İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

3-D ROBOT SIMULATION

M.Sc. Thesis by

Şerif ADALI

(F0498Y189)

Date of submission 11 June 2001

Date of defence examination 26 June 2001

Supervisor(Chairman) Doç.Dr. Coşkun SÖNMEZ

Members of the Examining Committee Prof. Dr. Bülent ÖRENCİK

Doç. Dr. Seta BOĞOSYAN

JUNE 2001

Foreword

This thesis is based on a 3-D robot simulator for the industrial robot Mitsubishi Movemaster EX that is used as the lecture tool for Robotics lessons in the Control & Computer Engineering department of Istanbul Technical University. The tool is an open source for other developers who would like to modify it for other robots.

The research phase really took a long time ,cause it is hard to find enough books about OPENGL. while avaliable ones are really complicated and mostly based on the C language. The internet was a good source for finding required documents and usefull code samples.tutorials and components for the project.Most of the libraries about OPENGL and technical data about the Movemaster EX is collected from the web.

I would like to thank all people who helped me during the development of the Robot Simulation software ,while special thanks goes to my thesis supervisor Doç.Dr. Coşkun Sönmez.for not only his support on the topic ,but also for his support in daily life.

June 2001

Şerif ADALI

Contents

List of Symbols

OPENGL	: Open Graphics Library
2-D/3-D	: 2/3 Dimensional Graphics
VCL	: Visual Component Library
CIME	: Computer Integrated Manufacturing and Engineering system
DIRECTX	: Microsoft's total solution to multimedia
CAD/CAM	: Computer aided design/manufacturing
GLSCNE	: Opengl library for Borland Delphi.
RGB	: (Red.Green,Blue)

List of Figures

List of Tables

3-D ROBOT SIMULATION-(SUMMARY)

A robot device is an instrumented mechanism used in science or industry to take the place of the human being.Robots are mechanical devices which can be programmed to perform some task of manipulation under automatic control.Most industrial robots are degisned to move materials ,parts,tools,or specalized devices through programmed motions for the performance of a variety of task.

This thesis consists of writing a 3-D simulation for the industrial robot Mitsubishi Movemaster EX (RV-M1). The main aspects were to get detailed information on the kinematics and programming of the Robot.The simulator must be capable of effectively simulating the motions of the real robot as much as possible. The Graphical Display and User Interface should be able to allow users to observe and interact effectively with the simulation. It is visualized in real time 3-D graphics which tries to achieve the same evironmental picture of the real-life platform .This platform is generally called the virtual production system and the objects are evaluated on the basis of the simulation results. User created models are formed through the graphical user interface. The user can easy add objects to the working environment and visa versa.And models can be stored in a model file so that different models including different objects can be loaded into the simulation software at anytime desired. The programs are written in the robot programming language which is based on a simple basic-like language which includes basic parametric commands and some structural commands as program control statements. A simulation echoes the behavior of the real world system as it changes from one state to another through time. A computer simulation provides a tool for studying real world systems in order to predict their behavior under a variety of conditions. Systems to be studied may already exist, or they may be on the drawing board. A simulation is an important tool that you can use to predict the course and results of certain actions,understand why observed events occur,identify problem areas before implementation,explore the effects of modifications,confirm that all variables are known,evaluate ideas and identify inefficiencies.gain insight and stimulate creative thinking ,communicate the integrity and feasibility of your plans.

3 BOYUTLU ROBOT SİMÜLASYONU – (ÖZET)

Robot teknolojisi,teknolojinin diğer dalları gibi hızla gelişmektedir.Günümüzde robotlar uzay araştırmaları, havacılık, araba otomasyonu gibi çok çeşitli branşlarda kullanılmaktadırlar.Bu tezde Mitsubishi firması tarafından geliştirmiş **Movemaster EX (RV-M1)** endüstriyel robot'u için 3-Boyutlu simülasyon yazılmıştır.Projenin gerçeklenmesi için en uygun uygulama geliştirme dili olarak yeni nesil nesneye yönelik programlama dili olan pascal tabanlı Delphi seçilmiştir.

I-GENEL BİGLİ

3-Boyutlu simülasyon kelime anlamına paralel olarak robot'un bulunduğu ortamın ve robot'a ait tüm hareketlerin ve davranışların gerçeğe en yakın şekilde gerçeklemesidir. Doğru kalibre edilmiş bir robotun kullanılması durumunda, simülasyon hareketleriyle k robot hareketleri aynı sonuçları vermektedir. Proje konusu seçilmesi aşamasında birçok bilgisayar mühendisliği konusunu içermesi açısından 3-boyutlu simülasyon konusu seçilmiştir.En önemli nedenler bilgisayar simülasyonu,3-Boyutlu bilgisayar grafiği,derleyici tasarımı,robot teknolojisi gibi konuları içermesidir. Bahsedilen bu konular bilgisayar mühendisliğinde ağrılıklı olarak kullanılmakta ve yazılım mühendisleri tarafından popüler ve güncel konular olarak kabul edilmektedir.

3-Boyutlu grafik programla temeli olarak Silicon Graphics firması tarafından geliştirilen ve grafik endüstrisinde yaygın olarak kullanılan OPENGL teknolojisi kullanılmıştır. Tez ile gerçeklenen simülasyonun özelliği , robot ile hiç bir şekilde bir bağlantı olmadan ve hatta robotun bulunduğu ortamda bulunmadan (örneğin ev), robotun istenildiği şekilde programlanabilmesi ve kaynak kodun çalıştırılarak robot hareketlerinin üç boyutlu grafik ortamında gözlenebilmesidir.Proje ileride daha da geliştirilerek üniversitemizde verilmekte olan Robot Teknolojileri lisans ve yüksek lisans derslerinde öğretim aracı olarak kullanılması için tasarlanmıştır. Bu sayede dersi alan öğrenciler bir diğer öğrencinin robot ile olan işini beklemeyecekler hatta artık robot üzerinde çalışmak için laboratuvar'a gelmek zorunda kalmayacaklardır.Bu projenin en önemli özelliği olarak gösterilebilir. Simülasyon yazılımı sayesinde doğru açı ve konum hesaplamaları sırasında ,robot'a gelebilecek zarar ve ortaya çıkabilecek hatalar önceden giderilmiştir olacaktır.

II- 3-BOYUTLU ROBOT SİMULASYONU

Tez'in temelinde Delphi 5 (Borland) için geliştirilmiş ve Delphi 6 ile standart olarak gelecek olan OPENGL komponent paketi olan GLSCENE paketi kullanılmıştır.

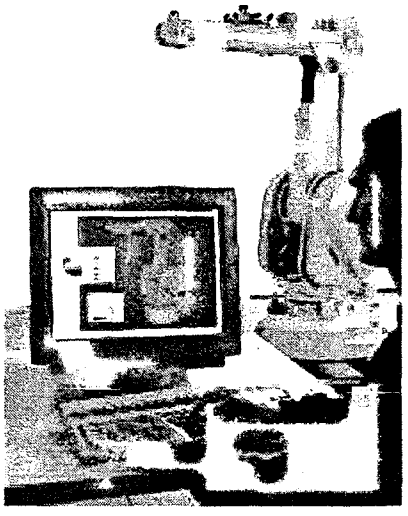
Projenin Amaçları

Günümüzde gelişmekte olan mikroişlemci ve yarı-iletken teknolojilerine paralel olarak gelişmekte olan yüksek kapasiteli görüntü kartları sayesinde artık çok daha hızlı ve kaliteli bilgisayar grafikleri elde edilmektedir. 3-Boyutlu grafik yıllardır bilgisayar mühendislerinin üzerinde çalıştıkları en popüler konulardan biridir.Konunun önemli olmasının en önemli nedeni çok geniş kullanım alanı olmasıdır. Günümüzde bilgisayar grafikleri kullanım alanları oyunlardan,simülasyonlara,film sektöründen bilgisayar destekli tasarım ve üretim gibi bir çok alanı kapsamaktadır.

Simülasyondaki en önemli amaç gerçek dünyadaki ortamın bilgisayar ortamında gerçeğe en yakın şekilde gerçeklenmesidir.Bunun için gerçek dünyadaki ortamın çok detaylı şekilde tüm olasılıklar ele alınarak incelenmesi gerekmektedir.Bir simülasyon sayesinde temel olarak aşağıda sıraladığımız kazançlar sağlanır.

- Olası durumların önceden kestirilmesi
- Meydana gelebilecek olan durumların nedenlerini bulmak
- Uygulamadan önce hataları görebilme
- Yapılan değişiklerin anında test edebilmek

Tez bünyesinde gerçeklenen simülasyon bilgisayar dünyasında hatsız simülasyon (off-line simulation) olarak adlandırılır. Bunun nedeni simülasyon sırasında programcıya gerekli olan sadece simülasyon yazılımıdır. Bu sayede hiçbir şekilde robot'a çevre birimler ile bağlı olmak gerekmektedir. İstenildiğinde ev ortamında da istenilen kaynak kodu yazılabilmeli ve 3-Boyutlu simülatör'de çalıştırılabilmelidir.



Şekil –1 : Örnek Robot Simülasyonu

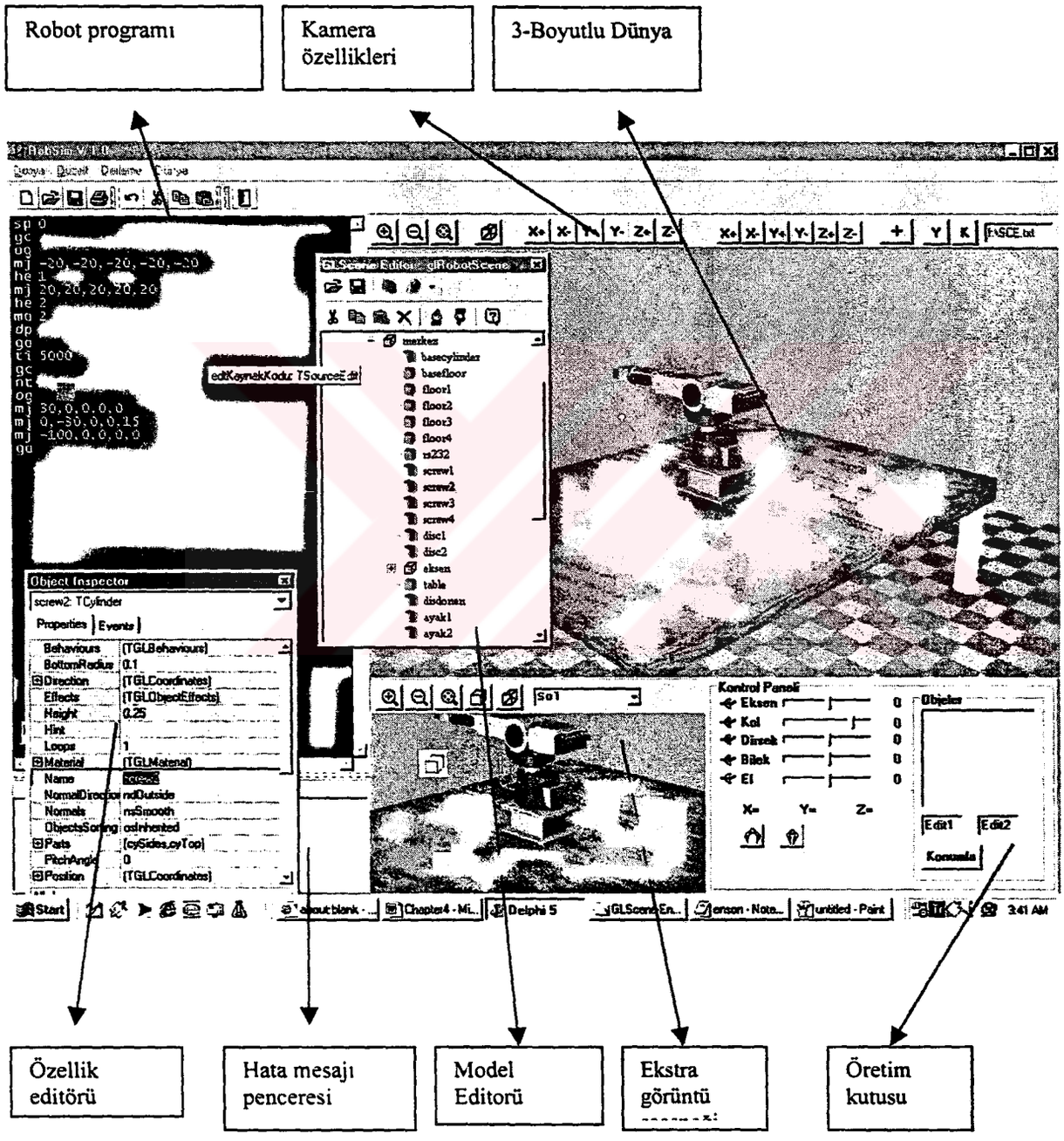
Simülatörler genellikle eğitim alanlarında yaygın olarak kullanılmaktadırlar.Bunlarda en çok bilinenleri Savunma ve havacılık sanayii'nde kullanılmakta olan uçuş simülatörleridir. Bu simülatörler gerçek bir uçuş sırasında karşılabilecek tüm durumları en gerçekçi şekilde yaratmaktadır.Simülatörler için ilk etapta harcanan paralar büyük olarak gözükse de daha uzun vadede çok büyük kazançlar sağlamaktadır. Amerika'da bulunan özel uçuş sertifikası veren okulların %80 de Microsoft Flight Simulator'un kullanılması herhalde bilgisayar simülasyonlarının ne derece önemli olduğunu daha iyi anlamamızı sağlayacaktır.

Robot programlarının birebir olarak robot'a yüklendiğinde çalışması için tüm robot komutları simülatöre yüklenmeye çalışılmıştır. Komutların çözülmesi ve parametrelerinin ayrılması için bir çözücü mekanizması yazılmıştır. Ayrıca komutların ve parametrelerin farklı renklerde ve yazı tiplerinde ekranda yansıtılması için TMemo adı verilen temel Delphi komponentini üzerine ek özellik ve yöntemler eklenerek yeni bir derleyici yazı kutusu yaratılmıştır. Bu komponent başka yazılımcılar tarafından da kolaylıkla derleyici tasarımları ile ilgili bir programda kullanılabilir.

Robot simülasyonun hatasız bir şekilde çalışabilmesi için gerekli olan şartlar vardır. Bunun için robot'un hareket olasılıkları,robot kinematiği ve oluşacak durumlar karşısında robotun nasıl davranacağı detaylı olarak saptanmalıdır.

İleri Yön Kinematik sayesinde , belli eklemler açıları sonucunda robotun konumunun hesaplanması söz konusudur,aynı şekilde geri yönde hesaplamaların yapılması gerekmektedir.Bu iki yöntem sayesinde istenilen konuma gidilmesi için gerekli eklemlerde kaç derecelik açılarda döndürme yapılması,yada belirtilen bir konuma gidilmesi için gerekli olan açıların hesaplanmasında kullanılmaktadır. Bu sayede pozisyon hatalarının en aza indirgenmesi ve robot 'a gelecek zararların en aza indirgenmesi sağlanmaktadır.

III-UYGULAMA



Şekil –2 : Örnek Simülasyon Ekranı

Kamera Ayarları sayesinde X,Y,Z eksenlerinde kamera'nın döndürülmesi sağlanmaktadır.Özellik Editörü kullanılarak 3-Boyutlu modelleme istenildiği şekilde kullanıcı tarafından düzenlenebilmektedir.Robot kaynak kodu solda görünen kutuda yazılmakta ve F8 tuşu kullanılarak çalıştırılmaktadır.Oluşabilecek hata mesajları ise hemen altındaki beyaz mesaj kutusunda belirecektir.Ekstra görüntü seçeneği sayesinde robot hareketleri önceden ayarlanmış konumlardan kolaylıkla izlenebilmektedir. Burada amaç genel simülasyon ekranındaki görüş açısını bozmadan robotun başka ek bir görüntü içerisinde izlenebilmesidir.Eğitim kutusu sayesinde robot'un her eklemi ve eli ile ilgili tüm fonksiyonlar manuel olarak kullanılabilir.

Eş zamanlama tekniği

Simülasyon kodlanmasında kullanılan en önemli teknik eş zamanlama (Thread) tekniğidir. Bu teknik sayesinde bir den fazla işlem(process) aynı anda çalıştırılmaktadır. Bunun için delphi dilinde kanalları iyi bir şekilde denetlememizi sağlayan TThread sınıfını kullanılmıştır.

```
//KOL
 TKolThread = class (TThread)
 private
  rKolAngle:Real;
 protected
  procedure SetKolAngle;
  procedure Execute; override;
 public
  rKolAcisi :Real;
  rKolilkAci:Real;
  rAciYon  :String;
  nRobotHizi :Real;
 end;
```

Yukarıdaki örnekte kol hareketi için kullanılan kanal kodunun TThread sınıfından türetilmesi gösterilmektedir.

CHAPTER-1

3-D ROBOT SIMULATION

Robot technology is a rapidly developing field during last 30 years. Nowadays robots are widely used in numerous applications varying from space researches to car automation.As a definition ,a robot device is an instrumented mechanism used in science or industry to take the place of the human being.Robots are also defined as mechanical devices which can be programmed to perform some task of manipulation or locomotion under automatic control.Most industrial robots are designed to move materials ,parts,tools,or specalized devices through variable programmed motions for the performance of a variety of task.

To comply with increasing levels of competition industrial companies use flexible automated integrated production systems more and more. Programmable machines such as robots, cnc-machines and plc's are a natural part of any automated production system (Computer Integrated Manufacturing and Engineering system, CIME) where Robot simulations took a high-prioriy palce in the production line as tool for planning in the virtual production system with 3D visualization.

This thesis is based on a 3-D simulation for the industrial robot called Mitsubishi Movemaster EX (RV-M1). The primary research was to find out detailed information on the kinematics and programming of the Robot. As a software engineer it was an important decision to chose which programming language to use for coding. Borland's Delphi 5 seemed to be a good solution for the project.The most powerful reason for chosing it was the ease of its visual environment.Delphi is a next generation object oriented visual language based on the standart pascal language. On the other hand pascal is widely known by most of the computer engineers as a base computing language.According to the project outline,the simulator must be capable of effectively simulating the motions of the real robot as much as possible. The simulator should be able to produce the Forward Kinematics for the robot as well as implementing the Inverse Kinematics . It should also be able to generate the Trajectory routes, implement a Forward Dynamic Algorithm, and implement simple control programs for the robot. The Graphical Display and User Interface should be able to allow users to observe and interact effectively with the simulation.

The next step was to choose the graphical environment for the project. This was without doubt the most critical part of the decision. There were two posssible approaches called OPENGL and DIRECTX . In the thesis the code is based on OPENGL code that is avaliable through the Delphi VCL library called the GLSCNE. GLScene briefly translated all OPENGL header files to Delphi Code so that the original code that's developed by SILICON GRAPHICS can exactly be used by the Delphi Coder. Borland is planning to ship Delphi 6 with the GLScene components for the 3-D geometry.The reason why OPENGL is very important is that the base 3-D environment will be created using the OPENGL objects and commands.Before starting the project some tests it was necessary to do test to see the performances of each graphical approach,which OPENGL seemed to pass the test.

The simulation is performed with basic geometry objects like cubes,cylinders. The simulation is visualized in real time 3-D graphics which tries to achieve the same evironmental scene of the real-life platform .This platform is generally called the virtual production system and the objects are evaluated on the basis of the simulation results. The models can be optimized through the model with the smallest error. After the achievement of a quality is reached the source code (robot code) are downloaded to the production system.Experiments can support a variety of different analysis focusing on criterias such as: equipment layout, production capacity, cycle time, and rearrangement costs.A well calibrated model can come close to 100% success with the actual working environment.The differences between the simulation model and the actual environment called the work cells is caused by non-calibrated robot .So there might be small differences between the positions of the actual robot and the simulated model.

User created models are formed through the graphical user interface. The user can easy add object to the working environment and visa versa.And these models can be stored in a model file so that different models including different objects can be loaded into the simulation software at anytime desired. The programs are written in the robot programming language which is based on a simple basic-like languge which includes parmetric commands and some structural commands as program control statements.

The model's object oriented design enables a large measure of openness for user built additions. The object oriented design provides a very open development environment where users easily can build specialized applications. The system is also avaliable and easy for development .

1.1 The aim of performing the project

3-D computer graphics using OPENGL

Nowadays in software engineering ,computer graphics specially 3-D stuff is becomming a popular brach day by day. By implementing 3-D world better and better, games,technical drawing packs, and other 3d stuff is getting more complicated and more detailed with a higher quality at the same time. By the rapid development in micro-chip technology so as the graphics adapters, more frames could be displayed with a higher rate of quality which causes a better reality of modelling the real-life objects and other real-time effects.

OpenGL is a software interface to graphics hardware. This interface consists of about 120 distinct commands, which you use to specify the objects and operations needed to produce interactive three-dimensional applications.

OpenGL is designed to work efficiently even if the computer that displays the graphics you create isn't the computer that runs your graphics program. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using. Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, you must build up your desired model from a small set of geometric primitive - points, lines, and polygons.

As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer. These objects are described as sequences of vertexes (which define geometric objects) or pixels (which define images). OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

OpenGL provides powerful set of drawing operations, and all higher-level drawing requires their use. Basic functions are as follows

1. Manipulating Images for Use in Texturing
2. Transforming Coordinates
3. Rendering Spheres, Cylinders, and Disks
4. Curves and Surfaces

Primitives and Commands

OpenGL draws primitives points, line segments, or polygons subject to several selectable modes. You can control modes independently of one another; that is, setting one mode doesn't affect whether other modes are set . Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls.

Primitives are defined by a group of one or more vertexes. A vertex defines a point, an endpoint of a line, or a corner of a polygon where two edges meet. Data consisting of vertex coordinates, colors, normals, texture coordinates, and edge flags is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way. The only exception to this rule is if the group of vertexes must be clipped so that a particular primitive fits within a specified region. In this case, vertex data may be modified and new vertexes created. The type of clipping depends on which primitive the group of vertexes represents.

Commands are always processed in the order in which they are received, although there may be an indeterminate delay before a command takes effect. This means that each primitive is drawn completely before any subsequent command takes effect. It also means that state-querying commands return data that is consistent with complete execution of all previously issued OpenGL commands.

GLSCENE IN DETAIL

GLScene is an OpenGL based 3D graphics library for Delphi. It provides visual components and objects allowing description and rendering of 3D scenes in an easy, no-hassle, yet with a powerfull manner.GLScene is not just an OpenGL utility library, it has grown to become a set of founding classes for a generic 3D engine with Rapid Application Development . Borland has translated all OPENGL C++ Header files to delphi libraries so that it has the same properties and same performance when compared to the original SILICON GRAPHICS's opengl applications.

GLScene allows you to quickly design and render 3D scenes without having to learn the intricacies of OpenGL, if you know how to design a TForm, you'll easily master the basic operations of TGLScene.The rest of the OPENGL satatements are created automatically at the

background by the GLSCENE code. With the ease of use, it is a rapid application development tool for delphi who want to develop opengl based CAD/CAM ,Graphics Packs,games and other technical stuff like simulations and so on.

Here are the highlights of the pack

- **Scene description**
    - Hierarchical objects structure helps to define parent and child object structure to define groups of objects so that they could act like one single object .The user can add as many objects as memory allows.
    - Interactive scene management is possible using the user-friendly GLSCENE editor which holds all objects included in the SCENE.
    - By the object oriented programming manner each object uses the same rotating,shifting,pitching algoritms causing the user easily rotate and move objects whenever needed.
    - Predefined objects (cube ,cylinder) are avaliable and ready to use using the GLSCENE editor.
    - For composite structures(grouping) can be done using the special goemetry objects which developed by the SILICON GRAPHICS called a dummy cube.
    - It directly supports classes for accessing OpenGL .This is a great opportunity for pure OPENGL coders.
    - camera and light objects that can be used anywhere in a scene objects hierarchy
- **Materials**
    - The materials are easy to use and can be easily stored in the memory.
    - Material library helps to share and reuse materials through the same code,especially when a material has got to be used more than once by multiple objects at the same time.
    - It supports for all OpenGL texture formats .

- Contains more than 150 predefined colors like clrCornflowerBlue or clrCoolCopper in addition to standard colors and direct RGB specifications
- It is easy to use texture movement and scaling properties, independently from texture coordinates
- It also supports 32 bits Bitmap support class .

- **Rendering**
  - automatically uses the hardware OpenGL driver if available
  - a well working camera model using focal length and targeting
  - multiple viewers for one or more scenes, easy change of view through camera selection
  - full screen support with dynamic resolution changes

- **Utilities**
  - Contains optimized geometry functions and utilities (vector, matrix...)
  - Has precise frame speed determination of the GLSCENE which can be used to optimize and calibrate the speed of the simulation depending on the sound card and the processor.
  - Contains an asynchronous timer (multi-threaded) for multi-threaded actions like moving all five joints of the robot arm simultaneously with different angles. [3]

Computer Simulation (Off-Line simulation)

A simulation echoes the behavior of the real world system as it changes from one state to another through time. A computer simulation provides a tool for studying real world systems in order to predict their behavior under a variety of conditions. Systems to be studied may already exist, or they may be on the drawing board.

Simulation provides a method for checking your understanding of the world around you and helps you to produce better results faster. A simulation is an important tool that you can use to:

- Predict the course and results of certain actions
- Understand why observed events occur
- Identify problem areas before implementation
- Explore the effects of modifications
- Confirm that all variables are known
- Evaluate ideas and identify inefficiencies
- Gain insight and stimulate creative thinking
- Communicate the integrity and feasibility of your plans

We know that simulators and simulations are used in a wide range of areas.The most known ones are.

-Space and Air Craft simulator
-Car simulators
-Robot Simulators
-Automation (Ex:Cars)

In the project , the environement and most of the most of the robot commands are written for the simulation. The type of simulation that is implemented in the project is called "Off-Line" simulation.In order to improve the efficiency of robotic automation in the Automotive Industry, interactive and graphics-based tools for planning which are called off-line simulations have been created.Most of the Car automations ,robot are tested for a long period using simulators to gain %100 success. This requires a long-range workout on positioning and other robot-specific actions.

Robot Programming (Writing a compiler for Movemaster EX)

As a computer engineer ,software is always an enjoyable field plus writing a compiler is also a hot topic . The simulation project required a simple compiler that it would compile the Movemaster code and run it on the 3-D environment. A robot program consists mainly of two parts: locations (position and alignment) and program logics (controller structures, communication, calculations). The program logics can effectively be developed off-line as effective debugging and simulation facilities are available here. The major part of movement commands can be created off-line by the reuse of CAD data and by interaction of the programmer.

Robot Kinematics

Before getting on with the project a lot of time is spent on the robot kinematics ,including its work space, abilities of each joint and other possible movements. This required to overview the technical and functional properties of the robot. Since different robot controllers pose different characteristics, specific controller models are required for an accurate simulation using these tools.Without detailed controller know-how, the implementation of models is very time-consuming.Hence, manufacturers know-how is a must for a rapid implementation of accurate models. However their integration is hindered by different interfaces in various simulation systems.

1.2 Reason for writing a robot simulation

The main project goal was to improve the simulation accuracy of industrial robot simulation systems in order to achieve a more realistic simulation of robot controllers.Generally people know robots as machines doing routine jobs ,like robots building a car.On the other hand there is a wide usage of robots in various fields.Think about the hubble space telescope, the pathfinder that is sent to mars, the robots used to maintain the hubble space telescope. All these robots are tested in simulators. For example the robot used to maintain the hubble is tested at an under-water simulators. All those examples clearly defines the importance of using simulators. Just imagine that 80% of the amateur pilots are trained using the Microsoft's Flight Simulator . After finishing those simulator lessons they start test flights. In a simulation you do not have to have an airplane or a car.Simulation means saving time and money. For a perfectly running simulation lots of test should be done .

1.3 Technical Aspects

In order to assure simulation accuracy and efficient implementation by original controller software the integrated controller software fulfils the requirements for an accurate simulation of

- motion behaviour,
- robot kinematics, and
- condition handling.

It has been proven that the deviation between simulated and real joint values is less than 0.001 radians which independent of given geometric data representations and user interface standards.

Availability

While designing and maintaining an production system the we need for tools to plan and operate the system. The simulation tools (Robot Off-line programming and Simulation systems) are used for this need. During projection and planning a virtual production system is developed in Robot Simulations. In the program experiments and analysis can be performed without the need for costly equipment. Experiments made in Robot Simulations can help to make decisions regarding if investments should be made out or not. In production the virtual production system can be calibrated and used for maintenance, supervision, data collecting and controlling.

Robot simulation is a model driven simulation system with 3D visualization. The program handles simulation and visualizing of the models. Any object that can be described using one or more of these components can be simulated in Robot Simulation. The object oriented design entails a natural openness for expansions and distributed objects. As a result of simulation's object oriented design, the spheres of application are numerous.

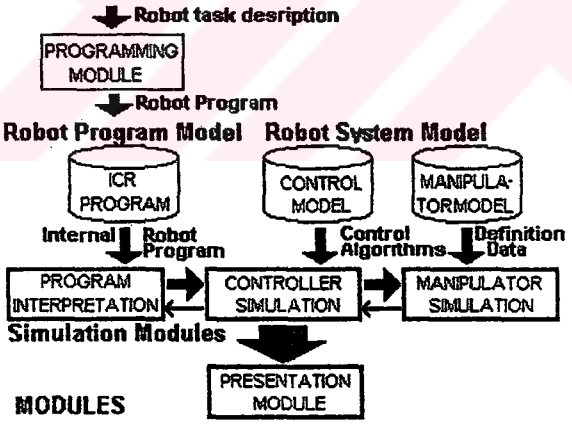


Figure 1.1: Simulation Model

1.4 Planning production

In production planning supported by simulation product design, robot programs, robot installation layout and cycle times can be optimized. This is often an iterative process, in which a diversity of equipment combinations, layouts, robot programs and product designs are tested together until an appropriate quality has been achieved.

In integrated production systems robots are often used in an interplaying process with different equipment such fixtures, conveyor belts, etc. in a workcell. The workcell layout is commonly a critical decision that can be very costly to change. In the process of workcell design which equipment to use and where to place it is determined. A part from this the cell must often be able to produce a diverse series of products and this in itself sets demands to the flexibility of the cell.

Through simulation experiments can be performed on virtual equipment models. The experiments allow for analyses of diverse layouts placing the focus on criterias such as the placing of equipment, production capacity, and cycle rates, costs etc. The quality of estimated cycle rates is determined by the quality and calibration of the models. Simulation provides precise dynamic simulations. A well calibrated model can come close to 100 % concordance to the actual workcell.

1.5 Design and optimization of robot programs

For off-line robot programming robot programs can be developed textually or interactively through the utilization of the 3D solid model of the workcell. This is illustrated in the figure below. The two methods are often used together as they both have different advantages. After this the program execution can be verified by simulation until a desirable quality has been reached. At the completition of this the robot programs can be transferred to the actual production system, after which few or no adjustments are necessary to get the production started.

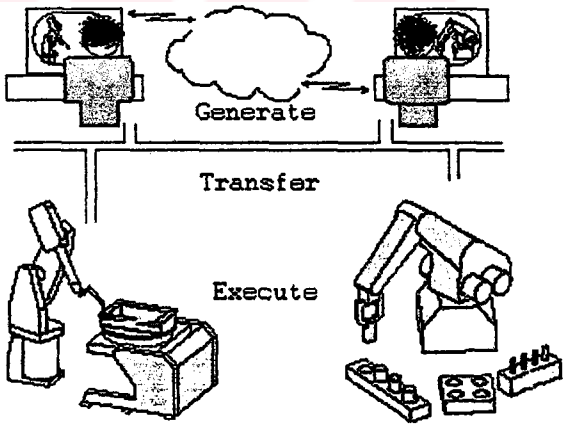


Figure 1.2: Robot simulation execution

1.6 Product design and optimization

The product design can be optimized for production on an existing or new production system. Any production system can be tested in the virtual production system. In this manner the production cost and effectiveness of the design can be analyzed. The virtual experiments provides the opportunity to decrease the time interlapse from design to production.

1.7 Supervision and data collecting

Through data collecting from the actual workcell a production system can be supervised. Data from the actual workcell is visualized alone or in interplay with the simulation.

Figure 1.3: Example simulation

The collecting of data has many different aspects, for instance the position of a robot, the internal communication in a workcell and so on (Ex:Collision tests). With the data taken into the program, simulation models can continuously be updated and then visualized. Data can also be collected and saved. The collected data can afterwards be analyzed through simulation and visualizing.

The program execution can be verified by simulation until a desirable program quality has been achieved. The robot programs can then be transferred to the actual production system after which little or no adjusting is necessary before the robots are in production. [5]

1.8 Construction of the robot

GLSCENE Editor is used for modelling robots and . Different combinations of the program, the controller and the manipulators can be analyzed through simulation. In simulation the

actual state of the robots can be visualized as 2D plot in time (for instance the angular position of a robot) or 3D plot in time (for instance position and orientation of the robot tool).

1.9 Education

In the education of engineers in robotics the need for a well structured understanding of what exactly a robot is. The robot model gives exactly this understanding. Model driven simulation is a valuable tool for understanding the properties of a robot.

1.10 Controller

The robot programs are generated partly in a text editor and partly interactively in the 3D visualization. The programs can then be debugged in simulation. When the programmes have been debugged they can be run the code through the actual robot.

1.11 Geometric models

Simulation model consists of a description of the robot mechanism .The model has a geometrical solid model, a kinematic model, and a dynamic model. The models can be built through the user interface. New models can be inserted.

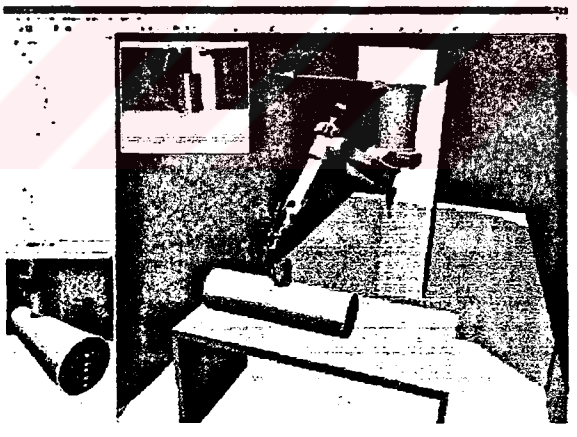


Figure 1.4: Example user interface

1.12 Graphical user interface

Simulation has a user-friendly graphical interface complete with windows, menus, icons and so forth. As well as standard window technique the geometry of the model is visualized in interactive 3D viewers. Using interaction with the 3D viewers the objects can be selected and manipulated based on the OPENGL technology. [6]

CHAPTER 2

MATRIX THEORY AND ROBOT KINEMATICS

2.1 Matrix theory

Matrices are linear functions for vectors. Consider U, V being vectors of any dimension. Then U := M*V is transforming one vector (V) into another (U). Here, we further assume the vectors being 3D vectors. A vector is not just a few numbers, it furthermore has a base defining the system :

Here, as usual the base is a right hand euclidian system :
E1 = X = (1,0,0) E2 = Y = (0,1,0) E3 = Z = (0,0,1)

The base vectors of a coordinate system are orthogonal to each other in respect of the applied scalar product (which is defined as <a,b> = sum(ai*bi) over i for this system), e.g the scalarproduct of the base vectors is zero for all combinations. There are other noneuclidian systems, such as the spherical one : (radius, longitude, latitude). Note that also here, the 3 basevectors are locally right angled. The length of a base vector is normalized to 1.

A vector defines a point in space with respect to the base.
P := x*E1 + y*E2 + z*E3
the (x,y,z) are called components of the vector and denote the partial content of each base vector.

The components are calculated formally as :
x:=<P,E1>, y:=<P,E2>, z:=<P,E3>
where <a,b> is again the scalar product, which in this case also is
|a|*|b|*cos(angle between), the product of the lengths times the cosine of the angle between.

Be V the old vector, M the matrix, and U the new vector :

U = M * V

or

|u1| |m11 m12 m13| |v1|

|u2| = |m21 m22 m23| * |v2|

|u3| |m31 m32 m33| |v3|

the calculation is done as :

u1 := m11*v1 + m12*v2 + m13*v3

u2 := m21*v1 + m22*v2 + m23*v3

u3 := m31*v1 + m32*v2 + m33*v3

a short notation : ui := mij*vj = { sum(mij*vj) over j }

We'll have a closer inspection. Use the base vectors E1,E2,E3 as input and we see that the columns in the matrix are the resulting vectors :

E1 -> (m11,m21,m31)

E2 -> (m12,m22,m32)

E3 -> (m13,m23,m33)

meaning : the columns show us how the base is transformed.

2.2 Matrix Multiplication

two matrices can be multiplied as:

H:=M*N

|h11 h12 h13| |m11 m12 m13| |n11 n12 n13|

|h21 h22 h23|:= |m21 m22 m23| * |n21 n22 n23|

|h31 h32 h33| |m31 m32 m33| |n31 n32 n33|

calculated as :

h11 := m11*n11 + m12*n21 + m13*n31

h12 := m11*n12 + m12*n22 + m13*n32

h13 := m11*n13 + m12*n23 + m13*n33

h21 := m21*n11 + m22*n21 + m23*n31

h22 := m21*n12 + m22*n22 + m23*n32

h23 := m21*n13 + m22*n23 + m23*n33

h31 := m31*n11 + m32*n21 + m33*n31

h32 := m31*n12 + m32*n22 + m33*n32

h33 := m31*n13 + m32*n23 + m33*n33

or

hij := mik * nkj = { sum(mik * nkj) over k }

note :matrices do generally not commute : N*M != M*N

2.3 Transformation Matrices

Although any non-singular matrix **M** represents a valid projective transformation, a few special matrices are particularly useful. These matrices are listed in the following paragraphs.

Translation

The call **glTranslate***(x, y, z) generates **T**, where:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and } \mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling

The call **glScale***(x, y, z) generates **S**, where:

$$\mathbf{S} = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and } \mathbf{S}^{-1} = \begin{bmatrix} \frac{1}{x} & 0 & 0 & 0 \\ 0 & \frac{1}{y} & 0 & 0 \\ 0 & 0 & \frac{1}{z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Notice that **S**-1 is defined only if *x*, *y*, and *z* are all nonzero.

2.4 Rotation

Rotational matrices belong to the few reversible transformations, their determinating value is 1, also meaning the length of a vector is preserved. There is one matrix for each rotation and each angle, only the anlge can be parametrized. The rotation is always around the origin. Off center rotations require first shifting the origin, then the object is rotated, then shifted back.

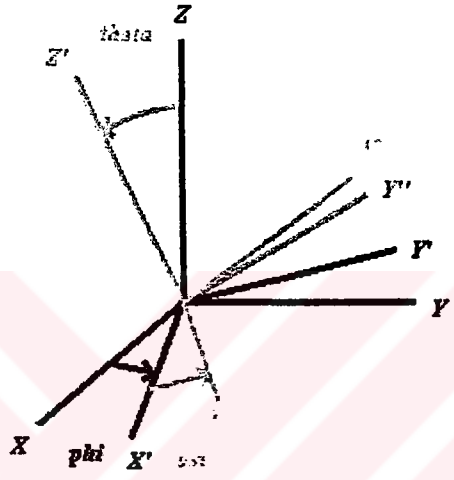


Figure 2.1: Rotation Around X,Y,Z

It is done as :

1. rotate around Z by the angle phi(blue), this gives (X'(blue),Y'(blue),Z(black))
2. rotate around X' by the angle theta(red), this gives (X'(blue),Y''(red),Z'(red))
3. rotate around Z' by the angle psi(green), this gives (X''(green),Y'''(green),Z'(red))

The call **glRotate***(*a*, *x*, *y*, *z*) generates **R** as follows.

Let v = (*x*, *y*, *z*)T, and **u** = v/||v|| = (*x*', *y*', *z*').

Also let

$$S = \begin{bmatrix} 0 & -z' & y' \\ z' & 0 & -x' \\ -y' & x' & 0 \end{bmatrix} \text{ and } M = uu^T + (\cos \alpha)\,(I - u\,u^T) + (\sin \alpha)\,S$$

Then

$$R = \begin{bmatrix} m & m & m & 0 \\ m & m & m & 0 \\ m & m & m & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ where } m \text{ represents elements from M, which is a 3x3 matrix.}$$

The **R** matrix is always defined. If $x=y=z=0$, then **R** is the identity matrix. You can obtain the inverse of **R**, **R**-1, by substituting -**a** for **a**, or by transposition.

The **glRotate*()** command generates a matrix for rotation about an arbitrary axis. Often, you're rotating about one of the coordinate axes; the corresponding matrices are as follows.

$$\text{glRotate*}(\alpha, 1, 0, 0): \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glRotate*}(\alpha, 0, 1, 0): \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glRotate*}(\alpha, 0, 0, 1): \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

As before, the inverses are obtained by transposition.

2.5 Implementing the Forward Kinematics Function

The Kinematics Function of the simulator simply returns a 4 x 4 transformation matrix as its output using a given set of joint angles and link parameters as input. This transformation matrix represent the position and orientation (the p_x,p_y,and $p_z$elements of the matrix shown below) of the end-effector with respect to the base frame of the manipulator.

The time required to perform Kinematics calculation is a always of big concern, due to the multiple additions and multiplications involving the transcendental functions, i.e., sines and cosines. Great effort was then made to avoid computing common terms over and over throughout the computations by making them available as a part of the standard library thereby reducing the amount of time spent.

The following equations shows how 12 elements of the transformation matrix were obtained, it is also hoped that it will successfully convey the issue of how time consuming Kinematics calculations could be.

$$r_{11} = c_1[c_{23}(c_4c_5c_6 - s_4s_6) - s_{23}s_5c_6] + s_1(s_4s_5s_6 + c_4s_6)$$

$$r_{21} = s_1[c_{23}(c_4c_5c_6 - s_4s_6) - s_{23}s_5c_6] - c_1(s_4c_5s_6 + c_4s_6)$$

$$r_{31} = -s_{23}(c_4c_5c_6 - s_4s_6) - c_{23}s_5c_6$$

$$r_{12} = c_1[c_{23}(-c_4c_5c_6 - s_4c_6) + s_{23}s_5s_6] + s_1(c_4c_6 - s_4c_5s_6)$$

$$r_{22} = s_1[c_{23}(-c_4c_5c_6 - s_4s_6) + s_{23}s_5s_6] - c_1(c_4c_6 - s_4c_5s_6)$$

$$r_{32} = -s_{23}(-c_4c_5c_6 - s_4c_6) + c_{23}s_5s_6$$

$$r_{13} = -c_1(c_{23}c_4s_5 + s_{23}c_5) - s_1s_4s_5$$

$$r_{23} = -s_1(c_{23}c_4s_5 + s_{23}c_5) + c_1s_4s_5$$

$$r_{33} = s_{23}c_4s_5 - c_{23}c_5$$

$$p_x = c_1[a_2c_2 + a_3c_{23} - d_4s_{23}] - d_3s_1$$

$$p_y = s_1[a_2c_2 + a_3c_{23} - d_4s_{23}] + d_3c_1$$

$$p_z = -a_3s_{23} - a_2s_2 - d_4c_{23}$$

$$Transformation\ Matrix = \begin{pmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Note : c_1 *and* s_1 $=$ cos_1 *and* sin_1 *respectively*

c_{23} $=$ $cos(Theta_2 + Theta_3)$ *and* s_{23} $=$ $sin(Theta_2 + Theta_3)$

2.6 Implementing the Inverse Kinematics Function

The Inverse Kinematics Function of the simulator generates all possible set of joint angles using the elements contained in the transformation matrix produced earlier by the Forward Kinematics Function as its input. The Inverse Kinematics in other words is another way of saying given the desired position and orientation of the end-effector or tool relative to the base frame (station), how do we compute the sets of joint angles which will achieve the desired results. There are generally two approaches employed in solving Inverse Kinematics problems, the Algebraic and Geometric solutions. The Algebraic solution was used in this particular case. The following equations were used in implementing the Inverse Kinematics Function for the simulator.

$$Theta_1 = atan2(p_y, p_x) - atan2(d_3, \sqrt{1 - (d_3^2/p^2)})$$

$$Theta_3 = atan2(a_3, d_4) - atan2(K, \sqrt{a_3^2 + d_4^2 - K^2})$$

Where k $= p_x^2 + p_y^2 + p_z^2 - a_2^2 - a_3^2 - d_3^2 - d_4^2/2a_2$

(**Note :** $Theta_3$ *is always computed before* $Theta_2$ **)**

$$Theta_2 = Theta_{23} - Theta_3$$

$$Theta_4 = atan2(-r_{13}s_1 + r_{23}c_1, -r_{13}c_1c_{23} - r_{23}s_1c_{23} + r_{33}s_{23})$$

$$Theta_5 = atan2(s_5, c_5)$$

$$Theta_6 = atan2(s_6, c_6)$$ [15]

CHAPTER 3

PROGRAMMING THE ROBOT

The greatest force of robots is their flexibility, their ability to rearrange for new production and their large movement range. Utilization of the robot's flexibility presupposes effective programming. The robot programming can take place in two principally different ways: on-line or off-line. In On-line programming the use of the robot and equipment is required, whereas off-line programming is based on computer models of the production equipment. Both these methods have advantages and disadvantages. In this section we will take a look at how the two methods can be combined.

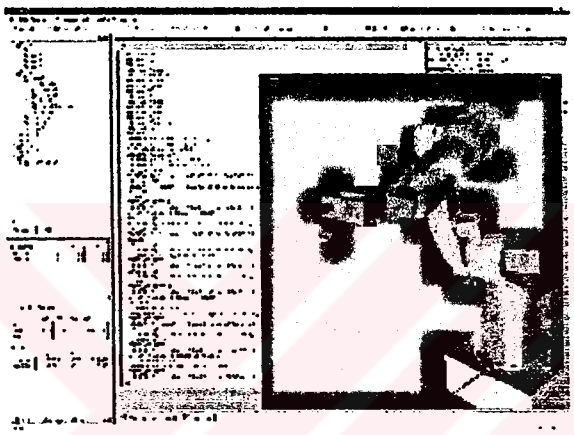


Figure 3.1: Screenshot from a commercial Robot Simulation Software

3.1 On-line programming

Off-line Robot Programming and Simulation makes the development of robot programs as well as the simulation of robot models possible. The robot programs can be tested and debugged on the simulated model before being executed on the actual robot itself. possible damage to the robot or to objects in the robot environment can be avoided .The practice of Off-line Robot Programming and Simulation also makes a great economic sense when applied to automated factories. A greater percentage of production level can still be maintained since the robot (production equipment) will not necessary be tied down when the need for reprogramming or testing arises.

Manipulators (robots) consist primarily of links which are in turn connected with joints that allows relative motion of neighboring links. The number of joints is usually equal to the number of degrees of freedom possessed by the robot. The big challenge in robot

programming and simulation is in how to accurately describe the position and orientation of the links of the manipulator, the parts, tools, and objects in the manipulators environment. The quantities are usually represented and manipulated mathematically.

Forward Kinematics deals with the problem of computing the position and orientation of the end effector (the free end of the chain of links that makes up the manipulator) given a set of joint angles. When the position and orientation is given, the Inverse Kinematics produces all possible set of joint angles that could be used to attain the given position and orientation.

Forward Dynamics computes the amount of actuator forces needed to move or accelerate a manipulator from rest or to simply glide at a constant end-effector velocity. Trajectory Generation tackles the problem of computing the motion functions which causes the robot or manipulator to move from one point to the other in a smooth and in a controlled fashion by simply causing each joint to move as specifed by a smooth function of time. These many principles and others basically defines the behavior and method of operation of a typical robot. A successful robot simulator and programming environment must then be able of simulating most or all of the underlying tasks or principles that makes a real robot what it is.

3.2 Advatages and Disadvantages of Online Programming

On-line programming takes place at the site of production itself and involves the workcell. The robot is programmed with a teach box. On-line programming has the following advantages and disadvantages compared to off-line programming:

Advantages	Disadvantages
1. Easily accessible.	1. Slow movement of the robot while programming.
	2. Program logic and calculations are hard to program.
	3. Suspension of production while programming.
	4. Cost equivalent to production value. Poorly documented.

Table 3.1 Advatages and Disadvantages of Online Programming

The most significant advantage of on-line programming is that the robot is programmed in concordance with the actual position of equipment and pieces. Contrary, the most significant disadvantage is that it occupies valuable production equipment.

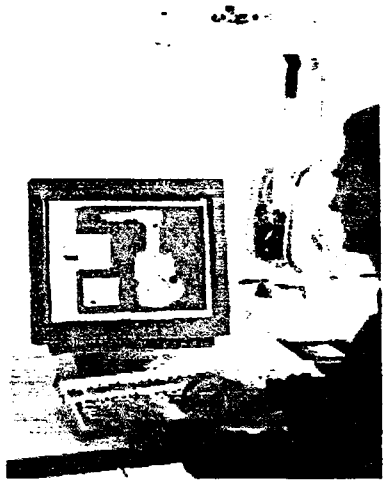


Figure 3.2: An example simulator

3.3 Off-line programming

Off-line programming takes place on a computer and models of the workcell with robot, pieces and surroundings are used. The robot programs can in most cases be created by the reuse of existing CAD data so that the programming will be quick and effective. The robot programs are verified in simulation and any errors are corrected.

3.4 Advatages and Disadvantages of Online Programming

Advantages	Disadvantages
1. Effective programming of program logics and calculations with state-of-the -art debugging facilities.	1. Demands investing in an off-line programming system.
2. Locations are built according to models and this can mean that programmers will have to fine tune programs on-line or utilize sensors.	
3. Effective programming of locations. Verification of program through	

simulation and visualization.

4. Well documented through simulation model with appropriate programs.
5. Reuse of existing CAD data.
6. Cost independent of production. Production can continue while programming.
7. Process support tools for instance selection of welding parameters.

Table 3.2 Advatages and Disadvantages of Off-line Programming

The biggest advantage of Off-line programming is that it does not occupy production equipment, and in this manner production can continue during the programming process. By far the largest proportion of robots are today being programmed on-line. This is mainly due to the fact that off-line programming has had a very high burden rate and demanded the need of expert users.

Advanced off-line programming tools contain facilities for debugging and these assist effective programming. The programming tools support utilization of supporting tools for the programming process, for instance optimization of the welding process.

The robot off-line programming and simulation system takes costs to an attractive level and runs under Microsoft Windows with well-known present-day user interface.

3.5 Planning in the virtual production system

Production planning in the virtual production system is carried out through model driven simulation. The most significant installations of the production system are described with models and through simulation their properties are visualized. Different combinations of equipment, layouts, programs, product designs, and the interplay of these items are tested and evaluated in simulation.

A robot simulation offers the following opportunities through planning in the virtual production system:

1. Minimizing rearrangement hours through simulation and off-line programming
2. Optimizing existing programs
3. Reuse of existing CAD data
4. Testing of new equipment in the production system before investments are made
5. Visualizing the virtual system in 3D computer graphics

3.6 Comparison On-line Vs Off-Line

The right combination of programming methods allow for a very effective programming and a quick rearrangements of production. An up-to-date developers environment supports the usage of a combination of on-line and off-line programming techniques.

The cost of on-line programming is equivalent to the production value, whereas off-line programming has no expenses other than that of a programmer and the off-line programming tool. As production value is likely to be somewhat higher than that of off-line programming more time can be used in planning supported by off-line programming than is possible with on-line programming.

In most cases off-line programming is much faster than on-line programming, leading to substantial cost reductions by using off-line programming.

The correct combination of on-line and off-line programming therefore leads to cost reductions in production adjustments. This implies that the limit of when a production adjustment is remunerative is moved considerably. Or, to put it in other words, production simply becomes much more flexible. [7]

CHAPTER 4

THE ENVIRONMENT

The 3-D world is generally called the working environment ,which simulates the real-world application. Once the real-world is perfectly modelled using 3-D objects and rendering methods ,the simulation software looks user-friendly to the user.OPENGL makes it possible to achieve the highest quality working environment,by its powerfull 3-D management and support.

4.1 Working Environment

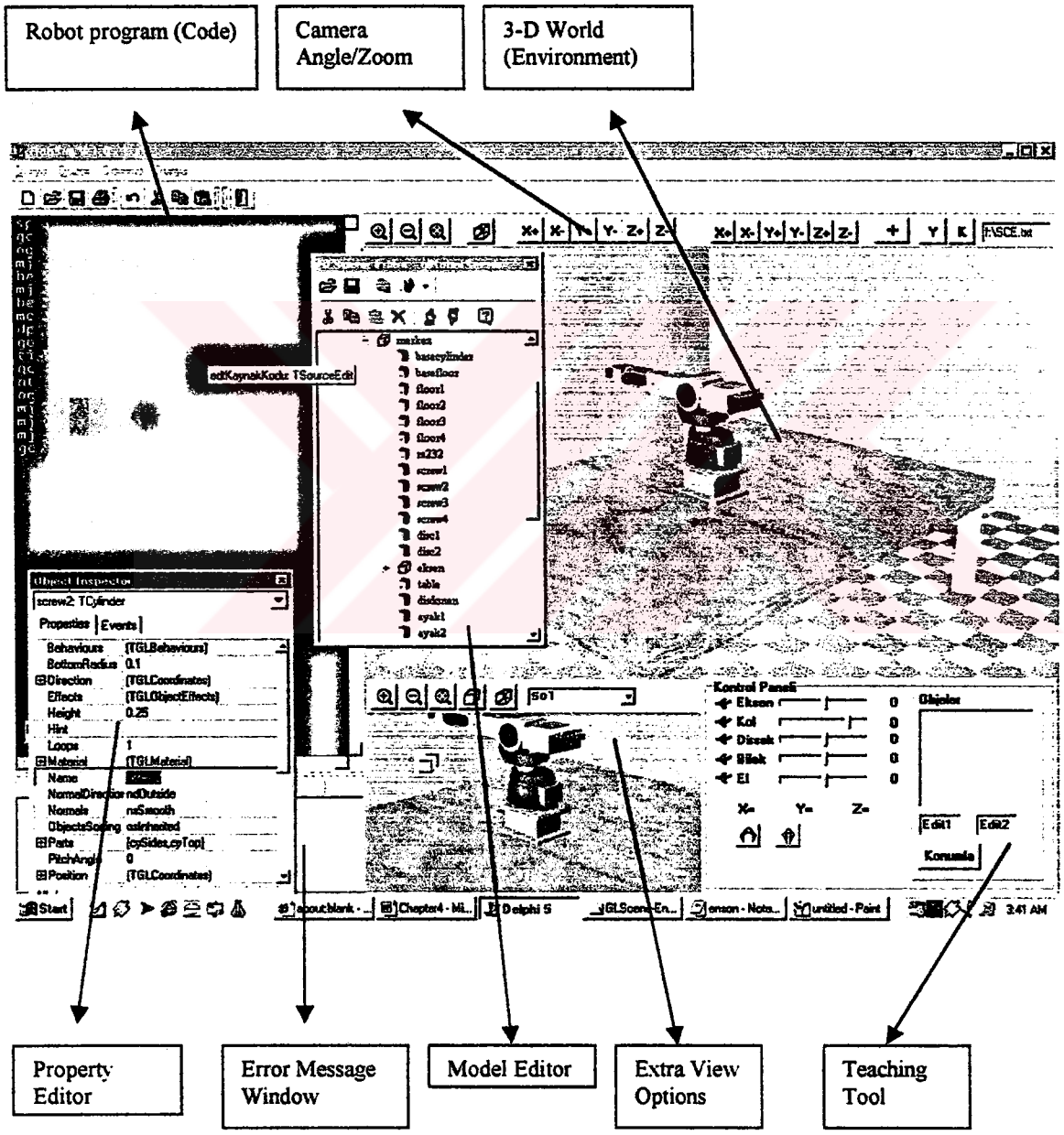


Figure 4.1: Working Environment

4.2 The main menu and tool buttons

The user-friendly designed application offers a menu and toolbar section for the coder. Also enables shorcuts to other functions. From the main menu user can load script (robot source code) and world files containing objects instead of the robot .Other options are save as,printing source code, converting source code to ready-to-run basic code ,or sending the code directly through the RS-232 port.You can also clear all world objects.

4.3 Camera Angle/Zoom

This option is a very important aspect for the coder. By using this function it's possible to change the eye-angle to look at the model at the desired angle.

X+ X- buttons enables the rotation around the X-Axis
Y+ Y- buttons enables the rotation around the Y-Axis
Z+ Z- buttons enables the rotation around the Z-Axis

This functions are crutial for defining the accurate position for eye-angle . Also using the the zoom in and zoom out button for zooming in and out of the SCENE(Model) .

4.4 The 3-D World

The 3-D world containst the world (3-Axis) the working environment,the robot and other objects that come together to complete the whole environment. This is the OPENGL 3-D stuff that is at the hearth of the system.

4.5 Property Editor

By using the propery editor the coder has the ability to set the properties of each objects in the 3-D world.

4.6 Robot Code

This textbox contains the source code of the robot simulation. A text highligting (command) text box component has been developed for the project. Using this component one can build its own compiler ,add new commands and defines its highligting options .

4.7 Error Message Window

Any compiler error is prompt at this window. Including the line number plus the error type .

4.8 Model Editor

This property editor is an ideal development tool for 3-D World design. Any kind of 3-D object can be inserted into the 3-D world.And their properties can be set using the property editor .

4.9 Extra View Window

This extra view helps to choose a constant angle of view so that the robot simulation can be tested from predefined camera angles.There are 5 options.Left,Right,Front,Back,and Top views .

4.10 Basic Motions

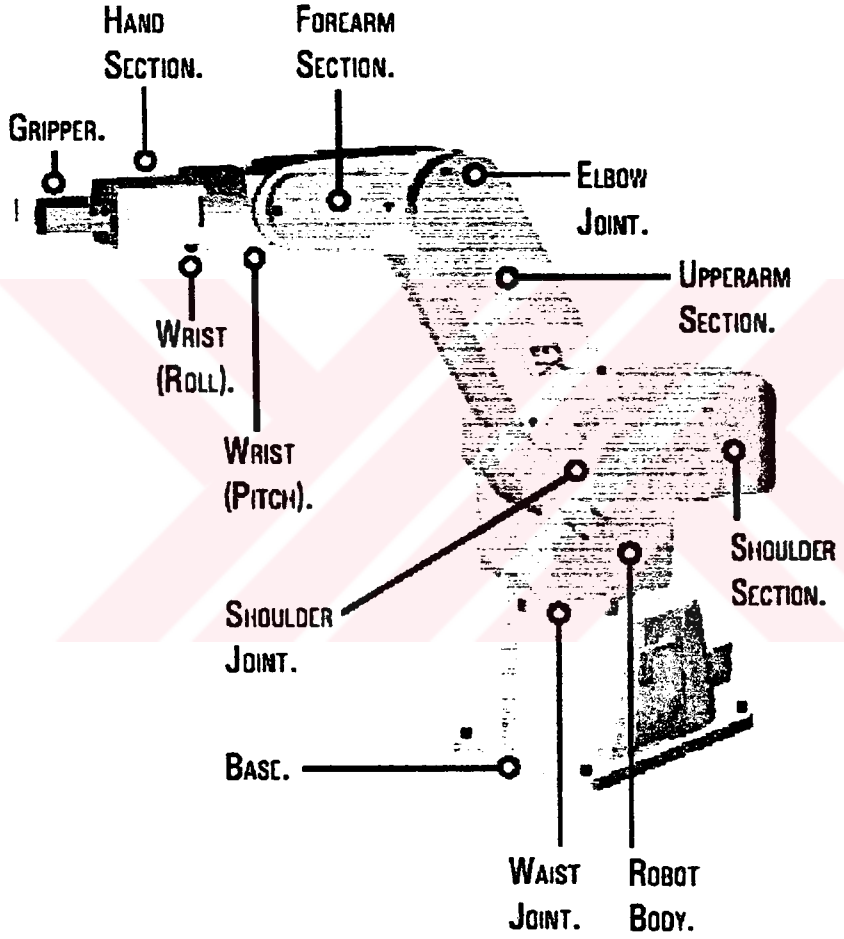


Figure 4.2: Robot parts

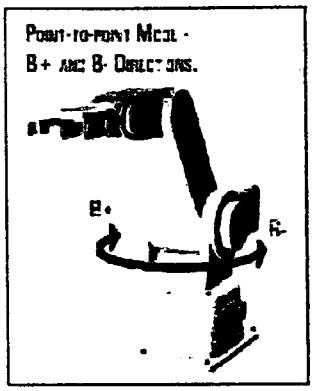


Figure 4.3: Moving around the base

Moving around the base. An example can be the following command .
MJ 30,0,0,0,0

Figure 4.4: Moving the upper arm

Moving the upper arm. An example can be the following command .
MJ 0,30,0,0,0

Figure 4.5: Moving the for arm

Moving the for arm. An example can be the following command .
MJ 0,0,30,0,0

Figure 4.6: Moving the hand

Moving the hand . An example can be the following command .
MJ 0,0,0,30,0

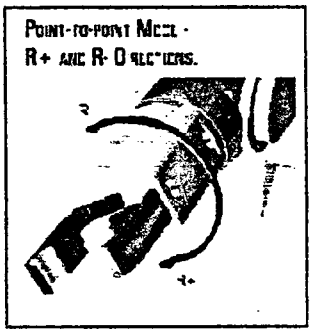


Figure 4.7: Moving the for arm

Moving the for arm. An example can be the following command .
MJ 0,0,0,0,30

Figure 4.8: Closing and opening grip

Closing and opening grip.
GC/GO

The possible actions aexplained above are most frequently used movements of the robot arm. Using different joints and different turn,roll,and pitch angles the robot handis positioned at a desired point in the 3-D world . [10]

4.11 Teaching Tool

This is simple control mechanism which looks similar to the original teaching tools ,with simplified actions. By using the teaching tool it is easy to control the robot manually and to define coordinates of a desired postion acurrately. It's has control on each joints of the robot arm and the its hand. Close/Open grip and adding new objects to the environment functions are also enabled in this section.

4.12 Possible Command Of the Simulation

DP (Decrement Position)

Moves the robot to a predefined position with a position number smaller than the current one.This is a very useful shortcut to optimize the source code.

Ex:MO 3

DW (Draw)

Moves the end of the hand to a position away from the position away from the current one covering the distance specified in the X,Y,Z axis directions.

Format :DW travel distance in x,travel distance in y,travel distance in z

Ex: DW 10,20,-30

IP(Increment Position)

Moves the robot to a predefined position with a position number greater than the current one.

Ex:IP

MA(Move Approach)

Moves the end of the hand from the current position to a osition way from a specified position in icrements as specified for another position.

Format : MA Position number a,Position number b, [O,C]

Ex : MA 2,3,C

MC(Move Continuous)

Move the robot continously through the predefined intermediate points between the two specified position numbers.

Format : MC Position number a,Position number b

Ex: MC 101,200

MJ (Move Joint)

Turn each joint the specifiead angle dfom the curremt position.

Format :MJ Waist turning angle, Shoulder turning angle,Elbow turning angle,Pitch angle,Roll Angle

Ex:MJ 10,20,40,50,30

MO (Move)

Moves the end of the hand to a specified position

Format :MO Position number [G,C]

Ex: MO 2,C

NT (Nest)

Returns the robot to mechanical origion

Ex:NT

OG (Origin)

Moves the robot to the reference in the cartesian coordinate system.

PC(Position Clear)

Clearsthe position data of specified position number of the numbers

Format :PC Position number a ,Position number b

Ex: PC 5,8

PD (Position Define)

Defines the coordinates (position and angle) of a specified position number.

Format :PD Position number, X-Axis coordinate, Y-Axis coordinate, Y-Axis coodinate,Pitch angle,Roll angle

Ex: PD 10,0,380,300,-70,-40

PL (Position Load)

Assigns the coordinates of a specified position number to another specified position number.

Format :PL Position number a,Position number b

Ex: PL 5,7

PX (Position Exchange)

Exchange the coordinates of a specified position number for those of another specified position.

Format : PX Position number a,Position number b

Ex: PX 2,3

SF(Shift)

Shift the coordinates of a specified position number in increments represent the coordinates of another specified number and redifines the new coordinates.

Format :SF Position number a,Position number b

Ex: SF 10,100

SP (Speed)

Sets the operating velocity and acceleration/decelaration time for the robot.

Format : SP speed level , [H ,L]

Ex: SP 7,H

TI (Timer)

Halts the motion for a specified period of time

Format :TI timer counter

Ex: TI 20

GC (Grip Close)

Closes the grip of the hand.

Ex:GC

GO (Grip Open)

Opens the grip of the hand.

Ex:GO [2] [10]

4.13 How required compoenets are defined in Delphi

The Source Code Editor Component

The following code explaing how the source code editor component is implemented for the project.

```
object edtKaynakKodu: TSourceEdit
  Left = 1
  Top = 1
  Width = 341
  Height = 548
  ScrollBars = ssBoth
  Bitmapped = False
  OnChange = edtKaynakKoduChange
  AlwaysShowCaret = False
  LeftMargin = 2
  TopMargin = 0
  TabSize = 2
  Align = alTop
  Color = clNavy
  Constraints.MinHeight = 64
  Constraints.MinWidth = 64
  Font.Charset = TURKISH_CHARSET
  Font.Color = clWhite
  Font.Height = -13
  Font.Name = 'Lucida Console'
  Font.Style = []
  ParentColor = False
  ParentFont = False
  TabOrder = 2
  SyntaxColoring.Enabled = True
  SyntaxColoring.SymbolColor = clWhite
  SyntaxColoring.SymbolStyle = []
  SyntaxColoring.SymbolCustomStyle = False
  SyntaxColoring.NumberColor = clWhite
```

```
SyntaxColoring.NumberStyle = []
SyntaxColoring.NumberCustomStyle = False
SyntaxColoring.WordLists = <
 item
  Caption = 'Komutlar'
  CustomColor = True
  Color = clYellow
  CustomStyle = True
  Style = [fsBold]
  CaseSensitive = False
  Words.Strings = (
   'DP'
   'DW'
   'GC'
   'GO'
   'HE'
   'IP'
   'MJ'
   'MO'
   'NT'
   'OG'
   'SP'
   'TI')
 end>
SyntaxColoring.CustomStyles = <>
SyntaxColoring.ParenthesisColors.Strings = (
 '$000000')
SyntaxColoring.ParenthesisStyle = []
SyntaxColoring.ParenthesisCustomStyle = False
AutoIndent = True
AutoIndentIncrease = False
AutoIndentIncreaseStart = '{'
AutoIndentIncreaseEnd = '}'
SplitOnFly = True
```

```
    end
  end
end
```

Defeining a GLSceneViewer

Following code shows how a scene can be defined in Delphi.

```
object GLSceneViewer1: TGLSceneViewer
  Left = 345
  Top = 525
  Width = 325
  Height = 175
  Cursor = crHandPoint
  FogEnvironment.FogColor.Color = {0000000000000000000000000000803F}
  FogEnvironment.FogStart = 10
  FogEnvironment.FogEnd = 1000
  FogEnvironment.FogMode = fmLinear
  BackgroundColor = clBtnHighlight
  Camera = GLCamera1
  Monitor = True
end
```

Defining a New Scene

```
object glRobotScene: TGLScene
  Left = 360
  Top = 80
  object glRobotIsik: TGLLightSource
    Ambient.Color = {0000000000000000000000000000803F}
    ConstAttenuation = 1
    Diffuse.Color = {0000803F0000803F0000803F0000803F}
    Position.Coordinates = {0000F0410000A041000020410000803F}
    Specular.Color = {0000000000000000000000000000803F}
    SpotCutOff = 180
    SpotDirection.Coordinates = {00000000000000000000080BF00000000}
  end
```

```
object merkez: TDummyCube
  TagFloat = 2
  Direction.Coordinates = {768C5CB1B102B9B10000803F00000000}
  Position.Coordinates = {0000000000000000000000000000803F}
  Scale.Coordinates = {0000803F0000803F0000803F00000000}
  Up.Coordinates = {BED18EBC0AF67F3FE696FD3200000000}
  CubeSize = 1
  EdgeColor.Color = {0000000000000000000000000000803F}
  object basecylinder: TCylinder
    Direction.Coordinates = {00000000000000000000803F00000000}
    Position.Coordinates = {000000003333333F00000000000803F}
    Scale.Coordinates = {3333333F0000803F0000803F00000000}
    Up.Coordinates = {000000000000803F0000000000000000}
    Material.FrontProperties.Emission.Color = {0000000000000000000000000000803F}
    Material.FrontProperties.Specular.Color = {0000000000000000000000000000803F}
    Material.BlendingMode = bmTransparency
    Material.MaterialOptions = []
    Material.Texture.TextureMode = tmBlend
    Material.Texture.TextureWrap = twNone
    Material.Texture.Disabled = False
    BottomRadius = 1.19000005722046
    Height = 1.33000004291534
    Stacks = 1
    TopRadius = 0.975000023841858
    Parts = [cySides, cyTop]
  end
```

Defining A Cube

```
  object basefloor: TCube
    Direction.Coordinates = {00000000000000000000803F00000000}
    Position.Coordinates = {000000000AD7233D00000000000803F}
    Scale.Coordinates = {0000803F0000803F0000803F00000000}
    Up.Coordinates = {000000000000803F0000000000000000}
    Visible = False
```

```
  Material.BackProperties.Emission.Color = {0000000000000000000000000000803F}
  Material.BackProperties.Specular.Color = {0000000000000000000000000000803F}
  Material.FrontProperties.Specular.Color = {0000000000000000000000000000803F}
  Material.MaterialOptions = []
  CubeSize = {3333F33F295C8F3D00002040}
 end
 object floor1: TCube
  Direction.Coordinates = {00000000000000000000803F00000000}
  Position.Coordinates = {C3F5683F000000001F856B3F0000803F}
  Scale.Coordinates = {0000803F0000803F0000803F00000000}
  Up.Coordinates = {000000000000803F0000000000000000}
  Visible = False
  Material.BackProperties.Emission.Color = {0000000000000000000000000000803F}
  Material.BackProperties.Specular.Color = {0000000000000000000000000000803F}
  Material.FrontProperties.Emission.Color = {0000000000000000000000000000803F}
  Material.FrontProperties.Specular.Color = {0000000000000000000000000000803F}
  Material.MaterialOptions = []
  Parts = [cpTop, cpFront, cpBack, cpLeft, cpRight]
  CubeSize = {6666263F2B87163E6666263F}
 end
```

Defining a Cylinder

```
object screw2: TCylinder
   Direction.Coordinates = {00000000000000000000803F00000000}
   Position.Coordinates = {3333333F9A9919BE666666BF0000803F}
   Scale.Coordinates = {0000803F0000803F0000803F00000000}
   Up.Coordinates = {000000000000803F0000000000000000}
   Material.FrontProperties.Emission.Color = {0000000000000000000000000000803F}
   Material.FrontProperties.Specular.Color = {0000000000000000000000000000803F}
   Material.BlendingMode = bmAdditive
   Material.MaterialOptions = []
   BottomRadius = 0.100000001490116
   Height = 0.25
```

```
    Slices = 9
    Stacks = 1
    TopRadius = 0.100000001490116
    Parts = [cySides, cyTop]
  end
```

Defining A Dummy Cube for rotating around a fixed point

```
  object eksen: TDummyCube
    Direction.Coordinates = {783D2C34BF47AF980000803F00000000}
    Position.Coordinates = {000000000000C03F3333B3BE0000803F}
    Scale.Coordinates = {0000803F0000803F0000803F00000000}
    Up.Coordinates = {50851D240000803FB4D7851600000000}
    CubeSize = 0.425000011920929
    EdgeColor.Color = {00000000000000000000000000000803F}
```

4.14 USING THREADS

For most applications, you can use a thread object to represent an execution thread in your application. Thread objects simplify writing multi-threaded applications by encapsulating the most commonly needed uses of threads.: Thread objects do not allow you to control the security attributes or stack size of your threads. If you need to control these, you must use the BeginThread function. Even when using BeginThread, you can still benefit from some of the thread synchronization objects and methods described in Coordinating threads.

Main idea for creating threads in the project is to take care of different robot actions occuring simulatenously.The situation occurs when you got to move more than one joints at a time.Therefore there is a need to create different proseses for each joint. Assume that we want to move all five joints with different angles ,causing to create 5 different processes to be created. The code below is derived from the classs Tthread. This class makes thread stuff easier for the coder to define a desired thread.

```
//KOL
 TKolThread = class (TThread)
 private
  rKolAngle:Real;
 protected
  procedure SetKolAngle;
  procedure Execute; override;
 public
  rKolAcisi :Real;
  rKolilkAci:Real;
  rAciYon  :String;
  nRobotHizi :Real;
 end;
```

The priority indicates how much preference the thread gets by the operating system. Use a high priority thread is the thread is a critical task, otherwise use a low priority thread to perform other tasks. To indicate the priority of your thread object, set the Priority property.

Waiting for other tasks to finish

This is a very important task to know when and which process will stop . If we want other processes to wait for our process than we have to tell the others to wait. This is the waitfor command.Therefore other processes will wait for the termination of the specific task.

Samples:

MyKolThread:= TElThread.Create (True); (creating a thread for moving hand)

MyKolThread.WaitFor; (wait for the thread moving the hand)

4.15 Components used in the Projects

GLCollision

Defines how fine collision bounding is for a particular object.

Possible values are :

1. cbmPoint : the object is punctual and may only collide with volumes
2. cbmSphere : the object is defined by its bounding sphere (sphere radius is the max of axis-aligned dimensions)
3. cbmEllipsoid the object is defined by its bounding axis-aligned ellipsoid
4. cbmCube : the object is defined by a bounding axis-aligned "cube"
5. cbmFaces : the object is defined by its faces (needs object-level support, if unavalaible, uses cbmCube code)

GLCollision

Collision detection behaviour.

Allows an object to register to a Tcollision Manager and be accounted for in collision-detection and distance calculation mechanisms.

An object may have multiple TGLBCollision, registered to multiple collision managers, however if multiple behaviours share the same manager, only one of them will be accounted for, others will be ignored.

GLCamera

Camera object. This object is commonly referred by TGLSCene Viewer and defines a position, direction, focal length, depth of view... all the properties needed for defining a point of view and optical characteristics.

Standard light source

The standard GLScene light source covers spotlights, omnidirectionnal and parallel sources Lights are colored, have distance attenuation parameters and are turned on/off through their Shining property.

Lightsources are managed in a specific object by the TGLScene for rendering purposes. The maximum number of light source in a scene is limited by the OpenGL implementation .

Describes a rendering material.

A material is basicly a set of face properties (front and back) that take care of standard material rendering parameters (diffuse, ambient, emission and specular) and texture mapping. An instance of this class is available for almost all objects in GLScene to allow quick definition of material properties. It can link to a TGLLibMaterial (taken for a material library).

Material Library

The TGLLibMaterial has more adavanced properties (like texture transforms) and provides a standard way of sharing definitions and texture maps.

Stores a set of materials, to be used and shared by scene objects. Use a material libraries for storing commonly used materials, it provides an efficient way to share texture and material data among many objects, thus reducing memory needs and rendering time.

Materials in a material library also feature advanced control properties like texture coordinates transforms.Component where the GLScene objects get rendered.

This component delimits the area where OpenGL renders the scene, it represents the 3D scene viewed from a camera (specified in the camera property). This component can also render to a file or to a bitmap.

Even if it is primarily a windowed component, it can handle full-screen operations : adjust display resolution with DisplayOptions, and simply make this component fit the whole screen (use a borderless form).

This viewer also allows to define rendering options such a fog, face culling, depth testing, etc. and can take care of framerate calculation.

Designning The 3-D World And the Robot

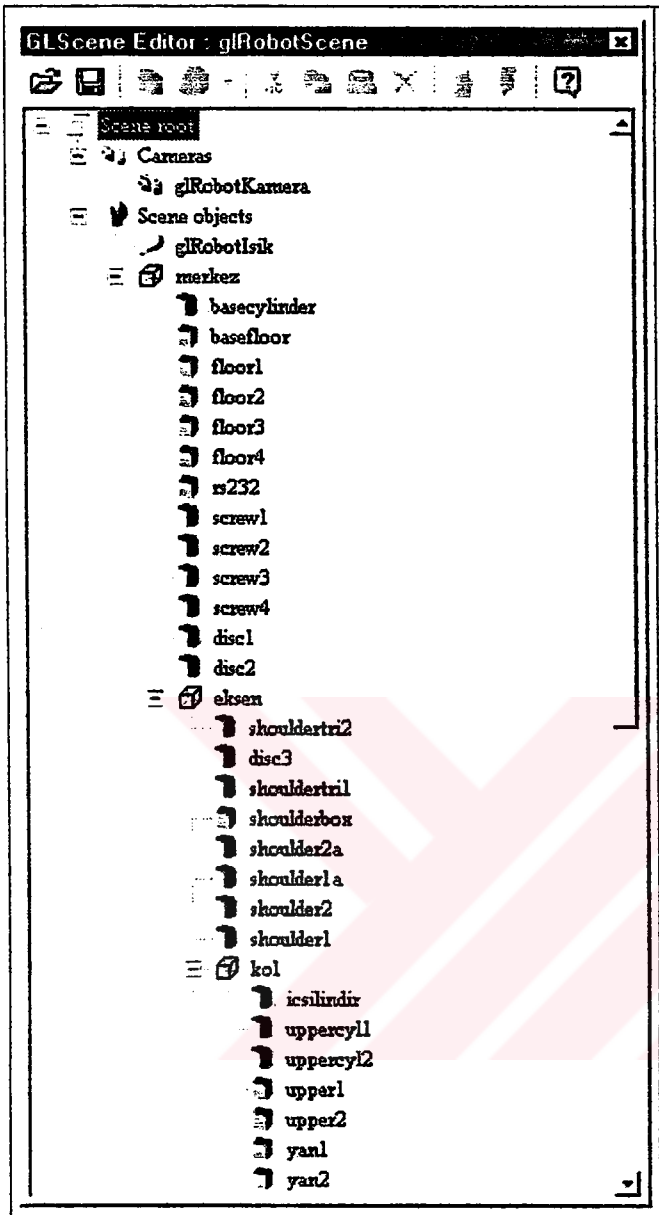


Figure 4.9: The property editor

The property designer on the left is the delphi opengl scene editor called the GLScene editor which helps the coder to model their 3-d world visually .

The root is called the scene root where you strart adding your cameras and other scene objects. In our project there is a main camera that is called the GLRobot Camera. This camera deals with the Light sources.

Dummy Cube: Generally an unvisible cube that is used for grouping objects. It is also used for rotation around a single point.

For ex: merkez,eksen,kol are samples of dummy cubes,while basecylinder,basefloor,floor,floor2 and other objects are called child objects of the dummy cubes(parent objects) .

As seen from the GLScene editor other basic OPENGL objects are used to build our model. These objects are generally cylinders and cubes and dummy cubes. Lets have alook at these object in detail.

The Dummy Cube

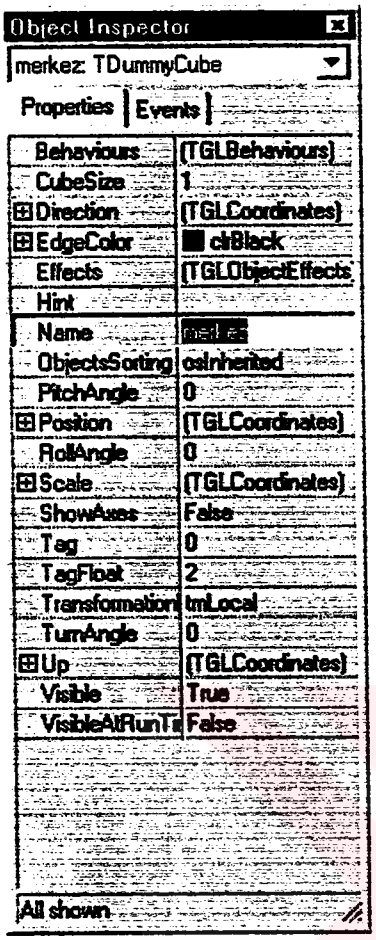


Figure 4.10: The Dummycube property editor

In the project the dummy cube is used mainly to group objects and to rotate around a desired point in any of the 3 axis.

Mostly dummy cubes are not visible at run time. So set the Visible at runtime propery to False.The dummy cube can be rotated in three different axis using the PithcAngle,Roll angle,and Turn angle properties which have the same action with the PitchAngle,Rollangle,and Turn angle of the other basic geometry objects like cylinders,cubes.

The important thing is that the position (X,Y,Z) of the dummy cube and the size of the cube should be defined exactly as the center of rotation. Note that cubesize=1 means that the cube has the size (1,1,1) in 3-dimensions.

If desired the ShowAxes property can be set to true to see the perfect positioning of objects.

The Cylinder

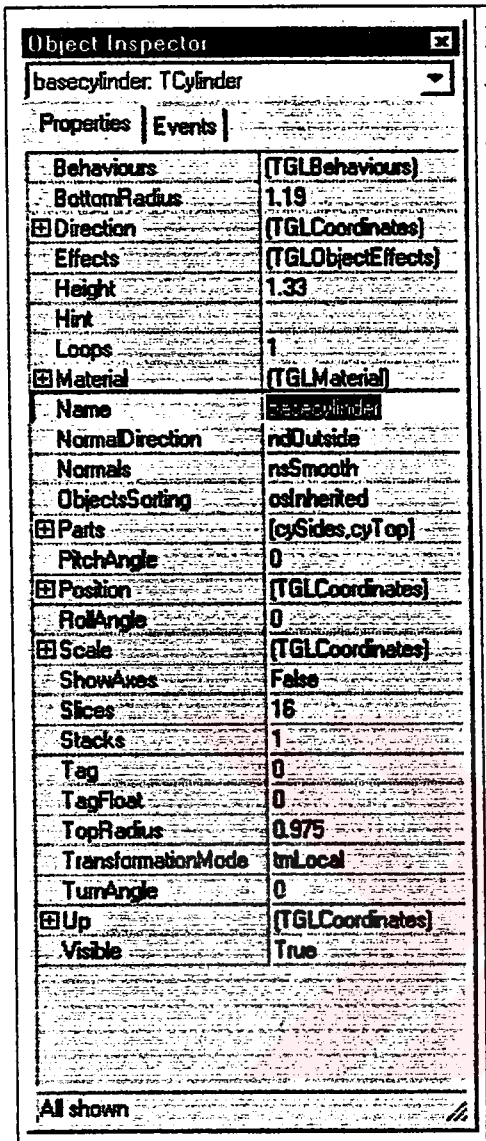


Figure 4.11: The Cylinder property editor

In the project the cylinder is most oftenly used object with the cube object. It is mainly used to draw the legs ofthe table and other rounded shaped parts of the robot arm.

In the scene it is clear that cylinders are child objects of some dummy cubes.Remember that the basecylinder object which is a cylinder is a child of the dummy cube merkez.That means if pitchs,rolls or shifts the objects that are the child of it will also do the same action with it. That makes it easier for the coder to write a single code to rotate more than one objects at the same time.

There are 3 mainly used properties .Top radius ,bottom radius and height. Keep in mind that sometimes scaling of the object might be usefull .These properties can be set using the Scale.X, Scale.Y, Scale.Z properties of the cylinder object.

The Cube

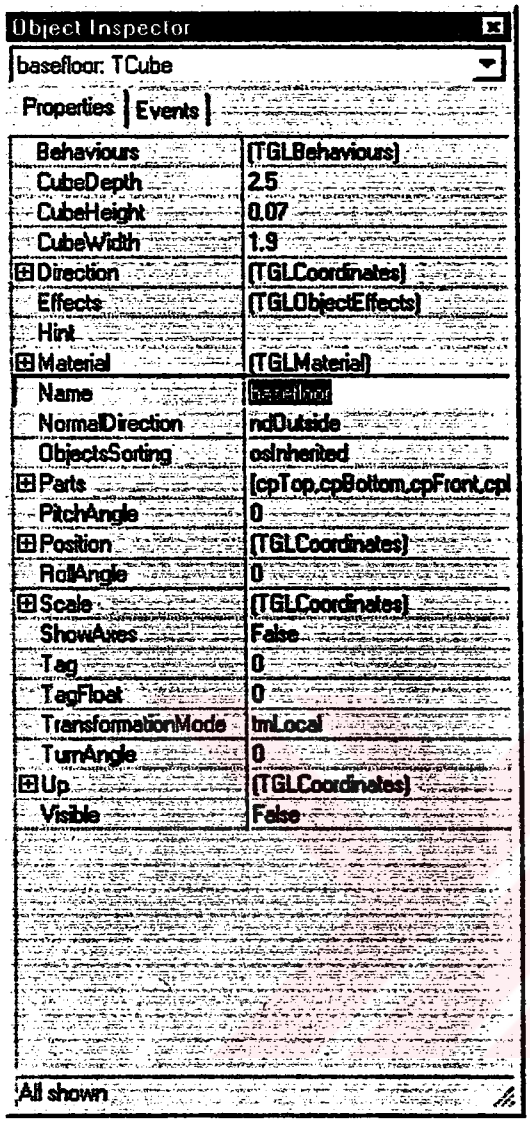


Figure 4.12: The cube property editor

In the project the cube object is also used object very frequently. It is mainly used to draw the table and the edged objects that are mostly the robots arms and other stuff like the basement of the robot .

There are 3 mainly used properties .Top radius ,bottom radius and height. Keep in mind that sometimes scaling of the object might be usefull .These properties can be set using the Scale.X, Scale.Y, Scale.Z properties of the cylinder object.

An object which is very close to a cube but has a different capability called the frustrum is also used in the robot object. It has the same properties like the cube,plus the possiblity of changing the dimensions of the top face of the cube. Because of this property it is also called the clylinderic cube .

Using The Material Library

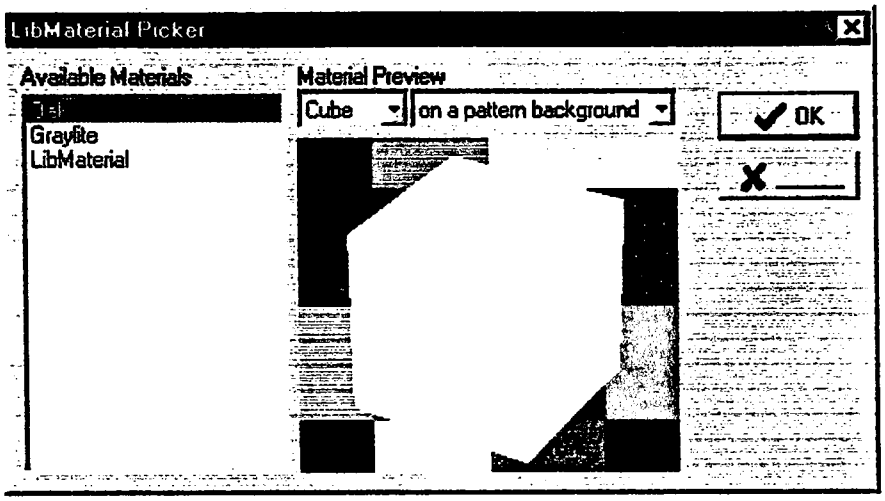


Figure 4.13: The Material Library

First of all , create a TGLMaterial Library. After that add the desired texture files into the library. This will be your source for any material that you want to fill your objects with.

For rendering objects using textures , different textures got to be kept in the memory to be used for different objects. To make it easier to organize texturing stuff a texture library is organized to hold the avaliable materials in the memory. It prevents not to waist memory for the same textures that is loaded for more than one . That means it is just enough to load the desired texture into the material library and link the texture to desired object when necessary, while there is only one copy of the each texture in the memory.

Basicly follow these steps.

1. Choose any object that you want to use texture on.
2. Select the Material Library option and then select the desired material to texture from the LimMaterial Picker that is shown in the figure above.

CHAPTER – 5

RESULTS AND RECOMENDATIONS

During the Robotics Technology lectures ,robotics seemed as a really hot topic for a computer engineer ,including both software and hardware.During that lecture ,a project has to be developed while the main problem was that one has to wait for other to finish their progress with the robot. The reason is that only one person can operate the robot and only one can excute ther source codes.

As we remember the disadvantages of the on-line simulation only one person has the ability of programming the robot and the situation we had in our robot lab was the same. So my supervisor and I decided to write a simulator for the robot in the robotics lab.
As we overview the project outline the project looked really detailed ,so we decided to develop a basic simulator so that other researchers can develope the code for a 100% accurately working simulator.At this point,80% of the simualtion is working properly,but it still needs some modifications . Some extra commands and control statements have to be added to the project . Multi-robot simulation is ready at the moment but not activated in the code, but it can be modified by other programmers .

One important highlight called collision detection is avaliable through the project so that the robot hand detects any collision between the robot's body ,and other objects like the table.This is done by the GLScene's Collision manager which is the standart collision detector developed by the Silicon graphics for the OPENGL unit.This is almost the most powerful feature of the simulation. Also real-time positioning of the hand ,grip status (open,close) and joint angles are displayed so that any position can be defined with the easiest way. The teaching box can be used to define the position manually using tracking bars reserved for each of the joints robot arm.Most off-line simulation coders use postion tracking method to define each coordinate using the real-time positioning and collision detection manager to define the postion of the robot hand with the smallest possible error.

Taking into consideration the project done, one computer engineer will clearly grasp that the project contains information about,compiler design,3D-Graphics,Robotics,and computer simulation.

The main aims of the thesis can me summarized as the following tasks

1. To study the concept of robot simulation,3-d graphics,OPENGL,compiler design
2. How to combine all those information to build software for the Robot
3. Examine and test the software .

A great research is made for documentation,componets,code samples, and tutorials on both computer graphics and robot stuff.

Completed Objectives

1. Designing a 3-D working environment
2. Designing a simple compiler (parser)
3. Implementation of robot commnad (80% completed)
4. Collision detection
5. Model and environment load/save
6. Holding object with the arm and set its is coordinates to another location.
7. Set robot position at anytime.
8. Create run-time modelling (adding and removing objects)

Feature work

1. Modelling may contain much detail
2. Multi robot simulation can be done
3. Predefined models can be added to the model.

REFERENCES

[1] Ben Zion Sandler ,P-Hall International Edition "ROBOTICS-Designing the Mechanism for Automated Machinery"

[2] Mitsubishi Movemaster Ex- Users Manual ,1992

[3] www.glscene.org

[4] http://www.slrsystems.com/rv-m1.htm

[5] http://www.rixan.com/rv_m1spc.htm

[6] http://www.meau.com/html/prodcat/rob/

[7] http://www.denford.com/dmt2004.html

[8] http://claymore.engineer.gvsu.edu/eod/courses/egr474/egr474-42.html

[9] http://www.rixan.com/rv_m1dwg.htm

[10] http://www.learningconcepts.net/rixan-mit/rv_m1.html

[11] http://home1.gte.net/jqjacobs/index.htm

[12] http://fly.srk.fer.hr/~unreal/theredbook/

[13] http://home1.gte.net/jqjacobs/index.htm

[14] http://www.frii.com/~martz/oglfaq/

[15] http://www.ibrtses.com/opengl/

RESUME

He was born in İstanbul in 1975.He graduated from Belde High School. He graduated from the Computer Engineering Department of Eastern Mediterenean University,Cyprus.