PARALEL KİNEMATİK SİSTEMLİ 6 AKTÜATÖRLÜ (HEXAPOD) CNC TASARIM İMALAT VE KONTROLÜ

Eyüp FINDIKLI

Kütahya Dumlupınar Üniversitesi
Lisansüstü Eğitim Öğretim ve Sınav Yönetmeliği Uyarınca
Fen Bilimleri Enstitüsü Makine Mühendisliği Anabilim Dalında
YÜKSEK LİSANS TEZİ
Olarak Hazırlanmıştır.

Danışman: Dr. Öğr. Üyesi Ağah AYĞAHOĞLU

Temmuz – 2019

KABUL VE ONAY SAYFASI

Eyüp FINDIKLI tarafından hazırlanan "Paralel Kinematik Sistemli 6 Aktüatörlü (Hexapod) CNC Tasarım İmalat ve Kontrolü" adlı tez çalışması, aşağıda belirtilen jüri tarafından Kütahya Dumlupınar Üniversitesi Lisansüstü Eğitim Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek OY BİRLİĞİ ile Kütahya Dumlupınar Üniversitesi Fen Bilimleri Enstitüsü Makine Mühendisliği Anabilim Dalında, YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

09/07/2019

Prof. Dr. Önder UYSAL
Enstitü Müdürü, Fen Bilimleri Enstitüsü

Prof. Dr. Ramazan KÖSE
Anabilim Dalı Başkanı, Makina Mühendisliği Anabilim Dalı

Dr. Öğretim Üyesi Ağah AYĞAHOĞLU
Danışman, Makina Mühendisliği Anabilim Dalı

Sınav Komitesi Üyeleri

Dr. Öğretim Üyesi Ağah AYĞAHOĞLU
Makina Mühendisliği Bölümü, Kütahya Dumlupınar Üniversitesi

Dr. Öğretim Üyesi Abdullah KEÇECİLER
Endüstri Mühendisliği Bölümü, Kütahya Dumlupınar Üniversitesi

Dr. Öğretim Üyesi Sezcan YILMAZ
Makina Mühendisliği Bölümü, Eskişehir Osmangazi Üniversitesi

ETİK İLKE VE KURALLARA UYGUNLUK BEYANI

Bu tezin hazırlanmasında Akademik kurallara riayet ettiğimizi, özgün bir çalışma olduğunu ve yapılan tez çalışmasının bilimsel etik ilke ve kurallara uygun olduğunu, çalışma kapsamında teze ait olmayan veriler için kaynak gösterildiğini ve kaynaklar dizininde belirtildiğini, Yüksek Öğretim Kurulu tarafından kullanılmak üzere önerilen ve Kütahya Dumlupınar Üniversitesi tarafından kullanılan İntihal Programı ile tarandığını ve benzerlik oranının %5 çıktığını beyan ederiz. Aykırı bir durum ortaya çıktığı takdirde tüm hukuki sonuçlara razı olduğumuzu taahhüt ederiz.

Dr. Öğr. Üyesi Ağah AYĞAHOĞLU

Eyüp FINDIKLI

PARALEL KİNEMATİK SİSTEMLİ 6 AKTÜATÖRLÜ (HEXAPOD) CNC TASARIM İMALAT VE KONTROLÜ

Eyüp FINDIKLI
Makina Mühendisliği, Yüksek Lisans Tezi, 2019
Tez Danışmanı: Dr. Öğretim Üyesi Ağah AYĞAHOĞLU

ÖZET

Paralel kinematik sistemli 6 aktüatörlü hexapod robot aynı zamanda Stewart Platformu diye de adlandırılır. Paralel yapılı sistemler, aynı maksimum yükü taşımak üzere oluşturulmuş olan seri yapılı bir sistemden daha yüksek bir rijitliğe ve daha düşük bir kütlesel atalet momenti değerine sahiptirler. Bu yüzden uçak ve araç simülatörleri, yüksek hassasiyetli takım tezgâhları ve işleme merkezleri, kazı makinaları gibi birçok alanda kullanım imkânı bulunmaktadır. Bunlara karşın, paralel mekanizmaların çalışma uzayları seri mekanizmalara kıyasla dardır, kısıtlı yönelim aralığına sahiptirler ve ileri kinematik hesabında mekanizmanın paralelliğinden dolayı zorluklarla karşılaşılmaktadır.

Stewart Platform mekanizması temel olarak kendi içinde seri 6 adet lineer aktüatör ve biri sabit diğeri hareketli olmak üzere iki platformdan oluşmaktadır. Hareketli ve sabit platform 6 noktasından aktüatörler ile birbirine bağlanmıştır. Lineer aktüatörler aynı zamanda uzuv işlevi görmekte olduğu bu yapıda, aktüatörler bir taraflarından sabit diğer taraflarından ise hareketli platforma küresel mafsallarla bağlanmıştır.

Bu çalışmada deneysel bir 6 aktüatörlü hexapod CNC sistemi tasarımı, imali ve kontrolü gerçekleştirilmiştir. Robotun mekanik tasarımı, kinematiği, analizi ve benzetimi bilgisayar ortamında Catia V5 CAD programında yapılmıştır. Sistemin statik analizi ise Ansys Workbench programında yapılmıştır. Sistemin ileri ve ters kinematik formüllerinin matematiksel ifadeleri çıkarılmıştır. Bu matematiksel ifadeler, Delphi ve Python dilleri ile modellenerek bilgisayar tabanlı kontrol yazılımı geliştirilmiştir. Bu yazılım sayesinde, geliştirilen mekanizma bilgisayar marifetiyle kontrol edilmeye çalışılmıştır.

Sonuç olarak, sanayide yaygın olarak CNC makinelerin kontrolünde kullanılan ISO G kodlarının, geliştirilen yazılım vasıtası ile yorumlanıp makinenin kontrolü sağlanmıştır. Elde edilen veriler doğrultusunda çok eksenli robotların çalışma hacmini daha iyi tanımlayabilecek "fındık" noktası tanımı geliştirilmiş, tasarım iyileştirmelerinin ortaya çıkması sağlanmış, matematik altyapının fiziksel yapıyla eşzamanlı çalışmasının doğrulukları araştırılmıştır.

Anahtar Kelimeler: CNC, Çalışma Hacmi, Stewart Platform.

DESIGN AND MANUFACTURING OF 6 ACTUATOR HEXAPOD CNC WITH PARALLEL KINEMATICS AND ITS CONTROL

Eyüp FINDIKLI
Mechanical Engineering, M.S. Thesis, 2019
Thesis Supervisor: Assist. Prof. Dr. Ağah AYĞAHOĞLU

SUMMARY

The hexapod structure, which has 6 actuators and parallel kinematic robot is also known as a Stewart Platform. Parallel systems have a higher rigidity and a lower mass moment of inertia than a serial systems designed to carry the same maximum load. Therefore, it is possible to use this system in many areas such as aircraft and vehicle simulators, high precision machine tools, machining centers and excavation machines. On the other hand, the working spaces of parallel mechanisms are narrow compared to serial mechanisms, they have limited orientation range and difficulties are encountered due to the parallelism of the mechanism in advanced kinematics calculation.

Stewart Platform mechanism consists of 6 linear actuators in series and two platforms, one fixed and one movable. The movable and fixed platform is connected to each other by actuators at point 6. In this structure, where the linear actuators also function as a limb, the actuators are connected from one side to the fixed platform and from the other side to the mobile platform by spherical joints.

In this thesis, the design, production and control of the experimental hexapod CNC system with 6 actuators was performed. The mechanical design, kinematics, and simulation of the robot are conducted in the computer environment with the Catia V5 CAD application. The structural analysis of the robot are calculated in the computer environment with the Ansys Workbench application. Mathematical expressions of forward and inverse kinematic formulas of the system have been extracted. Computer based control software was developed by modeling these mathematical expressions with Delphi and Python languages. With this software, the mechanism developed was tried to be controlled by computer.

As a result, ISO G codes commonly used in the control of CNC machines in the industry are interpreted by means of the developed software and the control of the machine is provided. According to the data obtained, the definition of “fındık” point, which can better define working volume of multi-axis robots, was proposed, design improvements were provided, and the mutuality of the mathematical infrastructure with mechanical structure was investigated.

Keywords: Stewart Platform, CNC, Working Volume.

TEŞEKKÜR

Yüksek lisans eğitimim boyunca danışmanlığımı yürüten ve gerek eğitimim gerekse yüksek lisans tezimin hazırlanışı sırasında gösterdiği kolaylıklar ve rehberliği, tecrübelerini paylaşarak bana yol gösterdiği için değerli hocam Dr. Öğr. Üyesi Ağah AYĞAHOĞLU'na teşekkürlerimi sunarım.

Ayrıca yüksek lisans eğitimim sırasında bana yardımcı olan, başta değerli hocalarım Arş. Gör. Kemal Cem KÖSE ve Prof. Dr. Mustafa Arif ÖZGÜR, Dr. Öğr. Üyesi Feridun KARAKOÇ'a, başta Bölüm Başkanımız sayın Prof. Dr. Ramazan KÖSE olmak üzere Makine Mühendisliği Bölümü hocalarıma teşekkürü bir borç bilirim.

Son olarak, anneme, babama ve kardeşlerime bana verdikleri destek ve anlayışları için ve yüksek lisans eğitimim boyunca destekleriyle yanımda olan özellikle tez hazırlanışı sürecinde gösterdiği sabır ve desteğiyle bana güç veren çok değerli eşime sonsuz minnettarlığımı sunarım.

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER (devam)

Sayfa

ŞEKİLLER DİZİNİ

Şekil **Sayfa**

ŞEKİLLER DİZİNİ (devam)

Şekil **Sayfa**

ÇİZELGELER DİZİNİ

Çizelge **Sayfa**

SİMGELER VE KISALTMALAR DİZİNİ

Simgeler	**Açıklama**
CNC	Computer Numerik Control
CPU	Central Processor Unit
ÇH	Çalışma Hacmi
DOF	Degree of freedom
G kod	(RS-274) ISO G kod
GL	Grafical Library
GPU	Grafical Processor Unit
IMU	Inertial Mesurement Unit
J	Jacobian Matrix
JM	Jacobian Matrix
kW	Kilo watt
N.m	Newton metre
NC	Numerik Control
OS	Operating System
PC	Personel Computer
rpm	Revolation per minute
RT	Real Time
SP	Stewart Platform
WS	Work Space
α	Global matrisdeki roll açısı
β	Global matristeki pitch açısı
γ	Global matristeki yaw açısı

1. GİRİŞ

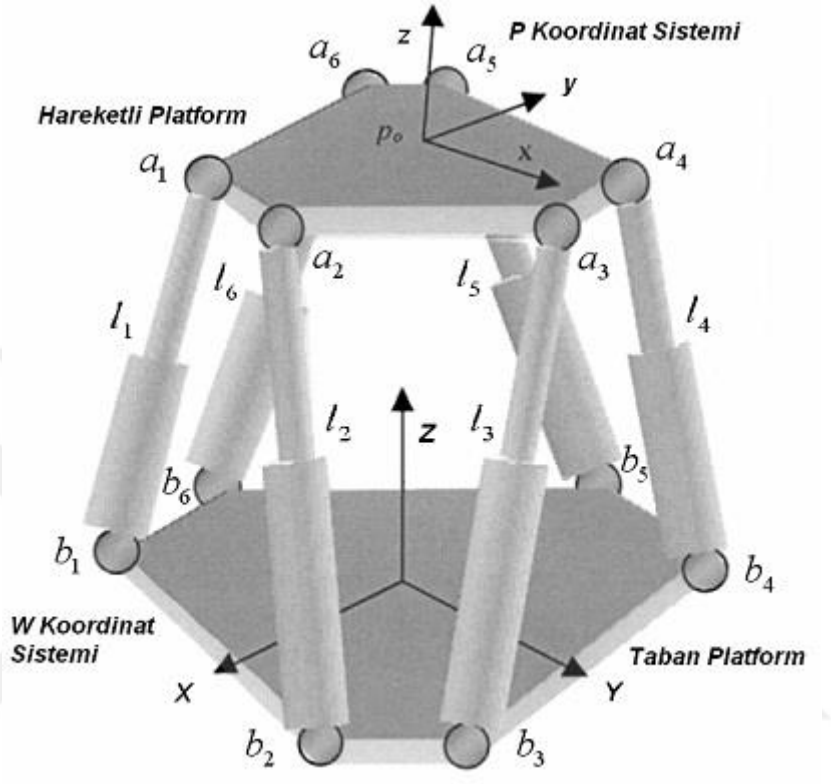
Stewart Platform aynı zamanda "Gough-Stewart" Platform olarakta bilinir. İlk olarak 1956 yallımda Gough tarafından tekerlek test mekanizması için tasarlanmıştır ve daha sonra bu mekanizmayı 1965 yılında stewart uçak simülatörü olarak kullanmıştır. O zamanlardan bu yana birçok imalat ve robotik uygulamalarda faydalanıldığından önemli bir araştırma alanı olarak ilgi çekmiştir. Örneğin Hunt platformu bir paralel robot kolu olarak kullanmayı önermiş, (Hung, 1978) Geng ve Haynes ise deneysel titreşim izolatör aleti tasarımında kullanmayı düşünmüşlerdir (Sunguray vd., 2014).

Şekil 1.1. Ticari bir hegzapod işleme merkezi olan Okuma PM-600 makinesi (Okuma, 2012).

Birçok ticari işleme merkezi üretici firma stewart platformunu CNC tezgâhı olarak tasarlayarak ürünler geliştirmiştir. Örnek olarak, Sekil 1.1 de Okuma firmasının PM-600 adlı işleme merkezi görülmektedir.

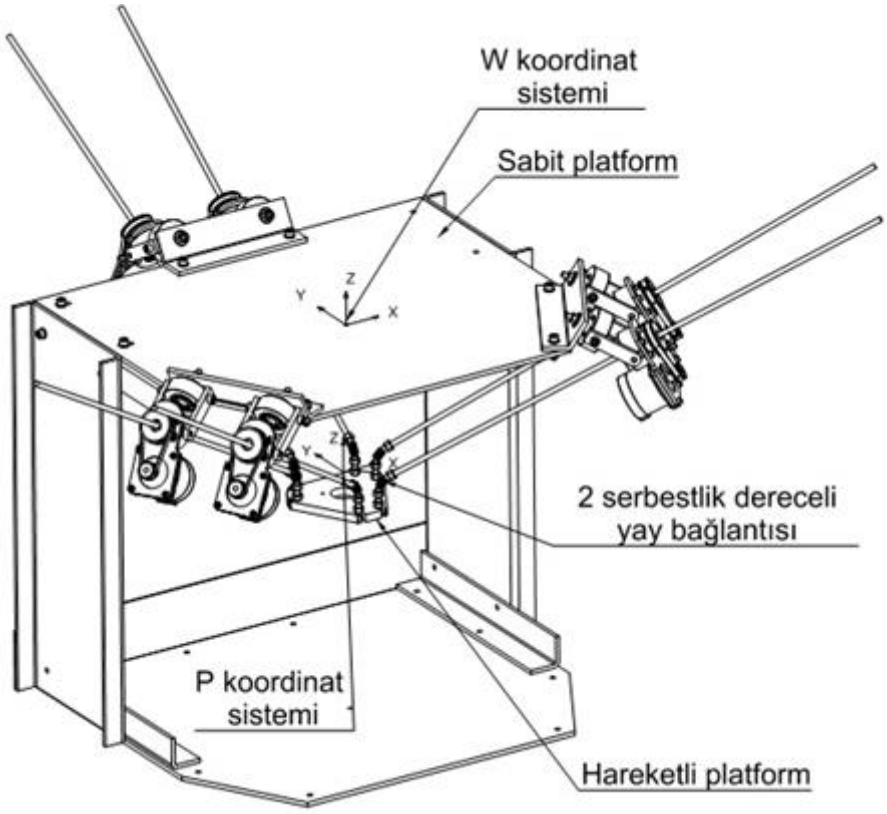
Şekil 1.2 de gösterildiği gibi hexapod CNC makinesi, 6 adet lineer kolun hareketli ve sabit platformlara, 2 serbestlik dereceli bağlantılarla paralel olarak bağlanmasıyla oluşturulur. Bu sayede hareketli platform, sabit platform üzerindeki bir eksen takımına göre 3 dönme ve 3 öteleme

hareketi olarak toplam 6 serbestlik derecesine sahip olmuş olur. Şekil 1.3 de SP mekanizmasının şematiği görülmektedir.



Şekil 1.2. Stewart Platform mekanizmasının şematik gösterimi.

Kolların tahrik yöntemi hidrolik, pnömatik veya elektriksel olabilir. Bu çalışamada aktüator kolunun uzayıp kısalma hareketini ise, step motor tahrikli bir somun trigger kayışla döndürülerek içerisindeki vidalı mille aktarılmıştır.



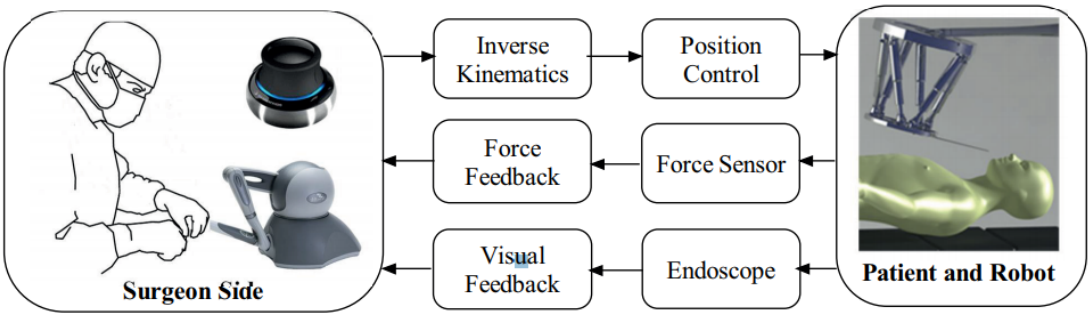
Şekil 1.3. Geliştirilen Stewart Platform tabanlı CNC makinesi.

2. LİTERATÜR ARAŞTIRMASI

Stewart platformu mekanizmasının ileri ve ters yönlü kinematik denklemleri genel olarak çözümlenmiştir ve birçok çalışmada çıkartılmıştır (Merlet, 2006). Bu platform hali hazırda artırılmış gerçeklik simülatörleri, takım tezgâhları, iş makinaları, savunma ve robotik cerrahi ekipmanları gibi alanlarda kullanılmaktadırlar. Günümüzdeki metal işleme teknolojileri sayesinde paralel robotlarda kullanılabilecek parçalar tasarlanıp üretilebilir hale gelmiştir. Bir robotun ana güç iletim organları olarak hassas ve boşluksuz redüktörler kullanılması halinde aktüatörler geri gelme hatalarından arındırılmış olur.

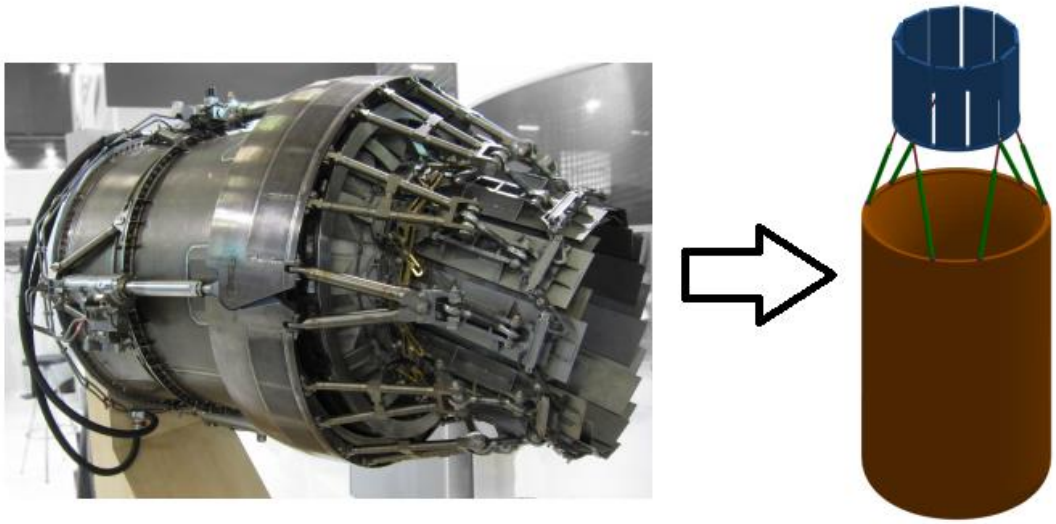
Yapılan araştırmalar sonucunda stewart platformu ve türevi paralel kinematik sistemli robotlar ile alakalı bazı çalışmalar aşağıdaki gibidir.

Stewart platformunun uç eyleyicisi x, y, z eksenlerinde doğrusal ve döner hareketleri belirli bir düzeyde yapabilme kabiliyetiyle 6 serbestlik derecesine sahiptir. Kiriz ve Bingül yaptıkları çalışmada, karmaşık cerrahi operasyonları tamamen robotlu otomasyon edilebilmek için, stewart platformunun hassas cerrahi müdahalerde kullanımı için sistem simülasyonu geliştirmiştir. İlgili çalışmanın şematiği Şekil 2.1'deki gibidir (Kizir ve Bingül, 2019).



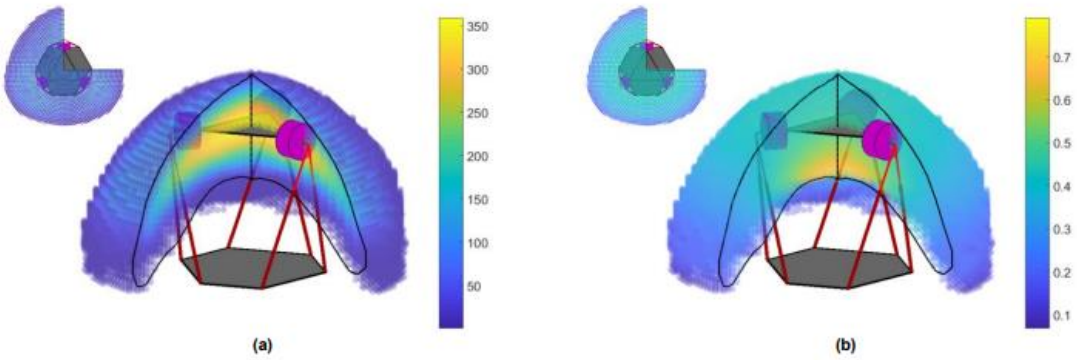
Şekil 2.1. Robotik cerrahi sisteminin genel yapısı (Kiriz, Bingül, 2019).

Başka bir çalışmada ise Neeraj ve arkadaşları, jet motoru yönlendirme nozulu kontrol sistemi olarak, yine bir Stewart Platformu tasarlamıştır. Geliştirilen sistem Şekil 2.2'deki gibidir (Neeraj vd., 2018).



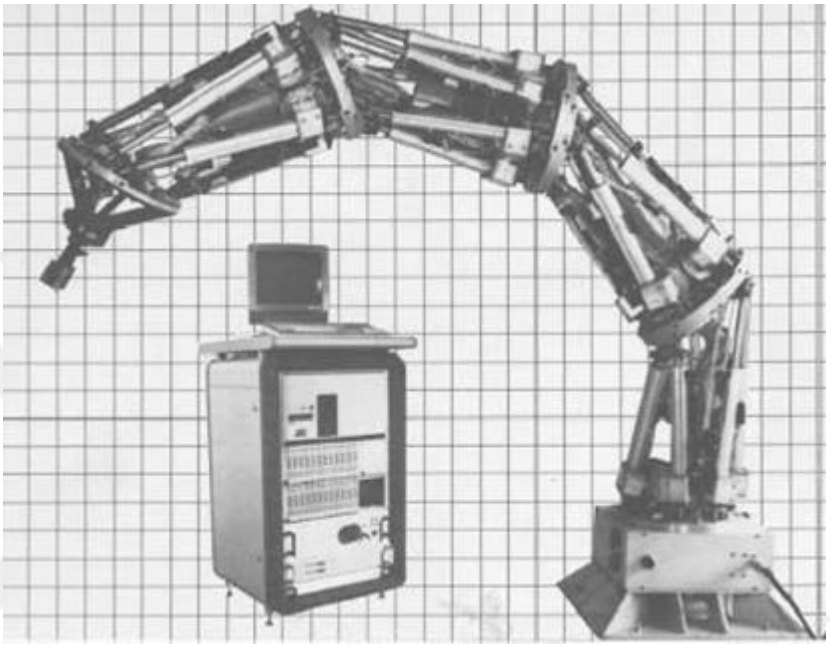
Şekil 2.2. EJ200 jet motoru yönlendirme nozulu yerine tasarlanan SP'li nozul sistemi (Neeraj vd., 2018).

Başka bir çalışmada McCann ve Dollar, Stewart Platformundan esinlenilmiş bir robot el tasarımı gerçekleştirmiştir. Belirli kriterler le optimum bir tasarıma ulaşılmaya çalışılmıştır. Bu tez çalışmasındaki robotların son eyleyicilerinin konfigürasyon hacmini daha iyi tanımlanması için geliştirilen, fındık noktaları benzer şekilde son eyleyici oryantasyonlarıda kapsayacak şekilde ele alınıp araştırılmıştır. Fakat bu tez çalışmasındaki fındık noktaları hem çalışma zamanındaki kullanışlılığından hem de daha açıklayıcı bilgiler içerdiğinden daha kapsamlıdır. İlgili robot elin çalışma hacmini temsil eden görsel Sekil 2.3'deki gibidir. (McCann ve Dollar, 2018) (a) Her noktadaki ulaşılabilinen 370'den fazla test edilen oryantasyon sayısı. (b) Local Transmission Index (LTI). Siyah çizgiler hacmin çeyreğinden alınan kesiti temsil eder.



Şekil 2.3. McCann ve Dolların çalışması sonucunda toplam ulaşılabilinen oryantasyona ve LTI ye göre SP nin çalışma hacmi (McCann ve Dollar, 2018).

Stewart platformların seri olarak bağlanmasından oluşan logabex robotu, Modeling, Identification and Control of Robots kitabı 8. bölümünde paralel robotlar için genel tanımlamalar yapılmış, logabex robotu hakkında bilgi verilmiştir. Logabex robot görseli sekil 2.4'deki gibidir (Khail ve Dombre, 2002).



Şekil 2.4. Logabex robotu LX4 model (Khail ve Dombre, 2002).

Bu tez çalışmanın başlarında mekanizmayı test etmek için linuxcnc (linuxcnc.org), sistemin konseptini doğrulamak amacıyla kontrol sistemi olarak kullanılmıştır. Daha sonra linuxcnc programı terk edilerek yeni kontrol yazılımı geliştirilmiştir. Linuxcnc tabanlı CNC hexapod robotları da geliştirilmiştir. Buna örnek olarak bu sistemli bir hexapod robotun ahşap iş parçası üzerinde işleme anına ait görseli şekil 4.5'deki gibidir (pkmcnc, 2012).

Şekil 2.5. PC tabanlı kontrolcü ile kontrol edilen hexapod robotun ahşap işleme anına ait görseli (pkmcnc, 2012).

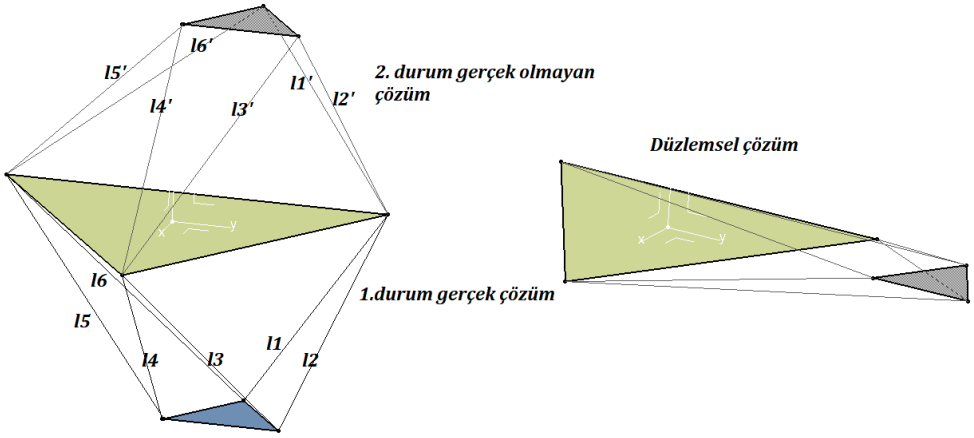
SP bir robot olarak kullanılmasının yanı sıra, herhangi bir 6 DOF bir robotu uzaktan kontrol edebilecek girdi mekanizması olarak ta kullanılabilir. Böyle bir çalışmada hareketli platform sabit platforma lineer sensörlerle bağlanır. Bu sensörler vasıtası ile hareketli platformun konum, yönelim hız bilgileri uzaktan kontrolü sağlanan cihaza aktarılır. Bu bağlamda Kim ve arkadaşları yaptıkları bir çalışmada Stewart platform tabanlı, gerçek zamanlı hareket takip sistemi geliştirmiştir (Kim vd., 2019).

Stewart platformunun konumu ve bulanık empedans-kuvvet kontrolü çalışmasında konum ve yörünge kontrolünün yanında kuvvet ve empedans kontrolünü de ele alan SP için etkin bir kontrol yöntemi sunulmuştur. Bu çalışmada SP kontrolü 3 kısma ayrılarak 1- Serbest uzayda konum kontrolü, 2- Temas halinde empedans kontrolü, 3- Kuvvet kontrolü incelenmiştir. Türetilen bulanık model filtre kazançları geliştirilen teknikle kinematik ve dinamik model benzetim ortamında denemiş ve arzu edilen performans değerleri elde edilmiştir (Kizir ve Bingül, 2012).

Branch ve arkadaşları tarafından geliştirilen bir diğer sistem ise bir ivme algılayıcı IMU sensörü vasıtası ile mevcut bir stewart robotun kontrol edilmeye çalışılmıştır (Branch vd., 2018).

3. 6 AKTÜATÖRLÜ (HEXAPOD) CNC KİNEMATİĞİ

Bu bölümde hexapod robotun ters ve ileri kinematiği üzerinde çalışılmıştır. Hexapod robotunun ters kinematiği (eyleyici konumundan uzuv boylarının bulunması) vektörel işlemler yapılarak bulunabilir ve bulunan çözüm gerçek çözümdür. İleri kinematiği ise (uzuv boylarından eyleyici konumunun bulunması) nispeten biraz daha karmaşıktır. Bunun nedeni eyleyicinin belirli bir konumunda 1 veya 2 adet gerçek çözümü vardır. Başka bir değişle son eyleyicinin herhangibir konum ve yöneliminde aktüatörlere bağlı olduğu kontrol noktaları sabit platforma göre uzunlukları belirli olmalıdır. Şekil 3.1 olası durumlar görülmektedir. 1. durum 2. durumun sabit platforma göre simetrik çözümüdür her iki durumda da koluzunlukları $ln' = ln$ eşittir. Düzelemsel durumda çözümün simetriği yoktur. Fakat 2.durum ve 3. durumlar genellikle fiziksel olarak ulaşılamayan durumlardır.



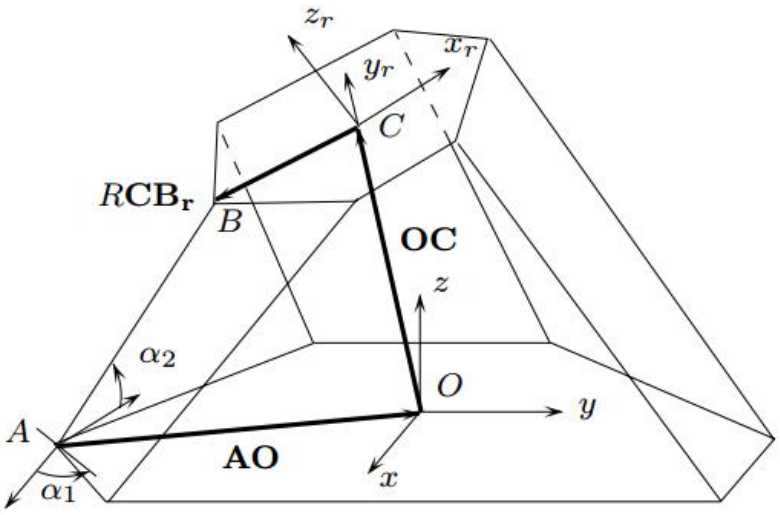
Şekil 3.1. Hexapod robotun çözüm durumları.

Kol uzunlukları iteratif metodla belirtilen toleransta çözüm vermeyecek şekilde verilmişse çözüme yakınsama sağlanamayabilir. Ayrıca mekaniksel tasarımda o an verilen konum ve rotasyon için kısıtlama yapıp yapmadığı araştırılmalıdır. Bu durumlarda uzuvların birbirlerine çarpıp çarpmadığı kontrol edilmelidir. Bu durumların üstesinde gelebilecek yöntemler, çalışma hacmi belirleme konusunda, önerilmiştir.

3.1. Hexapod CNC Ters Kinematiği

Ters kinematiğin hesaplanması temelde (Şekil 3.2)' deki gibi tek kol uzvu için genelleştirilmiş AB ye ait olan uzunluğunu bulmaktır. Hareket eden platformun sabit platforma

göre yönelimini ve konumunu bilmekle her bir kol uzvu için uzunluklar aşağıdaki genelleştirilmiş denkleme göre yazılabilir.



Şekil 3.2. Hexapod robotun ters kinematiğini oluşturmak için görsel (Merlet, 2006).

$$\mathrm{AB} = \mathrm{AO} + \mathrm{OC} + \mathrm{RCB}_r \tag{3.1}$$

Denklem 3.1, her bir kol uzunluğu veren AB vektörünü bulmak için uygulanabilir. OC hareketli platformun orijininin sabit platforma göre konumudur.Her bir kol için, AO ve CB_r kolların bağlı oldu hareketli ve sabit platform üzerindeki kontrol noktalarının bağlı olduğu platformalara göre konumlarıdır. Bu noktalar tasarımda seçilir ve hep sabittir. R ise hareketli platformun sabit platforma göre rotasyonudur.

$$\mathrm{AB} = \rho \mathrm{n} \tag{3.2}$$

Denklem 3.2 de her bir aktüatör AB vektörü için, ilgili kol uzunlu (ρ) ve (n) normalleri cinsinden de ifade edilebilir.

$$\rho_i = \left|\left|\mathrm{A}_i\mathrm{B}_i\right|\right| = ||\mathrm{A}_i\mathrm{O} + \mathrm{OC} + \mathrm{RCB}_{i_r}\,||\tag{3.3}$$

AO, CB, vektörleri sabit ve hareketli platforma göre konumlarıdır. Böylece tek kol uzuv için genelleştirilmiş AB vektörü bulunabilir. ρ_i kol uzunlukları denklem 3.3 dekigibi ifade edilebilir.

3.2. Hexapod Robotun İleri Kinematiği

Bu hexapod robotun ileri kinematik fonksiyonları “ilerikinematik.py” program dosyasında çözümlenmiştir. Daha sonra bu dosyadan elde edilen transformasyon matrisi “HexSimPos.exe” yazılımda kullanılmıştır.

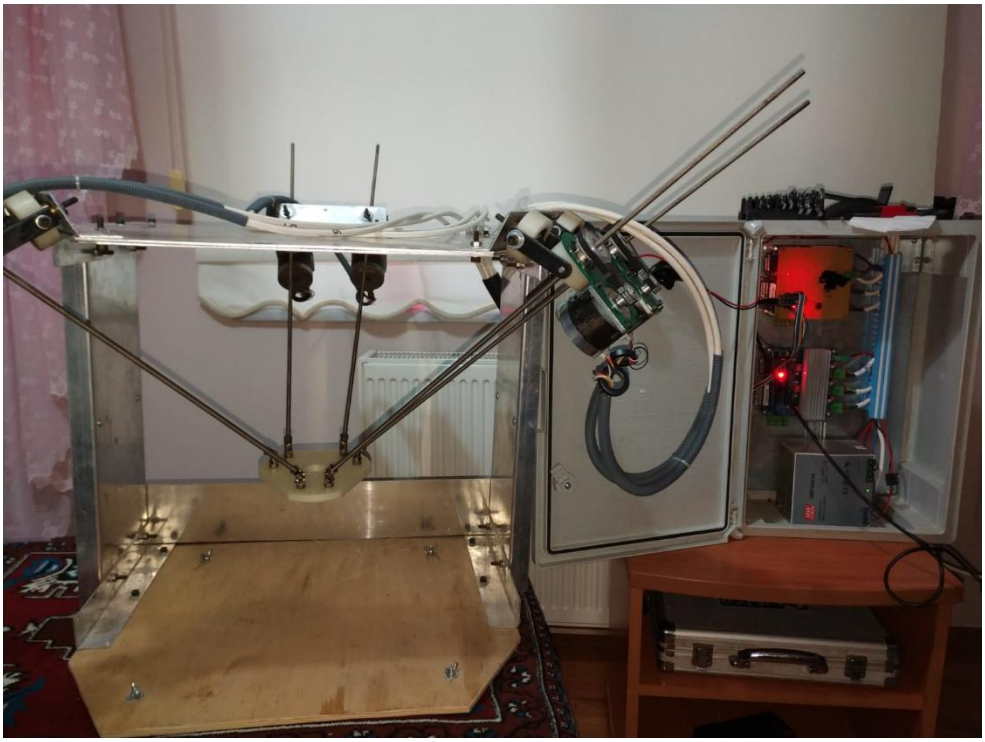
"ileriKinematik()" fonksiyonu bir iteratif algoritma kullanarak ileri kinematiği çözümler. Bu iteratif algoritmaların doğası gereği bir başlangıç değerlerine ihtiyaç duyarlar ve yaklaşık çözüme yakınsamaya çalışırlar. İleri kinematik için kol uzunluklarını alır ve hareketli eyleyicinin pozisyonunu ve rotasyonunu döndürür. Bu durumda birden fazla çözüm ortaya çıkar. "ileriKinematik()" fonksiyonu, verilen başlangıç değerine en yakın olan çözümlerden yalnızca birini döndürür. Verilen kol uzunluklarının tutarsızlığı çözümün yakınsamamasına ve çözümün bulunamamasına yol açabilir.

İleri kinematik için stewart platformun iterasyonal yöntem olan, numerik Newton raphson yöntemi kullanılarak çözümlenmesi tercih edilmiştir. Bunu sebebi analitik çözümlerin karmaşık ve birden fazla çözümlerinin olmasından dolayıdır. Bu çalışmada linuxcnc projesinde kullanılan ileri kinematik çözümlemeleri kullanılmıştır (linuxcnc, 2019).

4. 6 AKTÜATÖRLÜ (HEXAPOD) CNC TASARIMI

Yapılan bu çalışmada tasarlan hexapod robotun, matematiksel doğruluğunun kanıtlanması ve geliştirilen yazılımın G kod hareketlerini fiziksel olarak benzetim yapabilmesi amaçlanmıştır.

Mekanizma, Catia V5 programı kullanılarak modellenmiş ve üretim teknik resimleri oluşturulmuştur. Ansys Workbench programında da mekanizmanın kesme senaryosuna göre gerilme analizleri yapılmıştır.



Şekil 4.1. Geliştirlen hexapod CNC'nin mekanik ve elektronik sistemnin genel görünüşü.

Kolay taşınması için sistemin gövdesi AL 2024, 5 mm kalınlığında sac malzemeden tasarlanmıştır ve hafillik sağlanmıştır. Omuz bağlantilari, aktüatör hareketini sağlayabilmesi için yeniden tasarlanmıştır. Omuz uzvu tıpkı bir U bağlantı yapısı gibi 2 döner eksene sahiptir. Bu döner eksenler belirli bir noktada çakıştırılmıştır. Ve bu aktüatör hareketini sağlayan sonsuz vida bu noktadan geçecek şekilde tasarımı gerçekleştirilmiştir. Böylece aktüatör sistemi omuz yapısı üzerinde üniversal bir bağlantıya sahip olmuştur.

Aktüatör hareketini sağlaması için M6 paslanmaz sonsuz vidalı gijon kullanılmıştır. Omuz uzvunun üniversal baglı noktasına çift sıra bilyalı 3201 kodlu rulman ile yataklanan prinç

somun bu gijona hem destek hemde tahrik sağlamaktadir. Somunun ise üzerinde 40 disli 3M kayış kasnağı ile tahriklenmesi sağlanmıstır. Omuz yapısını Şekil 4.2 deki gibidir.

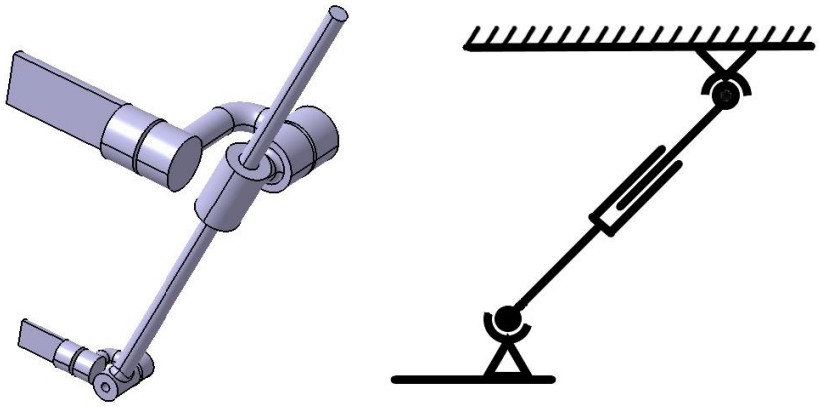
Şekil 4.2. Geliştirilen hexapod CNC'nin özgün tasarım omuz uzvu nun ara montajlı görünüşü.

Hareketli platformu aktüatörlere bağlayan bilek sistemi ise, hazır 6 x 6 mm mil deliğine sahip kardan mafsallar kullanılmıstır. Hareketli platform üzerine iş mili motoru veya herhangi bir son eyleyici takılabilmesi için ekstra deliklerle modellemistir. Hareketli platformun kendisi 3 boyutlu yazıcı marifetiyle PLA plastik malzemden üretilmiştir. Hareketli platformun ara montajlı hali şekil 4.3 deki gibidir.

Şekil 4.3. Geliştirilen hexapod CNC'nin hareketli platform ve bilek uzuvlarının ara montajlı görünüşü.

4.1. Mekanik Tasarım

Aktüatörlerin tasarımında sonsuz vida gijonlar, hareketli ve sabit platforma kardan mafsal bağlantılarla bağlanmıştır. Sabit platforma bağlı olan kardan mafsalın yapısı değiştirilerek istavroz uzvunun iç kısmına step motor tahrikli somun yataklamak suretiyle omuz eklemi elde edilmiş, sonsuz vida gijona uzalıp kısalabilen hareket yeteneği kazandırılmıştır. Hareketli kısma ise standart kardan mafsal kullanılarak bilek yapısı elde edilmiştir. Sonsuz vida gijon somunu step motora 1/2 oranında 3M triger kayış sistemi ile bağlanmıştır. Kayışlı sistemin tercih edilmesinin sebebi ise kayış gerginliği düzgün bir şekilde yapıldığı takdirde geri gelme (back lash) hatasının giderilebilmesindendir (Akbari ve Kheibari, 2013). Şekil 4.4 de bir kola ait kinematik diyagram görülmektedir.



Şekil 4.4. Tek kol mekanizmasının katı modeli ve şematiği.

Aktüatörlerin teorik hassasiyeti milin vida adımına step motorun bir turda aldığı adım sayısına step motor sürücünün mikro step oranına ve kayış kasnak sisteminde gelen orana bağlıdır. Bunu denklem 4.1 vasıtası ile formüle edilebilir.

$$Ah(mm) = \frac{h*o*\mu}{n} \tag{4.1}$$

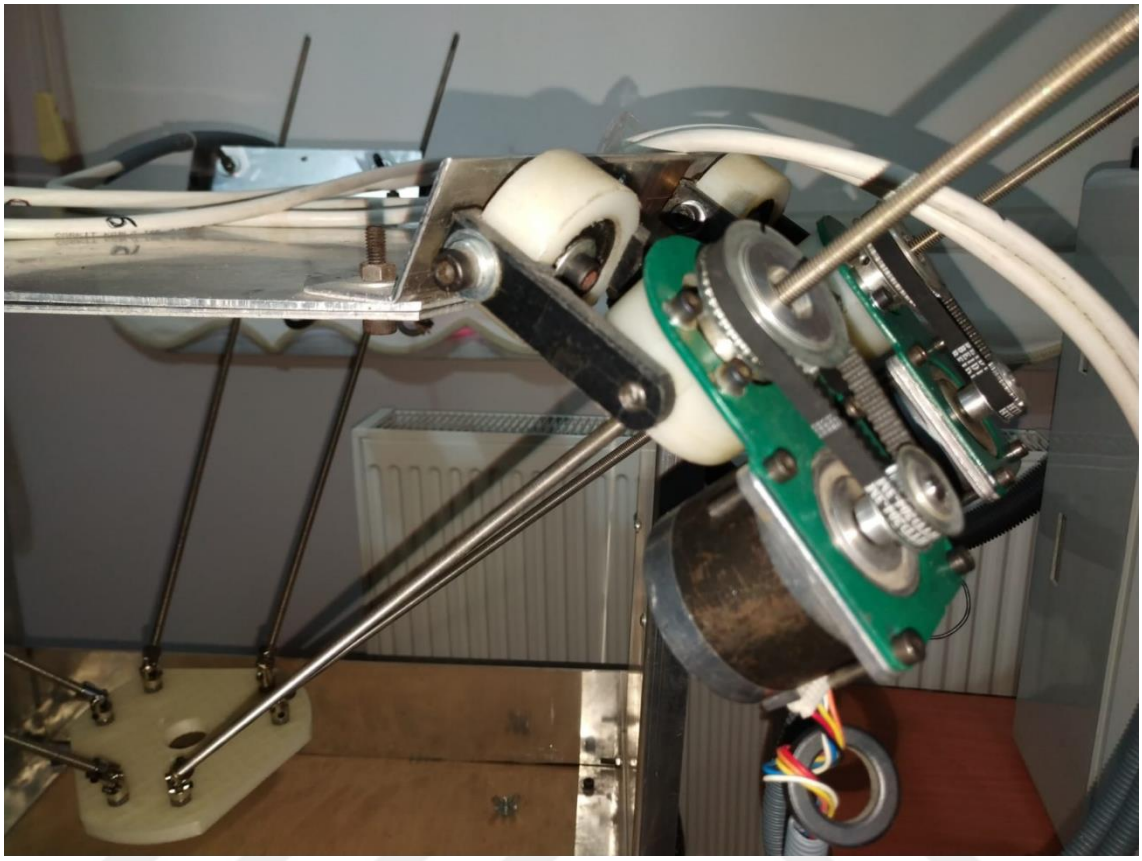
Burada; Ah: mm cinsinden aktüatör hassasiyeti, h step motor hatvesi, o 3M kayış kasnak aktarım oranı, μ step motor sürücü mikro step oranı, n step motorun 1 turdaki adım sayısı demektir.

Kullamlan mikro step oranı 1 dir ve step motorun bir turda yaptığı adımı 200 step/tur dur. Denklem 4.1 de değerleri yerine koyduğumuzda 0.0025 mm lik bir teorik hassasiyet elde etmiş oluruz.

Bu hassasiyet sistemin paralel kinematik yapısından dolayı, hareketli platformun farklı yönelimleri ve konumları için değişebilir.

Hareketli platformun 6 serbestlik derecesine sahip olabilmesi için bu yapıdan en az 3 tanesi paralel bir şekilde sabit platforma bağlanmalıdır. Fakat 3 kolun kendi içinde handikapları vardır. Bunlardan biri rijitliktir. Örneğin platformlara bağlantı noktaları eş olamaz. Aksi takdirde kinematik denge sağlanamaz.

Bu tasarımda Stewart platformununda ön gördüğü şekliyle 6 kol kullanılmıştır. Böylece sistemin rijitliği artırılmış, geliştirilen aktüatör yapısıyla tekillik (singularty) noktaları azaltılmıştır.

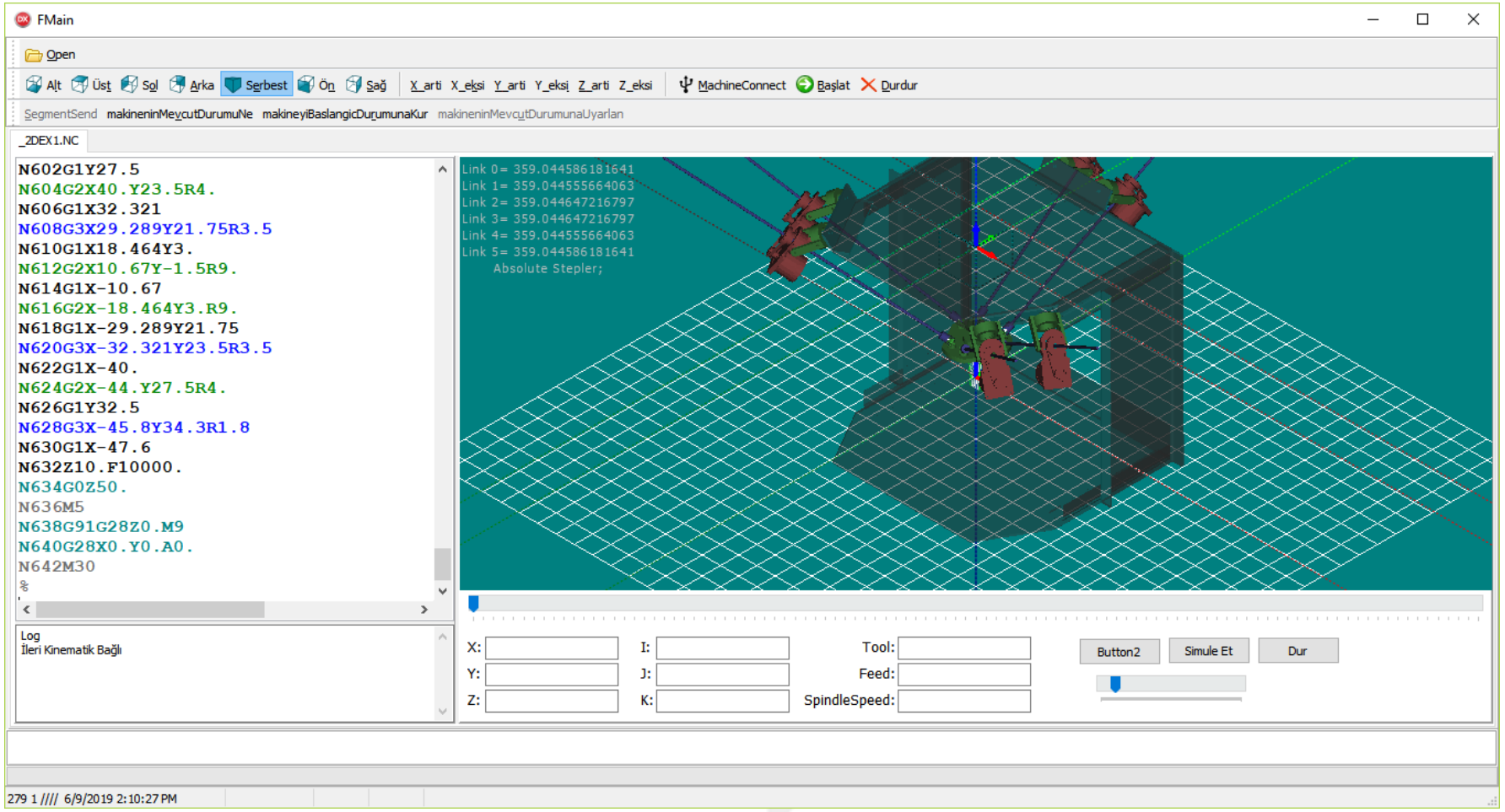


Şekil 4.5. Özgün tasarım olan actüatörlerin ara montajlı görünüşü.

4.2. Yazılım Tasarımı

Sistemin yazılım katmanı iki kısımdan oluşmaktadır. Birincisi HexSimPos adında masaüstü uygulaması ve diğer HexSimPos Firmware adında aygıt yazılımıdır. Her iki program çalışma zamanında (run time) koordineli bir şekilde haberleşir ve mekanik aksamın kontrol edilmesine olanak sağlar.

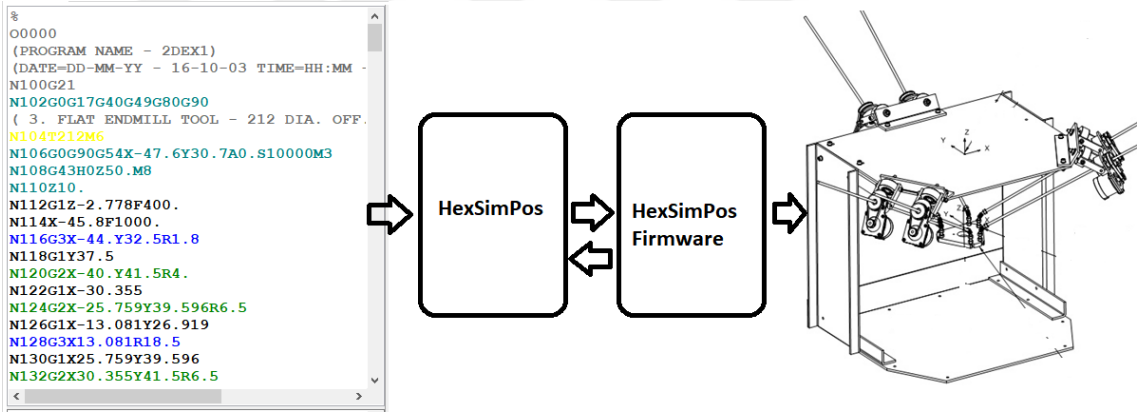
Masaüstü uygulaması HexSimPos, delphi ve python programlama dili kullanılarak geliştirilmiştir. Geliştirme ortamı olarak Embarcadero RAD studio ortamı kullanılmıştır. RAD studio, klasik Windows araçların yanısıra GLScene kütüphane'sinide kullanımına olanak sağlar. Geliştirilen HexSimPos yazılımının ekran görüntüsü Şekil 4.6'daki gibidir.



Şekil 4.6. Geliştirilen HexSimPos kontrol ve simülasyon yazılımının ekran görüntüsü.

Sistemin ara yüzü genel itibari ile delphi dilinde geliştirilmiştir. Kinematik hesaplamalar ve görselleştirme bu yapı üzerindedir. Sadece ileri kinematik fonksiyonları python dilinde geliştirilmiştir. Bunu nedeni ise GLScene kütüphanesinin, 6x6 matris işlemlerinin yapılamasında yeterli olmamasıdır. İleri kinematik fonksiyonlarında jacobiyen matrisinin yapısı, 6x6 ‘dır (robotun 6 adet kol uzvu ve 6 serbestlik derecesine sahip olmasından dolayı). 6x6 matris işlemleri yapabilen yazılım kütüphanesi eklemek yerine, ileri kinematik denklemlerinin çıkartılmasında kullanılan “numpy” kütüphanesi kullanılarak python da geliştirilen prototip uygulama kullanılmıştır.

Geliştirilen HexSimPos yazılımının özellik olarak, temel G kodlarını açıp, bunları yorumlayıp 3 boyutlu sahne ortamında simule etmesi ve gerekirse makineyi kontrol etmesi sağlanmıştır. Genel işleyiş hakkında Sekil 4.7’den bilgi edinilebilinir.



Şekil 4.7. Sistemin veri akış şeması.

Eksen boyları ve hızları verilen bir yörünge yani G koduna göre masaüstü uygulamasında hesaplanır. Hesaplanan bu konumlar aygıt yazılımına gönderilir. Aygıt yazılımı ise motorları kontrol etmek suretiyle hareket yapılmasını sağlar.

Aygıt yazılımı temel iki görevi vardır. Birincisi HexSimPos yazılımı tarafından gönderilen direktiflere uymak. İkincisi herhangi bir acil durum veya güç kesilmesi olduğunda motorları durdurup konumlarını kendi flaş belleğinde kayıt etmektir. Tabi, acil durum oluştuğunda da motorları durdurup masaüstü uygulamasına da haber verir.

4.2.1. Kontrol yazılımı

CNC lerde geleneksel kontrol yöntemleri, genelde özel geliştirilmiş kontrol kartları ve ara yüzler kullanılır. Bu yöntem 3 eksen kartezyen robotlarda yeterlidir. Fakat günümüzde çok

eksenli seri ve paralel robotlar kullanılmaya başlanmıştır. Bu durum kontrolcü kartların kullanılmasının yerine PC tabanlı yazılımlarının kullanılmasına zorlamıştır. Bunun en büyük nedeni fiziksel mekanizmanın sanal ikizinin oluşturulup önceden oluşabilecek hataların, çarpmaların önüne geçilebilmesi ve otonom işlerin kolay bir şekilde optimize edilmesini sağlayan bir ortam sunmasındandır.

Bu çalışmada geliştirilen HexSimPos masaüstü yazılımı da bir PC tabanlı kontrol yazılımıdır. HexSimPos, G kodları okur ve makineyi yönlendirecek kol uzunluklarını çıkartır.

HexSimPos yazılımı 3 boyutlu nesnelerin bir sahnede sanal ikizlerin çıkartılabilmesi için GLScene 3 boyutlu grafik kütüphanesini kullanır. Bu kütüphanenin tercih edilme nedenlerinde biriside 3 boyutlu sahne içerisindeki modellerin birbirleri içerisinde çarpma kontrolü sağlanabiliyor olmasıdır. Ayrıca GLScene kütüphanesi donanım hızlandırılmış bir çizim ortamı sunar. Bu da çalışma zamanında yazılım akıcı bir şekilde görevini yapabilmesini sağlar.

Hexapod robotun bilgilerini içeren çalıştırılabilir dosya kökündeki ".\Tanim\Tanim.rpr" dosyasını okur ve hexapod robotun bilgilerini alır. "Tanim.rpr" dosyası temelde bir metin dosyasıdır ve içerisinde robotun hareketli ve sabit eksen üzerindeki pivot noktalarının konumlarını içerir. Bu sayede farklı pivot noktalarına sahip Stewart robotu da tanımlanabilir. Yine ayni dosya robotun 3 boyutlu modellerinin isimlerini ve dosya yolu konumlarını bulundurur. HexSimPos yazılımı yine "Tanim" klasöründe bu stl formatındaki 3 boyutlu modelleri başlatılırken okur ve yükler.

HexSimPos çalıştırılabilir dosya kökünde ".\i1erikinematik \ilerikinematik.exe" çalıştırılabilir dosyası mevcuttur. Bu dosya robotun ileri kinematik hesaplamalarını barındırır.

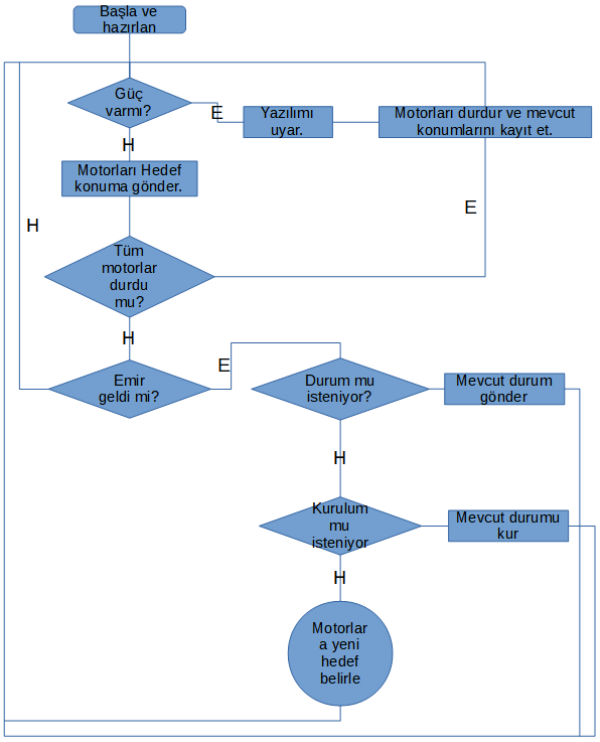
4.2.2. Aygıt yazılımı

Masaüstü uygulaması HexSimPos programı tarafından, aktüatör uzunlukları ve hızları hesaplandıktan sonra bu bilgiler gerçek zamanlı bir şekilde motor sürücülerine ve motorlara gönderilmelidir. Bunun için Arduino Nano geliştirme platformu kullanılarak, bir aygıt ara yüzü geliştirilmiştir. Tıpkı bir eksen kontrol kartı olarak çalışan bu yapı geleneksel kontrolcülerden farklı olarak kinematik hesaplamaları içerisinde barındırmaz. Bunun yerine daha önceden de belirtildiği gibi HexSimPos yazılımı ile yapılmıştır.

Bilgisayarı doğrudan sürücülere bağlamayarak, güç katındaki (motorlar ve sürücüler) olası dalgalanmalar ve parazitlerin HexSimPos yazılımı olumsuz etkilememesi amaçlanmıştır.

Bu Arduino Nona tabanlı aygıt arayüzün temel görevleri ise HexSimPos yazılımdan gelen emirleri yerine getirmek ve olasi acil durumlar için motor ve sürücüleri durdurmaktır.

Step motorlar açık döngü kontrol metodu kullanılarak kontrol edilmiştir. Dolayısıyla motorlar ve sürücülerin konumlarını olası durumlara karşın ve ileri kinematik hesaplamaların yapılabilmesi için saklanması gerekmektedir. Geliştirilen bu aygıt yazılımı oluşabilecek acil bir durumda motorların durdurmanın yanı sıra o anki konum bilgilerinide kendi flash belleğine kayıt eder. Bu bilgiler daha sonra HexSimPos yazılımı istediğinde ileri kinematik hesaplari yapabilmek için kullanabilecektir. Bu sayade sistemi yeniden başlatmak istendiğinde sistemin hazırlık zamanı kısaltılmış olur. Aygıt yazılımı Şekil 4.8 de görüldüğü gibi bir akış şemasına sahiptir.



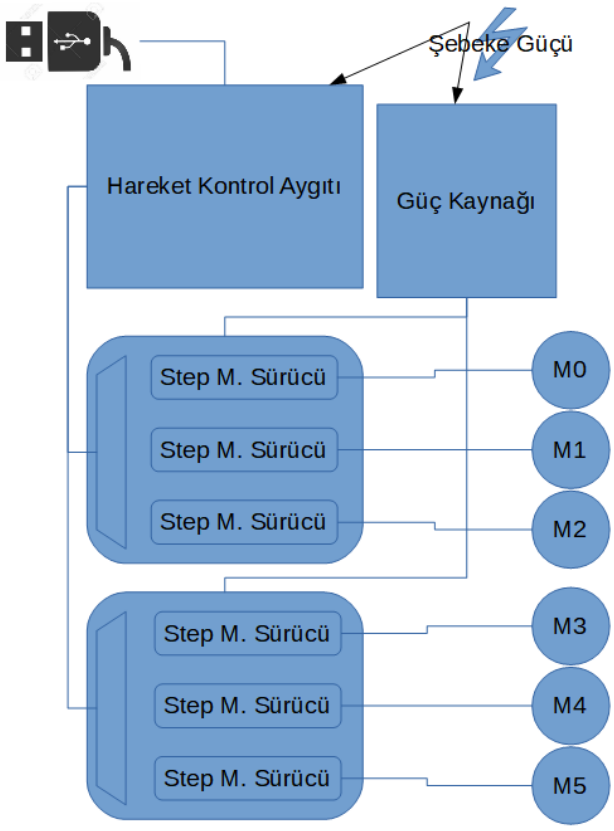
Şekil 4.8. Aygıt yazılımının çalışma algoritması.

4.3. Elektriksel Sistem Tasarım

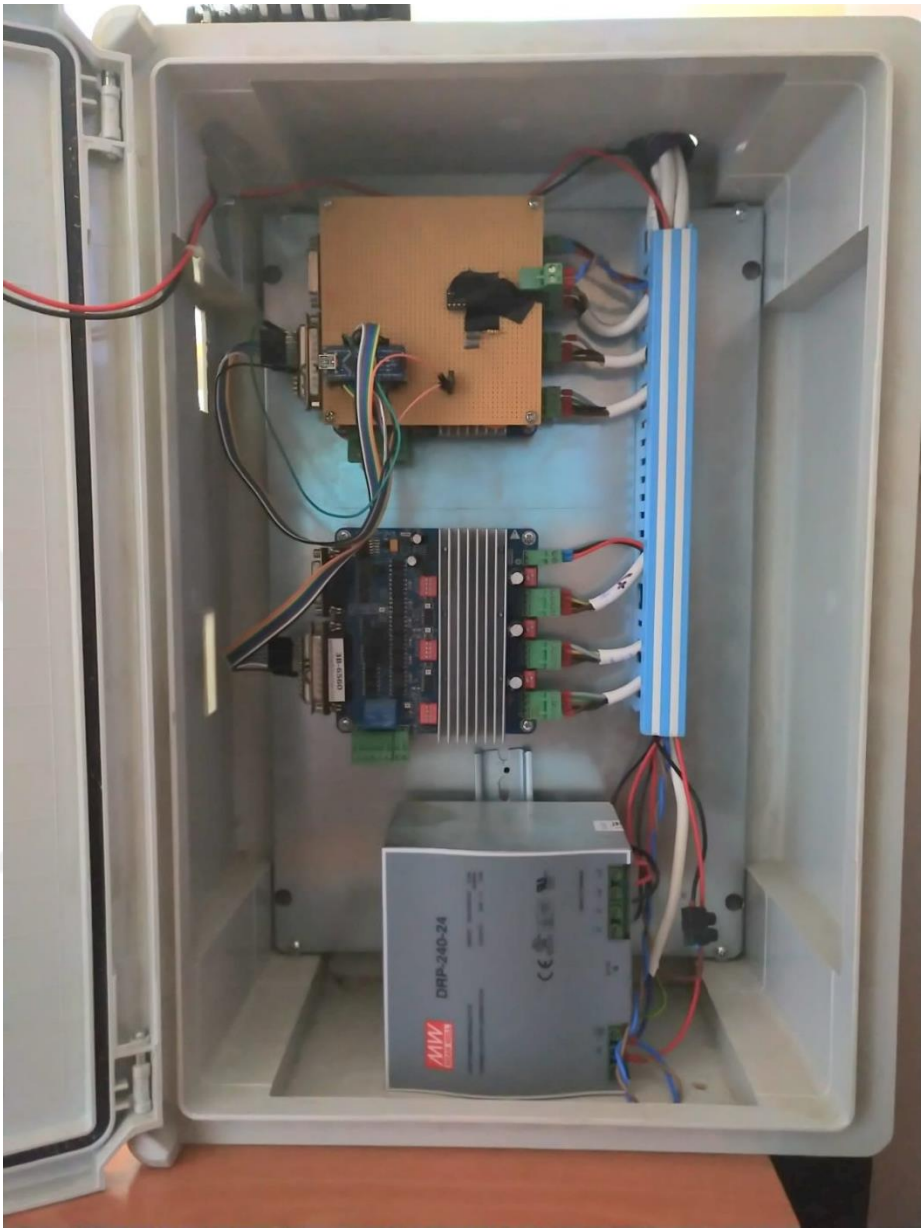
Elektrik-elektronik katmanı, içerisinde aygıt yazılımında bulunduğu hareket kontrol aygıtı ve step motor sürücülerinden oluşan temel bir yapıya sahiptir. Aygıt yazılımı, hesaplanan aktüatör uzunluklarını motorlara, oradan da aktüatörlere yansıtır. Firmware yazılımı actuator konumlarını da bu aygıt içerisinde kayıt edilir.

Hareket kontrol aygıtı temelde Arduino nano geliştirme kartı tabanlı Atmel atmega 328p mikro işlemcili bir devre kartıdır. Bu kartın step motorları olabildiğince hızında kontrol edebilmek için üretebilcekleri gerçek zamanlı adım miktarı belirlenmesi gerekir. Bu maksimum üretilebilinen puls miktarını belirlemek için her bir motora farklı hızlarda fakat aynı zamanlarda tamamlayabilecekleri komutlar dizi gönderilmiş ve her bir motorun aynı anda harekete başlayıp aynı anda hareketi tamamlaması beklenmiştir. Bu testler sonucunda yaklaşık her bir motor için 1000 puls sonrası durumlarda, motorların aynı anda başlayıp farklı zamanlarda durduğu göslemlenmiştir. Buda real time olarak üretilebilinen maksimum adım miktarı olarak belirlenmiştir.

Hareket kontorl aygıtı aynı zamanda içerisinde barındırdığı şebeke gücü kesintisialgılam sensörü sayesinde olası güç kesilmelerine karşın motorları durdurarak fazldan boşa gönderilecek pals adımlarının önüne geçilmiş olur. Aynı zamanda aygıt güç kesintisi esnasında motorları durdurmasının yanı sıra aktüatör boylarınında mikro işlemci eeprom hafızasına kayıt etmektedir. Böylece yeniden harakat verildiği anda yazılım ve makine yeniden kaldığı yerden devam edebilmektedir.



Şekil 4.9. Elektrik-Elektronik kontrol ve güç şeması.



Şekil 4.10. Elektrik-Elektronik kontrol panosunun genel görünüşü.

5. YAPISAL ANALİZ

Analiz aşamasında bir sonlu eleman çözümleyici olan Ansys Workbench programında statik fizik projesi oluşturulmuş ve sistem bu fizik ortamında analiz edilmiştir.

Sistemi analiz etmek için kullanılabilecek kısıtlardan biriside iş mili motorunun (spindle) gücüdür. Bunun yanısıra ivmelenmelerden gelen atalet kuvvetlerininde analize katılması gereklidir fakat bu çalışmada atalet kuvvetleri hesaba katılmamıştır. Bunu nedeni ise kullanılan step motorların oluşturacağı ataletler ihmal edilebilecek kadar küçük olacağı varsayılmıştır. Örneğin bu sistemde hareket kontrol kartı tarafından üretilebilinen adım miktarı 1000pals/saniye olarak tespit edilmiştir. Bu değer aktüatörler üzerinde 1000pals/saniye / 400 pals/mm (1mm deki step motor adım miktarı) denklemiyle eyleyiciye oluşturabilinecek maksimum ivme 0.0025m/saniye^2 olarak bulunur. 0.5 kg lık bir spindle motorunun oluşturabileceği kütlesel atalet kuvveti 0.00125 kilogram kuvvet oda yaklaşı olarak 0.012258 Newtonluk bir kuvvete eşittir. Bu değer hesaplanan kesme kuvvetinin oluşturacağı değerin yaklaşık % 1'i kadardır. Böylecek bunun yerine sistem hassasiyeti ve oluşturabileceği maksimum kesme kuvvetleri üzerinde durulmuştur.

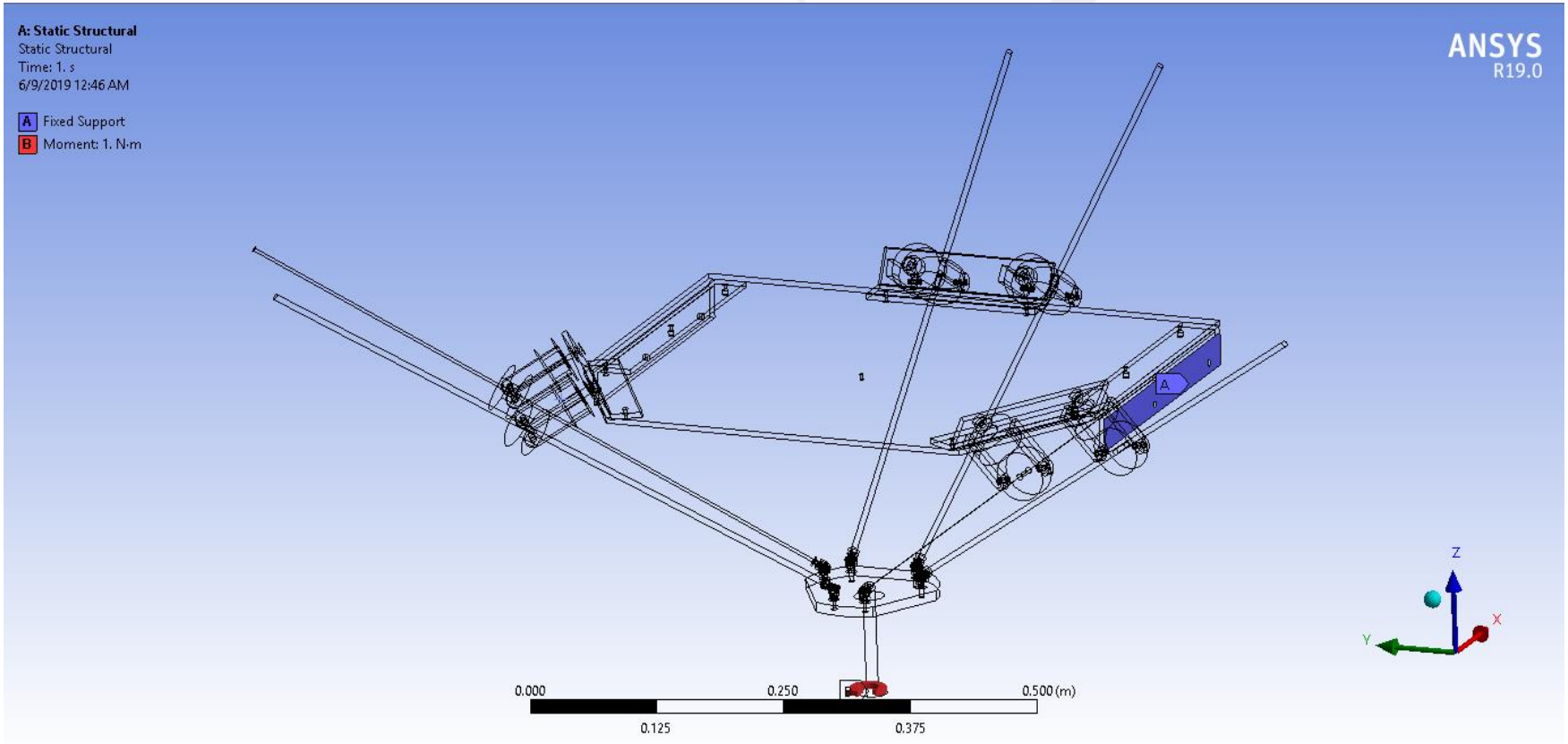
Örnek kesme senaryosu, sistemin park pozisyonundayken (Home position) 0.500 kW'lık spindle motorunun 20000 devir/dk hızdaki oluşturacağı yaklaşık tork degeri hesaplanıp kesme kuvveti olarak uygulanmistır. Motorun oluşturacağı tork miktarı denklem 5.2 de devir-güç-tork ilişkisinden hesaplanmıştır.

$$\mathrm{P}\ (kW) = \frac{\tau(Nm) * n\ (rpm)}{9550} \tag{5.1}$$

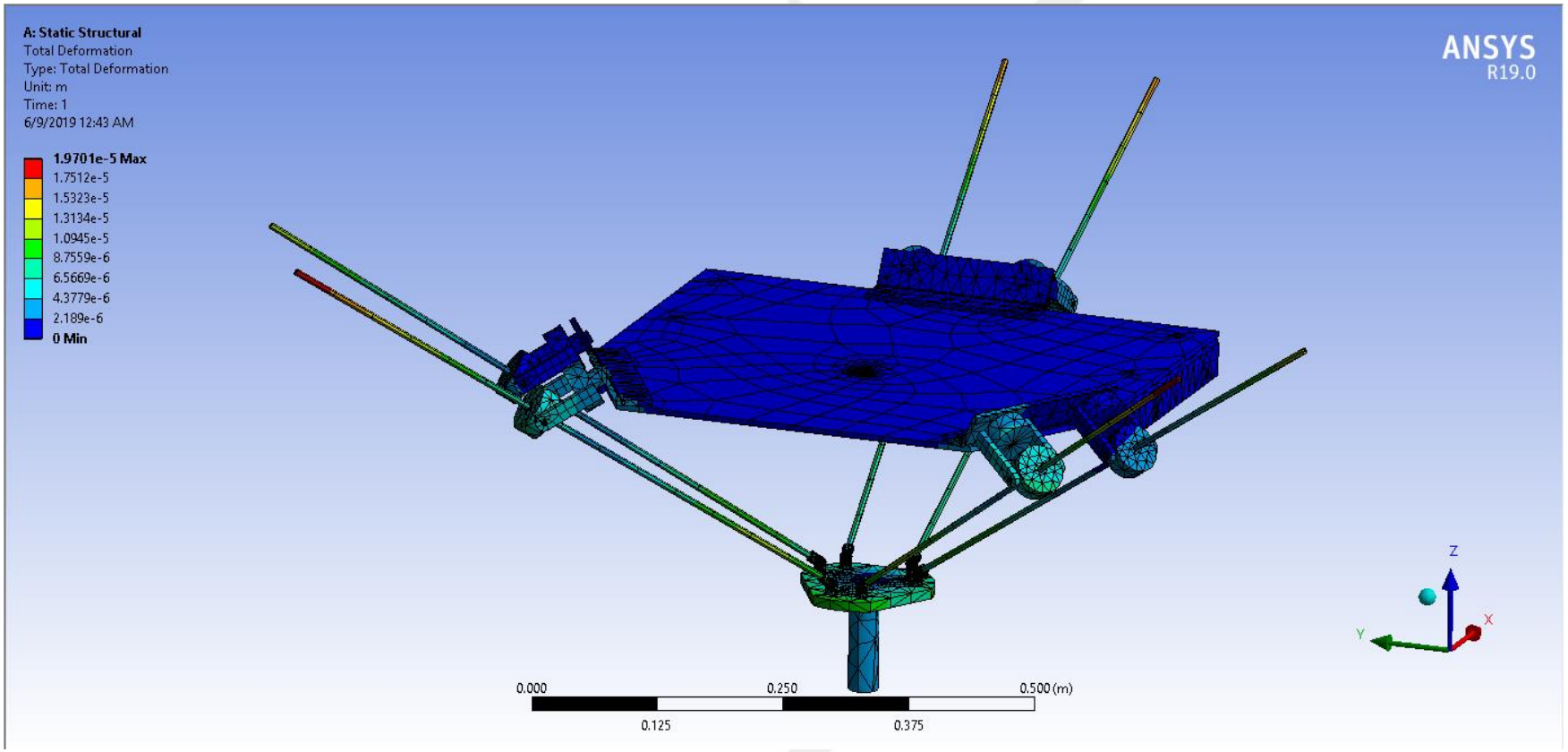
$$\tau(Nm) = \frac{\mathbf{P}(kW) * 9550}{n(rpm)} \tag{5.2}$$

Burada P kW cinsinden motor gücünü, τ Nm cinsinden motorun çalışma torkunu, n motorun dakika da yaptığı devri temsil etmektedir.

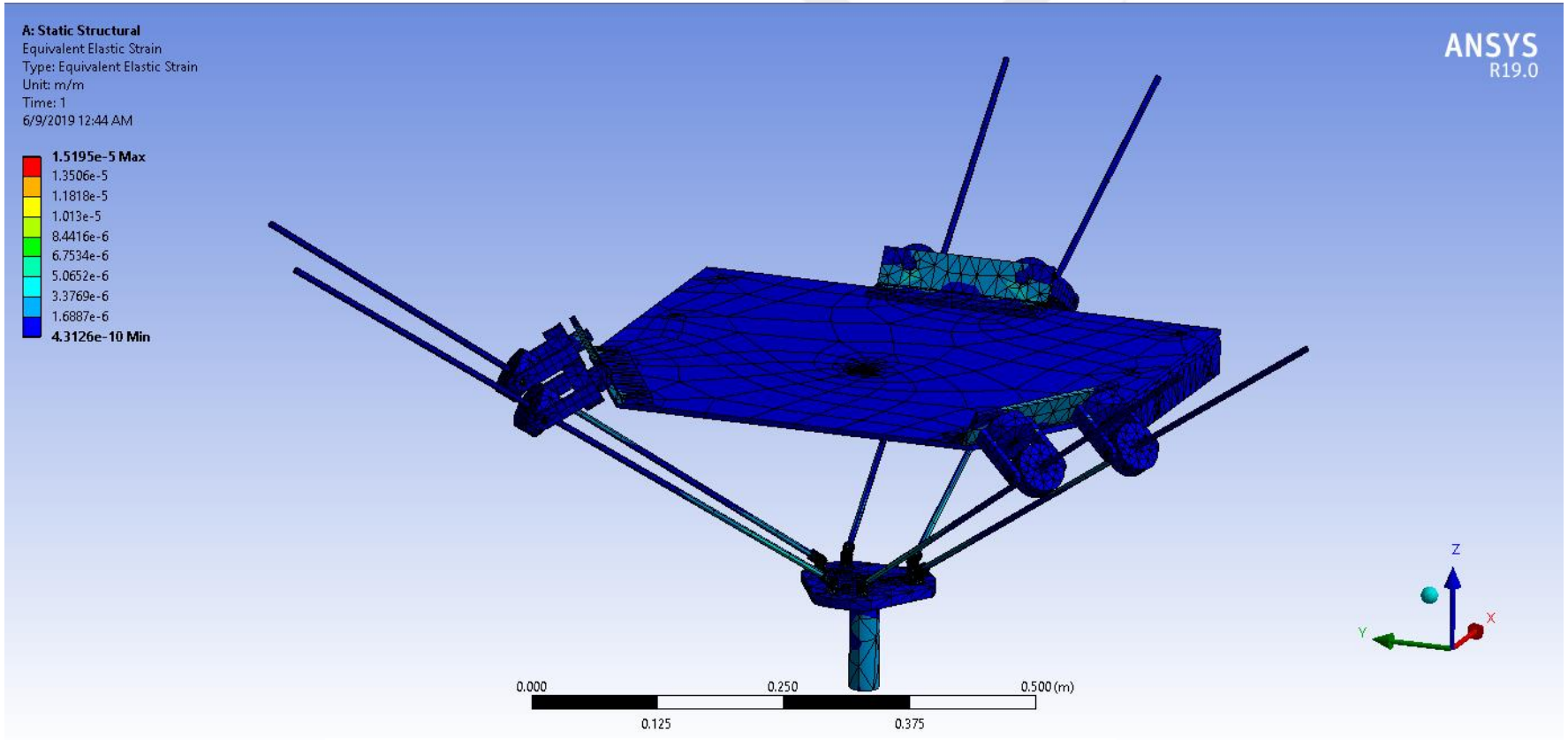
Denklem 5.1 denklem 5.2 olacak şekilde yazılabilir ve buna göre kesme torku l Nm olarak hesaplanmış, hareketli platformun 100 mm aşağısına şekil 5.1'deki gibi uygulanmıştır.



Şekil 5.1. Analiz modeline uygulanan yükler ve sabitleme yüzeyleri.



Şekil 5.2. Örnek senaryoda sistemin toplam yerdeğiştirme dağılımı.



Şekil 5.3. Örnek senaryoda sistemin gerilme dağılımı.

Sonuçtan da görüleceği üzere toplam sonlu eleman düğümlerinin maksimum yer değiştirmesi 0.000019m yani 0.019 mm dir ki bu 0.0025 mm olan aktüatör hassasiyetini 7.6 kat aşmaktadır. Gerilmelerden doğabilecek yer değiştirmelerin sistem hassasiyetini geçmemesi beklenir. Çünkü işleme esnasında titreşimler büyür ve gerilmelerin oluşturduğu gerinim iş parçası üzerinde olumsuz etki yapar. Bu durum, bilyalı vidalı mil ve hassas işlenmiş bağlantı elemanlarının kullanılması ile daha rijit bir sistem tasarlamak suretiyle giderilebilir. Bu çalışmadaki amaçta daha çok, oluşturulan matematik altyapının doğruluğunu, hareketlerin doğruluğunu ve sistemin çalışmasını doğrulamak için geliştirilmiştir.

Bu elde edilen 7.6 katlık fark, havacılık, uzay , metal işleme endüstrisinde hassas işler yapan bir CNC makinesi için çok büyük olsada daha az öneme sahip reklamcılık gibi endüstrilerde kullanımı uygundur.

6. ÇALIŞMA HACMİ VE FINDIK NOKTALARI

Robotlarda çalışma hacmi, hareketli platformun ulaşabildiği noktalar kümesi olarak tanımlanır. Bu tanım herhangi bir robotun karakteristik özelliğini belirleyen terim olarak kullanıla gelmiştir. Eğer bir kartezyen robottan söz ediyor olsaydık son eyeleyici noktadan noktaya gittiği için bu tanımı kullanmak yeterli olurdu. Farklı bir son eyleyici takıldığında çalışma hacmini taşıyarak veya kırparak yeniden belirleyebiliriz.

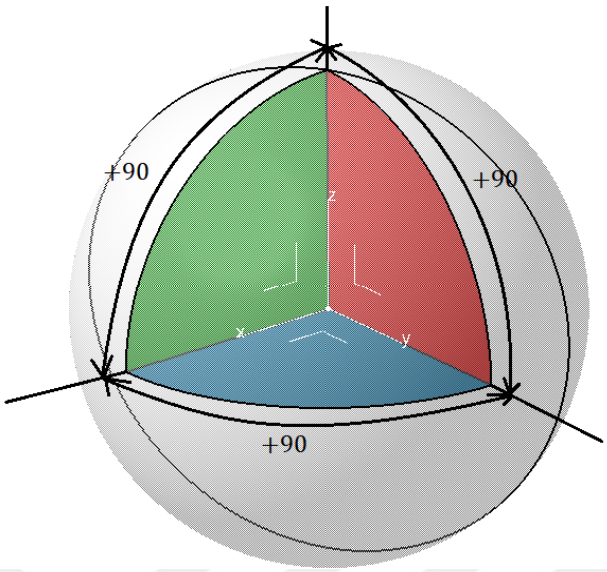
Fakat çalışma hacminin belirlenmesindeki amaçlardan biride robotun uzuvların iş parçasıyla veya uzuvların kendi içinde çarpışmasını önlemektir. Olası imkânsız bir noktaya konum verildiğinde bu noktanın çalışma hacmi içerisinde olup olmadığına bakılarak o noktaya gidilip gidilmeyeceğine karar vermektir. Böylece robotun kendine veya iş parçasına veya son eyleyiciye zarar vermesi önlenmiş olur.

Geleneksel çalışma hacmi tanımında, çalışma hacminin son eyleyicinin rotasyonlarıyle birlikte ele almadığı için çok eksenli robotlarda yeterli olmadığını görürüz. Çünkü çok eksenli robotlar son eyleyiciyi veya iş parçasını farklı yönelimlerde ele alınmalıdır. Yani geleneksel çalışma hacmi tanımı, içerisindeki bir noktada her yönelimin gerçekleştirildiğini kesin olarak söyleyemez.

Her zaman iş parçasının aynı konumda olduğunu ve her zaman aynı son eyleyici ile iş yaptığını varsayamayız. Genelde robotlar farklı işler yapılabilmesi için tasarlanırlar. Dolayısıyla bir robota farklı iş parçaları farklı son eyleyici takılabilir ve farklı görevlere programlanırlar.

Örneğin bir kaynak robotu düşünelim; son eyleyici olarak kaynak torçu takılabilir veya bir tutamaçta monte edilebilir. Robotun bu 2 durumda ulaşabileceği noktalar kümesi farklıdır. Ofsetleme veya kırma yöntemi çalışma hacminin yeniden belirlenmesinde yardımcı olmayabilir.

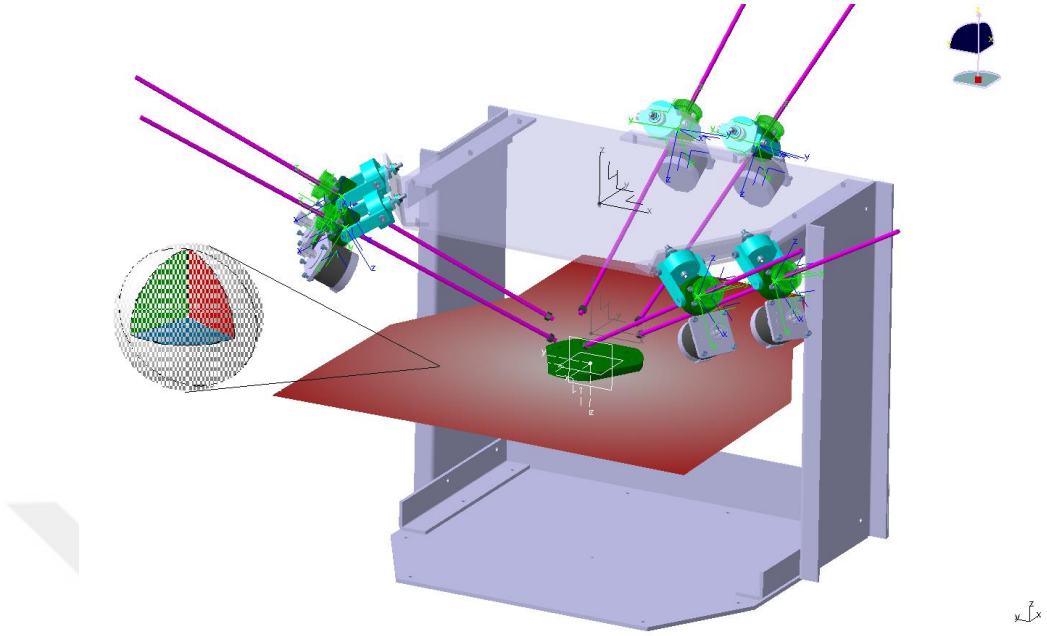
Bu çalışmada ise çalışma hacmi tanımı yeniden yapılmış ve çalışma hacmi ulaşıla bilinen noktalar kümesi yerine, "fındık" noktalar kümesi olarak yapılması önerilmiştir. Fındık noktası hareketli platformun veya son eyelyicinin bir noktadaki konumu herhangi bir açı setindeki yaptığı açılarıda içerir. Örneğin hareketli platform bir noktada konumu, roll, pitch, yaw açılarını şekil 6.1’deki gibi temsil edilebilir.



Şekil 6.1. Hareketli platformun +90 roll, +90 pitch, +90 yaw açısını yapabildiğini tanımlayan fındık noktası.

Bu noktaların rotasyon bilgileri belirli bir renk tarafından tanımlarsak fındık noktalar kümesinin bir araya gelerek oluşturduğu renk tayfı, o düzlem veya hacimde hareketli platformun hareket kabiliyetini daha iyi tanımlar.

Örneğin şekil 6.2'deki gibi keyfi bir düzlem için roll açılarının oluşturduğu renk tayfını izafi olarak görülmektedir. Burada beyaza kayan noktalar hareketli platformun yönelimlerde daha kabiliyetli olduğu anlaşılır. Hareket kabiliyeti azaldıkça siyaha yakın olacak ve hareket olmayan konumlarda fındık noktası bulunmayacaktır.



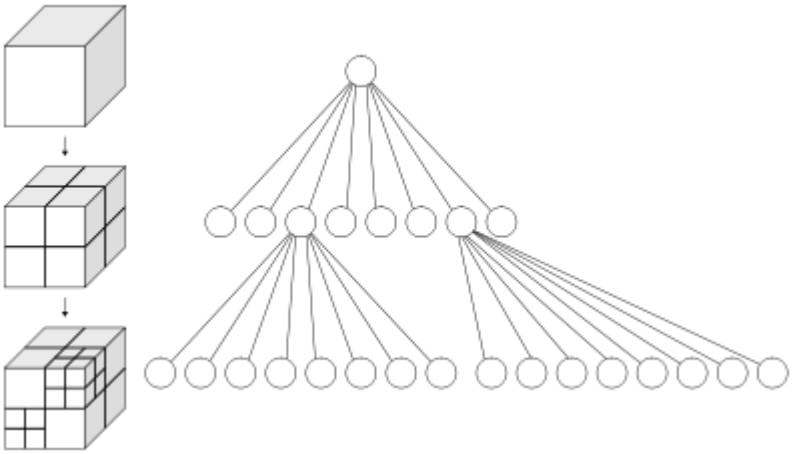
Şekil 6.2. Keyfi bir düzlemde hareketli platformun harketlerini tanımlayan izafi fındık noktaları kümesi.

Bu noktalar belirli aralıklarda ve belirli sınırlar içerisinde önceden araştırılarak bulunur. Çalışma zamanı içerisinde bu noktalara o anki konum ve yönelim için yakınındaki fındık noktalarından ara değerler bulunarak hareketin yapılıp yapılmayacağına karar verilir. Çalışma zamanın bu noktaların hızlı bir şekilde bulunabilmesi içinde farklı sayısal yöntemler kullanılabilir. Örneğin düz bir ızgara listesine yerleştirmek yerine "octree" yani iç içe geçmiş 8 li listelere yerleştirilebilinir. Bir sonraki konuda fındık noktalarının octree listeleri ile kullanılması açıklanmaya çalışılmıştır.

6.1. Fındık Noktalar Kümesinin Octree Yapılarıyla Birlikte Kullanılması

Fındık noktaları önceden bir ızgara yapısı üzerine biriktirilseydi çalışma zamanında bu noktaları tek tek araştırmak oldukça zaman alıcı bir işlem olurdu. Bunu daha hızlandırabilmek için octree yapıları kullanılabilir.

Octree, 3 boyutlu uzayda belirli bir hacmin doğrusal eksenler yönünde 2 şerli, öz yinemeli bir şekilde 8 eşit hacme bölünmesine denir (https://en.wikipedia.org/wiki/Octree). Şekil 6.3 de octree yapısı hakkında görsel verilmiştir.

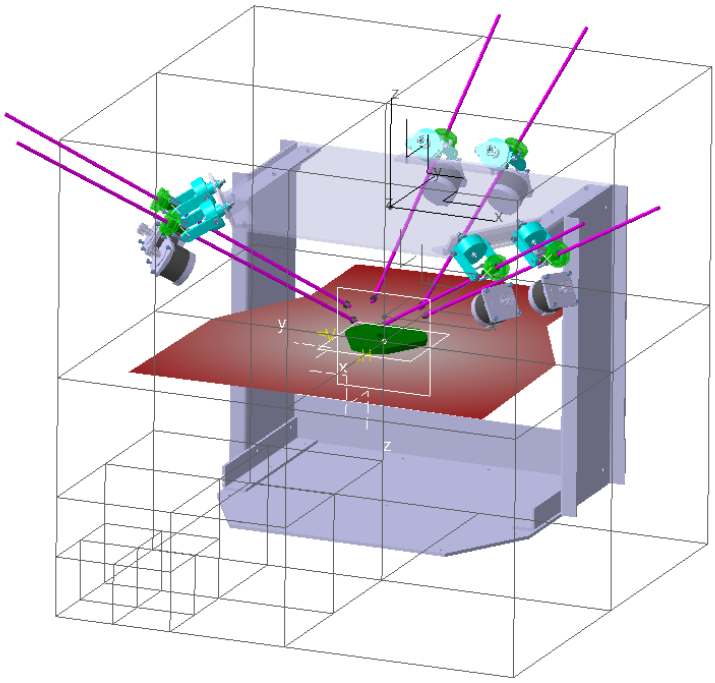


Şekil 6.3. Octree yapısı (Wikipedia, 2019).

Octree, yapıları 3 boyutlu grafik kütüphanelerinde sıklıkla kullanılır. Catia, Solidworks, Unigrapics gibi 3 boyutlu tasarım programları bu yapıyı farklı amaçlar için hali hazırda kullanmaktadırlar. Octree temelde bir ağaç yapısıdır. Her bir kök 8 tane daldan oluşur ve belirli bir birim aralığa ulaşıncaya kadar böyle devam eder. En son dal ilgili objeyi veya objeleri taşır. Bu obje bu çalışmada tabiki fındık noktaları olacaktır.

Küçük bir karşılaştırma yapılacak olursak;

10 cm lik birim aralığa sahip 400x400x400 cm bir hacme fındık noktalarını yerleştirelim. Baştan sonra fındık noktası bulmaya çalışalım. Elbette listedeki ilk fındık noktasına ulaşmak çok kolay olacaktır fakat en sondaki fındık noktasına ulaşmak 64000 iterasyon sonucunda bulunacaktır.



Şekil 6.4. Bu çalışmada uygulanabilinecek örnek bir octree.

Bu notaları octree üzerine yerleştirdiğimizi farz edersek ilk noktaya ulaşmak için ilk önce 400'lü kök, 200'lü kök , 100'lü kök , 50'li kök, 25'1i kök , 12.5'li kök ve ilgili fındık noktasına ulaşılır. En sondaki fındık noktasına ulaşmak istedik diyelim yine ilgili 400'1ü kök, 200'lü kök, 100'lü kök, 50'li kök, 25'li kök, 12.5'li kök ve ilgili fındık noktası bulunur.

Görüldüğü gibi her iki durumda da 6. iterasyonda ilgili fındık noktalarına ulaşılmıştır. Bu ayrıca bize her bir fındık noktasını yaklaşık aynı zamanda bulma garanti sinide verir.

6.2. Hexapod Robotun Simülasyonu ve Kontrolü için 3 Boyutlu Grafik Kütüphanelerini kullanmak

Günümüzde ekran kartları içerisinde barındırdığı GPU lar sayesinde çoklu işlem yapabilme kabiliyeti ve 3 boyutlu işlemler için hazır donanımsal destekler sunmaktadir. Bu donanımları efektif bir şekilde kullanımıyla "hardware accelerated" donanın destekli yazlımlar geliştirmek mümkün olmuştur. Bu donanımları kullanmamıza ve 3 boyutlu uygulama geliştirmemize yardımcı olan grafik yazılım kütüphaneleri vardır. Örneğin günümüzde yaygın bir şekilde kullanılan Catia, Solidworks, Unigrafics, Creo gibi 3 boyutlu modelleme uygulamaları bilgisayar ortamında sahne ortamı yaratılırken bu yazılım kütüphanelerinden yararlanırlar.

Geleneksel CNC kontrolcüleri özel geliştirilmiş hareket kontrol kartları ve ara yüz aygıtlarının kullanıla gelmiştir. Çok eksenli CNC ve robotların yaygınlaşmasıyla bu hareket kontrol kartları yeterli gelmemeye başlamıştır. Bunun nedeni karmaşık kinematik işlemlerinin yapılması ve güvenlik fonksiyonlarının eklenmesi zorunluluğundan dolayıdır. Geleneksel bir CNC kontrolcüsü kartezyen koordinat sisteminde çalışan 3 eksenli robotlar için yeterli olur. Fakat çok eksenli bir CNC tezgâhı için geleneksel CNC kontrolcüler yeterli gelmeyebilir. Bu durumda PC tabanlı uygulamalar geliştirilmiştir.

Mazak, Fanuc, ABB, Beckhoff gibi CNC, robotik ve otomasyon firmaları endüstri 4 ile birlikte PC tabanlı kontrolcüler geliştirmişler ve ticari olarak kullanmaktadırlar.

Belirli popüler PC tabanlı CNC kontrolcülerin karşılaştırılması çizelge 6.1.'deki gibidir,

Çizelge 6.1. PC tabanlı CNC kontrolcülerinin karşılaştırılması.

Kontrolcü	Gömülü kinematik	Maksimum eksen kontrolü	Çarpma kontrolü	Platform
HexSimPos	var	6	var	Windows OS
Linuxcnc	var	9	var	Linux OS
Mach 3	yok	5	yok	Windows OS
Mazatrol	var	-	-	Windows Embeded OS
Fanuc Robot UI	var	-	var	Windows OS
ABB	var	-	var	Windows OS

Bu çalışmada hareket kontrolcüsü olarak kişisel bilgisayar tabanlı bir uygulama kullanılması tercih edilmiştir. Bir PC nin gücü bir mikro yongalı sisteme göre kat kat çok yüksektir. Hem daha kullanışlı "user friendly" ara yüz geliştirilebilinir hemde ekran kartı kullanmak suretiyle donanım hızlandırılmış "hardware accelareted" uygulamalar geliştirilebilir. Bu çalışmada da GLScene 3 boyutlu grafik kütüphanesi kullanılmıştır.

GLScene kütüphanesi delphi ve borland C++ dillerinde uygulama geliştirme olanağı sağlar. GLScene kütüphanesi içerisinde 3 boyutlu sahne ortamı oluşturulmasına müsaade eder. Aynı zamanda modelleme programında modellenen robot uzuvlarını *.obj, *.stl gibi dosya formatlarından okuyabilmektedir. İçerisindeki yapı sayesinde bu uzuvların çarpma kontrolü sağlanmasına müsaade eder. Bu çalışmada kullanılması önerilen octree yazılım objelerini de içermektedir.

Çizelge 6.2. de GLscene gibi popüler yazılım kütüphanelerinin karşılaştırılması yapılmıştır.

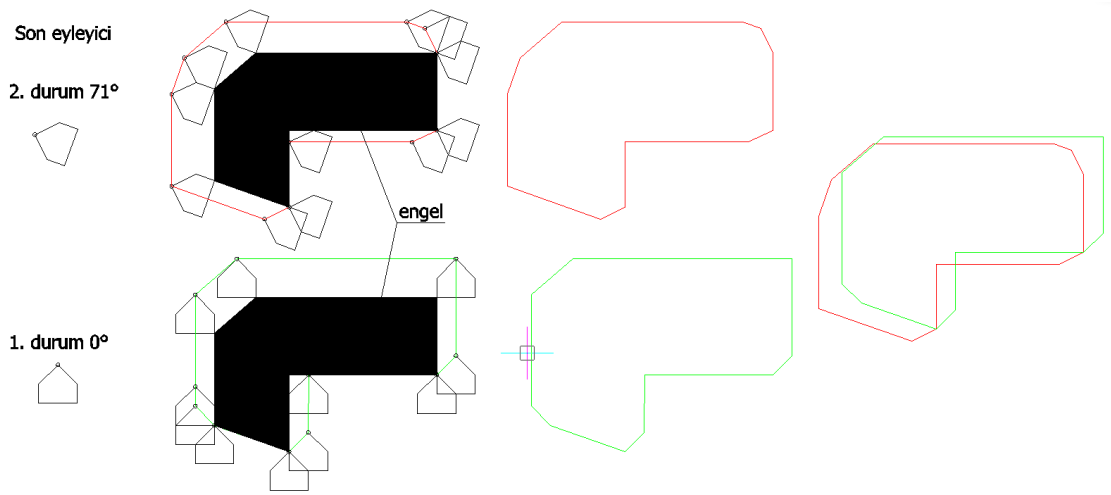
Çizelge 6.2. 3Boyutlu grafik kütüphanelerinin karşılaştırılması.

3B Grafik Kütüphanesi	Dil	Lisans	Çarpma kontrolü	Octree yapısı	Platform
GLScene	Delphi, Borland c++	Mozilla Public Licence 1.1	Var	Var	Windows OS
OpenSceneGraph	C++	LGPL	Var	Var	Cross P.
Three.js	Javascript	MIT	Var	Var	Web
Qt3D	C++, Python	LGPL	Var	Var	Cross P.
OGRE	C++, Lua	MIT	Var	Var	Cross P.
Processing 3D	Java	GPL v2	Yok	Yok	Cross P.
P5.js	Javascript	GPL v2	Yok	Yok	Web

6.3. Konfigurasyon Hacmi ve Octree Destekli Fındık Noktaları Arasındaki Farklar

Konfigurasyon hacimleri bir robotun son eyleyicisinin, süpürerek oluşturabileceği muhtemel noktalar kümesini oluşturan hacim veya alanlardır. Bu hacim arzu edilen farklı son eyleyici ve engeller için yeniden yeniden modellenerek farklı durumlar için robotun karekteristik özelliği berlilenmeye çalışılır.

Anlaşılacağı üzere bilinen konfigürasyon hacmi ve çalışma alanı tanımı bir roboun karateristik özelliğini tanımlamada yeterli olamamaktadır. Şekil 6.5. de 2 boyutlu bir ortamda son eyleyicinin 0° ve 71° roasyonları için aynı engel üzerinde yaptığı konfigürasyon alanları tanımlanmıştır.

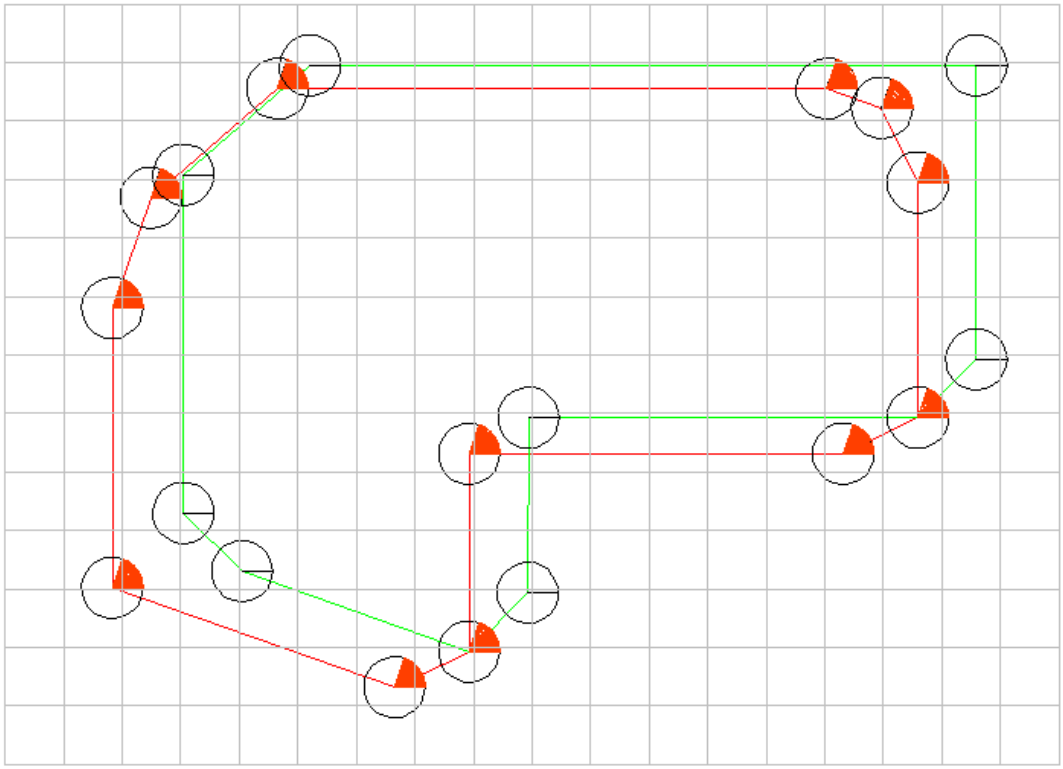


Şekil 6.5. 2 boyutlu ortamda verilen bir engel için farklı son eyleyici farklı rotasyonları için (0° ve 71°) konfigürasyon alanları.

Konfigurasyon hacimlerinin, çalışma alanlarının varlığı temelde robotun çalışma anında veya simülasyon ortamlarında verilen bir son eyleyici hedefinin doğru birşekilde yapabiliyor olması veya konumun gerçekte var olabiliyor olması veya robotun uzuvları veye iş parçası üzerinde çarma olup olmamadığı durumlarının kontrolünün yapılabilmesi için de gereklidir. Şekil 6.5 dende görüleceği üzere farklı iki durum için robotun konfigürasyon hacmini farklı iki alan için tanımlama gerekcektir. Bu durum son eyelyici değiştirilmesi veya engelin değişikliği durumum için farklı tanımlamalar yapmak gereklidir.

Yukarıda bahsedilen çarpışma durumları vs. öngörebilmek için genel bir tanıma ihtiyacımız oluduğu açıktır. Fındık noktaları bu tanımı genel çözümü için geliştirilmiş bir sayısal yöntemdir.

Şekil 6.6 da yukarıdaki örnek fındık noktaları cinsinden tanımlanmaya çalışılmıştır. Bu görselden anlaşılacağı üzre son eyleyicinin belirli konumlar için yapabilceği son eyleyici rotasyonlarının aynı koum üzerinde rotasyon bilgisini tutan noktalar oluşturulmuştur. F. noktalarının oluşturacağı renk tayfı görsel olarakta robot hakkında bir fikir vermektedir. Böylece önceki tanımlara göre daha sade, anlışılır ve kullanışlıdır.



Şekil 6.6. Örnek 2 boyutlu ortamda verilen bir engel için farklı son eyleyici farklı rotasyonları için (0° ve 71°) fındık noktaları.

Önceden robotun uzayında erişebilceği noktalar ve rotasyonlar için belirli bir ızgara aralılarında fındık noktalar kümesi çıkartıldığını farz edelim. Bu noktalar oluşturulurken son eyleyicinin robot uzuvları arasında çarpıp çarpmadığının kontrolüde yapılabilir. Çarpma kontrolünü, robot uzuvlarının gerçek modellerinden yaralanılarak grafik kütüphanelerince yapmak mümkündür. Izgara aralığı ne kadar küçük seçilirse o kadar yüksek çözünürlükte fındık noktaları kümesi elde etmiş oluruz. Çalışma anında veya simülasyon esnasında herhangi bir robot için verilen hedef noktası, o noktaya yakın fındık noktlarınca interpolasyon yapmak suretiyle, hareketin ulaşılabilir olup olmadığı bulunabilir.

Hedef noktanın doğruluğu fındık noktları ile tespit etmek, 3 aşamadan oluşur. 1. aşama hazırlık aşaması, robotun eklem kısıtları ve uzuvlarının birbirine çarpıp çarpmama durumuna göre belirli bir çözünürlükte keyfi bir noktada, son eyleyicinin rotasyonları erişelibilir olup olmadığı her bir fındık noktasına kayıt edilerek oluşturulur. 1. aşama aynı zamanda kalıcı aşamadır her bir robot için tasarım esnasına 1 kez oluşturulur. 2.aşama doğrulama aşaması, verilen hedef nokta kendine yakın olan fındık noktalarının interpolasyon edilmek suretiyle doğruluğu tespit edilir. Bu aşama anlaşılacağı üzere simülasyon veya çalışma esnasında çalıştırılır. 3. aşama ektra çarpma

kontrolü aşamasıdır. Bu aşamada farklı son eyleyici veya farklı iş parçası için birbirleri ve robot uzuvları arasındaki çarpma kontrolünün yapılmasıdır. Bu aşama yine simülasyon veya çalışma esnasında çalıştırlıcaktır. Eğer 1. aşamada kullanılan son eyleyici ve işparçası aynı ise 3. aşamaya gerek olmayacaktır. Böylece çarpmaların önceden öngörülebiliyor olacaktır.

Fındık noktlarının çözünürlüğü arttığı zaman her bir fındık noktasına ulaşmak sorun olabilir. Bunu aşmak için fındık noktları octree yapılarana listenmesi tavsiye edilir. Böylece arzu edilen 2. aşamada arzu edilen fındık noktalarına ulaşmak daha kolay olacaktır.

Fındık noktaları robotun karakteritik özelliğini belirlemede ve çalışma, simülasyon esnalarında çarpmaların önceden öngürülebilmesini sağlamada konfigürasyon hacimleri ve çalışma hacimlerine göre daha performaslı ve kullanışlı olabilirler.

7. SONUÇ VE ÖNERİLER

Bu çalışmada Stewart Platformundan esinlenerek, 6 kollu paralel kinematikli deneysel bir CNC makinesi tasarlanmış, imalatı yapılmış ve kontrolü için bir ara yüz/simülasyon yazılımı geliştirilmiştir. SP mekanizması birçok alanda kullanıla gelmesine karşın CNC makinesi olarak kullanma fikri ile alakalı yapılan literatür araştırmasında az olduğu görülmüştür.

Sıfırdan açık döngü kontrollü step motor tahrikli aktüatör tasarımı gerçekleştirilmiştir. Sistem ilk önce bilgisayar tabanlı kontrol yazılımı olan linuxcnc programında kontrolü sağlanmıştır. Linuxcnc programında tez çalışmasında öne sürülen bilgisayar tabanlı gerçek zamanlı çarpma kontrolü çalışması altyapısının yapılabilmesi ve yine bu çalışmada öne sürülen "fındık noktaları" tanımına deneysel zemin oluşturulabilecek kontrol yazılımı geliştirilmiştir. Bu kontrol yazılımı donanım destekli GLScene 3 boyutlu grafik kütüphanesi, delphi ve python programla dili kullanılarak geliştirilmiştir.

Genel hexapod kontrolü yapabilen bu yazılım, standart ISO G kodlarını okuyup, gerekli ters ve ileri kinematik denklemleri kendi içinde çözümleyip aktüatör uzunluklarını kontrol edecek veri akışı üretmektedir.

Bu veri akışları elektronik kontrol kartı ile, açık döngü kontrol metodu kullanılarak gerçek zamanlı step motor hareketlerine çevrilmiştir. Bu sayede aktüatörler konumlandırılmış ve robotun hareketi, 6 serbestlik dereceli olacak şekilde sağlanmıştır. Bu hareket kontrol kartı olası güç kesilmelerini algılayabilecek sensor vasıtası ile acil durum ve güç kesilmelerine karşı gerekli önlemler alabilmektedir. Yazılımla koordineli bir şekilde çalışan bu kontrol kartı gerektiğinde aktüatör boylarını hafızasına kayıt edip kullanılmasına olanak sağlamıştır. Örneğin ileri kinematikte gerekli aktüatör uzunlukları bu sayede elde edilebilmiştir.

Robotun karakteristik özelliği olan ve çalışma zamanında gerekli güvenlik önlemlerinin alınmasına fayda sağlayan çalışma hacmini daha iyi tanımlayabilecek “fındık noktaları” tanımı geliştirilmiştir. Çalışma anında da bu noktaların hızlı bir şekilde güvenlik fonksiyonlarının yerine getirilebilmesi için fındık noktalarının octree program listeleri üzerinde tutulması önerisi sunulmuştur. Böylece robotlar için önem arz eden kontrol ve güvenlik sorunları daha anlamlı ve doğru bir şekilde ele alınması sağlanabilinecektir.

G kodlarını incelediğimizde verilen bir noktanın sadece koordinat bilgisini içerdiğini görürüz. Prizmatik işlemlerde çoğu zaman bu yeterlidir. Fakat G kodları simultane hareketleri karşılayabilecek yeterli bilgi içermez. Simultane hareket yapabilen çok eksenli CNC tezgahların

G kodları ile kontrol edilmeye çalışılması, o tezgâhın yeteneğinden daha az özelliklerin kullanılmasına neden olabilir. Yani çok eksenli tezgahlar için daha farklı kontrol dosyaları geliştirilmelidir. Bazı CAM programları örneğin Catia bunu APTSource dosyası üreterek çözmüştür. Bu durumda ise üretilen bu dosyanın "post processor" denilen ayrı bir işleme tabi tutulmasına neden olmaktadır. Oysa CAM programları G kodu yerine robotun son eyleyicinin doğrudan konum ve yönelimlerini çıkartabilseydi, çok eksenli tezgahların daha efektif kullanmasını sağlayabilirlerdi. Bu bağlamda G kodlarının yeni bir endüstriyel standartın geliştirilmesi gerekmektedir.

7.1. Mekanik Öneriler

Bu çalışmanın öncelikli amacı ticari bir CNC makinesinden daha çok matematiksel yapıların doğruluğunu ve sistem alternatiflerin araştırması üzerinedir. Elde edilen veriler bize aşağıda belirtilen mekanik iyileştirmelerin yapılmasını ortaya koymuştur.

Robotun sabit ve hareketli platformlarını birleştiren, omuz eklemi aktüatör somun yataklı kardan mafsalı ile birlikte bilek eklemine doğrudan kardan mafsalı ile bağlıdır. Kullanılan kardan mafsallar bu çalışmada kolay bir tasarım sunsa da bazı handikaplarıda vardır. Bunlardan birisi kardan mafsalların aynı düzlemlerde çalışma zorunluluğudur. Bu zorunluluktan kaynaklanan farklardan, hatalı aktüatör konumlarına neden olmaktadır. Bu farklar hesaplanıp veya kapalı döngü kontrol metodu kullanılarak giderilebilir. Fakat bunun yanında kardan mafsalların istavroz uzuvlarından dolayı hassas üretilmeleri zordur. Dolayısıyla kardan mafsal kullanımı yerine yatay U bağlantılar kullanılabilir.

Aktüatörler temelde paslanmaz çelikten imal edilmiş M6 gijon vidalardan oluşmaktadır. Robot omuzlarında bu gijonlar ileri geri hareket ettiren pirinç somunlar bu miller üzerinde sürtünme kuvvetleri oluşturmaktadır. Bu yapının kullanımı yerine sürtünme ve geri gelme hataları az olan"backlash error" bilyalı vidaların kullanımı tercih edildiği takdirde daha efektif bir yapı elde edilebilir. Dahası bu aktüatörler yerine servo hidrolik veya bu iş için üretilmiş hazır mekanik aktüratörlerde kullanılabilir.

7.2. Elektronik Öneriler

Sistem açık döngü kontrol metodu ile kontrol edilmiştir. Bu durum step motor atlamaları veya mekanik sıkışmalar durumunda kaçırılan adımlar hakkında sistem habersiz kalmaktadır. Bunu önlemek için kapalı döngü kontrol medotu kullanılabilir.

Bu metotla daha stabil bir yapı elde edilebilir. Veri akışını yazılımdan alan ve motorları gerçek zamanlı kontrolünü sağlayan hareket kontrol kartı Atmel mega 328p işlemci tabanlı arduino nano geliştirme kartı kullanılmıştır. Bu işlemci bu deneysel çalışma için yeterli gelse de, sınırlı 32 KB lık program belliği ve 16MHz işlemci frekansı maksimum 1000 puls/saniye hızlarda step motor palsi üretmiştir. Bu da mevcut aktüatörlerde maksimium hızın saniyede 2.5mm ile sınırlandırmıştır. Bunun üstesinde gelmek için daha hızlı işlemciler kullanılarak veya görevi pals üretmek olan mesa fpga gibi kullanılabilir.

KAYNAKLAR DİZİNİ

Akbari, M. ve Kheibari, H. Z. (2013), *Timing belt gearbox in Ballbot robot*.

Branch, T., Liu, H., Nyugen, D. (2018), *Stewart Platform with Electronic Control and Leap Motion Interaction*. 1-7.

Geng, Z., Haynes, L. S. (1993), Six -Degree-of-Freedom Active Vibration Isolation Using a Stewart Platform Mechanism, *J. Robotic Systems* 10. N.5, 725-744

Hung, K. H. (1978), *Kinematic Geometry of Mechanisms*.

Khail, W., Dombre, E. (2002), *Modeling*, Identification & Control of Robots. 174-176.

Kim, Y. S., Shi, H., Dagalakis, N., Marvcl, J., Cheoka, G. (2019), *Mechanism and Machine Theory*. 84-94

Kizir, S., Bingül, Z. (2019), *Design and development of a Stewart platform assisted and navigated transsphenoidal surgery*. 1.

Linuxcnc, (2019), genhexkin.h, genhexkins.c, https://github.com/LinuxCNC/linuxcnc/tree/ master/src/emc/kinematics

Merlet, J. P., (2006), *Parallel Robots*. 98-103.

McCann, C. M., Dollar, A. M. (2018), *Analysis and dimen-sional synthesis of a robotic hand base on the stewart-gough platform*. 1-5.

Neeraj, B., Jonathan, K., Elvis, E. (2018), *Kinematic Analysis of 3 Designed Gough-Stewart Platform using CAD and Mathematics Software*. 1-5

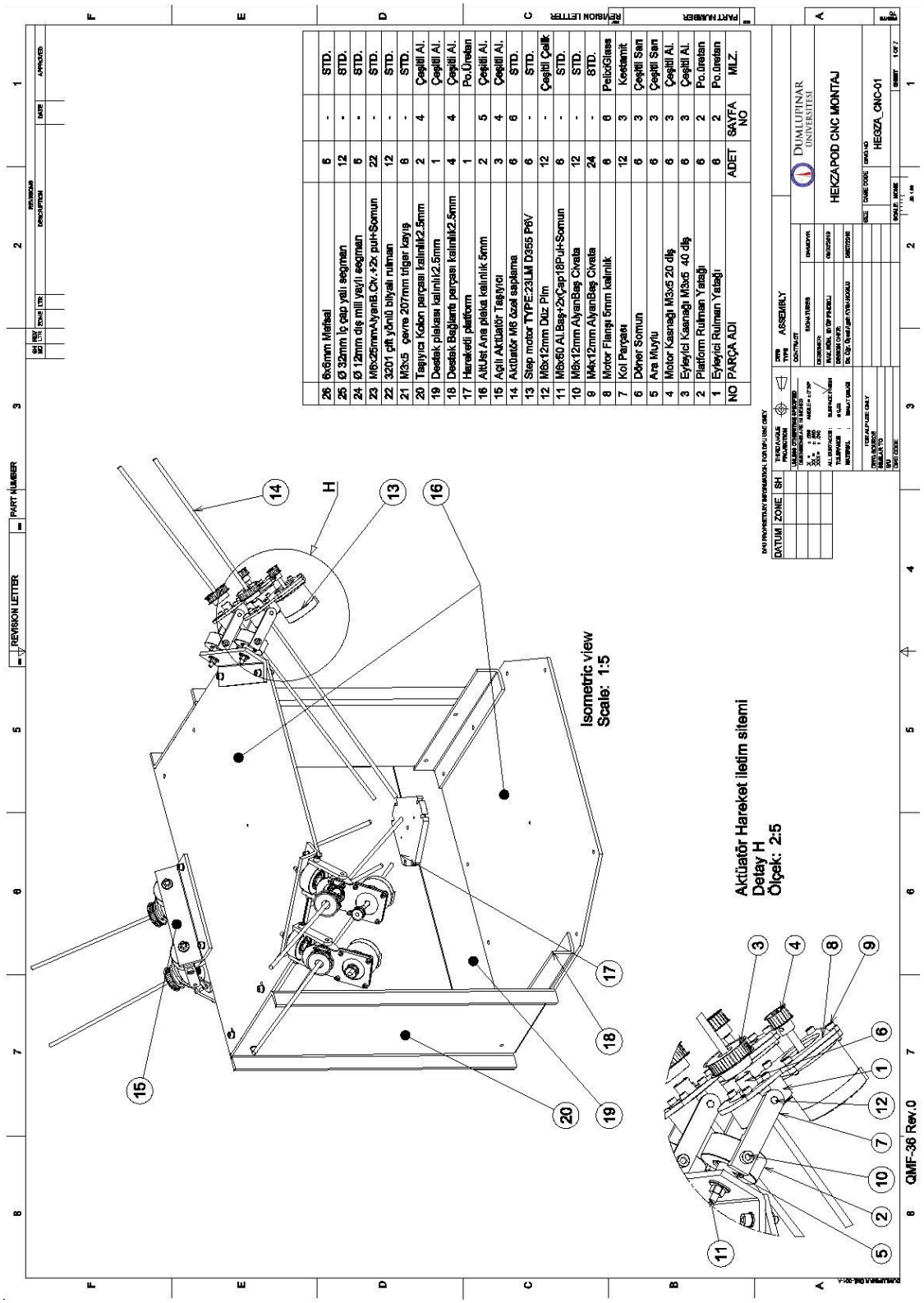
Okuma, (2010), https://www.youtube.com.com/watch?v=sPS096dUVmw.

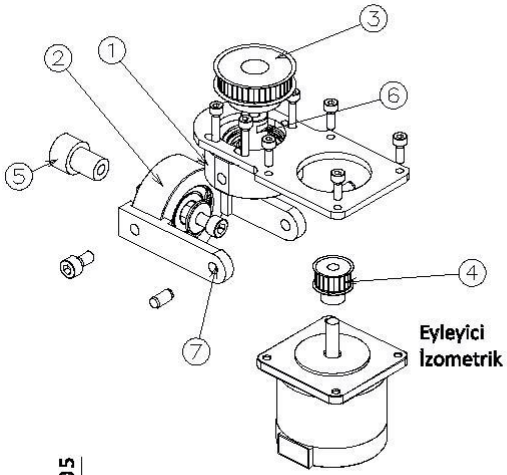
Pkmcnc, (2012), http://parallelrobots.blogspot.com/2012/. 1

Wikipedia, (2019), https://en.wikipedia.org/wiki/Octree.

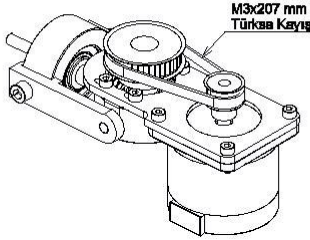
EKLER

Ek 1. Paralel kinematikli 6 aktüatörlü (Hexapod) robotun teknik resimleri



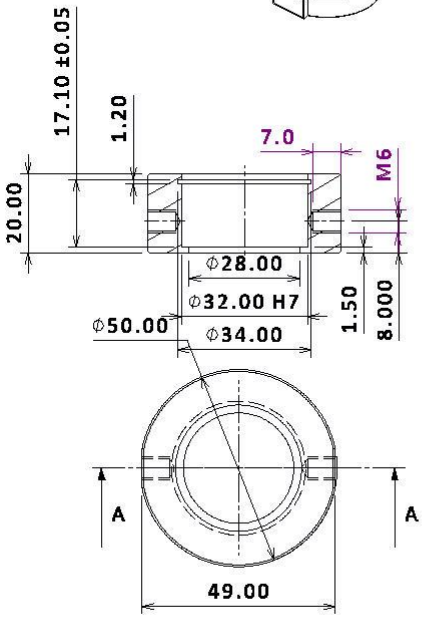


Eyleyici
İzometrik demontaj göünüşü



M3x207 mm 6mm kalıkta triger kayış
Türksa Kayıştarı Temin Edilebilir

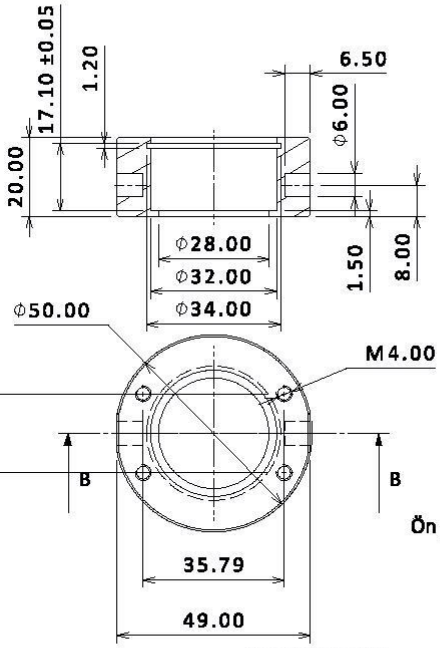
Eyleyici
İzometrik montaj göünüşü



②
Platform rulman yatağı
x6 Adet

A-A kesit görünüşü

Ön görünüş

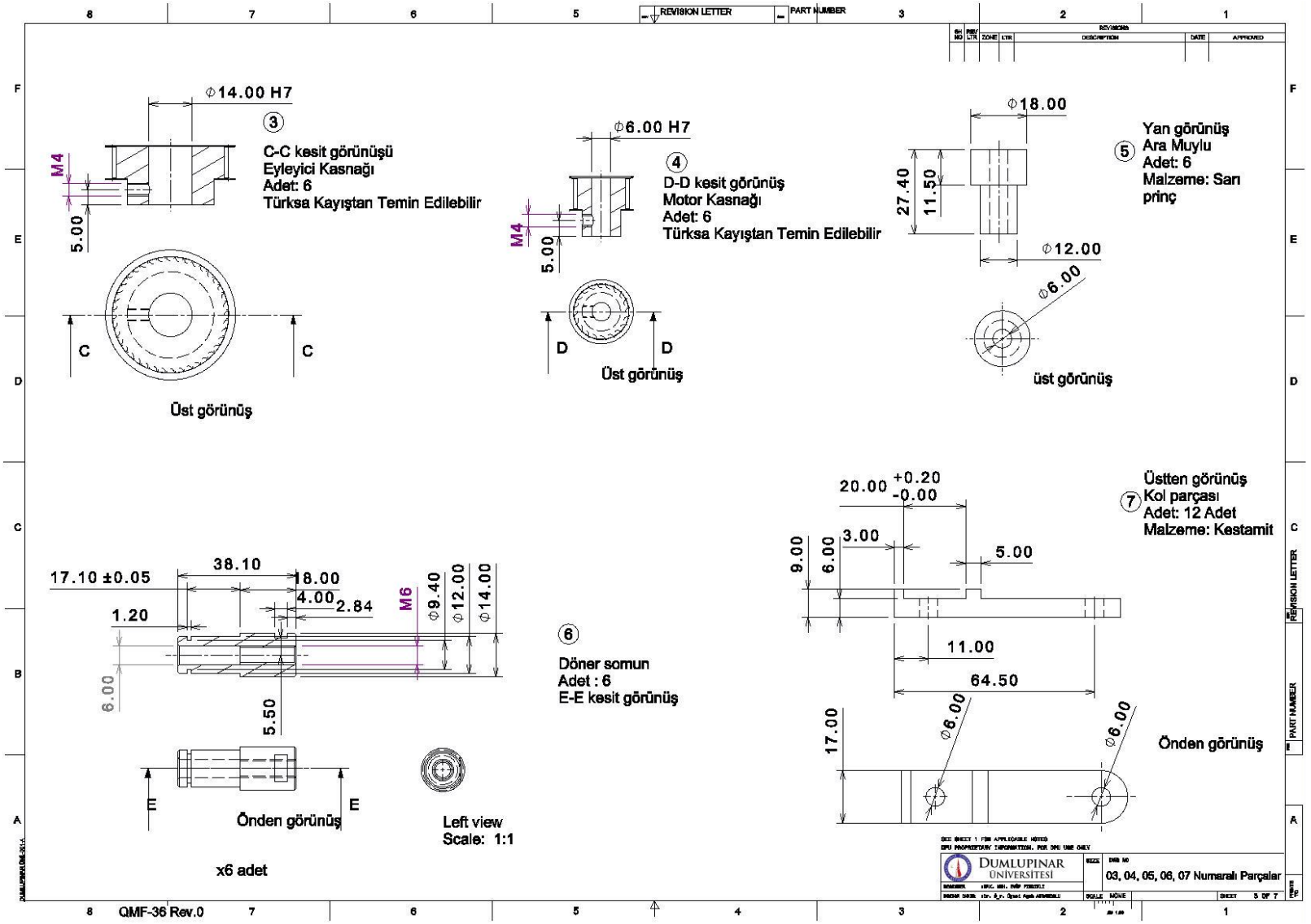


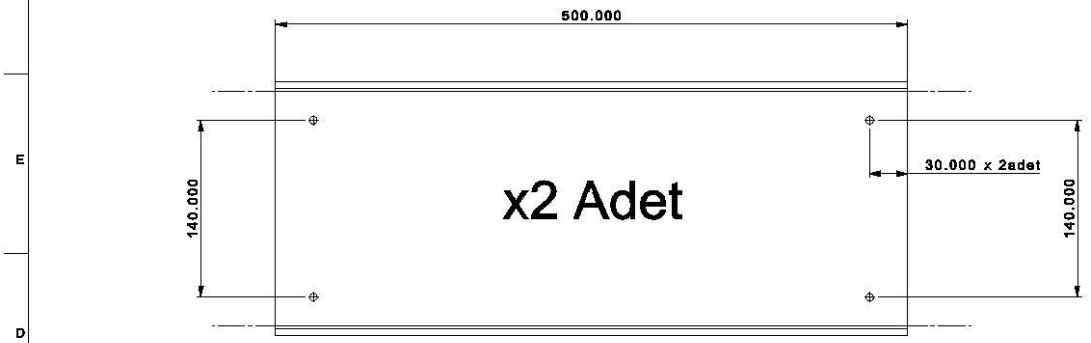
①
Eyleyici rulman yatağı
x6 adet

B-B kesit görünüşü

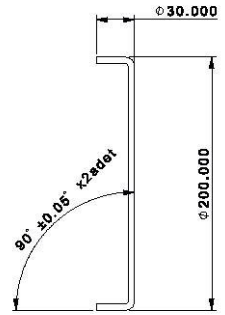
Ön görünüş

REVISION LETTER
PART NUMBER
REVISIONS
DESCRIPTION
DATE
APPROVED
DUMLUPINAR ÜNİVERSİTESİ
Hareket sistemi Montaj,
01 Parçası, 02 Parçası
SCALE NONE
SHEET 2 OF 7
QMF-36 Rev.0

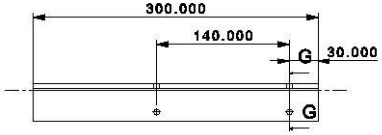

Φ14.00 H7
③
C-C kesit görünüşü
Eyleyici Kasnağı
Adet: 6
Türksa Kayıştan Temin Edilebilir
M4
5.00
C
C
Üst görünüş
Φ6.00 H7
④
D-D kesit görünüş
Motor Kasnağı
Adet: 6
Türksa Kayıştan Temin Edilebilir
M4
5.00
D
D
Üst görünüş
Φ18.00
27.40
11.50
Φ12.00
Φ6.00
⑤
Yan görünüş
Ara Muylu
Adet: 6
Malzeme: Sarı
prinç
üst görünüş
17.10 ±0.05
38.10
18.00
4.00
2.84
1.20
M6
Φ9.40
Φ12.00
Φ14.00
6.00
5.50
E
E
Önden görünüş
⑥
Döner somun
Adet : 6
E-E kesit görünüş
Left view
Scale: 1:1
x6 adet
20.00 +0.20 -0.00
3.00
5.00
9.00
6.00
11.00
64.50
17.00
Φ6.00
Φ6.00
⑦
Üstten görünüş
Kol parçası
Adet: 12 Adet
Malzeme: Kestamit
Önden görünüş
DUMLUPINAR ÜNİVERSİTESİ
03, 04, 05, 06, 07 Numaralı Parçalar
SCALE NONE
SHEET 3 OF 7
REVISION LETTER
PART NUMBER
QMF-36 Rev.0



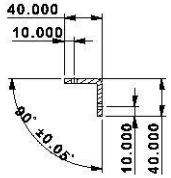
(20) Ön görünüş
Taşıyıcı kolon parcası
Adet: 2
Malzeme: kalınlık 2.5 Al. plaka



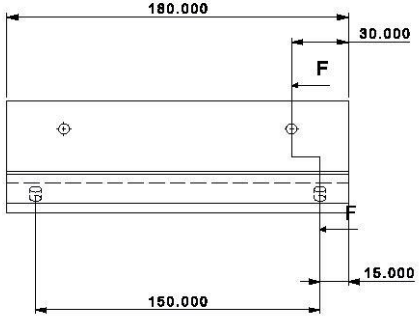
Yan görünüş



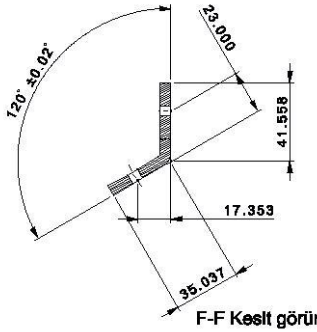
(18) Ön görünüş
Destek Bağlantı Parçası
Adet: 4
Malzeme: kalınlık 2.5 Al. plaka



G-G Kesit görünüşü



(15) Öngörünüş
Açılı Aktüatör Taşıyıcı
Adet: 3
Malzeme: kalınlık 2.5 Al. plaka



F-F Kesit görünüş

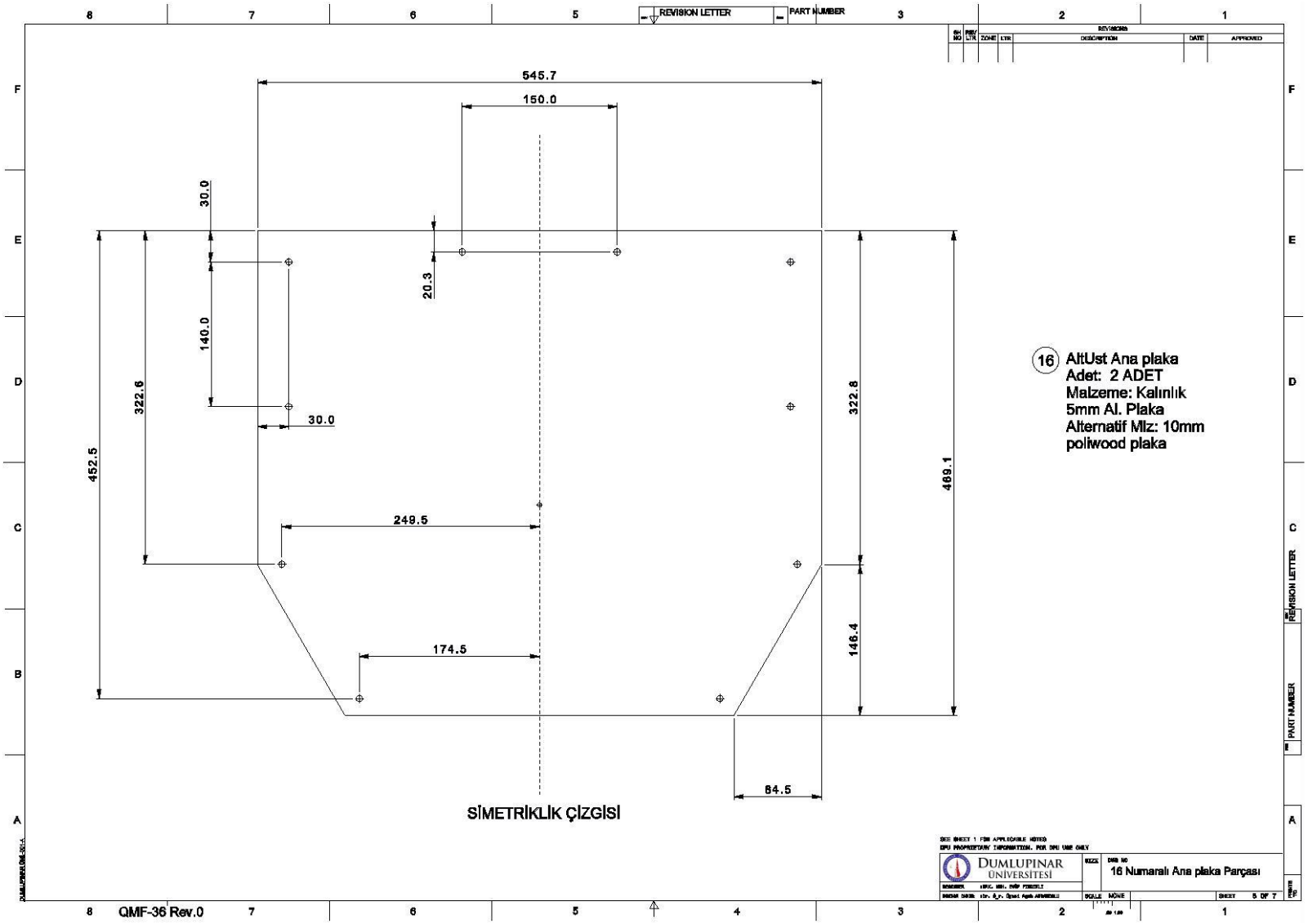
DUMLUPINAR ÜNİVERSİTESİ

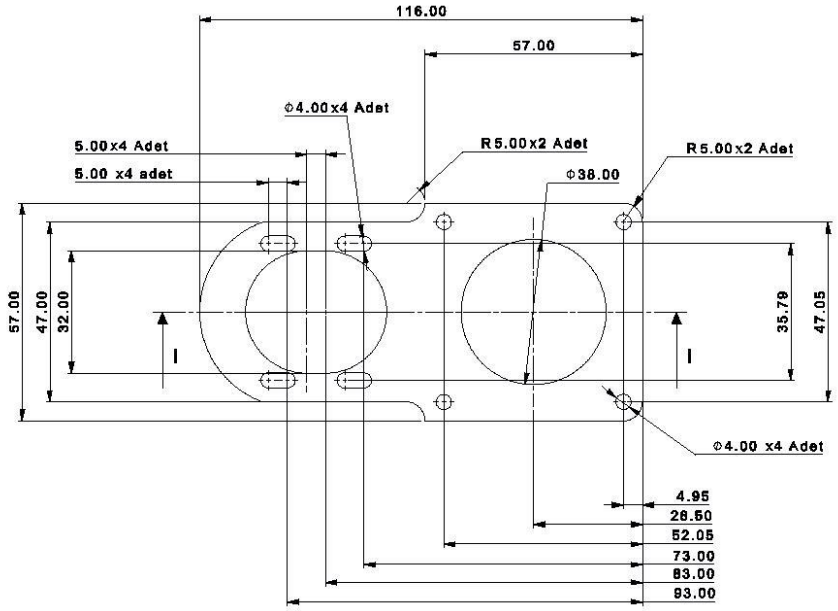
15, 18, 20 Numaralı Parçalar

SCALE NONE

SHEET 4 OF 7

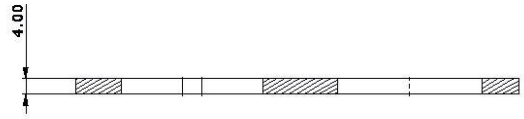
QMF-36 Rev.0


545.7
150.0
30.0
20.3
140.0
322.6
30.0
452.5
249.5
322.8
469.1
146.4
174.5
84.5
SİMETRİKLİK ÇİZGİSİ
16 AltUst Ana plaka
Adet: 2 ADET
Malzeme: Kalınlık
5mm Al. Plaka
Alternatif Mlz: 10mm
poliwood plaka
DUMLUPINAR ÜNİVERSİTESİ
16 Numaralı Ana plaka Parçası
QMF-36 Rev.0

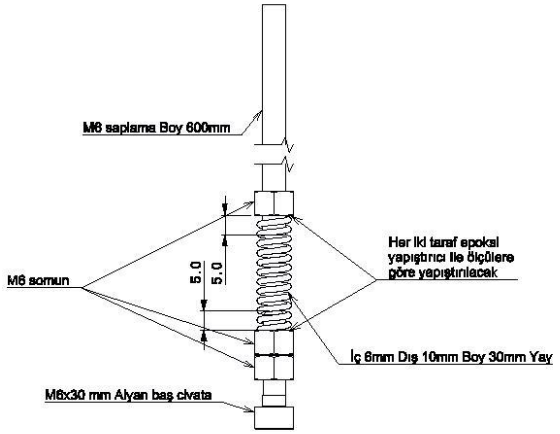


Ön görünüş
Ölçek: 1:1

(8) Motor Bağlantı flanşı
Malzeme: Aluminyum , ALTER. 3mm Genel İmalat Sac Çelik
Addet: 6



I-I Kesiti
Ölçek: 1:1



Not: Yay bağlantısı yerine 6x6mm Mafsal kuulanılabilir.

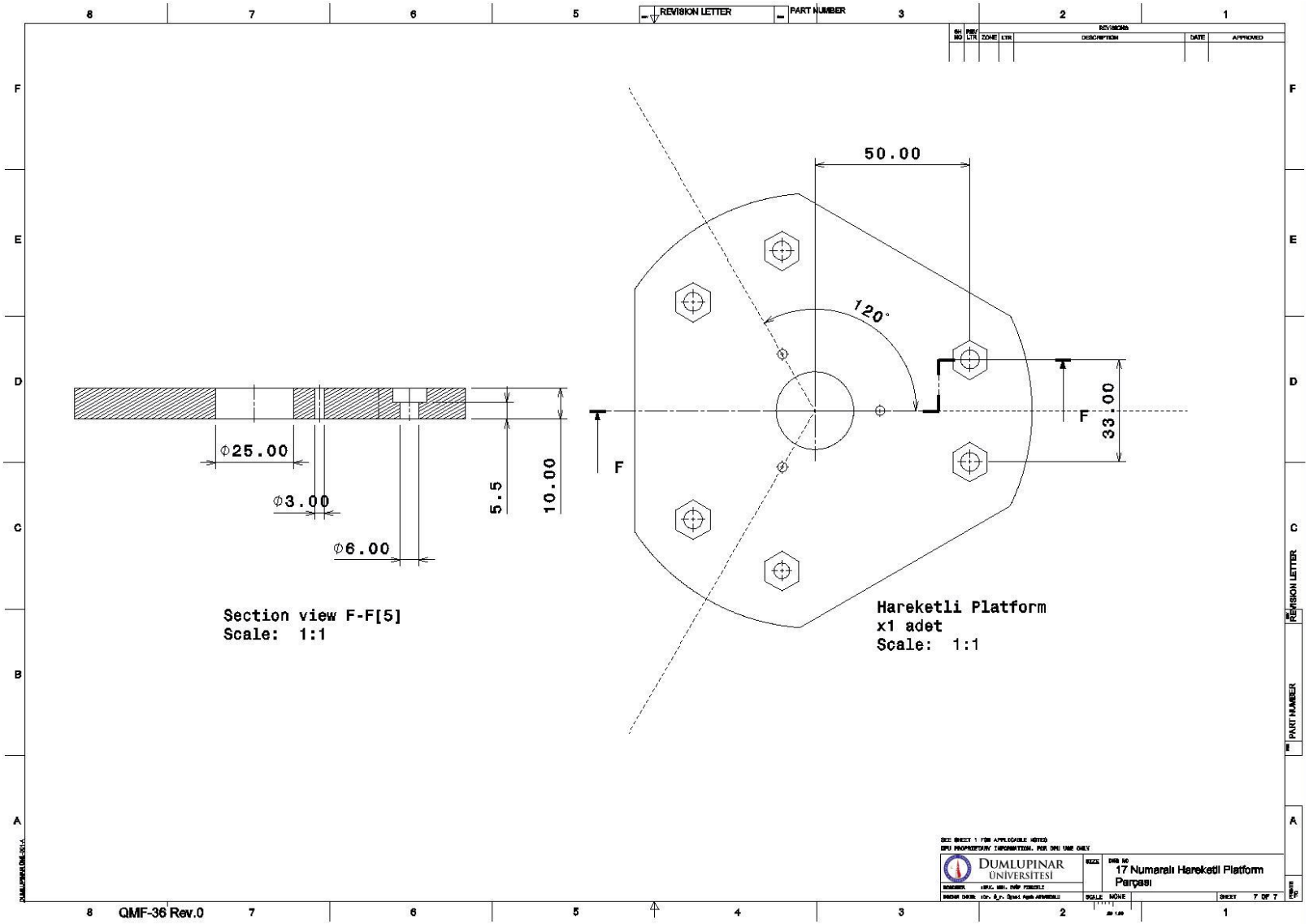
(14) Ön görünüş
Aktüatör M6 özel saplama
Adet: 6

DUMLUPINAR ÜNİVERSİTESİ

08, 14 Numaralı Parçalar

SCALE NONE

SHEET 6 OF 7


50.00
120°
33.00
F
F
Φ25.00
Φ3.00
Φ6.00
5.5
10.00
Section view F-F[5]
Scale: 1:1
Hareketli Platform
x1 adet
Scale: 1:1
DUMLUPINAR ÜNİVERSİTESİ
17 Numaralı Hareketli Platform Parçası
SCALE NONE
SHEET 7 OF 7
QMF-36 Rev.0

Ek 2. Paralel kinematikli 6 aktüatörlü (Hexapod) robotun aygıt yazılım kodları

```
#include "EEPROM_oku_yaz.h"

#include "ConstStepGen.h"

#include "Spindle.h"


const int Motor_pin = 9;

const int link0_step = A0;

const int link0_dir = A1;

const int link1_step = A2;

const int link1_dir = A3;

const int link2_step = A4;

const int link2_dir = A5;

const int link3_step = 2;

const int link3_dir = 3;

const int link4_step = 4;

const int link4_dir = 5;

const int link5_step = 6;

const int link5_dir = 7;

const int link0_address = 0; // software encoder için long int değerinin yazılacağı başlangıç eeprom adresi

const int link1_address = 4;

const int link2_address = 8;

const int link3_address = 12;

const int link4_address = 16;

const int link5_address = 20;
```

```
const int Power = 10;
//const int matrixEncoder_adress = 24;
//double matrixEncoder[4][4];

String gelenSeriBilgi = "", gidenSeriBilgi = "";
char c;
float negTamSayi, t0, tn;
boolean readyForNewMotion;
boolean isPowerDown;
boolean isPowerDownTriggered = false;
String absLinkSize1, absLinkSize2, absLinkSize3, absLinkSize4, absLinkSize5, absLinkSize6, desiredDeltaTime7;
String MatrixXX, MatrixXY, MatrixXZ, MatrixX0;
String MatrixYX, MatrixYY, MatrixYZ, MatrixY0;
String MatrixZX, MatrixZY, MatrixZZ, MatrixZ0;
String MatrixWX, MatrixWY, MatrixWZ, MatrixW0;

Spindle spindleMotor(Motor_pin);
ConstStepGen stepper0(link0_step, link0_dir, link0_address);
ConstStepGen stepper1(link1_step, link1_dir, link1_address);
ConstStepGen stepper2(link2_step, link2_dir, link2_address);
ConstStepGen stepper3(link3_step, link3_dir, link3_address);
ConstStepGen stepper4(link4_step, link4_dir, link4_address);
ConstStepGen stepper5(link5_step, link5_dir, link5_address);
```

```
void setup() {
  spindleMotor.invert();

  Serial.begin(115200);
  delay(1);

  pinMode(8, OUTPUT);

  stepper0.setMaxSpeed(1000);
  stepper0.setSpeed(1000);
  stepper0.setCurrentPositionFromEEPROM();
  stepper1.setMaxSpeed(1000);
  stepper1.setSpeed(1000);
  stepper1.setCurrentPositionFromEEPROM();
  stepper2.setMaxSpeed(1000);
  stepper2.setSpeed(1000);
  stepper2.setCurrentPositionFromEEPROM();
  stepper3.setMaxSpeed(1000);
  stepper3.setSpeed(1000);
  stepper3.setCurrentPositionFromEEPROM();
  stepper4.setMaxSpeed(1000);
  stepper4.setSpeed(1000);
  stepper4.setCurrentPositionFromEEPROM();
  stepper5.setMaxSpeed(1000);
```

```
  stepper5.setSpeed(1000);
  stepper5.setCurrentPositionFromEEPROM();

  //  fillMatrix();
  //  matrisYaz(matrixEncoder_adress);
  //  matrixEncoder[0][3] = 2.0;

  readyForNewMotion = false;

}

void loop() {
  //
  isPowerDown = digitalRead(Power);

  if( isPowerDown == LOW)
  {
    if(isPowerDownTriggered == true)
    {
      isPowerDownTriggered = false;
       Serial.println(">>POWERUP");
    }
      //eğer acil durum değilse run yap  spindle rölesi ????
      stepper0.run();
```

```
    stepper1.run();
    stepper2.run();
    stepper3.run();
    stepper4.run();
    stepper5.run();
    if (!(isMotion()) && readyForNewMotion) {
      stepper0.stop(); //mevcut konumlarını epprom a yazısınlar
      stepper1.stop();
      stepper2.stop();
      stepper3.stop();
      stepper4.stop();
      stepper5.stop();
      readyForNewMotion = false;
      Serial.println(">>");//hareket bitti yeni hareket için >> karakteri gönderiliyor
    }
}
else
{
  if(isPowerDownTriggered == false)
  {
    stepper0.stop(); //mevcut konumlarını epprom a yazısınlar
    stepper1.stop();
    stepper2.stop();
    stepper3.stop();
```

```
      stepper4.stop();
      stepper5.stop();
      readyForNewMotion = false;
      isPowerDownTriggered = true;
      Serial.println(">>POWERDOWN");
    }
  }


  if (Serial.available()) {
    char c = Serial.read();
    if (c == '\n') {
      parseSeriBilgi();
      gelenSeriBilgi = "";
    }
    else {
      gelenSeriBilgi += c;
    }
  }
}
boolean isMotion() {
  boolean flag = false;
  if (stepper0.isRunning())flag = true;
  if (stepper1.isRunning())flag = true;
```

```
  if (stepper2.isRunning())flag = true;
  if (stepper3.isRunning())flag = true;
  if (stepper4.isRunning())flag = true;
  if (stepper5.isRunning())flag = true;

  return flag;
}

void parseSeriBilgi() {

  if ((gelenSeriBilgi.substring(0, 3) == ">>d"))
  {
    //Serial.println(gelenSeriBilgi.substring(0, 3));
    //PC mevcut durumu istiyor;
    durumuGonder();
  }
  else if ((gelenSeriBilgi.substring(0, 3) == ">>s")) {
    //Serial.println(gelenSeriBilgi.substring(0, 3));
    //istenilen değerlere set ediliyor
    durumuKur();
  }
  else {
    isPowerDownTriggered = false;
    //hareket yapması sağlanıyor
```

```
    hareketYap();
  }
}
void durumuGonder() {
  Serial.print(stepper0.getCurrentPosition());
  Serial.print(" ");
  Serial.print(stepper1.getCurrentPosition());
  Serial.print(" ");
  Serial.print(stepper2.getCurrentPosition());
  Serial.print(" ");
  Serial.print(stepper3.getCurrentPosition());
  Serial.print(" ");
  Serial.print(stepper4.getCurrentPosition());
  Serial.print(" ");
  Serial.print(stepper5.getCurrentPosition());
  Serial.println("");
}

void hareketYap() {
  absLinkSize1 = gelenSeriBilgi.substring(0, gelenSeriBilgi.indexOf(" "));
  gelenSeriBilgi = gelenSeriBilgi.substring(gelenSeriBilgi.indexOf(" ") + 1);
  absLinkSize2 = gelenSeriBilgi.substring(0, gelenSeriBilgi.indexOf(" "));
  gelenSeriBilgi = gelenSeriBilgi.substring(gelenSeriBilgi.indexOf(" ") + 1);
  absLinkSize3 = gelenSeriBilgi.substring(0, gelenSeriBilgi.indexOf(" "));
```

```
  gelenSeriBilgi = gelenSeriBilgi.substring(gelenSeriBilgi.indexOf(" ") + 1);
  absLinkSize4 = gelenSeriBilgi.substring(0, gelenSeriBilgi.indexOf(" "));
  gelenSeriBilgi = gelenSeriBilgi.substring(gelenSeriBilgi.indexOf(" ") + 1);
  absLinkSize5 = gelenSeriBilgi.substring(0, gelenSeriBilgi.indexOf(" "));
  gelenSeriBilgi = gelenSeriBilgi.substring(gelenSeriBilgi.indexOf(" ") + 1);
  absLinkSize6 = gelenSeriBilgi.substring(0, gelenSeriBilgi.indexOf(" "));
  gelenSeriBilgi = gelenSeriBilgi.substring(gelenSeriBilgi.indexOf(" ") + 1);
  desiredDeltaTime7 = gelenSeriBilgi.substring(0, gelenSeriBilgi.indexOf(" "));
  stepper0.setMotion(absLinkSize1.toFloat(), desiredDeltaTime7.toFloat());
  stepper1.setMotion(absLinkSize2.toFloat(), desiredDeltaTime7.toFloat());
  stepper2.setMotion(absLinkSize3.toFloat(), desiredDeltaTime7.toFloat());
  stepper3.setMotion(absLinkSize4.toFloat(), desiredDeltaTime7.toFloat());
  stepper4.setMotion(absLinkSize5.toFloat(), desiredDeltaTime7.toFloat());
  stepper5.setMotion(absLinkSize6.toFloat(), desiredDeltaTime7.toFloat());

  readyForNewMotion = true;
 }

 void durumuKur() {
  absLinkSize1 = gelenSeriBilgi.substring(3, gelenSeriBilgi.indexOf(" "));
  gelenSeriBilgi = gelenSeriBilgi.substring(gelenSeriBilgi.indexOf(" ") + 1);
  absLinkSize2 = gelenSeriBilgi.substring(0, gelenSeriBilgi.indexOf(" "));
  gelenSeriBilgi = gelenSeriBilgi.substring(gelenSeriBilgi.indexOf(" ") + 1);
  absLinkSize3 = gelenSeriBilgi.substring(0, gelenSeriBilgi.indexOf(" "));
```

```
 gelenSeriBilgi = gelenSeriBilgi.substring(gelenSeriBilgi.indexOf(" ") + 1);
 absLinkSize4 = gelenSeriBilgi.substring(0, gelenSeriBilgi.indexOf(" "));
 gelenSeriBilgi = gelenSeriBilgi.substring(gelenSeriBilgi.indexOf(" ") + 1);
 absLinkSize5 = gelenSeriBilgi.substring(0, gelenSeriBilgi.indexOf(" "));
 gelenSeriBilgi = gelenSeriBilgi.substring(gelenSeriBilgi.indexOf(" ") + 1);
 absLinkSize6 = gelenSeriBilgi.substring(0, gelenSeriBilgi.indexOf(" "));
 stepper0.setCurrentPosition(absLinkSize1.toInt());
 stepper1.setCurrentPosition(absLinkSize2.toInt());
 stepper2.setCurrentPosition(absLinkSize3.toInt());
 stepper3.setCurrentPosition(absLinkSize4.toInt());
 stepper4.setCurrentPosition(absLinkSize5.toInt());
 stepper5.setCurrentPosition(absLinkSize6.toInt());

 stepper0.stop(); //mevcut konumlarını epprom a yazısınlar
 stepper1.stop();
 stepper2.stop();
 stepper3.stop();
 stepper4.stop();
 stepper5.stop();

 Serial.println(">>s OK");
 // readyForNewMotion = true;
}
```

Ek 3. Paralel kinematikli 6 aktüatörlü (Hexapod) robotun ters kinematik fonsiyon delphi dili yazılım kodları

```
procedure TFMain.AcilariHesapla;
var
  BuffV, Q1N, Q2N, Q2NN, Q1NN: TAffineVector;
  TM: TMatrix4f;
  Q1, Q2, Q1deg, Q2deg: Double;
begin
  LinkV[0] := VectorSubtract(OmuzBaseMafsal0.AbsoluteAffinePosition,
BilekBaseMafsal0.AbsoluteAffinePosition);
  LinkLine0.Nodes.Clear;
  LinkLine0.Nodes.AddNode(OmuzBaseMafsal0.AbsoluteAffinePosition);
  LinkLine0.Nodes.AddNode(BilekBaseMafsal0.AbsoluteAffinePosition);
  LinkN[0] := VectorNormalize(LinkV[0]);
  Link[0] := VectorLength(LinkV[0]);
  GLGameMenu1.Items[0] := 'Link 0= ' + FloatToStr(Link[0]);

  LinkV[1] := VectorSubtract(OmuzBaseMafsal1.AbsoluteAffinePosition,
BilekBaseMafsal1.AbsoluteAffinePosition);
  LinkLine1.Nodes.Clear;
  LinkLine1.Nodes.AddNode(OmuzBaseMafsal1.AbsoluteAffinePosition);
  LinkLine1.Nodes.AddNode(BilekBaseMafsal1.AbsoluteAffinePosition);
  LinkN[1] := VectorNormalize(LinkV[1]);
  Link[1] := VectorLength(LinkV[1]);
```

```
GLGameMenu1.Items[1] := 'Link 1= ' + FloatToStr(Link[1]);

LinkV[2] := VectorSubtract(OmuzBaseMafsal2.AbsoluteAffinePosition,
BilekBaseMafsal2.AbsoluteAffinePosition);
LinkLine2.Nodes.Clear;
LinkLine2.Nodes.AddNode(OmuzBaseMafsal2.AbsoluteAffinePosition);
LinkLine2.Nodes.AddNode(BilekBaseMafsal2.AbsoluteAffinePosition);
LinkN[2] := VectorNormalize(LinkV[2]);
Link[2] := VectorLength(LinkV[2]);
GLGameMenu1.Items[2] := 'Link 2= ' + FloatToStr(Link[2]);

LinkV[3] := VectorSubtract(OmuzBaseMafsal3.AbsoluteAffinePosition,
BilekBaseMafsal3.AbsoluteAffinePosition);
LinkLine3.Nodes.Clear;
LinkLine3.Nodes.AddNode(OmuzBaseMafsal3.AbsoluteAffinePosition);
LinkLine3.Nodes.AddNode(BilekBaseMafsal3.AbsoluteAffinePosition);
LinkN[3] := VectorNormalize(LinkV[3]);
Link[3] := VectorLength(LinkV[3]);
GLGameMenu1.Items[3] := 'Link 3= ' + FloatToStr(Link[3]);

LinkV[4] := VectorSubtract(OmuzBaseMafsal4.AbsoluteAffinePosition,
BilekBaseMafsal4.AbsoluteAffinePosition);
LinkLine4.Nodes.Clear;
LinkLine4.Nodes.AddNode(OmuzBaseMafsal4.AbsoluteAffinePosition);
LinkLine4.Nodes.AddNode(BilekBaseMafsal4.AbsoluteAffinePosition);
```

```
LinkN[4] := VectorNormalize(LinkV[4]);

Link[4] := VectorLength(LinkV[4]);

GLGameMenu1.Items[4] := 'Link 4= ' + FloatToStr(Link[4]);


LinkV[5] := VectorSubtract(OmuzBaseMafsal5.AbsoluteAffinePosition,
BilekBaseMAfsal5.AbsoluteAffinePosition);

LinkLine5.Nodes.Clear;

LinkLine5.Nodes.AddNode(OmuzBaseMafsal5.AbsoluteAffinePosition);

LinkLine5.Nodes.AddNode(BilekBaseMAfsal5.AbsoluteAffinePosition);

LinkN[5] := VectorNormalize(LinkV[5]);

Link[5] := VectorLength(LinkV[5]);

GLGameMenu1.Items[5] := 'Link 5= ' + FloatToStr(Link[5]);


// 0 ıncı bilek -kol doğrultma

BuffV := VectorAdd(LinkN[0],
AffineVectorMake(BilekBaseMafsal0.AbsolutePosition));

BuffV := BilekBaseMafsal0.AbsoluteToLocal(BuffV);

// BuffV burada hala BilekBaseMafsal10 uzayında;

// Q1N BilekBaseMafsal10 matrisinin yeni Z ti olacak fakat bi önceki eyleyici
matrisine taşınması lazım

Q1N := VectorNormalize(AffineVectorMake(BuffV.X, 0, BuffV.Z));

Q1N := VectorSubtract(VectorTransform(Q1N, (BilekBaseMafsal0.Matrix)^ ),
BilekBaseMafsal0.Position.AsAffineVector);

// Q2N BilekKol10 Matrisinin yeni Z ti olacak önceki matrise taşınmasına gerek yok

Q2N := VectorNormalize(AffineVectorMake(0, BuffV.Y, BuffV.Z));
```

```
Q2NN := CalcPlaneNormal(Q2N, AffineVectorMake(0, 0, 0), AffineVectorMake(1, 0, 0));

TM.X := VectorMake(AffineVectorMake(1, 0, 0), 0);

TM.Y := VectorMake(Q2NN, 0);

TM.Z := VectorMake(Q2N, 0);

TM.W := VectorMake(0, 0, 0, 1);

BilekKol0.SetMatrix(TM);


Q1NN := CalcPlaneNormal(Q1N, AffineVectorMake(0, 0, 0), AffineVectorMake(BilekBaseMafsal0.Matrix.Y));

TM := (BilekBaseMafsal0.Matrix)^;

TM.X := VectorMake(Q1NN, 0);

TM.Z := VectorMake(Q1N, 0);

BilekBaseMafsal0.SetMatrix(TM);


// 1 ıncı bilek -kol doğrultma

BuffV := VectorAdd(LinkN[1], AffineVectorMake(BilekBaseMafsal1.AbsolutePosition));

BuffV := BilekBaseMafsal1.AbsoluteToLocal(BuffV);

// BuffV burada hala BilekBaseMafsal10 uzayında;

// Q1N BilekBaseMafsal10 matrisinin yeni Z ti olacak fakat bi önceki eyleyici matrisine taşınması lazım

Q1N := VectorNormalize(AffineVectorMake(BuffV.X, 0, BuffV.Z));

Q1N := VectorSubtract(VectorTransform(Q1N, (BilekBaseMafsal1.Matrix)^ ), BilekBaseMafsal1.Position.AsAffineVector);

// Q2N BilekKol10 Matrisinin yeni Z ti olacak önceki matrise taşınmasına gerek yok
```

```
Q2N := VectorNormalize(AffineVectorMake(0, BuffV.Y, BuffV.Z));

Q2NN := CalcPlaneNormal(Q2N, AffineVectorMake(0, 0, 0), AffineVectorMake(1, 0, 0));

TM.X := VectorMake(AffineVectorMake(1, 0, 0), 0);

TM.Y := VectorMake(Q2NN, 0);

TM.Z := VectorMake(Q2N, 0);

TM.W := VectorMake(0, 0, 0, 1);

BilekKol1.SetMatrix(TM);


Q1NN := CalcPlaneNormal(Q1N, AffineVectorMake(0, 0, 0), AffineVectorMake(BilekBaseMafsal1.Matrix.Y));

TM := (BilekBaseMafsal1.Matrix)^;

TM.X := VectorMake(Q1NN, 0);

TM.Z := VectorMake(Q1N, 0);

BilekBaseMafsal1.SetMatrix(TM);


// 2 ıncı bilek -kol doğrultma

BuffV := VectorAdd(LinkN[2], AffineVectorMake(BilekBaseMafsal2.AbsolutePosition));

BuffV := BilekBaseMafsal2.AbsoluteToLocal(BuffV);

// BuffV burada hala BilekBaseMafsal20 uzayında;

// Q1N BilekBaseMafsal20 matrisinin yeni Z ti olacak fakat bi önceki eyleyici matrisine taşınması lazım

Q1N := VectorNormalize(AffineVectorMake(BuffV.X, 0, BuffV.Z));

Q1N := VectorSubtract(VectorTransform(Q1N, (BilekBaseMafsal2.Matrix)^ ), BilekBaseMafsal2.Position.AsAffineVector);
```

```
// Q2N BilekKol10 Matrisinin yeni Z ti olacak önceki matrise taşınmasına gerek yok
Q2N := VectorNormalize(AffineVectorMake(0, BuffV.Y, BuffV.Z));
Q2NN := CalcPlaneNormal(Q2N, AffineVectorMake(0, 0, 0), AffineVectorMake(1, 0, 0));
TM.X := VectorMake(AffineVectorMake(1, 0, 0), 0);
TM.Y := VectorMake(Q2NN, 0);
TM.Z := VectorMake(Q2N, 0);
TM.W := VectorMake(0, 0, 0, 1);
BilekKol2.SetMatrix(TM);

Q1NN := CalcPlaneNormal(Q1N, AffineVectorMake(0, 0, 0), AffineVectorMake(BilekBaseMafsal2.Matrix.Y));
TM := (BilekBaseMafsal2.Matrix)^;
TM.X := VectorMake(Q1NN, 0);
TM.Z := VectorMake(Q1N, 0);
BilekBaseMafsal2.SetMatrix(TM);

// 3 ıncı bilek -kol doğrultma
BuffV := VectorAdd(LinkN[3], AffineVectorMake(BilekBaseMafsal3.AbsolutePosition));
BuffV := BilekBaseMafsal3.AbsoluteToLocal(BuffV);
// BuffV burada hala BilekBaseMafsal30 uzayında;
// Q1N BilekBaseMafsal30 matrisinin yeni Z ti olacak fakat bi önceki eyleyici matrisine taşınması lazım
Q1N := VectorNormalize(AffineVectorMake(BuffV.X, 0, BuffV.Z));
```

```
Q1N := VectorSubtract(VectorTransform(Q1N, (BilekBaseMafsal3.Matrix)^ ), BilekBaseMafsal3.Position.AsAffineVector);

// Q2N BilekKol10 Matrisinin yeni Z ti olacak önceki matrise taşınmasına gerek yok

Q2N := VectorNormalize(AffineVectorMake(0, BuffV.Y, BuffV.Z));

Q2NN := CalcPlaneNormal(Q2N, AffineVectorMake(0, 0, 0), AffineVectorMake(1, 0, 0));

TM.X := VectorMake(AffineVectorMake(1, 0, 0), 0);

TM.Y := VectorMake(Q2NN, 0);

TM.Z := VectorMake(Q2N, 0);

TM.W := VectorMake(0, 0, 0, 1);

BilekKol3.SetMatrix(TM);


Q1NN := CalcPlaneNormal(Q1N, AffineVectorMake(0, 0, 0), AffineVectorMake(BilekBaseMafsal3.Matrix.Y));

TM := (BilekBaseMafsal3.Matrix)^ ;

TM.X := VectorMake(Q1NN, 0);

TM.Z := VectorMake(Q1N, 0);

BilekBaseMafsal3.SetMatrix(TM);


// 4 ıncı bilek -kol doğrultma

BuffV := VectorAdd(LinkN[4], AffineVectorMake(BilekBaseMafsal4.AbsolutePosition));

BuffV := BilekBaseMafsal4.AbsoluteToLocal(BuffV);

// BuffV burada hala BilekBaseMafsal40 uzayında;

// Q1N BilekBaseMafsal40 matrisinin yeni Z ti olacak fakat bi önceki eyleyici matrisine taşınması lazım
```

```
Q1N := VectorNormalize(AffineVectorMake(BuffV.X, 0, BuffV.Z));

Q1N := VectorSubtract(VectorTransform(Q1N, (BilekBaseMafsal4.Matrix)^ ),
BilekBaseMafsal4.Position.AsAffineVector);

// Q2N BilekKol10 Matrisinin yeni Z ti olacak önceki matrise taşınmasına gerek yok

Q2N := VectorNormalize(AffineVectorMake(0, BuffV.Y, BuffV.Z));

Q2NN := CalcPlaneNormal(Q2N, AffineVectorMake(0, 0, 0), AffineVectorMake(1, 0,
0));

TM.X := VectorMake(AffineVectorMake(1, 0, 0), 0);

TM.Y := VectorMake(Q2NN, 0);

TM.Z := VectorMake(Q2N, 0);

TM.W := VectorMake(0, 0, 0, 1);

BilekKol4.SetMatrix(TM);


Q1NN := CalcPlaneNormal(Q1N, AffineVectorMake(0, 0, 0),
AffineVectorMake(BilekBaseMafsal4.Matrix.Y));

TM := (BilekBaseMafsal4.Matrix)^;

TM.X := VectorMake(Q1NN, 0);

TM.Z := VectorMake(Q1N, 0);

BilekBaseMafsal4.SetMatrix(TM);


// 5 ıncı bilek -kol doğrultma

BuffV := VectorAdd(LinkN[5],
AffineVectorMake(BilekBaseMAfsal5.AbsolutePosition));

BuffV := BilekBaseMAfsal5.AbsoluteToLocal(BuffV);

// BuffV burada hala BilekBaseMafsal50 uzayında;
```

```
// Q1N BilekBaseMafsal50 matrisinin yeni Z ti olacak fakat bi önceki eyleyici matrisine taşınması lazım

Q1N := VectorNormalize(AffineVectorMake(BuffV.X, 0, BuffV.Z));

Q1N := VectorSubtract(VectorTransform(Q1N, (BilekBaseMAfsal5.Matrix)^ ), BilekBaseMAfsal5.Position.AsAffineVector);

// Q2N BilekKol10 Matrisinin yeni Z ti olacak önceki matrise taşınmasına gerek yok

Q2N := VectorNormalize(AffineVectorMake(0, BuffV.Y, BuffV.Z));

Q2NN := CalcPlaneNormal(Q2N, AffineVectorMake(0, 0, 0), AffineVectorMake(1, 0, 0));

TM.X := VectorMake(AffineVectorMake(1, 0, 0), 0);

TM.Y := VectorMake(Q2NN, 0);

TM.Z := VectorMake(Q2N, 0);

TM.W := VectorMake(0, 0, 0, 1);

BilekKol5.SetMatrix(TM);


Q1NN := CalcPlaneNormal(Q1N, AffineVectorMake(0, 0, 0), AffineVectorMake(BilekBaseMAfsal5.Matrix.Y));

TM := (BilekBaseMAfsal5.Matrix)^;

TM.X := VectorMake(Q1NN, 0);

TM.Z := VectorMake(Q1N, 0);

BilekBaseMAfsal5.SetMatrix(TM);


// 0 ıncı omuz -kol doğrultma

BuffV := VectorAdd(LinkN[0], AffineVectorMake(OmuzBaseMafsal0.AbsolutePosition));

BuffV := OmuzBaseMafsal0.AbsoluteToLocal(BuffV);
```

```
// BuffV burada hala BilekBaseMafsal10 uzayında;

// Q1N BilekBaseMafsal10 matrisinin yeni Z ti olacak fakat bi önceki eyleyici matrisine taşınması lazım

Q1N := VectorNormalize(AffineVectorMake(0, BuffV.Y, BuffV.Z));

Q1N := VectorSubtract(VectorTransform(Q1N, (OmuzBaseMafsal0.Matrix)^ ), OmuzBaseMafsal0.Position.AsAffineVector);

// Q2N BilekKol10 Matrisinin yeni Z ti olacak önceki matrise taşınmasına gerek yok

Q2N := VectorNormalize(AffineVectorMake(BuffV.X, 0, BuffV.Z));

Q2NN := CalcPlaneNormal(AffineVectorMake(0, 1, 0), AffineVectorMake(0, 0, 0), Q2N); // yeni x

TM.Y := VectorMake(AffineVectorMake(0, 1, 0), 0);

TM.X := VectorMake(Q2NN, 0);

TM.Z := VectorMake(Q2N, 0);

TM.W := VectorMake(0, 0, 0, 1);

OmuzKol0.SetMatrix(TM);


Q1NN := CalcPlaneNormal(AffineVectorMake(OmuzBaseMafsal0.Matrix.X), AffineVectorMake(0, 0, 0), Q1N); // yeni Y

TM := (OmuzBaseMafsal0.Matrix)^;

TM.Y := VectorMake(Q1NN, 0);

TM.Z := VectorMake(Q1N, 0);

OmuzBaseMafsal0.SetMatrix(TM);


// 1 ıncı omuz -kol doğrultma

BuffV := VectorAdd(LinkN[1], AffineVectorMake(OmuzBaseMafsal1.AbsolutePosition));
```

```
BuffV := OmuzBaseMafsal1.AbsoluteToLocal(BuffV);

// BuffV burada hala BilekBaseMafsal10 uzayında;

// Q1N BilekBaseMafsal10 matrisinin yeni Z ti olacak fakat bi önceki eyleyici matrisine taşınması lazım

Q1N := VectorNormalize(AffineVectorMake(0, BuffV.Y, BuffV.Z));

Q1N := VectorSubtract(VectorTransform(Q1N, (OmuzBaseMafsal1.Matrix)^ ), OmuzBaseMafsal1.Position.AsAffineVector);

// Q2N BilekKol10 Matrisinin yeni Z ti olacak önceki matrise taşınmasına gerek yok

Q2N := VectorNormalize(AffineVectorMake(BuffV.X, 0, BuffV.Z));

Q2NN := CalcPlaneNormal(AffineVectorMake(0, 1, 0), AffineVectorMake(0, 0, 0), Q2N); // yeni x

TM.Y := VectorMake(AffineVectorMake(0, 1, 0), 0);

TM.X := VectorMake(Q2NN, 0);

TM.Z := VectorMake(Q2N, 0);

TM.W := VectorMake(0, 0, 0, 1);

OmuzKol1.SetMatrix(TM);


Q1NN := CalcPlaneNormal(AffineVectorMake(OmuzBaseMafsal1.Matrix.X), AffineVectorMake(0, 0, 0), Q1N); // yeni Y

TM := (OmuzBaseMafsal1.Matrix)^;

TM.Y := VectorMake(Q1NN, 0);

TM.Z := VectorMake(Q1N, 0);

OmuzBaseMafsal1.SetMatrix(TM);


// 2 ıncı omuz -kol doğrultma
```

```
BuffV := VectorAdd(LinkN[2], AffineVectorMake(OmuzBaseMafsal2.AbsolutePosition));

BuffV := OmuzBaseMafsal2.AbsoluteToLocal(BuffV);

// BuffV burada hala BilekBaseMafsal10 uzayında;

// Q1N BilekBaseMafsal10 matrisinin yeni Z ti olacak fakat bi önceki eyleyici matrisine taşınması lazım

Q1N := VectorNormalize(AffineVectorMake(0, BuffV.Y, BuffV.Z));

Q1N := VectorSubtract(VectorTransform(Q1N, (OmuzBaseMafsal2.Matrix)^ ), OmuzBaseMafsal2.Position.AsAffineVector);

// Q2N BilekKol10 Matrisinin yeni Z ti olacak önceki matrise taşınmasına gerek yok

Q2N := VectorNormalize(AffineVectorMake(BuffV.X, 0, BuffV.Z));

Q2NN := CalcPlaneNormal(AffineVectorMake(0, 1, 0), AffineVectorMake(0, 0, 0), Q2N); // yeni x

TM.Y := VectorMake(AffineVectorMake(0, 1, 0), 0);

TM.X := VectorMake(Q2NN, 0);

TM.Z := VectorMake(Q2N, 0);

TM.W := VectorMake(0, 0, 0, 1);

OmuzKol2.SetMatrix(TM);


Q1NN := CalcPlaneNormal(AffineVectorMake(OmuzBaseMafsal2.Matrix.X), AffineVectorMake(0, 0, 0), Q1N); // yeni Y

TM := (OmuzBaseMafsal2.Matrix)^;

TM.Y := VectorMake(Q1NN, 0);

TM.Z := VectorMake(Q1N, 0);

OmuzBaseMafsal2.SetMatrix(TM);
```

```
// 3 ıncı omuz -kol doğrultma

BuffV := VectorAdd(LinkN[3], AffineVectorMake(OmuzBaseMafsal3.AbsolutePosition));

BuffV := OmuzBaseMafsal3.AbsoluteToLocal(BuffV);

// BuffV burada hala BilekBaseMafsal10 uzayında;

// Q1N BilekBaseMafsal10 matrisinin yeni Z ti olacak fakat bi önceki eyleyici matrisine taşınması lazım

Q1N := VectorNormalize(AffineVectorMake(0, BuffV.Y, BuffV.Z));

Q1N := VectorSubtract(VectorTransform(Q1N, (OmuzBaseMafsal3.Matrix)^ ), OmuzBaseMafsal3.Position.AsAffineVector);

// Q2N BilekKol10 Matrisinin yeni Z ti olacak önceki matrise taşınmasına gerek yok

Q2N := VectorNormalize(AffineVectorMake(BuffV.X, 0, BuffV.Z));

Q2NN := CalcPlaneNormal(AffineVectorMake(0, 1, 0), AffineVectorMake(0, 0, 0), Q2N); // yeni x

TM.Y := VectorMake(AffineVectorMake(0, 1, 0), 0);

TM.X := VectorMake(Q2NN, 0);

TM.Z := VectorMake(Q2N, 0);

TM.W := VectorMake(0, 0, 0, 1);

OmuzKol3.SetMatrix(TM);


Q1NN := CalcPlaneNormal(AffineVectorMake(OmuzBaseMafsal3.Matrix.X), AffineVectorMake(0, 0, 0), Q1N); // yeni Y

TM := (OmuzBaseMafsal3.Matrix)^;

TM.Y := VectorMake(Q1NN, 0);

TM.Z := VectorMake(Q1N, 0);

OmuzBaseMafsal3.SetMatrix(TM);
```

```
// 4 ıncı omuz -kol doğrultma

BuffV := VectorAdd(LinkN[4],
AffineVectorMake(OmuzBaseMafsal4.AbsolutePosition));

BuffV := OmuzBaseMafsal4.AbsoluteToLocal(BuffV);

// BuffV burada hala BilekBaseMafsal10 uzayında;

// Q1N BilekBaseMafsal10 matrisinin yeni Z ti olacak fakat bi önceki eyleyici
matrisine taşınması lazım

Q1N := VectorNormalize(AffineVectorMake(0, BuffV.Y, BuffV.Z));

Q1N := VectorSubtract(VectorTransform(Q1N, (OmuzBaseMafsal4.Matrix)^ ),
OmuzBaseMafsal4.Position.AsAffineVector);

// Q2N BilekKol10 Matrisinin yeni Z ti olacak önceki matrise taşınmasına gerek yok

Q2N := VectorNormalize(AffineVectorMake(BuffV.X, 0, BuffV.Z));

Q2NN := CalcPlaneNormal(AffineVectorMake(0, 1, 0), AffineVectorMake(0, 0, 0),
Q2N); // yeni x

TM.Y := VectorMake(AffineVectorMake(0, 1, 0), 0);

TM.X := VectorMake(Q2NN, 0);

TM.Z := VectorMake(Q2N, 0);

TM.W := VectorMake(0, 0, 0, 1);

OmuzKol4.SetMatrix(TM);


Q1NN := CalcPlaneNormal(AffineVectorMake(OmuzBaseMafsal4.Matrix.X),
AffineVectorMake(0, 0, 0), Q1N); // yeni Y

TM := (OmuzBaseMafsal4.Matrix)^;

TM.Y := VectorMake(Q1NN, 0);

TM.Z := VectorMake(Q1N, 0);

OmuzBaseMafsal4.SetMatrix(TM);
```

```
// 5 ıncı omuz -kol doğrultma

BuffV := VectorAdd(LinkN[5], AffineVectorMake(OmuzBaseMafsal5.AbsolutePosition));

BuffV := OmuzBaseMafsal5.AbsoluteToLocal(BuffV);

// BuffV burada hala BilekBaseMafsal10 uzayında;

// Q1N BilekBaseMafsal10 matrisinin yeni Z ti olacak fakat bi önceki eyleyici matrisine taşınması lazım

Q1N := VectorNormalize(AffineVectorMake(0, BuffV.Y, BuffV.Z));

Q1N := VectorSubtract(VectorTransform(Q1N, (OmuzBaseMafsal5.Matrix)^ ), OmuzBaseMafsal5.Position.AsAffineVector);

// Q2N BilekKol10 Matrisinin yeni Z ti olacak önceki matrise taşınmasına gerek yok

Q2N := VectorNormalize(AffineVectorMake(BuffV.X, 0, BuffV.Z));

Q2NN := CalcPlaneNormal(AffineVectorMake(0, 1, 0), AffineVectorMake(0, 0, 0), Q2N); // yeni x

TM.Y := VectorMake(AffineVectorMake(0, 1, 0), 0);

TM.X := VectorMake(Q2NN, 0);

TM.Z := VectorMake(Q2N, 0);

TM.W := VectorMake(0, 0, 0, 1);

OmuzKol5.SetMatrix(TM);

Q1NN := CalcPlaneNormal(AffineVectorMake(OmuzBaseMafsal5.Matrix.X), AffineVectorMake(0, 0, 0), Q1N); // yeni Y

TM := (OmuzBaseMafsal5.Matrix)^;

TM.Y := VectorMake(Q1NN, 0);

TM.Z := VectorMake(Q1N, 0);

OmuzBaseMafsal5.SetMatrix(TM);

end;
```

Ek 4. Paralel kinematikli 6 aktüatörlü (Hexapod) robotun ileri kinematik fonksiyon kodları

```
#!/usr/local/bin/python
# -*- coding: utf-8 -*-
from numpy.linalg import inv
import numpy as np
import math
import socket

hostname = socket.gethostname()
hostip = socket.gethostbyname(hostname)
baglantiPortu = 1234
baglantiTamponu = 1024
calistir = (hostip,baglantiPortu)
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(calistir)

# firsth of all we need kinematicForward
# for kinematicForward initial values are joints , poses

# The kinematicsForward function solves the forward kinematics using
#  an iterative algorithm.  Due to the iterative nature of this algorithm
#  the kinematicsForward function requires an initial value to begin the
#  iterative routine and then converges to the "nearest" solution. The
#  forward kinematics problem is given the strut lengths and returns the
#  pose of the platform.  For this problem there arein multiple
#  solutions.  The kinematicsForward function will return only one of
#  these solutions which will be the solution nearest to the initial
#  value given.  It is possible that there are no solutions "near" the
#  given initial value and the iteration will not converge and no
#  solution will be returned.  Assuming there is a solution "near" the
#  initial value, the function will always return one correct solution
#  out of the multiple possible solutions.

# Base omuz noktası
# 264.4968,-221.9894,-12.7500
# 324.4968,-118.0663,-12.7500
# 60.0,340.0557,-12.7500
# -60.0,340.0557,-12.7500
# -324.4968,-118.0663,-12.7500
# -264.4968,-221.9894,-12.7500

# ac bilek noktası
# 16.5000,-50.000,0
```

```
# 51.5513,10.7106,0
# 35.0513,39.2894,0
# -35.0513,39.2894,0
# -51.5513,10.7106,0
# -16.5000,-50.000,0


a = np.array([[16.5000, -50.000, 0],
              [51.5513, 10.7106, 0],
              [35.0513, 39.2894, 0],
              [-35.0513, 39.2894, 0],
              [-51.5513, 10.7106, 0],
              [-16.5000, -50.000, 0]])

aw = np.array([[0, 0, 0],   [0, 0, 0],   [0, 0, 0],   [0, 0, 0],   [0, 0,
0],    [0, 0, 0]])

b = np.array([[264.4968, -221.9894, -12.7500],
              [324.4968, -118.0663, -12.7500],
              [60.0, 340.0557, -12.7500],
              [-60.0, 340.0557, -12.7500],
              [-324.4968, -118.0663, -12.7500],
              [-264.4968, -221.9894, -12.7500]])

InvJakobiyen = np.array([[0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                         [0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                         [0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                         [0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                         [0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                         [0.0, 0.0, 0.0, 0.0, 0.0, 0.0], ])
Jakobiyen = np.array([[0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0], ])

rot_a = np.array([[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0],
[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]])
rot_a_cross = np.array(
    [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0],
[0.0, 0.0, 0.0], [0.0, 0.0, 0.0]])
InvKinKolVek = np.array(
    [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0],
[0.0, 0.0, 0.0], [0.0, 0.0, 0.0]])
InvKinKolVekBirim = np.array(
```

```
    [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0. 0, 0.0, 0.0], [0.0, 0.0, 0.0],
[0.0, 0.0, 0.0], [0.0, 0.0, 0.0]])
InvKinKolUzunluk = np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
KolUzunluguFark = np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])

# hareketli platform park pozisyonu
# 0,0,-207.25
xyz = np.array([0.0, 0.0, -207.75])
rpy = np.array([0.0, 0.0, 0.0])

rot = np.array([[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]])  #
Eyleyici başlangıç

# verilen kol uzunlukları  xyz 4,7,-207.25 rpy 0,0,0     noktasına göre
verilenKolUzunklugu = np.array([359.724741799506,
                                358.60465211455,
                                352.941546041853,
                                353.506596998744,
                                364.642872476462,
                                365.198855134846])

VKUınt = np.array([0, 0, 0, 0, 0, 0], dtype=int)

iv = 0
for item in verilenKolUzunklugu:
    VKUınt[iv] = round(item / 0.0025)
    iv = iv + 1

iv = 0
for item in VKUınt:
    verilenKolUzunklugu[iv] = item * 0.0025
    iv = iv + 1

delta = np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])


def RpyMatrisCevir(r, p, y):
    sa = math.sin(y)
    sb = math.sin(p)
    sg = math.sin(r)

    ca = math.cos(y)
    cb = math.cos(p)
    cg = math.cos(r)
    m = np.array([[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]])
    m[0][0] = ca * cb
```

```
    m[0][1] = ca * sb * sg - sa * cg
    m[0][2] = ca * sb * cg + sa * sg

    m[1][0] = sa * cb
    m[1][1] = sa * sb * sg + ca * cg
    m[1][2] = sa * sb * cg - ca * sg

    m[2][0] = -sb
    m[2][1] = cb * sg
    m[2][2] = cb * cg

    return m

iterasyon = 0
iterasyonBayrak = 1
def ForwardKinematik(linkInt0, linkInt1, linkInt2, linkInt3, linkInt4,
linkInt5):
    verilenKolUzunklugu[0] = linkInt0
    verilenKolUzunklugu[1] = linkInt1
    verilenKolUzunklugu[2] = linkInt2
    verilenKolUzunklugu[3] = linkInt3
    verilenKolUzunklugu[4] = linkInt4
    verilenKolUzunklugu[5] = linkInt5
    while (iterasyonBayrak):
        rot = RpyMatrisCevir(rpy[0], rpy[1], rpy[2])
        for i in range(0, 6):
            rot_a[i] = np.dot(rot, a[i])  # platform (ucu) dan kola olan
vector rotasyon ile çarpılıyor
            aw[i] = np.add(xyz, rot_a[i])  # platform (ucu) dan kola olan
vector efektor konumu ile çarpılıp bileğin wordedeki ucu bulunuyor
            InvKinKolVek[i] = np.subtract(aw[i], b[i])  # platform (ucu)
dan kola olan vector ile base (tavan word) dan kola olan vector
çıkartılıyor kolun vector bulunuyor
            InvKinKolUzunluk[i] = np.linalg.norm(InvKinKolVek[i])  # kol
uzunluğu alınıyor
            InvKinKolVekBirim[i] = np.dot(InvKinKolVek[i], (1 /
InvKinKolUzunluk[i]))  # birim vektör bulunuyor SIFIR BÖlüm HATASı
aranmalı
            KolUzunluguFark[i] = InvKinKolUzunluk[i] -
verilenKolUzunklugu[i]  # verilemn le eski kol uzunluk farkı bulunuyor
            rot_a_cross[i] = np.cross(rot_a[i], InvKinKolVekBirim[i])
            InvJakobiyen[i, 0] = InvKinKolVekBirim[i][0]
            InvJakobiyen[i, 1] = InvKinKolVekBirim[i][1]
            InvJakobiyen[i, 2] = InvKinKolVekBirim[i][2]
            InvJakobiyen[i, 3] = rot_a_cross[i][0]
            InvJakobiyen[i, 4] = rot_a_cross[i][1]
```

```
            InvJakobiyen[i, 5] = rot_a_cross[i][2]
        Jakobiyen = inv(np.matrix(InvJakobiyen))
        delta = np.dot(np.matrix(Jakobiyen),
np.ndarray.transpose(np.matrix(KolUzunluguFark)))
        print('xyz :\n', xyz, '\nrpy :\n', rpy)
        xyz[0] = xyz[0] - delta[0]
        xyz[1] = xyz[1] - delta[1]
        xyz[2] = xyz[2] - delta[2]
        rpy[0] = rpy[0] - delta[3]
        rpy[1] = rpy[1] - delta[4]
        rpy[2] = rpy[2] - delta[5]
        print('delta\n', delta)
        print('-> xyz :\n', xyz, '\nrpy :\n', rpy)

        iterasyon = iterasyon + 1

        iterasyonBayrak = 0;
        conv_err = 0
        for j in range(0, 6):
            if math.fabs(KolUzunluguFark[j]) > 0.0001:
                iterasyonBayrak = 1
            conv_err = conv_err + math.fabs(KolUzunluguFark[j])

while True:
    data = s.recv(baglantiTamponu).decode('utf-8')
    if data == "q":
        break
    elif len(data) == 0:
        pass
    else:
        print(data.split(" "))
        print( data )

s.close()
```

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, adı : Eyüp FINDIKLI
Doğum tarihi ve yeri : 1987-KÜTAHYA
e-mail : eyupfindikli@gmail.com

Eğitim

Derece	Eğitim Birimi	Mezuniyet Tarihi
Lisans	Dumlupınar Üniversitesi/Makine Müh.	2008

İş Deneyimi

Yıl	Yer	Görev
• Mayıs 2017-Halen	Adeko Teknoloji	CAD CAM yazılım geliştirme mühendisi
• Ekim 2016- Haziran 2017	Arel Üniversitesi	Öğretim Görevlisi
• Ağustos 2014- Şubat 2016	WhiteCAD Teknolojileri	CAD CAM yazılım geliştirme mühendisi
• Ağustos 2013- Ağustos 2014	DPÜ Yapı İş. T.D.B.	Bakım / Denetim Mühendisi
• Ekim 2012- Mart 2013,	Vig Metal	Mekanik Bakım Mühendisi
• Ağustos 2008- Ocak 2011	Alpata Teknoloji	CAD CAM Mühendisi