İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

VPYTHON ORTAMINDA FARKLI TOPOLOJİDEKİ SERİ MANİPÜLATÖRLER İÇİN KİNEMATİK MODEL ÇIKARIMI

YÜKSEK LİSANS TEZİ

Ece COŞGUN

Mekatronik Mühendisliği Anabilim Dalı

Mekatronik Mühendisliği Programı

OCAK 2015

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

VPYTHON ORTAMINDA FARKLI TOPOLOJİDEKİ SERİ MANİPÜLATÖRLER İÇİN KİNEMATİK MODEL ÇIKARIMI

YÜKSEK LİSANS TEZİ

Ece COŞGUN
(518121033)

Mekatronik Mühendisliği Anabilim Dalı

Mekatronik Mühendisliği Programı

Tez Danışmanı: Yrd. Doç. Dr. S. Murat Yeşiloğlu

OCAK 2015

İTÜ, Fen Bilimleri Enstitüsü'nün 518121033 numaralı Yüksek Lisans Öğrencisi **Ece COŞGUN**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"VPYTHON ORTAMINDA FARKLI TOPOLOJİDEKİ SERİ MANİPÜLATÖRLER İÇİN KİNEMATİK MODEL ÇIKARIMI "** başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Yrd. Doç. Dr. S. Murat YEŞİLOĞLU** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Prof. Dr. Metin GÖKAŞAN** İstanbul Teknik Üniversitesi	.....
	Yrd. Doç. Dr. Erhan AKDOĞAN Yıldız Teknik Üniversitesi	.....

Teslim Tarihi : **15 Aralık 2014**
Savunma Tarihi : **20 Ocak 2015**

ÖNSÖZ

Yüksek lisans tezimi gerçekleştirmemde yardımlarını esirgemeyen değerli danışmanım Yrd. Doç. Dr. S. Murat Yeşiloğlu ve tez boyunca kıymetli zamanından bana vakit ayıran Arş. Gör. Musa Yazar'a ve beni herzaman destekleyen aileme teşekkürü bir borç bilirim.

Ocak 2015

Ece COŞGUN
(Araştırma Görevlisi)

İÇİNDEKİLER

Sayfa

KISALTMALAR

JPL : Jet Propolsion Laboratories
m : metre
cm : santimetre

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

SEMBOL LİSTESİ

n : Serbestlik derecesi
R : Rotasyon matrisi
$\widehat{\omega}$: (3×1) Dönme ekseni vektörünün ters (skew) simetrik matrisi
$\vec{\omega}_{b1}$: b1 ekseninin açısal hız vektörü
$\vec{V}_{b1}$: b1 ekseninin doğrusal hız vektörü
$\vec{l}_{a,b1}$: a noktası ile b1 noktası arasındaki (3×1) uzunluk vektörü
$\emptyset$: (6×n, 6×n) Manipülatör propogasyon matrisi
$\overrightarrow{h_1}$: (6×1) Bir numaralı eklemin dönme ekseni vektörü
H : (6×n, n) Genel dönme eksen matrisi
$\underline{V}$: Eklemlerin uzaysal hız vektörü
$\vec{\vec{V}}$: (6×1) Uzaysal hız vektörü
$\vec{\vec{V}}_t$: (6×1) Uç işlevcinin hız vektörü
J : (6×n) Jakobyen matris
$\underline{\dot{\theta}}$: (n×1) Eklemlerin açısal hızı
$J^{\#}$: Jakobyen matrisinin sözde tersi

VPYTHON ORTAMINDA FARKLI TOPOLOJİDEKİ SERİ MANİPÜLATÖRLER İÇİN KİNEMATİK MODEL ÇIKARIMI

ÖZET

Bu çalışmada seri topolojik yapıya sahip manipülatörler uzaysal vektör cebri ile ters kinematik modellemesi yapılmıştır. Tezde bu amaçla ilk olarak uzaysal vektör cebri tanıtılıp bu metodla kinematik modellemenin nasıl yapılabileceği anlatılmıştır. Ardından da endüstride en çok kullanılan altı serbestlik dereceli seri manipülatörün bu metodla kinematik algoritması elde edilmiştir. Tezde ters kinematik algoritmasını test etmek için eşzamanlı görüntü ve yazılım çalıştırma avantajına sahip olan Vpython yazılımı kullanılmıştır. Vpythonda tasarım için grafik arayüzü bulunmamaktadır. Ayrıca program modülünde ekli üç boyutlu araç sayısı kısıtlı olmasına rağmen görüntü ve yazılımın eş zamanlı çalışması küçük boyutlu projeler için programın amacını karşılamaktadır. Aksi taktirde böyle bir uygulamayı gerçekleştirebilmek için ilk olarak herhangi bir cad programında manipülatörün çizilip ardından bir başka yazılım programında kullanacak olan kinematik modelin kodlanması son olarak da bu iki farklı platformlarda oluşturulan modellerin eş zamanlı olarak çalışabilmesi için tek bir yerde birleştirilmesi gerekmektedir. Bu amaçla kullanacak yazılım programlarının birbirini destekleyen yapıda olması gerekmektedir. Bu doğrultuda mühendislik yazılımı konusunda tercih edilen programlarından biri olan Matlab öne çıkmaktadır.

Cad programlarından Solidworks, Catia ve Proengineer gibi yazılım programlarıyla bağlantından Matlab'a bağlantı (link) yazılımları sayesinde bu ortamlarda çizilen montajlar Matlab ortamına kolay bir şekilde aktarılabilmektedir. Ayrıca Matlab benzer bir çalışmayı "Simmechanics Araçkutusu (Toolbox)" ile gerçekleştirmiştir. Bu programda, oluşturulan montaj dosyası Matlab/Simmechanics bağlantısı (link) yardımıyla Simmechanics'e atılmaktadır. Simmechanics de gelen modeli dinamik olarak modelleyip simülasyon gerçekleştirmektedir. Yalnız bu ortamdan istenilen veriyi çekmek mümkün olmamaktadır. Ayrıca benzer bir çalışma Solidwork'un "Cosmos Motion" programıdır. Bu program, sadece görüntü çıktısı vermekte herhangi bir data çıktısı alınamamaktadır.

Tezdeki amaç ters kinematik modellemesinin uzaysal vektör cebri ile farklı seri manipülatörler üzerine çalışma mekanizmasını ölçmek olduğundan algoritma eksik, tam ve artık manipülatörler üzerine aynı referans girişleri ve aynı örnekleme (sampling) zamanı ile simülasyon testleri yapılmıştır. Bu testler sırasında 0.01 cm hata teleronsına izin verilmiştir. Grafik çıktıları algoritmanın bu hata teleronsına uygun olduğunu göstermiştir.

KINEMATIC MODEL EXTRACTION FOR SERIAL MANIPULATORS WHICH HAVE IN DIFFERENT TOPOLOGIES IN VPYTHON ENVIRONMENT

SUMMARY

In this study, we aim at obtaining inverse kinematic model of a serial manipulator using Spatial Operator Algebra. For this purpose firstly Special Operator Algebra and it's use for kinematic modelling are introduced in this thesis. Secondly the kinematic algoritm of the manipulator which has six degree of freedom which is mostly used at the industy, is obtained using this method. At this thesis, to test the inverse kinematic algoritm Vpython software is used to work for simultaneous view and software. Although the number of 3D tools is limited, the software is suitable for minor projects. Otherwise to realize this application firstly at any cad software program the manipulator must be designed secondly at any software program the kinematic model must be coded, the last the generated models which are at different platform is combined to work simultaneously. For this reason the software programs must be support each other. In this respect Matlab which is a preferred for engineering software, combess forward.

From the cad program, Solidworks, Catia and Proengineer has the link program to provide the Matlab. Owing to the link the assembly can be transfered to the Matlab easily. Also Matlab made the same working with "Simmechanics Toolbox". The assemby files can be transfered to the Simmechanics with the link program. Simmechanics is modelled dynamically and makes the simulation. However from this environment lt can not get desired data information. Besides the same practice it is the Cosmos Motion software which is included Solidworks. This program gives just view output and wont give data output.

The objective is measure the working mechanism on the serial manipulator the with the spatial vector algebra. It is done various test using on the nonredundant, exact, redundant manipulator. During the test the same reference input and the same sampling time is used and made simulation tests. During the test, the error telorance is 0.05 cm. From the graph output the algoritm is suitable for the error tolerance.

Another aim is to analyze the inverse kinematics modeling problems get a little bit. To obtain the inverse kinematic there are generally two methods. First method is sembolic method. If that method is used, the software code must be coded sembolic. So the system works very slowly and if the number of degree of freedom is bigger than six, the software may not work. For the offline working it can be satisfy for the accuracy but the online working the time working is not satisfing. Second method is numeric method.

At this method the inverse kinematic is obtained from the inverse of Jacobian matrix. Normally to get inverse of any matrix it must have full rank it means it must be square matrix. But at robotic if the degree of freedom of manipulator is six, it is full rank. But from one to inf degree of freedom can be used. So the Jakobian matrix is

not full generally. So to get the inverse of the matrix there are some methods. These methods give some error. So they are compare due to the amount of the error. Usually the least square method is used. And at the software part it can be psode command. This command use the least square method and both Matlab and Python is the same. But while at Matlab code is written and it can working, at Python to work with this code part it must add module ,which is needed, at the top of the program. Also there are some methods to obtain the inverse Jakobian which is not full, that makes the matrix square. Nevertheless its some coloums or raws are useless. So it can be well analyse which coloumb or raws are useful.

So when we look at the 2 method, we used the numeric method. Because the used kinematic method is compac to code the any software program. So quickly the inverse kinematic is obtained and the software works very quick. This result is satisfy for online working.

To test the code as the input we use the mouse input. First of all from the user it is wanted to enter the needed information which is the manipulator's length, angle, rotation axis. After the code is working at backward and quickly it brings the wanted manipulator to the screen. And it waits to the command. The command is the needed the position information which is at cartesian coordinate space. This information gives with mouse left click.

At Python there is needed code parts and modules for this process. After that the end effector of manipulator comes the position. To see the result the graphs are ploting both desired and tracked dynamicly. When it is looked at the graphs, it can be seen that it is satisfation. Sometimes the manipulator can't reach the desired position. It can be based on singularity. So to express the thesis, the working consist of three main part.

At part one, it consists two minor parts. That is intoduction and literation research. At introduction minor part, totally at this thesis what it is done from the beginning to the end. At the research of literation part, it is explained that what are the similar working. However like explained at this part there is not exactly similar work. In general at the literature for application studies the another inverse kinematic methods are used.

At part two, first of all it is expain that what serial manipulation is, where it is used and comparison of the other topological manipulator kinds. The other manipulation kind is parallel and tree based structure. While It has some advantage according to the other ones, it also has the disadvantages. However it used more at industry. After the kinematic model method is introduced.

To explain the method two degrees of freedom serial manipulator is used first of all. From initial linear and angular velocity to end effector's linear and angular velocity it is obtained. So this algoritm is applied to a six degree of freedom manipulator. And the equations of this manipulator are given at table as both explicitly and implicitly. From the equations it can be seen that it is an exactly algoritm. So coding part is a little difficult according to the other kinematic modeling method but it also has easy part. It is inverse kinematic section. At this kinematic model it is used numerical method when the forward kinematic section is coded, for obtaining the inverse kinematic just one general while loop and the pseode command are needed. So it shorten the code distances and causes to shorten the time interval.

Next The sembolic Jacobian matrix is showed as sembolic. Normally when the Jakobian matrix is obtained from an another method, it can not guess it is true or not. For example a simple method that is iterative is calculate the the linear part of Jacobian matrix. This method uses the end effectors position information and it is getten respectively partial derivative according to joint angles. So when the method is analysed first it only provides linear parts and also if the result is incorrect, we cant know. But this method provides explicity Jakobian matrix closed situation. So for the initial position of the manipulator it can be showen the Jakobian matrix is correct or not. So this also shows the kinematic algoritm is correct or not.

To test the algoritm we design a grafical interface. It is Tkinter and it comes with Python. So it is no need to combine the Python and Tkinter. So someone works the program first the grafical interface window meets the user and when the needed text lines are filled, at another window the manipulator meets the users. And when the user clicks the windows somewhere, the manipulator exceeds to exist point. Testing the algoritm, at appendice-A there are many test for difference manipulator and differerent error value. From the graph plots show that when the manipulator degree of freedom is rise and the error value is minimize, the time cycle is rising and generally the error value will not become zero. So either the manipulators frame's are changed or the manipulators link length are changed.

At final part, The software program is introduced. To introduce which the release of the software is used and what it can be done with the program is explained. So the first the Python second the Vpython (which is our coding software program) is introduced and relatively which modules are used and what the differences of the project are. So to compac a code part the algoritm is coded class structure. So the helping of class structure if any mistake are found any place so it can be easily seen and fix. And to develop the code it is very useful. The class method is more popular for expert software user.

Lastly for testing the algoritm, different degree of freedom manipulators are entered the main code page. And it is showed that the code for the image is dynamic and the inverse kinematic algoritm is also dynamic because of the system goes the target position, and demostrated at graphs which satisfactoryly demonstrate the results.

1. GİRİŞ

Bu tezde uzaysal vektör cebri kullanılarak seri topolojik yapıya sahip manipülatörlerin ters kinematik modellemesi yapılmıştır. Ters kinematik çözüme ilişkin analitik yöntemler sembolik ya da nümerik olarak çözülebilmektedir.

Sembolik çözümün en büyük dezavantajı manipülatör serbestlik derecesi artması durumlarında oluşmaktadır. Özellikle türev alma operasyonunda gittikçe karmaşık hale gelen ifadeler oluşmaktadır. Bu anlamda nümerik yöntemlerin daha avantajlı olduğu bilinmektedir.

Kinematik modeli çıkartmak demek pratikte Jakobyen matrisini elde etmekle eş anlamlıdır. Denavit-Hertenberg [1] gibi literatürde bilinen ve yaygın olarak kullanılan yöntemlerle Jakobyen matrisini elde etmek, özellikle serbestlik derecesi yüksek olan manipülatörler için kolay değildir. Uzaysal vektör cebrinde ise Jakobyen matrisini sistematik ve kolay programlanabilir bir yöntemle nümerik olarak elde ederiz.

Yazılım dili olarak kullanılan Vpython aynı anda hem görüntü hem de kod yazma ve çalıştırma imkanı sağlamaktadır. Vpythonun 3 boyut görsel kütüphanesinde yaklaşık 15 tane seçenek ile olabildiğince temel araçlar bulunurken bu kütüphaneyi desteklemek için 2 boyutlu araçlar kısmıda yaklaşık bir o kadar kullanılabilecek araç bulundurmaktadır. Programın dosya/örnekler bölümünde çeşitli örnek çalışmalar bulunmaktadır. Bu örneklere bakıldığında programla ile her türlü 3 boyutlu çizimin yapılabileceği gözlenmektedir. Ama cad programlarının alışık olduğumuz grafik arayüzü bu tarz programlarda bulunmadığından karmaşık bir sistem ancak uzman düzeydeki kullanıcılar tarafından çizilebilmektedir.

Tezde gerçekleştirilen çalışmanın bir benzeri Andrew Lee tarafından en son 2009 yılında güncellemesi yapılan “Vpython Robotic Arm” adlı çalışmadır [30]. Yapılan çalışma en yaygın kullanılan manipülatörlerden Scara robotun Vpythonda tasarlanıp doğrusal ve sinüsoidal referans ile eklemlerin yörünge takibi gerçekleştirilmiştir. Ayrıca ekranda istenilen noktalara hareket ettirilmesi de sağlanmaktadır. Bu

çalışmada arka planda çalışan matematiksel işlemler robot geometrisinden elde edilmiştir. Sistem üç serbestlik dereceli ve seri bir manipülatör olduğundan matemetiksel denklemleri elde etmek zor olmamıştır. Ayrıca Vpython'da elde edilen görüntü, program kurulum ile beraber gelen "double pendulum" örneği değiştirilerek elde edilmiştir. Tasarımcı sitesinde bu ayrıntıları belirtmiştir.

1.1 Literatür Araştırması

Uzaysal vektör cebri ilk olarak 1991 yılında ABD NASA'nın bir merkezi olan Jet Propolsion Laboratories (JPL)'de Rodriguez ve ekibi [2, 3] tarafından yapılan çalışmalarla literatüre girmiştir. Genel olarak bakıldığında bu yöntemin temeli "screw theory" [4] olarak bilinmektedir. Bu temelden hareketle Featerstone [5] ve Angeles [6]'in çalışmaları da literatürde yerini almıştır. Uzaysal vektör cebri'nin çeşitli alanlara uygulaması ağırlıklı olarak JPL'de bu alanda çalışan Jain ve ekibi [7, 8, 9, 10] tarafından sürdürülmektedir. Bu teoriyi kullarak Yeşiloğlu [11] çoklu manipülatör yapılarının yardımlaşarak ortak taşıdıkları yük üzerinde çeşitli kontrol yöntemleri kullanılması için uygun dinamik modelleme algoritması geşiltirmiştir. Uzaysal vektör cebri teorisine katkı olarak Yeşiloğlu [12] kinematik yetersiz manipülatörlerin kapalı çevrim topolojide yer almaları durumunda kuvvet dağılımı problemini çözmesi olmuştur.

Iglev ve Graser [13] artık robot denilen yedi ve üzeri serbestlik derecesine sahip sistemlerde ters kinematik algoritmasının nasıl elde edilebileceğini gösterirken, Tolani ve ekibi [14] insan vücudunun herhangi bir parçasını modellemek için gerçek zamanlı ters kinematik algoritma üzerinde çalışmıştır. Aristodou ve Lasenby [15]'nin Cambridge üniversitesinde sunduğu teknik raporda, literatürde var olan ters kinematik yöntemler arasından gerçek zamanlı sonuç elde etmek için yüksek performanslı nümerik yöntemler incelenmiştir. Komura ve ekibinin [16] eklem kaslarının ters kinematiği ile ilgili algoritma tanıtılmıştır. Buss ve ekibi [17] ters kinematik için en küçük kareler yöntemini kullanmıştır. Berkeley ve Joubert [18] ters kinematikte nümerik metodları karşılaştırmış, Castellet [19] doktora çalışmasında interval yöntemi kullanarak elde ettiği ters kinematik çözümün avantajlarını açıklamıştır. Wang ve ekibi [20] mekanik sistemlerin ters kinematik algoritmasını çözmek için çeşitli optimizasyon algoritmaları kullanılmıştır.

Yazılım kısmında kullanılan Vpython, Scherer ve ekibinin [21] Rossum ve Boer'in [22] 1991 yılında Python adıyla tanıttığı programına üç boyutlu görsel kütüphane eklemiştir. Sands [23] çalışmasında Vpython'un algoritma yapısınının vektör tabanlı oluşundan bahsedip mekanizma tekniğinde kullanılabileceğini açıklamıştır. Wochal, Cekus and Warys [24] çalışmasında Czestochowa üniversitesinde bulunan mobil bir robotun görselleştirilmesini yapıp program ile kontrolünü gerçekleştirmişlerdir. Python'un literatürde kullanımı daha fazladır. Pedrogosa ve ekibi [25] Python kullanarak bir makine öğrenme programı gerçekleştirmiştir. Cock ve ekibi [26] çalışmasında Biopython'u tanıtıp Python'un hesaplamalı moleküler biyoloji için ayrıca bir modülünün bulunduğunu anlatmışlardır.

2. UZAYSAL VEKTÖR CEBRİ YÖNTEMİ İLE KİNEMATİK MODELLEME

2.1 Topolojik Yapılarına Göre Manipülatörler

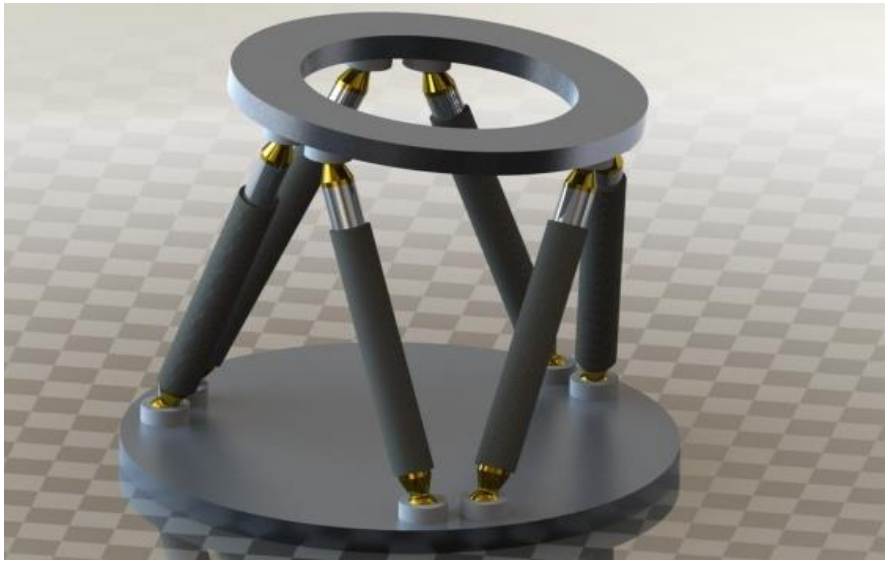
Manipülatörler; serbestlik derecesi, eyleyici tipleri, ve topolojik yapılarına göre çeşitli şekillerde sınıflandırılmaktadır. Topolojik yapı sınıflandırması ise genel olarak üçe ayrılmaktadır.

- Seri topolojik (açık çevrim) yapı
- Paralel topolojik (kapalı çevrim) yapı
- Ağaç (seri+paralel) topolojik yapı

Bu topolojik yapılardan seri yapı, eklemlerin birbiri ardına sırayla zincir şeklinde bağlandığı yapıdır. Uç işlevcisinin üzeinde tanımlanan bir kısıt bulunmamaktadır. Paralel yapıda manipülatör tabanıyla uç işlevci birbirine bağlıdır. Dolayısıyla uç işlevci üzerinde tanımlı kısıtlar vardır. Ağaç yapı ise ortak tabandan çıkan manipülatörlerin birden fazla uç işlevcisinin olduğu durumdur.

Kullanım yerleri herbiri için değişmektedir. Paralel manipülatörler çalışma uzaylarının darlığı, konumlandırma hassasiyetlerinin yüksekliği nedeniyle sanayide paketleme, taşıma robotları olarak sıklıkla kullanılmaktadır. Sanayinin bu konuda en çok kullandığı manipülatör ise delta robottur [27]. Anlı ve ekibinin [28] çalışmasında gösterilen uçuş simulasyonu için geliştirilen Stewart platformu [29] endüstride yaygın olarak kullanılmaktadır. Şekil 2.1'de bir Stewart platformu gösterilmiştir.

Ağaç topolojik yapıya sahip manipülatörler kontrolünün zorluğu sebebiyle sanayide pek fazla kullanılmamaktadır. İnsan vücudundan örnek vermek gerekir ise el parmaklarının bu sınıfta olduğu değerlendirilir.



Şekil 2.1: Stewart platformu.

Burada bilek mobil platformu oluştururken parmak uçlarının herbiri bu ağaç yapısının birer uç noktasıdır. Şekil 2.2'de bir robotik el bulunmaktadır.

Şekil 2.2 : Robotik el.

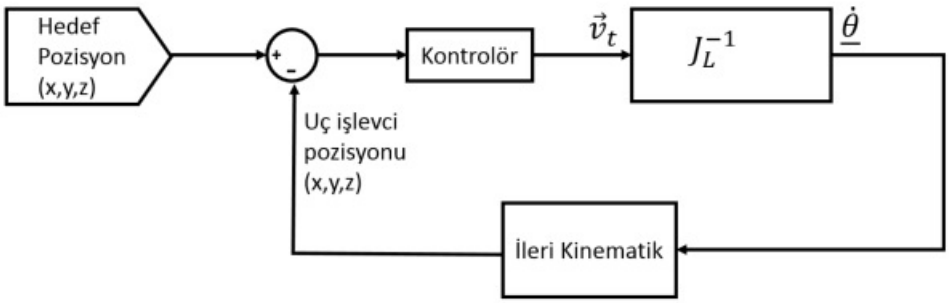
2.1.1 Seri Manipülatörler

Seri manipülatörler endüstride en fazla kullanılan robotik yapılardır. Eklemler ve linkler birbiri ardına bağlandığından paralel manipülatörlere göre dayanıklılığı daha düşük olmaktadır. Bunun sonucu olarak uç işlevcinin taşıyabileceği yük miktarı da azdır. Çalışma uzaylarının genişliği, kinematik ve dinamik modellemelerinin kolaylığı bu manipülatörleri avantajlı hale getirmektedir. Şekil 2.3' de Staubli firmasına ait TX200 6 serbestlik dereceli modeli verilmiştir.

Şekil 2.3 : Staubli TX200 altı serbestlik dereceli modeli.

2.2 Kinematik Modelleme

Kinematik modelleme, ileri ve ters kinematik olmak üzere 2 kısımdan oluşmaktadır. İleri kinematikte eklem uzayından uç işlevci kartezyen uzayına ters kinematikte ise uç işlevci kartezyen uzayından eklem uzayına geçilmektedir. Veri girişi olarak değerlendirildiğinde ise ileri kinematikte giriş değerleri eklemlerin açısal hızlarıyken çıkış değerleri ise uç işlevcinin 6 boyutlu hız değerleridir. Şekil 2.4 ters kinematiğin kapalı döngüsünü göstermektedir.



Şekil 2.4 : Kapalı döngüde ters kinematik şeması.

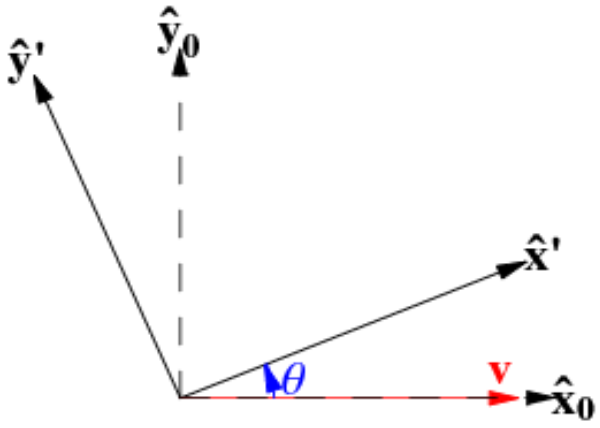
Modellemede rotasyon matrisleri önemlidir. Link eksen takımının sabit eksen takımına göre ifadesini elde etmek için rotasyon matrislerini kullanmamız gerekir. Bu yüzden önce rotasyon matrisinin uzaysal vektör cebrinde nasıl elde edildiği gösterilecektir. Ardından iki serbestlik dereceli bir manipülatörün uzaysal vektör

cebri ile modellemesi gösterildikten sonra bu yöntemin altı serbestlik dereceli bir robota uygulaması ile devam edilecektir.

2.2.1 Rotasyon matrisi

Rotasyon matrisi, 3 boyutlu uzayda bulunan bir vektörün dönme hareketine bağlı olarak bulunacağı yeni oryantasyonunu tespit etmede kullanılan matematiksel bir araçtır. Şekil 2.5'de eksenlerin z ekseni etrafında θ açısı kadar dönmesi gösterilmiştir. Yeni durumun eski koordinat düzlemi baz alınarak ifade edilişi denklem (2.1) ve (2.2) de gösterilmiştir. Denklem (2.3) ise bu iki denklemin bir arada bir matris formunda yazılışıdır. Eşitlik (2.3) de görülebildiği gibi eski koordinat sistemiyle yeni koordinat sistemi aradaki bağlantıyı sağlayan matris rotasyon matrisidir.

3 boyutlu uzayda sırasıyla x, y ve z eksenlerindeki herhangi bir açıyla eksenlerdeki dönme için hesaplanan formüller denklem (2.5) ile (2.7) arasında gösterilmiştir.



Şekil 2.5 : Düzlemde vektörlerdeki dönme.

$$\hat{x}' = cos\theta\hat{x}_0 + sin\theta\hat{y}_0 \quad \textbf{(2.1)}$$

$$\hat{y}' = cos\theta\hat{x}_0 - sin\theta\hat{y}_0 \quad \textbf{(2.2)}$$

$$\begin{bmatrix}\hat{x}' \\ \hat{y}'\end{bmatrix} = \begin{bmatrix}cos\theta & sin\theta \\ cos\theta & -sin\theta\end{bmatrix}\begin{bmatrix}\hat{x}_0 \\ \hat{y}_0\end{bmatrix} \quad \textbf{(2.3)}$$

$$R = \begin{bmatrix}cos\theta & sin\theta \\ cos\theta & -sin\theta\end{bmatrix} \quad \textbf{(2.4)}$$

Manipülatörün birden fazla eksende dönme hareketi durumunda ise dönmekte olan eksenlere ait matrislerin ardı ardına çarpılması gerekmektedir.

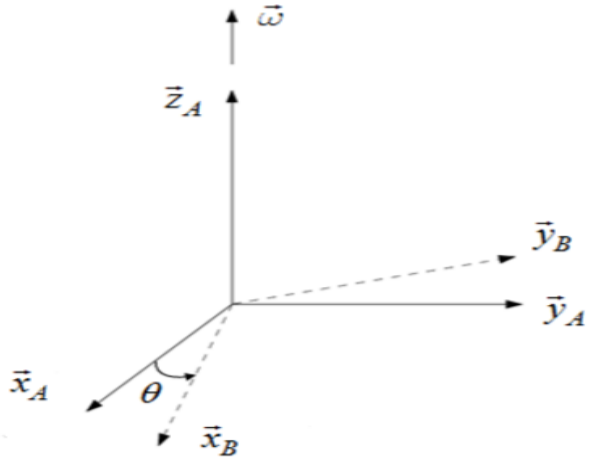
$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & sin\alpha \\ 0 & -sin\alpha & cos\alpha \end{bmatrix} \quad \textbf{(2.5)}$$

$$R_y(\beta) = \begin{bmatrix} cos\beta & 0 & -sin\beta \\ 0 & 1 & 0 \\ sin\beta & 0 & cos\beta \end{bmatrix} \quad \textbf{(2.6)}$$

$$R_z(\gamma) = \begin{bmatrix} cos\gamma & sin\gamma & 0 \\ -sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(2.7)}$$

Ancak bu yöntemde tekillik (singularity) oluşabilmektedir. Robot kinematik modelleme yöntemlerinden Denavit-Hartenberg yönteminde bir linkin rotasyon ve pozisyon bilgisini içeren 4x4'lük transformasyon matrisleri kullanılmaktadır. Kinematik modellemede herbir eklem için yazılan transformasyon matrisleri arka arkaya çarpılarak tabandan uç işlevciye robotun transformasyon matrisi bulunmaktadır.

Uzaysal vektör cebri kinematik modelleme metodu ise rotasyon matrisini hesaplamada Rodriguez formülünü kullanmaktadır. (2.5) ile (2.7) arasında verilen formüllerden daha genel olarak Rodriguez formülü herhangi bir vektör etrafında θ kadar dönmeye ilişkin rotasyon matrisini vermektedir. Rodriguez formülünün avantajı özellikle nümerik hesaplamada ortaya çıkmaktadır. Formüle giriş değeri olarak açı ve dönme eksen bilgisinin verilmesi yeterlidir. Denklem (2.8) de görülen $\widehat{\omega}$ ifadesi ω vektörünün ters (skew) simetrik halidir. Dönme eksen takımları şekil 2.7 de görülmekte olup 3x1 boyutlu birim vektör olarak atanır.



Şekil 2.6 : Uzaysal vektör analizinde dönme eksen takımı.

$$R = e^{\hat{\omega}\theta} = I + sin\theta\hat{\omega} + (1 - cos\theta)\hat{\omega}^2 \quad \textbf{(2.8)}$$

Denklem (2.8) deki $e^{\hat{\omega}\theta}$ ifadesinin açık hali eşitlik (2.9) da

$$e^{\hat{\omega}\theta} = I + \left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - ..\right)\hat{\omega} + \left(\frac{\theta^2}{2!} + \frac{\theta^4}{4!} + \frac{\theta^6}{6!}..\right)\hat{\omega}^2 \quad \textbf{(2.9)}$$

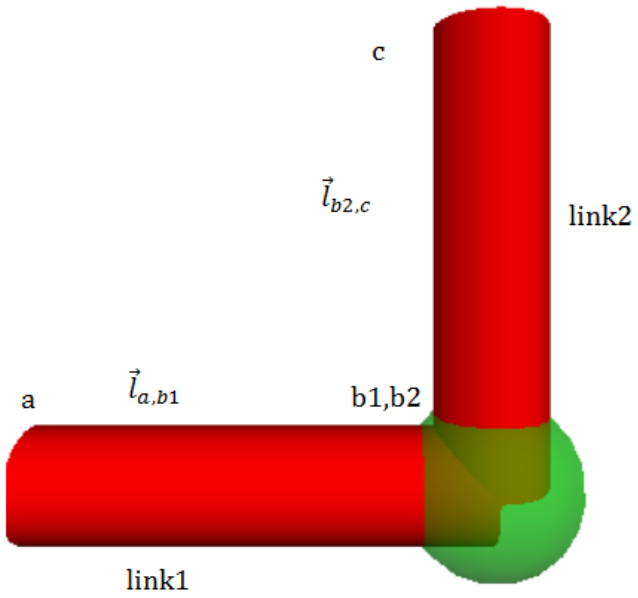
$$sin\theta = \left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - ..\right) \quad \textbf{(2.10)}$$

$$(1 - cos\theta) = \left(\frac{\theta^2}{2!} + \frac{\theta^4}{4!} + \frac{\theta^6}{6!}..\right) \quad \textbf{(2.11)}$$

$$\vec{\omega} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}, \hat{\omega} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \quad \textbf{(2.12)}$$

2.2.2 Rijit bir manipülatörün kinematik modeli

Şekil 2.8' de iki serbestlik derecesine sahip rijit bir manipülatörün modeli verilmiştir. Cisim başlangıçta hareketsiz olduğundan başlangıç doğrusal ve açısal hızları sıfırdır.



Şekil 2.7 : Rijit iki linkli manipülatör.

Link 1'deki doğrusal ve açısal hız denklemleri

$$\vec{\omega}_{b1} = \vec{\omega}_a = 0 = \vec{V}_a \quad \textbf{(2.13)}$$

$$\vec{V}_{b1} = \vec{V}_a + \frac{d}{dt}\vec{l}_{a,b1} \quad \textbf{(2.14)}$$

Link doğrusal hızı, denklem (2.16) da görülebileceği gibi dönme ekseni ile link uzunluğunun vektörel çarpımına eşittir.

$$\frac{d}{dt}\vec{l}_{a,b1} = \vec{\omega}_a \times \vec{l}_{a,b1} \quad \textbf{(2.15)}$$

$$\vec{V}_{b1} = \vec{V}_a + \vec{\omega}_a \times \vec{l}_{a,b1} \quad \textbf{(2.16)}$$

Şekil 2.8'de görülen b1 ve b2 noktaları çakışıktır. Açısal hız hesabında kullanılan $\dot{\theta}$ ifadesi eklem tipi dönelse hesaba katılır. Aksi taktirde işleme girmez ve bu iki noktanın açısal hızı eşit olur. Denklemlerdeki $\vec{h}$, 3x1 boyutundaki dönme eksenidir. Dönme eksen bilgisi için ilk olarak robotun başlangıç pozisyonundayken bütün eksenlere bir eksen ataması yapılıp ardından eklemlerdeki dönmeyle güncellenmesi gerekmektedir.

$$\vec{\omega}_{b2} = \vec{\omega}_{b1} + \dot{\theta}\vec{h} \quad \textbf{(2.17)}$$

Link 2'nin açısal ve doğrusal hızları

$$\vec{\omega}_c = \vec{\omega}_{b2} \quad \textbf{(2.18)}$$

$$\vec{V}_c = \vec{V}_{b2} + \vec{\omega}_{b2} \times \vec{l}_{b2,c} \quad \textbf{(2.19)}$$

Bu iki ifade matris formatında aşağıdaki gibi gösterilir.

$$\vec{\vec{V}}_{b1} = \begin{bmatrix} \vec{\omega}_{b1} \\ \vec{V}_{b1} \end{bmatrix} = \begin{bmatrix} I & 0 \\ -\hat{l}_{ab1} & I \end{bmatrix} \begin{bmatrix} \vec{\omega}_a \\ \vec{V}_{b1} \end{bmatrix} \quad \textbf{(2.20)}$$

$$\vec{\vec{V}}_{b1} = \emptyset_{b1,a}\vec{\vec{V}}_a \quad \textbf{(2.21)}$$

(2.20) ve (2.21) denklemlerindeki hızın üzerindeki çift ok vektörün 6x1 boyutunda olduğunu göstermektedir. Tezin tamamında sembollerin üzerindeki çift oklar bu anlama gelmektedir. Denklemlerde kullanılan $\emptyset$ matrisi propogasyon matrisidir. Uzaysal vektör cebrinin en önemli avantajlarından biri yukarıda görüldüğü gibi yapıların matris formatında kapalı bir hale getirilebilmesidir. Böylelikle ileri kinematikte Jakobyen matris elde edilebilecektir.

2.2.3 Altı serbestlik dereceli rijit bir seri manipülatörün kinematik modeli

Çizelge 2.2'de altı linkin hız değerleri kapalı ve açık formda elde edilip bulunan bu denklemler aşağıdaki matriste yerlerine yerleştirilmiştir. Burada dikkat edilmesi gereken 6x1 boyutundaki hız matrisinin ilk 3x1 lik kısmı açısal hız vektörünü gösterirken diğer 3x1 lik kısmın doğrusal hız vektörünü göstermesidir. Tam tersi yazılırsa denklemlerde değişiklik olmaktadır. Denklem 2.23 ve 2.24 çizelgedeki denklemleri daha kompak bir şekilde göstermektedir. Denklem 2.24 çizelgeyi tam açık bir şekilde gösterirken denklem 2.25 ise bu formüllerin kapalı halde nasıl olması gerektiğini göstermektedir. Bu denklem yapısı sadece altı serbestlik dereceli manipülatörler için değil serbestlik derecesi 6'dan farklı sistemlerde de kinematik modelleme algoritması aynıdır. Altı serbestlik dereceli sistem seçilmesi sanayide en çok kullanılan serbestlik dereceli sistem olmasından ileri gelmektedir.

Çizelge 2.2 : Altı hız vektörünün kapalı ve açık olarak gösterilişi.

$\vec{\vec{V}}_0$	$\vec{\vec{0}}$	$\vec{\vec{0}}$
$\vec{\vec{V}}_1$	$\vec{\vec{H}}_1\dot{\theta}_1$	$\vec{\vec{H}}_1\dot{\theta}_1$
$\vec{\vec{V}}_2$	$\emptyset_{21}\vec{\vec{V}}_1 + \vec{\vec{H}}_2\dot{\theta}_2$	$\emptyset_{21}\vec{\vec{H}}_1\dot{\theta}_1 + \vec{\vec{H}}_2\dot{\theta}_2$
$\vec{\vec{V}}_3$	$\emptyset_{32}\vec{\vec{V}}_2 + \vec{\vec{H}}_3\dot{\theta}_3$	$\emptyset_{32}\emptyset_{21}\vec{\vec{H}}_1\dot{\theta}_1 + \emptyset_{32}\vec{\vec{H}}_2\dot{\theta}_2 + \vec{\vec{H}}_3\dot{\theta}_3$
$\vec{\vec{V}}_4$	$\emptyset_{43}\vec{\vec{V}}_3 + \vec{\vec{H}}_4\dot{\theta}_4$	$\emptyset_{43}\emptyset_{32}\emptyset_{21}\vec{\vec{H}}_1\dot{\theta}_1 + \emptyset_{43}\emptyset_{32}\vec{\vec{H}}_2\dot{\theta}_2 + \emptyset_{43}\vec{\vec{H}}_3\dot{\theta}_3 + \vec{\vec{H}}_4\dot{\theta}_4$
$\vec{\vec{V}}_5$	$\emptyset_{54}\vec{\vec{V}}_4 + \vec{\vec{H}}_5\dot{\theta}_5$	$\emptyset_{54}\emptyset_{43}\emptyset_{32}\emptyset_{21}\vec{\vec{H}}_1\dot{\theta}_1 + \emptyset_{54}\emptyset_{43}\emptyset_{32}\vec{\vec{H}}_2\dot{\theta}_2 + \emptyset_{54}\emptyset_{43}\vec{\vec{H}}_3\dot{\theta}_3 + \emptyset_{54}\vec{\vec{H}}_4\dot{\theta}_4 + \vec{\vec{H}}_5\dot{\theta}_5$
$\vec{\vec{V}}_6$	$\emptyset_{65}\vec{\vec{V}}_5 + \vec{\vec{H}}_6\dot{\theta}_6$	$\emptyset_{65}\emptyset_{54}\emptyset_{43}\emptyset_{32}\emptyset_{21}\vec{\vec{H}}_1\dot{\theta}_1 + \emptyset_{65}\emptyset_{54}\emptyset_{43}\emptyset_{32}\vec{\vec{H}}_2\dot{\theta}_2 + \emptyset_{65}\emptyset_{54}\emptyset_{43}\vec{\vec{H}}_3\dot{\theta}_3 + \emptyset_{65}\emptyset_{54}\vec{\vec{H}}_4\dot{\theta}_4 + \emptyset_{65}\vec{\vec{H}}_5\dot{\theta}_5 + \vec{\vec{H}}_6\dot{\theta}_6$

Çizelge 2.2 'nin ikinci sütununda kapalı, üçüncü sütununda açık hali verilen denklem yapısında $\emptyset_{ac}$ propogasyon matrisi denklem (2.22) 'den elde edilmektedir.

$$\emptyset_{a,c} = \emptyset_{a,b}\emptyset_{b,c} \qquad \textbf{(2.22)}$$

$$\begin{bmatrix} \overrightarrow{\overrightarrow{V_1}} \\ \overrightarrow{\overrightarrow{V_2}} \\ \overrightarrow{\overrightarrow{V_3}} \\ \overrightarrow{\overrightarrow{V_4}} \\ \overrightarrow{\overrightarrow{V_5}} \\ \overrightarrow{\overrightarrow{V_6}} \end{bmatrix} = \begin{bmatrix} I & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} \\ \emptyset_{21} & I & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} \\ \emptyset_{31} & \emptyset_{32} & I & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} \\ \emptyset_{41} & \emptyset_{42} & \emptyset_{43} & I & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} \\ \emptyset_{51} & \emptyset_{52} & \emptyset_{53} & \emptyset_{54} & I & \overrightarrow{\overrightarrow{0}} \\ \emptyset_{61} & \emptyset_{62} & \emptyset_{63} & \emptyset_{64} & \emptyset_{65} & I \end{bmatrix} \begin{bmatrix} \overrightarrow{\overrightarrow{H_1}}\dot{\theta}_1 \\ \overrightarrow{\overrightarrow{H_2}}\dot{\theta}_2 \\ \overrightarrow{\overrightarrow{H_3}}\dot{\theta}_3 \\ \overrightarrow{\overrightarrow{H_4}}\dot{\theta}_4 \\ \overrightarrow{\overrightarrow{H_5}}\dot{\theta}_5 \\ \overrightarrow{\overrightarrow{H_6}}\dot{\theta}_6 \end{bmatrix} \quad \textbf{(2.23)}$$

Daha açık bir şekilde

$$\begin{bmatrix} \overrightarrow{\overrightarrow{V_1}} \\ \overrightarrow{\overrightarrow{V_2}} \\ \overrightarrow{\overrightarrow{V_3}} \\ \overrightarrow{\overrightarrow{V_4}} \\ \overrightarrow{\overrightarrow{V_5}} \\ \overrightarrow{\overrightarrow{V_6}} \end{bmatrix} = \begin{bmatrix} I & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} \\ \emptyset_{21} & I & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} \\ \emptyset_{31} & \emptyset_{32} & I & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} \\ \emptyset_{41} & \emptyset_{42} & \emptyset_{43} & I & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} \\ \emptyset_{51} & \emptyset_{52} & \emptyset_{53} & \emptyset_{54} & I & \overrightarrow{\overrightarrow{0}} \\ \emptyset_{61} & \emptyset_{62} & \emptyset_{63} & \emptyset_{64} & \emptyset_{65} & I \end{bmatrix} \begin{bmatrix} \overrightarrow{\overrightarrow{H_1}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} \\ \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{H_2}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} \\ \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{H_3}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} \\ \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{H_4}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} \\ \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{H_5}} & \overrightarrow{\overrightarrow{0}} \\ \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{0}} & \overrightarrow{\overrightarrow{H_6}} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \\ \dot{\theta}_5 \\ \dot{\theta}_6 \end{bmatrix} \quad \textbf{(2.24)}$$

Denklem (2.24) deki formülün kapalı hali

$$\underline{V} = \emptyset H \dot{\theta} \quad \textbf{(2.25)}$$

$\underline{V}$: uzaysal hız vektörü

$\emptyset$:genel propogasyon matrisi

H: dönme eksen matrisi

$\underline{\dot{\theta}}$: eklem hızı

2.2.4 Jakobyen matris

Jakobyen matris, kinematik olarak eklem hızlarından uç işlevcinin hızına ulaşmayı sağlayan ara bağlantı matrisidir. Dinamik olarak ise Jakobyen matrisinin traspozu, uç işlevcinin kuvvet ve torkunun eklemlere olan dağılımına ulaşmayı sağlamaktadır. Literatürde Jakobyen matris elde etmek için iteratif metod, doğrudan türev alma gibi yöntemler bulunur. Bu metodlar içerisinde serbestlık derecesiyle birlikte karmaşıklığı en çok artan doğrudan türev alma yöntemidir. Bu metotta uç işlevcinin konumu sırasıyla manipülatörün ilk ekleminden son eklem değişkenine kısmi türevlerinin alınmasıyla elde edilmektedir. Denklem (2.26) de J Jakobyen matrisini n ise serbestlik derecesini temsil etmektedir.

$$J = \begin{bmatrix} J_\omega \\ J_V \end{bmatrix} \epsilon \ (6 \times n) \quad \textbf{(2.26)}$$

Jakobyen matrisin eklem uzayından kartezyen uzayına geçişinin genel hali denklem (2.27) de gösterildiği gibidir. Bu denklemdeki $\vec{\vec{V}}_t$ uç işlevcinin hızıdır.

$$\vec{\vec{V}}_t = J\underline{\dot{\theta}} \quad \textbf{(2.27)}$$

Uç işlevcinin hızı, daha açık bir şekilde denklem (2.28) olduğu gibi gösterilebilir. Denklemdeki $\emptyset_t$ ifadesi son eklemle uç işlevci arasındaki propogasyon matrisini temsil etmektedir.

$$\vec{\vec{V}}_t = \emptyset_t \emptyset H \underline{\dot{\theta}} \quad \textbf{(2.28)}$$

$$\emptyset_t = [0_{6\times6} \quad 0_{6\times6} \quad 0_{6\times6} \quad 0_{6\times6} \quad 0_{6\times6} \quad \varphi_n]_{6\times36} \quad \textbf{(2.29)}$$

Denklem (2.27) ve (2.28) deki denklemlerden Jakobyen matris (2.30) daki gibi elde edilir.

$$J = \emptyset_t \emptyset H \quad \textbf{(2.30)}$$

Kartezyen uzaydan eklem uzayına geçiş aşağıdaki gibidir.

$$J^{\#} \vec{\vec{V}}_t = \underline{\dot{\theta}} \quad \textbf{(2.31)}$$

Ters kinematik model çözümünde $J^{\#}$ Jakobyen matrisinin sözde tersidir.

3. VPYTHON

Vpython, 2000 yılında David Scherer tarafından Python programa diline 3 boyutlu görsel arayüz eklenmiş programlama dilidir. Python 1990 yılında Guido van Rossum tarafından Amsterdam da geliştirilmeye başlanmıştır. Adını ambleminde bulunan ve Python dilinin türkçesini gösteren aynı isimli bir yılandan değil Guido Van Rossum'un çok sevdiği, Monty Python adlı altı kişilik bir ingiliz komedi grubunun Monty Python's Flying Circus adlı gösterisinden almıştır.

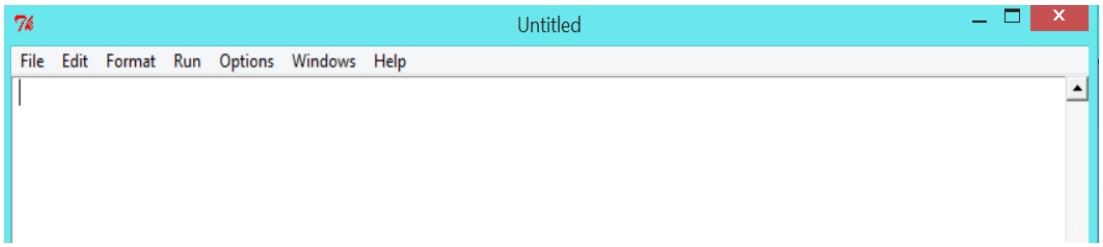
Günümüzde Python Yazılım Vakfı çevresinde toplanan gönüllülerin çabalarıyla sürdürülmektedir. Python, nesne yönelimli, yorumlanabilen, birimsel (modüler) ve etkileşimli bir programlama dilidir. Modüler yapısı, sınıf dizgesini (sistem) ve her türlü veri alanı girişini desteklemektedir. Hemen hemen her türlü platformda çalışabilmektedir. (Unix , Linux, Mac, Windows, Amiga, Symbian). Python ile sistem programlama, kullanıcı arabirimi programlama, ağ programlama, uygulama ve veritabanı yazılımı programlama gibi birçok alanda yazılım geliştirebilmektedir.

Python ilk sürümü 1.0 'ı 1994 yılında çıkarmıştır. Son kararlı sürümleri, 2.x serisinde Python 2.7.8 ve 3.x serisinde Python 3.4.2'dir. 2.x ve 3.x serileri ayrı ayrı geliştirilmeye devam etmektedir. 3.x sürümü de 2.x sürümünü desteklememektedir. Python kullanılarak geliştirilmiş programlarda 2.x sürümleri daha kararlı olması nedeniyle bu seri kullanmaktadır. Tez çalışması kapsamında "VPython-Win-64-Py 2.7-6.10" sürümü kullanılmıştır. Bu kodu geniş olarak açarsak windows 64 bit, Python 2.7.6 sürümü ve Vpython 6 dır. Örneğin Vpython'un bir eski versiyonu olan 5.74 versiyonu Python 3.2 yi desteklemektedir. Program Vpython anasayfasından Windows, Macintoch ve Linux işletim sistemleri için indirilmeye imkan vermektedir. Herbiri için de indirme metodu aynıdır. Öncelikle Python'un anasayfasından Pythonun ilgili sürümü indirilmeli ardından da ilgili Vpythonun sürümü indirilmelidir. Vpython ilgili Python sürümü haricinde çalışmamaktadır. Vpython'u yükledikten sonra Windows'da çalıştırıldığında 'vidle for Vpython' adında bir arayüzle karşılaşılmaktadır. Kodların bu ekrandan çalıştırabileceği gibi 'shell'

denilen kabuktanda çalıştırılabilmektedir. Yalnız diğer programlama dillerinde de olduğu gibi kabuktan yazılan veriyi program, kaydetmeye imkan vermemektedir.

3.1 Vpythona Giriş

Aşağıdaki şekilde görüldüğü gibi Vpythonun sade bir arayüzü vardır. Menüde file/open kısmından yapılmış örnekler bulunmaktadır. Örnekler, hem programlamada yardım etmede hem de bu programla neler yapılabileceğinin sınırlarını çizmede yardım etmektedir. Ayrıca bu örnekler sadece Vpython konusunda değil aynı zamanda kodlamada Python, grafik arayüz modülü olan Tkinter konusunda da bilgi sahibi olmaya imkan tanımaktadır.



Şekil 3.1 : Vpython arayüzü.

Program arayüzü sade gözüktüğü gibi kullanımı da oldukça kolaydır. Run menüsü 3 ana alt seçenek sunmaktadır. Bunlar sırasıyla Phyton shell, check module ve run module dür. 'Python shell' direk kabuk denilen yapıyı açıp buradan işlem yapmaya imkan vermektedir. Program üçüncü seçenek olan "run module" den çalıştırıldığında ise aynı anda hem derleme hem de çalışmakta ve sonucu göstermek için bu kabuğu açmaktadır. Eğer kodlama da herhangi bir hata varsa hatayla ilgili bilgi verip bizi yönlendirmektedir. Vpython'un eksik yanlarından biri, programlama da bir hata yapıldığında bu arayüzün hatanın sadece hangi kısımda olduğunu ve bu kısmın bulunduğu satır numarasını ekrana yazdırmasıdır. Ama şekil 3.1 den de görüleceği gibi program arayüzünde satır numaraları bulunmamaktadır. Basit ve önemsiz bir sorun gibi görülen bu durum sayfalarca kod yazanlar için hatayı görmekte sorun yaratabilen bir durumdur. 'Check module' ise programın kod olarak yanlış yazılıp yazılmadığını kontrol etmeye imkan vermektedir.

3.2 Kodlamaya giriş

3.2.1 Modüller

Vpython modüler yapılı bir program olduğundan program yazımında gerekli olan modülleri program satırının başına eklemek gerekmektedir. Eğer istenilen modül programa varsayılan (default) olarak ekli gelmişse modül isminin program satırının en başına yazılıp program çalıştırılmasıyla herhangi bir hata ile karşılaşılmamaktadır. Bu amaçla programlama sırasında hangi modüllerin kullanılacağı belirlendikten sonra o modülün ekli gelip gelmediği kontrol edilmelidir.

Bu yazılım için en temel modül ise görüntü oluşumunu sağlayan visual modulüdür. Bu modül, "from visual import*" kelimesinin program satırının başına eklenerek programa entegre edilebilmektedir. Modüller birden fazla şekilde programa eklenebilmektedir. En çok kullanılan method ise üstte verilen örnekte olduğu gibi "from modülismi import* kelimelerini program satırının en başına yazılmasıdır. Diğer bir yol ise "import modülismi" 'dir. İkisi arasındaki fark, ilk kullanılan method da o modülde gerekli olan kod parçacığının kullanabilmek için direk kod adını programa yazmak yeterli iken diğer method da "modülismi.kodismi" yazmak gerekmektedir. Bu yüzden genel kodlamada modüller tamamen from'la başlayarak yazılmıştır. Modül ismi eklemedeki bir diğer önemli noktada "from modülismi import*" kalıbından görülebildiği gibi import'un yanında bir yıldız işareti bulunmaktadır. Bu yıldız işareti o modülde bulunan bütün kullanabilecek özellikleri aktif etmeye yaramaktadır. Örneğin * yerine "from visual import box " yazılmış olsaydı visual modülünden sadece box kodu kullanılmaya aktif olacaktı. Tez çalışması kapsamında programın aşağıdaki modülleri kullanılmıştır.

- Visual (3 boyutlu görsel programlama)
- Math (Matematiksel araç)
- Numpy (Vektör matris işlemleri)
- Numpy.linalg (Matris tersi)
- Visual.graph (Plot çizimi)

Tez çalışması sırasında yazılmış olan modüller tamamen ihtiyacımı karşılamadığı için yoğun matematiksel işlemlerde devamlı kullanılan bazı matematiksel yapıları sınıf yapısı içinde oluşturdum. Kendi oluşturduğum modülleri yazılmış standart modüller gibi program satırının en başına ekleyip kullandım. Yeni bir modül yazıp kullanırken dikkat edilmesi gereken nokta, modülün kaydedileceği yerdir. Python

bir modülü kendi kütüphanesinde aramaktadır. Kütüphanenin adresi, programların kaydedildiği yere göre değişmekle beraber genel olarak "C:\Python27\Lib\ " dır. Eğer yazılan modül başka herhangi bir yere kaydedilip kaydedilen yerde arama yerleri listesine eklenmezse yazılan modül çağrıldığında program modülü bulamadığından hata mesajı vermektedir. Bir diğer dikkat edilmesi gereken nokta ise dosyaların uzantıları konusudur. Dosyalar kaydedilirken isim.py olarak kaydetmek gerekmektedir. Dosya ismine .py eklenmemiş dosyalarda hata mesajı almaya yol açmaktadır.

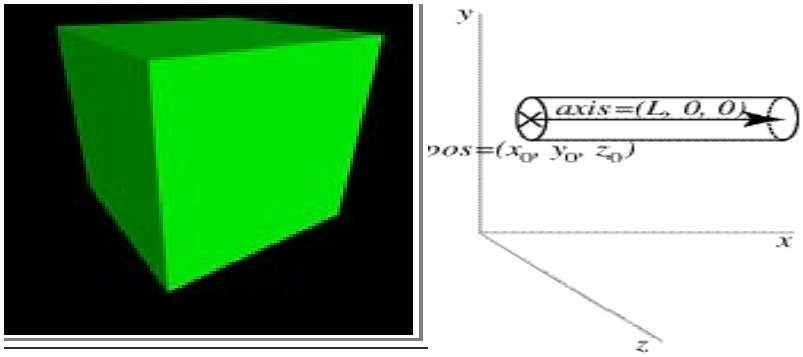
3.2.2 Görüntü oluşturma

Vpytonda 3 boyutlu manipülatör tasarımı için cylinder, box, cone, arrow objeleri kullanılmıştır. Program nesneleri oluştururken nesne tasarım özelliklerine göre değişmekle beraber nesneyi konumlandırmak için kullanıcıdan boyut bilgisi kadar pozisyon bilgiside istemektedir. Program default olarak nesnelerde aşağıdaki değerleri kullanmaktadır.

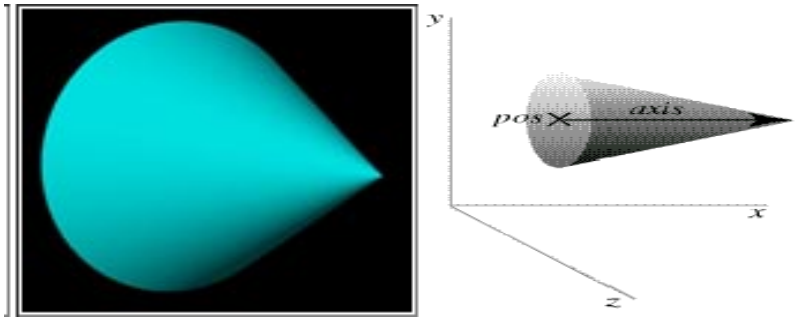
- Box()→box (pos=(0,0,0), size=(1,1,1))
- Cone() → cone (pos=(0,0,0), axis=(1,0,0), radius=1)
- Arrow() → arrow (pos=(0,0,0), axis=(1,0,0), radius=1)
- Frame() → frame (pos=(0,0,0), axis=(1,0,0))
- Sphere() → sphere (pos=(0,0,0), radius=1)

Yukardaki ifadelerdeki "pos" nesnelerin konumlandırılacağı 3 boyutlu uzaydaki yer, "size" nesne boyutu, "radius" yarıçap, "axis" ise yine nesne boyut bilgisidir. Axis ayrıca yön tayin etmekte de kullanılmaktadır. Kullanılan nesneye göre pozisyon ve eksen yerleşim yerinin nesne içinde konumlandırılışı farklı olduğundan aşağıda tez çalışması kapsamında kullanılan nesnelerin bilgisi görülmektedir.

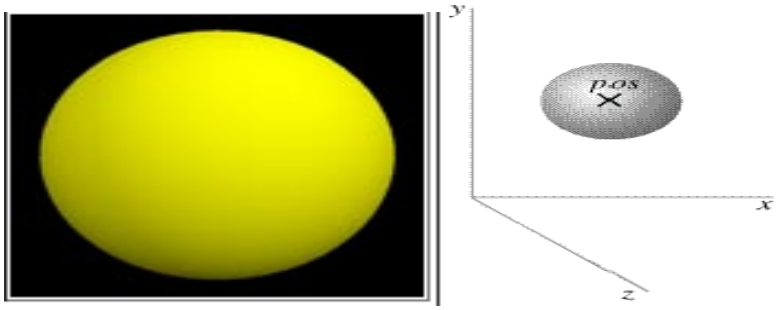
Şekil 3.2 ve şekil 3.4 den görülebileceği gibi kutu ve küre tasarımında pozisyon tam şeklin ortasındayken koni ve silindir de tabandadır. Tez çalışması kapsamında sabit platform "box", linkler "cylinder", eklemler "sphere" uç işlevci ise "cone" kullanılarak tasarlanmıştır. Manipülatör tasarımı, sabit platformdan uç işlevciye doğru gerçekleştirilmiştir. Sabit platformda "box" aracı kullanıldığından ve box'ın pozisyon bilgisi oluşturulan şeklin tam ortasında olduğundan pozisyon değeri (0, -height/2, 0) olarak belirlenmiştir.



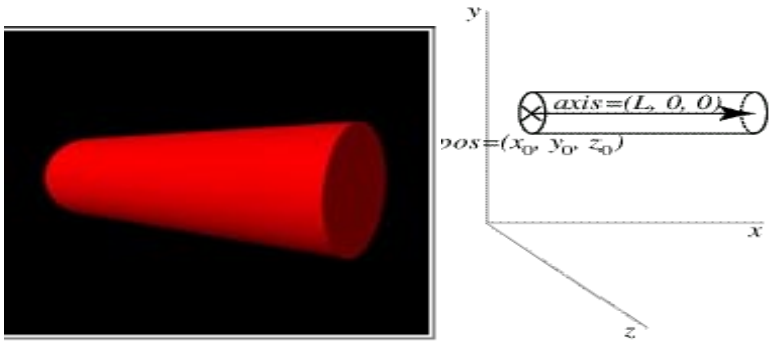
Şekil 3.2 : Dikdörtgenler prizmasının Vpython'da eksen ve pozisyon bilgisi.



Şekil 3.3 : Koninin Vpython'da pozisyon ve eksen yerleşimi.



Şekil 3.4 : Kürenin Vpython'da konum yerleşimi.



Şekil 3.5 : Silindirin Vpython'da konum ve eksen yerleşimi.

İlk linkin pozisyon bilgisi (0, 0, 0) ve jointlerin pozisyon bilgisi denklem (3.1) ile bulunurken diğer linklerin pozisyon bilgisi denklem (3.2) ile bulunmuştur.

Joint[i].pos=link[i].pos+link[i].axis **(3.1)**

Link[i].pos=joint[i-1].pos **(3.2)**

Manipülatörün sıfır pozisyonunu oluşturmak için kullanılan bu formülleri yazılım sabit bir bilgi olarak kabul etmemektedir. Eklemlerlerden herhangi birinin döndürülmesinde manipülatör eklem ve linklerinin dönen eklemi takip etmesi için yukarıdaki formülün yazılımda aynen tekrarlanıp konum bilgisinin yazılıma hatırlatılması gerekmektedir. Burada dikkat edilmesi gereken eklem ve linklerdeki kopmayı önlemek için yazılan kod parçacığının eklemleri döndürmek için yazılan döngü ile aynı döngüde kullanılmasıdır. Kod farklı bir döngüde yazılsa da ilgili joint ve linkler dönmekte olan parçayı zaman farkıyla takip etmektedir. Manipülatörü tek bir parça olarak görebilmek için aradaki bu zaman farkının olabildiğince az olması gerekmektedir. Çünkü gerçek hayatta manipülatör eklemleri dönerken parça parça olmamaktadır. İleri ve ters kinematik denklemleri kullanılarak uç işlevcinin x-y-z doğrultusunda elde edilmiş grafik çıktıları bulunmaktadır. Sistemizde örnekleme zamanı olarak 1 ms seçmemize rağmen grafik çıktıları sistemin ayrık zamanlı çalıştığını ispatlamaktadır.

Tez çalışmasında manipülatör serbestlik derecesi sabit olmadığından dinamik bir kod yazma ihtiyacı oluşmuştur. Bunun için ilk olarak linkler ve eklemler için 2 ayrı boş liste oluşturulup ardından Python'da listelerin içini doldurma komutu olan "insert" komutu kullanılarak ve başlangıçta verilen serbestlik derecesi baz alınarak link ve eklemler uzunluk verileri doğrultusunda dinamik olarak oluşturulmuştur. Bu amaçla kod yazımında frame() yani kordinat atamaya gerek kalmamıştır. Çünkü linkler ve jointler bir listenin elemanı olduğu için istenilen link veya jointle işlem yapmak istenildiğinde sadece adres belirtmek yeterli hale gelmektedir.

Görüntü hızı "rate (frekans)" ile ayarlanabilmektedir. Frekansın değeri 1'den büyük olmalıdır. Frekans değeri arttığında hareketin hızıda artmaktadır. Bunun tam tersi "sleep ()" komutudur. Sleep fonksiyonu, programın ne kadar süre herhangi birşey yapmadan beklemede kalması gerektiğini belirleyen komuttur. Programlama sleep komutu kullanılmamıştır. Görüntü konusundaki son önemli nokta Vpython'da fare (mouse) aracının kullanımıdır. Ters kinematikte, kullanıcının manipülatörün

kartezyen koordinatlar içerisinde gitmesi istediği noktayı klavye yerine fare ile hızlı bir şekilde girmesi sağlanılmıştır. Bu amaçla kullanıcının doğru pozisyonu seçebilmesi için ilk olarak fare ile gezilen noktalar kabuktan (shell) görüntülenip ardından istenilen pozisyon farenin sol tuş tıklanmasıyla manipülatörün kapalı çevrimde istenilen pozisyona ulaşmasını sağlayan bir dinamik bir yazılım parçası yazılmıştır.

3.2.3 Nesne yönelimli programlama

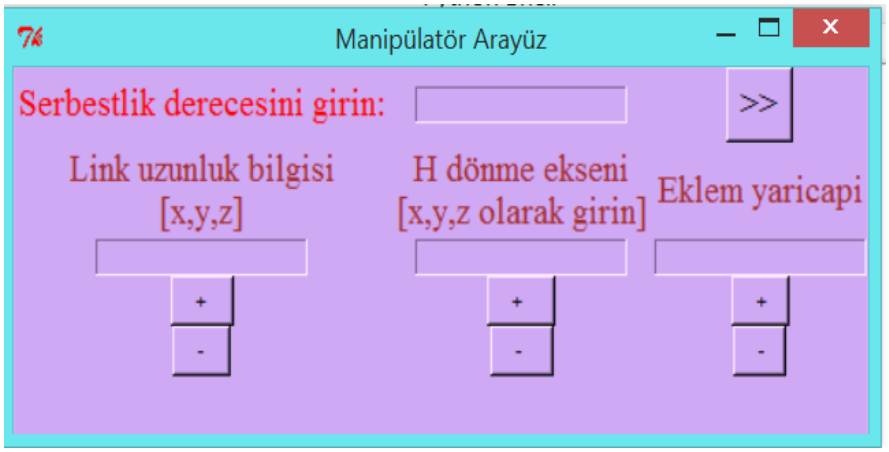
Vpython nesne yönelimli bir programlama dilidir. Nesne yönelimli programlama, ingilizcesi "Object Oriented Programming" olan bir bilgisayar programlama yaklaşımıdır. Günümüzde neredeyse pek çok programlama dili nesne yönelimlidir. Örnek olarak C++, C#, Matlab, Visual Basic, Fortran verilebilmektedir. Nesne yönelimli programlamada, programlama ortamındaki her şey bir nesne olarak kabul edilir ve nesnelerin özellikleri değiştirilerek onlara yeni biçimler verilebilmektedir. Ayrıca her nesnenin duyarlı olduğu durumlar da mevcuttur. Her nesne üzerine uygulanabilecek farklı metotlar oluşturulmuştur. Yapısal programlamada ağırlık programlama komutlarındayken, Nesne yönelimli programlamada yazılımcının ortamdaki nesneler, bunların özellikleri, duyarlı oldukları olaylar ve nesnelere uygulanabilecek metotlar hakkında ayrıntılı bilgi sahibi olması gerekmektedir.

Avantajları sıralandığında en başta yazılımın bakım kolaylığı gelmektedir. Nesne yönelimli programlama dillerinde, nesneler ait oldukları sınıfların sunduğu şablonlarla temsil edilirler. Birbirinin benzeri ya da aynı konu ile ilişkili sınıflar bir araya getirilerek paket (package) adı verilen sınıf kümeleri oluşturulur. Bu durumda yazılımın bakımı demek; ya mevcut sınıflarda değişiklik ya da yeni sınıf eklenmesi anlamına gelmektedir. Bu da yazılımın tümünü hiçbir şekilde etkilemeyecek olan bir işlemdir. Oysa klasik yapısal programlama dilleriyle geliştirilen programlarda yazılımın bakım maliyeti üretim maliyetinin neredeyse yüzde kırkı civarındadır. Bir diğer avantajı genişletilebilirlik özelliğidir. Nesne Yönelimli Programlama'da mevcut bir sınıfa yeni özellik ve metotlar ekleyerek artan işlevsellik sağlamak oldukça kolaydır. Son olarak ise üretilen kodun yeniden kullanılabilirliğidir. Nesnelerin üretildiği sınıflar, ortam içinde her uygulama geliştiricinin kullanımına açık olduğu için ortak bir kütüphane oluşturulmuştur. Her kullanıcı bu ortak kütüphanede

bulunan sınıfları kullanarak gerekli kodu yeniden yazmaktan kurtulur ve uygulamaya özgü kod parçaları, ortak kullanımdaki sınıfların kod uzunluklarına göre göz ardı edilebilir boyuttadır.

3.2.4 Grafik arayüz tasarımı

Grafik arayüz tasarımı için Python kurulumu ile beraber gelen Tkinter programı kullanılmıştır. Tkintere ait olan araçları kullanabilmek için program satırının başına Tkinter'in eklenmesi (import) gerekmektedir. Tkinter, Python'un diğer grafik arayüz tasarımlarına göre kullanımı daha kolay bir arayüz tasarımıdır. Şekil 3.6'da sistem için tasarlanan arayüz programı bulunmaktadır.

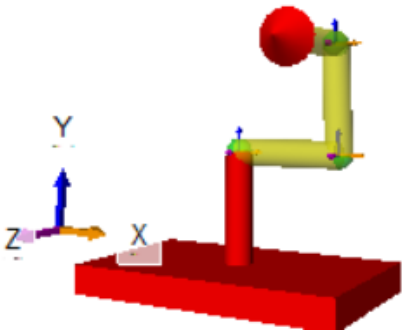


Şekil 3.6 : Tkinter arayüzü.

Üstteki kutulardan ilk olarak manipülatör serbestlik derecesinin girilmesi gerekmektedir. Ardından link uzunluk, H dönme ekseni ve eklem yarıçapları bilgileri girilmelidir. Link uzunluk bilgisi "[x, y, z]" formatında girilmeliyken, h dönme eksen bilgisi sırasıyla manipülatörün ilgili eklemi hangi eksende dönüyorsa o eksen bilgisi küçük harflerle yazılmalıdır. Eklem yarıçapında da manipülatör tasarımında linkler silindir, eklemler ise küre olarak tasarlandığından eklem için küre yarıçapının doldurulması gerekmektedir. Veriler text kutusuna girilip "+" tuşuna basıldığında sırasıyla altta açılacak liste kutusuna eklenmektedir. Hatalı yazılması durumunda ise hatalı yazılan veri seçilip "–" ile listeden çıkarılabilmektedir. Arka planda çalışan döngü bu verileri üstte yazan serbestlik derecesi kadar girmeye izin vermektedir. Verilerin eksiksiz doldurulmasıyla beraber bir başka pencerede manipülatörün sıfır pozisyonu ekrana gelmektedir. Şekil 3.7 ise örnek bir manipülatör tasarımı için grafik arayüzüne girilmiş verileri göstermektedir. Bu veriler doğrultusunda elde edilen manipülatör şekil 3.8'de bulunmaktadır.

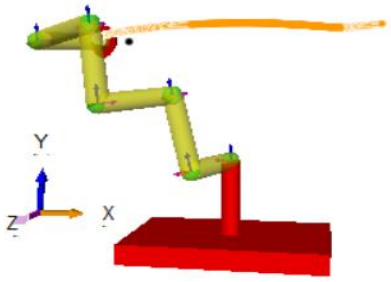
Şekil 3.7 deki verilerin girilmesiyle şekil 3.8'de görüldüğü gibi manipülatörün sıfır pozisyonu ekrana gelmektedir. Yapılan işlem bir ters kinematik algoritması testi olduğundan kartezyen uzaydaki veri farenin sol tuşu ile girilmektedir. Bu sırada da fare ile ekranda gezerken Vpython'un kabuk kısmıda ekranda gezilen noktaların pozisyon bilgisini kartezyen koordinat cinsinden ekrana bastırmaktadır.

Şekil 3.7 : Örnek grafik arayüz uygulaması.

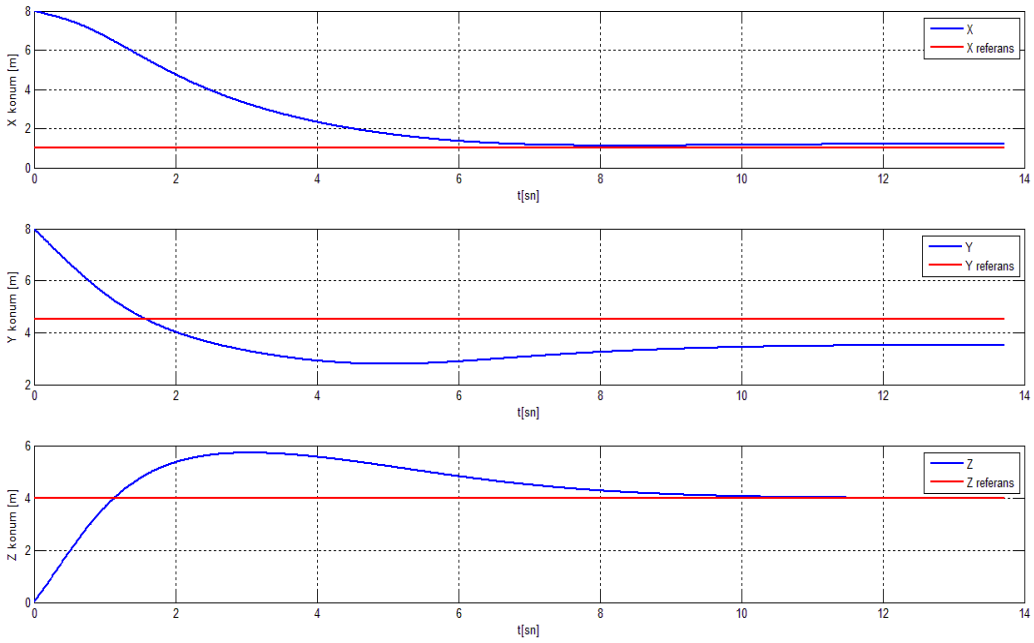


Şekil 3.8 : Şekil 3.7'deki verilerden elde edilen manipülatör.

Ekran, manipülatörün en alt kısmı olan sabit platfomunun bulunduğu yeri (0,0,0) pozisyon bilgisi olarak tanımaktadır. Bu amaçla fare ile gezinilen noktalar farenin ekrandaki pozisyon bilgisidir ve birimi metre cinsindendir. Ekran ise şekil 3.8'den görülebileceği gibi üç boyutludur. Bu yüzden ekranı fareyle oynatıp z ekseninde de bir nokta seçilebilmektedir. Bu girilen veriler ışığında manipülatör istenilen noktaya en kısa yolla gitmeye çalışmaktadır. Üç serbestlik derecesine sahip bir manipülatörün çalışma uzayı dar olduğundan şekil 3.9'da altı serbestlik derecesine sahip bir manipülatör tasarlanıp şekil 3.10'da ise bu manipülatöre ait grafik çıktısı gözlenmektedir.

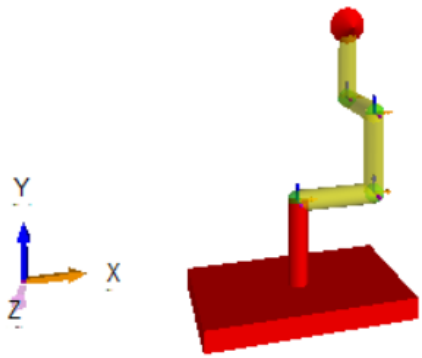


Şekil 3.9 : Altı serbestlik dereceli örnek bir manipülatör.

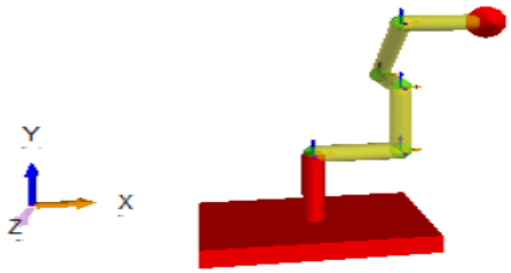


Şekil 3.10 : Şekil 3.9'daki manipülatörünün uç işlevcisinin konumunun grafik çıktıları.

Şekil 3.7 deki grafik arayüzü test etmek için şekil 3.11 ile 3.12 arasında farklı serbestlik derecesine sahip seri manipülatörler tasarlanmıştır.



Şekil 3.11 : Dört serbestlik dereceli seri bir manipülatör.

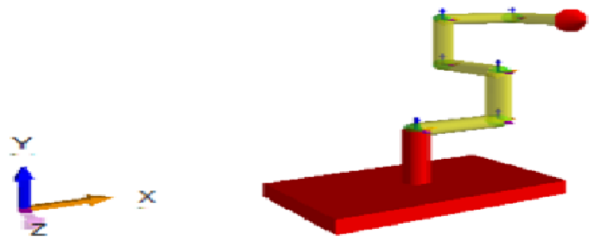


Şekil 3.12 : Beş serbestlik dereceli seri bir manipülatör.

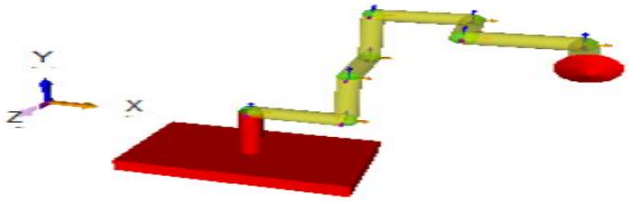
Sistem kapalı çevrimde çalışmaktadır. İlk olarak tekillik (singularity) problemini ortadan kaldırmak için manipütörün herbir eklemi başlangıç olarak 0.01 radyan açısından başlatılmıştır. Bu açı verisine göre bir Jakobyen matris elde edilip ardındanda bu Jakobyen matrisin tersi ile dışarıdan girilen uç işlevci konum bilgisi çarpılıp bu bilgi doğrultusunde eklem açıları radyan cinsinden elde edilmektedir. Bu veri ardından ileri kinematiğe gönderilip bu açı bilgisinin uç işlevcinin hangi pozisyon bilgisine tekabül ettiği hesaplanıp buna göre bu veri ile istenilen veri çıkarılıp bir hata vektörü elde edilmektedir. Normal şartlarda hata sıfıra inene kadar hesaplama işlemi devam etmektedir. Hatanın sıfıra ulaşması çalışma süresini uzatacağından sistemimizde eksenlerdeki 0.01 cm hata toleransına izin verilmiştir. Kişiler kendi sistemleri için bu hata payını artırıp azaltabilir. Ek A'da sistemin sağlıklı çalışıp çalışmadığını test etmek için gerekli veriler gönderilip ve süreler gösterilmiştir.

4. SONUÇLAR

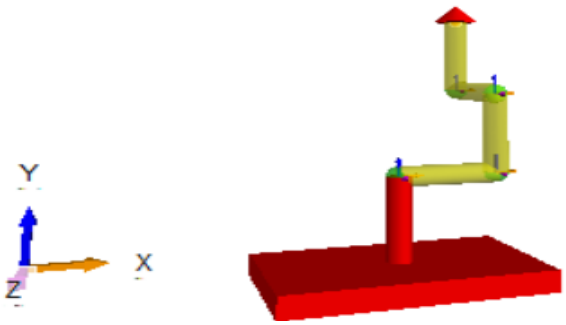
Ters kinematik algoritmasını test etmek için şekil 4.1 ile şekil 4.3 arasında gözlenen eksik, tam ve artık manipülatörler üzerine benzetim çalışmaları yapılmıştır. Gerçek bir karşılaştırma için bu üç farklı manipülatöre giriş olarak aynı referans değerleri verilmiştir. Giriş referans değeri, kullanıcı tarafından ekranda fare ile tıklanılan konumdur. Programda, referans değerine eksenlerde 0.01 cm hata teloransına izin verilmiştir.



Şekil 4.1 : Altı serbestlik dereceli örnek bir manipülatör.

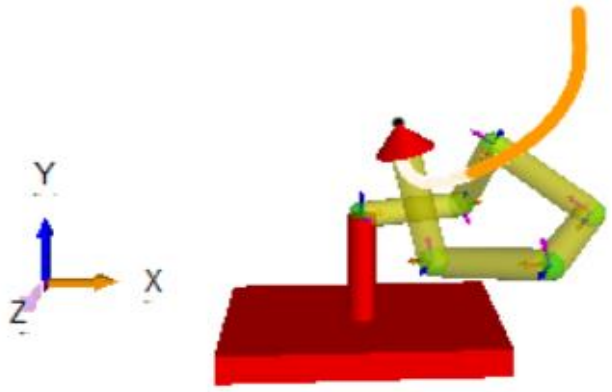


Şekil 4.2 : Sekiz serbestlik dereceli örnek bir artık manipülatör.

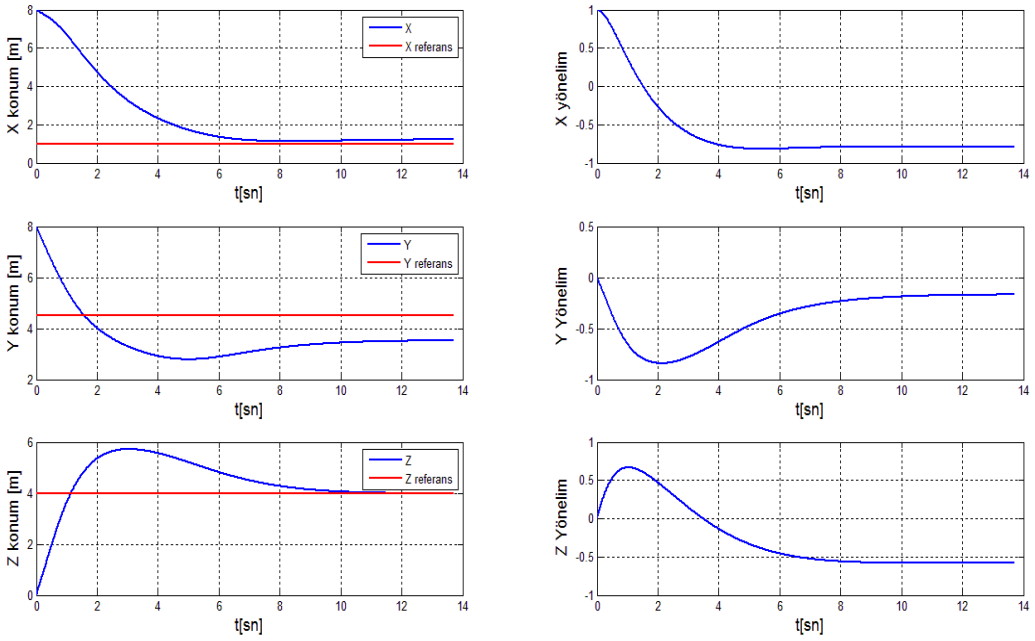


Şekil 4.3 : Dört serbestlik dereceli örnek bir eksik manipülatör.

Şekil 4.4 Altı serbestlik dereceli manipülatörün [1, 4.5, 4] m referans konumuna ters kinematik algoritmasıyla yaklaşımının grafiğini göstermektedir. Şekil 4.5, izlenen bu yörünge sırasında uç işlevcinin konum ve yönelim bilgilerinin grafik çıktılarını göstermektedir.

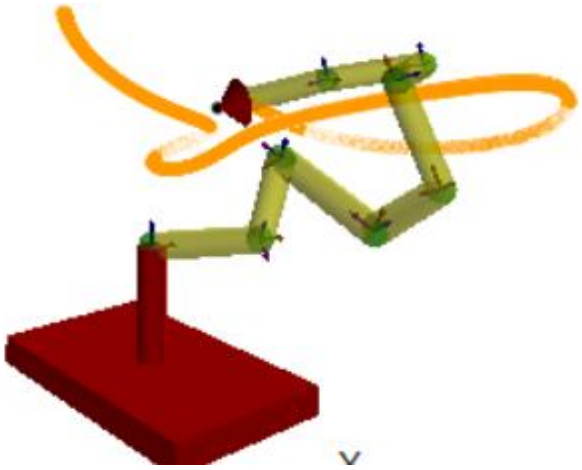


Şekil 4.4 : Altı serbestlik dereceli manipülatörün referans değer takibi.

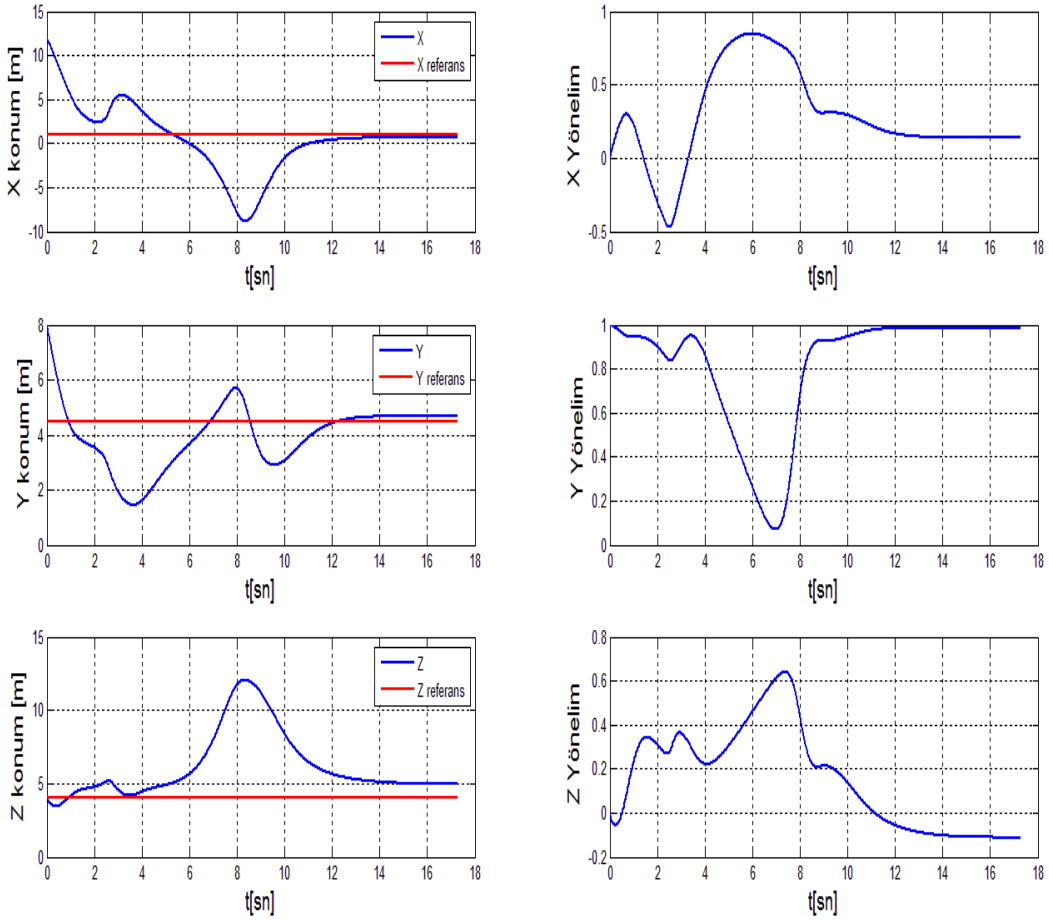


Şekil 4.5 : Altı serbestlik dereceli manipülatörün 10 ms örnekleme zamanı ile referans takibi.

Şekil 4.6, aynı referans değerine sekiz serbestlik dereceli artık bir manipülatörün ulaşabilmesi için izlediği yörüngeyi göstermektedir. Şekil 4.7 ise şekil 4.6'da izlenen yörünge sırasında manipülatör uç işlevcisinin pozisyon ve yönelim bilgisinin grafik çıktısını göstermektedir. Şekil 4.5 ve şekil 4.7'deki grafik çıktılarından altı serbestlik dereceli manipülatörün daha kararlı bir hareket yaptığı ve kapalı çevrimi daha erken tamamladığı gözlenmektedir.

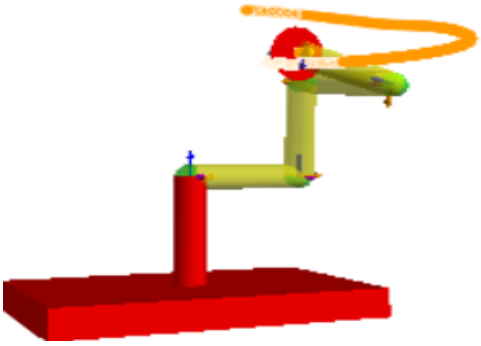


Şekil 4.6 : Sekiz serbestlik dereceli örnek bir manipülatör.

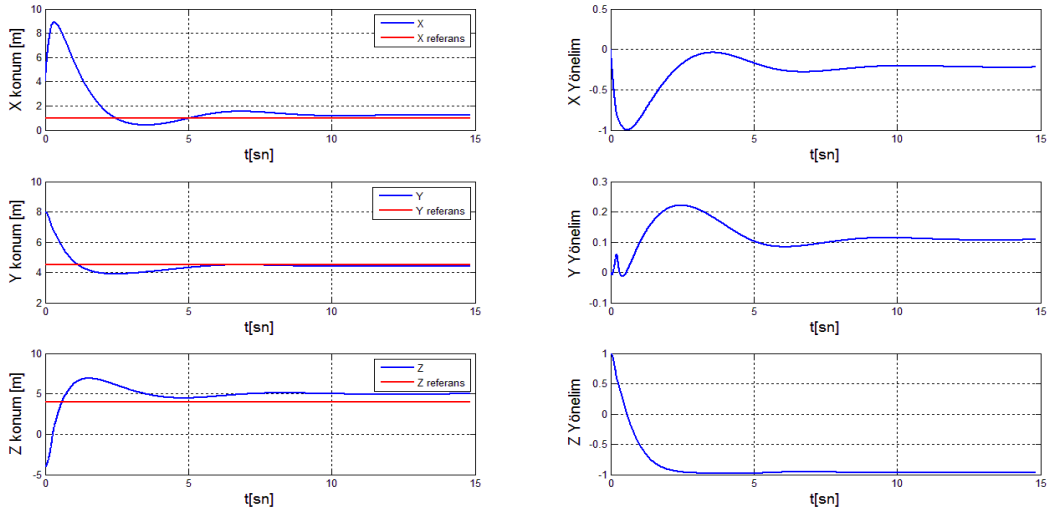


Şekil 4.7 : Şekil 4.6'da bulunan manipülatörün izlediği yörünge sırasında uç işlevcisinin konum ve yönelim grafiği.

Aynı referans değerine dört serbestlik dereceli eksik bir manipülatörün yaklaşımı şekil 4.8'de gözlenip şekil 4.9' da bu yörünge takibine ait olan uç işlevci konum ve yönelim grafik çıktıları bulunmaktadır.

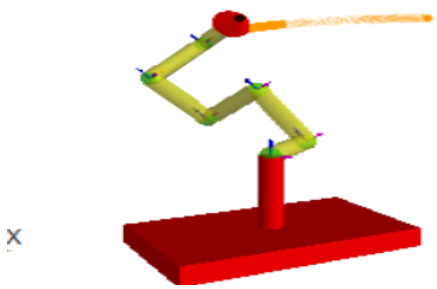


Şekil 4.8 : Eksik bir manipülatörün [1, 4.5, 4] m referans konumuna ulaşmak için izlediği yörünge.

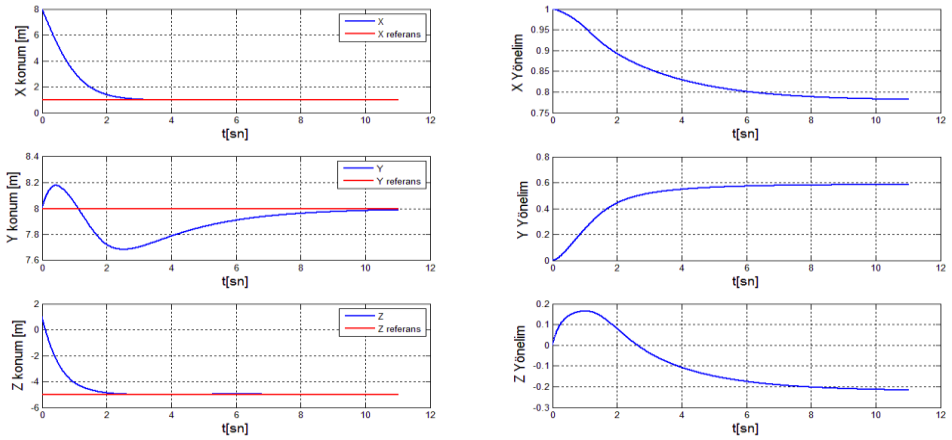


Şekil 4.9 : Şekil 4.8'de bulunan manipülatörün izlediği yörünge sırasında uç işlevcisinin konum ve yönelim grafiği.

Şekil 4.10 Altı serbestlik dereceli manipülatörün [1, 8, -5] m referans konumuna ters kinematik algoritmasıyla yaklaşımının grafiğini göstermektedir. Şekil 4.11, izlenen bu yörünge sırasında uç işlevcinin konum ve yönelim bilgilerinin grafik çıktılarını göstermektedir.

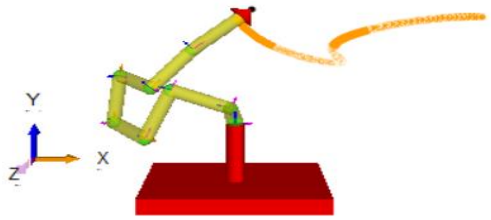


Şekil 4.10 : Altı serbestlik dereceli manipülatörün [1, 8,-5] m pozisyonuna ulaşımı.

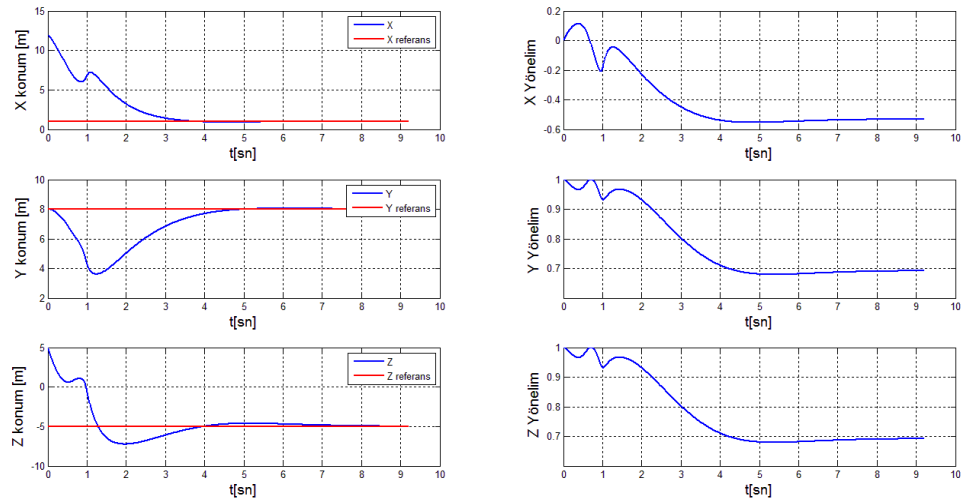


Şekil 4.11 : Şekil 4.10 ‘daki manipülatörün yörünge takibinin grafik çıktısı.

Şekil 4.12 Sekiz serbestlik dereceli manipülatörün [1, 8, -5] m referans konumuna ters kinematik algoritmasıyla yaklaşımının grafiğini göstermektedir. Şekil 4.13, izlenen bu yörünge sırasında uç işlevcinin konum ve yönelim bilgilerinin grafik çıktılarını göstermektedir.

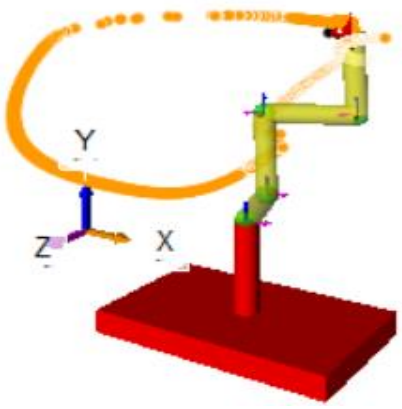


Şekil 4.12 : Sekiz serbestlik dereceli manipülatörün [1, 8,-5] m pozisyonuna ulaşımı.

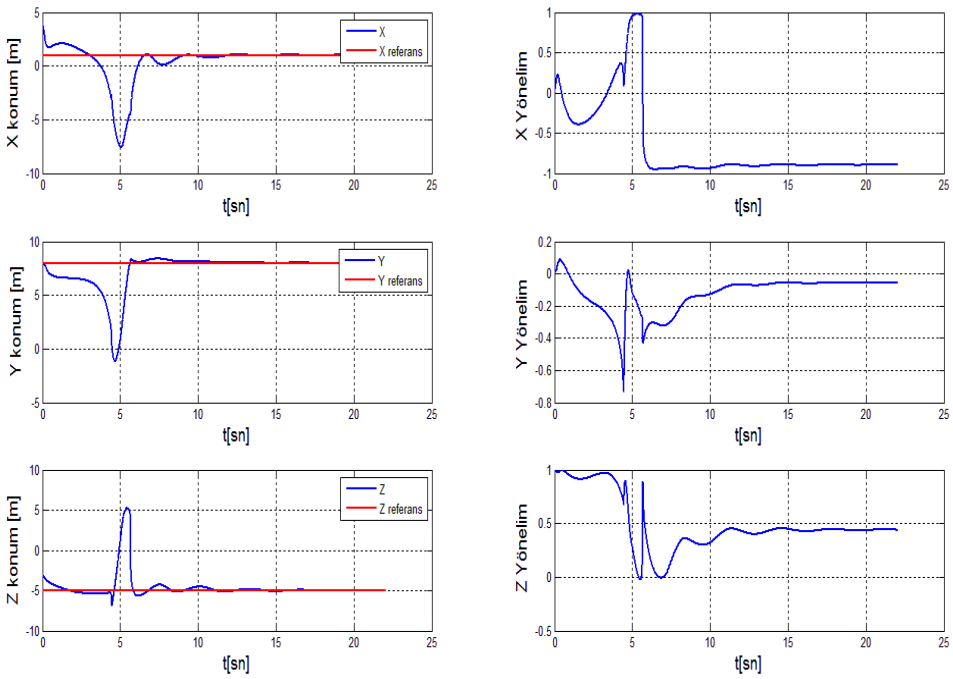


Şekil 4.13 : Şekil 4.12’de bulunan manipülatörün yörünge takibinin uç işlevcisinin pozisyon ve konum grafik çıktısı.

Şekil 4.13 Dört serbestlik dereceli manipülatörün [1, 8, -5] m referans konumuna ters kinematik algoritmasıyla yaklaşımının grafiğini göstermektedir. Şekil 4.14, izlenen bu yörünge sırasında uç işlevcinin konum ve yönelim bilgilerinin grafik çıktılarını göstermektedir.

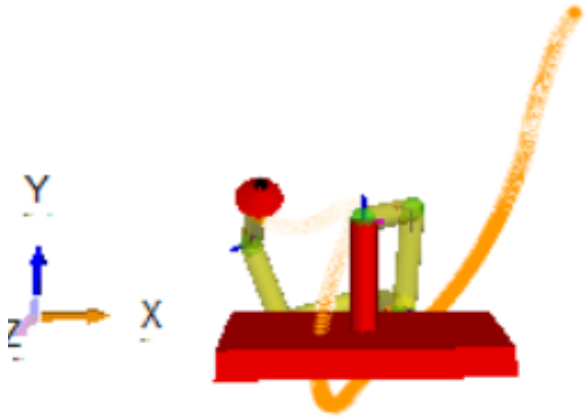


Şekil 4.14 : Dört serbestlik dereceli manipülatörün [1, 8, -5] m pozisyonuna ulaşımı.

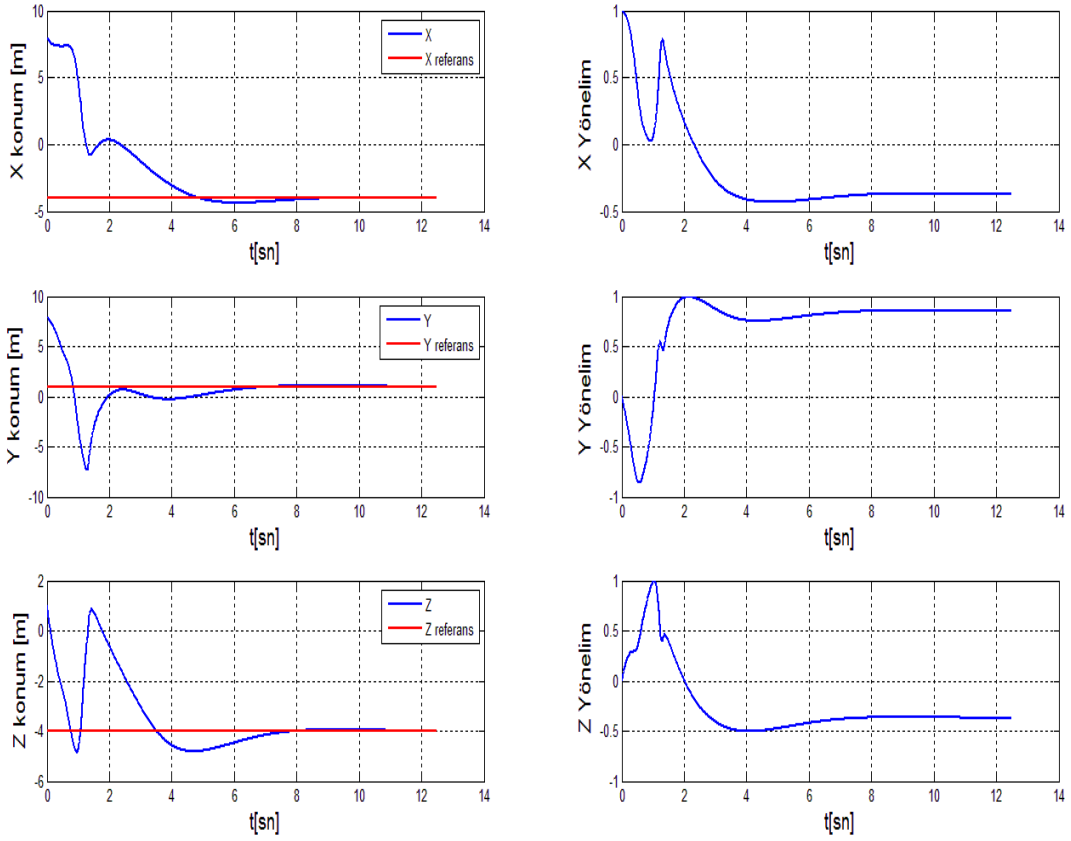


Şekil 4.15 : Şekil 4.14'de bulunan manipülatörün uç işlevcisinin pozisyon ve yönelim grafik çıktısı.

Şekil 4.16 Altı serbestlik dereceli manipülatörün [-4, 1, -4] m referans konumuna ters kinematik algoritmasıyla yaklaşımının grafiğini göstermektedir. Şekil 4.14, izlenen bu yörünge sırasında uç işlevcinin konum ve yönelim bilgilerinin grafik çıktılarını göstermektedir.

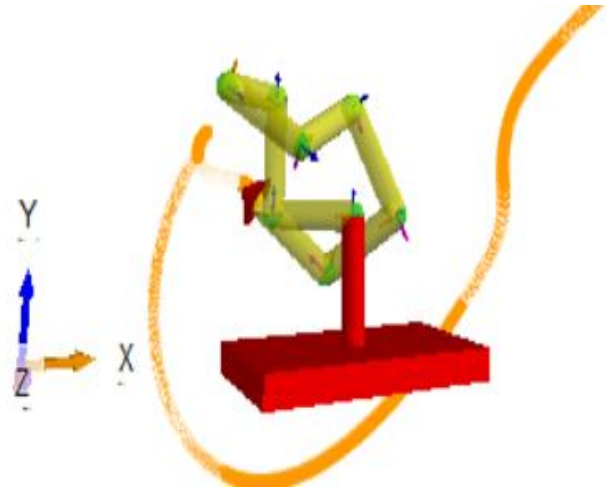


Şekil 4.16 : Altı serbestlik dereceli manipülatörün [-4,1,-4] m pozisyonuna ulaşımı.

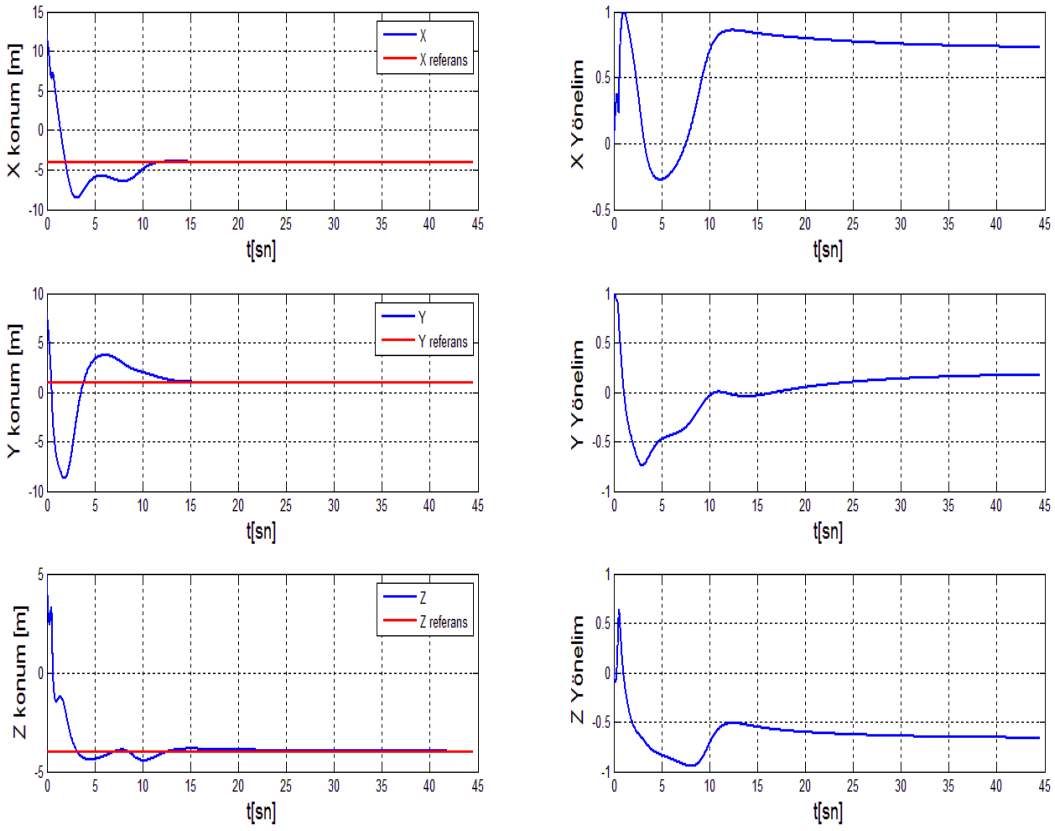


Şekil 4.17 : Şekil 4.16 'da bulunan manipülatörün uç işlevcisinin pozisyon ve yönelim grafik çıktısı.

Şekil 4.18 Sekiz serbestlik dereceli manipülatörün [-4, 1, -4] m referans konumuna ters kinematik algoritmasıyla yaklaşımının grafiğini göstermektedir. Şekil 4.19, izlenen bu yörünge sırasında uç işlevcinin konum ve yönelim bilgilerinin grafik çıktılarını göstermektedir.

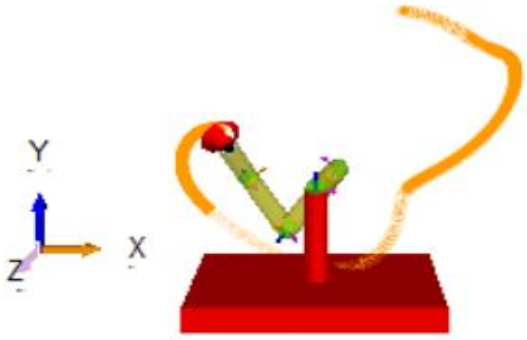


Şekil 4.18 : Sekiz serbestlik dereceli manipülatörün [-4,1,-4] m pozisyonuna ulaşımı.

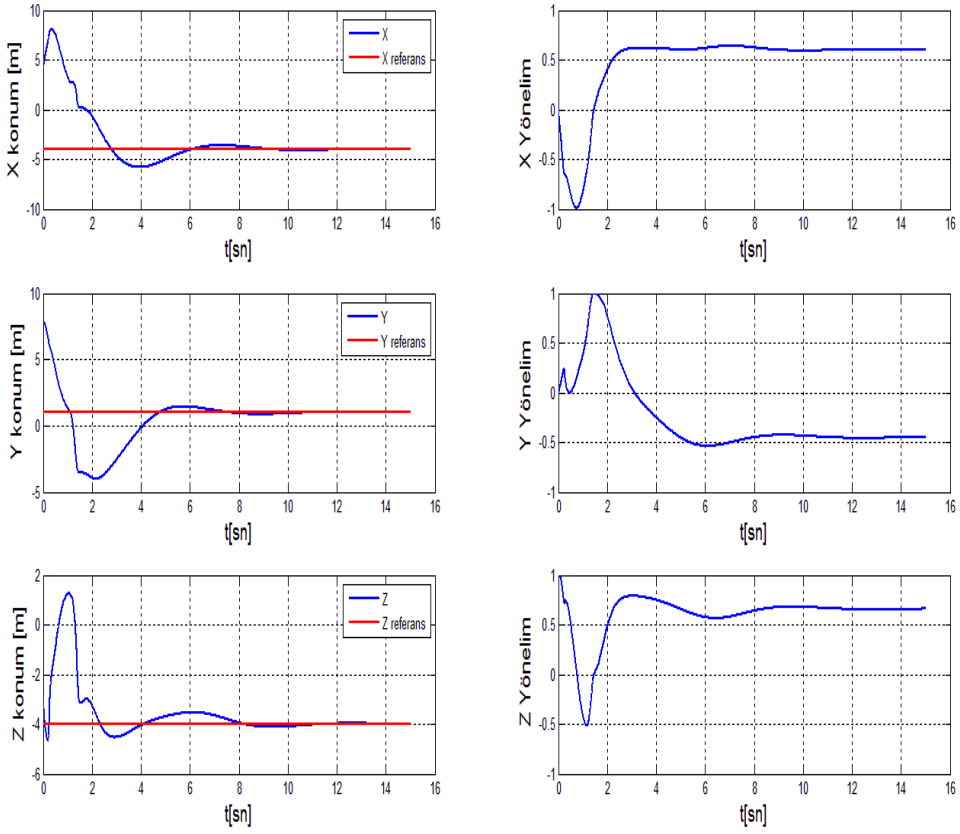


Şekil 4.19 : Şekil 4.18'de bulunan manipülatörün uç işlevcisinin pozisyon ve yönelim grafik çıktısı.

Şekil 4.20 dört serbestlik dereceli manipülatörün [-4, 1, -4] m referans konumuna ters kinematik algoritmasıyla yaklaşımının grafiğini göstermektedir. Şekil 4.21, izlenen bu yörünge sırasında uç işlevcinin konum ve yönelim bilgilerinin grafik çıktılarını göstermektedir.

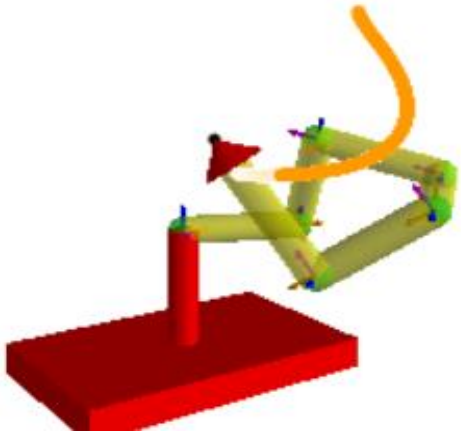


Şekil 4.20 : Dört serbestlik dereceli manipülatörün [-4,1,-4] m pozisyonuna ulaşımı.

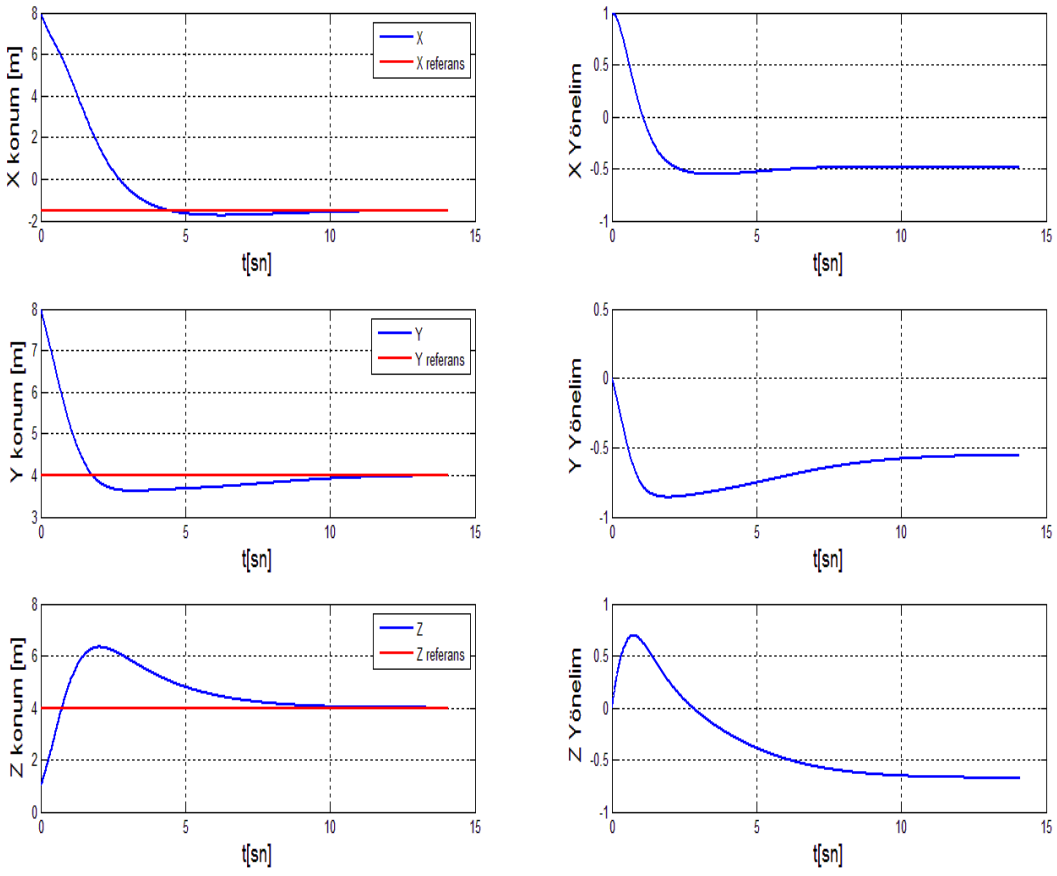


Şekil 4.21 : 4.20'de bulunan manipülatörün uç işlevcisinin pozisyon ve yönelim grafik çıktısı.

Şekil 4.22 altı serbestlik dereceli manipülatörün [-1.5, 4, 4] m referans konumuna ters kinematik algoritmasıyla yaklaşımının grafiğini göstermektedir. Şekil 4.23, izlenen bu yörünge sırasında uç işlevcinin konum ve yönelim bilgilerinin grafik çıktılarını göstermektedir.

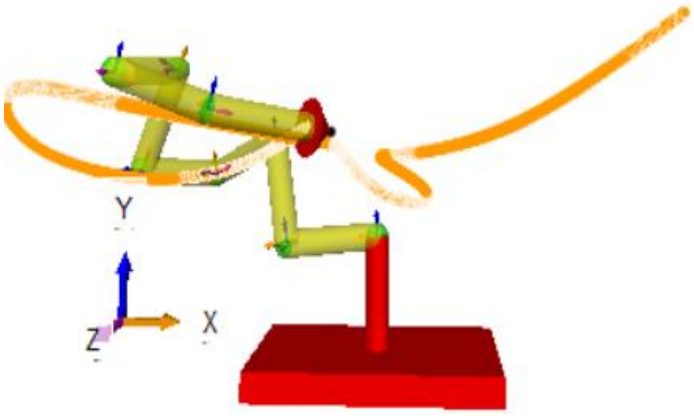


Şekil 4.22 : Altı serbestlik dereceli manipülatörün [-1.5, 4, 4] m pozisyonuna ulaşımı.

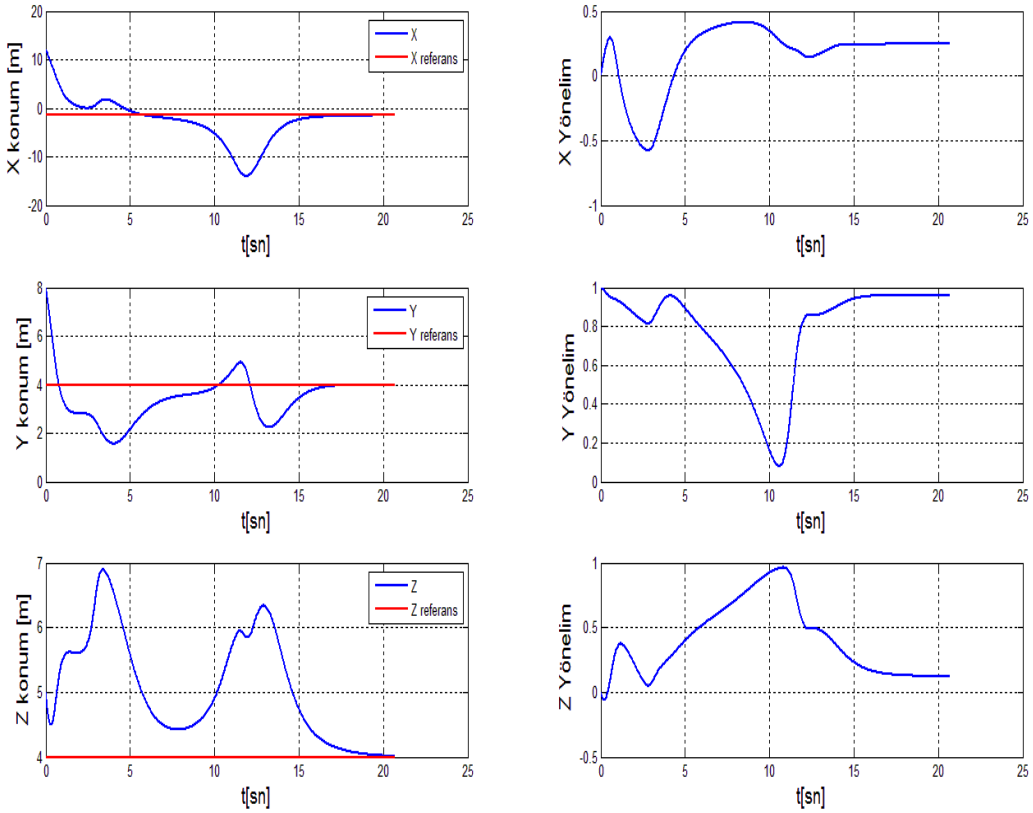


Şekil 4.23 : Şekil 4.22'de bulunan manipülatörün uç işlevcisinin pozisyon ve yönelim grafik çıktısı.

Şekil 4.24 Sekiz serbestlik dereceli manipülatörün [-1.5, 4, 4] m referans konumuna ters kinematik algoritmasıyla yaklaşımının grafiğini göstermektedir. Şekil 4.25, izlenen bu yörünge sırasında uç işlevcinin konum ve yönelim bilgilerinin grafik çıktılarını göstermektedir.

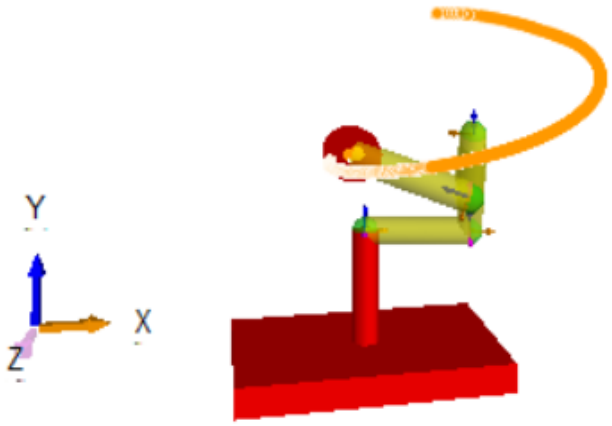


Şekil 4.24 : Sekiz serbestlik dereceli manipülatörün [-1.5, 4, 4] m pozisyonuna ulaşımı.

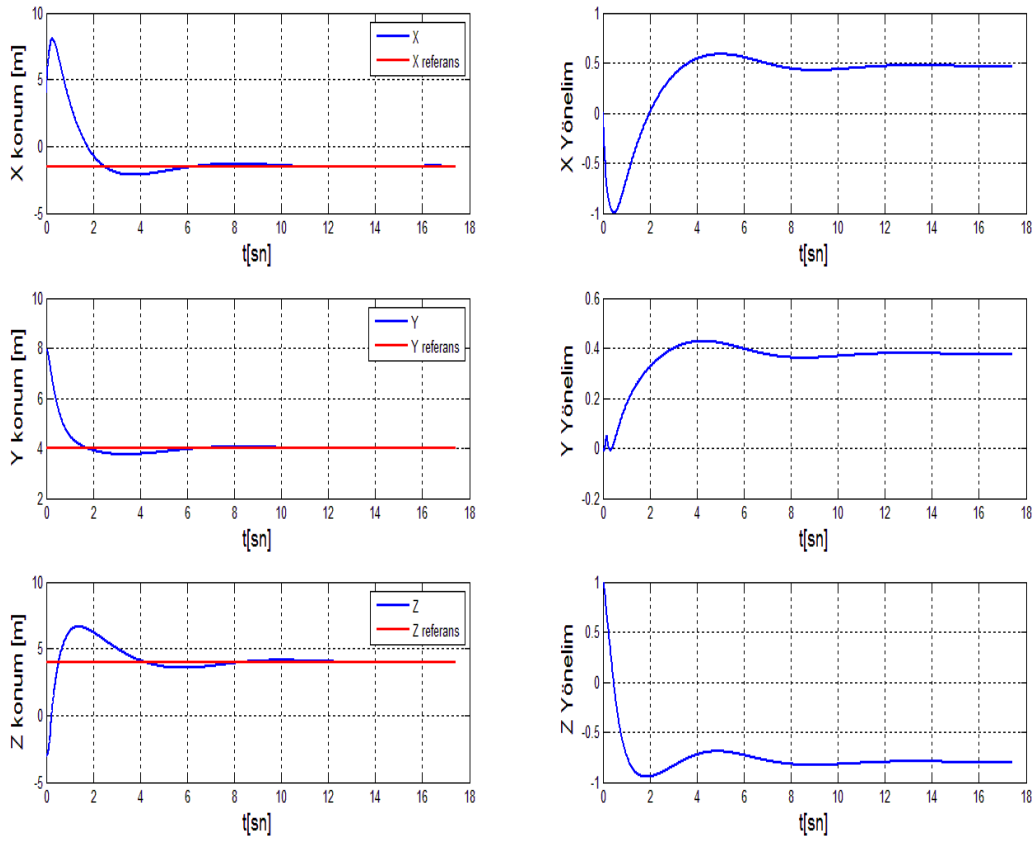


Şekil 4.25 : Şekil 4.24'de bulunan manipülatörün uç işevcisinin pozisyon ve yönelim grafik çıktısı.

Şekil 4.26 dört serbestlik dereceli manipülatörün [-1.5, 4, 4] m referans konumuna ters kinematik algoritmasıyla yaklaşımının grafiğini göstermektedir. Şekil 4.27, izlenen bu yörünge sırasında uç işlevcinin konum ve yönelim bilgilerinin grafik çıktılarını göstermektedir.



Şekil 4.26 : Dört serbeslik dereceli manipülatörün [-1.5, 4, 4] m pozisyonuna ulaşımı.



Şekil 4.27 : Şekil 4.26'da bulunan manipülatörün uç işlevcisinin pozisyon ve yönelim grafik çıktısı.

Yukarıdaki grafiklerden algoritmanın 3 farklı manipülatör türü için robust çalıştığı gözlenmektedir.

5. DEĞERLENDİRME

Tezde uzaysal vektör cebri yöntemiyle ters kinematik modelleme anlatılmış ve kinematik modelleme için gerekli temel bilgiler verilmiştir. Ardından kinematik modellemenin platform üzerine uygulamasını gösterebilmek için iki ve altı serbestlik dereceli seri topolojideki modeller kullanılmıştır. Ayrıca kinematik ve dinamik modellemede önemli bir yer tutan Jakobyen matrisin bu yöntemde nasıl oluşturulduğu tanıtılmıştır.

Üçüncü bölümde bu algoritmanın yazılım kısmını oluşturan Vpython anlatılmıştır. Önce programın temelini oluşturan Python yazılımı anlatılıp ardından Vpython tanıtılmıştır. Sonra da Vpythonda tez çalışması kapsamında yazılım kısmı için kullanılan modüller, görüntü kısmı için kullanılan araç kutuları tanıtılıp programın grafiksel arayüz tasarımı için sunduğu "Tkinter" modülüne geçiş yapılmıştır. Bu modül kullanılarak bir grafik arayüz tasarlanmıştır. Tasarlanan arayüz, yine aynı bölümde gösterip örnek çalışmalar yapılmıştır.

Son bölümde ters kinematik algoritmasının serbestlik derecesi farklı seri topolojideki manipülatörler için robust çalışıp çalışmadığını test etmek için eksik, tam ve artık manipülatör modelleri kullanılıp bu manipülatörler üzerine aynı referans girişleri verilerek simülasyon çalışmaları yapılmıştır. Giriş referans değerleri, kullanıcı tarafından görüntü ekranına fare ile tıklanılan yerdir. Vpythonda görüntü ekranı üç boyutludur. Bu yüzden ekranda tıklanılarak referans olarak verilen konum bilgiside üç boyutlu vektör yapısındadır. Sonuçlar ve Ek-A bülümlerinde bulunan simülasyon grafik çıktılarından algoritmanın verilen 0.01 cm hata toleransına iyi sonuç verdiği gözlenmektedir.

Bu tez, ileride uzaysal vektör cebri kullanılarak gerçekleştirilecek tüm kinematik ve dinamik modellemeler için Matlab'a alternatif bir yazılım ortamı sağlaması anlamında önemli bir başlangıçtır.

KAYNAKLAR

[1] Hartenberg, S. R. and Denavit, J. (1964). *Kinematic synthesis of linkages*. McGraw-Hill.

[2] Rodriguez, K. K. and Jain, A. (1991). A Special Operator Algebra for Manipulator Modeling and Control. *International Journal of Robotic Research*, 10, (371-381)

[3] Rodriguez, A. J. and Delgado, K. (1992). Spacial Operator Algebra for Multibody System Dynamics, *The Journal of The Astronautical Sciences*, Jan-March

[4] Ohwovoriole, M. and Roth, B. (1981). An Extension of Screw Theory, *Journal of Mechanical Design*, 103, 725-735

[5] Featherstone, R. (1983). The Calculation of Robot Dynamics Using Articulated-Body Inertias, *The International Journal of Robotics Research,* **2**, 13-30 doi: 10.1177/027836498300200102

[6] Gosselin, C. and Angeles, J. (1990). Singularity analysis of closed-loop kinematic chains, *Robotics and Automation, IEEE*, **15,** 281 – 290, doi: 10.1109/70.56660

[7] Jain, A .(1991). Unified formulation of dynamics for serial rigid multibody systems, *Journal of Guidance, Control, and Dynamics,* Vol. 14, No. 3 (1991), pp. 531-542.

[8] Jain, A. And Rodriguez, G. (1993). An analysis of the kinematics and dynamics of underactuated manipulators, *Robotics and Automation, IEEE,* 411 – 422

[9] Jain, A. And Rodriguez, G. (1995). Diagonalized Lagrangian robot dynamics Robotics and Automation, *IEEE*, 571 – 584

[10] Rodriguez, G. Jain, A. and Kreutz, K. (1992). Spatial operator algebra for multibody system dynamics, *The Journal of the Astronautical Sciences*, vol. 40, pp. 27-50,

[11] Yeşiloğlu, S. M., Temeltas, A. (2010). Dynamic Modelling of Cooperation Underactuated Manipulators for Space Manipulation, *Advanced Topics*, Vol. 3.No. 3, s. 325-341, ISSN: 0169-1864,

[12] Yeşiloğlu, S. M., (2007). High Performance Dynamical Modelling of Complex Topology Systems, PhD thesis, Istanbul Technical University, Turkey.

[13] Iglev, O. and Graser, A. (1997). An Analytical Method for the Inverse Kinematics of Redundant Robot, *Intelligent Otomotion and Active Systems Bremen*, (15-17)

[14] Tolani, D. Goswami, A. and Badler, N. (2000). Reel-Time Inverse Kinematics Techniques for Anthropomorphic Limbs, *Graphical Models*, 62, (353-388)

[15] Aristidou, A. and Lasenby, J. (2009). Inverse Kinematics: A Review of Existing Techniques and Introduction of a New Fast Iterative Solver, Technical Report, Cambridge University

[16] Komura, T., Shinagawa, Y. and Kunii, T. L. An Inverse Kinematic Method Based On Muscle Dynamics**,**

[17] Buss, S. R. (2009). Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Square Method,

[18] Joubert, N. and Berkeley, U. (2008). Numerical Methods for Inverse Kinematics.

[19] Castellet, A. (1998). Solving Inverse Kinematics Problems Using and Interval Method, Tesi Doctoral, Universitat Politecnica de Catalunya

[20] Wang, C. T. and Chan, C. C. (1991). A Combined Optimization Method for Solving the Inverse Kinematics Problem of Mechanical Manipulation, *Ieee Transcation On Robotics And Otomotion*, 7, 4

[21] Scherer, D. Dubois, P. And Scherwood, B. (2000). Vpython: 3D Interactive Scientific Graphics for Students, *Computer Science Engineering*, 5, 56-62

[22] van Rossum, G., & de Boer, J. (1991). Interactively testing remote servers using the Python programming language. *CWI Quarterly*, *4*(4), 283-303.

[23] Sands, D. (2010). First year mechanics taught through modelling in Vpython, *New Directions*, 6, 47-50, doi: 10.11120

[24] Wochal, M. Cekus, D. and Warys, P. (2012). The application of VPython to visualization and control of robot.

[25] Pedregosa, F. Ve diğerleri (2011). Scikit-learn: Machine Learning in Python, *The Journal of Machine Learning Research***,** 12, 2825-2830

[26] Cock, P. J. At all (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics, Science & Mathematics, Bioinformatics, volume 25, 11, 1422-1423

[27] Clavel, R. (1990). Device for the Movement and Positioning of an Element in Space, *Patent* No: US 4976582 A tarih:11.12.1990

[28] Anlı, E., Yurt, S., Özkol, İ. (2005). Paralel Mekanizmalarin Kinematiği, Dinamiği ve Çalişma Uzayi, Havacilik ve uzay teknolojileri dergisi, Cilt 2, sayı. 1, Sf. 19-36.

[29] Fichter, E. F. (1986). A Stewart platform-based manipulator: general theory and practical construction, *The International Journal of Robotics Research*, vol. 5 no. 2 157-182

[30] Url-1 <*http://new.math.uiuc.edu/math198/MA198-2009/lee522/#design* >

ÖZGEÇMİŞ

Adı Soyadı : Ece COŞGUN

Doğum Yeri ve Tarihi : Adana – 02.11.1988

E-Posta : ecosgun@itu.edu.tr

ÖĞRENİM DURUMU :

- **Lisans** : 2011, Kocaeli Üniversitesi, Mekatronik Mühendisliği