İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

BİR MATKAP İÇİN KULLANICI ARAYÜZ YAZILIMI

YÜKSEK LİSANS TEZİ

Müh. Cüneyt SABIRCAN

Tezin Enstitüye Verildiği Tarih: 3 Şubat 1992

Tezin Savunulduğu Tarih : 25 Şubat 1992

Tez Danışmanı : Prof.Dr. Eşref ADALI

Diğer Jüri Üyeleri : Prof.Dr. Nadir YÜCEL

Doç.Dr. Bülent ÖRENCİK

ŞUBAT 1992

ÖNSÖZ

Birçok konudaki bilgimin gelişmesine neden olan bu tezi bana veren ve yol gösteren danışman hocam Sayın Prof.Dr. Eşref ADALI'ya teşekkürlerimi sunarım. Ayrıca, tezin çeşitli aşamalarındaki yapıcı eleştirilerinden ve yardımlarından dolayı Yük.Müh. D.Turgay Altılar'a, Müh. Bülent Çınarkaya'ya, Yük.Müh. Aylin Tülay USTA'ya ve manevi desteklerinden dolayı tüm İ.T.Ü. Kontrol ve Bilgisayar Bölümü araştırma görevlisi arkadaşlarıma teşekkürü bir borç bilirim.

Xy tablasının mekanik tasarımında, motorların seçimi ve sürücü devrelerinin tasarımında çalışan ve bu konuda beni aydınlatan TÜBİTAK MAM Bilgisayar Destekli Tasarım ve Üretim bölümü çalışanlarına da ayrıca teşekkürlerimi sunarım.

Cüneyt SABIRCAN
İstanbul, 1992

İÇİNDEKİLER

ŞEKİL LİSTESİ

BÖLÜM 3

BÖLÜM 4

TABLO LİSTESİ

BÖLÜM 3

BÖLÜM 4

ÖZET

Günümüzde, sayısal elektronikteki gelişmelere paralel olarak kontrol sistemlerinde de sayısal yöntemler kullanılmaya başlanmış ve özellikle motor kontrolünde sıkça uygulama alanı bulmuşlardır. Motor çeşitlerinden , doğru ve alternatif akım motorları yüksek güçlerdeki çalışmalarda kullanılmalarına rağmen, kontrollerinde geri beslemeye ihtiyaç duymaları ve yüksek duyarlılıklarda çalışamamaları gibi bazı dezavantajlara sahiptirler. Sayısal işaretlerle doğrudan kontrol edilebilen Adım Motorları, diğer iki tip motorun kullanılamadığı bu tip uygulamalarda sıklıkla kullanılırlar. Maliyetlerinin düşük olması, sürücü devrelerinin basitliği gibi avantajlara sahip olan Adım Motorları, hız kontrollerinin dar bir aralıkta yapılabilmesi gibi bir dezavantaja rağmen özellikle xy tablalarında, yazıcı ve çizicilerde sıklıkla kullanılmaktadır.

X ve y eksenlerine yerleştirilmiş hibrit adım motorları sayesinde hareket eden bir xy tablası ve sabit matkaptan oluşan sistemimizde, kullanıcı tarafından matkabın deleceği noktaların koordinatları veri olarak girilmekte ve oluşturulan yazılım da bu bilgilerden adım motorlarına attırılacak adım sayılarını hesaplamaktadır. Koordinat bilgileri çeşitli düzenlerde olabileceği gibi işlenecek olan parçada harekete engel olacak bazı engeller bulunabilmekte ve xy tablasına buna bağlı olarak belli yörüngeler verilmektedir. Bu amaçla engel etrafından dolaşma algoritması geliştirilmiş ve kullanıcının verdiği parçaya ilişkin bilgilerden, parçanın içindeki işlenebilir alanların yerleri belli bir algoritma ile tesbit edilmiştir. Tezde, birçok veri yapısı ve algoritma kullanılmıştır. Grafikle ilgili çeşitli teknikler kullanılmış ve kullanıcıya programı mümkün olduğu kadar kolay kullanma imkanı sağlanmıştır. Veri girişinin her aşamasında verilerin kontrolü prensibi ön planda tutulmuş ve kullanıcının yapacağı hataların en aza indirilmesi amaçlanmıştır.

SUMMARY

A USER INTERFACE FOR A DRILLING SYSTEM

The mechanical part of the system mainly consists of two stages which can make linear motion. Step motors are used to make the stages move through screws which convert the rotation of the motor shaft to linear motion of the stages.

Accurate positioning is one of the most important problems in position control systems. As an actuator, DC or AC motors are widely used when the settling points are far from the starting point, but if the positioning requires small movements, conventional motors capabilities fail.

Accurate positioning with very small movements has been achieved after the development of the stepping motors. Stepping motors are electromagnetic incremental motion actuators which convert digital pulses to analog output motion. When properly controlled, the output steps of a step motor are always equal in number to input pulses. Each pulse advances the rotor shaft one step increment and latches it magnetically at the precise point to which it is stepped.

Although there is a wide range of stepping motor designs, most motors can be identified as a variation of the three basic types; variable reluctance (VR), permanent magnet, or hybrid step motors.

VR step motors have magnetic field which is produced solely by the winding currents. For the hybrid motors, the main source of the magnetic flux is a permanent magnet and DC currents flowing in one or more windings direct the flux along alternative paths. In both types of step motors accurate positioning of the rotor is generally achieved by magnetic allignment of the iron teeth on the stationary and rotating parts of the motor. Hybrid Stepper motors are used in our system to provide movement on the axises.

Hybrid motors have a small step length (typically 1.8) which can be great advantage when high resolution angular positioning is required. The torque producing capability of the hybrid motors are generally greater then VR motors and also sufficient for general applications. When the windings of the hybrid motor are unexcited the magnetic flux produces a small detent torque which retains the rotor at the step position. This feature can be useful in some applications where the rotor position must be preserved during a power failure.

As I mentioned above, stepping motor systems are popular for many reasons. The summary of these reasons below offers an introduction to various control schemes.

- Data are increasingly handled in digital form. Digital integrated circuits are inexpensive and if output motion is desired a digital actuator such as a stepping motor provides the ideal solution.

- Stepping motors are inherently very reliable because they are simple devices with only two moving bearings.

- More controls and processes are being automated because of the availability of inexpensive microprocessors.

- Stepping motors are ideal motion control devices because they are digitally controlled. They are easly used in incremental and continous motion control applications.

- More powerful stepping motors and more reliable, lower cost, solid state power devices have extended the range of application.

- Stepping motors needs simple and lower cost driver circuits comparing with other motors.

Stepping motors are often used as output devices for microprocessor based control systems. The essential feature of the systems is that the microprocessor program produces a result and the stepping motor then moves the load to the position corresponding to this result. There are various ways in which the microprocessor can be involved in control of the stepping motor. These are software intensive and hardware intensive.

In our system software intensive approach was used. In software intensive approach, microprocessor produces the phase control signals, and the program is responsible for timing and sequencing the signal to move the motor to the required position.

In hardware intensive approach, microprocessor program merely feed the target position information and a start command to hardware controller, which generates the phase control signal for the motor drive circuits and a final signal for the microprocessor when the target is reached.

The hardware parts of the developed system consists of control, drive and interfacing circuits. Operating with open-loop control at constant stepping rate is chosen.

When the hardware was implemented the software which would control the motors were considered. The characteristics of the motors were examined and parabolic curve was selected for acceleration. The motor control program was written in assembly language to properly control stepper motor.

The main part of the thesis consists of a Pascal program which has about 9000 source lines. It has different data types and algorithms. The data structures saves the data entered by user and the algorithms find the path between two holes specified by the coordinate data given by the users. The program supports mouse and mouse makes the use of the program easy.

As I have just mentioned, program have different data types. These data types are dynamic and they are not in memory unless they are used. The examples of this data type used in the thesis are given below:

```
TYPE
  LineP    = ^LineType;
  LineType = Record
               Xs, Ys, Xe, Ye : Integer;
               Dashed         : Boolean;
               UpView         : Boolean;
               Next           : LineP;
             end;
```

The example structure is one of the dynamic list structure used in the program. This structure is used to save the data about a part in memory. The technical picture of a part is represented as a dynamic list structure in memory. Xs, Ys, Xe, and Ye record parts show the starting and end coordinates of a line drawn for describing a part. Dashed boolean record part determines the type of the line. Upview describes from where the user looks at the part and the Next part of the record shows where the other records and data are.

The main structure which holds the coordinate data given by the user, has a dynamic matrix structure. This structure was identified in the program like it is written below:

```
TYPE
  MainNode     = Record
                   Data        : Real;
                   Point       : ExtNodePtr;
                   Next        : MainNodePtr;
                 end;
  ExtNode      = Record
                   X, Y, Z     : Real;
                   S, R, C, T  : Boolean;
                   Un, No      : Byte;
                   Link        : ExtNodePtr;
                 end;
  MainNodePtr  = ^MainNode;
  ExtNodePtr   = ^ExtNode;
```

This structure expands on two direction. Vertically, and horizontally. The main structure consists of the ExtNodes and when the radius datum of a coordinate isn't in the list then a node which has the radius of this coordinate is added to the mainnodelist. The procedures of these structures are given at the end of the thesis.

The algorithms which deal with the motor motion are also implemented. Some of these algorithms are Kruskal algorithm which find the minimum path between coordinates entered by the user, area line intersection algorithm, DDA line algorithm, dynamic list bubble sort algorithm, quick sort algorithm, orbit according to a part algorithm. These algorithms are given in appendixes.

The program consists of some menus and some sub menus. The main menu have six selection. These are coordinate, part, operate, options, initial conditions, and quit. Let's focus our attention, when we will use these selections.

- Coordinate option is used to enter the coordinate of the data where we will make the xy stage move. The data could be in different format. The data could be absolute or relative. They could be in cartesian or cylindrical coordinate system. In this option, we could see the data we had entered, on the screen or we could print them on a paper by a printer.

- Part option is used to describe the part we examine. Description of the part is made by means of its technical draw. The sections of it that we can see, is drawn as a solid line and the others are as a dashed line. When the draw is completed, the important lengths are asked.

-Operate option deals with calculating the step numbers for each axis and send them to the motor control software written in assembly language. The orbit of the motion could be seen by a simulation program.

- Options and initial settings submenus provide to change some global variables. Some of them are the radius of the drill, the height of the drill, the sensitivity of the mouse, the coordinate system that we work on, etc.

- Quit provides us to quit from the program if we are sure.

The performance of the system could be increased by using faster computers or some external devices such as microcontrollers. When such devices are added, these devices are dedicated to control the motors and computer could calculate the results faster and operate more systems.

BÖLÜM 1. GİRİŞ

Xy tablaları, mekanik olarak, genelde vida somun yöntemine göre tasarlanan ve x ile y yönündeki hareketlerin birer motor sayesinde verildiği sistemlerdir. Çiziciler bu tip sistemlere en güzel örnek olarak verilebilir.

Bu tür sistemlerde hareketi vermek üzere adım motorları sıklıkla tercih edilir. Bu tip motorlar, girişlerine uygulanan darbe dizilerine karşılık analog dönme hareketi yapabilen elektromagnetik elemanlardır. Bu özellikleri nedeni ile sayısal makina olarak da tanınırlar. Sargılarından doğru akım akıtılarak uyarılırlar. Ayrıca şu avantajlara da sahiptirler:

- Geri beslemeye ihtiyaç duymaksızın, açık çevrim çalıştırılabilirler. Stabilite problemleri yoktur.
- Kümülatif (her yeni adımla artan) konum hatası yoktur.
- Sayısal giriş işaretlerine cevap verirler, ve bu nedenle mikroişlemcili kontrol için uygundurlar.
- Mekanik olarak basit yapıya sahiptirlar ve bakım gerektirmezler.
- Yağlanma ve kirlenme gibi problemleri yoktur.
- Hasara neden olmadan defalarca çalıştırılıp, durdurulabilirler.

Bu avantajlarının yanında bu motorlar bazı dezavantajları da bünyesinde bulundurur. Bu dezavantajlar:

- Adım açıları sabittir, dönme hareketi sürekli değil artımsaldır. (Adım cevabı esnek değildir.)
- Klasik sürücülerle kullanıldıklarında verimleri düşüktür.
- Adım cevapları nisbeten büyük aşımlı ve salınımlıdır.
- Yüksek eylemsizlikli yüklerde yetenekleri sınırlıdır.

- Sürtünme kaynaklı yükler, hata kümülatif olmasa dahi açık çevrim çalışmada konum hatasını arttırırlar.
- Elde edilebilecek çıkış gücü ve momenti sınırlıdır.

Tüm bu dezavantajlarına rağmen bu motorlar xy tablalarında büyük moment gereksinimi olmadığından kullanım bulurlar. Bu motorların sürücü devrelerinin de uygun seçilmesi ile sistemimizin kontrolü için yeterli koşulları elde edebiliriz.

Sistem tasarlanıp, gerekli donanım kurulduktan sonra seçilecek kontrol yazılımı dili de önemli bir problem olarak karşımıza çıkar. Teorik olarak sonuçları en hızlı şekilde elde etmek için makinanın simgesel dilinde program yazmak en akıllı seçim olarak gözükse de, bu dille kullanıcıya işlemleri kolaylaştıracak bir arayüz oluşturmak mümkün olamamaktadır. Bu nedenle, yüksek düzeyde bir dilde ve grafik ortamda program yazmak daha uygulanabilir bir yaklaşımdır. Ancak, yüksek düzeyde dillerin yavaşlığı nedeni ile de en azından motora uygulanacak işaretlerin simgesel dilde yazılmış bir program tarafından üretilmesi kaçınılmazdır. Seçilecek yüksek düzey dildeki veri yapılarının çeşitli olması da, kullanıcının gireceği verilerin bellekte uygun şekilde tutulması açısından önem taşır. Programın büyük alanlar kaplamasından dolayı, dilin modüler olması, programın parçalara ayrılabilmesi ve programdaki hataların kolay bulunabilmesi amacı ile dilin Debug olanaklarının olması dikkate alınacak diğer hususlardır. Tüm bu özellikleri taşıyan Turbo Pascal, programın yazım dili olarak seçilmiştir.

BOLUM 2. MEKANİK DUZENLER

Bu alt başlık altında xy tablalarında sıklıkla kullanılan mekanik düzenler ve mekanik elemanlar kısaca tanıtılacaktır. Bu elemanların kullanılış nedenleri, özellikleri avantaj ve dezavantajları betimlenecektir.

Xy tablalarında, x ve y yönündeki hareket birbirinden bağımsız olarak incelenebilir. Bu nedenle, sadece bir yönde hareketi sağlayacak mekanik düzenler incelenip, bunun iki eksene genişletilmiş halinin xy tablalarında kullanılabileceği düşünülecektir. Bu sistemlerde hareket motorlar tarafından verildiğinden, mekanik olarak dönme hareketini öteleme hareketine dönüştürmemiz gerekmektedir. Bu amaçla bir çok yöntem kullanılmaktadır. Bu yöntemler aşağıda kullanım önceliklerine göre verilecektir.[1]

2.1. Vida Somun Yöntemi

Bu yöntem, xy tablalarında sıklıkla kullanılır. Motor miline bağlanan bir vidanın döndürüldüğünde üzerinde yer alan somunun ilerlemesi ilkesine dayanır. Böylece, dönme hareketi öteleme hareketine dönüştürülür. Bu yöntemin avantajı, somunun elektriksel eşdeğerinin bir diyot olmasıdır. Vida tarafından somuna bir kuvvet etkimekte fakat buna karşılık somunda oluşan bir kuvvet motora yük olarak etki etmemektedir.

Bu yöntemde, somunla vida arasındaki boşluklardan dolayı konumlandırma hataları oluşabilmektedir. Bu boşlukları azaltmak için, vida ile somun arasına bilye konulur ve böylece hata azaltılır. Ancak, maliyeti çok arttıran bu uygulama, çok büyük duyarlılıklara çıkılmadıkça kullanılmaz.

2.2. Kayış Kasnak Yöntemi

Bu yöntemde bir kayış, sabit fakat üzerinde kayma yapabileceği bir nokta ile motor mili arasına bağlanır. Motor mili döndürüldüğünde kayışın üzerinde yer alan herhangi bir nokta öteleme hareketi yapmış olur. Yazıcılarda kullanılan bu yöntemde motorun çok az miktarda dönmesi gerektiğinden sağlıklı bir yöntem değildir. Ayrıca kayışın pratikte karşılaşılan esneme ve uzama gibi sorunlarından dolayı sık kullanılmaz.

Kayışın sürtünme ya da kayma hareketi sırasındaki istenmeyen hareketlerini engellemek için kayış, bisikletlerde hareketi ileten kayışlar gibi girintili çıkıntılı üretilerek sabit noktadaki dişlere geçirilir. Ancak, bu şekilde dahi çoğu uygulamalar için gerekli duyarlılık sağlanamaz.

2.3. Krameyer Dişlisi

Dönme hareketini öteleme hareketine çeviren mekanik bir sistemdir. Birbirinin içine giren ve biri yuvarlak diğeri düz iki dişliden oluşur. Bu sayede, yuvarlak dişli bir adım döndüğünde, düz dişli dönüş yönüne bağlı olarak bir adım öteleme hareketi yapar. Pahalı bir düzendir. Maliyetinden dolayı uygulamada nadiren kullanılır. Her iki dişli de birbirinden etkilendiğinden büyük yüklerde motora milinde yüksek yük momenti oluşmasına neden olur. Kısacası, yükü motordan yalıtamaz.

2.4. Yardımcı Düzenler

Uygulamalarda, yukarda belirtilen sistemlerle birlikte kullanılan bazı mekanik elemanlar da bulunmaktadır. Bu elemanlar yataklar, taşıyıcılar, dişliler olarak sınıflandırılabilir. Sistemlerdeki hareketi kolaylaştırmak için kullanılırlar.

Büyük moment gerektiren uygulamalarda hareketi kolaylaştırmak amacı ile taşıyıcılar ve yataklar kullanılır. Taşıyıcılar hareketli parçanın içinden geçirilen rijit bir çubuk olarak tanımlanabilir. Bu çubuk, yükün motor miline etkisini azaltmak amacı ile kullanılır. Hareketli parçanın bu çubuk üzerinde kayabilmesi tek şart olarak karşımıza çıkar. Bu da parçanın yataklanması ile sağlanır. Yataklar içinde yer alan hava aralığı ya da yağ hareketin sürtünmesiz şekilde yapılabilmesini sağlar.

Motorun momentinin yetersiz kaldığı uygulamalarda ise dişliler momenti arttırmak için kullanılır. Dişliler hareket yönünü değiştirmek ya da hassas hareketler elde etmek amacı ile de kullanılabilir. Dişli çeşitlerinden biri olan harmonik dişli, hareketleri hassas hale getirmek amacı ile uygulamalarda sıklıkla kullanılmaktadır. Pahalı olan bu mekanik eleman sayesinde çok hassas hareketler elde edilebilmektedir.

Sistemimizde bulunan xy tablasının mekanik yapısında, her iki eksende de vida somun yöntemi kullanılmış ve bu sistem bir çerçeve yapı ile yataklanmıştır. Mekanik düzende hareket 25 mikron duyarlılıkla yapılabilmektedir. Bu uzunluk adım motorunun bir adımına karşılık gelmektedir.

BÖLÜM 3. EKSENLERDE HAREKET

Bu bölümde, XY tablalarında kullanılmakta olan motorların genel özellikleri, avantaj ve dezavantajları irdelenecektir. Bu arada, motorlara ait genel sürücü devreler de bölüm kapsamında anlatılacaktır. Motorlar ve sürücülerin incelenmesinde temel kavramlar üzerinde durulacak ve ayrıntılardan kaçınılacaktır.

3.1. Motor Çeşitleri ve Sürücü Devreleri

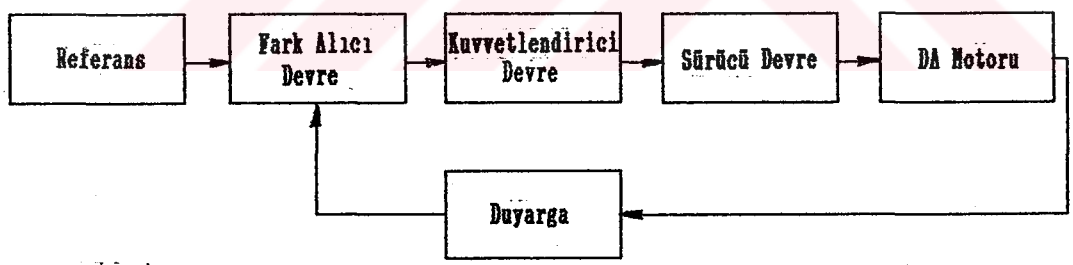
Eksenlerde hareketin sağlanması için Doğru Akım, Alternatif akım ya da Adım Motorları kullanılır. Motorların gerek fiziksel yapısından kaynaklanan farklar, gerekse dayandığı esaslar sürücü devrelerinde farklılıklara yol açar. Önemli olan, tablayı mümkün olduğu kadar az hata ile konumlandırmak ve bunu yaparken minimum maliyet elde etmektir. Ayrıca, motorun yüke bağlı davranışı ve hız karakteristiği de uygulamalarda motor seçimini etkileyen önemli karakteristikler olarak karşımıza çıkar. Bu özelliklerini de belirterek, motorlara ait temel karakteristikleri inceleyelim.[2]

3.1.1. Doğru Akım Motorları

Tablalarda sıkça kullanılan Doğru Akım (DA) Motorları, hızının geniş bir aralıkta ve kolayca kontrol edilebilmesi nedeniyle birçok uygulamada maliyeti yükseltmesine rağmen kullanılırlar. Uyarma ya da faz akımı, ilişkin direncin değiştirilmesi ile kontrol edilir ve motorun hız kontrolü yapılmış olur. Elde edilen moment birçok uygulama için yeterlidir. Genelde, adım motorlarının yeterli momenti sağlayamadığı uygulamalarda tercih edilirler. Daha yüksek güçlere çıkıldığında ise yetersiz kalırlar.[3]

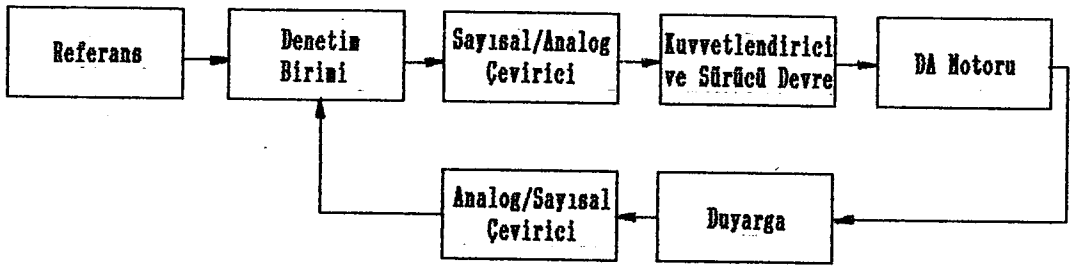
Kolay kontrol edilebilmesine karşılık, DA Motorları bazı olumsuz özellikleri de bünyesinde bulundururlar. Bu Motorlar genelde, kapalı çevrim kontrol sistemleri ile birlikte kullanılırlar. Konum ve hız kontrolü için duyargalar kullanılır. Duyargaların hassasiyeti, motorun konumlandırılma duyarlılığını belirler. Konum duyargaları analog olabileceği gibi sayısal da olabilir. Günümüzde, genelde sayısal duyargalar kullanılır. Mutlak veya bağıl optik konum kodlayıcılar en sık kullanılanlarıdır. Duyargalardan kaynaklanan duyarlılık belirsizliğinden dolayı bu motorlar yüksek duyarlılık istenen uygulamalar için uygun değildirler.

Tipik bir DA Motoru kapalı çevrim kontrol sisteminin blok diyagramı Şekil 3.1'de verilmiştir. Referans, istenen durum bilgisini içeren elektriksel bir büyüklüktür. Motorun durum bilgisini içeren işaret duyarga tarafından üretilip referans bilgisi ile farkı alınarak bir fark işareti elde edilir. Bu fark işareti de, bu işareti kuvvetlendiren ve DA Motorunu kontrol edebilecek büyüklüklere dönüştüren bir sürücü devreye uygulanır.



Şekil 3.1 Kapalı Çevrim DA Motor Kontrol Sistemi

Mikroişlemcili sistemlerdeki gelişmelere paralel olarak analog kontrol yöntemleri yerine sayısal kontrol yöntemleri kullanılmaya başlanmıştır. Bu tür bir DA Motor kontrol sistemi Şekil 3.2'deki gibidir. Bu sistemde belirtilen denetim birimi bir mikroişlemci ya da bir mikroişlemcili sistem olabilir. Referans bilgisi sayısal olarak denetim birimine aktarılır ve duyargadan gelen sayısal konum bilgisi ile birlikte değerlendirilir. Sonuçta, motora

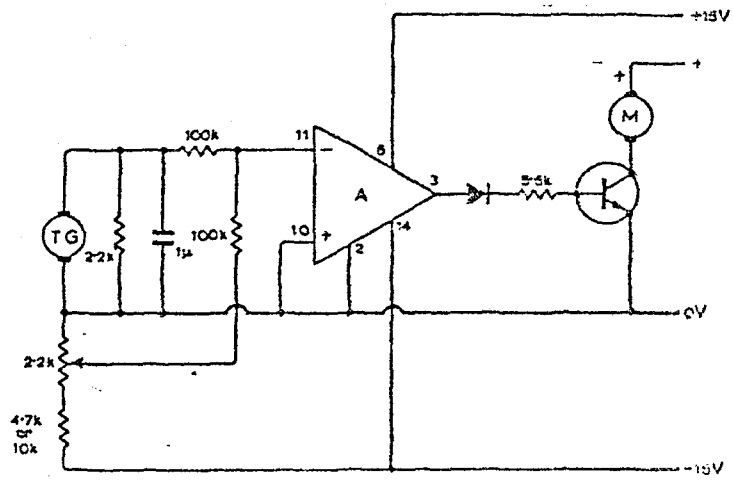


Şekil 3.2 DA Motorunun Sayısal Kontrolü

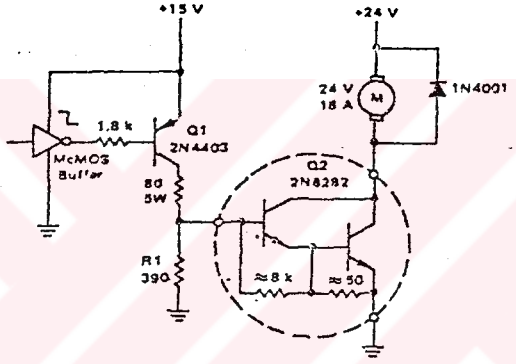
uygulanacak işarete ilişkin bir sayısal bilgi Sayısal Analog Çeviriciye uygulanır ve böylece elde edilen işaret motora uygulanmak üzere sürücüye gönderilir. Şayet, duyargamız analog bilgi veriyorsa, blok geri besleme yolu üzerinde bir Analog Sayısal Çeviriciye ihtiyaç olmaktadır. Fakat, sayısal tekniklerde bu tür ek elemanları ortadan kaldırmak için sayısal duyargalar kullanılır. Bu tür sistemlerde, motora kaç farklı değerde işaret uygulayabileceğimizi belirleyen Sayısal Analog Çeviricinin ayırıcılığı da DA Motoru kontrolümüzün duyarlılığını sınırlar.

DA Motorları, alan sargısı ve uyarma sargısı için iki tane kaynak gerektirirler. Bu bir dezavantaj gibi gözükse de son yıllarda sürekli mıknatıslı, az kayıplı DA Motorları üretilmesi ile bu dezavantaj ortadan kaldırılmıştır.

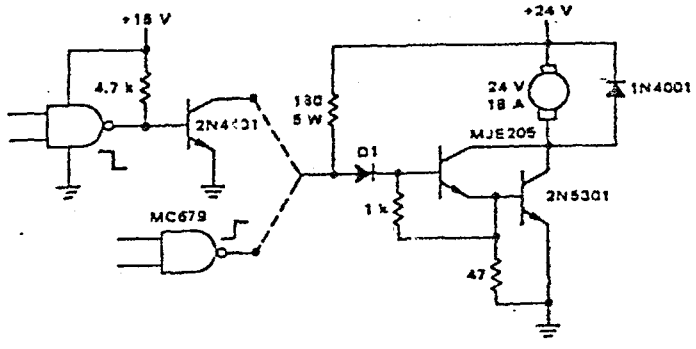
Kontrol edilebilme kolaylığı nedeniyle DA Motorlarının sürücü devreleri de oldukça basittir. Sırf bu amaçla yapılmış entegre devreler bulunmaktadır. Bu devreler, önlerindeki devreleri koruma özelliğinin yanı sıra, yüksek gerilim ve akım üretme özelliğine de sahiptirler. Tanımlanan bu entegrelerin dışında, işlemsel kuvvetlendiricili, CMOS kontrollü ve Darlingtonlu çiftli üç adet sürücü devre Şekil 3.3 'de verilmiştir. Bu sürücülerden ilkinde bir motor takogeneratör olarak kullanılarak DA Motoru için iyi bir hız kontrolü sağlanmıştır. Tranzistör, motorun akım ihtiyacını karşılayacak şekilde seçilmelidir. Diğer iki örnek, DA Motorunun lojik elemanlar kullanılarak kontrol edilebildiğini göstermesi açısından ilgi çekicidir.



(a)



(b)



(c)

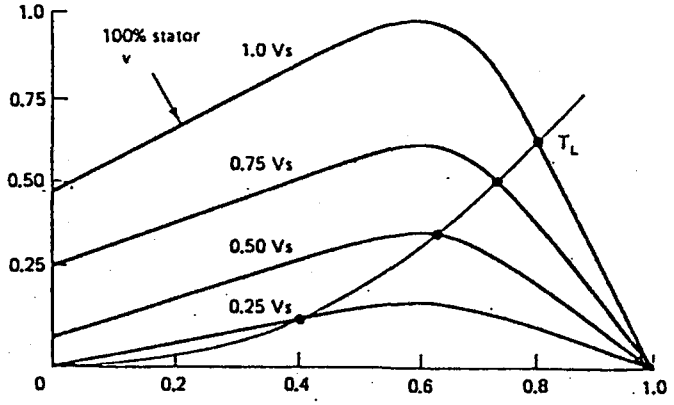
Şekil 3.3 DA Motoru Sürücü Devre Örnekleri

3.1.2. Alternatif Akım Motorları

Bu motorlar (özellikle sincap kafesli olanları), diğer elektrik motorlarına göre oldukça ucuzdur. Fakat, buna karşılık, motorun hız ve konum kontrolü zordur. Sürücü devreleri oldukça karmaşık ve pahalıdır. Bu nedenle, istisnai durumlar haricinde XY tablalarında pek kullanılmazlar. Özellikle yüksek güçlerdeki çalışmalarda, güçlü diyotlar, tristörler, triyaklar, güç tranzistörleri ve güç mosfetleri gerektirirler. Hareket denetiminde ve DA motoruna göre daha fazla güç gerektiren uygulamalarda kullanım alanı bulurlar.

Açık ve kapalı çevrim olarak kullanılabilirler. Açık çevrimde sabit hızda çalışmada, kapalı çevrimde ise ayarlı hızda çalışmada seçilirler. Kapalı çevrimde kullanıldıklarında sürücü devresi oldukça karmaşıklaşır.

Sincap Kafesli Alternatif Akım (AA) Motorlarının en ekonomik hız kontrol yöntemi frekansı sabit tutulmak koşulu ile stator gerilimini değiştirmektir. Bu yöntemin dayandığı temele ilişkin şekil, Şekil 3.4'te verilmiştir. Hız kontrolünde bir başka yöntem ise frekansı değiştirmektir.



Şekil 3.4 Değişik Gerilimlerde Hız-Moment Eğrileri

Sürücüleri karmaşık olduğundan burada örneği verilmeyecektir. AA Motorlarının kullanım alanını sınırlayan bu etkene rağmen sabit hız gerektiren uygulamalarda açık çevrim olarak kullanılırlar. Yüksek güçlerde de başka bir seçenek olmadığından sürücü devresinin dezavantajlarına rağmen, kapalı çevrim kontrol sistemleri ile birlikte kullanılırlar.

3.1.3. Adım Motorları

Günümüzde, XY tablalarında en sık kullanılan motorlardır. Kontrol edilebilme kolaylığı, doğrudan dijital işaretlerle çalışabilmesi, bir çok uygulamada kapalı çevrime ihtiyaç duymaması nedeniyle tercih edilirler. Buna karşılık dar bir çalışma aralığı olması ve karşılayabileceği yük momentinin düşük olması bazı uygulamalarda bu motorları yetersiz kılar.[4]

Adım motoru, uygun bir biçimde kodlanmış elektriksel darbeleri eşit büyüklükte ayrık açısal dönme hareketlerine dönüştüren elektromekanik bir elemandır. Uygun şekilde kontrol edildiğinde, çıkış adım sayısı, girişe uygulanan darbe sayısına eşittir. Adım motoru, doğası gereği ayrık hareketler yapabilen, bu bakımdan sayısal kontrol tekniklerine uyumlu bir düzendir.

Adım motoru, farklı kutupların magnetik olarak birbirini çekmesi ve benzer kutupların itmesi ilkesine göre çalışır. Temel olarak üç tipi vardır. Bunlar Değişken Relüktanslı, Daimi Mıknatıslı ve Karma (Hibrit) Adım Motorlarıdır. Karma motorda magnetik akının kaynağı sabit mıknatıs ve bir ya da daha fazla sargıdan akan doğru akım iken, değişken relüktanslı adım motorunda magnetik akı yalnızca sargı akımları tarafından üretilir.[5]

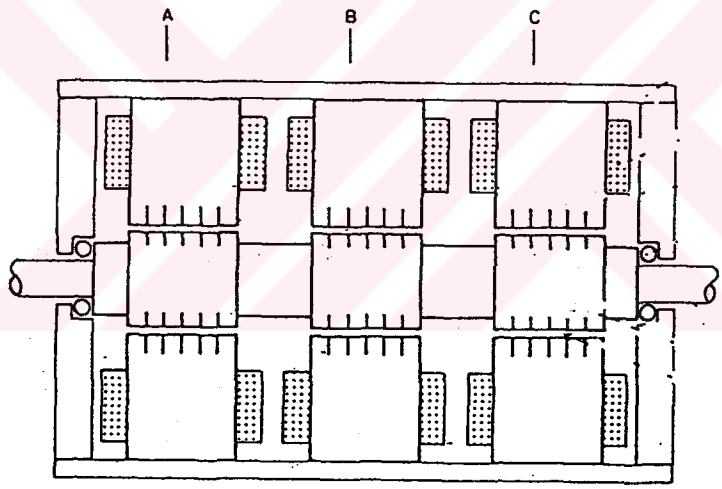
Kataloglarda birçok adım motoru çeşidine rastlanılabilir. Ancak, bu çeşitler aşağıda belirtilecek adım motorlarından türetilmiştir.

3.1.3.1. Değişken Relüktanslı Adım Motoru

Çalışma ilkesi, uyarılan bir stator kutbunun rotor çekirdeğini magnetik olarak çekmesine dayanır. Magnetik akı, daha önceden de belirtildiği gibi yalnızca sargılardan geçen akım tarafından üretilir. Tek katlı ve çok katlı olarak kendi içinde iki çeşide ayrılır.

3.1.3.1.1. Çok Katlı Değişken Relüktanslı Adım Motoru

Ekseni boyunca magnetik olarak birbirinden yalıtılmış katlara bölünmüştür. Her bir kat diğerlerinden bağımsız bir sargıya sahip olduğundan, bağımsız olarak uyarılabilir. Şekil 3.5'te üç katlı bir motor gözükmektedir. Bu tür motorlar kat sayısı en fazla yedi olacak şekilde üretilirler.



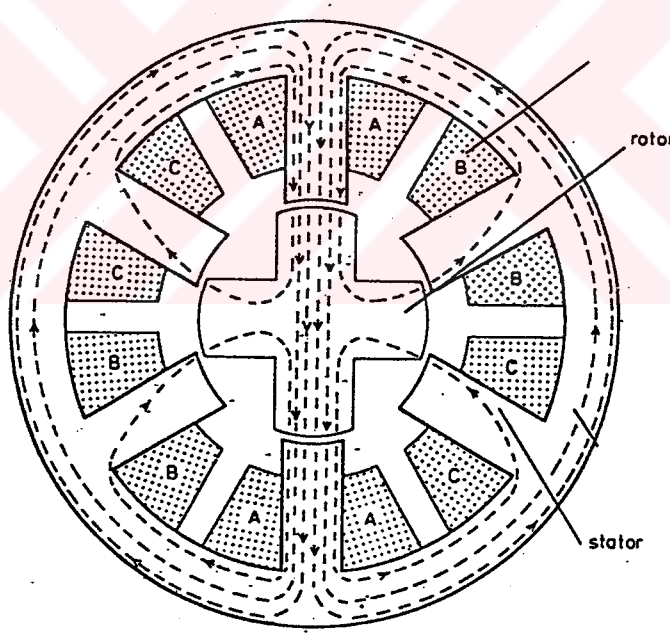
Şekil 3.5 Üç Katlı Değişken Relüktanslı Adım Motoru

Her kat, motorun dış kılıfına monte edilmiş bir sabit (stator) ve bir dönen (rotor) parçadan oluşur. Rotor ve stator, motorun içindeki magnetik alanın büyük girdap akımları oluşturmadan değişebilmesini sağlamak için özel olarak katkılandırılmış demirden yapılır. Her bir kutup, komşu kutubun tersi yönünde sargı ile sarılmıştır. Bir faz sargısı sıra ile tüm kutupları dolaşır.

Çok katlı adım motorları, kullanıcılara adım uzunluğu seçmede kolaylık sağlarlar. İlave katlar yardımıyla adımların küçültülmesi mümkündür. Ancak,her yeni kat yeni bir faz olacağından sürücü devreleri karmaşıklaşır.

3.1.3.1.2. Tek Katlı Değişken Relüktanslı Adım Motoru

Bu adım motorlarında her diş, doğru akımla uyarıldığında radyal yönde bir akı oluşturacak şekilde sarılmış birbirinden bağımsız sargılara sahiptir. Faz sargılarının yerleşim şekli, radyal magnetik alanın bir dişten diğerine doğru yönlenmesini sağlar. Magnetik akının bir kısmının Şekil 3.6'da görüldüğü gibi uyarılmamış stator dişi üzerinden devrini tamamlayıp, motorun faz sargıları arasında ortak bir endüktans oluşturması mümkündür.



Şekil 3.6 Tek Katlı Değişken Relüktanslı Adım Motoru

Bu tür motorlarda, rotor diş sayısı ile stator diş sayısı birbirine eşit değildir. Bu özellik motorun en büyük özelliğidir. Bir faz uyarıldığında, asıl akıyı sadece iki rotor dişi taşır. Bu arada diğer rotor diş çifti ise

uyarılmamış stator dişlerinin tam ortasında yer alır. Eğer uyarılan fazlarda değişiklik yapılırsa, uyarılan stator dişleri ile karşı karşıya gelecek yeni rotor dişleri yine bu dişler olacaktır.

Bu motorda adım genişliği, faz sayısı ve rotor diş sayısı ile doğrudan ilintilidir. n fazlı bir motorda, n fazın sırayla uyarılması sonucu gelinen konum başlangıçtaki konumun eşdeğeridir. Rotor dişleri, yine başlangıçtaki ile aynı stator dişlerinin karşısına gelmiştir. Ancak, rotor bir diş genişliği kadar dönmüş durumdadır. Motor p rotor dişine sahipse, bir rotor diş genişliği 360/p derecedir. n adımlık bir hareket, bu açıya karşılık geldiğine göre adım genişliği 360/n.p olarak bulunur. Stator dişlerinin sayısı, faz sayısı ve rotor dişleri sayısı ile sınırlıdır. Her faz birkaç stator dişi üzerine dağıtılmış olduğundan ve akının rotora girip çıkabilmesi için çift sayıda stator dişine ihtiyaç olduğundan, stator dişlerinin sayısı, stator faz sayısının çift katı olmalıdır.

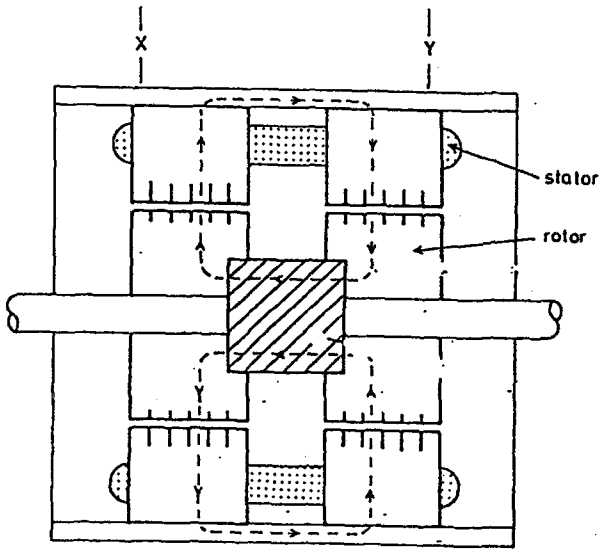
3.1.3.2. Daimi Mıknatıslı Adım Motoru

Bu motorların çalışma ilkesi, uyarılan bir stator kutbunun rotor çekirdeğini magnetik olarak çekmesine dayanır. Stator kutup sayıları ve/veya rotor diş sayıları arttırılarak adım açısı 1.8 dereceye kadar küçültülmüş daimi mıknatıslı adım motorları vardır.

3.1.3.3. Karma Adım Motorları

Karma adım motorlarının rotorunda, eksenel doğrultuda N-S çubuğu şeklinde monte edilmiş bir sabit mıknatıs bulunur. Magnetik akının yolu Şekil 3.7'deki gibidir.

Motorun sürekli dönme hareketi yapması, faz sargılarının sırayla uyarılması ile mümkündür. Karma motorun tam bir uyarma çevrimi dört durumdan oluşmaktadır ve bu da rotor hareketinin dört adımına karşılık gelir. Stator ve



Şekil 3.7 Karma Motor

rotor dişleri bu durumlar sonunda sekans uygulanmadan önceki konumları aynı olacak şekilde yer alırlar. Buradan, rotorun dört adımlık hareketinin motorun bir diş genişliğine eşit olduğu anlaşılır. p dişli motorda bir diş genişliği 360/p olacağından, adım genişliği 90/p derece olur. Tipik adım büyüklüğü 1.8 derecedir.

Adım açısının küçük olması, duyarlılık gerektiren uygulamalarda değişken relüktanslı motorlara üstünlük sağlamasına neden olur. Ayrıca, daha büyük moment üretebilmeleri nedeniyle de tercih edilirler. Karma motorların sargıları uyarılmamış durumda iken sabit mıknatıstan kaynaklanan ve rotoru adım konumunda tutan bir momentin bulunması da güç kaybı durumunda rotor konumunun değişmemesi gereken uygulamalarda yararlı olur.

Buna karşılık, değişken relüktanslı adım motorlarının bu türe iki önemli avantajı bulunmaktadır. Birincisi, adım genişliklerinin büyük olması nedeniyle büyük hareket gerektiren uygulamalarda hızlı sonuç almayı sağlarlar. Ayrıca, uyarma değişimi sayısının az olması kontrol düzenlerinde de yalınlığa neden olur. İkinci olarak, rotorda sabit mıknatıs

bulunmadığından rotorun eylemsizliği küçüktür. Bu da motora gelen yükü oldukça azalttığından ivmelenmeye yardımcı olur.

Adım motorları, faz işaretlerinin uygun sıralanımı ile yarım adımlık hareketler yapabilir. Bu durumda adım açısı, tam adım açısının yarısına eşittir.

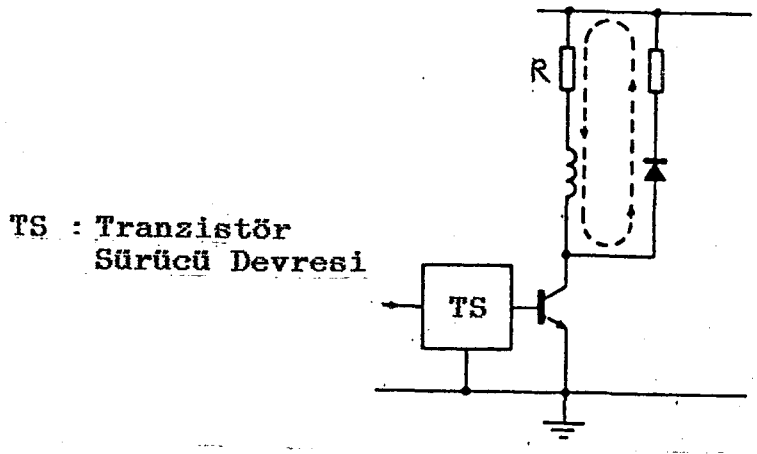
3.1.3.4. Adım Motoru Sürücü Devreleri

Bir adım motorunun adım atabilmesi için motor fazlarına uygulanacak olan darbeler genelde, bir kontrol lojiği tarafından üretilmektedir. Böyle bir lojik devre eğer TTL elemanlardan oluşmaktaysa, devreden en fazla 5V ve 18mA lik çıkışlar alınabilecektir. Bununla birlikte tipik bir adım motorunun uyarma değerleri 5V ve 3A dir. Buradan da görülebileceği gibi, kontrol katı ile motor arasında akım kuvvetlendirmesi yapacak bir başka katın daha bulunması gerekmektedir. Klasik bipolar tranzistörler kullanarak bir kaç sürücü katı ile, ya da alan etkili güç tranzistörleri kullanarak bir katta sürücü devremizi tasarlamak mümkün olmaktadır.

3.1.3.4.1. Temel Sürücüler

Değişken relüktanslı adım motorlarının uyarılması için fazlardan sadece tek yönde akım akıtılır. Dolayısıyla, tek yönlü sürücü devreler yeterli olmaktadır. Karma ya da sabit mıknatıslı adım motorlarında ise fazlardan her iki yönde de akım akmalıdır. Bu amaçla iki yönlü sürücü devreler kullanılır. Bazı tür karma adım motorları için sürücü devrelerde basitlik sağlamak amacıyla, fazlar birbirine ters yönde sarılmış ikişer sargıdan oluşur. Bu sargıların her birinden sadece tek yönde akım akıtılır. Bu tür motorların her fazı iki adet tek yönlü sürücüyle sürülebilmektedir.

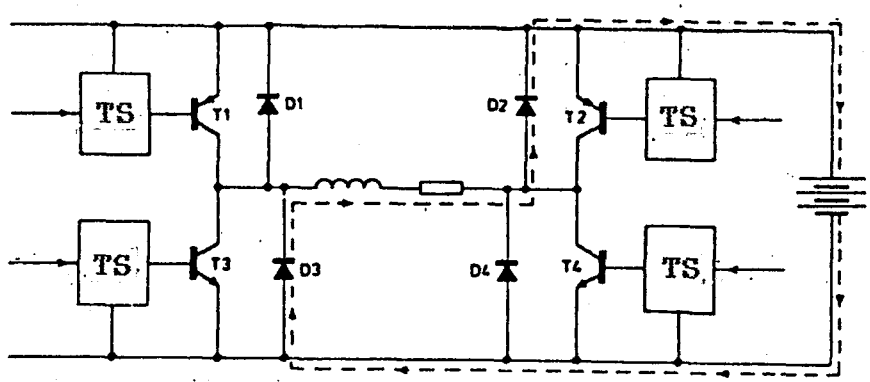
Tek yönlü sürücü devreler, değişken relüktanslı adım motorlarını sürmekte kullanılmaktadırlar. Bir faza ait



Şekil 3.8 Bir Faz için Tek Yönlü Sürücü Devre

ideal devre şeması Şekil 3.8 de verilmiştir. Her bir faz düşük güçlü faz kontrol işareti ile kontrol edilen bağımsız bir sürücü devreye sahiptir. Faz sargısının uyarılabilmesi için, anahtarlama tranzistörü yeterince yüksek bir baz akımı ile doyurulur. Şekilde görülen zorlama direnci R, faz sargısının elektriksel zaman sabitini küçültmek için kullanılır. Böylelikle, faz sargısından nominal akım akıtılabilmesi için kaynak geriliminin yükseltilmesi gerekse de elektriksel zaman sabiti küçültüldüğünden, yüksek hızda çalışmada verimlilik artmaktadır.

Faz sargılarının endüktansı nedeniyle, faz akımı birden kesilememektedir. Tranzistör bazının sürülmesi birden kesilecek olursa, sargılarda endüklenecek gerilim tranzistörün emetör ve kollektörü arasına uygulanacak ve bu da sürücü devreye hasar verecektir. Bu ihtimalin oluşması, faz akımı için serbest geçiş devresi olarak adlandırılan alternatif bir akım yolu oluşturma ile önlenir. Anahtarlama tranzistörü açık devre edildiğinde, faz akımı akışına serbest geçiş diyodu ve R_f direncinin oluşturduğu yol üzerinden akarak azalır. Faz endüktansında kapanma sırasında oluşan enerji, serbest geçiş devresindeki sargı direnci, zorlama direnci, ve serbest geçiş direnci üzerine dağıtılır. Tek yönlü sürücü devrelerin bazı temel özelliklerini belirttikten sonra sabit mıknatıslı ve karma adım motorlarında kullanılan iki yönlü sürücülere değinelim.



TS : Tranzistör Sürücü Devresi

Şekil 3.9 Bir Faz için iki Yönlü Sürücü Devre

Şekil 3.9 da gösterilen tranzistör köprülü, iki yönlü sürücü devrelerde istenen akım yönüne göre tranzistörler çifter çifter anahtarlanırlar. Örneğin, fazın pozitif olması için T1 ve T4 tranzistörleri devreye alınırken, negatif uyarma için T2 ve T3 tranzistörleri anahtarlanır ve faz sargısının akım yönü tersine döner.

Tranzistör köprü sürücülerde, iki faz kontrol işaretinin kuvvetlendirilmesi için dört anahtarlama tranzistörü ayrı ayrı baz sürücü devrelerine ihtiyaç duyarlar. T1 ve T2 tranzistörleri, çeşitli gerilimlerde olabilen pozitif kaynak hattını referans alırlar. Bu nedenle üst hatta bağlı olan baz sürücülerine gelen faz kontrol işaretleri optik olarak izole edilmelidirler.

Anahtarlama tranzistörlerine ters paralel bağlı dört diyotlu köprü serbest geçiş akımı için bir yol oluşturur. Şekildeki D2 ve D3 diyotlarının oluşturduğu serbest geçiş yolu, T1 ve T4 tranzistörlerinin kesime gitmesinin hemen ardından oluşmuştur. Serbest geçiş yolu, DA kaynağı da içerdiğinden, faz sargılarının endüktansında tranzistörün kesime gitmesi sırasında biriken enerjinin bir kısmı tekrar kaynağa dönmektedir. Bu durum, iki yönlü köprü sürücüler için dikkate değer bir avantaj sağlar. Serbest geçiş akımı bu tür sürücülerde DA kaynak geriliminin de serbest geçiş yolu üzerinde bulunması nedeniyle çabuk azalır.

3.1.3.4.2. Geliştirilmiş Sürücü Devreler

Hem karma hem de değişken relüktanslı adım motorlarının hızlarının değişim sınırı faz dirençlerine bağlıdır. Adım motorlarının yüksek hızlarda çalışabilmesi için faz akımının sargılarda hızlı yükselmesi gerekmektedir. Bu da sargıların endüktans/direnç zaman sabitlerinin küçük olmasını gerektirir. Bu amaçla, bir önceki bölümde anlatılan basit sürücülerden büyük değerli zorlama direnci kullanılarak yararlanılabilir. Fakat, faz akımının sürekli halde nominal değerine yükselebilmesi için kaynak geriliminin de zorlama direnciyle orantılı olarak arttırılması gereklidir. Küçük motorlar için kaynak gerilimini arttırmak problem olmazken, büyük güçlü motorlarda bu yöntem iyi sonuç vermez.

Büyük kaynak gerilimi ve faz direnci yalnızca motor yüksek hızlarda çalışacaksa gereklidir. Eğer motor hareketsizse, kaynak gücünün büyük kısmı seri zorlama direnci üzerine dağıtılacak ve bu direnç yeterince soğutulamıyorsa buradaki sıcaklık, problemlere neden olacaktır. Bu nedenle zorlama direnci hız değişim bölgesini genişletmede çok verimli bir yöntem değildir. Yüksek hızlarda mekanik çıkış gücü arttırılsa bile, düşük hızlarda kaynağın gücü dirençler üzerinde harcanmaktadır.

Sonuç olarak, sürücü devreden beklenen özellik yüksek hızlarda büyük kaynak gerilimi üretebilmesi, düşük hızlarda ise temel seri dirençli sürücü devre yöntemindeki gibi güç kaybına neden olmaksızın faz akımını sınırlamasıdır. Bu amaçla pek çok çeşit sürücü devre tasarlanmıştır.

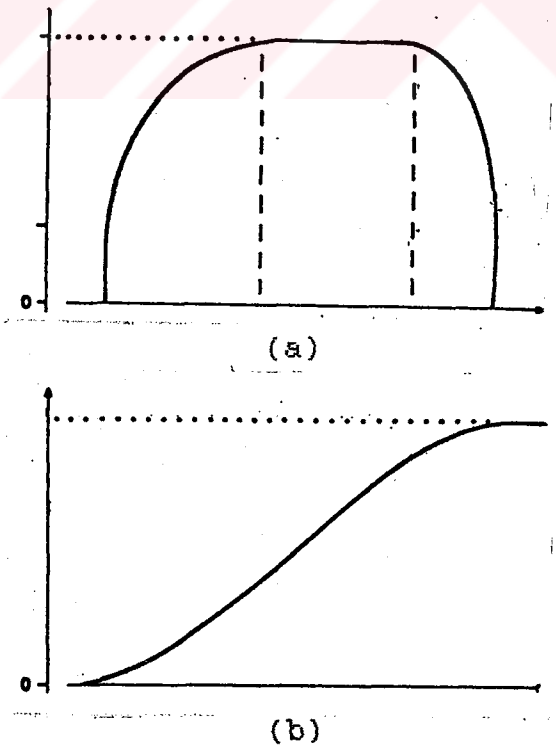
3.1.3.5. Adım Motoru Kontrol Düzenleri

Adım Motoru kontrol düzenleri açık ve kapalı çevrim olarak başlıca iki gruba ayrılmaktadır. Kısa sürede hızlanma ve yavaşlama gerektiren, yüksek hızlarda ve değişken yük momenti altında çalışılacak yerlerde kapalı çevrim

kontrol sistemleri kullanılır. Çok yüksek hız gerektirmeyen ve yük momentindeki değişmelerin kritik olmadığı uygulamalarda ise açık çevrim kontrol sistemleri tercih edilir.

3.1.3.5.1. Açık Çevrim Kontrol Düzenleri

Bu tür sistemlerde çalışmayı anlamak açısından birkaç önemli tanımı vermek yararlı olacaktır. Verilecek gerekli tanımlardan ilki, bir adım motorunun kalkışta maksimum hız değeridir. Bu değer, genelde sürekli rejimdeki maksimum hızından daha aşağıdadır. Bu nedenle yükü belli bir konuma getirmek için harcanacak zaman, başlangıçtan itibaren motor hızını sürekli rejimdeki maksimum hıza çıkana kadar arttırarak ve harekete bu maksimum hızda devam edip hedeflenen konuma yaklaşıldığında motorun hedefte durmasını sağlayabilmek için hızı maksimum durma hızına düşürerek azaltılabilir. Sistemdeki bir başka önemli tanım hız profili tanımıdır. Bu kavram adımlama hızının zamana göre değişimi olarak verilir. Şekil 3.10'da örnek bir hız profili verilmiştir. Görülmektedir ki, yavaşlama hızlanmadan daha çabuk



Şekil 3.10 Motorun Hız Profili ve Konum-Zaman Eğrisi

olmaktadır. Bunda, sistemdeki yük momentinin yavaşlatıcı etkisinin rolü büyüktür. Bir açık çevrimli adım motoru kontrol sisteminde optimum hız profilini üretmek için, adım motoruna ait sürekli rejimde maksimum yük momenti eğrilerinden yararlanılır. Hız profilinin kesin şekli, düşük maliyet ve yüksek performans istekleri arasında bir oran belirlenerek saptanabilir. Örnek olarak, optimum bir hızlanma için üstel bir eğri istenebilir, fakat uygulanması pahalıdır. Böyle bir durum çok ucuz maliyetli lineer bir rampa fonksiyonu ile kompanze edilir.

Hız profilini gerçeklemek için çeşitli yöntemler kullanılabilir. Bu yöntemler, aşağıda genel başlıkları altında verilecektir.

i) Mikroişlemcili tasarım, hız profili çıkarmada en çok kullanılan yöntemlerdendir. Bir mikroişlemci, adım motorunun kontrolünde kullanılacak olan faz uyarma darbelerinin üretilmesi için çok uygundur. Ancak, mikroişlemciyi sadece tek bir adım motoruna atamak pahalı bir yöntem olacaktır.

ii) Özel Donanımlar Kullanılması, hız profili çıkarmada bir başka yöntemdir. Eğer sistemin hızlanması, bir kaç adımlık bir hareket sonunda sona eriyorsa, faz uyarma darbeleri üretecine gönderilecek saat darbelerinin bir lojik devre tarafından üretilmesi daha uygun olacaktır. Sistemin hızlanması ve maksimum hızda hareketi sırasında atacağı adım sayısını belirlemek için bir sayıcı kullanılabilir. Bu arada, hızlanma ve yavaşlama işlemi için üretilecek darbeler, birer geciktirici lojik devre dizisi tarafından üretilir.

iii) Darbe eleme yönteminde, uyarma darbelerinin zamanlamaları diğer yöntemlerdeki kadar iyi kontrol edilemese de basitlik ve düşük maliyet bakımından avantajlıdır. Yöntemin esası, sabit frekanslı bir saat üretecinden gelen darbelerin bir kısmının adım atma darbesi olarak alınıp bir

kısmının elenmesine dayanır. Adımlar, saat darbelerinin bir flip-flop devresinden geçirilerek elenmesi sonunda farklı hızlarda üretilirler. Çalışma hızını arttırmak için saat frekansı arttırılır ve darbe eleme işlemi çoğaltılır. Ancak, yine de hız değişimi sınırlıdır. Tek bir saat işaretinden elde edilen ayrı hızlar, sadece 10 un katları şeklinde olabilmektedir.

iv) Analog hızlanma ve yavaşlama, motorun sürekli rejimde maksimum hızı değişmiyorsa kullanılabilir. Böyle bir durumda hızlanma hareketindeki ivme sabittir ve sistemden optimum performans alabilmek için hızlanma bir lineer rampa şeklinde olmalıdır. Bu ise, motorun hızlanması için devreye giren yavaşlaması için ise devreden çıkan bir işaretin integralinin alınması ile mümkündür.

3.1.3.5.2. Kapalı Çevrim Kontrol Düzenleri

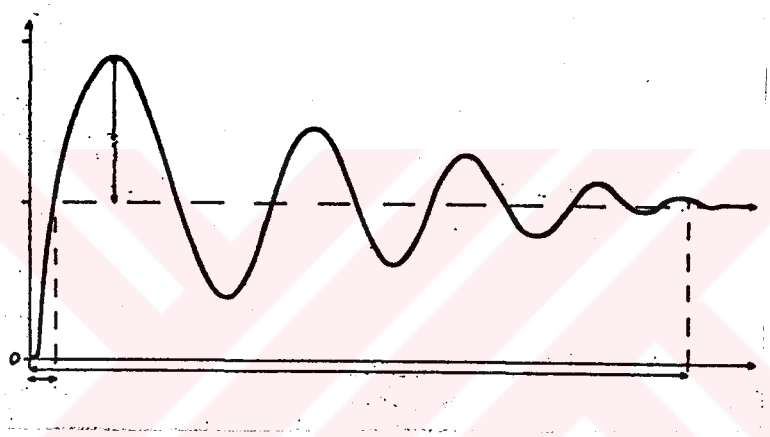
Kapalı çevrimli adım motoru kontrol sistemlerinde rotor konumunun ani değerinden denetim birimine bir geri besleme alınması, giriş darbeleriyle rotor hareketi arasındaki senkronizasyonu sağlar.

Bu sistemlerde üzerinde durulması gereken noktalar, rotor konumunun algılanması, maksimum moment elde edebilmek için hız değiştikçe uyarma açılarının buna bağlı olarak ayarlanması ve yavaşlama işlemi için gerekli olan optimum adım sayısının saptanmasıdır. Gerek çok fazla ihtiyaç duyulmadıkça kullanılmaması, gerekse sistemimizde açık çevrim kontrollü adım motoru kullanılması nedeniyle bu bölümle ilgili daha fazla açıklama yapılmayacaktır.

3.1.3.6. Adım Motorunun Tek Adım Cevabı

Motor fazlarından biri uyarılmış durumda iken motor, kararlı bir adım konumundadır. Bu fazın uyarılması kaldırılıp bir diğer faz uyarılırsa motor bir adım atacaktır. Rotor konumunun zamana göre bu değişimi tek adım cevabı

olarak tanımlanır. Tek adım cevabı, motorun adım hareketinin hızını, cevabın aşım ve salınım miktarını, adım açısının hassaslığını veren önemli bir büyüklüktür. Şekil 3.11 de tipik bir tek adım cevabı gözükmektedir. Yeni bir faz uyarıldığında, rotor ve stator dişleri arasında oluşan moment, rotoru yeni kararlı konuma doğru çekecektir. Bu moment, rotor adım konumuna gelene kadar devam edecek, rotor bu konumu geçip aşım yapınca da ters yönde bir moment olarak rotora etki edecektir. Rotor, adım konumundan tekrar geriye geçince, bu moment yine pozitif yönde rotora etki edecek ve bu hareket rotorun kinetik enerjisinin tümü kaybolana kadar devam edecektir.



Şekil 3.11 Adım Motorunun Tek Adım Cevabı

İşte, adım motorunun halledilmesi gereken sorunlarından biri bu salınımların söndürülmesidir. Bu amaçla çeşitli yöntemler geliştirilmiştir. Bu yöntemler :

1) Mekanik Söndürme Yöntemleri : i) Frenler, ii) Viskoz eylemsizlik söndürücü, iii) Sürtünme ve eylemsizlik söndürücü, iv) Viskoz Söndürücü, v) Girdap akımı ve eylemsizlik söndürücü, vi) Pnömatik söndürücü, vii) Elastik kuplajlı söndürücü.

2) Kontrol Devresiyle Söndürme Yöntemleri : i) Dirençle söndürme, ii) Kapasite ile söndürme, iii) Tek faz uyarma ile söndürme, iv) iki faz uyarma ile söndürme.

3) Elektronik Söndürme Yöntemleri : i) Geri faz uyarma ile söndürme (bang-bang kontrol), ii) Son adım gecikmesi ile söndürme.

4) Motor Parametrelerinin Tasarımıyla Söndürme Yöntemleri : i) Yardımcı stator sargıları, ii) Diş şekillerinin etkisi ile, iii) Yardımcı sabit mıknatısla.

Kullanılması ve ilginçliği nedeniyle, elektronik söndürme yöntemlerinden biri olan son adım gecikmesi ile söndürme yöntemine değinilecektir.

Adım motorunun adım yanıtında 80-90% lara ulaşan bir aşım olduğundan, bu aşımdan çok adım atma sırasında yararlanılması düşünülmüştür. Motor birden fazla adım hareketi yaparak bir konuma gelecekse, son adımdan bir önceki adımda motora uygulanan giriş darbeleri dizisine son verilir ve rotorun bir sonraki adım konumuna yaklaşan bir aşım yapması beklenir. Son darbe, bu maksimum aşım noktasında motora uygulanır ve motor ya çok az ya da aşımsız bir şekilde istenen konuma oturur. Bu nedenle, son adım gecikme söndürmesinin uygulanabilmesi için motor en azından iki adım atacak olmalıdır. Ancak, pratikte bir adımın sonunda motor aşımla bir sonraki adım konumuna gelebilecek kadar hız kazanmış olamaz. Aşımın bir sonraki adıma ulaşabilmesi için en azından iki adım atılarak motorun hızlandırılmış olması gereklidir. İkinci adımın uygulama zamanı da aşımda istenen bu genlik değerini oluşturabilecek şekilde ayarlanmalıdır. Dolayısıyla, en az üç adımlık hareketlerde bu yöntemi kullanmak daha iyi sonuç verir. Pratikte, üç fazlı motorlar için üç adım, dört fazlı motorlar için dört adımlık hareketler yapmak yapmak yararlıdır. Bu yolla hareket hep aynı fazın ayarlanması ile başlar ve aynı fazla biter. Faz yanıtında değişiklik olmaz ve konum hatası olasılığı azalır. Bu yöntem, parametre değişimlerine ve bozucu etkilere az duyarlıdır.

3.2. Sistemdeki Motorlar ve Sürücü Devresi

Tezin konusu olan örnek XY tablasında, eksenlerdeki hareketler kolay kontrol edilebilirliği ve mikroişlemcili sistemlerde kullanılabilirliği sebebiyle karma (hibrit) adım motorları ile verilmekte ve sistemdeki motorlar bir bilgisayar tarafından açık çevrim olarak kontrol edilmektedir. Adım motorunu sürmek için Bölüm 3.1.3.4.1. de açıklanan ve hakkında gerekli bilgiler verilen iki yönlü sürücü devre kullanılmıştır.

Mikroişlemcili kontrol sistemlerinin bir örneği olan bu sistem yazılım ya da donanım ağırlıklı tasarlanabilirdi. Yani, motor için gerekli faz işaretlerinin sırası, işaretlerin zamanlaması ve gerekli hesaplamalar mikroişlemci tarafından yapılabilirdi (Yazılım Ağırlıklı). Ya da hesaplamalar yine mikroişlemcide fakat faz işaretleri sıralanımı ve zamanlama donanım tarafından üretilebilirdi (Donanım Ağırlıklı). Bir sistem tasarlanırken, tasarımcı hangi yöntemi kullanacağını adım motoruna ve işlemciye göre belirlemelidir. Donanım maliyetini azaltmak için yazılım ağırlıklı yönteme başvurulabilir. Ancak, bu yöntemde hızın sınırlı olması kısıt olarak karşımıza çıkmaktadır. Mikroişlemcinin performansı ve sığası bu hızı belirler. Bu nedenle bazı uygulamaları donanım ağırlıklı yapmak kaçınılmazdır. Bazı uygulamalarda ise bu iki yöntemin birleşimi kullanılabilmektedir. Bizim tercihimiz, maliyeti arttırmamak için yazılım ağırlıklı tasarımdan yana olmuştur.

3.2.1. Adım Motoru Kontrol Yazılımı Temel Yapısı

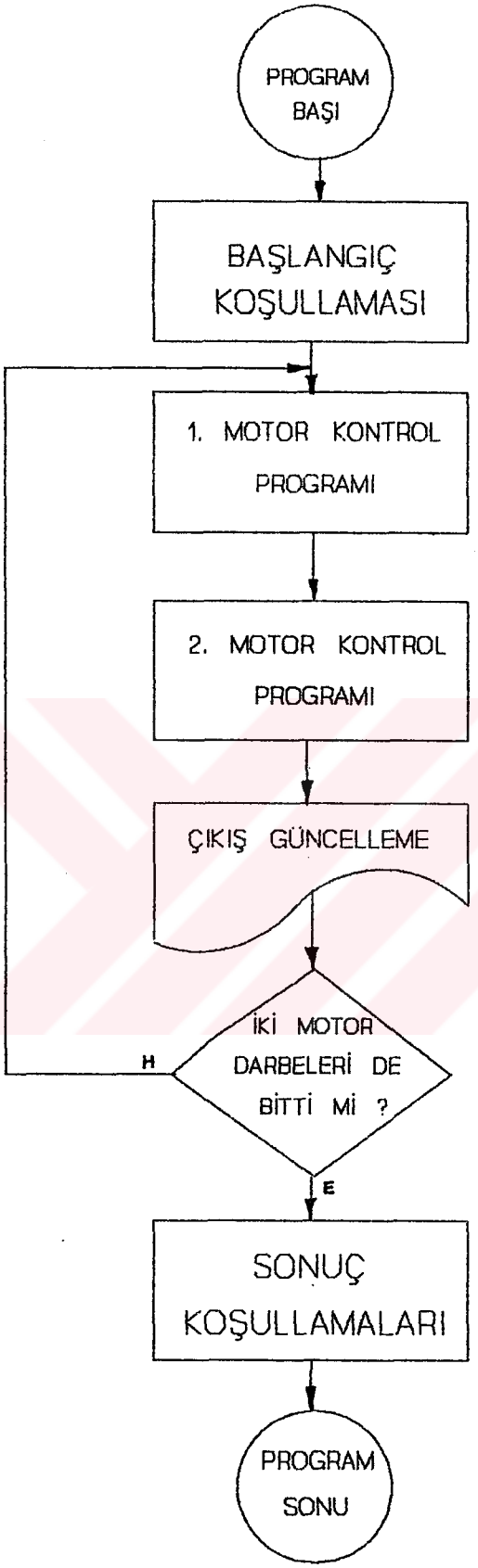
Motorları sürmek için koşturulan yazılım, Intel 8086 birleştirici dilinde yazılmış bir programdır. Kod uyumluluğu sebebiyle, aynı ailenin üst düzey işlemcilerinde de koşturulabilir. Bu program, yüksek düzeyde (PASCAL) yazılmış bir program tarafından çağrılmaktadır. Bu üst düzey programdan alınan, her iki eksen için adım sayısını ve yönü

belirten bilgiler sayesinde, motorlara belirtilen yönde ve belirtilen kadar adım hız-moment eğrisine göre attırılır.

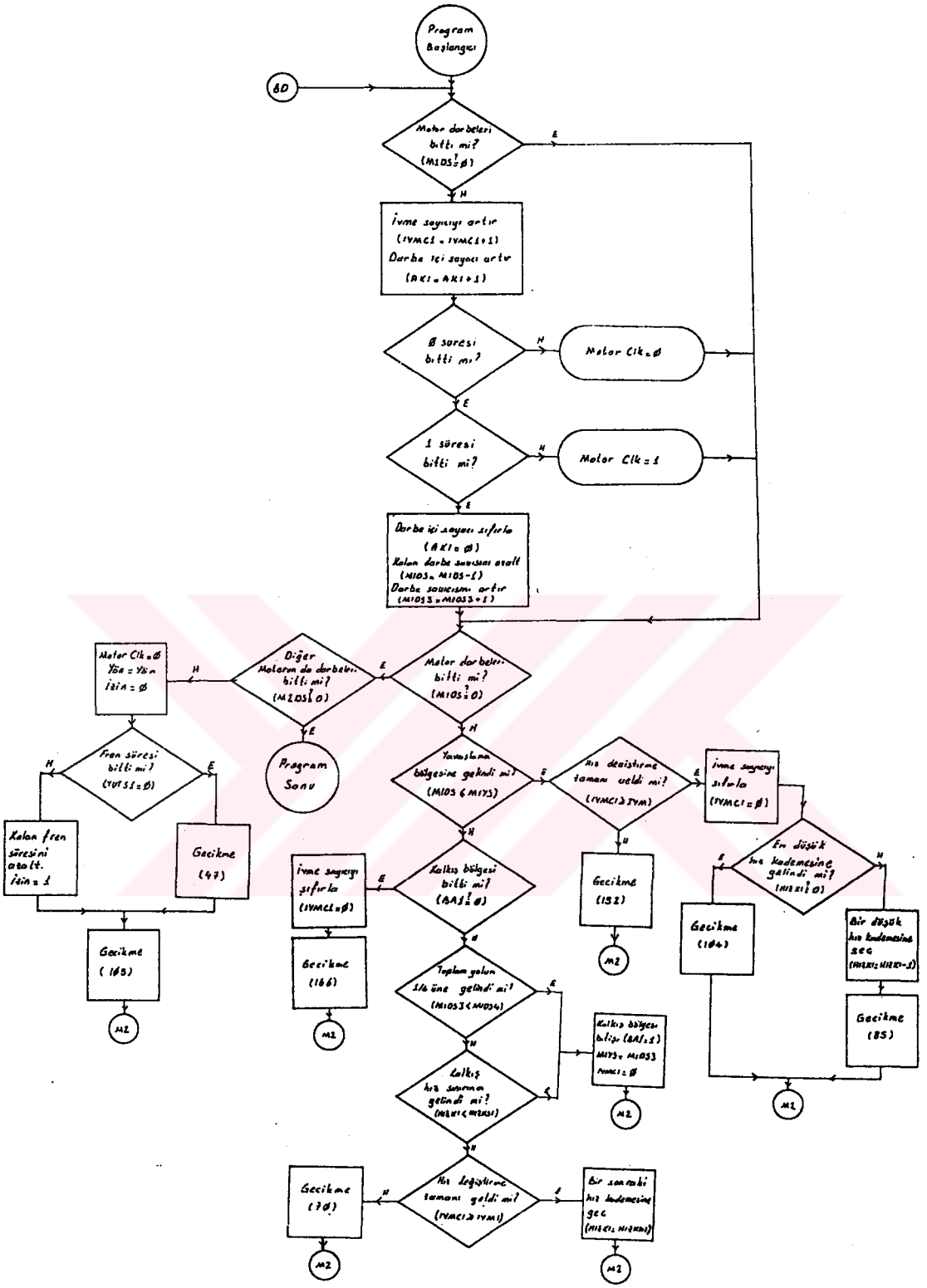
İşlemcinin iş saklayıcıları 16 bit olduğundan,bu yazılım bir adım motoruna bir seferde ve bir yönde ancak 0 ile 65535 adım aralığı kadar yol aldırabilmektedir. Bu miktar, bizim uygulamamız için yeterli olmaktadır. Bu yazılımda ayrıca her bir motora ilişkin bir adet izin biti de kullanıcıya sunulmuştur. Hareket etmesi istenmeyen bir motor, bu izin bitinin 0 verilmesi ile bloke edilebileceği gibi adım sayısı 0 verilerek de hareket etmemesi sağlanabilir.

Program, genel yapı olarak başlangıç ve sonuç koşullamaları dışında gövdesinde koşan ana bir döngüden oluşmaktadır. Bu temel yapı Şekil 3.12'de de görülmektedir. Bu döngüden çıkmanın tek yolu, her iki motor için de hareketin bitmesidir. Bu döngüden çıkıldığında da program sonuçlandırılır. Sadece bir tek motorun hareketi bitti ise bu motora ait program parçası motorun hareketini durdurur. Fakat, motorun hareketli olduğu zaman kadar bir gecikme oluşur. Böylece bir motorun hareketi bittiği zaman diğerinin hızı değişmemektedir. Temel olarak, point to point, yani motorların birbirinden bağımsız hareketini öngören bu algoritma ve kontrol yöntemi, motorların adım sayıları oranlanması ve motorlardan az adım atacak olanının tek adım çalıştırılmasıyla doğru hareketi yapan bir program parçasına çevrilebilmektedir. Nitekim, program bu şekli ile kullanılmış ve verilmiştir. Yalnız, programın bu şeklinin de x ve y de atılacak adım sayıları oranına bir sınır getireceği de kuşkusuzdur. Adım motorlarının dar bir frekans bandında kontrol edilebilmesi ve elimizdeki bilgisayarın hızının sınırı nedeniyle bu oran 10 lu mertebelerde olmaktadır.

Hareketin çeşitli aşamalarında motor hızlarının istek dışı değişmelerine karşı program gerekli yerlerine gecikmeler konularak programın her koşulda aynı sürede işlemesi



Şekil 3.12 İki Eksen, Adım Motor Kontrol Algoritması



Şekil 3.13 Tek Motor için Kontrol Algoritması

sağlanmıştır. Bu yapı, Şekil 3.13 de ayrıntılı olarak gözükmektedir. Bu yapıyı kurmamızın temel nedeni, istek dışı hız değişimlerinin neden olabileceği ivmelendirmedeki hataları, adım kaçırma ve ani moment düşmesini önlemektir.

3.2.2. Sistem Hız Sınırları Ve Nedenleri

Bu çalışmada, yaklaşık 1700 Hz lik (3 program çevrimi) bir hıza ulaşılmıştır. Motorun ve işlemcinin izin verdiği maksimum hızların (10 KHz - 2700 Hz) altında kalınmasının çeşitli sebepleri bulunmaktadır. Maksimum hız motor yükü ile doğrudan ilintilidir. Motor miline süreceği sisteme eşdeğer bir atalet diski bağlandığı vakit hızda bu değerlere ulaşıldığına rağmen, bu disk çıkarıldığında motor yaklaşık olarak 1000 Hz. den yüksek frekanslarda doyarak durmaktadır. Bunun nedeni, atalet diskinin hareket ivmesindeki ani değişimleri bastırmasıdır. Daha detaylı olarak açıklayacak olursak; program hız değişimlerini ayrıklaştırmaktadır. Örneğin, bir motor darbesi üç program çevriminde verilirken motor hızı yaklaşık olarak 1700 Hz. dir. Bir sonraki kademede üretilebilen en yaklaşık hız iki program çevrimi sürer ki bu da 2680 Hz. e karşılık gelmektedir. Bu artış (3/2 oranında), yüksek hızlarda mümkün olmayan bir ivmeye sebep olur. Sonuçta, hızlanması beklenen motor ani bir moment kaybı ile doyarak durur. Atalet diski, işte bu etkiyi hafifletmekte ve daha yüksek hızlara ulaşmamızı olanaklı kılmaktadır.

3.2.3. Motorun İvmeli Kalkış-İniş Eğrisinin Seçilmesi

Motorun, ivmeli kalkış ve iniş eğrisinin elde edilme yöntemleri de bundan önceki bölümlerde irdelenmişti. En basitinden bu eğrinin rampa fonksiyonu, daha karmaşığın ise üstel fonksiyonlar olduğu bilinmektedir.

Öncelikli olarak, sistemimizin kalkış eğrisi üzerinde çalışmalar yapılmıştır. Bu eğri, ilk olarak bir lineer rampa fonksiyonu alınmış ve denemeler yapılmıştır. Fakat,

gerek düşük hızlarda fazla zaman kaybedilmesi, gerekse yüksek hızlarda programın ayrıklaştırma etkisinin büyük olması nedeniyle başarılı sonuçlar elde edilememiştir. Daha sonra, üstel kalkış eğrisi denenmiş ve iyi sonuçlar elde edilmiştir. Bu eğrinin genel şekli (3.1) nolu formülde verilmiştir.

$$y = k\cdot(1-e^{(-t/\tau)}) \qquad (3.1)$$

Bu çalışmadan sonra, iniş eğrisi üzerinde aynı yazılım çalışmaları devam ettirilmiştir. Bu eğrinin de kalkış eğrisi ile aynı olması gerektiği sonucuna varılmıştır. Yük özelliklerinin hızlanma ve yavaşlama açısından simetrik olması (Sürtünme çok az, sadece atalet var), nedeni ile bu sonuç normal bulunmuştur. Kalkış ve iniş eğrilerinin aynı olması programı basitleştirmekte ve hızı da arttırmaktadır.

Çalışmanın son kısmına doğru parabolik hızlanma ve yavaşlama eğrisi üzerinde çalışmalar yapılmış ve daha önceki çalışmalar ile karşılaştırıldığında daha iyi sonuçlar alındığı görülmüştür. Bu nedenle, sistemde kullanılmasına karar verilmiştir. Bu eğri 3.2 genel formunda verilebilir.

$$v = -a\cdot(t-t_0)^2+b \qquad (3.2)$$

Bu eğri 3.3 ve 3.4 formüllerinde verildiği şekilde ayrıklaştırılıp, bir peryot tablosuna dönüştürülmüştür.

$$v(i) = \frac{-(v_{max}-v_{min})}{i_{max}^2}\cdot(i-i_{max})^2+v_{max} \qquad (3.3)$$

$$p(i) = Round(f_{pr}/v(i)) \qquad (3.4)$$

Verilen formüllerde, vmax: Sonuçta ulaşılacak motor hızı, vmin: Motor kalkış hızı, yani minimum hız, i: Hız kademe sayıcısı (1..imax), imax: vmax a ulaşıldığındaki kademe, fpr: Programın çıkış güncelleştirme sıklığıdır.

Sonuçta, motor minimum hızdan başlatılarak (i=1) mekanik özelliklerce belirlenen ivmenin belirttiği sıklıkta hız arttırımıyla maksimum hızına ulaştırılır. Maksimum hıza ulaşıldığında yavaşlama bölgesine gelinceye kadar sabit hızla devam edilir. Yavaşlama bölgesinde aynı tablo geriye doğru taranarak yavaşlama ve durma sağlanır.

Yavaşlama bölgesinin nereden başladığına dair karar, hızlanma sırasında geçen süreye bakılarak verilir. Kısa mesafelerde motorun yüksek hıza çıkabilmesi durmayı zorlaştırdığından, yolun 1/4 üne gelindiğinde hız arttırılması durdurulur. Bu hız, o hareketteki maksimum hızdır. Fakat, yol uzun olduğundan motorun izin verdiği maksimum hıza yolun 1/4 ünden önce ulaşılır ve hız arttırımı orada kesilir. Maksimum hız ne olursa olsun, sonuçta hedefe ulaşmak için geçen süre yavaşlama bölgesi genişliği olarak tutulur. Motor, hedefe bu yavaşlama bölgesi genişliği kadar yaklaştığında yavaştılmaya başlanır, darbeler bittiğinde ise durdurulur. Her bir motor durduğunda, kısa bir süre elektriksel fren yapılır. Bu, hareketli sistemin ataletini ve sistemdeki salınımları yenmek için gereklidir.

3.2.4. Sistemde Performans Arttırma Yöntemleri

Uygulamada karşılaşılan zorluklardan biri kullanılan mikroişlemcinin yavaşlığıdır. Açıklanan program ilk olarak PASCAL yüksek düzey dilinde yazılmış, ancak 200 Hz. lik frekanslara ulaşılabilirken daha sonra birleştirici dilde yazılan programla çok daha yüksek performanslara ulaşılmıştır. Bu program, 8088 işlemcili bir makinada koşturulduğundan, performans yine de iyi olamamıştır. Örneğin, bir 80286 lı makina kullanılırsa, mikroişlemciden doğan sistem sınırı kalkacak ve sadece motorun sürülebileceği maksimum frekans sınırı sistemi sınırlayacaktır. Böylece, kalkış iniş eğrisi de daha hassas olarak taranabilecektir. Buna rağmen maliyet açısından kullanılan işlemci yeterli bulunmuştur. Her türlü inceleme ve açıklama kullanılan işlemcinin davranışına göre yapılmıştır.

Sistemde performans arttırmak için bir başka yöntem ise, yukarda anlatılan motor kontrol yazılımının PC ye takılabilen akıllı bir birimde koşturulmasıdır. Bu birimde yer alacak, yeteri derecede hızlı bir mikroişlemci motora yeterli hassasiyette bir kontrol uygulayabilecektir. Bu sayede, bilgisayar da motorlardan bağımsız olarak işlem yapabilecektir. Bu yöntemle, birden fazla tezgah PC de performans kaybına neden olmadan kontrol edilebilecektir. Tek bir tezgah için pratik olamayan bu çözüm, tezgah sayısı arttıkça kullanılabilir bir tasarım haline gelecektir.

3.2.5. Sistem Donanımı

Mikroişlemcili XY tablası kontrol sistemlerinde, gerek sürme özelliklerinden (genelde mikroişlemciler 1 TTL yük sürebilecek yapıdadırlar), gerekse mikroişlemciyi sistemden biraz dahi olsa yalıtabilmek için paralel iletişim arabirimleri kullanılır. Bu arabirimleri sistemimize ekleyebilmek için, sistem adres uzayımızda bir adrese yerleştirmemiz gerekmektedir. Bu amaçla da adres kod çözücü devrelere ihtiyacımız olmaktadır. Adres kod çözücü devreler, basit lojik kapılardan oluşabileceği gibi, bu amaçla üretilmiş özel entegre devrelerden biri de olabilir.

Sistemimizde paralel iletişim arabirimi (PIA) olarak, kullandığımız mikroişlemci ile uyumlu olması bakımından I8255 kullanılmıştır. Bu birimin temel özellikleri bundan sonraki paragraflarda irdelenecektir.[6]

I8255 PIA sı içinde, dört adet iskele bulunmaktadır. A, B, Calt, Cüst olarak adlandırılan bu iskelelerden, A ve B sekiz bitlik, Calt ve Cüst ise dörder bitliktir.

I8255, 40 bacaklı bir tümdevre olarak üretilmiştir. Bacakları ve bu bacakların işlevleri şunlardır :

Reset : Lojik 1 de etkin olan bu giriş etkinleştirildiğinde, A, B, C ve denetim kütüklerinin içerikleri sıfırlanır.

CS : Kırmığın seçimini sağlayan giriştir. Lojik 0 da etkindir.

RD : Birimden okuma yapılacağını belirten giriştir. Lojik 0 da etkindir.

WR : Birime yazma yapılacağını belirten giriştir. Etkin durumu Lojik 0 dır.

A0, A1 : Bu girişler, I8255 içindeki alt birimlerin seçilmesinde, Okuma ve Yazma girişleri ile birlikte kullanılır. A0, A1, RD, WR, ve CS e göre seçimin nasıl yapıldığı Tablo 3.1 de verilmiştir.

Tablo 3.1 I8255 içindeki Birimlerin Seçim Yöntemi

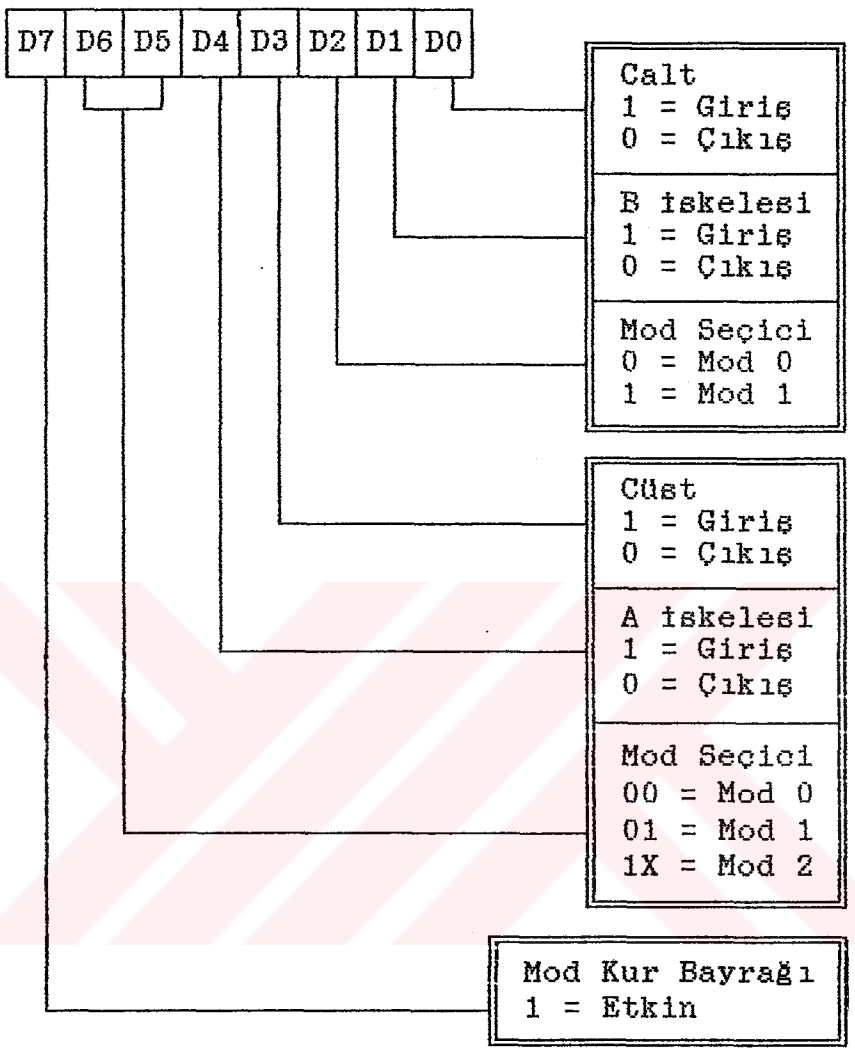
A1	A0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	İŞLEM
0	0	0	1	0	A İSKELESİ ---> VERİ YOLU
0	1	0	1	0	B İSKELESİ ---> VERİ YOLU
1	0	0	1	0	C İSKELESİ ---> VERİ YOLU
0	0	1	0	0	VERİ YOLU ---> A İSKELESİ
0	1	1	0	0	VERİ YOLU ---> B İSKELESİ
1	0	1	0	0	VERİ YOLU ---> C İSKELESİ
1	1	1	0	0	VERİ YOLU ---> DENETİM K.

Veri Yolu : PIA ile mikroişlemci arasında çift yönlü veri alış verişini sağlar.

İskeleler : A ve B iskeleleri sekiz bitlik olup hem alıcı hem de verici olarak koşullanabilir. İskelenin alıcı veya verici koşullanması durumunda o iskelenin tümü koşullanmış olur. Yani aynı iskelenin bir biti alıcı iken farklı bir biti verici olamaz. C iskelesi, sekiz bitlik bir iskele olarak kullanılabileceği gibi dörder bitlik iki ayrı iskele olarak da kullanılabilir. C iskelesinin üst yarısı, A alt yarısı, B ile birlikte de kullanılabilir. Bu tür kullanım genelde, el sıkışmalı çalışmalarda gerekli olmaktadır.

İskelelerin kullanım biçimleri, Denetim kütüğü içine yazılan verilerle belirlenir. I8255 in koşullanması üç farklı şekilde olabilmektedir. Bu çalışma biçimlerinin

Tablo 3.2 Denetim Kütüğünün Koşullanması ve Modlar



nasıl sağlandıkları Tablo 3.2 de verilmiştir. Salt yazılabilen denetim kütüğünün nasıl seçilebildiği Tablo 3.1 de verilmişti.

0 Modunun seçilmesi durumunda, A ve B iskelesinin tüm kapıları ya alıcı ya da verici durumda koşullanmış olurlar. C iskelesinin kapıları ise, Denetim kütüğünün D0, D1, D3, D4 bitlerine uygun şekilde koşullanırlar. Tablo 3.3 de bu bitlere göre A, B ve C iskelelerinin alacağı durumlar gösterilmiştir.

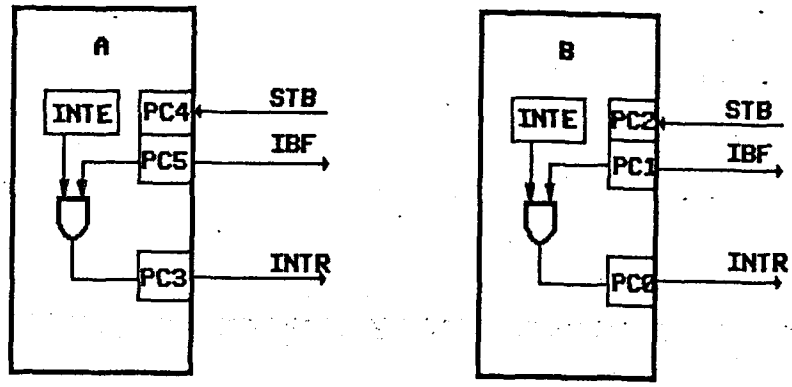
1 Modunun seçilmesi durumunda, A ve B iskelelerinin tüm kapıları ya alıcı ya da verici olarak koşullanabilir.

Tablo 3.3 0 Modda iskelelerin Alabileceği Durumlar

A		B		A KÜMESİ		B KÜMESİ	
D4	D3	D1	D0	A-isk	C üst	B-isk	C alt
0	0	0	0	Çıkış	Çıkış	Çıkış	Çıkış
0	0	0	1	Çıkış	Çıkış	Çıkış	Giriş
0	0	1	0	Çıkış	Çıkış	Giriş	Çıkış
0	0	1	1	Çıkış	Çıkış	Giriş	Giriş
0	1	0	0	Çıkış	Giriş	Çıkış	Çıkış
0	1	0	1	Çıkış	Giriş	Çıkış	Giriş
0	1	1	0	Çıkış	Giriş	Giriş	Çıkış
0	1	1	1	Çıkış	Giriş	Giriş	Giriş
1	0	0	0	Giriş	Çıkış	Çıkış	Çıkış
1	0	0	1	Giriş	Çıkış	Çıkış	Giriş
1	0	1	0	Giriş	Çıkış	Giriş	Çıkış
1	0	1	1	Giriş	Çıkış	Giriş	Giriş
1	1	0	0	Giriş	Giriş	Çıkış	Çıkış
1	1	0	1	Giriş	Giriş	Çıkış	Giriş
1	1	1	0	Giriş	Giriş	Giriş	Çıkış
1	1	1	1	Giriş	Giriş	Giriş	Giriş

Bu modda, C iskelesinin üst kısmı A ve alt kısmı B iskelesine el sıkışma işlemleri için destek verir. PC2, B iskelesi için hazır girişi olarak görev yaparken PC1 aynı iskele için al çıkışı olarak görev yapar. Hazır bilgisinin alınması ile kesme üretilmek isteniyorsa, bu da PC0 ile belirtilir. Bu Moddaki çalışma Şekil 3.14 de gösterilmiştir.

2. Mod sadece A kümesi, yani A iskelesi ve Cüst için geçerlidir. Bu moddaki çalışmada A iskelesi iki yönlü olarak kullanılabilmektedir.



Şekil 3.14 Mod 1 de Çalışmada El Sıkışma Ve Kesme

Sistemimizdeki I8255 PIA sı taban adresi $0300 olacak şekilde sistemimize yerleştirilmiştir. Kod çözücü devre, $0300-$0303 arası bir adres üretildiğinde ve bu adresin bir giriş çıkış birimine ait olduğunu belirten, mikroişlemcinin IO/MEM çıkışı etkin olduğunda birimi seçecek işareti üretmektedir. PIA koşullanırken tüm iskeleler verici olarak koşullanmış ve basit olan Mod 0 da çalışma benimsenmiştir. Bu modla ilgili gerekli bilgi yukarda verilmiştir. İskelelerden A ve B iki ayrı motoru sürmeye atanmıştır. İşlemleri kolaylaştırması açısından böyle bir yola gidilmesi yararlıdır. Sistem donanımı sadece bir PIA ve kod çözücü devreden oluştuğundan daha fazla açıklama yapılmayacaktır.

BÖLÜM 4. SİSTEM YAZILIMI

Tezin esas kısmı olan bu bölümde yazılan programın temel yapıları, özellikleri, karşılaşılan zorluklar ve ulaşılan noktalar betimlenecektir. Yazılan program, TURBO PASCAL 5.5 programlama dilinde ve grafik ortamda hazırlanmıştır. Kaynak kod olarak yaklaşık olarak 9000 satır olan bu program, 10 bölüm (UNIT) programı, 6 içerik (INCLUDE) programı ve bir ana programdan oluşmaktadır. Ana program gerek bölüm gerekse içerik programları arasındaki bağlantıyı sağlamakta ve sistemin temel programı olarak görev yapmaktadır. Diğer program parçaları ise kendilerine ait özel görevleri yapmaktadırlar. X ve Y yönündeki motorları kontrol eden ve bir önceki bölümde açıklanan yazılım da, yine ana program tarafından çağrılmakta ve böylece sistemin kontrolü tek bir program aracılığıyla sağlanmaktadır.[7]

4.1. Kullanılan Veri Yapıları

Sistem yazılımı, yapısı gereği birçok veriyi bünyesinde toplamaktadır. Bu verilerin gerek bellekte gerekse diskte belirli düzende saklanması gerekmektedir. Bu amaçla programda kullanılan veri yapıları, aşağıda verilecektir.

Programda, veri yapısı olarak mümkün olduğu kadar dinamik yapıların kullanılmasına özen gösterilmiş bu sayede hem performans hem de bellek kullanımı açısından iyi sonuçlar elde edilmiştir.

Veri yapılarına geçmeden önce statik ve dinamik değişkenlerin ne olduğu ve aralarındaki farkların neler olduğuna değinmekte yarar olacaktır. Statik değişkenler, programda kullanılmasalar dahi bellekte yer işgal ederler. Kullandığımız dizi, matris veya değişkenler aksi söylenmedikçe bu

türdendir. Derleyici, sembol tablosu oluştururken bu değişkenlere birer alan tahsis eder ve bu alan, sözedilen değişkenin ünlülüğü (Global) bitince iade edilir. Oysa, Pascal ve C gibi dillerde yer alan işaretçi tipindeki değişkenler dinamik yapıdadırlar ve belleği kullandıkları kadarı ile işgal ederler. Bu değişkenlerle, dizi ve matris yapıları da oluşturulabilir. Yeni bir değişken veya dizi yada matris elemanı, ancak kullanılacağı vakit bellekten istenir. Bu sayede, bellek israfından kaçınılınır. Bu yapılar, kullanıldıklarından dolayı ilerki bölümlerde incelenecek-lerdir.

4.1.1. Dinamik Liste Yapısı

Dinamik liste yapısı, statik olarak dizi yapısına karşılık gelir. Temel olarak, her bir eleman bir kayıt olarak tanımlanır ve her kayıt düğüm olarak adlandırılır. Düğümler, veri ve bağlantı alanlarından oluşurlar. Veri alanı her hangi bir tipten olabilir. Bağlantı alanı ise bir sonraki düğümün adresini içerir. Şayet, sözedilen kayıt, listemizin son kayıdı ise, bu düğümü başka bir düğümün takip etmediğini belirtmek üzere bağlantı alanına özel bir değer yazılması gereklidir. Pascal'da bu değer NIL olarak adlandırılır.[8]

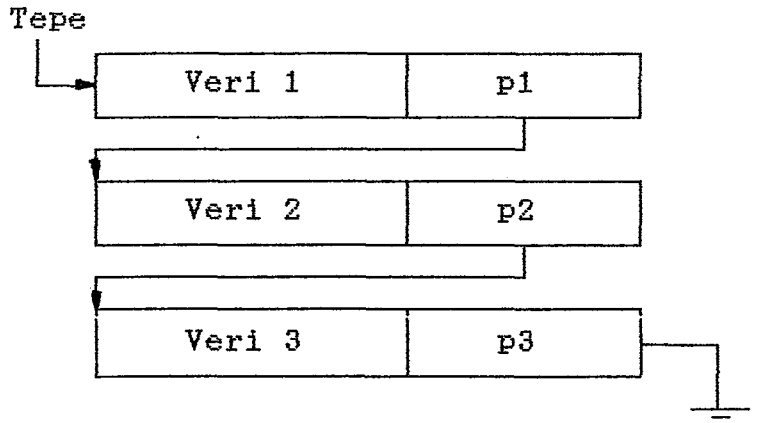
Dinamik liste yapılarında, listenin başını, yani listenin ilk düğümünün adresini içeren özel bir liste başı işaretçisi değişken olarak tanımlanmak zorundadır. Bu tanım yapılmadığı takdirde listemizi oluştursak dahi onun elemanlarına ulaşamayız.

Gerek liste yapılarında, gerek dinamik değişkenlerle ilgili diğer uygulamalarda bellekten, kullanıma açık belirli uzunlukta bir bellek alanı kullanımı istendiğinde bu bir komutla belirtilir. Pascal'da bu New komutudur. Bu komutun kullanım şekli NEW(işaretçi) şeklindedir. Parantez içindeki işaretçi değişkeni, değişken tanımlama bölümünde bir kayıdı veya bir değişkeni işaret eden Pointer olarak

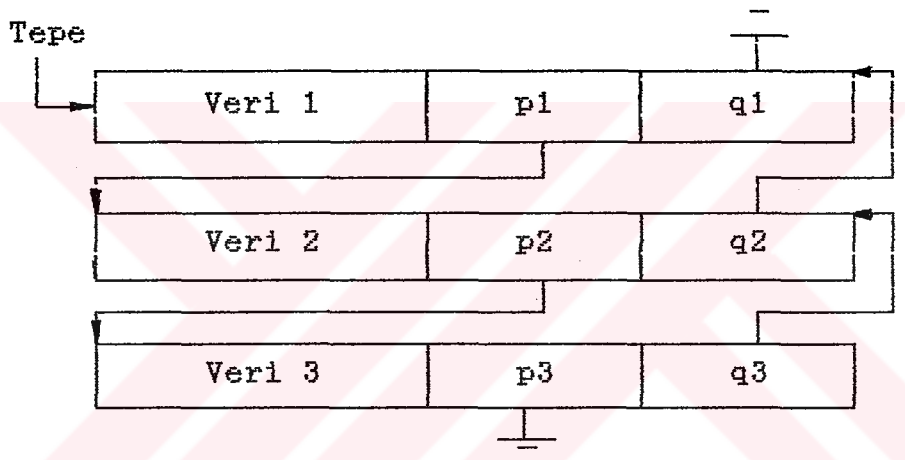
tanımlanmalıdır. Bu komut yürütüldüğünde, o anda bellekte kullanılmayan, ve istenen uzunlukta bir alanın başlangıç adresi işaretçi değişkenine atanarak geri döndürülür. Böylece, programcının bu alanı kullanması sağlanır. Yalnız, bu alanla ilgili işlemler doğrudan doğruya adres bilgisi ile yapılmayıp, fakat bu adresteki veriyle yapılacağından eşitliklerde, bu değişkenlerin adres olduklarını belirtmek üzere " ^ " imi kullanılır. Bu im görüldüğünde, işlemin belirtilen adresle değil adresteki veri ile yapılacağı anlaşılır. Benzer şekilde, şayet bir verinin adresi ile ilgili bir işlem yapılacaksa, o vakit de " @ " imi kullanılır. Bir düğümün ya da dinamik değişkenin kullanımı sona erdiğinde, bu verinin adresinin tekrar kullanılabilir alan olarak iade edilmesi gerekmektedir. Bu da Dispose komutu ile sağlanır. Kullanımı DISPOSE(işaretçi) şeklindedir.

Liste yapılarında, her düğüm bir sonraki düğümü işaret eden bir alana sahipti. Bu alana ek olarak, her düğüm bir önceki düğüme işaret eden bir alana daha sahip olabilir. Liste yapısının aldığı bu yeni şekle, çift bağlantılı liste adı verilir ve kayıt alanlarını büyüttüğünden, liste yapılarının çok büyümediği uygulamalarda kullanılmazlar. Ancak, tek bağlantılı liste yapılarında akışın ancak bir yönde olması nedeniyle çift bağlantılı listelerin kullanımının zorunlu olduğu haller de bulunabilir. Örneğin, yüz elemanlı bir tek bağlantılı listede, yüzüncü elemanda iken doksandokuzuncu elemana ulaşmamız için, doksandokuz atama işlemi yapmamız gerekirken, çift bağlantılı listelerde bir atama ile bu düğüme erişebiliriz. Şekil 4.1 ve 4.2 de tek ve çift bağlantılı liste yapıları gözükmektedir. Şekillerde, Tepe değişkeni listelerin başını göstermek için kullanılmaktadır. Elektriksel toprak, NIL değerini belirtir.

Sistem yazılımında, liste yapısı olarak tek bağlantılı yapı kullanılmıştır. Bu yapı, kullanıcının işlenecek parçayı tanımlaması sırasında ekrana çizdiği doğruları bellekte tutmak için kullanılmıştır. Tek bağlantılı listenin genel yapısı ve programda kullanılan şekli aşağıdadır.



Şekil 4.1 Tek Bağlantılı Liste Yapısı



Şekil 4.2 Çift Bağlantılı Liste Yapısı

Tek bağlantılı listedeki düğümler en genel şekli ile şu şekilde oluşturulabilir :

```
TYPE
  Liste_Düğümü      = Record
                        Veri      : Herhangi_Bir_Tip;
                        Bağlantı  : Liste_işaretçisi;
                      end;
  Liste_işaretçisi  = ^Liste_Düğümü;
```

Listemizin başını gösteren TEPE değişkeni Liste_işaretçisi tipinden tanımlanacaktır.

Çift bağlantılı liste yapısındaki bir düğümün alanları ise temel olarak aşağıdaki gibi tanımlanabilir :

```
TYPE
  Liste_Düğümü      = Record
                        Veri     : Herhangi_Bir_Tip;
                        Alt_Bağ  : Liste_işaretçisi;
                        Üst_Bağ  : Liste_işaretçisi;
                      end;
  Liste_işaretçisi = ^Liste_Düğümü;
```

Programda kullanılan tek bağlantılı liste yapısı ile ilgili işlemler, STRUCT2 isimli program bölümü tarafından yapılır. Bu bölümde, listeye düğüm ekleme, listeden düğüm çıkarma ve listeyi boşaltma işlemlerini yapan alt programlar bulunmaktadır. Düğüm temel yapı olarak şu şekildedir :

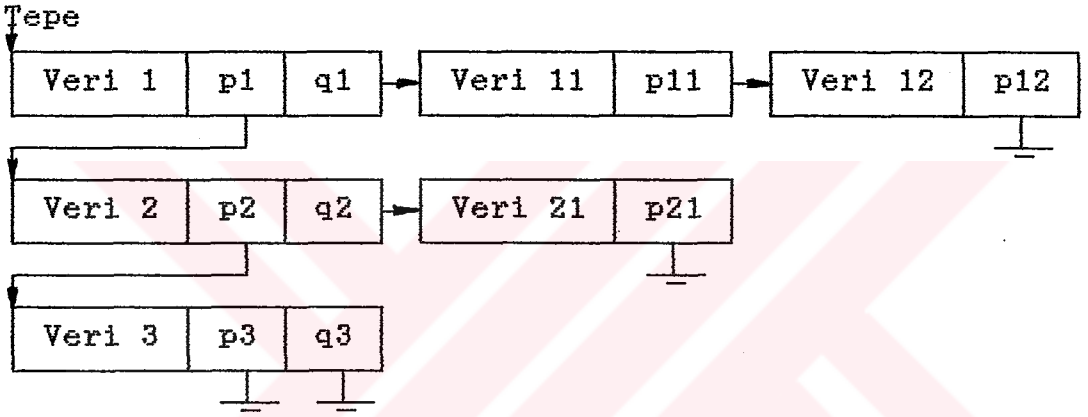
```
TYPE
  LineP       = ^LineType;
  LineType    = Record
                  Xs, Ys, Xe, Ye : Integer;
                  Dashed         : Boolean;
                  UpView         : Boolean;
                  Next           : LineP;
                end;
VAR
TopL, Last : LineP;
```

Her bir düğüm, kullanıcı tarafından çizilen doğrunun başlangıç ve bitiş noktalarının koordinatlarını içeren verilerin yanı sıra, çizilen doğrunun tipini ve hangi taraftan görünüş olarak çizildiğini belirten birer bayrak taşır. Bu veri alanları dışında, bir sonraki düğüme ulaşmamızı sağlayan bağlantı alanı da bu düğümde yer alır. Bu bölümde değişken olarak, listenin başına işaret eden TopL isimli işaretçi tipinden bir değişken ve listenin son düğümünün adresini içeren Last isimli işaretçi tipinden bir değişken yer almaktadır. Liste başı işaretçisinin gerekliliği daha önceden belirtilmişti. Son düğümün adresini içeren işaretçi ise, listeye yeni bir eleman ekleneceği zaman listenin tekrar taranmasını önler. Böylece hızdan kazanılmış olur. Bölümün program dökümü Ek A da verilmiştir.

4.1.2. Dinamik Matris Yapısı

Statik olan Matris yapısına karşılık düşen bu yapı, hem enine hem de boyuna genişleyen bir yapıdadır. Bağlantılar tek veya çift yönlü olabilmektedir. Bu tür yapılarda

veri ekleme ve çıkarma işlemleri güç olduğundan fazla kullanılmazlar. Ancak, arama süresini kısaltması açısından büyük bilgi bloklarının kullanıldığı uygulamalarda tercih edilirler. Genel yapı olarak Şekil 4.3 de verilen yapıda, boyuna genişleyen yapıdaki düğümler ile enine genişleyen yapıdaki düğümler farklı tipte olabilir. Yani, veri ve bağlantı alanları değişik tipten olabileceği gibi sayıları da farklı olabilir. Bu yapı, liste yapısının geliştirilmiş bir şekli olarak da düşünülebilir. Şöyle ki, liste yapısına, başka bir listenin başını işaret eden bir kayıt alanı eklenmesi bu yapıyı ortaya çıkarır.



Şekil 4.3 Dinamik Matris Yapısı

Açıklanan yapının genel tanımı şu şekilde yapılabilmektedir.

```
TYPE
  Satır_Düğümü      = Record
                        Veri       : Herhangi_Bir_Tip;
                        Alt_Bağ    : Satır_işaretçisi;
                        Sütun_Bağ  : Sütun_işaretçisi;
                      end;
  Sütun_Düğümü      = Record
                        Veri       : Herhangi_Bir_Tip;
                        Sağ_Bağ    : Sütun_işaretçisi;
                      end;
  Satır_işaretçisi = ^Satır_Düğümü;
  Sütun_işaretçisi = ^Sütun_Düğümü;
```

Yapının başını gösteren Tepe değişkeni, Satır_işaretçisi tipinden tanımlanmalıdır. Bu yapıyı, hem satırda hem de sütunda çift bağlantılı hale getirmek mümkündür. Bunun

için Satır_Düğümüne Üst_Bağ adlı ve Satır_işaretçisi tipinden bir alan, Sütun_Düğümüne de Sol_Bağ adlı Sütun_işaretçisi tipinden bir alanın ilave edilmesi yeterli olacaktır. Daha sonra, düğüm ekleme alt programında bu alanlar bir önceki ya da bir sonraki alanları işaret edecek şekilde düzenleneceklerdir.

Sistem yazılımında bu yapı, kullanıcı tarafından girilen koordinat bilgilerini, ve bu bilgilerle ilgili özellikleri taşımak için kullanılır. Her satır düğümü, girilen koordinatın çap bilgisini içerir ve bu çap bilgisine sahip düğümler o çapa ilişkin listeye, ana düğümün içerdiği liste başı işaretçisi sayesinde yerleşirler. Çaplara ilişkin ayrı ayrı listelerin tutulması, her çapa ilişkin düğümlerin kendi arasında değerlendirme görmesi nedeniyle bu uygulamaya uygun düşmektedir. Bu yapı, STRUCT isimli program bölümü tarafından oluşturulmakta ve yapı üzerindeki işlemler de yine bu bölüm tarafından yapılmaktadır. Bu program bölümünde yapı tipi şu şekilde verilmiştir :

```
TYPE
  MainNode     = Record
                   Data        : Real;
                   Point       : ExtNodePtr;
                   Next        : MainNodePtr;
                 end;
  ExtNode      = Record
                   X, Y, Z     : Real;
                   S, R, C, T  : Boolean;
                   Un,   No    : Byte;
                   Link        : ExtNodePtr;
                 end;
  MainNodePtr  = ^MainNode;
  ExtNodePtr   = ^ExtNode;

VAR
  Top          : MainNodePtr;
```

Bu yapıda, MainNode boyuna genişleyen yapıdaki bir düğümü tanımlarken, ExtNode ise enine genişleyen yapıdaki düğümü tanımlar. MainNode'daki kayıt alanları, Data, Point ve Next dir. Bu alanlardan Data, koordinata ilişkin çap bilgisini, Point, ilişkili listenin başını gösteren adresi, Next ise bir sonraki düğümün adresini içerir. ExtNode'da ise, kendi listesinin başını tutan düğümdeki çap bilgisini

paylaşan koordinat bilgileri ve bunların özellikleri yer alır. Bu bilgiler: X, Y, Z: delinecek deliğe ilişkin koordinat bilgileri, (bu bilgilerden z, yazılımın üç boyuta da genişletilebilmesi için yapıya konulmuştur) S: koordinata ilişkin soğutucu bilgisi, ki bu bilgi belirtilen koordinata delik delinecekken soğutucunun açılıp açılmayacağını gösterir, R, girilen koordinatın bir önceki koordinata göre bağıl mı yoksa mutlak mı olduğunu belirten bayrak, C, girilen koordinatın kartezyen ya da silindirik koordinat sistemine göre verildiğini belirten bayrak, T, girilen bilginin sadece xy bilgilerinimi yoksa xyz bilgilerinin hepsinimi içerdiğini gösteren bayrak, Un: girilen koordinatların hangi birime göre yapıldığını belirten sayı (mikron, inç, inçmetrik), No: verinin giriliş sırasını belirten sayı, ve Link: bir sonraki ExtNode u gösteren işaretçi alanıdır.

Bu veri yapısı üzerinde, STRUCT adlı program bölümünde şu işlemler yapılmaktadır : Yapıya, ExtNode tipinden bir düğüm eklemek (eğer bu düğümü eklerken bir MainNode oluşturmak gerekiyorsa bu düğümü oluşturmak), bir ExtNode'u yapıdan çıkarmak (eğer bu düğümü çıkarırken bir MainNode'un da listeden çıkarılması gerekiyorsa bunu da yapmak), bütün yapıyı ortadan kaldırmak, belirtilen sırada girilen bir düğümü bulmak ve bunu çağrıldığı programa döndürmek, ve bu yapıyı çap bilgisine göre küçükten büyüğe sıralamak. Bu bölüm programı dökümü EK A da verilmiştir.

4.1.3. Statik Yapılar

Programda, yukarda belirtilen dinamik yapıların yanı sıra bazı statik yapılar da kullanılmıştır. Genelde dizi yapısı tercih edilmiş ve dizinin her bir elemanı kayıt yapılarından oluşturulmuştur.

Bu yapılardan ilki, gerek derlemede gerekse çalıştırmada verilerin tutulduğu veri yapısıdır. Bu yapıda, x ve y de motorlara attırılacak adım sayılarının bulundurulduğu alanlar, matkap ucu çap ve soğutucu bilgisi yer alır.

Yapının son elemanı x, y ve r bilgisi olarak 0 değerini taşır. Bu yapı ilgili bölümünde şu şekilde tanımlanmıştır :

```
TYPE
  CompRec = Record
              Xcc      : LongInt;
              Ycc      : LongInt;
              Rcc      : LongInt;
              Cold     : Boolean;
            end;
  CompAr  = Array[0..255] of CompRec;
```

Bu yapıya benzer yapılar, delikler arasındaki minimum yolu bulan program bölümünde, parça tanımlamalarının yapıldığı ve parçalar içinde matkabın girebileceği alanları tutmak için kullanılmıştır. Bu programlar ve yapıları EK A da verilmiştir.

Kayıtlardan oluşan dizi yapıları dışında kullanılan bir yapı, kayıt yapısını tek başına kullanmaktır. Bu tür kullanım, farklı tipten değişkenlerin bir dosyaya yazılması ve bir bilgi bütünlüğü oluşturması için seçilir. Yazılımda kullanılan bu tip bir yapı aşağıda verilmiştir :

```
TYPE
  Flags    = Record
               CartCoor : Boolean;
               RelRead  : Boolean;
               TwoDim   : Boolean;
               Unt      : Byte;
               Mr       : Byte;
               FileN    : FName;
               OrgX     : LongInt;
               OrgY     : LongInt;
               DrillR   : LongInt;
               DrillH   : LongInt;
             end;
```

Kayıttaki alanlardan CartCoor, RelRead, TwoDim ve Unt, ExtNode daki kayıt alanlarına eşdeğer alanlardır. Mr : Farenin duyarlılığını belirleyen değişken, FileN : ilk değer dosyasının adı, OrgX, OrgY : merkez x ve y koordinatları, DrillR, DrillH : matkabın çap ve yükseklik bilgileri olarak kullanılmışlardır. Bu veri yapısındaki alanlar, sistemin temel değişkenleridir ve program ilk çalıştırıldığında VERİ.İLK adlı dosyanın varlığı araştırılarak bu değerler okunur. Dosyanın bulunamaması durumunda, bu değişkenler

program tarafından verilen ilk değerlerini alırlar. Bu, değişik veri yapılarından oluşmuş kayıdı bir dosyaya yazmak için, Flags tipinden tanımlanmış ve kayıt alanlarına gerekli değerler atanmış bir değişkeni, örneğin Data değişkenini, Write komutu ile Write(f, Data) şeklinde kullanmak yeterli olur. Bu komutta belirtilen f dosya değişkeni , değişken tanımlama kısmında file of flags, yani belirlenen tipin dosya değişkeni olarak tanımlanmalıdır.

Bu tip kullanıma bir başka örnek, kullanıcı tarafından çizilen bir parçanın dosya olarak saklanması sırasında görülmektedir. Bu program parçasının dökümü PARTDESC.PAS olarak EK A da verilmiştir.

Program bünyesinde daha birçok yapı bulunmasına karşılık, bu yapılar yukarda belirtilen yapıların bir alt kümesi veya benzeridir. Bu nedenle, benzeri yapılara ilerki bölümlerde önemleri ölçülerinde değinilecektir.

4.2. Algoritmalar

Bu başlık altında, programda kullanılan bazı algoritmalar incelenip, temel dayanakları verilecektir. Kullanılan algoritmalar genelde grafik kökenli olup, bir çok alanda karşımıza çıkabilecek türdendir. Alt başlıklarda, algoritmaların anlatımı dışında, bu problemlere getirilebilecek başka çözümler varsa bunlar da irdelenecektir.[9]

4.2.1. Doğru Algoritması

Bu algoritma, xy tablasının hareketini simüle etmek için kullanılır. Doğru çiziminde, döngü içine gecikme konularak hareket canlandırılmaya çalışılmıştır.

Doğru çizim algoritmaları çok çeşitlidir. Bunlardan en belli başlıları Bresenham ve DDA algoritmalarıdır. Yazılımda, DDA algoritması doğru çizimi için kullanılmıştır. Bu algoritma aşağıdaki sembolik şekli ile gösterilebilir :

```
Procedure Line(Xs, Ys, Xe, Ye);
begin
  if Abs(Xe-Xs)>=Abs(Ye-Ys) then Len:= Abs(Xe-Xs)
                            else Len:= Abs(Ye-Ys)
  endif
  DeltaX:= (Xe-Xs)/Len
  DeltaY:= (Ye-Ys)/Len
  X:= Xs+0.5*Sign(DeltaX)
  Y:= Ys+0.5*Sign(DeltaY)
  i:= 1
  While i<=Len do
    Plot(Integer(X), Integer(Y))
    X:= X+DeltaX
    Y:= Y+DeltaY
    i:= i+1
  End While
end;
```

Bu algoritmada, daha az hareket edilecek olan eksendeki hareket diğer eksendeki harekete bağlı olarak hesaplanır. Algoritma, döngü içinde hiçbir karşılaştırma işlemi içermediğinden oldukça hızlıdır. Algoritmaya ilişkin alt program, RUNSYS.INC dosyası içinde alt program olarak EK A da verilmiştir.

DDA algoritması yerine Bresenham algoritması da kullanılabilirdi. Bu algoritmanın yapısı aşağıda kısaca verilecek ve diğer algoritmalara geçilecektir.

```
Procedure Line(Xs, Ys, Xe, Ye);
begin
  X:= Xs
  Y:= Ys
  DeltaX:= Xe-Xs
  DeltaY:= Ye-Ys
  e:= DeltaY/DeltaX-0.5
  For i:=1 to DeltaX do
    Plot(X, Y)
    While e>=0 do
      y:= y+1
      e:= e-1
    End While
    x:= x+1
    e:= e+DeltaY/DeltaX
  Next i
end;
```

Bresenham algoritması, bu görünüşü ile DDA algoritmasına göre daha fazla işlem içerir ve daha yavaştır. Ancak, e sayısına ey:= 2e*DeltaX dönüşümü uygulayacak olursak, e sayısı tamsayıya dönüşür ve işlemler hızlanır.

4.2.2. Doğru-Alan Kesişim Algoritması

Bu algoritmada amaç, başlangıç ve bitiş noktalarının koordinatları verilmiş bir doğru ile, üst sol ve sağ alt köşelerinin koordinatları verilmiş bir alanın kesişip kesişmediğini belirlemek ve eğer kesişiyorsa kesişim noktalarını bulmaktır.

Bu algoritmanın yazılımda kullanılmasının temel sebebi, bir parça tanımlanmışken, izlenecek yörüngenin parça alanına girip girmeyeceğini belirlemek ve eğer bu alana giriliyorsa gerekenin yapılmasını sağlamaktır. Program, simgesel olarak aşağıda verilmiştir :

```
Procedure Kesişim (X1, Y1, X2, Y2: LongInt
                   Var Adet: Byte
                   Var Xs, Ys, Xe, Ye: LongInt)
begin
  Adet:= 0
  if (Y2=Y1).OR.(X2=X1) then
    if Y2=Y1 then
      if (Y1>=Ys).AND.(Y1<=Ye).AND.(X1<Xe).AND.(X2>Xs)
      if NOT((X1>Xs).AND.(X2<Xe) then
        Adet:= 1:
        Yq1:= Y1
        if X1>Xs Xq1:= Xe
        if X2<Xe Xq1:= Xs
        if (X1<=Xs).AND.(X2>=Xe) then
          Yq2:= Y1: Xq1:= Xs: Xq2:= Xe
          Adet:= 2
        End If
      End If
    End If
    if X2=X1 then
      if (X1>=Xs).AND.(X1<=Xe).AND.(Y1<Ye).AND.(Y2>Ys)
      if NOT((Y1>Ys).AND.(Y2<Ye)) then
        Adet:= 1
        Xq1:= X1
        if Y1>Ys Yq1:= Ye
        if Y2<Ye Yq1:= Ys
        if (Y1<=Ys).AND.(Y2>=Ye) then
          Xq2:= X1: Yq1:= Ys: Yq2:= Ye
          Adet:= 2
        End If
      End If
    End If
  End If
```

```
  Else
    m:=(Y2-Y1)/(X2-X1)
    c:= Y2-Round(m*X2)
    Yq1:= Round(m*Xs+c)
    Yq2:= Round(m*Xe+c)
    if(Yq1>=Ys).AND.(Yq1<=Ye).AND.(Yq1>=Y1).AND.(Yq1<=Y2)
    then
      Adet= Adet+1
      Xq1:=Round(1/m*(Yq1-c))
      Arr[Adet].X:= Xq1
      Arr[Adet].Y:= Yq1
    End If
    if(Yq2>=Ys).AND.(Yq2<=Ye).AND.(Yq2>=Y1).AND.(Yq2<=Y2)
    then
      Adet= Adet+1
      Xq2:= Round(1/m*(Yq2-c))
      Arr[Adet].X:= Xq2
      Arr[Adet].Y:= Yq2
    End If
    Xq1:= Round(1/m*(Ys-c))
    Xq2:= Round(1/m*(Ye-c))
    if(Xq1>=Xs).AND.(Xq1<=Xe).AND.(Xq1>=X1).AND.(Xq1<=X2)
    then
      Adet= Adet+1
      Yq1:= Round(m*Xq1+c)
      Arr[Adet].X:= Xq1
      Arr[Adet].Y:= Yq1
    End If
    if(Xq2>=Xs).AND.(Xq2<=Xe).AND.(Xq2>=X1).AND.(Xq2<=X2)
    then
      Adet= Adet+1: Yq2:= Round(m*Xq2+c)
      Arr[Adet].X:= Xq2
      Arr[Adet].Y:= Yq2
    End If
    for c:= 1 to Adet-1 do
    for y:= c+1 to Adet do
      if (Arr[c].X=Arr[y].X).AND.(Arr[c].Y=Arr[y].Y) then
        Arr[y].X:= 0: Arr[y].Y:= 0
      End If
    Next y
    Next c
    if Adet>0 then
      Xq1:= Arr[1].X
      Yq1:= Arr[1].Y
      y:= 2
      While y<=Adet do
        if Arr[y].X+Arr[y].Y<>0 then
          Xq2:= Arr[y].X
          Yq2:= Arr[y].Y
          y:= Adet: Adet:= 2
        End If
        y= y+1
      End While
      if Adet<>2 Adet:= 1
    End If
  End If
end
```

Yukarda verilen algoritmada, X1, Y1, X2, Y2: Doğrunun başlangıç ve bitiş noktalarının koordinatlarını, Xs, Ys, Xe, Ye ise, girişte alanın, çıkışta kesişim noktalarının koordinatlarını taşır. Adet, çıkış değişkeni ise kesişim nokta sayısını verir.

Algoritma ilk olarak doğrunun denklemini çıkarır. Bu amaçla iki noktaya ilişkin değerler kontrol edilerek, doğrunun eğiminin sıfır ya da sonsuzdan farklı değerleri için y=mx+c formundaki m ve c sabitleri elde edilir. m nin sıfır ve sonsuz olması durumları için özel inceleme yapılır. Bu koşullar dışında doğru denklemi elde edilir ve bu denkleme alanın başlangıç ve son noktalarının x değerleri konularak elde edilen y lerin doğrunun üzerinde olup olmadığı kontrol edilir. Şayet denklem sağlanıyorsa verilen x ve y değerlerinde kesişme vardır. Bu işlem tamamlandıktan sonra x=1/m*(y-c) formülüne göre alanın başlangıç ve son noktalarının y değerleri için aynı işlem tekrarlanır. Bu iki aşama sonunda elde edilen noktaların aynı olanları listeden çıkarılarak kesişim noktaları ve adedi elde edilir.

4.2.3. Minimum Yol Algoritması

Bu algoritma, kullanıcı tarafından girilen aynı türden delikler içindeki en kısa yolu bulmak için kullanılır. Algoritma, graf teorisinde çok önemli bir yer tutan minimum maliyetli kapsayan ağaç (Minimum Cost Spanning Tree) bulunmasını sağlayan Kruskal Algoritması kullanılarak geliştirilmiştir. Bu alt başlıkta önce Kruskal daha sonra da minimum yol algoritması incelenecektir.

Kruskal algoritması, v düğüme, e ayrıta sahip bir grafta, minimum maliyete sahip ve v düğümlü ei<=e ayrıtlı bir alt graf bulma yöntemini verir. Bu algoritmada temel yöntem, tüm kenarların maliyetlerine göre sıralanması ve daha sonra bu kenarların çevre oluşturmayacak şekilde sıra ile alt grafa alınmasıdır. Bu işlem, tüm düğümler alt

grafta yer alana dek sürdürülür. Algoritmada, alınan kenarların bir çevre oluşturup oluşturmadığını kontrol etmek için bir bağlantı dizisi kullanılmış, kenar maliyetlerine göre sıralama ise QuickSort yöntemi ile yapılmıştır. Bu algoritma simgesel olarak aşağıda verilmiştir.

```
Procedure Kruskal;
begin
  Ayrıt_Dizi:= 0
  For i=1 to e do
    DiziyeEkle(Ayrıtı)
  Next i
  Sırala(Ayrıt_Dizi)
  Düğ_Adet:= 1
  While Düğ_Adet<e do
    if ÇevreOluşturmaz(Ayrıt_Dizi[Düğ_Adet])
    AltGrafaEkle(Ayrıt_Dizi[Düğ_Adet])
    Düğ_Adet:= Düğ_Adet+1
  End While
end;
```

Bu algoritmanın incelenmesinden sonra, programda nasıl kullanıldığını inceleyelim. Kullanıcı tarafından girilen koordinat bilgilerinin, delinecek delik çaplarına göre bir matris liste yapısında bulunduğunu belirtmiştik. Bu yapıda, satır listesindeki elemanlar, kendi aralarında bir graf olarak incelenir ve daha sonra diğer satır listelerinin oluşturduğu graflarla bağlantılar sağlanır. Her bir satır listesi, tam graf oluşturacak şekilde düzenlenir, yani her noktadan diğer noktalara bir ayrıt olduğu düşünülür, ve oluşturulan bu graf Kruskal algoritması tarafından değerlendirilerek bir alt graf oluşturulur. Ayrıtlar incelenirken ayrıtın maliyeti olarak düğümleri arasındaki uzaklık alınır ve buna göre inceleme yapılırsa elde edilen alt graf düğümler arasındaki minimum yolu verir. Bu alt grafın son düğümü, bir sonraki satır listesinden oluşturulacak alt grafın ilk düğümünü belirler. Son düğümle arasında minimum uzaklık bulunan düğüm, yeni grafın ilk düğümü olarak alınır. Programın dökümü EK A da verilmiştir.

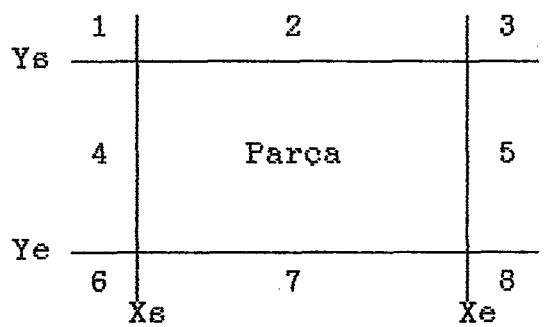
Minimum yol algoritması kullanıldıktan sonra elde edilen graf üzerinde de bazı değişiklikler yapılır. Çünkü, elde edilen graf bazı durumlarda minimum yolu vermemekte ve bu nedenle değişikliğe ihtiyaç duyulmaktadır.

4.2.4. Parçaya Göre Hareket Algoritması

Programda, kullanıcı tarafından bir parça tanımlanmadıkça bir problem söz konusu değildir. Ancak, belirlenen koordinatlar arasına hareketi engelleyecek bir parça konulursa, çizilecek yörüngenin düzenlenmesi gerekmektedir. Bu amaçla programda geliştirilen algoritma bu alt başlıkta açıklanacaktır.

Bu problem, algoritmalar arasında en çok incelenenlerden biridir. Yöntemde, hareketli cisimi noktasal hale getirmek ilk aşamayı oluşturur. Sistemimizde, tablayı sabit kabul edip matkabı hareketli varsayarsak, matkabın çapını, engele eklemek ilk aşamamızı oluşturur. Bunun sonucunda, tanımlanan parça ya da engelimiz boy ve en olarak matkap çapı kadar büyüyecektir. Bu ilk aşamadan sonra iki koordinat arasında çizeceğimiz yörüngenin parça sınırına girip girmediği, yukarda açıklanan alan doğru kesişim algoritması ile incelenir. Eğer kesişme varsa, aşağıda açıklanan yönteme başvurulur.

İlk olarak, parçanın etrafı Şekil 4.4 te gösterildiği gibi bölgelere ayrılmıştır. Başlangıç ve varış noktalarına



Şekil 4.4 Parçanın Etrafındaki Bölgeler

göre koordinatlar bu bölgelerde yer işgal ederler. Başlangıç ve bitiş noktalarının işgal ettiği bölgelere göre de koordinat listesine yeni bir düğüm eklenir ve bu işlem iki nokta arasında engel kalmayana kadar devam eder. Bölgeler

arasındaki geçişe göre eklenecek koordinat düğümleri Tablo 4.1 de verilmiştir.

Tablo 4.1 Bölge Geçişlerine Göre Eklenecek Koordinatlar

1.Bölge	2.Bölge	Eklenen Nokta
1	5	XeYs
	7	XsYe
	8	XeYs
2	4	XsYs
	5	XeYs
	6	XsYs
	7	XsYs V XeYs
	8	XeYs
3	4	XsYs
	6	XsYs
	7	XeYe
4	5	XsYe V XsYs
	7	XsYe
	8	XsYe
5	4	XeYs V XeYe
	6	XeYe
	7	XeYe

Bu kontrol noktaları listeye eklenirken parçanın koordinat sisteminde herhangi bir eksende sınıra yerleştirilmesi kontrolü de yapılır. Örneğin, parça x ekseninde sıfıra yerleştirilmişse, eklenecek noktanın x koordinatı xs olsa dahi bu ekleme yapılmaz ve eklenecek kontrol noktasının x'i daima xe alınır. Kontrol noktaları, çap bilgisi sıfır verilerek belirtilir. Bu noktalarda matkap beklemez.

Koordinat listesinin sonu x, y ve yarıçap bilgilerinin toplu olarak sıfır verilmesiyle belirlendiğinden, listeye istenildiği kadar nokta eklenebilmektedir. Algoritma basit olarak aşağıdaki şekilde gösterilebilir:

```
Procedure Engel;
begin
  Koor_Sayıcı:= 1
  While NOT Son(NoktaDizisi[Koor_Sayıcı]) do
    Kesişim(NoktaDizisi[Koor_Sayıcı],
            NoktaDizisi[Koor_Sayıcı+1],
            Parça, KesişimNoktası);
    If KesişimNoktası<>0 then
      DiziKaydır(Koor_Sayıcı)
      NoktaEkle(1.Bölge, 2.Bölge, Koor_Sayıcı)
      Koor_Sayıcı:= Koor_Sayıcı-1
    End If
    Ekle(NoktaDizisi[Koor_Sayıcı],
         NoktaDizisi[Koor_Sayıcı+1])
    Koor_Sayıcı:= Koor_Sayıcı+1
  End While
end;
```

Algoritmadan da görüldüğü gibi rekürsif bir yapı kurulmuştur. Şayet, bir kontrol ara noktası ekleniyorsa yeni bir bölge durumu oluşmakta ve belki de bu yeni durum bir başka kontrol noktasının listeye eklenmesini sağlamaktadır.

Bu yöntem oldukça hızlı bir yöntemdir. Ancak bu yöntem yerine uygulamalarda, graf teorisinden yararlanılarak üretilen Kaynak-Kuyu arası minimum yol algoritması kullanılır. Bu algoritmada, kaynak ve varış noktaları arasında üretilebilecek tüm yol kombinasyonları ve maliyetleri incelenerek bir sonuca varılır. Ancak, parçaya bağlı olarak gidilebilecek yolların bulunmasında problemlerle karşılaşılır. Bu uygulamada önerilen yöntem daha hızlı sonuç vermekte ve daha basit bir yapı kullanmaktadır.

Verilen bu algoritmalar dışında daha birçok algoritma kullanılmıştır. Bu algoritmalar, konuya özel olduğundan burada değinilmeyecek sadece EK A da program dökümü olarak verilecektir. Algoritmalar içinde, Kabarcık sıralama (Bubble Sort), hızlı sıralama(Quick Sort), parçanın tanımlanması sırasında oluşturulan doğru listesinden, parçanın

boyutlandırılacak değerlerinin bulunması yöntemi de yer almaktadır.

4.3. Program Özellikleri

Bu başlık altında programın yeteneklerine değinilecek fakat bu yeteneklerin nasıl kazandırıldığından, daha önceki bölümlerde incelendiğinden dolayı, sözedilmeyecektir.

Program, kullanıcı ile bir matkabın altına yerleştirilmiş xy tablası arasında arayüz oluşturmayı amaçlamaktadır. Fare desteği verilerek, kullanıcının istediklerini daha kolay ve daha hızlı yapması sağlanmakta ve bir önceki menü ekranda bulundurularak, her an programın hangi aşamasında bulunulduğu gözlenmektedir. Sistemdeki matkabın deleceği deliklerin koordinatları kullanıcıdan istenmekte ve buna bağlı olarak da, matkabın altında bulunan xy tablası hareket ettirilerek istenen koordinatlara konumlanma sağlanmaktadır.

Programda, farenin sol tuşu, Enter tuşu yada gerekli seçeneğin baş harfinin tuşu o menüye geçmemizi sağlarken, farenin ikinci tuşu ve Esc tuşu iptal yada bir önceki menü anlamında kullanılır. Sistemde yapılacak önemli işler, her seferinde kullanıcının onayı alınarak yapılır ve böylece kullanıcının dikkatsizce yapacağı hatalar minimuma indirilir. Yazılımda, bulunulan her alt menüde en son kullanılan seçenekler aktif kalır. Örneğin koordinat alt menüsünde en son gözlem seçeneği kullanıldıysa, bu alt menüye bir daha geçişte aktif seçenek olarak gözlem seçeneği görülür. Böylece sık kullanılan seçenekler için zaman kaybedilmez ve kolayca istenilen seçeneğe ulaşılır.

Program bir ana ve altı alt menüden oluşmaktadır. Ana menüde, Koordinat, parça, çalıştırma, ilkdeğer, seçenekler ve çıkış seçenekleri bulunmakta ve yeni alt menüler açılmasına neden olmaktadır. Bu alt menüler ve sağladıkları bundan sonraki alt başlıklarda incelenecektir.

4.3.1. Koordinat Alt Menüsü

Ana menüden bu seçenek seçildiğinde yeni bir alt menü açılır ve ekrana yeni seçenekler gelir. Temel olarak, koordinat bilgilerinin girilmesini ve bu bilgilerle ilgili bazı işlemleri içerir. Bu alt menüde şu seçenekler ve özellikler bulunmaktadır:

4.3.1.1. Veri Giriş

Bu alt seçeneğe geçildiğinde, ekranın altında koordinat verilerini sağlayacak pencereden bilgi girişi beklenir. Eğer ilk kez bilgi girilecekse, girilecek kayıt no olarak bir gözükürken, daha önceden bilgi girilmişse bu bilgilerin devamı niteliğinde kayıt okunur. Kayıtlar okunurken koordinat bilgilerinin genel şekli Seçenekler alt menüsünde belirlendiği şekildedir. Koordinatlar, bir önceki koordinata bağıl olarak verilebileceği gibi mutlak olarak da verilebilir. Yazılımda, üç boyutlu kontrol uygulamalarına da destek vermek amacıyla iki ya da üç boyutlu veri girişi sağlanmıştır. Veriler, kartezyen ya da silindirik koordinatlarda verilebilmekte, metrik, inç veya inçmetrik birimlerinde olabilmektedir. Tüm bu özellikler, Seçenekler alt menüsünde belirlenmektedir.

Bu alt seçenekte, kullanıcıya, koordinat bilgileri girilirken, özellikle bağıl veri girişlerinde kolaylık olması açısından bir önceki kayıt verilir ve kullanıcı isterse bu kayıdı Enter ya da farenin birinci tuşu ile yeni kayıt olarak belirleyebilir. Bu bölümde bazı tuşlara belirli görevler verilmiştir. Bu tuşlar ve görevleri aşağıda belirtilmiştir.

Aşağı Ok Tuşu: Bir önceki kayıdı aktif kayıt yapar ve bu kayıtı değiştirme imkanı sağlar.

Yukarı Ok Tuşu: Bir sonraki kayıdı aktif kayıt yapar. Bu kayıdı değiştirmemiz mümkündür.

GeriBoşluk Tuşu: Koordinat girişi sırasında en son karakteri silmemizi sağlar.

Sol Ok Tuşu: O anda aktif bulunan kayıdı yeniden girmemizi sağlar.

Alt_D Tuşu: O anki aktif kayıtla ilgili durum bilgilerini içeren bir pencerenin açılıp açılmayacağını belirler. Koordinat alt menüsünden çıkılıp geri dönülmesi sırasında değer değiştirmez.

Alt_V: O anki aktif kayıdın veri giriş şeklini değiştirir. Bir sonraki ya da bir önceki kayıdı etkilemez.

Alt_B: Aktif kayıdın, çap bilgisi hariç, hangi birim sisteminde girildiğini belirler. Sadece atandığı kayıdı etkiler.

Alt_Y: Tüm koordinat bilgilerinin iptal edilmesine ve yeni kayıt girişine neden olur. Tüm kayıtlar silinmeden kullanıcıya bir uyarı mesajı ile emin olup olmadığı sorulur.

Alt_S: O anki aktif kayıdın silinmesine yol açar. Aktif kayıt silinmeden kullanıcıya uyarı mesajı verilir.

Alt_O: Belirlenen aktif kayıda ilişkin delik delinirken soğutucunun açık olup olmayacağını belirler. Değiştirilmediği sürece, her kayıtta aynı durumda yer alır.

Alt_R: Aktif kayıdın çap bilgisini güncellemede kullanılır. Bu tuşla değiştirilmedikçe, her kayıtta aynı değerde yer alır. Sayının yanına getirilen `'` imi girilen çap bilgisinin inç, `"` imi ise inçmetrik biriminde olduğunu gösterir ve anında metrik birime çevrilir. Çap bilgisi olarak girilen sıfır ise belirtilen koordinatın bir kontrol noktası olduğunu ve bu koordinata konumlanılacağını fakat delik delinmeyeceğini belirtir.

Yukarda belirtilen tuşlar, kayıdın herhangi bir koordinatına veri girişi yapılmadan önce işlerliktedir. Örneğin bir kayıdın y koordinatının girilmesi başlamadan bu tuşlar görev yaparken, birinci basamağın girilmesi ile bir sonraki koordinat girişine kadar işlerlikten kalkar.

Aşağı yada yukarı ok tuşlarına basıldığında, yeni kayıdın getirilmesi, her seferinde yapının baştan itibaren taranması ile yapılmaktadır. Yapıda n nolu kayıdın yeri

belli olmadığından her kayıdın sahip olduğu ve giriş sırasını belirtilen alana bakılarak gerekli kayıt bulunur.

4.3.1.2. Gözlem

Bu alt başlık, girilen koordinat verilerinin yerini ve boyutunu görmek amacıyla kullanılır. Bu alt seçenek onaylandığında ilk olarak, aktif koordinat verisi olup olmadığı incelenir. Şayet gözlenebilecek herhangi bir koordinat verisi bulunmuyorsa, bir mesaj verilerek üst menüye dönülür. Koordinat verilerinin bulunması durumunda ise ölçekli olarak delikler bir tabla üzerinde belirlenir. Bu tabla üzerindeki herhangi bir alan Büyüt seçeneğinin onaylanması ile daha ayrıntılı olarak görülebilir. Bu seçenek seçilince fare kalağı şekil değiştirir ve bir alanı belirleyene dek bu şeklinde kalır. Alan belirlendikten sonra, bu alan ekranla aynı ölçeğe gelecek şekilde yeniden düzenlenir ve büyütülmüş şekli, değiştirilmiş koordinat eksen bilgileri ile birlikte görüntülenir. Herhangi bir tuşa basılması ile eski görüntüye dönülür. Bu alt seçenek, girilen koordinat bilgilerinin doğruluğunun belirlenmesi açısından önem taşır.

4.3.1.3. Yükle

Koordinat bilgilerinden oluşmuş bir veri bloğunun yazılı bulunduğu ortamdan okunması için kullanılır. Kullanıcıdan okunan dosya ismi, katalogda aranır ve bulunması durumunda belleğe aktif koordinat listesi olarak yüklenir. Yükleme sırasında bellekte aktif kayıtlar bulunmaktaysa bu kayıtlar silineceğinden kullanıcı uyarılır ve yükleme yapılıp yapılmayacağı tekrar sorulur. Belirlenen dosyanın bulunamaması durumunda gerekli hata mesajları verilir. Dosya adı olarak * veya ? dosya adlarının araştırılması amacı ile verilebilir. Bu iki im DOS daki görevleri ile kullanılırlar. Açılan pencerede belirlenen düzendeki dosyalar görüntülenir ve istenen dosya üzerine gidilerek yüklenebilir

4.3.1.4. Sakla

Bu seçenek, bellekte yer alan koordinat bilgilerine ait kayıtların kalıcı bir ortamda (diskte ya da diskette) saklanması amacı ile kullanılır. Okunan dosya adının DOS daki dosya adı yazım kurallarına uyup uymadığı, bir önceki seçenekte olduğu gibi kontrol edilir ve doğru bir düzende dosya adı verilene kadar dosya adı sorulur. Daha sonra, dosyanın yazılacağı ortamda aynı adda bir dosyanın olup olmadığı araştırılarak eski dosyanın kaybolması engellenir. Kullanıcı eğer dosyayı aynı adda kaydetmek istiyorsa kayıt işlemi yapılır. Ortamda kayıdı kaydedecek yer kalmaması durumu da göze alınarak gerekli kontrol yapılmıştır.

4.3.1.5. Rapor

Bellekteki kayıtlarla ilgili bilgileri ekrana ya da yazıcıya gönderir. Bellekte kayıt olmaması durumunda kullanıcıya hata mesajı vererek bir üst menüye döner. Bu alt seçeneğin onaylanması ile raporun ekrana mı yoksa yazıcıya mı yapılacağı sorulur. Şayet, onay tuşlarından birine basılmışsa, rapor ekrana yazılır. Sadece "Y" tuşuna basılması yazıcıya raporu aktarır. Raporda koordinat bilgileri ve bu bilgilere ait ek özellikler yer alır. (Soğutucu, birim, veri giriş şekli, koordinat sistemi vb.) Rapor ekrana çıkarılacaksa, rapor ekranının boyutu kayıt sayısı ile ilişkili olur. Şayet sekizden fazla kayıt varsa, rapor ekranı en büyük alanına ulaşır ve kayıtlar sayfa sayfa verilir. Yazıcıdan rapor alınacaksa, yazıcıya ilişkin tüm kontroller yapılır. Sistemde yazıcı olup olmadığı, yazıcının açık olup olmadığı, yazıcıda kağıt olup olmaması ya da hatta (OnLine) olup olmadığı kontrol edilir ve eksiklikleri durumunda ilişkili hata mesajları verilir. Yazıcıdan alınan rapor ile ekranda görüntülenen rapor içerik olarak aynı olmasına karşılık düzen olarak birbirinden farklıdır. Bu nedenle, ekrandaki raporun yazıcıdan alınması beklenmemelidir.

4.3.2. Parça Alt Menüsü

Bu alt menü, temel işlev olarak işlenecek parçanın özelliklerinin tanıtıldığı bölümdür. Kullanılması mecburi değildir. Yani, sistemin çalışması bu alt menüde bir parça tanımlanmasına gerek duymaz. Ancak, işlenecek parça matkaba etki edecek boyutlara ya da engellere sahipse, parça bu alt başlıkta tanımlanabilir. Bu tür kritik parçalar tanımlanmasa dahi eklenen kontrol noktaları ve yol tarifi ile tanımlanmadan da kullanılabilirler. Tanımlanan parçaların oyuklarının homojen olduğu kabul edilir. Yani parçada yer alan bir oyuğun, yer aldığı eksen boyunca sürdüğü kabul edilmektedir. Parçada, bir eksen boyunca değişik yüksekliklerde oyuklar varsa bunlardan en yükseği oyuk olarak tanıtılmalıdır. Çünkü, bu tanımlamadan amaç matkabın girebileceği alanların saptanmasıdır. Bu nedenle parça mümkün olduğu kadar sade ve yüksek toleranslarda boyutlandırılmalıdır. Matkabın çapı olarak bir adet veri girildiğinden ve bu değer de matkabın belli bir parça oyuğuna girip giremeyeceğini belirlediğinden bu değer de dikkatli seçilmelidir.

Alt menü dört seçenekten oluşmaktadır. Bu seçenekler, önden görünüş, yandan görünüş, yükle, ve sakladır. Seçeneklerden önden ve yandan görünüş parçanın temel olarak tanımlandığı bölüm, yükle ve sakla ise bu bilgilerin daha sonraları kullanılmasını sağlayan bölümlerdir. Sakla ve yükle, genel olarak koordinat alt menüsünde yer alan kontrolleri yaptığından ve tek fark olarak parça bilgisi üzerinde çalıştığından bu konuda açıklama verilmeyecektir. Bilinmesi gereken şudur ki, buradaki sakla ve yükle seçeneklerinde parçanın temel çizim bilgileri dışında boyutlandırılma bilgileri de yer alır ve bu sayede, boyutlandırılarak kaydedilmiş bir parçanın tekrar boyutlandırılması gerekmez. Parçanın önden ve yandan görünüşlerinde aynı işlemler yapıldığından, sadece parça tanımlamak için neler yapılacağı ve hangi aşamalardan geçmek gerektiği belirtilecektir.

Parça tanımlama ile ilgili bir alt bölüme geçildiğinde yeni bir pencere açılır ve bu pencere içinde gridler yer alır. Bu gridlerin yoğunluğu Seçenekler alt menüsünde yer alan fare duyarlılığı seçeneği ile doğrudan ilgilidir. Seçenekler alt menüsündeki fare duyarlılığı arttırılırsa bu gridlerin sayısı artacaktır. Kullanıcının parçayı daha rahat çizebilmesi için Snap To Grid adı verilen ve sadece gridler arasında hareketi mümkün kılan bir yöntem kullanılmıştır. Bu şekilde doğrular daha rahat çizilebilmektedir. Kullanıcı, parça tanımlarken ilk olarak parçanın teknik çizimini yapar. Bunun için, parçanın o cepheden görülen parçası kaba olarak çizilir. Çizimde iki tip doğru tipi kullanılır. Kesikli çizgiler, arkada kalan ve gözükmeyen kısımlar için kullanılırken, düz çizgiler görülen kenarları belirtmede kullanılır. Kesik çizgiler yazılımın üç boyutlu uygulamaları desteklemesi amacı ile kullanılmıştır. Kullanıcı isterse çizmiş olduğu doğruları Silme seçeneğini kullanarak yok edebilir ve bu şekilde istenilen şekiller oluşturulur.

İstenen görünüş oluşturulduktan sonra, program karmaşık bir algoritma sonucunda parçaya ilişkin önemli uzunluk bilgilerini tesbit eder. Kullanıcı boyutlandırma seçeneğini onayladığında, bu bilgiler istenir ve değerlendirilir. Hem önden hem yandan görünüşte okunacak bilgiler benzerlik gösterir. Yalnız, parçanın yerleştirileceği x ve y koordinatları bir kere ve önden görünüş sırasında okunur. Bu nedenle, önden görünüş boyutlandırılmadan, yandan görünüş boyutlandırılamaz. Parçanın tam olarak tanımlanması için her iki görünüşün de boyutlandırılması gerekmektedir. Boyutlandırma sonunda parçanın hangi bölgelerinin matkap tarafından delinebileceği, hangi bölgelerinin yasak bölge olduğu bulunur ve derleme aşamasında bu bilgiler koordinat bilgileri ile birlikte değerlendirilir.

O anda aktif olan parçanın yerine yeni bir parça çizmek için parça yükleme alt menüsüne geçilip, aktif parçanın kaybolması sağlanır ve sonra yüklemeden vazgeçilir.

4.3.3. Çalıştırma Alt Menüsü

Bu alt menü, gerekli bilgilerinin girilmesinden sonra geçilecek bir menüdür. Bu menüye geçilebilmesi için en azından koordinat bilgilerinin bellekte yer alması gerekir. Aksi takdirde, bu menünün herhangi bir seçeneğine geçiş mümkün olmaz. Bu alt menüde, girilen bilgilerin uyumu kontrol edilir, bilgiler derlenerek x ve y yönündeki motorlara attırılacak adım miktarları hesaplanır ve motorlara uygulanır.

Alt menüde, beş seçenek yer alır. Bunlar, derle, prova, çalıştır, yükle, sakla. Bu alt seçeneklerden yükle ve sakla, diğer alt menülerde yer alan yükleme ve saklama seçeneklerinde de yer alan hata kontrollerini yapar ve oluşturulan çalıştırma dosyalarını belleğe ya da diske alır. Burada diğer seçenekler incelenecektir.

4.3.3.1. Derle

Bu alt başlık seçildiğinde, yeni bir pencere açılır ve En Kısa, Tarif, Otomatik şeklinde üç şık belirir. Bu seçeneklerden, aksi söylenmedikçe otomatik seçeneği kabul edilir. Diğer seçeneklere sağ, sol ok tuşları ya da baş harfleri yardımıyla erişilebilir. Bu alt menüde, girilen koordinat ve varsa parça bilgileri düzen ve sınır açısından kontrol edilir. Tablanın sınırlarını aşan değerlerde hata verilir. İşlenemeyecek parça bölgelerine ait koordinat bilgileri varsa bu noktalarda da hata oluşur. Hatasız bir derleme sonunda motorlara attırılacak adım sayıları elde edilir.

Derleme seçeneklerinden otomatik, koordinatların girilen sırada değerlendirilerek bir yörünge çizilmesine neden olur. Parça tanımlanmışsa, bu parçaya bağlı olarak yörünge ayarlaması yapılır. Tarif alt seçeneği, koordinatlar arasındaki geçişi kullanıcının belirlemesine olanak verir. Deliklerin delinis sırası kullanıcıya sorulur ve

buna bağlı olarak bir yörünge takip edilir. En Kısa seçeneğinde ise delikler arasındaki en kısa yol izlenir. Son iki seçenekte de parça tanımlı olması halinde yörünge tekrar düzenlenir.

Derleme sırasında, kayıtların yüzde kaçının derlendiği ekrandan izlenebilir. Hatanın kaçıncı kayıtta olduğu ve cinsi de kullanıcıya bildirilir.

4.3.3.2. Prova

Bu seçenek, başarılı bir derleme sonunda veya bir çalıştırma dosyasının yüklenmesi sonucunda yürütülebilir. Oluşturulan ve motorlara uygulanacak sonuçların son bir defa kontrolü amacıyla kullanılır. Bu seçenekte xy tablasının izleyeceği yörünge belirtilir. Şayet bir parça tanımlı ise, bu parça da tablada yerleştirildiği noktada görüntülenir. Ekran prova sonunda temizlenmez. Şayet, ekranın temizlenmesi isteniyorsa ana menüde iken CTRL_S tuşuna basılır. Bundan sonraki alt menü olan Çalıştırma alt menüsünde ise motorlara işaretler uygulanır. Bu alt menüde ALT_T tuşu ile tekrarlı çalışma sağlanabilir.

4.3.4. İlk Değer Alt Menüsü

Bu alt menü, sistem tarafından önemli olan bazı büyüklükleri içerir ve bu değerlerin yeniden belirlenebilmesini sağlar. Bu değerler, matkap çapı, matkap yüksekliği ve merkezin x, y koordinatlarıdır. Girilen değerler, seçenekler alt menüsünde "VERİ.İLK" dosyasına kayıt yapılmadığı sürece yereldir ve program durdurulup yeniden çalıştırıldığında değişir.

4.3.5. Seçenekler Alt Menüsü

Sistemle ilgili önemli parametreleri taşır. "VERİ.İLK" isimli dosyaya kayıt edilmediği sürece, programın o anki çalışması sırasında geçerlidir. Şu büyüklükleri içerir:

Fare Duyarlılığı: Farenin program içindeki hareket duyarlılığını belirler. 0 ile 50 arasında bir sayıdır. Sayı arttırıldıkça duyarlılık artar.

Veri Giriş Şekli: Bağıl ya da mutlak veri girişinden hangisinin seçildiğini belirler.

Boyut Seçimi: İki veya üç boyut olarak seçilebilir ve koordinat veri girişleri bu değere göre yapılır.

Koordinat Sistemi: Kartezyen ya da silindirik koordinatlarda veri girişi yapılabilir.

Birim Seçimi: Metrik, inç, inçmetrik çalışılabilir.

Yükle veya Sakla: Yukarda belirtilen büyüklükler bir dosyada saklanabilir veya saklanmış bir bilgi dosyası yüklenebilir. Saklama ve yükleme sırasında, ilk değer alt menüsünde yer alan bilgiler de saklanır ve yüklenir.

4.3.6. Çıkış

Programdan çıkışı sağlar. Ana menüde iken "X" tuşuna basmak ta aynı görevi yapar. Programdan çıkılması konusunda kullanıcının emin olup olmadığı sorulur ve şayet kullanıcı eminse programdan çıkılır.

Program bu özelliklerinin dışında, daha bir çok hata durumunu kontrol etmekte ve oluşabilecek hata durumlarını kotarabilmektedir.

SONUÇLAR VE ÖNERİLER

Sonuç olarak, bilgisayar yardımıyla kontrol edilebilen bir xy tablasında, kullanıcı mümkün olduğu kadar sistemden yalıtılmış ve adım motoru kullanımı ile de oldukça yüksek doğruluklarda çalışma sağlanmıştır. Kullanıcının sisteme kolayca hükmedebilmesini sağlamak amacı ile bir çok seçenek sunulmuştur. Sistemde bir yörünge, birçok şekilde elde edilebilmekte ve böylece sistem yönetimi kullanıcının isteğine bırakılmaktadır.

Sistemde performans kaybına uğramamak için motorların kontrolü sırasında zaman alacak herhangi bir işlem yapılmamıştır (Simülasyon vs. gibi). Daha yüksek performanslara ulaşmak için daha hızlı bir bilgisayarla çalışmak ya da daha önceden belirtildiği gibi motorların kontrolüne akıllı bir birimin atanması gereklidir. Maliyeti arttıran bu etken, birçok tablanın bir bilgisayar tarafından kontrol edilmesi sayesinde kullanım bulabilir. Bu sayede, kesmeli çalışma yöntemi de kullanılarak bilgisayarın başka işlemlerde de kullanılması mümkün olur.

KAYNAKLAR

[1] ROBILLARD, M.J., Microprocessor Based Robotics
Howard W. Sams & Co., Inc., 1985

[2] FITZGERALD, A.E., KINGSLEY, C.JR., KUSKO, A., Electric Machinery,
The Processes, Devices, and Systems of Electromechanical Energy Conversion
Mc Graw-Hill Kogakusha Ltd.

[3] KUO, B.C., DC Motors and Control Systems,
SRL Pub. Company, 1978

[4] ACARNLEY, P.P., Stepping Motors: A Guide to Modern Theory and Practice,
IEE Control Engineering Series 16,
London, UK. Second Edition 1984

[5] KUO, B.C., Step Motors and Control Systems,
Incremental Motion Control Volume 2
SRL Pub. Company, Illinois, 1979

[6] ADALI, E., Mikroişlemciler-Mikrobilgisayarlar
Üçler Fotokopi Merkezi
Mart-1991 İstanbul

[7] SCANLON, L.J., 8086/8088/80286 Assembly Language,
Prentice-Hall Inc. New York, 1988

[8] KRUSE, R.L., Data Structures and Program Design,
Prentice-Hall Inc. New Jersey, 1984

[9] ROGERS, D.F., ADAMS, J.A., Mathematical Elements for Computer Graphics,
Mc Graw-Hill Book Company, 1976

EK A. PROGRAM DÖKÜMLERİ

```
{*********************************************************
PROGRAM ADI           : MATKAP
ALT PROGRAMLAR        : INITIALIZE
                        EXECUTE
                        DISPOSEPOINTERS
İÇERDİĞİ DOSYALAR     : MOUSE.PAS
                        GRAPHICS.PAS
                        STRUCT.PAS
                        STRUCT2.PAS
                        PRN_UNT.PAS
                        DIRECTOR.PAS
                        SCR_UNT.PAS
                        MINPATH.PAS
                        PARTDESC.PAS
                        SCAN.PAS
                        SET_UP.INC
                        DATAREAD.INC
                        SHOW.INC
                        COORDIN.INC
                        DRLINIT.INC
                        RUNSYS.INC
SABİTLER              : Step: Mikron olarak adım motorunun
bir adımına karşılık gelir.
*********************************************************}
Program Matkap;
Uses Crt, Graph, Mouse, Graphics,
     Dos, Struct, Struct2, Prn_Unt,
     Director, Scr_Unt, MinPath,
     PartDes, Scan;
Const
  Step    = 25;
Type
  CompRec = Record
              Xcc       : LongInt;
              Ycc       : LongInt;
              Rcc       : LongInt;
              Cold      : Boolean;
            end;
  RecPtr  = Array[1..4] of Pointer;
  CompAr  = Array[0..255] of CompRec;
  GapAr   = Array[1..30] of ViewPortType;
Var
  Pr, StInfPt           : Pointer;
  Qt, StInf, Err        : Boolean;
  Temp                  : Flags;
  Dataa                 : ExtNode;
  Sx, Sy, Sz, Sr        : String;
  R, Rold               : Real;
```

```
  Inte                    : LongInt;
  Num                     : Byte;
  DataFName               : FName;
  CompName                : FName;
  CArray                  : CompAr;
  GapArray                : GapAr;
  CABool                  : Boolean;

{$I Set_Up.Inc}
{$I DataRead.Inc}
{$I Show.Inc}
{$I Coordin.Inc}
{$I DrlInit.Inc}
{$I RunSys.Inc}

{*********************************************************
PROCEDURE ADI         : INITIALIZE
GİRİŞ PARAMETRELERİ   : YOK
ÇIKIŞ PARAMETRELERİ   : YOK
SABİTLER              : YOK
GÖREVİ                : ilk değerleri içeren VERİ.İLK dosya-
sını katalogda aramak ve bunları yüklemek aksi takdirde
bunlara daha önceden belirlenmiş ön değerlerini atamak.
*********************************************************}
Procedure Initialize;
Var
  FVar  : File of Flags;
begin
  SetPalette(LightRed, 60);
  SetPalette(Yellow, 62);
  SetPalette(Brown, 38);
  Top:= Nil;
  TopL:= Nil;
  Point_Num:= 1; Num:= 1;
  {$I-}
  Assign(FVar, 'VERI.ILK');
  Reset(FVar);
  if IOResult=0 then
  begin
    Read(FVar, Data);
    Close(FVar);
  end
  {$I+}
  else
  begin
    Data.TwoDim  := True;
    Data.CartCoor:= True;
    Data.RelRead := True;
    Data.Mr := 5;
    Data.Unt:= 1;
    Data.OrgX:= 0;
    Data.OrgY:= 0;
    Data.DrillR:= 0;
    Data.DrillH:= 0;
  end;
  Data.FileN:= 'VERI.ILK';
  DataFName:= '*.BIL';
```

```
  PartName:= '*.PAR';
  CompName:= '*.CAL';
  Part[2]:= 1; Part[3]:= 1; Part[4]:= 1;
  Part[5]:= 1; Part[6]:= 1; Part[7]:= 1;
  Part[8]:= 3;
  X[1]:= InX+In2X+MSp; Y[1]:= InY+In2Y+MSp;
  X[2]:= GetMaxX-Bx+MSp; Y[2]:= InY+40+MSp;
  X[3]:= 450+MSp; Y[3]:= 45+MSp;
  X[4]:= 100+MSp; Y[4]:= 70+MSp;
  X[5]:= 417; Y[5]:= 87;
  X[6]:= 455; Y[6]:= 110;
  X[7]:= 135; Y[7]:= 180;
  Qt  := False; Err:= False;
  Sx  := '0'; Sy:= '0';
  Sz  := '0'; Sr:= '0';
  R   := 0  ; Inte:= 9;
  CArray[0].Xcc:= 1;
  CArray[0].Ycc:= 1;
  CArray[0].Rcc:= 1;
  Hght[1]:= 0; Hght[2]:= 0;
  Lngth[1]:= 0; Lngth[2]:= 0;
  GapSum[1]:= 0; GapSum[2]:= 0;
  GapNum:= 0;
  StartX:= 0; StartY:= 0;
  XBool:= False; YBool:= False;
  for Num:= 1 to 20 do
  begin
    GapArr[Num].X1:= 0;
    GapArr[Num].Y1:= 0;
  end;
  StInf:= False; CABool:= False;
  Dataa.X:= 0; Dataa.S:= False;
  Dataa.Y:= 0; Dataa.Z:= 0;
  GetMem(StInfPt,ImageSize(InX+In2X+5,GetMaxY-By+In2y-55,
         GetMaxX-Info+10, GetMaxY-By+In2Y-10));
end;
{*****************************************************
PROCEDURE ADI        : EXECUTE
GiRiŞ PARAMETRELERi  : i: Alt menü no.
ÇIKIŞ PARAMETRELERi  : YOK
SABiTLER             : YOK
GOREVi               : Belirlenen alt menüye geçişi sağlaya-
rak akışı ilişkili altprograma devretmek.
*****************************************************}
Procedure Execute(i: Byte);
begin
  Case i of
  1: begin
       QSub:= False; Menu_Coor;
     end;
  2: PartDesc;
  3: Run;
  4: InitDev;
  5: begin
       QSub:= False;
       Settings;
     end;
```

```
  6: DBox(200, 300,
     'Programdan Çıkmak istediğinize Eminmisiniz?', Qt);
  end;
end;
{*********************************************************
PROCEDURE ADI        : DISPOSEPOINTERS
GİRİŞ PARAMETRELERİ  : YOK
ÇIKIŞ PARAMETRELERİ  : YOK
SABİTLER             : YOK
GÖREVİ                : Program sırasında heap ten alınan
bellek bloklarının heap e iadesini sağlamak.
*********************************************************}
Procedure DisposePointers;
begin
  DisposeMPtrs;
  FreeMem(Clear, Sze);
  FreeMem(StInfPt, ImageSize(InX+In2X+5,
          GetMaxY-By+In2y-55,
          GetMaxX-Info+10, GetMaxY-By+In2Y-10));
end;
{*********************************************************
PROCEDURE ADI        : CLOSEMESG
GİRİŞ PARAMETRELERİ  : YOK
ÇIKIŞ PARAMETRELERİ  : YOK
SABİTLER             : YOK
GÖREVİ               : Programdan çıkış sırasında programla
ilgili mesaj bırakmak.
*********************************************************}
Procedure CloseMesg;
begin
  TextBackGround(Blue);
  TextColor(Yellow);
  Write(' Kullanıcı - Matkap Arayüz Programı     ');
  Write(' Cüneyt Sabırcan/1992/YUKSEK LİSANS TEZİ');
end;

begin
  {$M 32000, 0, 655360}
  ClrScr;
  OpenGraph;
  Initialize;
  InitMouse;
  SetArrowRatio(Data.Mr, Data.Mr);
  Screen;
  Repeat
    Mouse_(2);
    Execute(Part[2]);
    Delay(150);
  Until Qt;
  DisposePointers;
  CloseGraph;
  DisposeLineList;
  DisposeLists;
  CloseMesg;
end.
```

```
{*********************************************************
UNIT ADI              : MOUSE
ALT PROGRAMLAR        : INITMOUSE
                        SETHORIZ
                        SETVERT
                        SHOWARROW
                        HIDEARROW
                        GETSTATUS
                        SETARROW
                        SETARROWRATIO
                        CHANGEARROW
                        DISPOSEMPTRS
GÖREVİ               : Fare ile ilgili temel görevleri yeri-
ne getirmek.
*********************************************************}
Unit Mouse;

Interface
Uses
  Dos, Crt;
Const
  ArrNum       = 4;
Var
  Arrow, Back : Pointer;
  Arrows      : Array[0..ArrNum] of Pointer;
  Size        : Word;
Procedure InitMouse;
Procedure SetHoriz(Min, Max: Word);
Procedure SetVert(Min, Max: Word);
Procedure ShowArrow(XCoor, YCoor: Word);
Procedure HideArrow(XCoor, YCoor:Word);
Procedure GetStatus(Var XSet, YSet, Button: Word);
Procedure SetArrow(XSet, YSet: Word);
Procedure SetArrowRatio(Xr, Yr: Byte);
Procedure ChangeArrow(ArrowNo: Byte);
Procedure DisposeMPtrs;

Implementation

Uses
  Graph;
Var
  Regs         : Registers;
{*********************************************************
PROCEDURE ADI        : CREATEARROW
GİRİŞ PARAMETRELERİ  : YOK
ÇIKIŞ PARAMETRELERİ  : YOK
SABİTLER             : Arrow_ , oluşturulacak işaretçinin
koordinatlarını taşıyan nokta dizisi.
GÖREVİ               : Fare işaretçisini oluşturarak Arrow
işaretçisine atamak.
*********************************************************}
Procedure CreateArrow;
Var
  i       : Byte;
  y       : Integer;
  FVar    : File;
```

```
Const
  Arrow_  : Array[1..8] of PointType =
        ((x:   0; y: 0),
         (x:   5; y: 11),
         (x:   9; y: 10),
         (x:  13; y: 16),
         (x:  16; y: 13),
         (x:  10; y: 9),
         (x:  11; y: 5),
         (x:   0; y: 0));

Procedure ArrowCreation;
begin
  SetViewPort(0, 0, 20, 20, True);
  SetLineStyle(SolidLn, 0, ThickWidth);
  SetColor(LightCyan);
  FillPoly(SizeOf(Arrow_) div SizeOf(PointType), Arrow_);
  GetImage(0, 0, 17, 17, Arrows[0]^);
  ClearViewPort;
  SetColor(White);
  SetLineStyle(SolidLn, 1, NormWidth);
  Line(0, 0, 14, 0);
  Line(0, 0, 0, 14);
  GetImage(0, 0, 17, 17, Arrows[1]^);
  ClearViewPort;
  SetColor(DarkGray);
  SetLineStyle(SolidLn, 1, ThickWidth);
  Line(5, 5, 5, 17);
  Line(13, 5, 13, 17);
  Line(5, 17, 13, 17);
  Line(5, 5, 8, 2);
  Line(13, 5, 10, 2);
  SetColor(LightGray);
  FillEllipse(10, 3, 1, 1);
  SetColor(Blue);
  SetLineStyle(SolidLn, 1, NormWidth);
  Line(9, 7, 9, 17);
  GetImage(0, 0, 17, 17, Arrows[2]^);
  ClearViewPort;
  SetColor(Red);
  SetLineStyle(SolidLn, 1, ThickWidth);
  Line(2, 17, 5, 10);
  Line(5, 10, 9, 13);
  Line(9, 13, 13, 4);
  Line(14, 2, 9, 7);
  Line(14, 2, 14, 8);
  GetImage(0, 0, 17, 17, Arrows[3]^);
  ClearViewPort;
  SetLineStyle(SolidLn, 1, ThickWidth);
  SetColor(White);
  Line(2, 2, 15, 15);
  Line(15, 2, 2, 15);
  GetImage(0, 0, 17, 17, Arrows[4]^);
  SetViewPort(0, 0, GetMaxX, GetMaxY, True);
end;
```

```
begin
  Size := ImageSize(0, 0, 17, 17);
  GetMem(Arrow, Size);
  GetMem(Back, Size);
  for i:= 0 to ArrNum do
  GetMem(Arrows[i], Size);
  {$I-}
  Assign(FVar, 'Mouse.Dat');
  Reset(FVar, 1);
  if IOResult<>0 then
  begin
    Close(FVar); ArrowCreation;
    ReWrite(FVar, 1);
    if IOResult<>0 then
    begin
      for i:= 0 to ArrNum do
      BlockWrite(FVar, Arrows[i]^, Size, Y);
      Close(FVar);
    end;
    {$I+}
  end
  else
  begin
    for i:= 0 to ArrNum do
    begin
      BlockRead(FVar, Arrows[i]^, Size, Y);
      if Size<>Y then
      begin
        i:= ArrNum; ArrowCreation;
      end;
    end;
    Close(FVar);
  end;
  Arrow:= Arrows[0];
end;
{*********************************************************
PROCEDURE ADI        : CHANGEARROW
GIRIŞ PARAMETRELERI : ArrowNo: Degistirilecek fare işaret-
çisinin nosu.
ÇIKIŞ PARAMETRELERI : YOK
SABITLER             : YOK
GÖREVI              : Fare işaretçisini belirtilen nolu i-
şaretçi ile değiştirmek.
*********************************************************}
Procedure ChangeArrow(ArrowNo: Byte);
begin
  Arrow:= Arrows[ArrowNo];
end;
{*********************************************************
PROCEDURE ADI        : HIDEARROW
GIRIŞ PARAMETRELERI : XCOOR, YCOOR: Fare işaretçisinin gö-
rünmez yapılacağı nokta koordinatları.
ÇIKIŞ PARAMETRELERI : YOK
SABITLER             : YOK
GÖREVI              : Fare işaretçisini belirtilen koordi-
natlarda görünmez yapmak.
*********************************************************}
```

```
Procedure HideArrow(XCoor, YCoor: Word);
begin
  PutImage(XCoor, YCoor, Back^, NormalPut);
end;
{**********************************************************
PROCEDURE ADI        : SHOWARROW
GİRİŞ PARAMETRELERİ : XCOOR, YCOOR: Fare işaretçisinin gös-
terileceği nokta koordinatları.
ÇIKIŞ PARAMETRELERİ : YOK
SABİTLER             : YOK
GÖREVİ              : Fare işaretçisini belirtilen koordi-
natlarda göstermek.
**********************************************************}
Procedure ShowArrow(XCoor, YCoor: Word);
begin
  GetImage(XCoor, YCoor, XCoor+17, YCoor+17, Back^);
  PutImage(XCoor, Ycoor, Arrow^, XorPut);
end;
{**********************************************************
PROCEDURE ADI        : INITMOUSE
GİRİŞ PARAMETRELERİ : YOK
ÇIKIŞ PARAMETRELERİ : YOK
SABİTLER             : YOK
GÖREVİ              : Fareyi kullanabilmek için gerekli ilk
düzenlemeleri yapmak.
**********************************************************}
Procedure InitMouse;
begin
  Regs.AX:= 0;
  Intr($33, Regs);
  CreateArrow;
end;
{**********************************************************
PROCEDURE ADI        : SETHORIZ
GİRİŞ PARAMETRELERİ : MIN, MAX: Fareyi düşeyde sınırlayan
nokta koordinatları.
ÇIKIŞ PARAMETRELERİ : YOK
SABİTLER             : YOK
GÖREVİ              : Fareyi düşeyde belirtilen koordinat-
lar arasında sınırlamak.
**********************************************************}
Procedure SetHoriz(Min, Max: Word);
begin
  Regs.AX:= 7;
  Regs.CX:= Min;
  Regs.DX:= Max;
  Intr($33, Regs);
end;
{**********************************************************
PROCEDURE ADI        : SETVERT
GİRİŞ PARAMETRELERİ : MIN, MAX: Fareyi yatayda sınırlayan
nokta koordinatları.
ÇIKIŞ PARAMETRELERİ : YOK
SABİTLER             : YOK
GÖREVİ              : Fareyi yatayda belirtilen koordinat-
lar arasında sınırlamak.
**********************************************************}
```

```
Procedure SetVert(Min, Max: Word);
begin
  Regs.AX:= 8;
  Regs.CX:= Min;
  Regs.DX:= Max;
  Intr($33, Regs);
end;
{*********************************************************
PROCEDURE ADI        : GETSTATUS
GİRİŞ PARAMETRELERİ : YOK
ÇIKIŞ PARAMETRELERİ : XSet, YSet: Farenin bulunduğu X ve Y
koordinatlarını belirler. Button: Farenin herhangi bir
tuşuna basılıp basılmadığını ve basılmışsa bunun hangi tuş
olduğunu belirler.
SABİTLER             : YOK
GÖREVİ               : Farenin o anki durum bilgilerini ge-
tirmek.
*********************************************************}
Procedure GetStatus(Var XSet, YSet, Button: Word);
begin
  Regs.Ax:= 3;
  Intr($33, Regs);
  Button:= Regs.BX;
  XSet:= Regs.CX; YSet:= Regs.DX;
end;
{*********************************************************
PROCEDURE ADI        : SETARROW
GİRİŞ PARAMETRELERİ : XSet, YSet: Fare işaretçisinin konum-
landırılacağı koordinat bilgileri.
ÇIKIŞ PARAMETRELERİ : YOK
SABİTLER             : YOK
GÖREVİ               : Fareyi belirtilen koordinata konum-
landırmak.
*********************************************************}
Procedure SetArrow(XSet, YSet: Word);
begin
  Regs.AX:= 4;
  Regs.CX:= XSet;
  Regs.DX:= YSet;
  Intr($33, Regs);
end;
{*********************************************************
PROCEDURE ADI        : SETARROWRATIO
GİRİŞ PARAMETRELERİ : Xr, Yr: Farenin X ve Y deki hareket
duyarlılığını belirlerler.
ÇIKIŞ PARAMETRELERİ : YOK
SABİTLER             : YOK
GÖREVİ               : Fareyi belirtilen X ve Y duyarlılı-
ğında konumlandırmayı sağlar.
*********************************************************}
Procedure SetArrowRatio(Xr, Yr: Byte);
begin
  Regs.AX:= 15;
  Regs.CX:= Xr;
  Regs.DX:= Yr;
  Intr($33, Regs);
end;
```

```
{*********************************************************
PROCEDURE ADI         : DISPOSEMPTRS
GiRiŞ PARAMETRELERi   : YOK
ÇIKIŞ PARAMETRELERi   : YOK
SABiTLER              : YOK
GOREVi               : Alınan fare isaretçilerini geri iade
etmek.
*********************************************************}
Procedure DisposeMPtrs;
Var
  i : Byte;
begin
  FreeMem(Arrow, Size);
  FreeMem(Back, Size);
  for i:= 0 to ArrNum do
  FreeMem(Arrows[i], Size);
end;
end.

{*********************************************************
UNIT ADI              : STRUCT
ALT PROGRAMLAR        : ADDEXT
                        DELETEEXTNODE
                        DISPOSELISTS
                        GETNO
                        SORT
GOREVi               : Veri yapısı ile ilgili düzenlemeleri
yapmak.
*********************************************************}
Unit Struct;
Interface
Type
  ExtNodePtr   = ^ExtNode;
  ExtNode      = Record
                   X, Y, Z     : Real;
                   S, R, C, T  : Boolean;
                   Un,   No    : Byte;
                   Link        : ExtNodePtr;
                 end;
  FileRec      = Record
                   X, Y, Z     : Real;
                   S, R, C, T  : Boolean;
                   Un,   No    : Byte;
                   Rr          : Real;
                 end;
  MainNodePtr  = ^MainNode;
  MainNode     = Record
                   Data        : Real;
                   Point       : ExtNodePtr;
                   Next        : MainNodePtr;
                 end;
Var
  Top          : MainNodePtr;
  Point_Num    : Word;
Procedure AddExt(R: Real; Dat: ExtNode);
Procedure DeleteExtNode(Var PtM: MainNodePtr;
                        Pt: ExtNodePtr);
```

```
Procedure GetNo(Ind: Byte; Var Dt: FileRec);
Procedure Sort;
Procedure DisposeLists;

Implementation
Uses Graphics;
{**********************************************************
PROCEDURE ADI        : SORT
GİRİŞ PARAMETRELERİ  : YOK
ÇIKIŞ PARAMETRELERİ  : YOK
SABİTLER             : YOK
GÖREVİ               : Matris yapısındaki listeyi çapa göre
sıralamak.
**********************************************************}
Procedure Sort;
Var
  Mp, Cp : MainNodePtr;
  TmpP   : ExtNodePtr;
  Tmp    : Real;
begin
  Mp:= Top;
  While Mp<>Nil do
  begin
    Cp:= Mp;
    While Cp<>Nil do
    begin
      Cp:= Cp^.Next;
      if Cp^.Data<Mp^.Data then
      begin
        Tmp:= Cp^.Data;
        Cp^.Data:= Mp^.Data;
        Mp^.Data:= Tmp;
        TmpP:= Cp^.Point;
        Cp^.Point:= Mp^.Point;
        Mp^.Point:= TmpP;
      end;
    end;
    Mp:= Mp^.Next;
  end;
end;
{**********************************************************
PROCEDURE ADI        : GETNO
GİRİŞ PARAMETRELERİ  : Ind: Belirtilen kayıt no.
ÇIKIŞ PARAMETRELERİ  : Dt: Giriste verilen kayıt nosunu ta-
şıyan kayıt, Rp: Ait olduğu çap.
SABİTLER             : YOK
GÖREVİ               : Belirtilen nolu kayıdı getirmek.
**********************************************************}
Procedure GetNo(Ind: Byte; Var Dt: FileRec);
Var
  Pm             : MainNodePtr;
  Pe             : ExtNodePtr;
  Found          : Boolean;
begin
  Pm:= Top;
  Found:= False;
```

```
  While (Pm<>Nil) and Not Found do
  begin
    Pe:= Pm^.Point;
    While (Pe<>Nil) and Not Found do
    begin
      if Pe^.No=Ind then Found:= True
      else
      Pe:= Pe^.Link;
    end;
    if Not Found then
    Pm:= Pm^.Next;
  end;
  if Found then
  begin
    Dt.X:= Pe^.X;
    Dt.Y:= Pe^.Y;
    Dt.Z:= Pe^.Z;
    Dt.S:= Pe^.S;
    Dt.R:= Pe^.R;
    Dt.C:= Pe^.C;
    Dt.T:= Pe^.T;
    Dt.Un:= Pe^.Un;
    Dt.Rr:= Pm^.Data;
  end;
end;
{*************************************************************
PROCEDURE ADI         : GETMAIN
GİRİŞ PARAMETRELERİ   : Re: Matkap ucu çapı.
ÇIKIŞ PARAMETRELERİ   : Ptr: Matkap ucu çapının veri olarak
içinde yeraldığı ana listeye ait düğüme işaret eden işa-
retçi.
SABİTLER              : YOK
GÖREVİ               : Matkap ucu çapının veri olarak bulun-
duğu ana listeye ait bir düğüm oluşturmak.
*************************************************************}
Procedure GetMain(Var Ptr: MainNodePtr; Re: Real);
begin
  New(Ptr);
  Ptr^.Data:= Re;
  Ptr^.Point:= Nil;
  Ptr^.Next:= Nil;
end;
{*************************************************************
PROCEDURE ADI         : ADDMAIN
GİRİŞ PARAMETRELERİ   : R: Matkap ucu çapı.
ÇIKIŞ PARAMETRELERİ   : P: Matkap ucu çapının veri olarak
içinde yeraldığı ana listeye ait düğüme işaret eden işa-
retçi.
SABİTLER              : YOK
GÖREVİ               : Matkap ucu çapının veri olarak bulun-
duğu ana listeye ait bir düğümü ana listeye eklemek.
*************************************************************}
Procedure AddMain(Var P: MainNodePtr; R: Real);
Var
  Pt_  : MainNodePtr;
```

```
begin
  GetMain(P, R);
  Pt_:= Top;
  if Top<>Nil then
  begin
    While Pt_^.Next<>Nil do
    Pt_:= Pt_^.Next;
    Pt_^.Next:= P;
  end
  else
  Top:= P;
end;
{*************************************************************
PROCEDURE ADI         : GETEXT
GiRiŞ PARAMETRELERi : Dat: Delinecek delikle ilgili koordi-
nat ve diğer bilgileri içeren kayıt.
ÇIKIŞ PARAMETRELERi : Adde: Giriste verilen kayıda işaret
eden işaretçi.
SABiTLER              : YOK
GOREVi               : Giriş parametresi olarak verilen bil-
gileri içeren bir düğüm oluşturarak işaretçisini çağrıldığı
programa döndürmek.
*************************************************************}
Procedure GetExt(Dat: ExtNode; Var AddE: ExtNodePtr);
begin
  New(AddE);
  Dat.Link:= Nil;
  AddE^:= Dat;
end;
{*************************************************************
PROCEDURE ADI         : ADDEXT
GiRiŞ PARAMETRELERi : Dat: Delinecek delikle ilgili koordi-
nat ve diğer bilgileri içeren kayıt. R: Delinecek deliğin
çapı. Veya veri yapısındaki alt liste kimliği.
ÇIKIŞ PARAMETRELERi : YOK
SABiTLER              : YOK
GOREVi               : Giriş parametresi olarak verilen bil-
gileri belirlenen çaptaki delikler listesine eklemek.
*************************************************************}
Procedure AddExt(R: Real; Dat: ExtNode);
Var
  Pt          : MainNodePtr;
  PtE, AddE   : ExtNodePtr;
  Find        : Boolean;
begin
  Pt:= Top;
  Find:= False;
  While (Pt<>Nil) and (Not Find) do
  if Pt^.Data=R then Find:= True
  else
  Pt:= Pt^.Next;
  if Not Find then AddMain(Pt, R);
  PtE:= Pt^.Point;
  if PtE<>Nil then
  begin
    While PtE^.Link<>Nil do
    PtE:= PtE^.Link;
```

```
    GetExt(Dat, AddE);
    PtE^.Link:= AddE;
  end
  else
  begin
    GetExt(Dat, AddE);
    Pt^.Point:= AddE;
  end;
end;
{*********************************************************
PROCEDURE ADI        : DELETEEXTNODE
GİRİŞ PARAMETRELERİ  : Pt: Silinecek kayıdı gösteren
işaretçi
PtM: Silinecek kayıdın bulunduğu listenin başı.
ÇIKIŞ PARAMETRELERİ : YOK
SABİTLER             : YOK
GÖREVİ               : Belirlenen kayıdı listeden çıkarmak.
*********************************************************}
Procedure DeleteExtNode(Var PtM: MainNodePtr;
                        Pt: ExtNodePtr);
Var
  PtE  : ExtNodePtr;
begin
  PtE:= PtM^.Point;
  if PtE=Pt then
  begin
    if Pt^.Link<>Nil then
    begin
      PtM^.Point:= Pt^.Link;
      Dispose(Pt);
    end
    else
    begin
      Dispose(Pt);
      if PtM=Top then
      Top:= PtM^.Next; Dispose(PtM);
    end;
  end
  else
  begin
    While PtE^.Link<>Pt do
    PtE:= PtE^.Link;
    PtE^.Link:= Pt^.Link;
    Dispose(Pt);
  end;
end;
{*********************************************************
PROCEDURE ADI        : DISPOSELISTS
GİRİŞ PARAMETRELERİ : YOK
ÇIKIŞ PARAMETRELERİ : YOK
SABİTLER             : YOK
GÖREVİ              : Oluşturulmuş olan listeleri boşaltır.
*********************************************************}
Procedure DisposeLists;
Var
  Old         : MainNodePtr;
  Top1, Old1  : ExtNodePtr;
```

```
begin
  While Top<>Nil do
  begin
    Old:= Top;
    Top1:= Top^.Point;
    While Top1<>Nil do
    begin
      Old1:= Top1;
      Top1:= Top1^.Link;
      Dispose(Old1);
    end;
    Top:= Top^.Next;
    Dispose(Old);
  end;
end;
end.

{*********************************************************
UNIT ADI              : STRUCT2
ALT PROGRAMLAR        : ADDLIST
                        DELETENODE
                        DISPOSELINELIST
GÖREVİ               : Parça tanımlaması ile oluşan veri ya-
pısı ile ilgili işlemleri içerir.
*********************************************************}
Unit Struct2;
Interface
Type
  LineP       = ^LineType;
  LineType    = Record
                  Xs, Ys, Xe, Ye : Integer;
                  Dashed         : Boolean;
                  UpView         : Boolean;
                  Next           : LineP;
                end;
  LineT       = Record
                  Xs, Ys, Xe, Ye : Integer;
                  Dashed         : Boolean;
                  UpView         : Boolean;
                end;
Var
  TopL, Last : LineP;
  Dash       : Boolean;
Procedure AddList(L: LineT);
Procedure DeleteNode(Var L: LineT; Var Found: Boolean);
Procedure DisposeLineList;

Implementation
{*********************************************************
PROCEDURE ADI         : ADDLIST
GİRİŞ PARAMETRELERİ : L: Parçanın bir kenarının başlangıç
ve bitiş koordinatları ile o doğruya ait özel bilgileri
içeren bir kayıt.
ÇIKIŞ PARAMETRELERİ : YOK
SABİTLER              : YOK
GÖREVİ               : Doğrular listesine bir kayıt eklemek.
*********************************************************}
```

```
Procedure AddList(L: LineT);
Var
  p : LineP;
begin
  New(p);
  p^.Xs:= L.Xs;
  p^.Ys:= L.Ys;
  p^.Xe:= L.Xe;
  p^.Ye:= L.Ye;
  p^.Dashed:= L.Dashed;
  p^.UpView:= L.UpView;
  p^.Next:= Nil;
  if TopL=Nil then TopL:=p
              else Last^.Next:=p;
  Last:=p;
end;
{*********************************************************
PROCEDURE ADI        : DELETENODE
GİRİŞ PARAMETRELERİ : L: Parçanın bir kenarının başlangıç
ve bitiş koordinatları ile o doğruya ait özel bilgileri
içeren bir kayıt.
ÇIKIŞ PARAMETRELERİ : L: Yukardaki ile aynı, Found: Kayıdın
başarı ile silindiğini belirten değişken.
SABİTLER             : YOK
GÖREVİ               : L ile belirtilen kayıdı doğrular lis-
tesinden çıkarmak.
*********************************************************}
Procedure DeleteNode(Var L: LineT; Var Found: Boolean);
Var
  p, q : LineP;
  Tp   : Word;
begin
  p:= TopL;
  Found:= False;
  While (p<>Nil) and Not Found do
  begin
    if (((p^.Xs=L.Xs) and (p^.Xe=L.Xe)) and
       ((p^.Ys=L.Ys) and (p^.Ye=L.Ye))) or
       (((p^.Xs=L.Xe) and (p^.Xe=L.Xs)) and
       ((p^.Ys=L.Ye) and (p^.Ye=L.Ys))) then
    if (p^.UpView=L.UpView) then
    begin
      if ((p^.Xs=L.Xe) and (p^.Xe=L.Xs)) and
      ((p^.Ys=L.Ye) and (p^.Ye=L.Ys)) then
      begin
        Tp:= L.Xs;
        L.Xs:= L.Xe;
        L.Xe:= Tp;
        Tp:= L.Ys;
        L.Ys:= L.Ye;
        L.Ye:= Tp;
      end;
      Found:= True;
      if p=TopL then
      TopL:= p^.Next
      else
      q^.Next:= p^.Next;
```

```
      Dash:= p^.Dashed;
      Dispose(p);
    end;
    q:= p;
    p:= p^.Next;
  end;
  Last:= TopL;
  While Last^.Next<>Nil do
  Last:= Last^.Next;
end;
{*************************************************************
PROCEDURE ADI         : DISPOSELINELIST
GİRİŞ PARAMETRELERİ   : YOK
ÇIKIŞ PARAMETRELERİ   : YOK
SABİTLER              : YOK
GÖREVİ                : Oluşturulan parça listesini yoketmek.
*************************************************************}
Procedure DisposeLineList;
Var
  q: LineP;
begin
  While TopL<>Nil do
  begin
    q:=TopL; TopL:=TopL^.Next;
    Dispose(q);
  end;
end;
end.
{*************************************************************
UNIT ADI              : MINPATH
ALT PROGRAMLAR        : KRUSKAL
GÖREVİ                : Delikler arasındaki en kısa yolu
bulmak.
*************************************************************}
Unit MinPath;
Interface
Const
  MaxV      = 50;
  MaxE      = Round(MaxV*(MaxV-1)/2);
Type
  Edge      = Record
                V1, V2, W : LongInt;
              end;
  EdgesT    = Array[1..MaxE] of Edge;
Procedure Kruskal(Var EdgesPar: EdgesT;VNum: Byte);
Implementation
Uses
  Crt;
Type
  Circ      = Array[1..MaxV] of Byte;
Var
  E, m      : Integer;
  Sp, i, V  : Byte;
  x, y      : Byte;
  Edges     : EdgesT;
  SpTree    : EdgesT;
  Cir_C     : Circ;
```

```
{**********************************************************
PROCEDURE ADI          : QUICKSORT
GİRİŞ PARAMETRELERİ  : A: Sıralanacak kenar dizisi, Lo:
Sıralamanın yapılacağı alt dizi indis sınırı, Hi: Sırala-
manın yapılacağı alanı sınırlayan üst dizi indisi.
ÇIKIŞ PARAMETRELERİ : A: Sıralanmış dizi.
SABİTLER               : YOK
GÖREVİ                 : Giriste belirtilen diziyi sıralayıp
yine aynı dizi üzerinden geri döndürmek.
**********************************************************}
Procedure Quicksort(Var a: EdgesT; Lo,Hi: Integer);
{**********************************************************
PROCEDURE ADI          : SORT
GİRİŞ PARAMETRELERİ : l, r: Sıralanacak dizi alanını sınır-
layan sağ ve sol indisler.
ÇIKIŞ PARAMETRELERİ : YOK
SABİTLER               : YOK
GÖREVİ                 : Giriste belirtilen diziyi QuickSort
yöntemi ile sıralamak.
**********************************************************}
Procedure Sort(l,r: Integer);
Var
  i,j : Integer;
  x,y : Edge;
begin
  i:= l; j:= r; x:= a[(l+r) Div 2];
  Repeat
    While a[i].w<x.w do i:= i+1;
    While x.w<a[j].w do j:= j-1;
    if i<=j then
    begin
      y:= a[i]; a[i]:= a[j]; a[j]:= y;
      i:= i+1; j:= j-1;
    end;
  Until i>j;
  if l<j then Sort(l,j);
  if i<r then Sort(i,r);
end;
begin
  Sort(Lo,Hi);
end;
{**********************************************************
PROCEDURE ADI          : SET_CIR
GİRİŞ PARAMETRELERİ : YOK
ÇIKIŞ PARAMETRELERİ : YOK
SABİTLER               : YOK
GÖREVİ                 : Ayrıtların çevre oluşturup oluşturma-
dığını kontrol etmek için kullanılan Cir_C dizisine ilk
değerlerini vermek.
**********************************************************}
Procedure Set_Cir;
Var
  d : Byte;
begin
 for d:= 1 to V do Cir_C[d]:= d;
end;
```

```
{*********************************************************
FUNCTION              : CONT_CIR
GİRİŞ PARAMETRELERİ  : a, b: Bağlantısı olup olmadığı
araştırılacak düğüm noları.
SABİTLER              : YOK
GÖREVİ               : Ayrıtların çevre oluşturup oluşturma-
dığını kontrol etmek.
*********************************************************}
Function Cont_Cir(a, b: Byte): Boolean;
Var
  Flag : Boolean;
  c, d : Byte;
begin
  Flag:= False;
  if Cir_C[a]<Cir_C[b] then
  begin
    c:= Cir_C[b];
    for d:= 1 to V do
    if Cir_C[d]=c then Cir_C[d]:= Cir_C[a];
  end
  else begin
         if Cir_C[a]>Cir_C[b] then
         begin
           c:= Cir_C[a];
           for d:= 1 to V do
           if Cir_C[d]=c then Cir_C[d]:= Cir_C[b];
         end
         else Flag:= True;
       end;
  Cont_Cir:= Flag;
end;
{*********************************************************
FUNCTION              : FIND
GİRİŞ PARAMETRELERİ : a, b: Spanning Tree de olup olmadığı
araştırılacak düğüm noları.
SABİTLER              : YOK
GÖREVİ               : Verilmiş düğüm nolarının bir ayrıt o-
larak Spanning Tree de bulunup bulunmadığını kontrol etmek.
*********************************************************}
Function Find(a,b: Byte): Boolean;
Var
  Ok1, Ok2 : Boolean;
  d        : Byte;
begin
  Find:= False;
  Ok1:= False;
  Ok2:= False;
  if Sp>0 then
  for d:= 1 to Sp do
  if (SpTree[d].v1=a) and (SpTree[d].v2=b) then Ok1:= True;
  if Not Ok1 then
  for d:= 1 to Sp do
  if (SpTree[d].v1=b ) and (SpTree[d].v2=a) then
  Ok2:= True;
  if (Ok1) or (Ok2) then Find:= True;
end;
```

```
{*********************************************************
PROCEDURE ADI          : KRUSKAL
GİRİŞ PARAMETRELERİ : EdgesPar: Ayrıtlar dizisi. Bu dizide
ayrıtın başlangıç ve bitiş düğümleri ile maliyeti yer
almaktadır, VNum: Grafta yer alan düğüm sayısı.
ÇIKIŞ PARAMETRELERİ : Edges Par: Kruskal algoritmasına göre
oluşturulan minimum Spanning Tree yi oluşturan ayrıtları i-
çeren dizi.
SABİTLER               : YOK
GÖREVİ                 : Kruskal algoritmasına göre minimum
Spanning Tree yi bulmak.
*********************************************************}
Procedure Kruskal(Var EdgesPar: EdgesT;VNum: Byte);
begin
  m:= 1;
  i:= 0;
  Sp:= 0;
  V:= VNum;
  E:= Round(V*(V-1)/2);
  Edges:= EdgesPar;
  Quicksort(Edges, 1, E);
  Set_Cir;
  Repeat
    x:= Edges[m].V1;
    y:= Edges[m].V2;
    Inc(m);
    if (Not(Find(x, y))) and (Not(Cont_Cir(x, y))) then
    begin
      Inc(Sp);
      SpTree[Sp]:= Edges[m-1];
      Inc(i);
    end;
  Until (m>E) or (i=V-1);
  EdgesPar:= SpTree;
end;
end.
```

Aşağıdaki alt programlar çeşitli bölümler ve dosyalardan alınmış önemli algoritmaları içerirler:

```
{*********************************************************
FUNCTION ADI           : IN_
GİRİŞ PARAMETRELERİ : Xst, Yst: Nokta Koordinatları.
SABİTLER               : YOK
GÖREVİ                 : Verilen koordinatın parça alanına gi-
rip girmediğini kontrol etmek.
*********************************************************}
Function In_(Xst, Yen: LongInt): Boolean;
begin
  In_:= False;
  if ((Xst+Trunc(Data.DrillR/2)>=StartX) and
      (Xst-Trunc(Data.DrillR/2)<=StartX+Lngth[1])) and
     ((Yen+Trunc(Data.DrillR/2)>=StartY) and
      (Yen-Trunc(Data.DrillR/2)<=StartY+Lngth[2])) then
  In_:= True;
end;
```

```
{*********************************************************
FUNCTION ADI          : ONTO
GİRİŞ PARAMETRELERİ : X, Y: Nokta Koordinatları, A, B, C,
D: Dikdörtgen koordinatları.
SABİTLER              : YOK
GÖREVİ                    : Verilen koordinatın belirtilen
dikdörtgen üzerinde olup olmadığını incelemek.
*********************************************************}
Function Onto(X, Y, A, B, C, D: LongInt): Boolean;
begin
  Onto:= False;
  if ((X=A) or (X=C)) and ((Y>=D) and (Y<=B)) then
  Onto:= True;
  if ((Y=B) or (Y=D)) and ((X<=C) and (X>=A)) then
  Onto:= True;
end;
{*********************************************************
PROCEDURE ADI         : INTERSEC
GİRİŞ PARAMETRELERİ : X1, Y1, X2, Y2: Doğru koordinatları,
Xs, Ys, Xe, Ye: Bölge sınır koordinatları.
ÇIKIŞ PARAMETRELERİ : Xs, Ys, Xe, Ye: Kesişim noktaları,
Count: Kaç noktada kesişiyorlar.
SABİTLER              : YOK
GÖREVİ                    : Bir doğru ile bir bölgenin kesişip
kesişmediğini incelemek ve eğer kesişiyorsa kesişim
noktalarını bulmak.
*********************************************************}
Procedure InterSec(X1, Y1, X2, Y2: LongInt;
                   Var Xs, Ys, Xe, Ye: LongInt;
                   Var Count: Byte);
Type
  Point_Rc = Record
               X, Y: LongInt;
             end;
  Point_Ar = Array[1..4] of Point_Rc;
Var
  Arr       : Point_Ar;
  Xq1, Xq2  : LongInt;
  Yq1, Yq2  : LongInt;
  c         : LongInt;
  m         : Real;
  y         : LongInt;
begin
  Count:= 0;
  if Y2<Y1 then
  begin
    Xq1:= X1; X1:= X2; X2:= Xq1;
    Yq1:= Y1; Y1:= Y2; Y2:= Yq1;
  end;
  if (Y2=Y1) or (X2=X1) then
  begin
    if Y2=Y1 then
    begin
      if ((Y1>=Ys) and (Y1<=Ye)) and
         ((X1<Xe) and (X2>Xs)) then
      if Not ((X1>Xs) and (X2<Xe)) then
```

```
      begin
        Count:= 1; Yq1:= Y1;
        if (X1>Xs) then Xq1:= Xe;
        if (X2<Xe) then Xq1:= Xs;
        if (X1<=Xs) and (X2>=Xe) then
        begin
          Yq2:= Y1; Xq1:= Xs;
          Xq2:= Xe; Count:= 2;
        end;
      end;
    end;
    if X2=X1 then
    begin
      if ((X1>=Xs) and (X1<=Xe)) and
         ((Y1<Ye) and (Y2>Ys)) then
      if Not ((Y1>Ys) and (Y2<Ye)) then
      begin
        Count:= 1; Xq1:= X1;
        if (Y1>Ys) then Yq1:= Ye;
        if (Y2<Ye) then Yq1:= Ys;
        if (Y1<=Ys) and (Y2>=Ye) then
        begin
          Xq2:= X1; Yq1:= Ys;
          Yq2:= Ye; Count:= 2;
        end;
      end;
    end;
  end
  else
  begin
    m:=(Y2-Y1)/(X2-X1);
    c:= Y2-Round(m*X2);
    Yq1:= Round(m*Xs+c);
    Yq2:= Round(m*Xe+c);
    if (Yq1>=Ye) and (Yq1<=Ys) and
       (Yq1>=MinF(Y1, Y2)) and (Yq1<=MaxF(Y1, Y2)) then
    begin
      Inc(Count);
      Xq1:=Round(1/m*(Yq1-c));
      Arr[Count].X:= Xq1;
      Arr[Count].Y:= Yq1;
    end;
    if (Yq2>=Ye) and (Yq2<=Ys) and
       (Yq2>=MinF(Y1, Y2)) and (Yq2<=MaxF(Y1, Y2)) then
    begin
      Inc(Count);
      Xq2:= Round(1/m*(Yq2-c));
      Arr[Count].X:= Xq2;
      Arr[Count].Y:= Yq2;
    end;
    Xq1:= Round(1/m*(Ys-c));
    Xq2:= Round(1/m*(Ye-c));
    if (Xq1>=Xs) and (Xq1<=Xe) and
       (Xq1>=MinF(X1, X2)) and (Xq1<=MaxF(X1, X2)) then
    begin
      Inc(Count);
      Yq1:= Round(m*Xq1+c);
```

```
      Arr[Count].X:= Xq1;
      Arr[Count].Y:= Yq1;
    end;
    if (Xq2>=Xs) and (Xq2<=Xe) and
       (Xq2>=MinF(X1, X2)) and (Xq2<=MaxF(X1, X2)) then
    begin
      Inc(Count);
      Yq2:= Round(m*Xq2+c);
      Arr[Count].X:= Xq2;
      Arr[Count].Y:= Yq2;
    end;
    for c:= 1 to Count-1 do
    for y:= c+1 to Count do
    begin
      if (Arr[c].X=Arr[y].X) and (Arr[c].Y=Arr[y].Y) then
      begin
        Arr[y].X:= 0;
        Arr[y].Y:= 0;
        if Count=2 then Dec(Count);
      end;
    end;
    if Count>0 then
    begin
      Xq1:= Arr[1].X; Yq1:= Arr[1].Y;
      y:= 2;
      While y<=Count do
      begin
        if Arr[y].X+Arr[y].Y<>0 then
        begin
          Xq2:= Arr[y].X; Yq2:= Arr[y].Y;
          y:= Count; Count:= 2;
        end;
        Inc(y);
      end;
      if Count<>2 then Count:= 1;
    end;
  end;
end;
{*************************************************************
PROCEDURE ADI        : FINDPATH
GİRİŞ PARAMETRELERİ  : YOK
ÇIKIŞ PARAMETRELERİ  : YOK
SABİTLER             : YOK
GÖREVİ               : Elde edilen derleyici dizisini parça-
nın varlığına göre yeniden düzenlemek ve gerekiyorsa hata
vermek.
*************************************************************}
Procedure FindPath;
Var
  Indi, k    : Byte;
  PNum, i    : Byte;
  Okay, Okk  : Boolean;
  Ss         : String;
  Count      : Byte;
  Rx1, Ry1   : LongInt;
  Rx2, Ry2   : LongInt;
  Xs, Ys     : LongInt;
```

```
  Xe, Ye     : LongInt;
  Tmp        : LongInt;
begin
  Indi:= 1;
  While (((CArray[Indi].Xcc<>0) or (CArray[Indi].Ycc<>0))
        or (CArray[Indi].Rcc<>0)) and Not Err do
  begin
     if In_(CArray[Indi].Xcc, CArray[Indi].Ycc) then
     begin
       Okay:= False;
       for k:= 1 to GapSum[1] do
       begin
         if (CArray[Indi].Xcc-Trunc(Data.DrillR/2)
             >=GapArr[k].X1) and
            (CArray[Indi].Xcc+Trunc(Data.DrillR/2)
             <=GapArr[k].X2) then
         begin
           Okay:= True; k:= GapSum[1];
         end;
       end;
       if Not Okay then
       for k:= 1 to GapSum[2] do
       begin
         if (CArray[Indi].Ycc-Trunc(Data.DrillR/2)
             >=GapArr[k].Y1) and
            (CArray[Indi].Ycc+Trunc(Data.DrillR/2)
             <=GapArr[k].Y2) then
         begin
           Okay:= True; k:= GapSum[2];
         end;
       end;
       FillRegion(MiX+10, MiY+151,
                  MiX+11+Trunc((189/(Point_Num-1))*Indi),
                  MiY+169, Yellow);
       if Not Okay then
       begin
         SetColor(Blue);
         OutTextXY(MiX+70, MiY+187,
                   'Derlemede Hataya Rastlanmadı.');
         Err:= True;
         Sound(500); Delay(200); NoSound;
         Str(Indi, Ss);
         SetColor(LightRed);
         if Part[8]=3 then
         begin
           OutTextXY(MiX+70, MiY+187, 'Kayıt No');
           OutTextXY(MiX+138, MiY+187, Ss+'.');
           OutTextXY(MiX+160, MiY+187, 'Parça işleme
           Hatası.');
         end
         else
         OutTextXY(MiX+70, MiY+187,
         'Parça işleme Hatası.'); WaitKey;
       end;
     end;
     Inc(Indi);
  end;
```

```
if Not Err then
begin
  FillRegion(MiX+10, MiY+151,
             MiX+11+189, MiY+169, Yellow);
  OutTextXY(MiX+70, MiY+187,
            'Derlemede Hataya Rastlanmadı.');
  CABool:= True; Indi:= 0;
  PNum:= Point_Num;
  Xs:= StartX-Trunc(Data.DrillR/2);
  Ys:= StartY+Lngth[2]+Trunc(Data.DrillR/2);
  Xe:= StartX+Lngth[1]+Trunc(Data.DrillR/2);
  Ye:= StartY-Trunc(Data.DrillR/2);
  if Xe>900000 then Xe:= 900000;
  if Ye<0 then Ye:= 0;
  if Xs<0 then Xs:= 0;
  if Ys>600000 then Ys:= 600000;
  While ((CArray[Indi].Xcc<>0) or (CArray[Indi].Ycc<>0))
        or (CArray[Indi].Rcc<>0) do
  begin
    if Indi=0 then
    InterSec(0, 0, CArray[1].Xcc, CArray[1].Ycc,
             Xs, Ys, Xe, Ye, Count)
    else
    InterSec(CArray[Indi].Xcc, CArray[Indi].Ycc,
             CArray[Indi+1].Xcc, CArray[Indi+1].Ycc,
             Xs, Ys, Xe, Ye, Count);
    if (Count=1) and
       (Onto(CArray[Indi].Xcc, CArray[Indi].Ycc,
             Xs, Ys, Xe, Ye) or
        Onto(CArray[Indi+1].Xcc, CArray[Indi+1].Ycc,
             Xs, Ys, Xe, Ye)) then
    Count:= 0;
    if Count<>0 then
    begin
      if Indi<>0 then
      begin
        if CArray[Indi].Ycc<CArray[Indi+1].Ycc then
        begin
          Rx1:= CArray[Indi+1].Xcc;
          Ry1:= CArray[Indi+1].Ycc;
          Rx2:= CArray[Indi].Xcc;
          Ry2:= CArray[Indi].Ycc;
        end
        else
        begin
          Rx1:= CArray[Indi].Xcc;
          Ry1:= CArray[Indi].Ycc;
          Rx2:= CArray[Indi+1].Xcc;
          Ry2:= CArray[Indi+1].Ycc;
        end;
      end
      else
      begin
        Rx1:= CArray[1].Xcc;
        Ry1:= CArray[1].Ycc;
        Rx2:= 0; Ry2:= 0;
      end;
```

```
if Indi<>0 then
for i:= PNum Downto Indi+1 do
CArray[i+1]:= CArray[i]
else
for i:= PNum Downto 1 do
CArray[i+1]:= CArray[i];
if Count=2 then begin
  if Rx1<=Xs then                    { 1&4 }
  begin
    if Ry1<Ys then                   { 4 }
    begin
      if Ye=0 then
      begin
        CArray[i].Xcc:= Xs;
        CArray[i].Ycc:= Ys;
        CArray[i].Rcc:= 0;
      end
      else
      begin
        CArray[i].Xcc:= Xs;
        CArray[i].Ycc:= Ye;
        CArray[i].Rcc:= 0;
      end;
    end
    else                             { 1 }
    begin
      if Rx2>=Xe then                { 1-5&8 }
      begin
        CArray[i].Xcc:= Xe;
        CArray[i].Ycc:= Ys;
        CArray[i].Rcc:= 0;
      end
      else
      begin                          { 1-7 }
        CArray[i].Xcc:= Xs;
        CArray[i].Ycc:= Ye;
        CArray[i].Rcc:= 0;
      end;
    end;
  end;
  if (Rx1>Xs) and (Rx1<Xe) then      { 2 }
  begin
    if Rx2<Xe then
    begin                            { 2-4&6&7 }
      CArray[i].Xcc:= Xs;
      CArray[i].Ycc:= Ys;
      CArray[i].Rcc:= 0;
      if Xs=0 then
      CArray[i].Xcc:= Xe;
    end
    else                             { 2-5&8 }
    begin
      CArray[i].Xcc:= Xe;
      CArray[i].Ycc:= Ys;
      CArray[i].Rcc:= 0;
    end;
  end;
```

```
  if Rx1>=Xe then                          { 3&5 }
  begin
    if Ry1>Ys then                         { 3 }
    begin
      if (Rx2<=Xe) and (Rx2>=Xs) then { 3-7 }
      begin
        CArray[i].Xcc:= Xe;
        CArray[i].Ycc:= Ye;
        CArray[i].Rcc:= 0;
        if Ye=0 then
        CArray[i].Rcc:= Ys;
      end
      else                                 { 3-4&6 }
      begin
        CArray[i].Xcc:= Xs;
        CArray[i].Ycc:= Ys;
        CArray[i].Rcc:= 0;
      end;
    end
    else                                   { 5 }
    begin
      CArray[i].Xcc:= Xe;
      CArray[i].Ycc:= Ye;
      CArray[i].Rcc:= 0;
      if (Ry2>=Ye) and (Ry2<Ys) then { 4-5 }
      begin
        if Ys=600000 then
        CArray[i].Ycc:= Ye;
        if Ye=0 then
        CArray[i].Ycc:= Ys;
        CArray[i].Xcc:= Xs;
        CArray[i].Rcc:= 0;
      end;
    end;
  end;
end
else
begin
  if  Not (((Rx1>=StartX) and
            (Rx1<=StartX+Lngth[1]))
  and ((Ry1>=StartY) and (Ry1<=StartY+Lngth[2])))
  then begin
    Tmp:= Rx1; Rx1:= Rx2; Rx2:= Tmp;
    Tmp:= Ry1; Ry1:= Ry2; Ry2:= Tmp;
  end;
  Okay:= False;
  for k:= 1 to GapSum[1] do
  begin
    if (Rx1>=GapArr[k].X1) and
       (Rx1<=GapArr[k].X2) then
    begin
      Okay:= True; k:= GapSum[1];
    end;
  end;
  if Okay then
  begin
```

```
            if (Abs(Ry2-Ys)>Abs(Ry2-Ye)) and (Ye<>0) then
            begin
              CArray[i].Xcc:= Rx1;
              CArray[i].Ycc:= Ye;
              CArray[i].Rcc:= 0;
            end
            else
            begin
              CArray[i].Xcc:= Rx1;
              CArray[i].Ycc:= Ys;
              CArray[i].Rcc:= 0;
              if Ys=600000 then
              CArray[i].Ycc:= Ye;
            end;
          end;
          if Not Okay then
          begin
            if (Abs(Rx2-Xs)>Abs(Rx2-Xe)) and
               (Xe<>900000) then
            begin
              CArray[i].Xcc:= Xe;
              CArray[i].Ycc:= Ry1;
              CArray[i].Rcc:= 0;
            end
            else
            begin
              CArray[i].Xcc:= Xs;
              CArray[i].Ycc:= Ry1;
              CArray[i].Rcc:= 0;
              if Xs=0 then
              CArray[i].Xcc:= Xe;
            end;
          end;
        end;
        Inc(PNum); Dec(Indi);
      end;
      Inc(Indi);
    end;
  end;
end;
{*********************************************************
PROCEDURE ADI        : FOLLOWLINE
GiRiŞ PARAMETRELERi : X1, Y1, X2, Y2: Doğrunun başlangıç ve
bitiş koordinatları, Clor: Doğrunun çizileceği renk.
ÇIKIŞ PARAMETRELERi : YOK
SABiTLER             : YOK
GÖREVi                : DDT algoritmasına göre belirtilen
renkteki doğruyu çizmek.
*********************************************************}
Procedure FollowLine(X1, Y1, X2, Y2: Integer; Clor: Byte);
Const
  Dly = 10;
Var
  Xinc, Yinc, X, Y : Real;
  Len, Tmp1, Tmp2  : Integer;
  i                : Word;
```

```
{*********************************************************
FUNCTION ADI          : SIGN
GİRİŞ PARAMETRELERİ   : A: işareti incelenecek reel sayı.
SABİTLER              : YOK
GÖREVİ              : Girilen sayının SIGN fonksiyonuna gö-
re değerini vermek.
*********************************************************}
Function Sign(A: Real): Integer;
begin
  Sign:= 0;
  if A<0 then Sign:= -1
  else
  if A>0 then Sign:= 1
end;
begin
  Tmp1:= X2-X1; Tmp2:= Y2-Y1;
  if Abs(X2-X1)>=Abs(Y2-Y1) then Len:= Abs(Tmp1)
                            else Len:= Abs(Tmp2);
  if Len=0 then
  begin
    Xinc:= 0; Yinc:= 0;
  end
  else
  begin
    Xinc:= Tmp1/Len; Yinc:= Tmp2/Len;
  end;
  X:= X1+0.5*Sign(Xinc);
  Y:= Y1+0.5*Sign(Yinc);
  i:= 1;
  While (i<Len) do
  begin
    PutPixel(Round(X), Round(Y), Clor);
    X:= X+Xinc; Y:= Y+Yinc;
    Inc(i);
    if Clor=White then Delay(Dly);
  end;
end;
{*********************************************************
PROCEDURE ADI         : SIMULATION
GİRİŞ PARAMETRELERİ   : YOK
ÇIKIŞ PARAMETRELERİ   : YOK
SABİTLER              : Dvs: Adım sayısı/ekran ölçeği oranı.
GÖREVİ              : Çalıştırılacak programın ekranda pro-
vasını yapmak.
*********************************************************}
Procedure Simulation;
Const
  Dvs    = 80;
Var
  t, X1, Y1   : Word;
  Xo, Yo, Noo : Word;
  Ro, R       : LongInt;
  Tm1, Tm2    : LongInt;
  Cntr        : Boolean;
  St          : String;
  PSpa        : Pointer;
```

```
begin
  GetMem(PSpa, ImageSize(0, 0, 8, 8));
  SetColor(Cyan);
  FillRegion(InX+In2X, InY+In2Y,
             GetMaxX-Bx, GetMaxY-By, Cyan);
  GetImage(InX+In2X, InY+In2Y,
           InX+In2X+8, InY+In2Y+8, PSpa^);
  SetColor(Black);
  for t:= 1 to 10 do
  begin
    Line(50+10, 370+t, 500+10, 370+t);
    Line(500+t, 70+10, 500+t, 370+10);
  end;
  SetColor(Yellow);
  SetLineStyle(DottedLn, 1, NormWidth);
  for t:= 2 to 9 do
  Line(t*50, 10+InY+In2Y+50, t*50, 10+InY+In2Y+350);
  for t:= 2 to 6 do
  Line(50, 10+InY+In2Y+t*50, 500, 10+InY+In2Y+t*50);
  SetLineStyle(SolidLn, 1, NormWidth);
  SetColor(Blue);
  Rectangle(50, 70, 500, 370);
  X1:= 50;
  Y1:= 370;
  if (TopL<>Nil) then
  begin
    Tm1:= StartX+Lngth[1];
    Tm2:= StartY+Lngth[2];
    if Tm1>900000 then Tm1:= 900000;
    if Tm2>600000 then Tm2:= 600000;
    SetColor(Yellow);
    SetFillStyle(SlashFill, Blue);
    FillRegion(X1+Trunc(StartX/2000),
               Y1-Trunc(Tm2/2000),
               X1+Trunc(Tm1/2000),
               Y1-Trunc(StartY/2000), Blue);
    SetFillStyle(SolidFill, Blue);
    SetColor(Blue);
    for t:= 1 to GapSum[1] do
    FillRegion(X1+Trunc(GapArr[t].X1/2000),
               Y1-Trunc(StartY/2000),
               X1+Trunc(GapArr[t].X2/2000),
               Y1-Trunc(Tm2/2000), Blue);
    for t:= 1 to GapSum[2] do
    FillRegion(X1+Trunc(StartX/2000),
               Y1-Trunc(GapArr[t].Y1/2000),
               X1+Trunc(Tm1/2000),
               Y1-Trunc(GapArr[t].Y2/2000), Blue);
    SetColor(LightRed);
    SetLineStyle(DashedLn, 0, NormWidth);
    for t:= 1 to GapSum[1] do
    begin
      Line(X1+Trunc(GapArr[t].X1/2000),
           Y1-Trunc(StartY/2000),
           X1+Trunc(GapArr[t].X1/2000),
           Y1-Trunc(Tm2/2000));
```

```
    Line(X1+Trunc(GapArr[t].X2/2000),
         Y1-Trunc(StartY/2000),
         X1+Trunc(GapArr[t].X2/2000),
         Y1-Trunc(Tm2/2000));
  end;
  for t:= 1 to GapSum[2] do
  begin
    Line(X1+Trunc(StartX/2000),
         Y1-Trunc(GapArr[t].Y1/2000),
         X1+Trunc(Tm1/2000),
         Y1-Trunc(GapArr[t].Y1/2000));
    Line(X1+Trunc(StartX/2000),
         Y1-Trunc(GapArr[t].Y2/2000),
         X1+Trunc(Tm1/2000),
         Y1-Trunc(GapArr[t].Y2/2000));
  end;
  SetLineStyle(SolidLn, 0, NormWidth);
  SetColor(Yellow);
  Rectangle(X1+Trunc(StartX/2000),
            Y1-Trunc(Tm2/2000),
            X1+Trunc(Tm1/2000),
            Y1-Trunc(StartY/2000));
end;
Cntr:= False;
t:= 1; Ro:= -1; Noo:= 1;
MoveTo(50, 370);
X1:= 50;  Xo:= 50;
Y1:= 370; Yo:= 370;
SetColor(White);
While ((CArray[t].Xcc<>0) or (CArray[t].Ycc<>0)) or
      (CArray[t].Rcc<>0) do
begin
  R:= CArray[t].Rcc;
  X1:= X1+Round(CArray[t].Xcc/Dvs);
  Y1:= Y1-Round(CArray[t].Ycc/Dvs);
  Str(Noo, St);
  SetFillStyle(SolidFill, Red);
  if (R<>0) and ((X1<>Xo) and (Y1<>Yo)) then
  begin
    FillEllipse(X1, Y1, 5, 5);
    PutImage(X1-4, Y1-15, PSpa^, NormalPut);
    OutTextXY(X1-4, Y1-15, St);
  end;
  if R<>0 then
  Inc(Noo); Inc(t);
end;
t:= 1; Ro:= -1; Noo:= 1;
MoveTo(50, 370);
X1:= 50;  Xo:= 50; Y1:= 370; Yo:= 370;
While ((CArray[t].Xcc<>0) or (CArray[t].Ycc<>0)) or
      (CArray[t].Rcc<>0) do
begin
  R:= CArray[t].Rcc;
  if (R<>Ro) and (R<>0) then Message(R);
  X1:= X1+Round(CArray[t].Xcc/Dvs);
  Y1:= Y1-Round(CArray[t].Ycc/Dvs);
  SetFillStyle(SolidFill, Cyan);
```

```
    if (t<>1) and Not Cntr then
    FillEllipse(Xo, Yo, 5, 5);
    FollowLine(Xo, Yo, X1, Y1, White);
    FollowLine(Xo, Yo, X1, Y1, LightRed);
    Str(Noo, St); SetFillStyle(SolidFill, White);
    if R<>0 then
    begin
      FillEllipse(X1, Y1, 5, 5);
      PutImage(X1-4, Y1-15, PSpa^, NormalPut);
      OutTextXY(X1-4, Y1-15, St);
    end;
    Delay(400); Xo:= X1; Yo:= Y1;
    if R<>0 then
    begin
      Ro:= R; Cntr:= False; Inc(Noo);
    end
    else Cntr:= True;
    Inc(t);
  end;
  SetFillStyle(SolidFill, Cyan);
  if Not Cntr and (Noo<>1) then
  FillEllipse(Xo, Yo, 5, 5);
  FollowLine(X1, Y1, 50, 370, White);
  FollowLine(X1, Y1, 50, 370, LightRed);
  WaitKey; Delay(200);
  GetImage(MiX, MiY, MaX, MaY, Pp^);
  ScreenDsgn; LigDrk;
  FreeMem(PSpa, ImageSize(0, 0, 8, 8));
end;
{*************************************************************
PROCEDURE ADI        : ADDGAP
GİRİŞ PARAMETRELERİ : Xb, Yb, Xs, Ys: Parçada yer alan boş-
luğu tanımlayan koordinatlar.
ÇIKIŞ PARAMETRELERİ : YOK
SABİTLER             : YOK
GÖREVİ               : Belirtilen boşluğu boşluk dizisine a-
tamak. Not: Bu boşluk koordinatları ekran koordinatı olarak
tutulmaktadır.
*************************************************************}
Procedure AddGap(Xb, Yb, Xs, Ys: Word);
begin
  Inc(GapNum); GapArray[GapNum].X1:= Xb;
  GapArray[GapNum].Y1:= Yb;
  GapArray[GapNum].X2:= Xs;
  GapArray[GapNum].Y2:= Ys;
end;
{*************************************************************
PROCEDURE ADI        : RATIORD
GİRİŞ PARAMETRELERİ : XySec: Xy yada Yz koordinatlarında
çalıştığımızı belirleyen lojik değişken. Bu değişken önden
yada yandan görünüşte değişik değerler alır.
ÇIKIŞ PARAMETRELERİ : YOK
SABİTLER             : YOK
GÖREVİ               : Parçanın tam olarak tanımlandığı bö-
lümdür. Teknik çizimi yapılmış parça bu alt program
tarafından boyutlandırılır.
*************************************************************}
```

```
Procedure RatioRd(XySec: Boolean);
Type
  Intersec = Record
               X : Word;
               Y : Word;
             end;
  DotArray = Array[1..25] of Intersec;
Var
  MiY, MaY, MiX, MaX, XCur, XCc : Word;
  XCur2, YCc, YCur, YCur2       : Word;
  Xq, Yq, Bt, LeftMar, RightMar : Word;
  DotNum                        : Byte;
  q                             : LineP;
  DotAr                         : DotArray;
  Yy                            : GapRec;
  Ss                            : String;
  Ec, Toggle, Okk, Kk           : Boolean;
  Pp, Pq                        : Pointer;
  Tmm, Cc                       : LongInt;
  Mm                            : Real;
  Cd                            : Integer;
begin
  if XySec or XBool then
  begin
  SetWriteMode(XorPut);
  MiY:= GetMaxY; MiX:= GetMaxX;
  XCur:= GetMaxX; XCur2:= GetMaxX;
  MaY:= 0; MaX:= 0; DotNum:= 0; GapNum:= 0;
  Toggle:= False;
  SetLineStyle(SolidLn, 1, NormWidth);
  q:= TopL;
  While q<>Nil do
  begin
    if q^.UpView=Not XySec then
    if Not q^.Dashed then
    begin
      Toggle:= True;
      if q^.Ys>=MaY then if q^.Ys=MaY then
                begin
                  if q^.Xs<XCur2 then XCur2:= q^.Xs;
                end
                else
                begin
                  XCur2:= q^.Xs;
                  MaY:= q^.Ys;
                end;
      if q^.Ye>=MaY then if q^.Ye=MaY then
                  begin
                  if q^.Xe<XCur2 then  XCur2:= q^.Xe;
                end
                else
                begin
                  XCur2:= q^.Xe;
                  MaY:= q^.Ye;
                end;
      if q^.Ys<=MiY then if q^.Ys=MiY then
```

```
            begin
              if q^.Xs<XCur then
              XCur:= q^.Xs;
            end
            else
            begin
              XCur:= q^.Xs;
              MiY:= q^.Ys;
            end;
if q^.Ye<=MiY then if q^.Ye=MiY then
            begin
              if q^.Xe<XCur then
              XCur:= q^.Xe;
            end
            else
            begin
              XCur:= q^.Xe;
              MiY:= q^.Ye;
            end;
if q^.Xs>=MaX then if q^.Xs=MaX then
                   begin
                     if q^.Ys>YCur2 then
                     YCur2:= q^.Ys;
                   end
                   else
                   begin
                     MaX:= q^.Xs;
                     YCur2:= q^.Ys;
                   end;
if q^.Xe>=MaX then if q^.Xe=MaX then
                   begin
                     if q^.Ye>YCur2 then
                     YCur2:= q^.Ye;
                   end
                   else
                   begin
                     MaX:= q^.Xe;
                     YCur2:= q^.Ye;
                   end;
if q^.Xs<=MiX then if q^.Xs=MiX then
                   begin
                     if q^.Ys>YCur then
                     YCur:= q^.Ys;
                   end
                   else
                   begin
                     MiX:= q^.Xs;
                     YCur:= q^.Ys;
                   end;
if q^.Xe<=MiX then if q^.Xe=MiX then
                   begin
                     if q^.Ye>YCur then
                     YCur:= q^.Ye;
                   end
                   else
```

```
                        begin
                          MiX:= q^.Xe;
                          YCur:= q^.Ye;
                        end;
    end;
    q:= q^.Next;
  end;
  if Toggle then
  if MiY<>MaY then
  begin
    Toggle:= True;
    YCc:= Trunc((MaY+425)/2);
    Repeat
      SetColor(White);
      Line(MiX, YCc, MaX, YCc);
      Line(MiX, YCur, MiX, YCc+10);
      Line(MaX, YCur2, MaX, YCc+10);
      Line(MiX, YCc, MiX+9, YCc-2);
      Line(MiX+9, YCc-2, MiX+9, YCc+2);
      Line(MiX+9, YCc+2, MiX, YCc);
      Line(MaX, YCc, MaX-9, YCc-2);
      Line(MaX-9, YCc-2, MaX-9, YCc+2);
      Line(MaX-9, YCc+2, MaX, YCc);
      if Toggle then
      begin
     GetMem(Pp, ImageSize(20, 133, 196, 164));
     GetImage(20, 133, 196, 164, Pp^);
     ClearRegion(26, 139, 196, 164);
     FillRegion(20, 133, 190, 158, Yellow);
     GetMem(Pq, ImageSize(70, 133, 145, 158));
     GetImage(70, 133, 145, 158, Pq^);
     SetColor(Red);
     if XySec then
     OutTextXY(35, 144, 'x :')
     else
     OutTextXY(35, 144, 'z :');
     OutTextXY(155, 144, 'µ');
     SetColor(Black);
     if XySec then Str(Lngth[1], Ss)
     else Str(Lngth[2], Ss);
     Format(Ss);
     OutTextXY(70, 145, Ss);
     Okk:= False;
     Repeat
       GraphRead(70, 145, NumLen, Ss, Ec, Pq);
       if Ss='' then
          if XySec then
          Str(Lngth[1], Ss)
          else
          Str(Lngth[2], Ss);
       if Not Ec then
       begin
         Val(Ss, Tmm, Cd);
         if (Cd<>0) or (Tmm<=0) then
         begin
           PutImage(70, 133, Pq^, NormalPut);
```

```
      if XySec then Str(Lngth[1], Ss)
      else Str(Lngth[2], Ss);
      Format(Ss);
      OutTextXY(70, 145, Ss);
      end
      else
      begin
           if XySec then
           begin
             XBool:= False;
          Lngth[1]:= Tmm
           end
           else
           begin
             YBool:= False;
             Lngth[2]:= Tmm;
           end;
        Okk:= True;
      end;
    end;
 Until Okk or Ec;
 PutImage(20, 133, Pp^, NormalPut);
 FreeMem(Pq, ImageSize(70, 133, 145, 158));
 FreeMem(Pp, ImageSize(20, 133, 196, 164));
 end;
 Toggle:= Not Toggle;
Until Toggle;
if Not Ec then
begin
  Toggle:= True;
  XCc:= Trunc((MiX+25)/2);
  Repeat
    SetColor(White);
    Line(XCc, MaY, XCc, MiY);
    Line(XCc-10, MiY, XCur, MiY);
    Line(XCur2, MaY, XCc-10, MaY);
    Line(XCc, MiY, XCc+2, MiY+9);
    Line(XCc+2, MiY+9, XCc-2, MiY+9);
    Line(XCc-2, MiY+9, XCc, MiY);
    Line(XCc, MaY, XCc-2, MaY-9);
    Line(XCc-2, MaY-9, XCc+2, MaY-9);
    Line(XCc+2, MaY-9, XCc, MaY);
    if Toggle then
    begin
    GetMem(Pp, ImageSize(20, 133, 196, 164));
    GetImage(20, 133, 196, 164, Pp^);
    ClearRegion(26, 139, 196, 164);
    FillRegion(20, 133, 190, 158, Yellow);
    GetMem(Pq, ImageSize(70, 133, 145, 158));
    GetImage(70, 133, 145, 158, Pq^);
    SetColor(Red);
    OutTextXY(35, 144, 'y :');
    OutTextXY(155, 144, 'µ');
    SetColor(Black);
    if XySec then
    Str(Hght[1], Ss)
```

```
      else
      Str(Hght[2], Ss);
      Format(Ss);
      OutTextXY(70, 145, Ss);
      Okk:= False;
      Repeat
        GraphRead(70, 145, NumLen, Ss, Ec, Pq);
        if Ss='' then
        if XySec then
        Str(Hght[1], Ss)
        else
        Str(Hght[2], Ss);
        if Not Ec then
        begin
          Val(Ss, Tmm, Cd);
          if (Cd<>0) or (Tmm<=0) then
          begin
            PutImage(70, 133, Pq^, NormalPut);
            if XySec then
            Str(Hght[1], Ss)
            else
            Str(Hght[2], Ss);
            Format(Ss);
            OutTextXY(70, 145, Ss);
         end
         else
         begin
           if XySec then
           Hght[1]:= Tmm
           else
           Hght[2]:= Tmm;
           Ec:= True;
           if XySec then
           Cc:= Data.DrillH-Hght[1]
           else
           Cc:= Data.DrillH-Hght[2];
           if Cc>HghtTol then
           begin
             if XySec then XBool:= True
                      else YBool:= True;
             Okk:= False;
           end
           else Okk:= True;
         end;
       end;
     Until Okk or Ec;
     PutImage(20, 133, Pp^, NormalPut);
     FreeMem(Pq, ImageSize(70, 133, 145, 158));
     FreeMem(Pp, ImageSize(20, 133, 196, 164));
     end;
     Toggle:= Not Toggle;
   Until Toggle;
end;
if Okk then
begin
  SetColor(Black);
  GetMem(Pq, ImageSize(20, 133, 296, 164));
```

```
GetImage(20, 133, 296, 164, Pq^);
ClearRegion(26, 139, 296, 164);
FillRegion(20, 133, 290, 158, Yellow);
Str(Data.DrillH, Ss);
Format(Ss);
Ss:= Ss+' µ';
OutTextXY(30, 145, Ss);
OutTextXY(120, 145, 'Çizgisini Belirleyin.');
SetColor(White); YCc:= MiY+2;
Delay(150);
Line(25, YCc, 430, YCc);
Repeat
Bt:= 0;
SetArrow(480, 300);
GetStatus(Xq, Yq, Bt);
if Yq<>300 then
begin
  Line(25, YCc, 430, YCc);
  if Yq>300 then
  Inc(YCc, 2)
  else
  Dec(YCc, 2);
  if YCc<MiY+2 then
  YCc:= MiY+2;
  if YCc>MaY-2 then
  YCc:= MaY-2;
  Line(25, YCc, 430, YCc);
end;
 Until Bt<>0;
 Line(25, YCc, 430, YCc);
 PutImage(20, 133, Pq^, NormalPut);
 FreeMem(Pq, ImageSize(20, 133, 296, 164));
 if Bt<>2 then
 begin
   q:= TopL;
   While q<>Nil do
   begin
     if (((YCc>q^.Ys) and (YCc<q^.Ye)) or
         ((YCc>q^.Ye) and (YCc<q^.Ys))) and
        (((q^.Ys<>q^.Ye) and Not q^.Dashed) and
          (XySec<>q^.UpView)) then
     begin
       if q^.Xs<>q^.Xe then
       begin
         Mm:= (q^.Xs-q^.Xe)/(q^.Ys-q^.Ye);
         Cc:= Round(q^.Ys-1/Mm*q^.Xs);
         XCc:= Round(Mm*(YCc-Cc));
       end
       else
       XCc:= q^.Xs;
       Inc(DotNum);
       DotAr[DotNum].X:= XCc;
       DotAr[DotNum].Y:= YCc;
     end;
     q:= q^.Next;
   end;
```

```
for Bt:= 1 to DotNum-1 do
for Xq:= Bt+1 to DotNum do
if DotAr[Bt].X>DotAr[Xq].X then
begin
  Yq:= DotAr[Bt].X;
  DotAr[Bt].X:= DotAr[Xq].X;
  DotAr[Xq].X:= Yq;
  Yq:= DotAr[Bt].Y;
  DotAr[Bt].Y:= DotAr[Xq].Y;
  DotAr[Xq].Y:= Yq;
end;
q:= TopL;
if DotAr[1].X<>MiX then
begin
  While q<>Nil do
  begin
    if Not q^.Dashed then
    if q^.UpView<>XySec then
    begin
      if (q^.Xs<DotAr[1].X) and
         (q^.Ys<DotAr[1].Y) then
      begin
        DotAr[1].X:= q^.Xs;
        Yq:= q^.Ys;
      end;
      if (q^.Xe<DotAr[1].X) and
         (q^.Ye<DotAr[1].Y) then
      begin
        DotAr[1].X:= q^.Xe;
        Yq:= q^.Ye;
      end;
    end;
    q:= q^.Next;
  end;
  if MiX<>DotAr[1].X then
  AddGap(MiX, Yq, DotAr[1].X, Yq);
end;
q:= TopL;
if DotAr[DotNum].X<>MaX then
begin
  While q<>Nil do
  begin
    if Not q^.Dashed then
    if q^.UpView<>XySec then
    begin
      if (q^.Xs>DotAr[DotNum].X) and
         (q^.Ys<DotAr[DotNum].Y) then
      begin
        DotAr[DotNum].X:= q^.Xs;
        Yq:= q^.Ys;
      end;
      if (q^.Xe>DotAr[DotNum].X) and
         (q^.Ye<DotAr[DotNum].Y) then
      begin
        DotAr[DotNum].X:= q^.Xe; Yq:= q^.Ye;
      end;
    end;
```

```
    q:= q^.Next;
  end;
  if MaX<>DotAr[DotNum].X then
  AddGap(DotAr[DotNum].X, Yq, MaX, Yq);
end;
Bt:= 2;
While Bt<DotNum do
begin
  q:= TopL;
  RightMar:= DotAr[Bt].X;
  While q<>Nil do
  begin
    if Not q^.Dashed then
    if q^.UpView<>XySec then
    begin
      if (((q^.Xs>RightMar) and
           (q^.Xs<DotAr[Bt+1].X))
          and (q^.Ys<DotAr[Bt].Y)) then
      begin
        RightMar:= q^.Xs;
        Xq:= q^.Xe;
        YCur:= q^.Ys;
        YCur2:= q^.Ye;
      end;
      if (((q^.Xe>RightMar) and
           (q^.Xe<DotAr[Bt+1].X))
          and (q^.Ye<DotAr[Bt].Y)) then
      begin
        RightMar:= q^.Xe;
        Xq:= q^.Xs;
        YCur:= q^.Ys;
        YCur2:= q^.Ye;
      end;
    end;
    q:= q^.Next;
  end;
  q:= TopL;
  LeftMar:= DotAr[Bt+1].X;
  While q<>Nil do
  begin
    if Not q^.Dashed then
    if q^.UpView<>XySec then
    begin
      if (((q^.Xs<LeftMar) and
           (q^.Xs>DotAr[Bt].X))
           and (q^.Ys<DotAr[Bt+1].Y)) then
      LeftMar:= q^.Xs;
      if (((q^.Xe<LeftMar) and
           (q^.Xe>DotAr[Bt].X))
           and (q^.Ye<DotAr[Bt+1].Y)) then
      LeftMar:= q^.Xe;
    end;
    q:= q^.Next;
  end;
  if LeftMar<>RightMar then
  AddGap(LeftMar, Yq, RightMar, Yq)
```

```
    else
    begin
      if Trunc(Abs((YCur-YCur2)/(Xq-RightMar)))<>
         Trunc(Abs((DotAr[Bt+1].Y-YCur2)/
                   (DotAr[Bt+1].X-Xq))) then
      AddGap(LeftMar, Yq, DotAr[Bt].X, Yq)
      else
      AddGap(DotAr[Bt+1].X, Yq, RightMar, Yq);
    end;
    Inc(Bt, 2);
  end;
  Ec:= False;
  if GapNum<>0 then
  begin
    for Bt:= 1 to GapNum-1 do
    begin
      for Xq:= Bt+1 to GapNum do
      if MinF(GapArray[Xq].X1, GapArray[Xq].X2)<
         MinF(GapArray[Bt].X1, GapArray[Bt].X2) then
      begin
        Yy:= GapArray[Xq];
        GapArray[Xq]:= GapArray[Bt];
        GapArray[Bt]:= Yy;
      end;
    end;
    Bt:= 1;
    While Bt<=GapNum do
    begin
      Toggle:= True;
      Repeat
        SetColor(White);
        q:= TopL;
        LeftMar:= GetMaxY;
        RightMar:= GetMaxY;
        While q<>Nil do
        begin
          if Not q^.Dashed then
          if XySec<>q^.UpView then
          begin
            if q^.Xs=GapArray[Bt].X1 then
            if q^.Ys<LeftMar then
            LeftMar:= q^.Ys;
            if q^.Xe=GapArray[Bt].X1 then
            if q^.Ye<LeftMar then
            LeftMar:= q^.Ye;
            if q^.Xs=GapArray[Bt].X2 then
            if q^.Ys<RightMar then
            RightMar:= q^.Ys;
            if q^.Xe=GapArray[Bt].X2 then
            if q^.Ye<RightMar then
            RightMar:= q^.Ye;
          end;
          q:= q^.Next;
        end;
        if LeftMar=GetMaxY then LeftMar:= YCc;
        if RightMar=GetMaxY then RightMar:= YCc;
        Yq:= Trunc((MiY+175)/2);
```

```
Line(GapArray[Bt].X1, LeftMar,
     GapArray[Bt].X1, Yq-10);
Line(GapArray[Bt].X2, RightMar,
     GapArray[Bt].X2, Yq-10);
Line(GapArray[Bt].X1, Yq, GapArray[Bt].X2,
     Yq);
Xq:=GapArray[Bt].X1;
XCc:= GapArray[Bt].X2;
if GapArray[Bt].X2<Xq then
begin
  Xq:= GapArray[Bt].X2;
  XCc:= GapArray[Bt].X1;
end;
Line(Xq, Yq, Xq+9, Yq+2);
Line(Xq+9, Yq+2, Xq+9, Yq-2);
Line(Xq+9, Yq-2, Xq, Yq);
Line(XCc, Yq, XCc-9, Yq+2);
Line(XCc-9, Yq+2, XCc-9, Yq-2);
Line(XCC-9, Yq-2, XCc, Yq);
if Toggle then
begin
GetMem(Pp, ImageSize(20, 133, 196, 164));
GetImage(20, 133, 196, 164, Pp^);
ClearRegion(26, 139, 196, 164);
FillRegion(20, 133, 190, 158, Yellow);
GetMem(Pq, ImageSize(70, 133, 145, 158));
GetImage(70, 133, 145, 158, Pq^);
SetColor(Red);
Str(Bt, Ss);
OutTextXY(35, 144, 'l'+Ss+' :');
OutTextXY(155, 144, 'µ');
SetColor(Black);
if XySec then
Str(GapArr[Bt].X1, Ss)
else
Str(GapArr[Bt].Y1, Ss);
Format(Ss);
OutTextXY(70, 145, Ss);
Okk:= False;
Repeat
  GraphRead(70, 145, NumLen, Ss, Ec, Pq);
  if Ss='' then
  if XySec then
  Str(GapArr[Bt].X1, Ss)
  else
  Str(GapArr[Bt].Y1, Ss);
  if Not Ec then
  begin
    Val(Ss, Tmm, Cd);
    if (Cd<>0) or (Tmm<=0) then
    begin
      PutImage(70, 133, Pq^, NormalPut);
      if XySec then
      Str(GapArr[Bt].X1, Ss)
      else
      Str(GapArr[Bt].Y1, Ss);
      Format(Ss);
```

```
            OutTextXY(70, 145, Ss);
          end
          else
          begin
            if XySec then
            GapArr[Bt].X1:= Tmm
            else
            GapArr[Bt].Y1:= Tmm;
            Okk:= True;
          end;
        end;
      Until Okk or Ec;
      PutImage(20, 133, Pp^, NormalPut);
      FreeMem(Pq, ImageSize(70, 133, 145, 158));
      FreeMem(Pp, ImageSize(20, 133, 196, 164));
      end;
      Toggle:= Not Toggle;
    Until Toggle;
    if Ec then Bt:= GapNum+1
    else
    begin
      Tmm:= 0;
      for Xq:= 1 to Bt do
      if XySec then
      Tmm:= Tmm+GapArr[Xq].X1
      else
      Tmm:= Tmm+GapArr[Xq].Y1;
      if XySec then
      Cc:= Lngth[1]
      else
      Cc:= Lngth[2];
      if Tmm>=Cc then
      begin
        DataErr(1);
        Bt:= 0;
      end;
      Inc(Bt);
    end;
  end;
end;
if Not Ec then
begin
  Cc:= 0;
  Yq:=Trunc((MaY+425)/2);
  Line(MiX, YCur, MiX, Yq);
  for Bt:= 1 to GapNum do
  if(MinF(GapArray[Bt].X1,GapArray[Bt].X2)<>MiX)
and (MaxF(GapArray[Bt].X1, GapArray[Bt].X2)<>MaX)
then
  begin
    if GapArray[Bt].X1<GapArray[Bt].X2 then
    begin
      Xq:= GapArray[Bt].X1;
      YCur2:= GapArray[Bt].Y1;
    end
    else
```

```
begin
  Xq:= GapArray[Bt].X2;
  YCur2:= GapArray[Bt].Y2;
end;
Scan(XySec, Xq, YCur2);
Toggle:= True;
LeftMar:= Trunc((MaY+Yq)/2);
Repeat
  SetColor(White);
  Line(Xq, YCur2, Xq, Yq);
  Line(MiX, LeftMar, Xq, LeftMar);
  Line(MiX, LeftMar, MiX+7, LeftMar+2);
  Line(MiX+7, LeftMar+2, MiX+7, LeftMar-2);
  Line(MiX+7, LeftMar-2, MiX, LeftMar);
  Line(Xq, LeftMar, Xq-7, LeftMar+2);
  Line(Xq-7, LeftMar+2, Xq-7, LeftMar-2);
  Line(Xq-7, LeftMar-2, Xq, LeftMar);
  if Toggle then
  begin
 GetMem(Pp, ImageSize(20, 133, 196, 164));
 GetImage(20, 133, 196, 164, Pp^);
 ClearRegion(26, 139, 196, 164);
 FillRegion(20, 133, 190, 158, Yellow);
 GetMem(Pq, ImageSize(70, 133, 145, 158));
 GetImage(70, 133, 145, 158, Pq^);
 SetColor(Red);
 Str(Bt, Ss);
 OutTextXY(35, 144, 'l'+Ss+' :');
 OutTextXY(155, 144, 'µ');
 SetColor(Black);
 Str(Cc, Ss);
 OutTextXY(70, 145, Ss);
 Okk:= False;
 Repeat
   SetColor(Black);
   GraphRead(70, 145, NumLen, Ss, Ec, Pq);
   if Ss='' then Str(Cc, Ss);
   if Not Ec then
   begin
     Val(Ss, Tmm, Cd);
     if ((Cd<>0) or (Tmm<=Cc)) or
        (((Tmm>=Lngth[1]) and XySec) or
        ((Tmm>=Lngth[2]) and Not XySec)) then
     begin
       if Cd=0 then
       DataErr(2);
       PutImage(70, 133, Pq^, NormalPut);
       SetColor(Black);
       Str(Cc, Ss);
       Format(Ss);
       OutTextXY(70, 145, Ss);
     end
     else
     begin
          Cc:= Cc+Tmm;
          if XySec then
```

```
                begin
                GapArr[Bt].X2:= GapArr[Bt].X1+Tmm;
                GapArr[Bt].X1:= Tmm;
                end
                else
                begin
                  Tmm:= Lngth[2]-Tmm;
                  GapArr[Bt].Y2:= Tmm;
                  GapArr[Bt].Y1:= Tmm-GapArr[Bt].Y1;
                end;
                Okk:= True;
            end;
          end;
       Until Okk or Ec;
       PutImage(20, 133, Pp^, NormalPut);
       FreeMem(Pq, ImageSize(70, 133, 145, 158));
       FreeMem(Pp, ImageSize(20, 133, 196, 164));
       end;
       Toggle:= Not Toggle;
     Until Toggle;
   end
   else
   if MinF(GapArray[Bt].X1, GapArray[Bt].X2)=MiX
   then begin
     if XySec then
     begin
       GapArr[Bt].X2:= GapArr[Bt].X1;
       GapArr[Bt].X1:= 0;
     end
     else
     begin
       GapArr[Bt].Y2:= Lngth[2];
       GapArr[Bt].Y1:= Lngth[2]-GapArr[Bt].Y1;
     end;
   end
   else
   begin
     if XySec then
     begin
       GapArr[Bt].X2:= Lngth[1];
       GapArr[Bt].X1:= Lngth[1]-GapArr[Bt].X1;
     end
     else
     begin
       GapArr[Bt].Y2:= GapArr[Bt].Y1;
       GapArr[Bt].Y1:= 0;
     end;
   end;
   Line(MiX, YCur, MiX, Yq);
end;
if Not Ec then
if XySec then
begin
  SetColor(White);
  Line(30, MaY, MiX, MaY);
  Line(MiX, MaY, MiX-9, MaY-2);
  Line(MiX-9, MaY-2, MiX-9, MaY+2);
```

```
 Line(MiX-9, MaY+2, MiX, MaY);
 GetMem(Pp, ImageSize(20, 102, 196, 164));
 GetImage(20, 102, 196, 164, Pp^);
 ClearRegion(26, 109, 196, 164);
 FillRegion(20, 102, 190, 158, Yellow);
 GetMem(Pq, ImageSize(70, 103, 145, 125));
 GetImage(70, 103, 145, 125, Pq^);
 SetColor(Red);
 OutTextXY(35, 115, 'Ux :');
 OutTextXY(155, 115, 'µ');
 OutTextXY(35, 142, 'Uy :');
 OutTextXY(155, 142, 'µ');
 SetColor(Black);
 Str(StartX, Ss);
 Format(Ss);
 OutTextXY(70, 115, Ss);
 Str(StartY, Ss);
 Format(Ss);
 OutTextXY(70, 142, Ss);
 Okk:= False;
 Repeat
 GraphRead(70, 115, NumLen, Ss, Ec, Pq);
 if Ss='' then Str(StartX, Ss);
 if Not Ec then
 begin
  Val(Ss, Tmm, Cd);
  if (Cd<>0) or (Tmm<0) then
  begin
    PutImage(70, 103, Pq^, NormalPut);
    Str(StartX, Ss);
    Format(Ss);
    OutTextXY(70, 115, Ss);
  end
  else
  begin
    PutImage(70, 103, Pq^, NormalPut);
    Format(Ss);
    OutTextXY(70, 115, Ss);
    StartX:= Tmm;
    Okk:= True;
  end;
end;
Until Okk or Ec;
SetColor(White);
Line(30, MaY, MiX, MaY);
Line(MiX, MaY, MiX-9, MaY-2);
Line(MiX-9, MaY-2, MiX-9, MaY+2);
Line(MiX-9, MaY+2, MiX, MaY);
if Not Ec then
begin
  Line(MiX, 405, MiX, MaY);
  Line(MiX, MaY, MiX-2, MaY+9);
  Line(MiX-2, MaY+9, MiX+2, MaY+9);
  Line(MiX+2, MaY+9, MiX, MaY);
  Okk:= False;
  Repeat
     SetColor(Black);
```

```
            GraphRead(70, 142, NumLen, Ss, Ec, Pq);
            if Ss='' then Str(StartY, Ss);
            if Not Ec then
            begin
              Val(Ss, Tmm, Cd);
              if (Cd<>0) or (Tmm<0) then
              begin
                PutImage(70, 134, Pq^, NormalPut);
                Str(StartY, Ss); Format(Ss);
                OutTextXY(70, 142, Ss);
              end
              else
              begin
                StartY:= Tmm; Okk:= True;
              end;
            end;
          Until Okk or Ec;
          SetColor(White);
          Line(MiX, 405, MiX, MaY);
          Line(MiX, MaY, MiX-2, MaY+9);
          Line(MiX-2, MaY+9, MiX+2, MaY+9);
          Line(MiX+2, MaY+9, MiX, MaY);
          end;
        PutImage(20, 102, Pp^, NormalPut);
        FreeMem(Pq, ImageSize(70, 103, 145, 125));
        FreeMem(Pp, ImageSize(20, 102, 196, 164));
        end;
        if Not Ec then
        begin
           for Bt:= 1 to GapNum do
           if XySec then
           begin
             GapArr[Bt].X1:= GapArr[Bt].X1+StartX;
             GapArr[Bt].X2:= GapArr[Bt].X2+StartX;
           end
           else
           begin
             GapArr[Bt].Y1:= GapArr[Bt].Y1+StartY;
             GapArr[Bt].Y2:= GapArr[Bt].Y2+StartY;
           end;
           if XySec then
           begin
             GapSum[1]:= GapNum;
             XBool:= True;
           end
           else
           begin
             GapSum[2]:= GapNum;
             YBool:= True;
           end; end; end; end;
  end;
  SetWriteMode(NormalPut); Delay(150);
  end
  else
  DBox(200, 300,
       'İlk Olarak Önden Görünüş Ölçeklenmelidir!', Kk);
end;
```

ÖZGEÇMİŞ

Cüneyt SABIRCAN, 1968 yılında İstanbul'da doğdu. İlkokulu Bakırköy Pilot Cengiz Topel İlkokulunda, ortaöğretimi Muhittin Üstündağ İlköğretim okulunda tamamladıktan sonra, 1982 yılında Kabataş Erkek Lisesinde lise öğrenimine başladı. 1985 yılında liseyi iyi derece ile tamamladı ve aynı yıl İ.T.Ü. Elektrik-Elektronik Fakültesi Kontrol ve Bilgisayar Mühendisliği bölümünde lisans öğrenimine başladı. 1989 yılında bu bölümden iyi derece ile mezun olduktan sonra yüksek lisans giriş sınavında başarı gösterdi ve 1989-90 sezonunda yüksek lisans hazırlık sınıfını birincilikle bitirerek yüksek lisans öğrenimine başladı. 1990 yılından beri Kontrol ve Bilgisayar Mühendisliği Bölümü, Bilgisayar Bilimleri A.B.D.'da araştırma görevlisi olarak çalışmaktadır.