır ve çıkan sonuçlar verilecektir. Bölüm 4 iki kısma ayrılmıştır, ilk bölümde ters kinematik ve hareket planlama uygulamalarının nasıl gerçekleştirildiği ve elde edilen eklem konum ve tork sonuçları bulunmaktadır. İkinci bölümde ise Kinect algılıyıcısı ile etkileşim ve elde edilen eklem konum sonuçlar bulunmaktadır. Tüm uygulamalar ROS işletim sisteminin Fuerte dağıtımında yapılmıştır.ROS'un Fuerte dağıtımının kurulumu [56]'da bulunabilir. ROS'a ev sahipliği yapan işletim sistemi ise Ubuntu 11.10'dur.

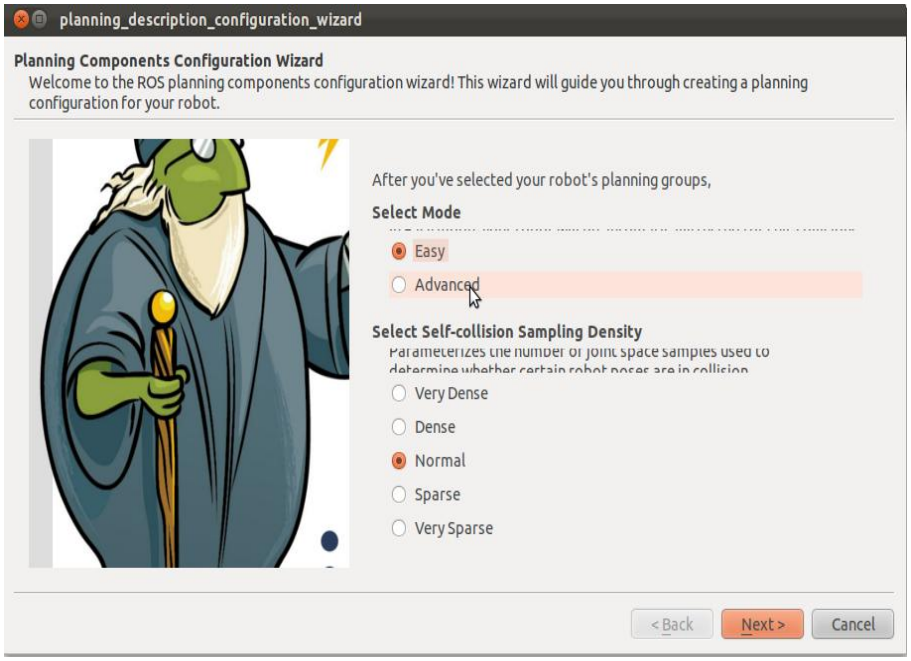
4.1 Hareket Planlama Uygulamaları

Hareket planlamasına başlanmadan önce UMAY'ın Bölüm 2.3.5'te anlatılan robot tanım dosyası çıkarılmıştır. Uygulamanın ilk basamağında, eklem limitleri de göz önünde tutularak robotun çarpışma bulunmayan eklem yapılandırma uzayının tespit edilmesi yapılmıştır. Eklemlerin çarpışma bulunmayan eklem yapılandırma uzayının çıkarılması kullanılmış olan kol hareket planlama paketi [57]'de bulunabilir. Bu paket grafik arayüze sahip kullanımı kolay bir pakettir. Ubuntu'da komut satırına aşağıdaki komut yazılarak, bu grafik arayüzü çalıştırılır.

roslaunch planning_envoriment planning_description_configuration_wizard.launch urdf_package:=UMAY_description urdf_path:=urdf/UmayKol.urdf.xacro **(4.1)**

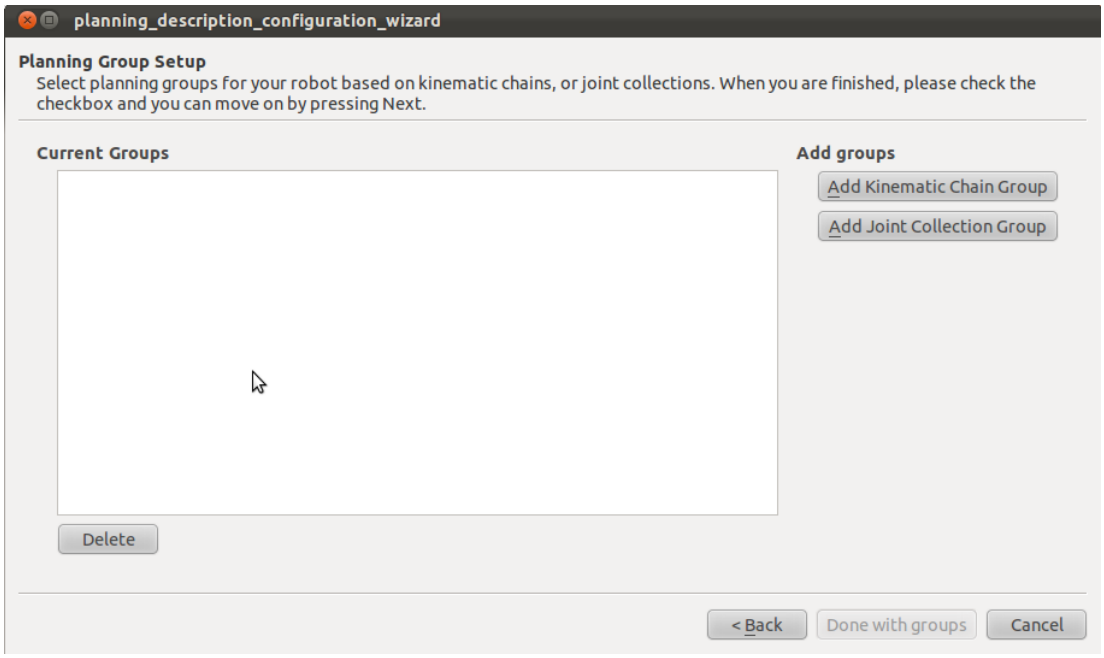
Yukarıda bulunan *roslaunch* komutu bir çok düğümü aynı anda çalıştırabilen bir ROS komutudur. Kullanıcağımız paketin adı *planning_envoriment,* grafik arabirimin bulunduğu dosya ise *planning_description_configuration_wizard.launch'*dır. Geriye kalan kısımlar ise, UMAY'ın yapılandırma dosyalarının bulunduğu paketi ve UMAY'ın kolunun robot tanım dosyasını içermektedir. Bu komut yazıldıktan sonra grafik arabirim açılacaktır ve gerekli tercihler yapılıp eklemlerin çarpışmasız yapılandırma uzayı bulunacaktır.

Komut yazıldıktan sonra ekrana çıkacak olan ilk görüntü Şekil 4.1'de verilmiştir.



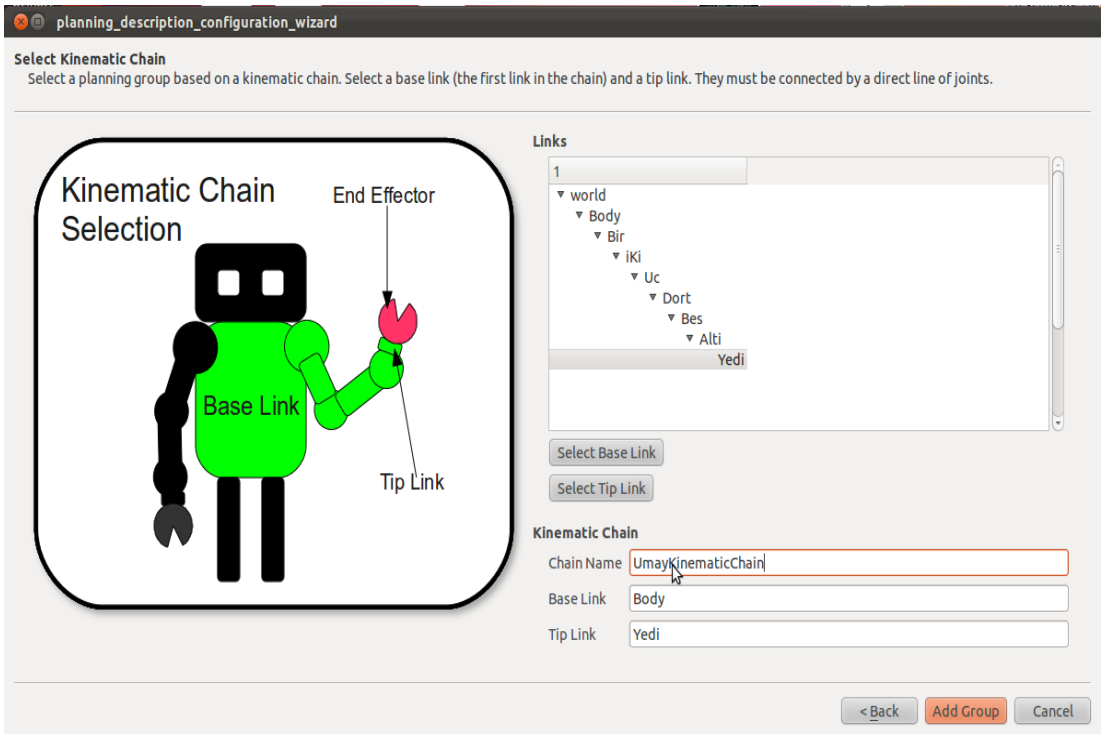
Şekil 4.1 : Çarpışması eklem yapılandırma programının arayüzü.

Planlama yapılandırmasında karşımıza iki adet seçenek çıkmaktadır. Bunlardan ilki ileri ya da kolay düzeyde hazırlayabileceğimiz programı kullanım yapılandırması diğeri ise örnekleme yoğunluğudur. Örnekleme yoğunluğu seçime göre çok büyük sayılarda eklem konumlarını rastgele olarak robota uygular ve çapışma olan yapılandırmalar yapılandırma uzayından atılır[58]. Görüldüğü gibi örnekleme yoğunluğunda beş adet seçenek bulunmaktadır. Yukardan aşağıya doğru örnekleme sayısı artmaktadır. Örnekleme sayısının artması daha gerçekçi bir model oluşturulmasını sağlasa da hesaplama zamanı bilgisayarın hızına göre artmaktadır. UMAY'da programın kullanım yapılandırması *kolay* olarak seçilmiştir ve örnekleme yoğunluğu *normal* olarak seçilmiştir. Bu işlemlerin hemen ardından ileri tuşuna basılarak bir sonraki aşamaya geçilir. Bundan sonraki aşamada, robot tanım dosyasından elde edilen verilerle robotun kinematik zinciri seçilmektedir. Kinematik zincir seçilirken robot kolun hangi kısımların bu kinematik zincirin içinde bulunması gerektiğini göz önünde bulundururuz böylece hareket planlama seçilen kinematik zincire göre yapılır. İleri tuşuna basıldıktan sonra karşımıza planlama grubu seçimi ekranı çıkacaktır. Planlama grubu seçimi ekranı Şekil 4.2'te verilmektedir.



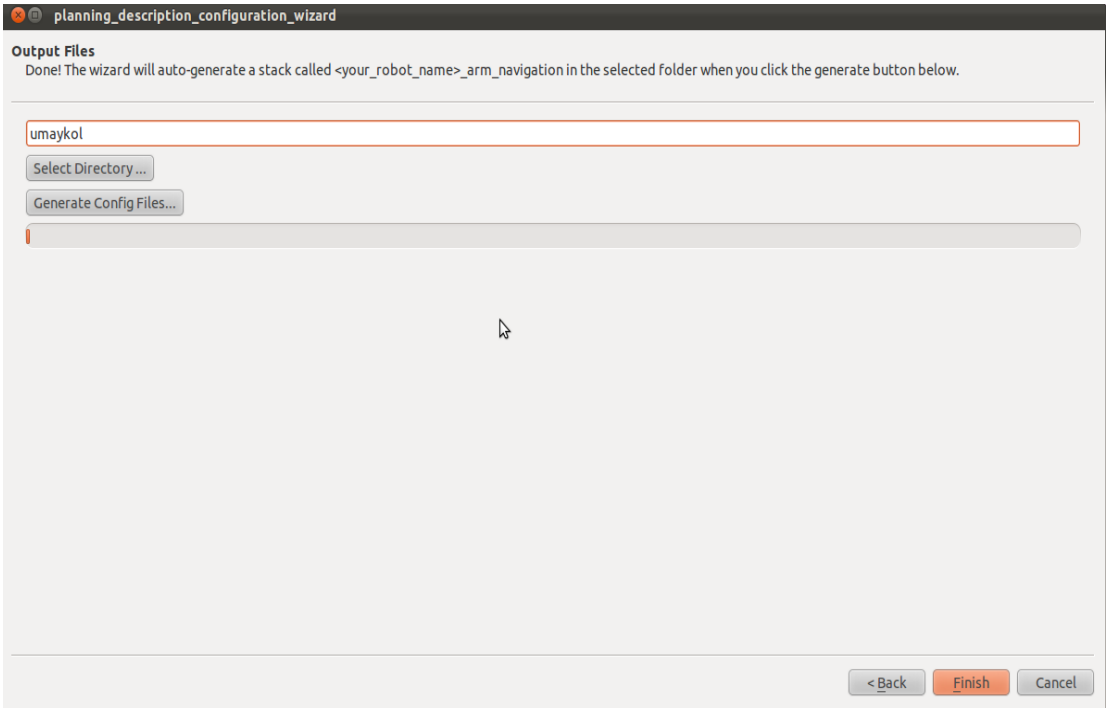
Şekil 4.2 : Planlama grubu için seçim ekranı.

Burada iki adet seçenek bulunmaktadır; kinematik zincir gubu ve eklem grubu. UMAY kinematik zincirden oluşmuş bir seri manipulatör olduğu için ilk seçenecek olan *Add Kinematic Chain Group* seçeneğinin tıklanması gerekmektedir. Bu tuş tıklandıktan sonra kinematik zincirin içine koyacağımız bağları seçmek için kullanılacak ekran karşımıza çıkar. Bu ekran Şekil 4.3'te verilmiştir.



Şekil 4.3 : Kinematik zincir seçim ekranı.

Şekil 4.3'de verilen ekranda zincirimizin ismi *UMAYKinematicChain* olarak seçilmiştir. Ana kordinat ekseninin koyulacağı bağ *Body* olarak isimlendirilen insansı yapıdır bu yapı Şekil 3.11'de gösterilmiştir. Robot kolunun çevresiyle etkileşime girip, nesneleri hareket ettirebildiği tutucu yapısından bir önceki kısma uç bağ denilmektedir, UMAY'ın ilk tasarımında bir tutucu yapılandırması bulunmadığı için uç bağ *Yedi* olarak seçilmiştir. Kinematik zincir grubu ekleme aşaması bittikten sonra gelen kısım otomatik olarak yapılandırma dosyalarını istelinen bir klasöre koyan ekrandır. Bu ekran, Şekil 4.4'de verilmiştir.



Şekil 4.4 : Yapılandırma dosyalarını hazırlayan ekran.

Bu ekranda, tüm kol için gerekli olacak hareket planlamaya ait yapılandırma dosyaları otomatik olarak hazırlanacaktır ve bir ROS yığını haline çevrilicektir. Bu ROS yığını içinde 2 tane alt klasör bulunmaktadır, bunlardan birincisi *config* isimli klasör diğer ise *launch* isimli klasördür. İlk klasör, hareket planlama için gerekli tüm yapılandırma dosyalarının bulunduğu klasördür. Bu klasör içindeki dosyalar aşağıdaki gibidir.

- *joint_limits.yaml* : Bu dosyada eklemlerin hız,ivme sınırları bulunmaktadır bu limitler daha iyi sonuçlar almak için istenildiği taktirde elle değiştirilebilir [58].

- *umay_planning_description.yaml* : Çarpışma kontrol programının çalışması için gerekli yapılandırma değişkenlerinn bulunduğu dosya. Kısaca bu dosyada hangi eklemlerin hangi eklemlerle çarpışmaması gerektiği belirtilmektedir.
- *ompl_planning.yaml* : Bu dosyada OMPL'nin kullanacağı planlayıcıların yapılandırma değişkenleri bulunmaktadır.

Otomatik olarak oluşturulan ikinci klasörde ise aşağıdaki dosyalar bulunmaktadır

- *UMAY_planning_envoriment.launch* : UMAY'ın planlama çevresi hakkında detaylı bilgi veren bir dosyadır ve bu çevre bilgilerinin alınıp işlenebilmesi için gerekli bir çok düğüm bu dosya ile çalıştırılır.
- *constraint_aware_kinematics.launch* : Bu dosya, eklemlerin sınırlarını göz önünde bulundurarak ters kinematik işlemini yapan kontrolörü çalıştırmak için kullanılır. Bu kontrolör, Bölüm 3'te anlatılan kontrolör yazılım eklentisinin altyapısına sahiptir.
- *move_arm.launch* : Planlayıcının çalıştırıldığı dosyadır.
- *trajectory_filter_server.launch* : Planlayıcı, çarpışmasız yol haritasını çıkardıktan sonra eklemlere yollanacak olan konum bilgilerinin yörünge planlamasını yapan düğümler bu dosyada çalıştırılır.

Yukarda açıklanan otomatik olarak hazırlanan dosyaların hakkında detaylı bilgi [59]'da bulunmaktadır. Hareket planlama yöntemlerini, gerçekleştirebilmek için Bölüm 2.3.6'da bahsedilen *Warehouse Viewer'*ı kullanacağız. Bu program, ileri hareket planlama görevlerini kayıt edebilen, saklayabilen ve tekrar oynatabilen ileri seviyede bir programdır [58]. Hareket planlamanın çıkış değerlerini simülasyon ortamında gözlemleyebilmek için ilk önce robotun Gazebo ortamında çalıştırılması gerekmektedir. UMAY'ın Gazebo ortamına aktarılmasını sağlayan komut aşağıdaki gibidir.

roslaunch umay_description umaykol.launch **(4.2)**

Yukarıdaki komutta *umay_description,* UMAY'ın tüm robot tanım dosyalarını içerisinde tutan bir ROS paketidir. Komutun devamında bulunan *umaykol.launch* ise robotun tanım dosyalarını ROS'un değişken sunucusuna atan ve Gazebo ortamını çalıştıran düğümlerin hepsini çalıştırmaya yaramaktadır. Bundan sonraki aşamada,

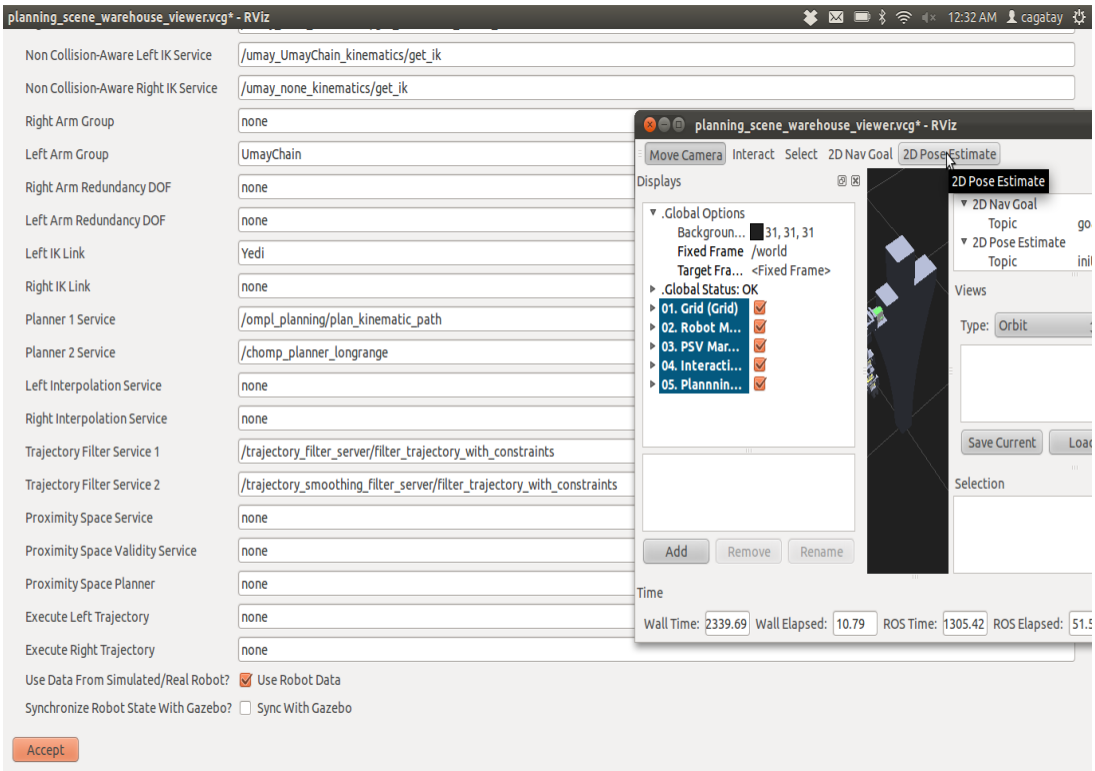
bize robot kolun eklemlerinin kontrolör altyapısını sağlayan düğümlerin çalıştırılması vardır. Kontrolörün çalıştırılması için gerekli komut **(4.3)**'de verilmiştir.

roslaunch umay_description r_arm_contoller.launch **(4.3)**

Kontrolör Bölüm 3'de anlatılan yapıda, PID kontrolü ile pozisyon kontrolü yapan bir kontrolördür. Hareket planlayıcıyı çalıştırmak için gerekli dosya bir önce anlatılan çarpışmasız eklem yapılandırma uzayını yaratan program tarafından otomatik olarak yaratılmıştır. Bu program aşağıdaki komutun, komut satırına yazılmayasıyla çalıştırılabilir.

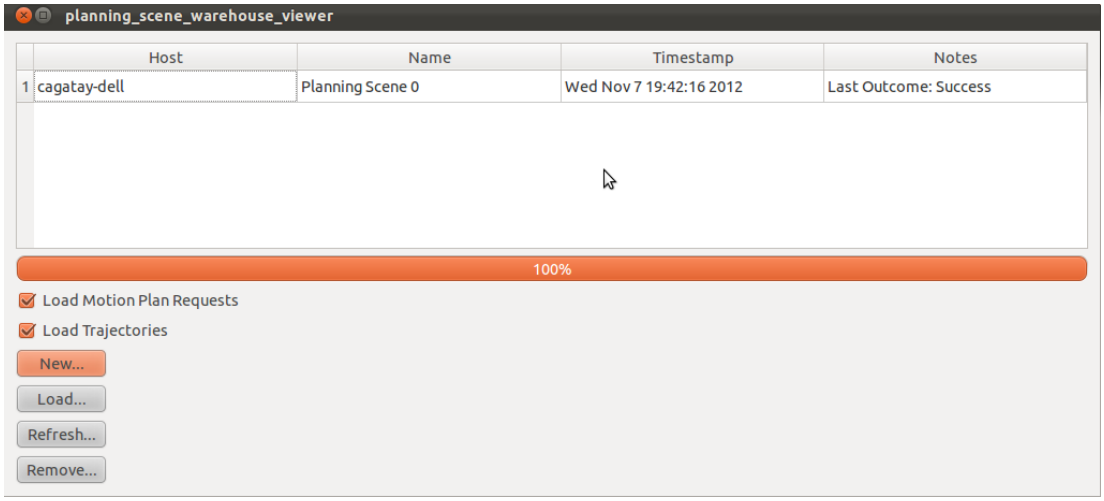
roslaunch umay_arm_navigation planning_scene_warehouse_viewer.launch **(4.4)**

Yukarıdaki komutta *umay_arm_navigation*, UMAY'ın kolunun otomatik olarak yaratılan hareket planlaması için gerekli yapılandırma dosyalarını tutan ROS yığınıdır, *planning_scene_warehouse_viewer.launch* ise hareket planlama kullanıcağımız programı çalıştırmaya yaramaktadır. Bu komut yazıldıktan sonra, robotuda görebilmemiz için ***rviz*** programı da açılmaktadır. Komuttan hemen sonra ekrana gelicek olan görüntü Şekil 4.5'te verilmiştir.



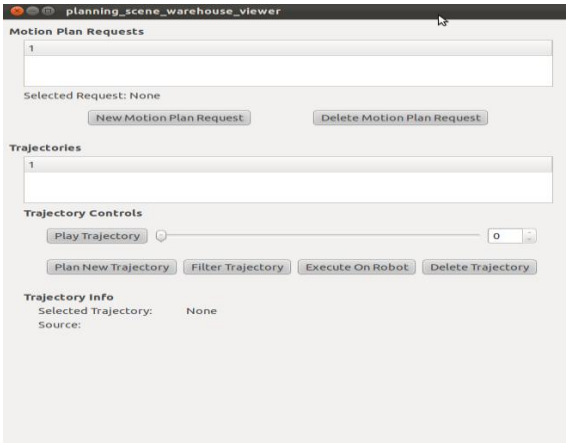
Şekil 4.5 : Hareket planlıyıcının arayüzü.

Şekil 4.5'te görülen arayüz ile yazılan veya hazır olarak kullanılmak istenilen ters kinematik, yörünge süzgeçleri, yörünge kontrolörleri için gerkeli olan servisler uygun yerlere yazılır bunlara ek olarak hareket planlayıcıyı Gazebo ortamında çalıştıracağımız için *Use Robot Data* kutucuğu tıklanır. Kabul et düğmesine basıldıktan sonra karşımıza çıkacak ekranda, daha önceden kaydettiğimiz yörünge planlamalar ve hareket planlamalar gözükecektir. İstenildiği taktirde kaydedilen planlar robotun üstünde tekrar çalıştırılabilir. Bu ekranda, yeni düğmesine basarak yeni hareket planlama ve yörünge planlama işlemleri yapılır. Hareket planlarını kaydeden ekran Şekil 4.6'da verilmiştir.



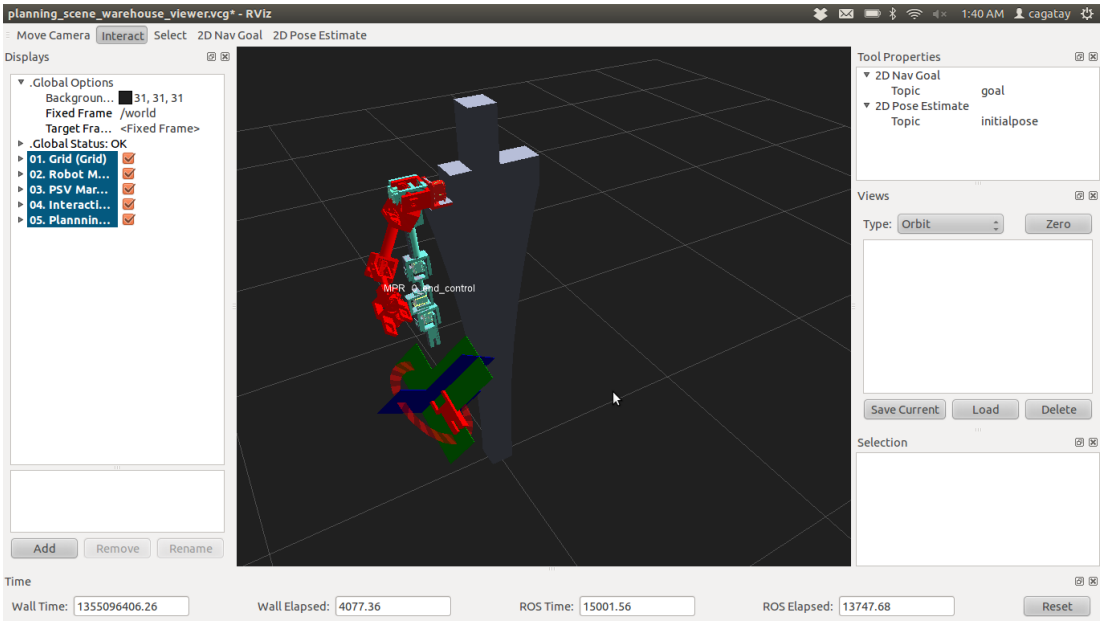
Şekil 4.6 : Hareket planlarını yapılandırma ekranı.

Hareket planlarını yapılandırma ekranında yeni düğmesine basarak, yeni hareket planlanlama işlemine başlanır. Hareket planı yapılandırma işlemi Şekil 4.7'deki ekranda yapılmaktadır.



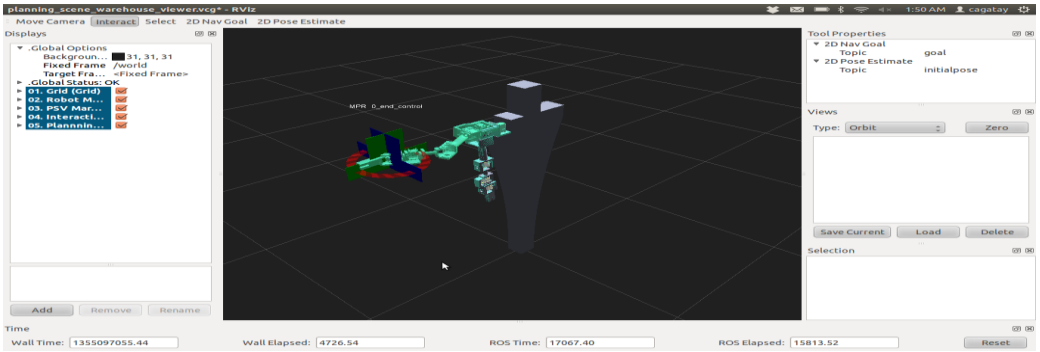
Şekil 4.7 : Hareket planı yapılandırma ekranı

Şekil 4.7'deki hareket planı yapılandırma ekranında, yeni bir planlayıcı yaratmak için yeni hareket planı isteği düğmesine basılır. Bu düğmeye basıldıktan sonra planlayıcıya giden bu istek ***rviz***'de görülmektedir. Hareket planlayıcının çalışması için ilk gerekli koşul, uç bağın yeni konumunun ve yöneliminin belirtilmesidir. Bu işlem ***rviz***'de yapılır. Ters kinematik servisi, uç eklemin istenilen konuma gidip gidemeceğini bize rapor eder. Bu işlem Şekil 4.8'de verilmiştir.



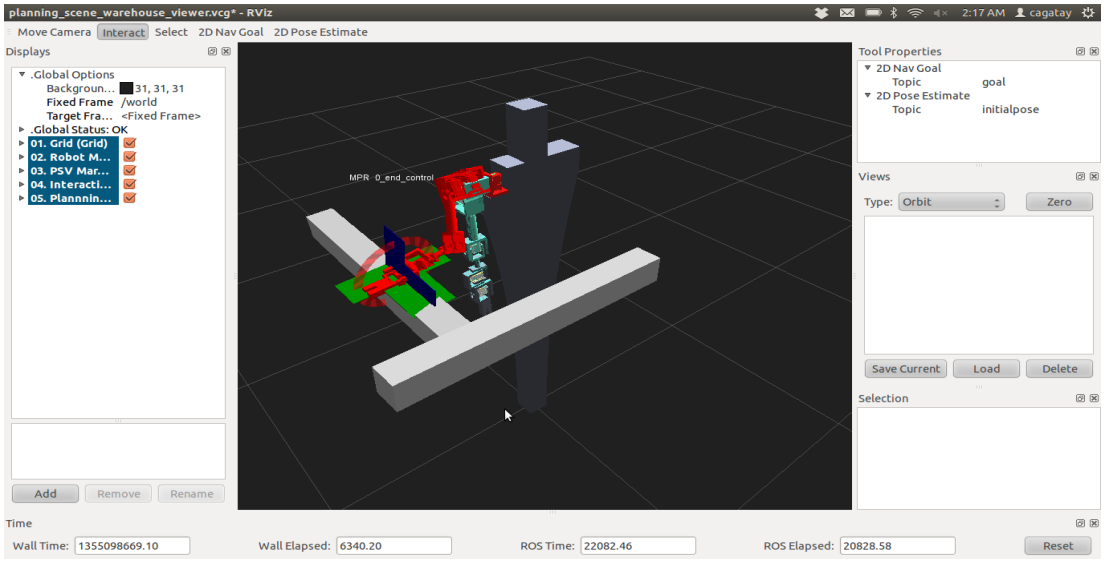
Şekil 4.8 : Hatalı uç bağ yapılandırması.

Şekil 4.8'de görüldüğü gibi uç eklemin gitmek istediği konum ve yönelim ters kinematik servisi tarafından ulaşılamaz olarak gösterilmiştir. Hedef konum ve yönelimin ulaşılamaz olduğunu UMAY'ın kol renginin kırmızıya dönmesiyle anlamaktayız. Ters kinematik servisinin hata bulmadığı bir uç bağ yapılandırması Şekil 4.9'da verilmektedir.



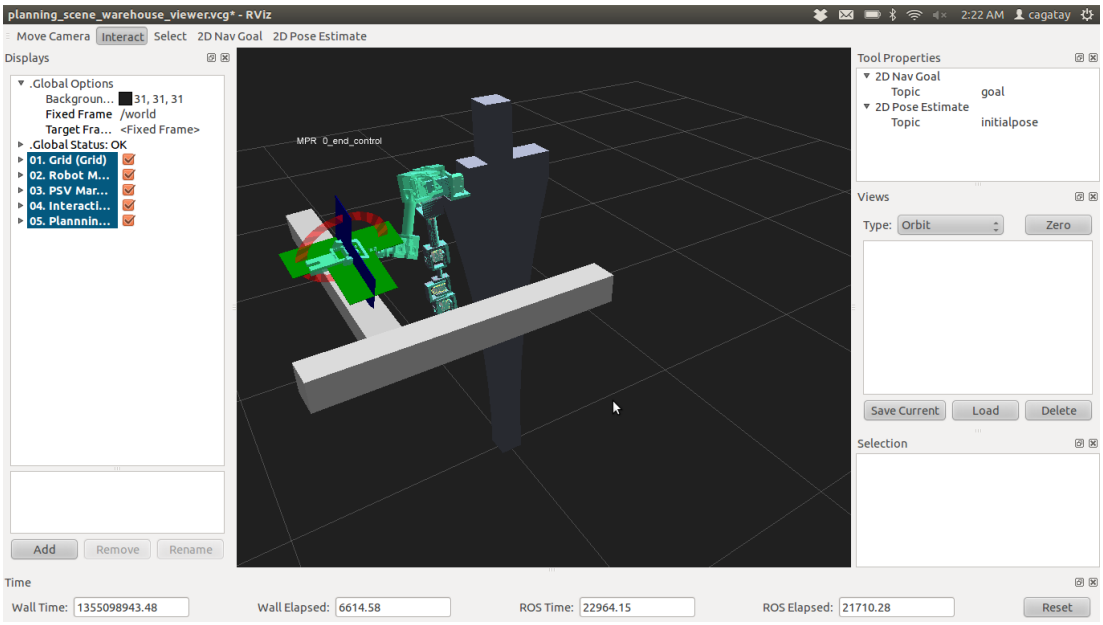
Şekil 4.9 : Hatasız uç bağ yapılandırması.

Hatasız bir uç eklem yapılandırması elde edildikten sonra, tekrar Şekil 4.7'deki hareket planlayıcı yapılandırma ekranı açılır ve yeni yörünge planlama düğmesine basılır böylece istenilen konuma gidecek olarak bir hareket planımız olmuş bulunmaktadır. Eğer bu hareket planının yörüngesini süzgeçten geçirmek istersek, Yörünge süzgeçleme düğmesine basılması yeterlidir. Buraya kadar anlatılan hareket planlama süreçlerinde hiç çarpışma nesnesi konulmamıştır. Hareket planlayıcı yapılandırma ekranından çarpışma nesnesinin boyutlarını şeklini ayırlamak ve çarpışması nesnesini ***rviz***'e koymak mümkündür. Böyle bir yapılandırma Şekil 4.10 'da verilmiştir.



Şekil 4.10 : Çarpışma durumundaki uç eklem yapılandırması.

Şekil 4.10'da görülen yapılandırma çarpışma durumundadır bu yüzden ters kinematik servisi tarafından hatalı olarak rapor edilmiştir bu yüzden hareket planlayıcı herhangi bir plan yapamamaktadır. Çarpışmasız uç eklem yapılandırması Şekil 4.11'de verilmiştir. Çarpışmasız uç bağ yapılandırması bulunduktan sonra aynı hareket planı yapılandırma işlemleri tekrar edilerek yeni bir plan elde edilir. En son aşamada, hareket planlama yapılandırma ekranındaki robotun üzerinde çalıştır düğmesine basılır ve hareket planlayıcı ile Gazebo eş olarak çalışmaya başlar. Kontrolör altyapısı hareket planlıyıcıdan gelen eklem yörünge yapılandırmalarına uygun bir şekilde Gazebo'ya aktarılmış olan robotun eklemleri üzerinde kontrol algoritmasını çalıştırır. Tüm eklem yapılandırmaları *joint_states* başlığı altında kaydedilip grafiğe aktarılabilinir.

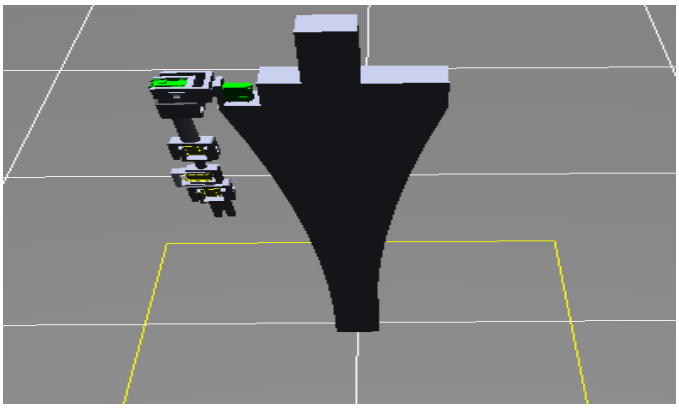


Şekil 4.11 : Çarpışma halinde bulunmayan uç bağ yapılandırması.

Şekil 4.11'de yapılandırmanın çarpışma halinde bulunmadığı ve ters kinematik servisi tarafından *kabul edilebilecek* bir konum, yönelim durumunda olduğu, robot kolunun turkuaz bir renk almasından anlaşılmaktadır.

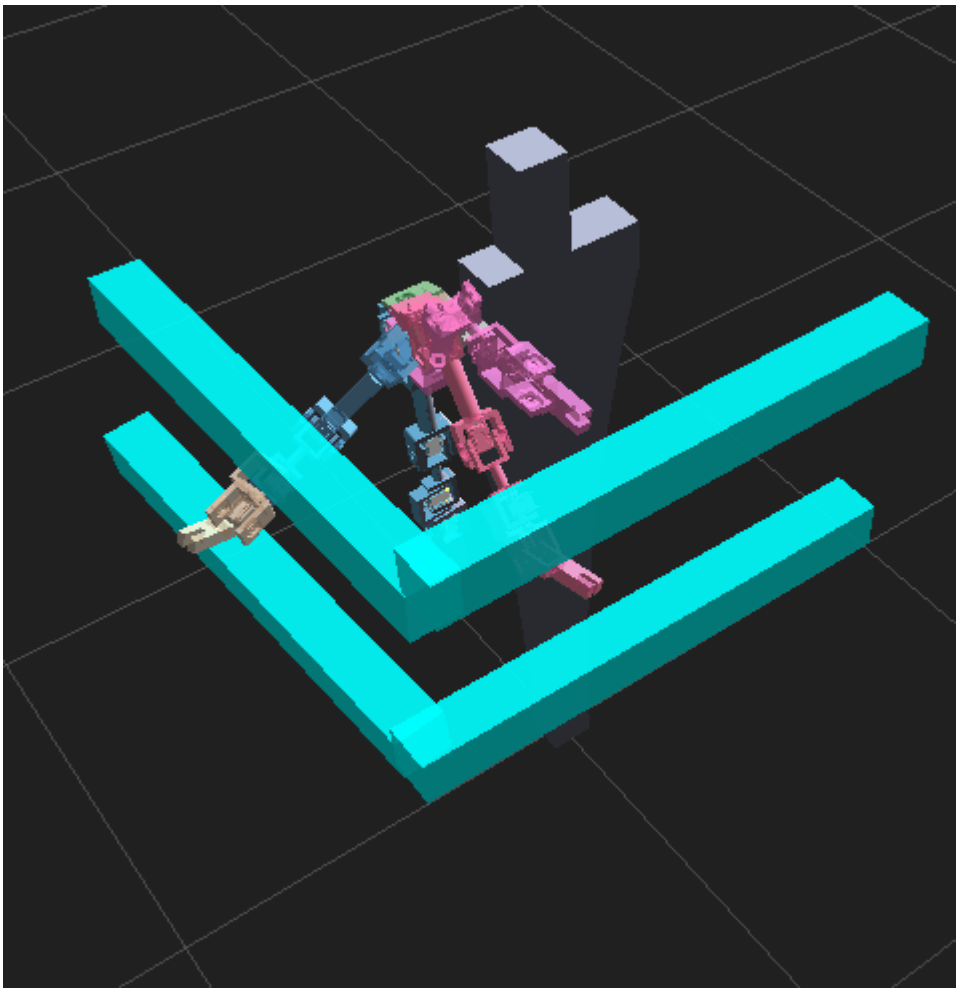
4.1.1 Hareket planlama uygulamasından elde edilen veriler

Hareket planma yapıldıktan sonra, planlama yapılandırmaları Gazebo ortamında çalıştırılmıştır ve Gazebo programının fizik motorundan faydalınarak eklemlere binen torklar elde edilmiştir. Bu bölümde farklı yapılandırmalar için eklemlere binen tork değerlerinin grafiği ve eklem konumlarının grafikleri verilmiştir. Herhangi bir hareket planlaması yapılmadan önce Gazebo ortamında UMAY'ın kolunun konum ve yönelimi Şekil 4.12'de gösterilmektedir.



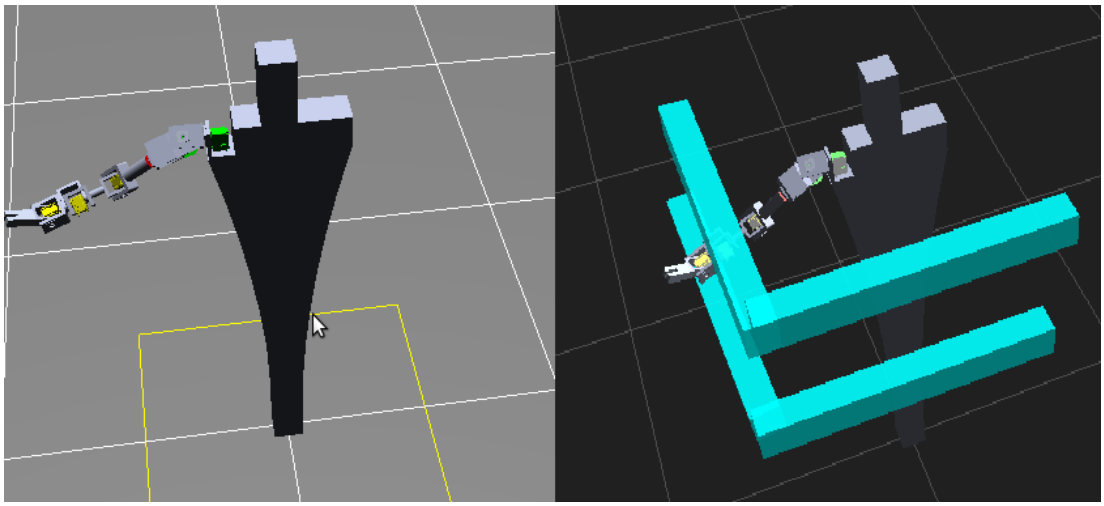
Şekil 4.12 : Hareket planlamadan önce UMAY'ın kolu.

Hareket planlamanın içerisinde çarpışabilecek engeller bulunmaktadır. Yapılan simülasyonda, UMAY'ın kolu herhangi bir engele çarpmadan üç ayrı yapılandırmaya ulaşacaktır ve bu yapılandırmalardayken UMAY'ın eklemlerine binen torklar gözlenecektir. UMAY'ın hareket planlama yapılandırmaları ve eksenler Şekil 4.13'te verilmiştir.



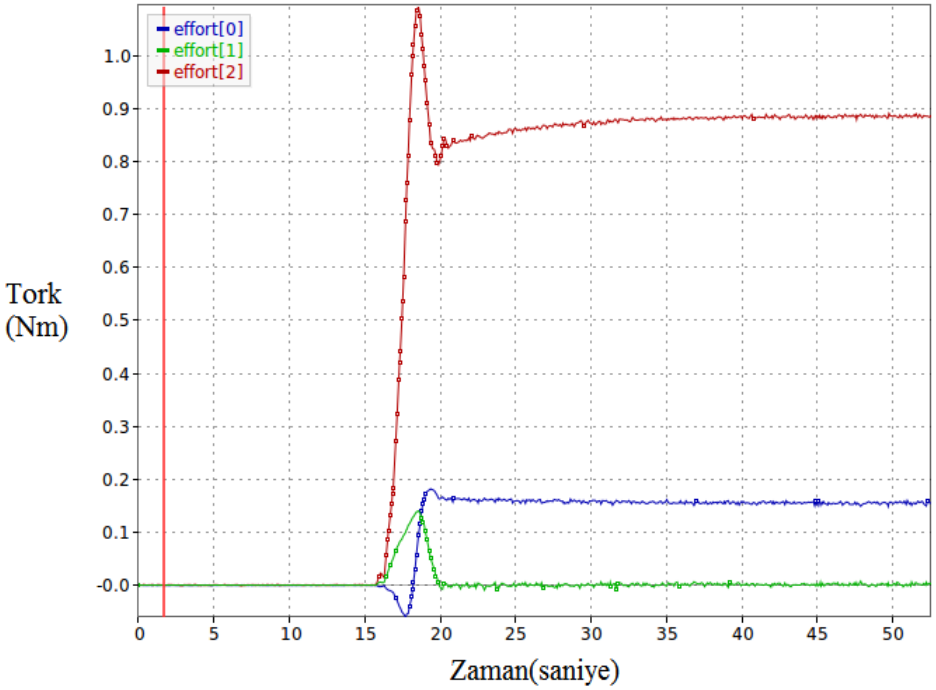
Şekil 4.13 : Hareket planlama yapılandırmaları.

Hareket planlama yapılandırmaları için bulunan yolların hepsi yörünge süzgeçinden geçirilmiştir ve arkada çalışan bir kontrolör yazılım eklentisi, süzgeçten gelen konumları robotun eklemlerine yansıtmaktadır. Hareket planlama yapılandırmalarının görüntülerini Bölüm 4.1'de anlatığı gibi *rviz*'de sağlamaktadır. İlk hareket planlama yapılandırması ve UMAY'ın Gazebo'daki konumu Şekil 4.14'te verilmiştir. Şekil 4.14'te, sağdaki görüntü Gazebo ortamına, soldaki görüntü ise *rviz* ortamına aittir.



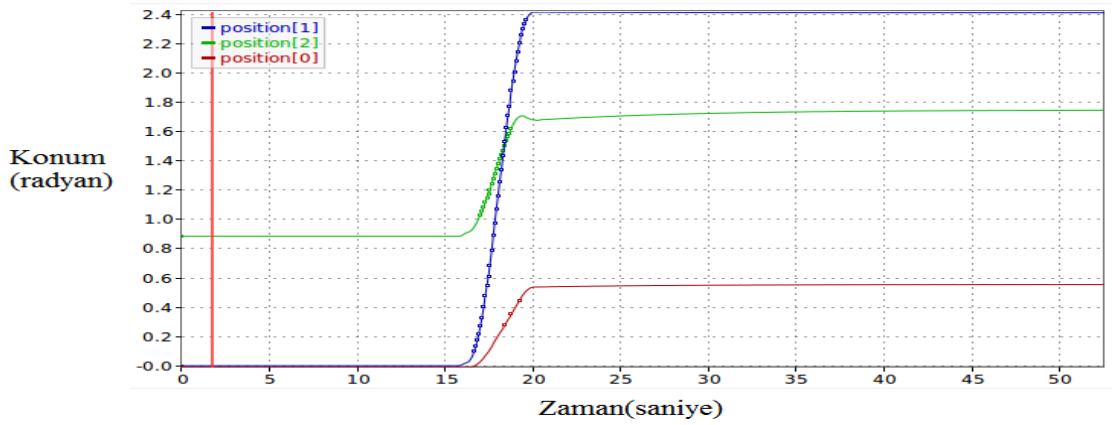
Şekil 4.14 : İlk yapılandırmadan sonra eklemlerin konumları.

Eklem sayıları 0'dan 2'ye kadar olmak üzere dirsek ve bilekteki üç eklemi temsil etmektedir. Eklem 0 ve eklem 1 bilekte bulunan eklemlerdir, eklem 2 ise dirsekte bulunan eklemdir. Eklem 3, 4, 5 ise omuza serbestlik derecesi veren eklemleri temsil etmektedir. İlk üç ekleme binen tork değerlerinin grafiği Şekil 4.15'te verilmiştir.



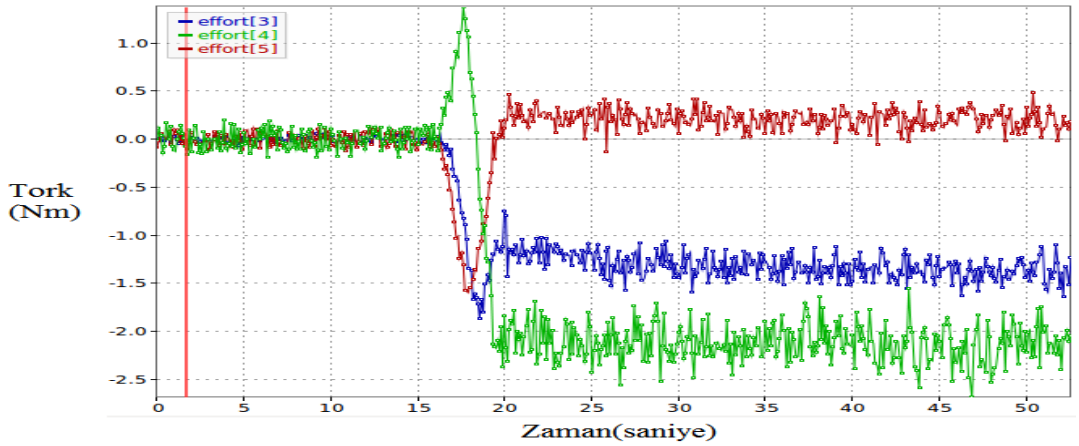
Şekil 4.15 : Birinci yapılandırma için bilekteki ve dirsekteki eklemlerin torkların grafiği.

Bilek ve dirseklerin ilk yapılandırma için aldıkları konumların grafiği Şekil 4.16 'da verilmiştir.

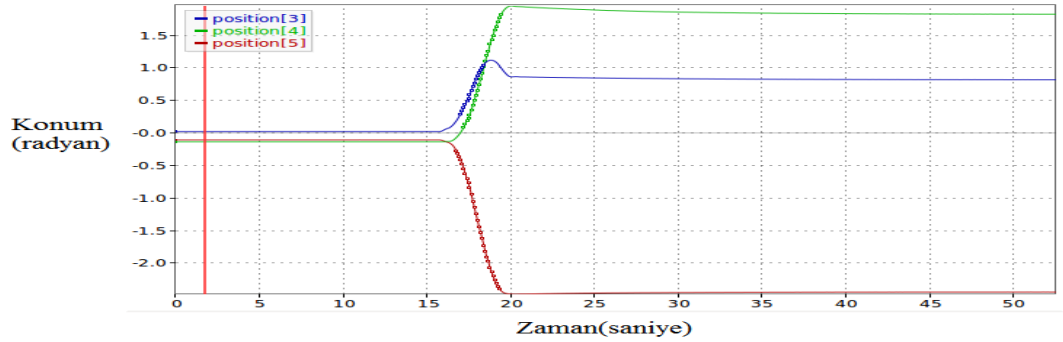


Şekil 4.16 : Birinci yapılandırma için dirsekteki ve bilekteki eklemlerin konum grafikleri.

Omuz bölgesindeki eklemlerin tork ve konum grafikleri Şekil 4.17 ve Şekil 4.18'de verilmiştir.

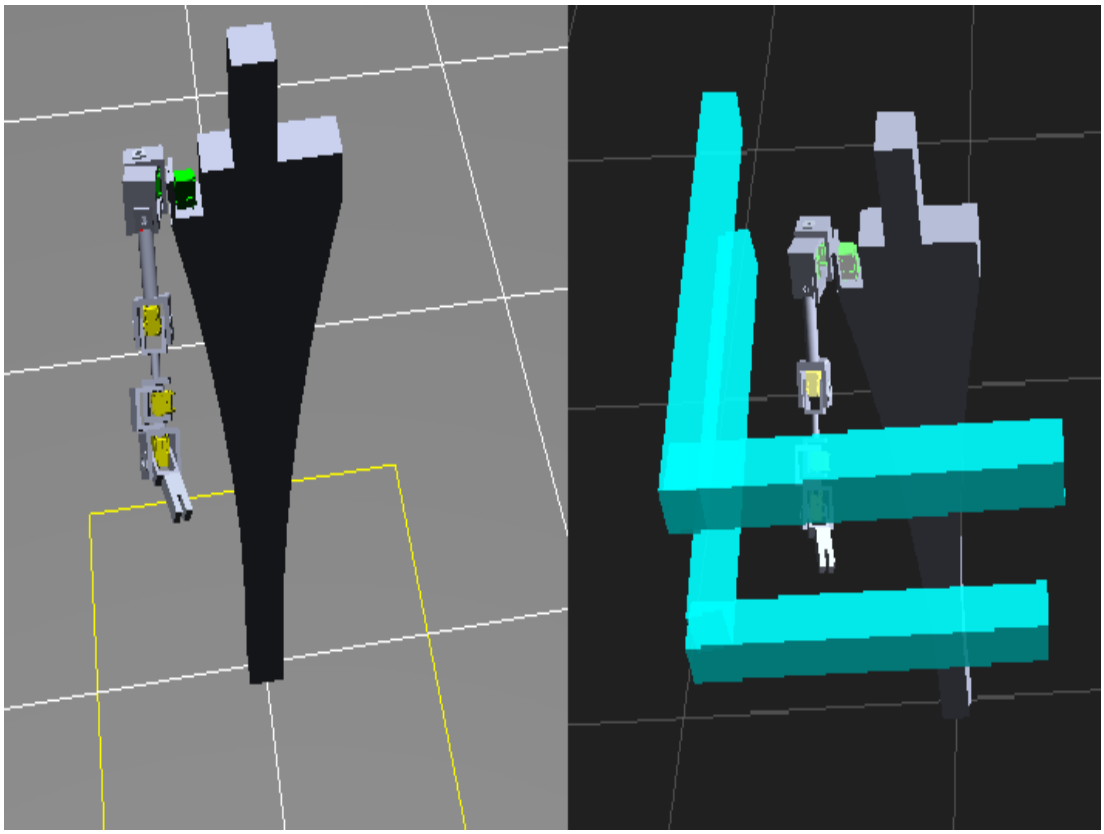


Şekil 4.17 : Birinci yapılandırma için omuzdaki eklemlerin tork grafiği.



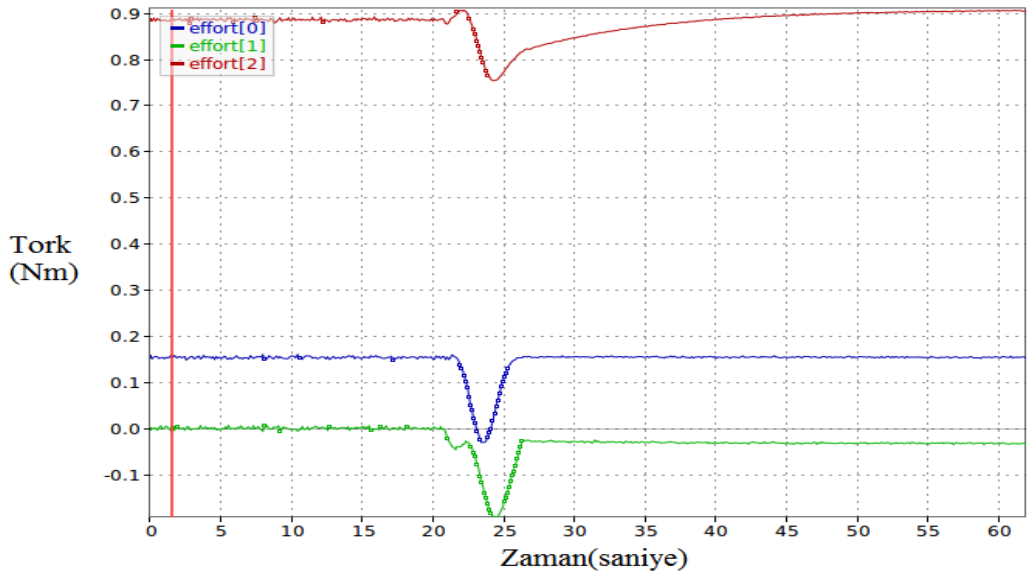
Şekil 4.18 : Birinci yapılandırma için omuzdaki eklemlerin konum grafiği.

İkinci hareket planlama yapılandırmasında yine istenilen yapılandırma Şekil 4.19'da sağda, Gazebo'daki kolun yapılandırması solda verilmiştir.

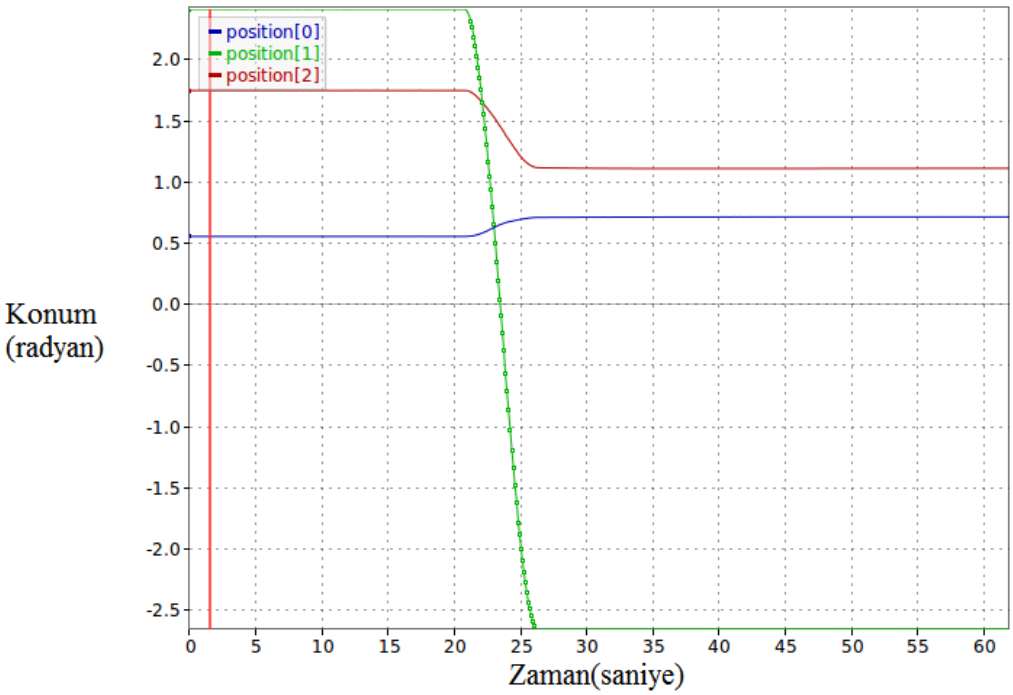


Şekil 4.19 : İkinci yapılandırmadan sonra eklemlerin konumları.

İkinci yapılandırmadan sonra 0, 1 ve 2 numaralı eklemlerin tork grafiği Şekil 4.19'da, konumlarının grafiği ise Şekil 4.20'de verilmiştir.

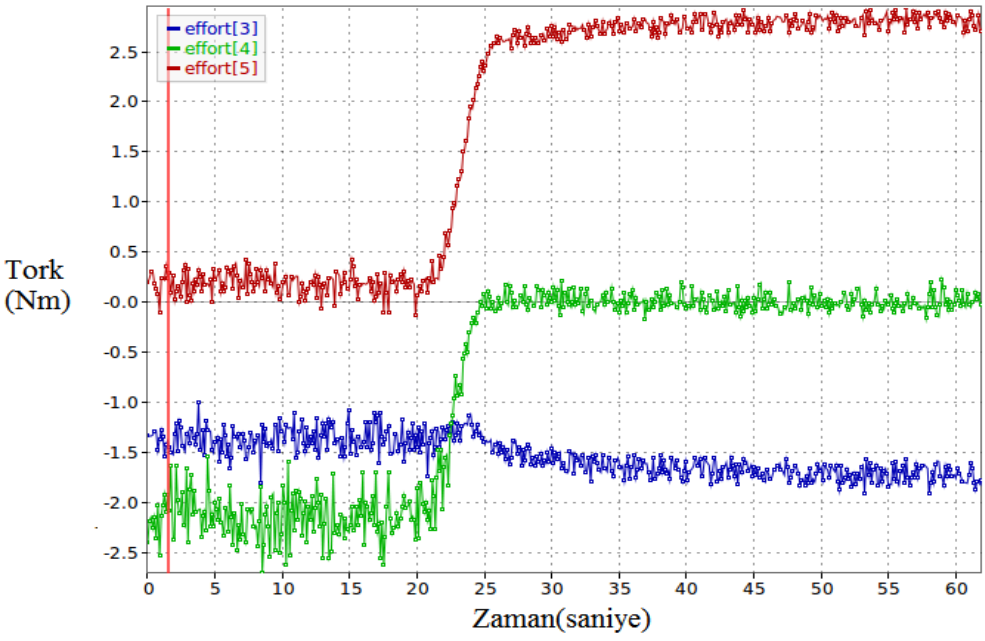


Şekil 4.20 : İkinci yapılandırmadan için dirsekteki ve bilekteki eklemlerin tork grafiği.

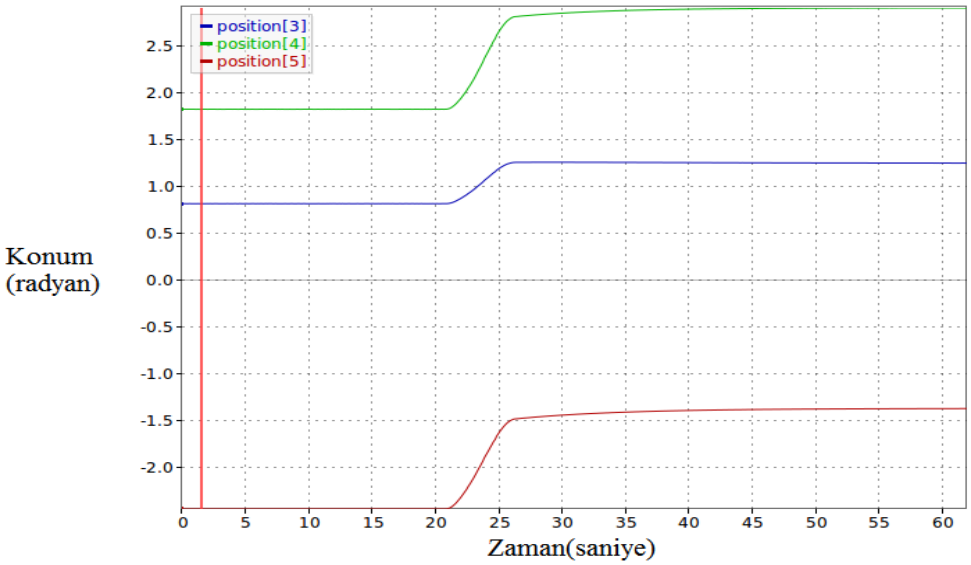


Şekil 4.21 : İkinci yapılandırmadan için dirsekteki ve bilekteki eklemlerin konum grafiği.

İkinci yapılandırmada 3, 4 ve 5 numaralı eklemlerin üzerine binen torkların grafiği Şekil 4.22'de, eklemlerin konum grafiği ise Şekil 4.23'te verilmiştir.

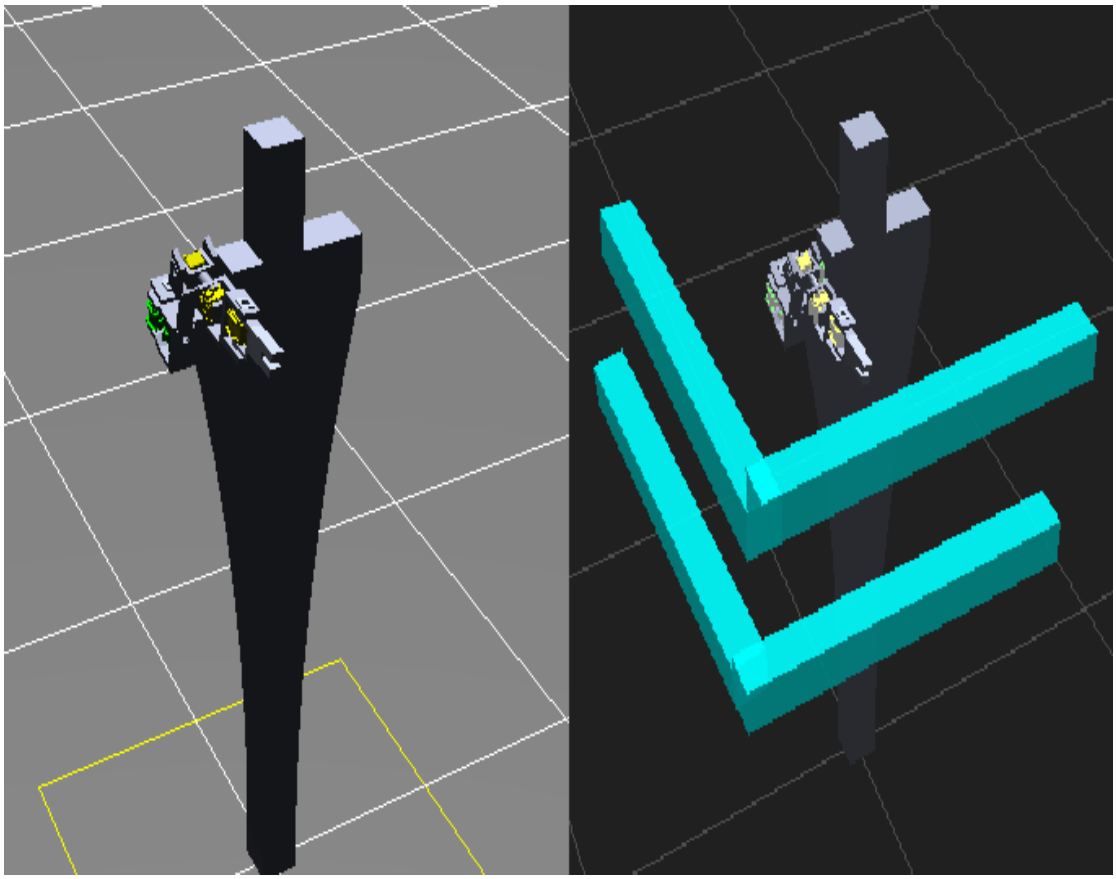


Şekil 4.22 : İkinci yapılandırma için omuzdaki eklemlerin tork grafiği.



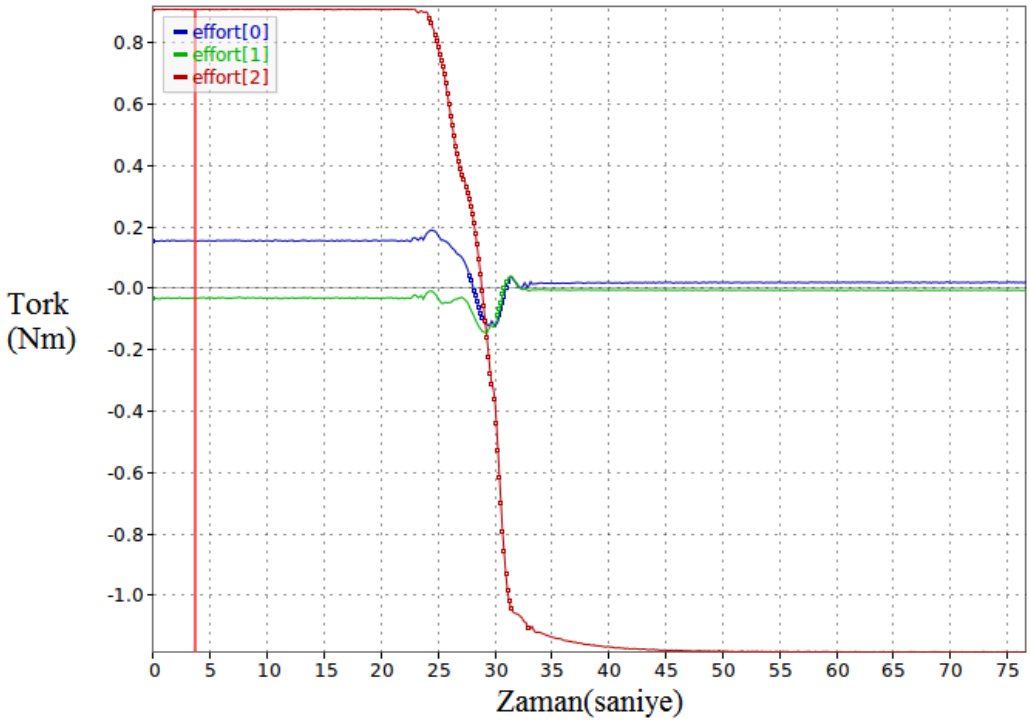
Şekil 4.23 : İkinci yapılandırma için omuzdaki eklemlerin konum grafiği.

Üçüncü yapılandırmadan sonra UMAY'ın kolunun Gazebo'daki ve *rviz*'deki görüntüleri şekil 4.24'te verilmiştir

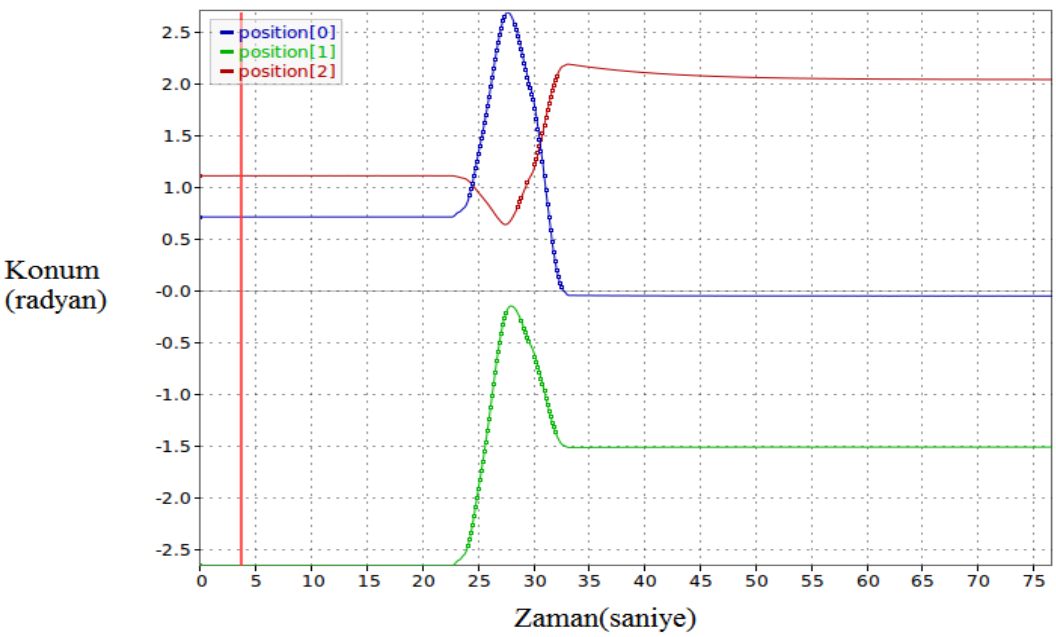


Şekil 4.24 : Üçüncü yapılandırmadan sonra eklemlerin konumları.

Üçüncü yapılandırmadan sonra dirsekteki ve bilekteki eklemlerin üzerine binen torkların grafiği Şekil 4.25'te, eklemlerin konumlarının grafiği Şekil 4.26'da verilmiştir.

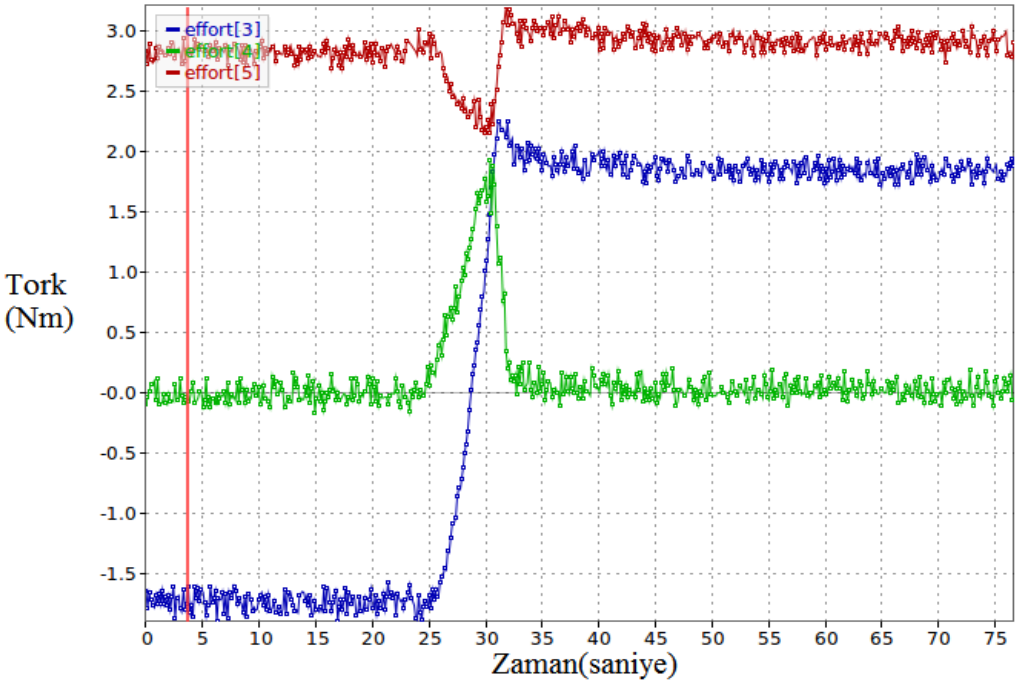


Şekil 4.25 : Üçüncü yapılandırmadan sonra dirsekteki ve bilekteki eklemlerin tork grafiği.

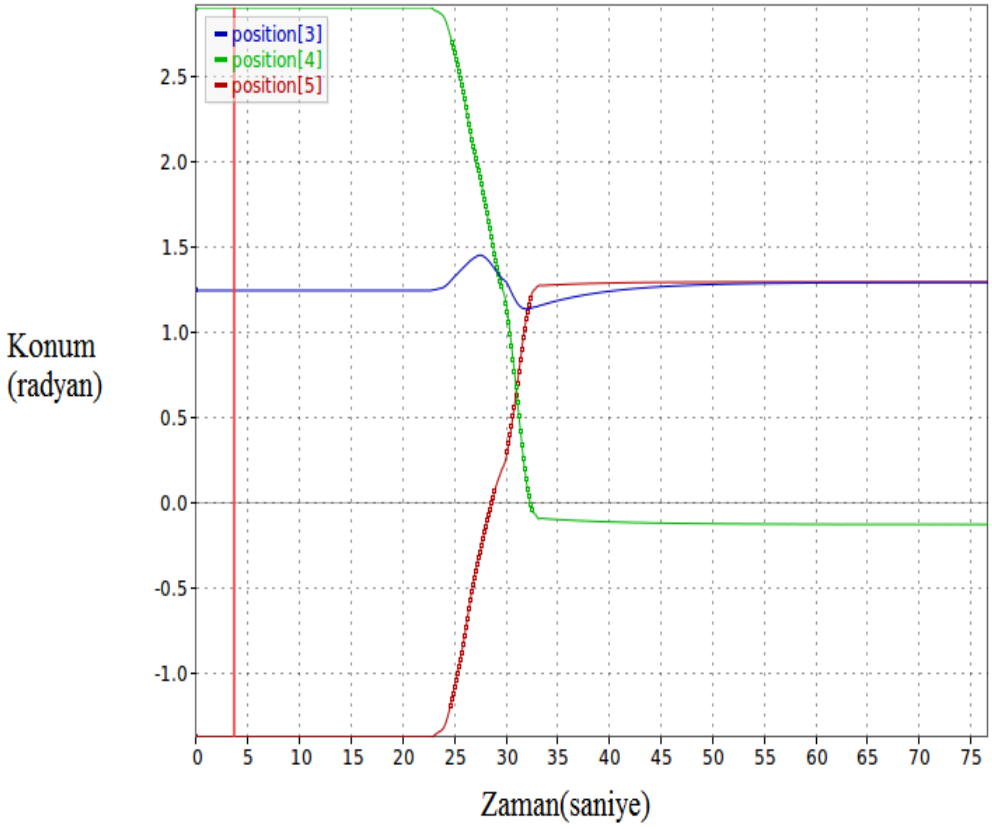


Şekil 4.26 : Üçüncü yapılandırmadan sonra dirsekteki ve bilekteki eklemlerin konum grafiği.

Omuzlardaki eklemlere binen torkların grafiği Şekil 4.27'de, eklemlerin konumlarının grafiği ise Şekil 4.28'de verilmiştir.



Şekil 4.27 : Üçüncü yapılandırma için omuzdaki eklemlere binen torkların grafiği.



Şekil 4.28 : Üçüncü yapılandırma için omuzdaki eklemlerin konum grafiği.

4.2 Kinect İle Etkileşim Uygulaması

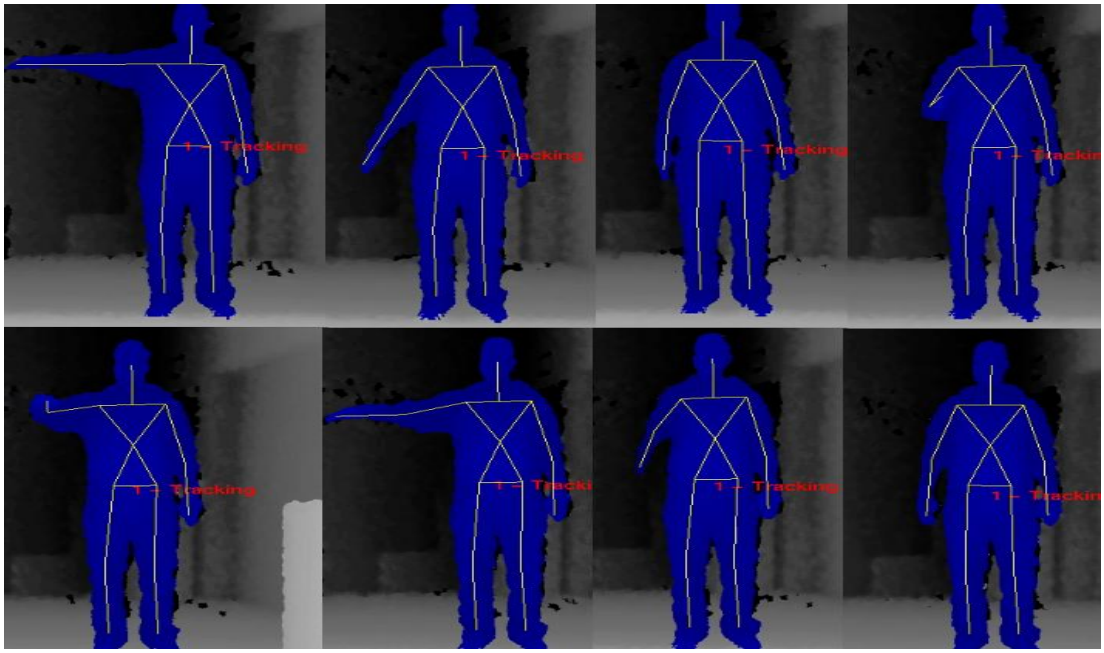
Kinect ile UMAY'ın etkileşime geçebilmesi için ilk önce OpenNI kütüphanesinin çalıştırılması gereklidir. OpenNI kütüphanesinin yüklenmesinden sonra aşağıdaki komut, komut satırına yazılarak Kinect'in sürücüleri sistem tarafından görülür.

roslaunch openni_launch openni.launch **(4.5)**

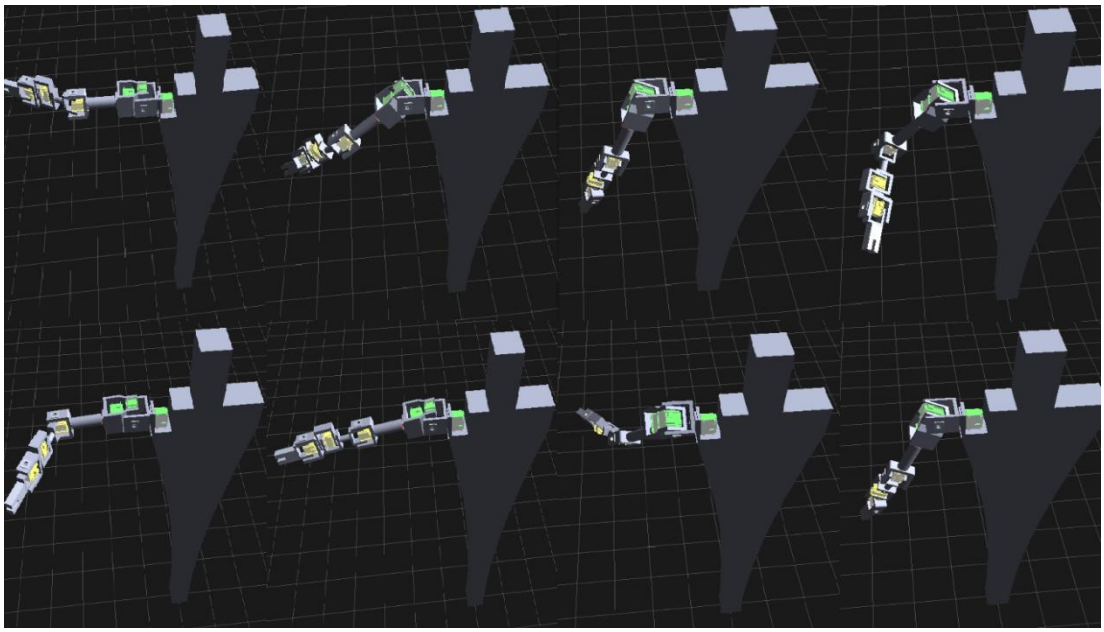
Yukaridaki komutta geçen *openni_launch* paketi sürücülerin tutulduğu pakettir, *openni.launch* dosyası Kinect'in doğru çalışabilmesi için gerekli ROS düğümlerini çalıştırmaktadır. Bu komuttan hemen sonra, OpenNI'ın sağladığı iskelet takibi programını çalıştırmak için UMAY'ın iskelet takibi için derlenmiş paketindeki bir kaç dosyanın daha çalıştırılması gerekmektedir.

roslaunch umay_tracker skeleton_tracker.launch **(4.6)**

Bu komut, komut satırına yazıldıktan sonra iskelet takibi programı açılacaktır. Bu ekranda kullanıcı, kendini Kinect ile kalibre ettirdikten sonra üzerine oturtturulan iskelet sayesinde kullanıcının hareketleri izlenmeye başlanır. Kullanıcının eklem hareketleri, dönüşüm paketinin sağladığı *tf* başlığı altında ROS'da yayınlanır ; bu paketten gelen kullanıcının sol kolundaki eklem verileri UMAY'ın koluna aktarılır. Şekil 4.29'da kullanıcının hareketleri ve Şekil 4.30'da UMAY'ın kolunun hareketleri verilmiştir.

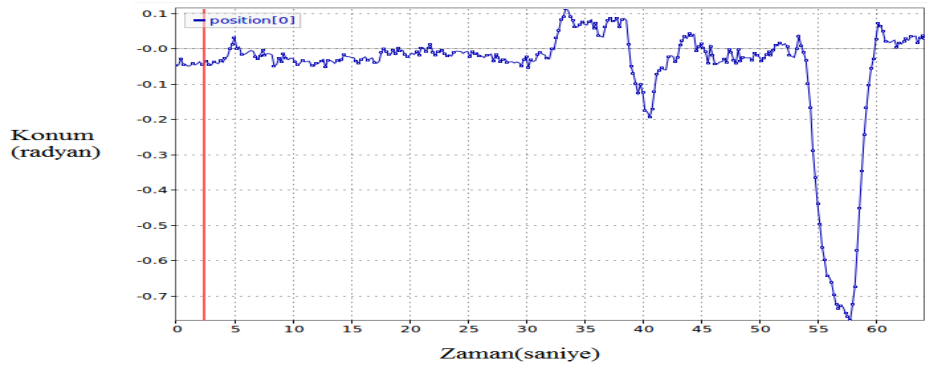


Şekil 4.29 : Kullanıcı hareketleri.

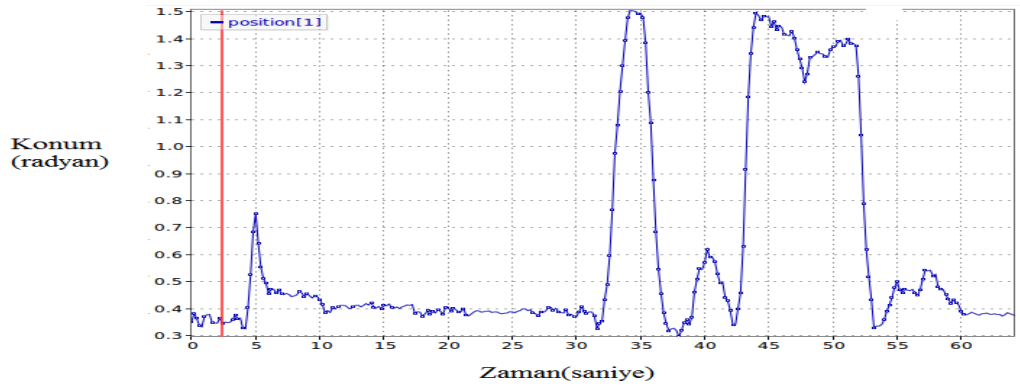


Şekil 4.30 : UMAY'ın kullanıcıyı takliti.

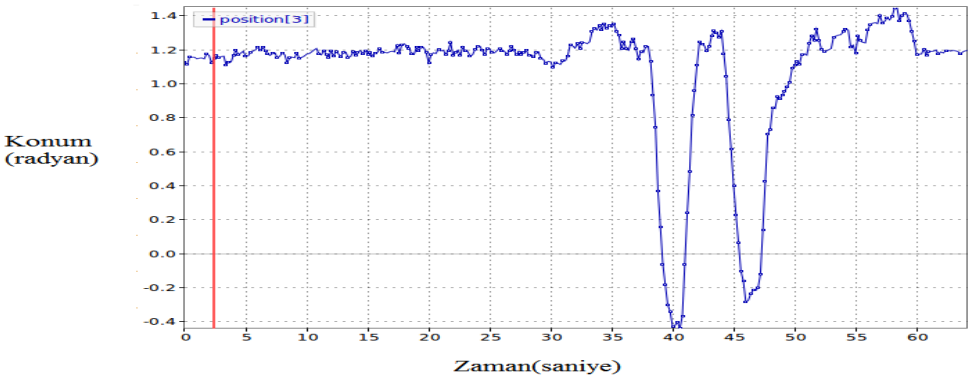
Eklem 0 ve 1 omuzlara ait olan eklemler ve eklem 3 dirsekteki eklem olmak üzere, UMAY'ın kolunun eklem konumlarının grafikleri Şekil 4.31, Şekil 4.32 ve Şekil 4.33'te verilmiştir.



Şekil 4.31 : Eklem numarası 0 için konum grafiği.



Şekil 4.32 : Eklem numarası 1 için konum grafiği.



Şekil 4.33 : Eklem numarası 3 için konum grafiği.

5. SONUÇ

Yapılan bu çalışmada, bir servis robotunun kontrol ve veri iletişimi altyapısı oluşturulmuştur. Servis robotunun koluna binen torklar simülasyon ortamında test edilip, seçilen eyleyicilerin gereksinimleri karşıladığı tespit edilmiştir. Bu altyapıları oluşturmak için bir çok kütüphane ve program test edilmiştir.

Simülasyon programı olan Gazebo'nun kullanımı yeni başlayanlar için zorlayıcı olacaktır. KDL kütüphanesinin kullanımı ise ters kinematik problemini çözmekte büyük rahatlık sağlamıştır, Gazebo'ya nispeten programlanması ve kullanılması kolaydır.

OpenNI kütüphanesinin kullanımı ve kurulumu kolay olmasına rağmen bu kütüphaneyi kullanarak ROS ve Kinect üzerinde yazılım geliştirmenin kolay olmadığı tespit edilmiştir.

ROS işletim sisteminin kolay kuruluma sahip olması bir avantaj sağlasa bile ROS ile üzerinde çalıştığı Linux İşletim Sistemi arasında çıkabilecek herhangi bir sorunu çözmek için ileri seviyede Linux İşletim Sistemi bilgisi gerekmektedir. ROS'un çok fazla eğitim kaynağı olsa da, bir servis robotunu ROS'a tamamiyle adapte etmek için birden fazla insan ve ileri düzeyde bir ROS altyapı bilgisine sahip olmak gerekmektedir.

Yörünge süzgeçlerinin kullanımı ve uygulanması çalışmanın en kolay kısmını oluşturmaktadır.

OMPL Kütüphanesinin kullanımı kolay gözükse bile bu kütüphanenin ROS ile birlikte çalışması için bir çok araç kullanılmaktadır bu yüzden bu kütüphanenin düzgün kullanılması için geliştiricinin ileri seviyede ROS bilgisine sahip olması gerekmektedir.

Başlangıç seviyesinde bulunanlar için kullanıma ve yazılım kolaylığına göre kullanılan sistemlerin çizelgeleri, Çizelge 5.1 ve Çizel 5.2 'de verilmiştir.

Çizelge 5.1 : Yazılım altyapısının kullanılım kolaylığı

Kullanım Kolaylığı	**Zor**	**Orta**	**Kolay**
ROS		**X**	
Gazebo	**X**		
KDL			**X**
OMPL	**X**		
OpenNI			**X**

Çizelge 5.2'de ise yazılım altyapısını programlamakta karşılaşılan zorlukların değerlendirilmesi yapılmıştır.

Çizelge 5.2 : Yazılım altyapısının programlanma kolaylığı

Programlama Kolaylığı	**Zor**	**Orta**	**Kolay**
ROS		**X**	
Gazebo	**X**		
KDL			**X**
OMPL	**X**		
OpenNI		**X**	

KAYNAKLAR

[1] **Kerr, J., Nickels, K.** (2012).Robot Operating Systems: Bridging the Gap between Human and Robot, *44th International Symposium on System Theory*. Jacksonville,Florida, ABD, Sf.99-104, 11-13 Mart.

[2] **Cousins, S.** (2010). Welcome to ROS Topics, *IEEE,Robotics and Automation Magazine,* Cilt ***17***, Sayı 1, Sf.13-14.

[3] **Cousins, S.** (2012). Is ROS Good for Robotics, *IEEE,Robotics and Automation Magazine,* Cilt ***19***, Sayı 2, Sf.13-14.

[4] **Url-1** <*http://www.osrfoundation.org/blog/getting-ready-for-the-darpa-robotics-challenge.html*>, alındığı tarih 14.11.2012.

[5] **Url-2** <*http://www.willowgarage.com/blog/2011/05/12/google-io-2011-cloud-robotics-ros-java-and-android*>, alındığı tarih 15.12.2012.

[6] **Badger, Julia M., Hart, Stephen W., Yamokoski, J. D** (2011). Towards Autonomous Operation of Robonaut 2, *NASA, AIAA Infotech@Aerospace,* Garden Grove, CA, ABD, 21 Aralık.

[7] **Url-3** <*http://www.ros.org/news*>, alındığı tarih 14.12.2012.

[8] **Cousins, S.** (2010). ROS on the PR2, *IEEE,Robotics and Automation Magazine,* Cilt ***17***, Sayı 3, Sf.23-15.

[9] **Gerkey, B., Conley, K.** (2011).Robot Developer Kits, *IEEE,Robotics and Automation Magazine,* Cilt ***18***, Sayı 3, Sf.16.

[10] **Edwars, S.M.** (2012). Ros-Industrial – Accelerating Research To Applications, *The Robotics Industry Forum,* Orlando, Florida, ABD, 20 Ocak.

[11] **Edwars, S.M.** (2011). Leveraging the Open Source Robot Operating System (ROS) for New Industrial Applications, *The Robotic Summit, Robotic Trends Virtual Conference Series*, 14 Ekim.

[12] **Url-4** <*rosindustrial.org/about-ros-i.htm*>, alındığı tarih 14.12.2012.

[13] **Zaman, S., Slany, W., Steinbauer, G.** (2011). ROS-based mapping, localization and autonomous navigation using a Pioneer 3-DX robot and their relevant issues, *2011 Saudi International, Electronics, Communication and Photonics Conference,* Sf. 1-5, 24-26 Nisan.

[14] **DeMarco, K., West, M.E., Collins, T.R.** (2011). An implementation of ROS on the Yellowfin autonomus underwater vehicle(AUV), *OCEANS 2011*, Sf. 1-7, 19-21 Eylül.

[15] **Gong, S., Liu, H., Zhang, J.** (2012). Robot Operating Systems: ROS-based object localization using RFID and laser scan, *International*

Conference on Information and Automation. Shenzen, Çin, Sf.406-411. 6-8 Haziran.

[16] **Cunha, J., Azevedo, J.L., Cunha, M. B., Fonseca, P., Lau, N., Martins, C., Moura, C., Neves, A. J. R., Pedrosa, E., Pereira A., Teixeira, A. J. S.** (2012). *CAMDA'2012 Team Description Paper* (çevrimiçi makale). Adres: http://www.ieeta.pt/atri/cambada/robocup2012.htm

[17] **Bharatheesha, M., Rudinac, M., Chandarr, A., Gaisser, F., Rueda, S.P., Küpers, B., Driessen, S., Bruinnik, M., Wisse, M., Jonker P.** (2012). *Delft Robotics Robocup@Home 2012 Team Description Paper* (çevrimiçi makale). Adres: http://delftrobotics.nl/content/team-description-paper

[18] **Vargas, H.S., Olmedo, E., Martínez, A.D., López, M.L.M., Medina, E., Perez,J.L.,**ve diğ (2012). *Donaxi@HomeProject*(çevrimiçi makale). Adres: http://wla.orgfree.com/prueba/english/pagina1.php

[19] **Luing, V., Sarnecki, L., Grün, T., Knauf, Barthen, A., Syre , L.,** ve diğ (2012). *Robocup 2012 – homer@UniKoblenz* (çevrimiçi makale). Adres: http://robots.uni-koblenz.de/

[20] **Stückler, J., Droeschel, D., Grave, K.,** ve diğ (2012). *Nimbro@Home 2012 Team Description*(çevrimiçi makale). Adres: http://www.nimbro.net

[21] **Werber, K.** (2011). *Intuitive Human Robot Interaction and Workspace Surveillance by means of The Kinect Sensor* (yüksek lisans tezi). Adres: http://www.lunduniversity.lu.se

[22] **Campana, R.** (2011). *People Detection and Tracking with Kinect for Mobile Platforms* (yüksek lisans tezi). Adres: http://tesi.cab.unipd.it

[23] **Mojtahedzadeh, R.** (2011).*Robot Obstacle Avoidance using the Kinect* (yüksek lisans tezi). Adres: http://kth.diva-portal.org

[24] **Tolgyessy, M. ve Hubinsky, P.**(2011).The Kinect Sensor in Robotics Education.*In Proceedings of 2nd International Conference on Robotics in Education,* Viyana, Avusturya, Sf: 143-146.

[25]**MINI-ITX.** (t.y.).*Wikipedia.* Alındığı tarih: 15.04.2012, Adres: http://en.wikipedia.org/wiki/Mini-ITX

[26] **Url-5** <*http://www.ros.org/wiki/ROS/Concepts*>, alındığı tarih 3.12.2012.

[27] **Url-6** <*http://www.ros.org/doc/api/sensor_msgs/html/msg/LaserScan.html*>, alındığı tarih 3.12.2012.

[28] **Url-7** < *http://www.orocos.org/kdl*>, alındığı tarih 5.12.2012.

[29] **Quigley,M.,Conley, K.,Gerkey,B. P.,Faust, J.,Foote, T.,Leibs, J.,Wheeler, R.,** ve diğ (2009).ROS: an open-source Robot Operating System. *ICRA Workshop on Open Source Software,* Kobe, Japonya, 12-17 Mayıs.

[30] **Url-8** <*http://www.ros.or/*>, alındığı tarih 30.04.2012.

[31] **Url-9** <*http://www.openni.org*> alındığı tarih 30.04.2012.

[32] **Url-10** <*http://ros.org/wiki/tf*> alındığı tarih 3.12.2012.

[33] **Url-11** <*http://ros.org/wiki/tf/Tutorials*> alındığı tarih 3.12.2012.

[34] **Url-12** <*http://ros.org/wiki/rviz*> alındığı tarih 3.12.2012.

[35] **Url-13** <http://www.ros.org /wiki/urdf/ Tutorials/ Understanding PR2URDFAdvanced> alındığı tarih 4.12.2012.

[36] **Url-14** < *http://www.ros.org/wiki/urdf/Tutorials*> alındığı tarih 4.12.2012.

[37] **Url-15 <***http://www.ros.org/wiki/arm_navigation/Tutorials/tools/Planning Description Configuration Wizard*> alındığı tarih 4.12.2012.

[38] **Url-16 <***http://www.ros.org/wiki/arm_navigation/Tutorials/tools/Warehouse Viewer***>** alındığı tarih 4.12.2012.

[39] **Url-17** <*http://gazebosim.org*> alındığı tarih 5.12.2012.

[40] **Şucan, A.I., Moll, Mark., Karvaki, L.E.** (baskıda). The Open Motion Planning Library, *IEEE,Robotics and Automation Magazine.*

[41] **Bruyninckx, H., Soetens, P.** (2007). *The Orocos Project* (çevrimiçi makale). Adres: http://www.orocos.org

[42] **Url-18** <*http://ros.org/wiki/pr2_mechanism*> alındığı tarih 1.12.2012.

[43] **Url-19** <*http://ompl.kavrakilab.org*> alındığı tarih 1.12.2012.

[44] **Url-20 <***http://ros.org/wiki/pr2_controller_interface*> alındığı tarih 1.12.2012.

[45] **Url-21 <** *http://www.ros.org/wiki/robot_mechanism_controllers*> alındığı tarih 1.12.2012.

[46] **Url-22 <***http://www.trossenrobotics.com/dynamixel-ex-106-robot-actuator.aspx*> alındığı tarih 2.12.2012.

[47] **Url-23 <***http://www.crustcrawler.com/motors/RX28/index.php?prod=66*> alındığı tarih 2.12.2012.

[48] **Url-24 <***http://www.ros.org/wiki/simmechanics_to_urdf*> alındığı tarih 2.12.2012.

[49] **Murray, R.M., Li, Z., Sastry, S.S.** (1994). A Mathematical Introduction To Robotic Manipulation, Second Edition, CRC, PressINC, ABD

[50] **Spong, M.W., Hutchinson, S., Vidyasagar, M.** (2005). Robot Modeling and Control, Wiley, ABD

[51] **Url-25 <***http://people.mech.kuleuven.be/~rsmits/kdl/api/*> alındığı tarih 2.12.2012.

[52] **Tesch, J.** (2005). The Newton Raphson Method and its Application to Fixed Points(ders notu). Alındığı tarih 3.12.2012 Adres: http://chess.eecs.berkeley.edu/bipeds/documents/Newton.pdf

[53] **Stone, H.W., Chinloy, P.** (1987). Kinematic Modeling, Identification and Control of Robotic Manipulators, Springer-Verlag, Newyork, LLC, ABD.

[54] **Sanchez, G., Latombe, J.C.** (2001). A Single-query Bi-directional Probablistic Roadmap planner with Lazy Collision Checking, *The 10th International Symposium on Robotics Research.* Sf.403-417.

[55] **Küçük, S., Bingül, Z.** (2008). Robot Dinamiği ve Kontrolü, Birsen Yayınevi / Mühendislik Dizisi, İstanbul, Türkiye

[56] **Url-24 <***http://www.ros.org/wiki/ROS/Installation*> alındığı tarih 9.12.2012.

[57] **Url-25 <**http://www.ros.org/wiki/arm_navigation/Tutorials/tools/Planning

Descriptiob Configuration *Wizard*> alındığı tarih 9.12.2012.

[58] **Sachin, C., Jones, E.G., Sucan, I.A,** ve diğ (2011). Motion Planing for Real Robots.(eğitim notu). *IEEE/RSJ International Conference on Intelligent Robots and Systems.* San Fransisco, California, ABD, 25-30 Ekim

[59] **Url-26 <***http://www.ros.org/wiki/arm_navigation/Tutorials*> alındığı tarih 9.12.2012.

EKLER

Ek A : Yazılım kaynak kodları
Ek A1 : Ters kinematik kodu
Ek A2 : Kinect takip kodu
Ek A3 : Kübik eğri yörünge kodu
Ek A4 : Ters kinematik servis kodu

EK A1

```
#include <kdl_parser/kdl_parser.hpp>
#include <ros/ros.h>
#include <kdl/chain.hpp>
#include <boost/scoped_ptr.hpp>
#include <sensor_msgs/JointState.h>
#include <kdl/chainfksolver.hpp>
#include <kdl/chainfksolverpos_recursive.hpp>
#include <kdl/chainiksolvervel_pinv.hpp>
#include <kdl/chainiksolverpos_nr_jl.hpp>
#include <kdl/chainiksolverpos_nr.hpp>
#include <kdl/frames_io.hpp>
class KDLInverseKinematics {
private:
  ros::NodeHandle n_priv;
  KDL::Tree my_tree;
  KDL::Chain chain;
  bool exit_value;
  KDL::JntArray qMin;
  KDL::JntArray qMax;
  KDL::JntArray velMin;
  KDL::JntArray velMax;
  KDL::JntArray q_init;
  KDL::JntArray q;
  sensor_msgs::JointState   LeftShoulder;
  ros::Publisher jointStatePublisher;
  ros::Rate *loop_rate;
  KDL::Frame destFrame;
  boost::scoped_ptr<KDL::ChainIkSolverPos_NR_JL> iksolverpos;
  boost::scoped_ptr<KDL::ChainFkSolverPos_recursive> fksolver;
  boost::scoped_ptr<KDL::ChainIkSolverVel_pinv> iksolvervel;
  const double  x_dot_trans_max;
  const double  x_dot_rot_max;
  const double  x_dot_trans_min;
  const double  x_dot_rot_min;
```

```
  const double  low_pass;
  const int iter;
  const double eps;
  double x,y,z;
  double roll,pitch,yaw;
public:
  KDLInverseKinematics(double x,double y,double z,
                       double roll,double pitch,double yaw,
                       double eps=1e-6,double iter = 100):
                        x_dot_trans_max(1.0),x_dot_rot_max(1.0),
                        x_dot_trans_min(0.0),x_dot_rot_min(0.0),
                        low_pass(0), iter(iter), eps(eps)
  {


                       if(!kdl_parser::treeFromFile("/home/cagatay/ros_workspace/murtaza
                       _tracker/src/UmayKol.urdf.xacro", my_tree))
    {
      ROS_ERROR("Failed to construct kdl tree");
    }
    this->x = x;
    this->y = y;
    this->z = z;
    this->roll = roll;
    this->pitch = pitch;
    this->yaw = yaw;
    exit_value = my_tree.getChain("Body","Yedi",chain);
    if(exit_value) {
      ROS_INFO("chain is get");
    } else {
      ROS_ERROR("unable to get chain");
    }
   qMin.resize(chain.getNrOfJoints());
   qMax.resize(chain.getNrOfJoints());
    velMin.resize(chain.getNrOfJoints());
    velMax.resize(chain.getNrOfJoints());
```

```
 q_init.resize(chain.getNrOfJoints());
for(unsigned int i = 0 ;  i < chain.getNrOfJoints() ; i++)
 {
  velMin(i) = 0.0;
  velMax(i) = 1.0;
  qMin(i) = -M_PI;
  qMax(i) = M_PI;
  q_init(i) = 0.0;
 }
 qMax(3) = M_PI/3;
 qMax(4) = M_PI/4;
 fksolver.reset(new KDL::ChainFkSolverPos_recursive(chain));
 iksolvervel.reset(new KDL::ChainIkSolverVel_pinv(chain));
iksolverpos.reset(new KDL::ChainIkSolverPos_NR_JL(chain,
                                        qMin,
                                        qMax,
                                        *fksolver,
                                        *iksolvervel,
                                        iter,
                                        eps));
  destFrame.p = KDL::Vector(this->x,this->y,this->z);
 destFrame.M = KDL::Rotation::RPY(this->roll,this->pitch,this->yaw);
 jointStatePublisher =
                n_priv.advertise<sensor_msgs::JointState>("/joint_states",1);
  LeftShoulder.position.resize(chain.getNrOfJoints());
 LeftShoulder.velocity.resize(chain.getNrOfJoints());
 LeftShoulder.name.push_back("BirikiJoint");
 LeftShoulder.name.push_back("iKiUcJoint");
 LeftShoulder.name.push_back("UcDortJoint");
 LeftShoulder.name.push_back("DortBesJoint");
 LeftShoulder.name.push_back("BesAltiJoint");
 LeftShoulder.name.push_back("AltiYediJoint");
 for(unsigned int i = 0 ; i < LeftShoulder.name.size() ; i++ )
 {
  LeftShoulder.position[i] = 0.0;
```

```
LeftShoulder.velocity[i] = 0.0;
    q_init(i) = 0.0;
   }
 loop_rate = new ros::Rate(1);
int ret;
 int k=0;
 ros::Time begin = ros::Time::now();
 while((ret = iksolverpos->CartToJnt(q_init,destFrame,q)) < 0 )
  {
    k++;
    q_init.data.setRandom();
  }
 ros::Time end = ros::Time::now();
  ROS_INFO("solution found: %d, iterated: %d times and %0.3f secs",ret,k,(end-
begin).toSec());
  ROS_INFO("Pos(1): %0.3f , Pos(2): %0.3f, Pos(3): %0.3f , Pos(4): %0.3f ,
                Pos(5): %0.3f , Pos(6): %0.3f",q_init(0),
          q_init(1),
          q_init(2),
          q_init(3),
          q_init(4),
          q_init(5));
  ROS_INFO("q(1): %0.3f , q(2): %0.3f, q(3): %0.3f , q(4): %0.3f , q(5): %0.3f ,
                q(6): %0.3f",q(0),
          q(1),
          q(2),
          q(3),
          q(4),
          q(5));
  for(unsigned int i = 0 ; i < chain.getNrOfJoints() ; i++)
    LeftShoulder.position[i]= q(i);
 }
 void spin(ros::NodeHandle & nh)
 {
  unsigned int k = 0x0;
  while(nh.ok())
```

```
   {
  LeftShoulder.header.stamp = ros::Time::now();
     if(k) {
       for(unsigned int i = 0 ; i < chain.getNrOfJoints() ; i++)
         LeftShoulder.position[i]= q(i);
    } else {
       LeftShoulder.position[0]= 0.0;
       LeftShoulder.position[1]= 0.0;
       LeftShoulder.position[2]= 0.0;
       LeftShoulder.position[3]= 0.0;
       LeftShoulder.position[4]= 0.0;
       LeftShoulder.position[5]= 0.0;
}
     jointStatePublisher.publish(LeftShoulder);
     k = ~k;
     ros::spinOnce();
     loop_rate->sleep();
   }
  }
  ~KDLInverseKinematics() { jointStatePublisher.shutdown(); }
};
int main(int argc, char **argv)
{
ros::init(argc, argv, "invknm");
  ros::NodeHandle n;
  KDLInverseKinematics k(-0.178, 0.375, 1.625,-0.117, 0.006, 3.021);
  k.spin(n);
  return(0);
}
```

EK A2

```
#include <ros/ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Header.h>
#include <murtaza_tracker/Skeleton.h>
#include <geometry_msgs/Vector3.h>
#include <sensor_msgs/JointState.h>
#include <tf/transform_broadcaster.h>
#include <tf/transform_listener.h>
#include <kdl/frames.hpp>
#include <boost/scoped_ptr.hpp>
#include <string>
#include <cmath>
#include <tf/tf.h>
#define HEAD          0
#define NECK          1
#define TORSO         2
#define LEFT_SHOULDER  3
#define LEFT_ELBOW     4
#define LEFT_HAND      5
#define LEFT_FOOT     11
class Commander {
private:
  ros::Subscriber skeletonSub;
  ros::Publisher  cmdPub;
  ros::Publisher  cmdTfPub;
  ros::Publisher  jointStatePublisher;
  ros::NodeHandle nh;
  murtaza_tracker::Skeleton skeleton;
  murtaza_tracker::Skeleton preSkeleton;
  sensor_msgs::JointState   LeftShoulder;
  KDL::Rotation rotLeftHand;
  KDL::Vector   transLeftHand;
  ros::Rate *loop_rate;
  tf::TransformListener listener;
```

```
 geometry_msgs::TransformStamped msg;
 geometry_msgs::TransformStamped msg1;
 tf::StampedTransform transform;
 tf::StampedTransform transform1;
 tf::Vector3 elbow;
 tf::Vector3 shoulder;
 tf::Vector3 ShoulderNormalize;
 tf::Vector3 hand;
 tf::Vector3 HandNormalize;
 double norm;
void teleop_joints()
 {
  LeftShoulder.header.stamp = ros::Time::now();
   tf::Vector3 elbow; tf::Vector3 shoulder; tf::Vector3 ShoulderNormalize;
   tf::Vector3 hand;  tf::Vector3 HandNormalize;
   tf::Vector3 left_foot;
   tf::vector3MsgToTF(skeleton.position[LEFT_ELBOW],elbow);
   tf::vector3MsgToTF(skeleton.position[LEFT_SHOULDER],shoulder);
   tf::vector3MsgToTF(skeleton.position[LEFT_HAND],hand);
   tf::vector3MsgToTF(skeleton.position[LEFT_FOOT],left_foot);
   ShoulderNormalize = elbow - shoulder;
   HandNormalize = hand - elbow;
   ShoulderNormalize.normalize();
   HandNormalize.normalize();
   LeftShoulder.position[1] = -(asin(ShoulderNormalize.getX()));
   LeftShoulder.position[0] = -(asin(ShoulderNormalize.getZ()));
   KDL::Vector
                v1(ShoulderNormalize.getX(),ShoulderNormalize.getY(),ShoulderNor
                malize.getZ());
   KDL::Vector
                v2(HandNormalize.getX(),HandNormalize.getY(),HandNormalize.get
                Z());
   LeftShoulder.position[3] =  ((M_PI/2)-acos(KDL::dot(v1,v2)));
   LeftShoulder.position[4] =   asin(HandNormalize.getY());
   jointStatePublisher.publish(LeftShoulder);
 }
```

```
public:
  Commander()
  {
                skeletonSub=nh.subscribe<murtaza_tracker::Skeleton>("/skeleton",1
                0,&Commander::skeletonCallback,this);
    cmdPub=nh.advertise<geometry_msgs::Twist>("/cmd_vel",10);

                cmdTfPub=nh.advertise<geometry_msgs::TransformStamped>("/righ
                t_hand",10);
    jointStatePublisher = nh.advertise<sensor_msgs::JointState>("/joint_states",1);
    LeftShoulder.name.resize(6);
    LeftShoulder.position.resize(6);
    LeftShoulder.name[0] = "BirikiJoint";
    LeftShoulder.name[1] = "iKiUcJoint";
    LeftShoulder.name[2] = "UcDortJoint";
    LeftShoulder.name[3] = "DortBesJoint";
    LeftShoulder.name[4] = "BesAltiJoint";
    LeftShoulder.name[5] = "AltiYediJoint";
    for(unsigned int i = 0 ; i < LeftShoulder.name.size() ; i++ )
      LeftShoulder.position[i] = 0.0;
    loop_rate = new ros::Rate(5);
  }
 virtual ~Commander()  {
    skeletonSub.shutdown();
    cmdPub.shutdown();
  }
  void skeletonCallback(const murtaza_tracker::Skeleton::ConstPtr& msg)
  {
    skeleton = *msg;
  }
  void skeletonLoop()
  {
    while(nh.ok()) {
      try{
        listener.lookupTransform("/camera_link", "/left_shoulder",
                                 ros::Time(0), transform);
```

```
      teleop_joints();
    }
    catch (tf::TransformException ex)
    {
      ROS_WARN("%s",ex.what());
    }
    jointStatePublisher.publish(LeftShoulder);
    ros::spinOnce();
    loop_rate->sleep();
   }
  }
};

int main(int argc, char** argv)
{
  ros::init(argc,argv,"commander");
  Commander c;
  c.skeletonLoop();
  return(0);
}
```

EK A3

```
#include <ros/ros.h>
#include <pr2_controllers_msgs/JointTrajectoryAction.h>
#include <actionlib/client/simple_action_client.h>

typedef actionlib::SimpleActionClient<
                pr2_controllers_msgs::JointTrajectoryAction > TrajClient;

class RobotArm
{
private:
  TrajClient* traj_client_;
public:
  RobotArm()
  {
    traj_client_ = new TrajClient("r_arm_controller/joint_trajectory_action", true);
    while(!traj_client_->waitForServer(ros::Duration(5.0))){
      ROS_INFO("Yörünge sunucusu beklemede");
    }
  }
  ~RobotArm()
  {
    delete traj_client_;
  }
  void startTrajectory(pr2_controllers_msgs::JointTrajectoryGoal goal)
  {
    goal.trajectory.header.stamp = ros::Time::now() + ros::Duration(1.0);
    traj_client_->sendGoal(goal);
  }
  pr2_controllers_msgs::JointTrajectoryGoal armExtensionTrajectory()
  {
    pr2_controllers_msgs::JointTrajectoryGoal goal;
    goal.trajectory.joint_names.push_back("BirikiJoint");
    goal.trajectory.joint_names.push_back("iKiUcJoint");
    goal.trajectory.joint_names.push_back("UcDortJoint");
```

```
 goal.trajectory.joint_names.push_back("DortBesJoint");
 goal.trajectory.joint_names.push_back("BesAltiJoint");
 goal.trajectory.joint_names.push_back("AltiYediJoint");
 goal.trajectory.points.resize(2);
 int ind = 0;
 goal.trajectory.points[ind].positions.resize(6);
 goal.trajectory.points[ind].positions[0] = 0.0;
 goal.trajectory.points[ind].positions[1] = M_PI/2;
 goal.trajectory.points[ind].positions[2] = 0.0;
 goal.trajectory.points[ind].positions[3] = 0.0;
 goal.trajectory.points[ind].positions[4] = 0.0;
 goal.trajectory.points[ind].positions[5] = 0.0;
 goal.trajectory.points[ind].velocities.resize(6);
 for (size_t j = 0; j < 6; ++j)
 {
   goal.trajectory.points[ind].velocities[j] = 0.0;
 }
 goal.trajectory.points[ind].time_from_start = ros::Duration(10.0);
 ind += 1;
 goal.trajectory.points[ind].positions.resize(6);
 goal.trajectory.points[ind].positions[0] =  0.0;
 goal.trajectory.points[ind].positions[1] =  0.0;
 goal.trajectory.points[ind].positions[2] =  0.0;
 goal.trajectory.points[ind].positions[3] =  0.0;
 goal.trajectory.points[ind].positions[4] =  0.0;
 goal.trajectory.points[ind].positions[5] =  0.0;
 goal.trajectory.points[ind].velocities.resize(6);
  for (size_t j = 0; j < 6; ++j)
  {
    goal.trajectory.points[ind].velocities[j] = 0.0;
  }
  goal.trajectory.points[ind].time_from_start = ros::Duration(20.0);
  return goal;
}
actionlib::SimpleClientGoalState getState()
```

```
  {
    return traj_client_->getState();
  }
 };

int main(int argc, char** argv)
{
  ros::init(argc, argv, "umay_spline_traj");
  RobotArm arm;
  arm.startTrajectory(arm.armExtensionTrajectory());
  while(!arm.getState().isDone() && ros::ok())
  {
    usleep(100000);
  }
}
```

EK A4

```
/*  Created on: Sep 13, 2012
 *      Author: cagatay, header file
 */
#ifndef ARM_KDL_INVERSE_KINEMATICS_H_
#define ARM_KDL_INVERSE_KINEMATICS_H_
#include <umay_controllers_gazebo/inverse_kinematics.h>
#include <pr2_arm_kinematics/pr2_arm_kinematics_utils.h>
#include <kdl/chainfksolverpos_recursive.hpp>
#include <kdl/chainiksolvervel_pinv.hpp>
#include <kdl/chainiksolverpos_nr_jl.hpp>
class ArmKdlInverseKinematics : public InverseKinematics
{
      public:
      ArmKdlInverseKinematics(const urdf::Model &robot_model,
                  const std::string &robot_description,
                  const std::string &root_name,
                  const std::string &tip_name);
            virtual ~ArmKdlInverseKinematics();
            int CartToJnt(const KDL::JntArray& q_init,
                  const KDL::Frame& p_in,
                  std::vector<KDL::JntArray>& q_out);
            void getSolverInfo(kinematics_msgs::KinematicSolverInfo &response);
      private:
            int CartToJnt(const KDL::JntArray& q_init,
                  const KDL::Frame& p_in,
                  KDL::JntArray& q_out);
      private:
            KDL::Chain _chain;
            std::vector<double> _min_angles;
            std::vector<double> _max_angles;
            kinematics_msgs::KinematicSolverInfo _solver_info;
};
#endif
```

```
/*
 * arm_kdl_inverse_kinematics.cpp
 *
 *  Created on: Sep 13, 2012
 *      Author: cagatay
 */
#pragma once
#include <umay_controllers_gazebo/arm_kdl_inverse_kinematics.h>
#include <umay_controllers_gazebo/solver_info_processor.h>
ArmKdlInverseKinematics::ArmKdlInverseKinematics(const urdf::Model &robot_model,
        const std::string &robot_description,
        const std::string &root_name,
        const std::string &tip_name)
{
    if (!pr2_arm_kinematics::getKDLChain(robot_description, root_name, tip_name, _chain)) {
        ROS_ERROR("KDL ağacı bulunamadı");
        ROS_ASSERT(false);
    }

    SolverInfoProcessor solver_info_processor(robot_model, tip_name, root_name);
    _solver_info = solver_info_processor.getSolverInfo();

    for (unsigned int i = 0; i < _solver_info.joint_names.size(); i++) {
        _min_angles.push_back(_solver_info.limits[i].min_position);
        _max_angles.push_back(_solver_info.limits[i].max_position);
    }
}
ArmKdlInverseKinematics::~ArmKdlInverseKinematics()
{
}
int ArmKdlInverseKinematics::CartToJnt(const KDL::JntArray& q_init,
        const KDL::Frame& p_in,
        std::vector<KDL::JntArray>& q_out)
```

```
{
    KDL::JntArray q;
    int res = CartToJnt(q_init, p_in, q);
    if (res > 0) {
        q_out.clear();
        q_out.push_back(q);
        return 1;
    } else {
        q_out.clear();
        return -1;
    }
}
void
ArmKdlInverseKinematics::getSolverInfo(kinematics_msgs::KinematicSolverInfo
&info)
{
    info = _solver_info;
}
int ArmKdlInverseKinematics::CartToJnt(const KDL::JntArray& q_init,
        const KDL::Frame& p_in,
        KDL::JntArray& q_out)
{
    KDL::JntArray q_min(_min_angles.size());
    KDL::JntArray q_max(_max_angles.size());
    for (unsigned int i = 0; i < _min_angles.size(); i++) {
        q_min(i) = _min_angles[i];
        q_max(i) = _max_angles[i];
    }
    KDL::ChainFkSolverPos_recursive fksolver1(_chain);
    KDL::ChainIkSolverVel_pinv iksolver1v(_chain);
    KDL::ChainIkSolverPos_NR_JL iksolverpos(_chain, q_min, q_max, fksolver1,
iksolver1v, 1000, 1e-6);
    int ret = iksolverpos.CartToJnt(q_init, p_in, q_out);
ROS_DEBUG("q_init: %f %f %f %f %f", q_init(0), q_init(1), q_init(2), q_init(3),
q_init(4));
    ROS_DEBUG("q_out: %f %f %f %f %f", q_out(0), q_out(1), q_out(2),
q_out(3), q_out(4));
```

```
    if (ret < 0) {
            ROS_DEBUG("Servis Çözüm bulamadı, hata: %i", ret);
          return -1;
       } else {
            ROS_DEBUG("Servis çözüm buldu");
          return 1;
       }
}
```

ÖZGEÇMİŞ

Ad Soyad: Aydın Çağatay Sarı

Doğum Yeri ve Tarihi: İstanbul / 25.11.1983

Adres: Şemsettin Günaltay Cad. Efes. Apt. No 196 / 20 , Erenköy İstanbul

E-Posta: acagataysari@gmail.com

Lisans: Yıldız Teknik Üniversitesi Elektrik Mühendisliği Bölümü **(** 2003-2009)

Mesleki Deneyim:

Proje Sorumlus - SK Teknoloji Araştırma ve Geliştirme Sanayi ve Tic. Aş (Şubat 2012-)

Stajer - Bosch Sanayi ve Tic. AŞ. (Eylül 2011 – Ocak 2011)

Arge Mühendisi - Assan Elektronik Tic. (Mart 2009 – Temmuz 2009)

Stajer - Gersan Elektrik Tic. Ve Sanayi AŞ. (Ocak 2007 – Şubat 2007)