Dynamic System Simulation on the Web

By

Khaled Mahbub, B. Sc. Eng.

This thesis is submitted as the fulfilment of the
Requirement for the award of degree of
Master of Engineering (M.Eng.)

to

Dublin City University

September 2002

Research Supervisor: Professor M. S. J. Hashmi
School of Mechanical & Manufacturing Engineering

DECLARATION

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Master of Engineering is entirely my own work and has not been taken from the work of others save to the extent that such work has been cited and acknowledged within the text of my work

Signed: khaled mahbub (Candidate)

ID No.: 50162551

Date: september 25, 2002

Acknowledgements

First, I extend my deepest gratitude to my supervisor, Professor M.S.J. Hashmi, for his constant encouragement and assistance. He is a patient and nurturing mentor, who could always boost my confidence in times of self-doubt or frustration. His insistence on perfection was relentless and sometimes painful, but undoubtedly this will serve as a guide for the rest of my life. Without his guidance and continuous cooperation throughout this research it would not have been possible to finish this dissertation. I am honoured to have his name on this work.

I wish to express my sincere thanks to Keith Hicky, school system administrator, for his tireless services. Without his assistance during computer hardware failure things could be dreadful to me. He also provided all the necessary software for the project at the high time. I like to thank my fellow graduate students for their spontaneous supports, comments and particularly their joyfulness that made my hard times smoother.

Finally, I make special mention of my parents, who always motivated me for higher studies. They have supported and understood me every time and everywhere. Special thanks to them for everything they have done and given to me.

Dynamic System Simulation on the Web

by

Khaled Mahbub

(ABSTRACT)

Computer simulation is the discipline of designing a model of an actual or theoretical physical system, executing the model on digital computer, and analysing the execution output. Of late, simulation has been influenced by an increasingly popular phenomenon – the World Wide Web or WWW. Java is a programming language for the WWW that brings a high level of dynamism to Web applications. Java makes it particularly suitable to represent applications on the Web. It has created an illusion of machine independence and interoperability for many applications. Therefore WWW can be considered as an environment for providing modelling and simulation applications. Research in the area of Web-based simulation is developing rapidly as WWW programming tools develop. Bulk of this research is focused only on discrete event simulation. This dissertation introduces dynamic system simulation on the Web. It presents and demonstrates a Web-based simulation software (*SimDynamic*), entirely developed in Java, for modelling, simulating, and analysing dynamic systems with 3D animated illustration, wherever applicable. *SimDynamic* can also be used as a non Web-based application on a PC. In both cases, it supports complete model creation and modification capabilities along with graphical and numerical output. Detail design and functional ability of *SimDynamic* are provided. Some real world systems have been modeled using *SimDynamic* and results are presented. Characteristic features of the software are discussed from software engineering point of view. Complete source code and installation instructions are included. Current *SimDynamic* limitations and potential customization and expansion issues are explored.

Table of Contents

Appendices:

List of Figures

List of Tables

Chapter One

1 Introduction

1.1 Virtual Reality

Virtual Reality (VR) is the simulation of a real or imagined environment that can be experienced visually in the three dimensions of width, height, and depth and that may additionally provide an interactive experience visually in full real-time motion with sound and possibly with tactile and other forms of feedback. The simplest form of virtual reality is a 3D image that can be explored interactively at a personal computer, usually by manipulating keys or the mouse so that the content of the image moves in some direction or zooms in or out. This type of experience is known as *desktop VR* or *non-immersive VR*.

On the other hand, the strict definition of VR involves the sense of total immersion. To experience a virtual reality, a user dons special gloves, earphones, and goggles, all of which receive their input from the VR system. In this way, at least three of the five senses are controlled by the VR system. Therefore, the user is in computer-generated 3D, artistic renderings of real or imagined spaces.
From these above definitions Virtual Reality can be divided into:

- The simulation of real environments such as a manufacturing process or a spaceship often with the purpose of training or education.
- The development of an imagined environment, typically for a game or educational adventure.

1.1.1 Types of VR System

VR systems can be classified based on the mode with which they interface to the user. The most common modes used in VR systems are as follows:

a) Window on the World Systems (WoW)

In this system a conventional computer monitor is used to display the visual world. This is sometimes called Desktop VR or a Window on the World (WoW). This

concept traces its lineage back through the entire history of computer graphics. One must look at the screen as a window through which one beholds a virtual world. It is up to the computer graphics to make the picture in the window look real, sound real and the object real.

b) Video Mapping

Video Mapping can be considered as an extension of WoW approach. A video input of the user's silhouette is merged with a 2D computer graphic in a WoW system. So the user watches a monitor that shows his body's interaction with the virtual world.

c) Immersive System

Immersion is a key issue in VR systems as it is central to the paradigm where the user becomes a part of the simulated world. The ultimate VR systems completely immerse the user's personal viewpoint inside the virtual world. It has to meet four conditions: (1) a head-mounted device (HMD) with a wide field of view; (2) tracking the position and attitude of the user's body; (3) transducers that interpret user's natural behaviours, and (4) negligible delays in the rate at which the virtual environment is updated in response to user's movements and actions.

d) Telepresence

Telepresence is defined as the experience of presence in an environment by means of a communication medium. In other words, it refers to the mediated perception of an environment. This environment can be either a temporarily or spatially distant real environment or an animated but non-existent virtual world synthesized by a computer.

e) Mixed Reality

Merging the telepresence and VR systems gives the Mixed Reality or Seamless Simulation systems. Here the computer generated inputs are merged with telepresence inputs and/ or the user's view of the real world. For example a fighter pilot sees computer generated maps and data displays inside his helmet visor or on cockpit displays.

1.2 Simulation

A system is defined to be a collection of entities, e.g. people or machines that act and interact together toward the accomplishment of some logical end. A model is a simplified representation of a system at some specific point in time and or space intended to promote understanding of the real system. Simulation is the discipline of designing a model of an actual or theoretical physical system and manipulating the model in such a way that it operates on time or space to compress it, thus enabling one to perceive the interactions that would not otherwise be apparent because of their separation in time or space. It is a discipline for developing a level of understanding of the interaction of the parts of a system, and of the system as a whole. And this level of understanding is seldom achievable via any other discipline. Simulations are generally iterative in their development. One develops a model, simulates it, learns from the simulation, revises the model, and continues the iterations until an adequate level of understanding is developed.

1.2.1 Types of Simulation Models

a) Static vs. Dynamic Simulation Models

Static models describe a system mathematically, in terms of equations, where the potential effect of each alternative is ascertained by a single computation of the equation. Static models ignore time-based variances. So they represent a system at a particular point of time or a system in which time simply plays no role. Also, this type of model does not take into account the synergy of the components of a system, where the actions of separate elements can have a different effect on the total system than the sum of their individual effects would indicate.

On the other hand, a dynamic simulation model represents a system as it evolves over time. It is a representation of the dynamic or time-based behaviour of a system. While a static model involves a single computation of an equation, dynamic modelling is iterative. A dynamic model constantly re-computes its equations as time changes. Dynamic modelling can predict the outcomes of possible courses of action and can account for the effects of variances or randomness. One cannot control the occurrence

of random events, but can use dynamic modelling to predict the likelihood and consequences of their occurring.

b) Deterministic vs. Stochastic Simulation Models

A deterministic model can be identified as a model that does not contain any probabilistic (i.e. random) components. In such models output is determined once the set of input quantities and relationships in the model have been specified.

On the other hand, many systems must be modelled as having at least some random input components, and these give rise to stochastic simulation models. Most queuing and inventory systems are modelled stochastically, and these models produce output that is itself random.

c) Continuous vs. Discrete-Event Simulation Models

Discrete event simulation concerns the modelling of a system as it evolves over time by a representation in which the state variables change instantaneously at separate points in time. These points in time are the ones at which an event occurs, where an event is identified as an instantaneous occurrence that may change the state of the system.

Continuous simulation models concern the modelling over time of a system by a representation in which the state variables change continuously with respect to time. Typically, continuous simulation models involve differential equations that give relationships for the rates of change of the state variables with time. If the differential equations are particularly simple, they can be solved analytically to give the values of the state variables for all values of time as a function of the values of the state variables at time 0. For most continuous models analytic solutions are not possible, however, and numerical-analysis techniques, e.g. Runge-Kutta integration, are used to integrate the differential equations numerically, given specific values for the state variables at time 0.

1.3 World Wide Web

The Internet is a collection of interconnected computer networks operating under a common communications protocol. The World Wide Web (WWW), currently the fastest growing segment of the Internet, is a collection of electronic documents called pages, which reside on computers called WWW servers. These documents contain a combination of text, images, audio and video. The pages on the Internet are public and anyone can generate a page and put it on a WWW server. The pages and the associated files must meet a standard protocol.

The text portions of the document are written in a mark-up language called HTML (Hypertext Markup Language) [1]. Hypertext is text with links such that the user can follow a non-linear path through a document or set of documents. The unique characteristics of WWW hypertext documents is that the links can point to positions in the same document, positions in the other documents on the same server, or to WWW pages on any server anywhere on the Internet. They can also link to images, audio or video. The links are indicated by highlighted text and the link is established by pointing and clicking on the highlighted text.

A person, group or organization creates a page because they have certain information, which they wish to make available to other people. Usually this information is related to and hence references information others have decided to make available. Hence the pages point to one another and, as they multiply, form a web. Thus the WWW is a body and organization of information, which has no top down structure but is organized in a completely bottom up fashion by the individuals creating the individual pages.

The addresses of the pages are called URL's (Universal Resource Locators) [1] and there is a standard communications protocol (HTTP) [1] for requesting and obtaining any public Web page from any public server. The user of the Web operates on a client, a workstation or PC running a Windows, UNIX, Macintosh or OS/2 operating system. The client has software, called a browser, which enables it to request a page from the server and process the HTML file presenting the processed material to the user.

Therefore, the Internet and the WWW hold tremendous potential for the communication and the general distribution of information within groups with a common professional interest.

1.4 Web – based Simulation

In recent years the Internet in general, and World Wide Web (WWW or Web) in particular, have grown rapidly as dissemination tools for different kinds of information resources. Frequently, the Web is used for deployment of educational and commercial material. Educators are using the Web to post course notes, syllabi, homework assignments, and even exams and quizzes. Companies are using the Web for advertising, publicity, and to sell products. Through JAVA [2], an object oriented programming language, people can also customize WWW based computational programs as they would with traditional non-Internet based programs, and there is a growing interest in using the Web as a new platform for computer programs. The disciplines concerned with computer simulation are no exception to this phenomenon; the concept of *web-based simulation* has been introduced and is currently the subject of much interest to both simulation researchers and simulation practitioners. Web-based simulation is an attempt to exploit Web technology to support the future of computer simulation. Apart from providing other services, the World Wide Web is being looked upon as an environment for hosting modelling and simulation applications. Existing computer simulation support is either language-based or a library approach. In either case, they suffer from lack of portability to other environments. Also markets in educational software are small. There are thus little commercial interests in the production of simulation software for the educational market. Any simulation software, which is produced is aimed at the industrial market and is often too expensive for the educational purchaser. Moreover the demand for consulting in modelling and simulation has grown faster than the consulting companies can offer. Use of the Internet and its supporting tools such as the virtual environment and the interactive distributed simulation has the potential to overcome these factors limiting the wider use of simulation. The web's ability to service large and diverse audiences allows the simulation community to legitimately provide models and simulations as end products. Recent advances in web technology have

made the web a viable mechanism for performing, publishing, and distributing simulation.

1.5 Advantages and Disadvantages of Web – based Simulation

A Web-based simulation program provides several beneficial features that are lacking in available non Web-based packages. These features include wide availability, controlled access, user friendliness and efficient maintenance etc. On the other hand, simulation over the WWW does have a few potential drawbacks. These advantages and disadvantages should be considered when deciding which simulation package best meets a user's needs. This section discusses some of these features provided by a Web-based simulation program.

Wide Availability: In case of non Web-based simulation software, a change in platform (whether change in operating system or computer hardware) forces the recompilation of the simulation model (programme). Utilizing this type of package requires access to a computer containing the proper simulation software, as well as an appropriately compiled copy of the model source. The portability of a Web-based model onto various platforms enables the user to access and run the model from wide spread locations. There is no need to transport hardware or software to these sites or recompile the code. Moreover Internet is usually available twenty-four hours a day, access to the simulation software is not limited by time constraints. So the users can work within their own time schedule.

Controlled Access: With non Web-based package, user can access the model and software directly from the computer on which the model is running. There is every possibility of inadvertent modification to the model whenever the user has free access to the complete source code of the model and the software. On the other hand, Web-based simulation software and models can be protected from unauthorized modifications by imposing password and time limit restrictions on either the simulation package or the entire site.

User Friendliness: Most of the non Web-based packages need proper installation of the software and working knowledge of the operating commands. Web-based simulation software and models developed with Java require only a Java-compatible

Web browser for viewing. Such browsers like Internet Explorer or Netscape are readily available, easy to install and already installed on many computer systems. Since many users are familiar with navigating Web browsers, the total software learning curve using a model is a minimal.

Furthermore, web-based simulation environment provides the same user interface for all users. Although some non Web-based groupware (computer based systems, that support groups of people engaged in a common task) simulation packages provide a common interface to a shared environment, these software solutions have two main shortcomings. Firstly, most of these packages are system dependent, all participants in a project are bound to use a system that is supported by the groupware application. All participants must own the software to use. The second disadvantage of any existing groupware is the lack of simulation awareness. It will support the communication of project participants, exchange of files, but it is still detached from the actual application, simulation.

Efficient Maintenance: Web-based simulation programmes enable reliable version control and frequent model modifications. In most of the traditional groupware packages, modifications can be tedious and time consuming. Users must be made aware of the need for and existence of an updated model. Then update must either be made on each copy of the model, or the new model should be delivered to each participant and the old one deleted. In Web-based modelling there is only one model at a particular time that is residing on the Web-server. The existence of a single working model enables modifications with smaller degree of error. Additionally, the model creator has access through the Internet to the model's source code on the serving computer. This allows that person to access the server from distant site. Also the most up-to-date version of the model is instantly available through the server to all authorized users, regardless of the physical location.

Despite these advantages of WWW based simulation, traditional simulation packages may be useful over Web-based simulation in many cases. Traditional packages are specialized to idiosyncrasies of a single platform, making maximum use of its capabilities. This may increase efficiency of simulation runs. Loading times for traditional programmes are dependent on the computer and not on the current volume

of Internet usage. During heavy Internet traffic, models with complex or extensive amounts of code may initially take large amount of time to download.

1.6 Objective of the Project

The objective of the project is to develop a virtual reality based simulation software (*SimDynamic*). Proposed simulation engine will be powerful enough to handle both discrete event simulation and dynamic system simulation. In case of dynamic systems, it will support linear and non-linear systems, modelled in continuous time, sampled time, or a hybrid of the two. Systems can also be multi-rate, i.e. have different parts that are sampled or updated at different rates. Java will be used to develop the package, so it will be accessible over the Internet. Using any java enabled browser, a simulation problem can be modelled in *SimDynamic*, and such java-based models allow the user the same degree of interactive, multimedia capability, as do non-Internet based programs. *SimDynamic* can also be used as a non Web-based application on a PC. For modeling, the package provides a graphical user interface (GUI) for building models as block diagrams, using click-and-drag mouse operations. After a model has been defined, it can be simulated, using a choice of integration methods. Finally the results of the simulation can be viewed as both graphical and numerical output as well as 3D animation.

This thesis is organized into seven chapters and five appendices. A literature review is presented in chapter two. It mainly covers previous work carried out by different researchers in the area of virtual reality & simulation, dynamic system simulation and Web-based simulation. Chapter three introduces *SimDynamic* from a user's point of view. This chapter provides all the information needed to build a model, execute and handle the GUI. Chapter four describes the architectural design of the software. Chapter five can be considered as an extension of chapter four, but they can never be merged. This chapter describes the functional ability of the core components of *SimDynamic*. Chapter six focuses on the application of the software. Some real world systems have been modelled using *SimDynamic* and this chapter demonstrates the results obtained from the simulation. Finally chapter seven outlines the conclusions and the recommendations for further work.

Chapter Two

2 Literature Survey

2.1 Virtual Reality and Simulation

The popular media representation of Virtual Reality has tended to emphasize the entertainment aspects of the technology. But a virtual environment can be used as an effective tool for simulation, training and education. Barnes [3] provides an awareness of Virtual Reality with respect to simulation. The newer workstations and PCs provide performance that support VR at an affordable price. VR software is now available to run on these platforms. Some of this software comes with simulation engines that support simulation modelling and analysis. The paper describes the tools and methods that are particular to VR and illustrates how these are being applied to simulation. The applications presented in the paper focus on virtual manufacturing and the VR world animated with behaviour controlled by a simulation engine, which uses simulated behaviour rules and model data.

Sikorsky Aircraft Co. USA and British Aerospace, UK, used simulation software like QUEST [4], IGRIP [5] to recreate the actual manufacturing process. Visualization allows the user to assemble the components in a Virtual Manufacturing cell. This could be by utilizing the 3D models of the part designs using popular CAD packages, such as Unigraphics, Intergraph, CATIA or Pro/ ENGINEER to visualize how the components relate to each other. In the model, the user can manipulate and view each of the components to develop a manufacturing approach. Virtual Manufacturing modelling allows the user to introduce the engineering design to the processes that will be used to create the actual part, assembly, or installation. The tool design is imported to the model, combined with the part and dynamic representations of the machines that will produce the part. Machines are programmed to operate identically in the virtual environment as they would in the real environment. Design problems such as collisions, clearances, missing manufacturing features, fit, and manufacturing sequence issues are quickly identified. Manufacturing and design concepts are easily developed with minimal cost to the program. The ability to insert people in the environment and analyze their activities minimizes ergonomic problems. Factory

modeling allows engineers to predict the cost and schedule impact of potential design or process changes. The information provided by the simulation enables "what if" analysis. The model provides support personnel with a tool to optimize the manufacturing, try new concepts without disrupting manufacturing and predict changes due to load variations. The three common elements in the evolution of a product are cost, schedule, and information. Virtual Manufacturing allows engineers, designers, suppliers, and others to understand the cost and schedule impact of decisions and to consolidate processes. Overhead costs are addressed by improving schedule performance. The amount of time it takes to design, plan, tool and manufacture a product directly impacts overhead costs. A reduction in time it takes to get a product to market equates to a return on investment (ROI) improvement.

General Dynamics Electrical Boat Division for the US navy developed prototypes of a nuclear attack submarine, which was designed, evaluated and optimized in a virtual environment. Electric Boat demonstrated the feasibility of state-of-the-art simulation-based design (SBD). The objective was to implement an accurate, efficient, and dynamic environment for design, rapid prototyping, concurrent product and process development, mission planning, operation maintenance, and training. By simulating the kinematic, dynamic, mechanical, and other characteristics of the submarine, its components and subsystems, engineers can create a multi-disciplinary environment in which to evaluate a wide range of parameters and optimize the design based on the results. Electric Boat selected simulation software ENVISION [5] to visualize design concepts and integrate components and subsystems. Virtual engineering allows the parties involved with design, manufacturing, operations, and maintenance to jointly contribute to the design process early in the cycle. Electric Boat engineers often experience communication difficulties when trying to describe innovative ideas to people not trained in their particular discipline. The physically accurate 3D models simulated in ENVISION help engineers and technicians from various disciplines to comprehend new ideas and concepts. In addition, virtual reality technologies such as immersion enable engineers and their naval customers to "walkthrough" the model with visual feedback. Immersion provides for evaluation other design criteria, such as ergonomics, before a design is built.

The most recent development in the use of Virtual Manufacturing software is the use of the Virtual Collaborative Engineering (VCE) environment that links multiple users at multiple locations to discuss, analyze, and review simulations over a wide area network. VCE users interactively evaluate design concepts, manufacturing tooling, processes, and factory layouts — even at geographically remote locations. Any VCE user can assume control of a simulation, make changes, or view changes made by others on the VCE network. Engineers, manufacturing personnel, system operators, or other users interact within the same simulation, creating a "virtual conference room" for design discussion, evaluation, and approval.

Jones, *et al* [6] describe a software engineering project, initiated to explore how virtual reality might impact simulations and to gain insight into bringing the technology to commercially available simulation software packages. AutoSimulations, Inc. [7] provided licenses and source code to their AutoMod and AutoView simulation software for the project. Human Interface Technology Laboratory (HITL), University of Washington, provided the virtual interface hardware, software, and necessary software engineering support. The VR software used in the project was actually two components of a suite of virtual reality software developed at HITL: VEOS (Virtual Environment Operation System), a distributed database and data transport package created for research and rapid prototyping of virtual environments, and Image Library created for research into VR related rendering issues. The objective of the project was to create virtual factory that would allow a user to experience a three-dimensional "playback" of a simulation. The user could move around in the factory but would not be able to make changes to the simulation other than starting and stopping simulation time. The user has to don a head-mounted display (HMD), which conveys visual information to each eye via two separate video screens. A tracking device allows the VR system to monitor the user's head position and orientation at all times. A computer with special graphics hardware continually renders for each eye a perspective image of the virtual environment from the eye's instantaneous viewpoint, based on the user's head location. When the user sees these images through the HMD, the images fuse to form a single, stereoscopic view of the virtual environment. The user can turn in any direction and see objects in the environment in the same way that one can turn one's head and discover what is behind him. Additionally, the user can move or 'fly' through the environment by

means of a hand-held joystick. For testing the project, a simulation of the wheel assembly area of Derby Cycle, a producer of recreational and sport bicycles in Kent, Washington, was constructed. Production and manufacturing personnel were used as subjects for evaluating the interface. Five subjects from Derby Cycle participated and were shown either the virtual environment or the AutoView animation on a computer screen. Subjects were asked questions to determine their understanding of the virtual factory and follow-up written survey and group discussion was also conducted. One notable conclusion from the survey and group discussion was that the virtual factory captured the imagination of the Derby employees. Participants who viewed the virtual factory were much more talkative and involved in the exploration of the simulation than the subjects who experienced AutoView.

Marcedie, *et al* [8] cite many recent examples of virtual environments that can be used as effective tool for training and education. The paper also introduces a novel technology to support distributed virtual environment. Existing VR systems lack distributed access; they have no central computer for event scheduling or conflict resolution. Computers (or autonomous simulation nodes) on which the system is running are responsible for maintaining the state of one or more simulation entities. Simulation nodes communicate primarily changes of their state and they are responsible for determining what is perceived. Such a system suffers from high communication volume. The proposed new architecture overcomes these problems of existing systems by taking advantages of the features of the object oriented programming paradigm.

As an object oriented simulation system, the basic system elements in the proposed architecture are objects located at different machines. These machines are called nodes and are connected by the network. At any single node, there are three major objects.

a) **The message handling object** controls the messages passed between local objects and the network. Unlike current approaches, which broadcast all messages on the network, it uses both broadcast and point-to-point approaches to send the messages to keep the number of redundant messages as low as possible.

b) With update messages, **the situation delay object** is used to display the states of all entity objects distributed at the nodes. The display updates with simulation time advance.

c) **The simulation object** at a node contains a series of entity objects, which represent physical entities in the real world. It is responsible for creating and deleting its derivative entity objects. It transfers the messages from its derivative entity objects to its other derivative entity objects or to the other objects located at other nodes via the computer network. Another important task of the simulation object is to control the local simulation time advance.

In short, this prototype object oriented architecture is composed of non-autonomous simulation nodes. With an appropriate user interface, the nodes appear as an integrated virtual environment to the user. Simulation entities are modelled as simulation entity objects, which are distributed at different nodes and are migrated dynamically between the nodes during the simulation run. This mechanism of object management presents the possibility of reducing message volume. Communication efficiency can therefore be improved by two measures. One is to dynamically group closely related simulation entity objects to a specific node. Alternatively, a point-to-point communication approach can be used instead of broadcast.

Grant and Lai [9] present SMART, a simulation modelling tool that provides a virtual reality interface for building graphical simulation models. The simulation models, comprised of nodes and arcs, are constructed in three dimensions. As the user builds a model, he may immerse himself in it using virtual reality hardware and software tools and take advantage of the three-dimensional environment provided by SMART. It has been developed to explore the use of virtual reality in building simulation models, and it serves as a prototype for testing the feasibility of creating a virtual reality simulation modeling software system on a relatively low cost personal computer. SMART offers three-dimensional interface using virtual reality hardware, which includes an electronic glove and head-mounted display. The specific hardware is the 5DT Glove [10] and VIO I-Glasses [11] respectively.

The 5DT Glove is used by SMART as the primary manual input device. The electronic glove is plugged into a PC's serial port. The configuration parameters of the 5DT Glove such as the bending angle of each finger and, pitch and roll of the wrist are constantly sampled by the PC's serial connection and sent to SMART for processing. The user controls SMART using a set of gestures to cause actions to be taken when building simulation models. When a recognizable gesture is detected, SMART responds with the appropriate action and provides audio feedback confirming the action. The glove's configuration and its relative position in the virtual world are continuously animated by a robot-like hand (figure 2.1). Every motion of the user's fingers is reflected by the animated hand in the virtual world.

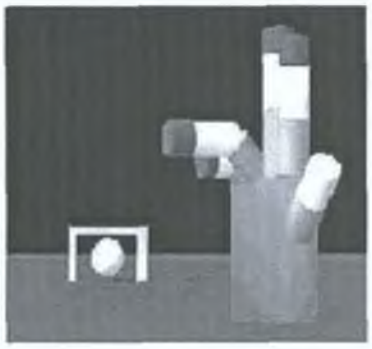


Fig 2.1: Animated hand for 5DT Glove

Fig 2.2: VIO I-Glasses pitching

Fig 2.3: VIO I-Glasses yawing

In addition to the 5DT Glove, SMART also uses the VIO I-Glasses as another virtual reality hardware interface especially for simulation model immersion enhancement. The device is plugged into a PC's serial port. The VIO I-Glasses are designed to give the user the impression that he or she is physically present in the virtual modeling world. This is accomplished by providing a virtual view using the VIO I-Glasses, which reacts directly to two primary head motions: pitching (figure 2.2) and yawing (figure 2.3). Pitching is equivalent to nodding the head up and down and yawing is swinging the head left and right. To use the I-Glasses to build simulation models, the user simply needs to put on the VIO I-Glasses and look around the way he usually does in the real world. As the orientation of the user changes, the virtual world is rendered accordingly.

2.2 Dynamic System Simulation

Kimbrough [12] provides APL algorithms for control system development and demonstrates their use by solving a typical control problem. The paper outlines useful numerical techniques for simulating dynamic systems and for solving some of the central equations of the control theory.

APL functions are presented to check observability, controllability and stabilizability of a linear dynamic system expressed in the following standard form,

$$\frac{dx}{dt} = Ax + Bu + D(x,u,t)v$$
$$y = Cx$$

where the shapes of *A, B, C, D, x, y, u* and *v* are *(n,n), (n,p), (n,q), (m,n), n, m, p* and *q* respectively. The vector *x* is the state vector, the vector *y* is the measurement vector, the vector *u* is the linear control, and the variable *v* is the non-linear control. The matrices *A, B* and *C* are time invariant but *D* is allowed to vary with *x, u* and *t*, the time.

Simulation of dynamic systems is typically conducted by using a numerical integration routine. A fourth-order Runge-Kutta integration routine is also provided in the paper. This is a fixed step routine based on the following familiar equations,

$$k_0 = f(y,t)h$$
$$k_1 = f(y + 0.5k_0, t + 0.5h)h$$
$$k_2 = f(y + 0.5k_1, t + 0.5h)h$$
$$k_3 = f(y + k_2, t + h)h$$
and
$$y^{n+1} = y^n + (k_0 + k_3)/6 + (k_1 + k_2)/3$$

Besides using simulation it is possible to study the system response using frequency domain techniques such as Fourier transform and the Laplace transform. Fundamental to frequency techniques is the notion of the transfer function. For multi-variable time-invariant linear systems the transfer functions are obtained by taking the Laplace transform of

$$\frac{dx}{dt} = Ax + Bu$$
$$y = Cx$$

which yields,

$$y = C(sI - A)^{-1}Bu = H(s)u\,.$$

An APL function is also provided to generate *H(s)*.

All the algorithms presented in the paper can analyze, with reasonable response, systems with up to 40 state variables while running on a mainframe and systems with up to 20 state variables while running on a microcomputer. This capacity is sufficient for many "real world" control problems and is more than enough for exploring the features of control engineering.

Schmid [13] presents CADACS-PC Real Time Toolbox, an open software implementation of the KEDDC [14] real-time suite, which runs on personal computer and provides special real-time environments for developing microprocessor based industrial control devices. Such control devices can be connected directly to the plant to be controlled and support on-line identification of the plant dynamics, as well as evaluation of prototype control systems. The proposed software frame is independent of the hardware and operating system, quite simple and also meets the requirements for a real time control environment, such as monitoring, supervisory control or scheduling etc. Although such a frame is contained in the real time suite of the KEDDC system, the contribution focuses to the implementation of the real time suite on PC-based systems, where the entire power of KEDDC is packed into one low-cost and portable system, which can be taken to the plant to be analysed and controlled. CADACS-PC supports a wide variety of control engineering methods. It offers tools for the systematic design strategies and high-accuracy numerical solutions. In addition to modern analysis and design concepts, classical methods are also implemented and may be used in combination with modern control techniques. CADACS application environment consists of several components and the most important components consist of a group of interactive programs for the management of dynamic systems, systems analysis, controller synthesis, simulation and signal processing. A unified database for system models allows a smooth exchange of data between the different programs and between different groups of a development or research project. Due to

the wide range of methods, the system may be used for the task of industrial planning of plants, as well as for the subsequent analysis of measured data; also for the modeling of dynamic systems and for the design of control systems. The multi-purpose real-time suite has highly modular structure and provides a high degree of portability, which results from an interface of CADACS to the operating system and to hardware-dependent functions. Some basic routines build an interface to the world outside of the programs and modules. The user can both monitor and store the real-time behavior of any external or internal signal. At the same time or at a later date s/he can evaluate and present results.

Fishwick [15] develops SimPack, a collection of C and C++ libraries and executable programs for computer simulation. The purpose of the SimPack toolkit is to provide the user with a set of utilities that illustrate the basics of building a working simulation from model description. Special purpose simulation programming languages can be easily constructed using language translation software with the SimPack utilities, which act as the "assembly language".

Most of the existing simulation packages cover one of two areas: discrete event or continuous. Some available software can perform both types of simulation; however, bulk support is usually available in only one form. SimPack has the ability to overcome this problem. It is designed

- To support the variety of available model types.
- To create template algorithms for many cases.
- To avoid learning a special language syntax.
- To illustrate the relationship between event and process oriented discrete event simulation.

SimPack supports simulation development for a wide variety of modeling types including the following:

- **Declarative Models:** An emphasis on explicit state-to-state changes as found in finite state automata and Markov models.

- **Functional Models:** A focus on "function" or "procedure" as in queuing networks, block models, pulse processes and stochastic Petri nets.
- **Multimodels:** Defined as conglomerates of other models connected in a graph or network to solve combined simulation problems at multiple abstraction levels.

- **Constraint Models:** The constraint part of SimPack includes capabilities for modeling 1) difference equation systems, 2) differential equation systems, and 3) delay differential equation systems. In most cases, the most uniform method of simulation is to convert the equation(s) into first order form and then to simulate the system by updating the state vector.

Otter [16] presents DSSIM, a general-purpose simulation for dynamic systems. DSSIM is a part of ANDECS [17] environment, which is a powerful and flexible software package for the analysis and design of controlled dynamic systems. ANDECS consists of a wide collection of methods, such as basic mathematical methods like matrix computation, interpolation of signals or root finding of nonlinear functions. Analysis and design methods for linear dynamic systems like linear simulation, calculation of poles and zeros, pole placement. Standard diagrams like 2D line, Bode, Nyquist and root locus diagrams. Special diagrams like parallel coordinates to visualize optimization criteria. DSSIM is the run time environment of ANDECS to simulate dynamic systems. It uses following well-tested numerical integration routines,

- DEABM – Multi-step solver of Shampine/Watts for non-stiff and moderately stiff ODEs.
- LSODE – Multi-step solver of Hindmarsh for stiff and non-stiff ODEs.
- LSODAR – Multi-step solver of Petzold/Hindmarsh, which switches automatically between a non-stiff and a stiff integration algorithm along the solution. LSODAR also provides a root finder.
- RK45/78 – Runge-Kutta-Fehlberg solvers of Kraft/Fuhrer of fixed orders 5 and 8 with variable step-size using the Prince-Dormand coefficients.

- GRK4T – A stable linearly implicit Rosenbrock type single-step solver of fixed order 4 for stiff and oscillating ODEs of Arnold.
- DASSL/RT – Multi-step solvers of Petzold for DAEs and for DAEs with root finder.
- ODASSL/RT – Multi-step solvers of Fuhrer based on DASSL/RT of Petzold for ODAEs and for ODAEs with root finder.
- MEXX – Extrapolation solvers of Lubich for a restricted class of index-2 ODAEs.

There are a wide variety of options available to define a simulation experiment. The result of simulation experiment is a set of computed signals, which are automatically stored on a database and visualized with any available graphics module. All input data of an experiment, e.g. integration method or length of communication interval, are stored on database as well. Therefore every simulation run is completely documented and reproducible.

Elmqvist, *et al* [18] present a new methodology for object-oriented modeling of hybrid systems. Hybrid models contain both continuous and discrete parts. In simulation programs, the continuous parts are described by sets of differential equations and algebraic equations in either explicit form or implicit form. The discrete parts are expressed with event descriptions. Object-oriented programming has evolved to support the reuse of software components. This programming paradigm was first developed in the context of discrete-event simulation and carried over to continuous systems modeling. Dymola [19], an object oriented modeling language, for continuous systems, was designed for this purpose. It represented an important step forward towards the reuse of continuous systems models in a truly environment-independent fashion. A continuous system modeling methodology that doesn't allow for descriptions of discontinuities is not generally useful since all but the most trivial engineering models of dynamic systems contain some sort of discontinuities. The paper discusses an extension of Dymola language definition to allow descriptions of models of dynamic systems with discontinuous behavior in a truly reusable object-oriented fashion.

There are some packages like VisSim, Simulink, Maple available in the market for dynamic system simulation. Both VisSim and Simulink support modelling and simulation of complex continuous non-linear dynamic systems. They combine an intuitive drag and drop block diagram interface with powerful simulation engine that provides fast and accurate solutions for linear, non-linear continuous time, discrete time, time varying and hybrid system designs. The visual block diagram interface offers a simple method for constructing, modifying and maintaining system models.

2.3 Web Based Simulation

Research on distributed interactive simulation; its feasibility and application began in the mid 1970's. In the 1980's Miller and Thorpe [20] in partnership, with the US Army created SIMulation NETworking (SIMNET), a networked system of computers running a single simulation programme. SIMNET was sponsored by ARPA (then called DARPA), and it was an attempt to make the use of simulators and simulation techniques more feasible for military defense operations. This programme demonstrated the feasibility of linking together hundreds or thousands of simulators (representing tanks, infantry fighting vehicles, helicopters, fixed-wing aircraft etc.) to create a consistent, virtual world in which all participants experience a coherent, logical sequence of events. In SIMNET, standard Ethernet networks were used for all local area network (LAN) connections. Ethernet bridges were used via 56 kbps dial-up links to connect two or more LANs to form a single, logical Ethernet LAN. The success of SIMNET resulted in a standardized simulation networking protocol, Distributed Interactive Simulation (DIS). After several revisions and extensions of the SIMNET protocols, in March 1993, the first standards were formally approved [21]. DIS is a protocol for communicating position and other information to other entities in a simulated battlefield. Each entity can see each other and interact in a virtual environment. Although DIS began in the military environment, it is now being used increasingly often in non-military applications such as air traffic control, intelligent vehicle highway systems, and interactive multi-user computer games.

Neilson and Thomas [22] discuss the use of the Interact Simulation Environment (ISE), created as an aid to teaching engineering students. At the heart of ISE is an interactive environment in which a simulation can be integrated into a distributed

hypermedia system. Such integration permits a simulation to be treated as just another medium of expression, equivalent to media such as the text, graphics and sound. It allows a simulation to be treated as a resource along with other resources such as video clips, graphics etc. It allows a simulation to be used not only with a wide variety of supporting material but also in a wide variety of contexts thus increasing the usefulness of the simulation as a learning resource. Some additional contexts are: use as a stand-alone simulation in a lecture with no supporting hyper textual material; use within a laboratory class for demonstrations, structured enquiries, open ended enquiries and projects; use in studies of parametric changes; use in optimisation of system parameters and in tests of sensitivity; giving students experience of modelling systems; input and output datasets can be provided from which students have to deduce the model which gave rise to that output; given a specification students could be asked to provide a model to achieve that specification. Students could be required to produce hypertext reports of their work, which include links to the simulation in various states. Because of differences in student's background computer knowledge, the ISE is set up to utilize simulation as a modelling tool without requiring the student to interact with the actual simulation interface.

Cole and Tooker [23] have developed WWW-based physics tutorials to assist physics students. Making these simulation models available over the WWW greatly expands the range of possible access locations. Like ISE, the physics tutorials allow students to see interesting cases of a given simulation model without requiring prior knowledge of the parameters defining these cases or of the background programming languages involved. Additionally, the use of familiar WWW browsers such as Netscape virtually eliminates the amount of time necessary for distributing and learning viewing software. The tutorials use Apple's OpenDoc Frameworks to provide a basic simulation environment. OpenDoc is based on the idea of container documents and components that act on documents. Using OpenDoc one can build software components, in this case physics simulation models. It can be then embedded in container documents such as a web browser document or word processing document depending upon the need of the user. The main attribute of the environment is its extensibility. One can extend the power of the environment by adding parts. This extensibility means it is functional for a broad spectrum of users.

Fishwick [24] focuses on three important aspects of simulation that might be affected by incorporating it in the Web:

a) **Education and training:** Using the WWW for education and training allows and encourages the reuse of knowledge by providing us with effectively infinite storage, unlike the CD-ROM and diskettes. The storage is on the Internet and not limited to one's own machine. Therefore simulation packages, which include help text and information about the pieces of the model, can have these pieces on the Web so that they do not require local storage. It is not necessary for a company to re-build every piece of information about a manufacturing process, for instance, from scratch. Information on automated guided vehicles (AVGs), machine specifications, and automated conveyance mechanisms may already be located somewhere on the Web. These sorts of devices are common in simulation programs built for manufacturing analysis. The Web encourages this sort of global view with a reuse of knowledge and information. The most profound affect the Web may have on the method of teaching simulation lies with the use of multimedia. The Web encourages distance learning more so than the typical simulation textbook. On any web page it is possible to include images and video of the instructors along with synchronized slides or overheads. This immerse the student in a synthetic learning environment that is more congenial than one they would get simply by reading a book or watching a videotape.

b) **Publications:** Additionally, he discusses the timesaving aspects of using WWW tools in reading publishing research articles. The WWW is valuable in all stages of producing an article, from accessing documents via electronic database or URLs cited in bibliographies, to transmitting electronic copies of the publication to reviewers and for the final publication. Simulation models can be embedded in web documents that can contain videos, images, and audio in addition to the usual text that traditional documents contain.

c) **Simulation Programmes:** Typically scripting languages such as Perl, Java Script, or Java are used to build the model. The paper gives an example of a

simulation model using Perl script to demonstrate current workstation resource, queuing sizes based on user input gathered through an HTML form. Besides the value of simulation in aiding understanding, Fishwick discusses the possibilities of WWW based simulation or simulation interfaces for multi-user situations such as multi-user dungeons (MUD), where users typically co-operate to reach a solution, and DIS discussed earlier.

SimKit, created by Buss and Stork [25] is a set of Java classes for creating discrete event simulation models. They use the event graph design approach to build models. SimKit models, particularly geared to military applications, can easily be implemented as applets and executed in a web browser. The main simulation facilitating package, JavaSim, is designed to allow for expansion in order to accommodate frameworks for various types of simulations. JavaSim makes extensive use of Java interfaces, which add defined behaviours to identified classes without imposing the hierarchical structure of class inheritance. This structure allows a modeller to add customized tools into SimKit without making changes to JavaSim. SimKit permits user interaction through a detailed, model entry form. Additionally, SimKit is combined with a Java based graphic package designed by Leigh Brookshaw, to allow a useful output of statistics and graphs.

Miller, *et al* [26] present JSIM, a java-based simulation and animation environment. It demonstrates WWW-based simulation through a unique combination of java applets and query driven databases. JSIM provides the user with a simulation and animation environment in which she/he can build simulation, execute the models, watch the animation of a static design diagram, embellish the animation with special icons, and generate statistics. A model is constructed by building a graph with nodes and edges. Using the built in features of JSIM, the design diagram may be animated when the simulation is run. For the purpose of storing and retrieving simulation models and results, JSIM incorporates database connectivity with simulation classes. When a user queries a simulation system, the system first tries to locate the required information in the database, since it might have stored as the result of an earlier model execution. If the required data is present, it is simply retrieved and presented to the user. If it is not present, the system instantiates the relevant model, executes it and shows the result of the execution to the user.

SimJava, presented by McNab and Howell [27] is a process based discrete event simulation package for building working models of complex systems, with animation facilities. It is actually a collection of three packages, *simjava*, *simanim* and *simdiag*. *simjava* is a package for building stand alone text only java simulations, which produce a trace file as the output by default. *simanim* is tightly integrated with the text only simulation package, and provides skeleton applet for easily building a visualisation of a simulation. *simdiag* is a collection of JavaBeans based classes for displaying simulation results. A SimJava simulation model is a collection of entities (from the Sim_entity class) each running in its own thread within Java Virtual Machine (JVM). These entities are connected together by ports (from the Sim_port class) and can communicate with each other by sending and receiving event (from the Sim_event class) objects. A central system class, Sim_system, controls all the threads, advances the simulation time and delivers the events. The progress of the simulation is recorded through trace messages produced by the entities and saved in a file.

Utilizing the Remote Method Invocation (RMI) facilities of Java, Page, Moose and Griffin [28] extends the SimJava (as described earlier) simulation package to enable the construction of distributed, web-based simulation. In the Java distributed object model, a remote object is one whose methods can be invoked from another JVM, possibly on different hosts. An object of this type is described by one or more *remote interfaces*, which are Java interfaces that declare the methods of the remote object. A client can call a remote object in a server, and that server can also be a client of other remote objects. To add the RMI capabilities to SimJava, a simple master-slave architecture has been designed. In this architecture, a master server is one that encapsulates the central system class (Sim_system) and this class co-ordinates the activities of objects (Sim_entities) that are distributed across the network. Given this architecture, remote interfaces must be constructed for the Sim_system, Sim_entity, Sim_port and Sim_event classes. To illustrate the concept of the architecture the paper also provides a self-explanatory example.

Lorenz, *et al* [29] describe three basic approaches, their advantages and disadvantages, for Simulation and Animation (S&A) on the web.

a) **Remote S&A:** Through an HTML data entry form the user specifies parameters for a simulation model. A Common Gateway Interface (CGI) transfers the data to server; an appropriate CGI script starts the simulation and prints the results in a new HTML page when the simulation is finished.

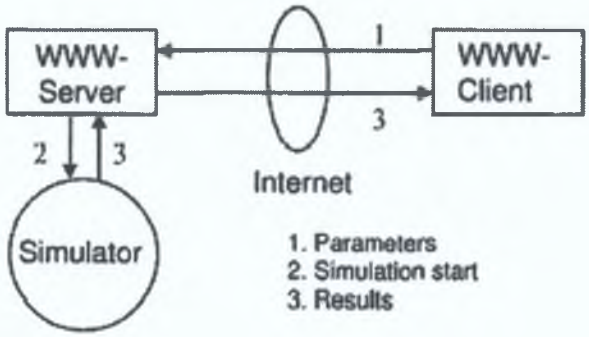


Fig 2.4: Remote S&A and data transfer

This approach is suitable for existing S&A software, but user has no interaction with a running simulation.

b) **Local S&A Based on Loading Applets:** the user loads a Java applet into the web browser, runs the simulation on the client machine.

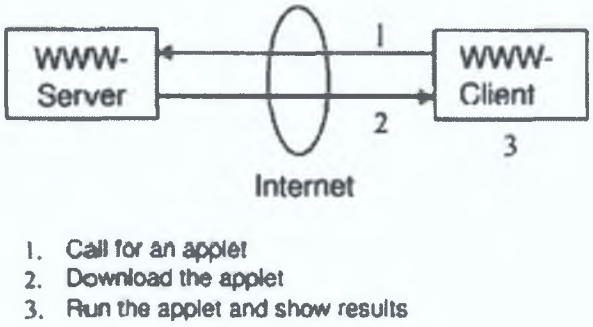


Fig 2.5: Client-Site simulation with loaded applets

It supports user interaction and animation, but not suitable for the existing S&A tools.

c) **Animation and Manipulation using a Java Data Server:** the user begins by loading some applets; a connection is then built to a Java server. The simulation runs remotely on this server, the results are transferred to the client

browser, and visualized locally. The user can interact with the model by using buttons on the HTML page.

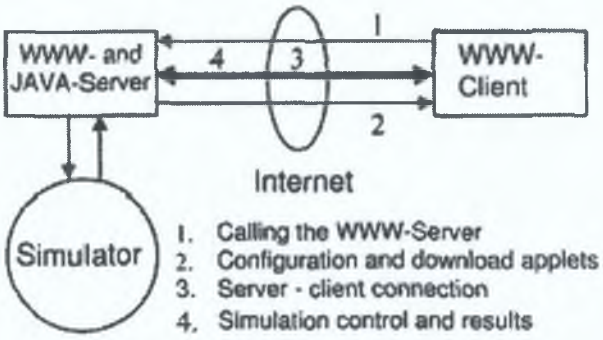


Fig 2.6: Remote simulation and local visualization

Finally the paper introduces Skopeo, a Java-based, platform-independent 2D-Animation system.

Yücessan, *et al* [30] introduces a Parallel Discrete Event Simulation (PDES) support system to distribute simulation experiments over the Internet. There are several approaches to exploiting parallelism in discrete event simulation; unfortunately each of those has its own drawbacks. For example, a) in Dedicated Execution approach, where dedicated functional units execute specific sequential simulation functions, the speed tends to be limited as the management of the future events list becomes the bottleneck. b) in Hierarchical Decomposition, where a model is decomposed in a hierarchical fashion such that an event consisting of several sub-events to be processed concurrently, is largely model dependent. c) in Parallel Replication, where several replications of a sequential simulation are executed independently on different processor, each processor must contain enough memory to hold the entire model. The paper describes a prototype of a software architecture that would support PDES. The main objective of the proposed system is to plan parallel simulation experiments and dynamically manages their execution. Via the Internet the system has access to a large number of sites (a PC or Workstation), from where it can dynamically selects one to execute part (parallel replication) of an experiment. The system uses OCBA [31] to implement the experiment planner. The system has also data collection and analysis capabilities to collect and analyse output data from each individual processor. Finally,

the system contains a reporting mechanism, which compiles and presents the result in a multimedia format, consisting of tabular and graphical summaries.

Veith, *et al* [32] presents Netsim, a discrete event simulation package based on the event graph approach. Netsim provides a maximum amount of user interaction with the simulation model. A programming interface provides a blank template with text fields for the various parameters of a simulation model, such as event name and state variables. A second interface allows user interaction with running the simulation model. This interface not only provides start, pause and stop capabilities and data output, but also an animation of the model. The object-oriented structure offered by Java and maintained in Netsim allows easy expansion of the package as well as compatibility with other Java-based tools.

Elmaghraby, *et al* [33] demonstrate a software model that can be used for transforming a legacy simulation system into Web-accessible application. The paper describes the initial results from the conversion effort on an existing distributed simulation visualization system. There are two potential methodologies to run application from the World Wide Web.

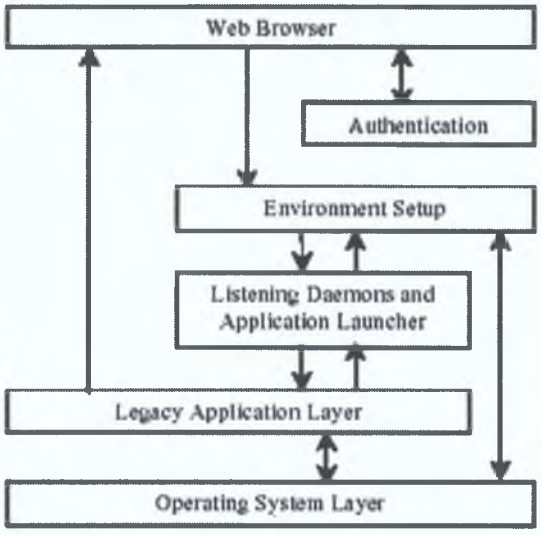


Fig 2.7: Web-enabling software model

a) to develop the application to be web aware using Web development techniques such as: HTML, Java, CGI etc.

b) to transform an already existing application, which was not designed for the web, into a Web-accessible application.

But there are a number of problems to convert an existing application to be web accessible, such as limited environment, security risks, concurrency etc. To answer all this problems a Web-enabling software model has been introduced. The proposed layered model (figure 2.7) has several layers to link an existing application to a web browser. The Web browser layer is the interface layer where users can interact with the application from the Web. Authentication layer authenticates the user and validates the login name and password. The environment set-up layer will control concurrency, create run profile and talk to the listening daemons layer, which is the heart of the model. Listening Daemons layer receives user request and the start-up settings then launches the Distributed Simulation.

Salisbury, *et al* [34] discusse a browser-based simulation visualizer that has the ability to display simulated entities to any number of distributed sites. Legacy visualizers have a number of drawbacks, such as, the tool must be present at the inception of the simulation, no matter the observer is only interested to observe a relatively small time slice. Most existing visualization tools are dependent on the machine for which the application was written. The paper introduces a tool Jview-lite to solve these problems. Jview-lite is the successor of a modular programme Jview, which supports the simulation community's diverse set of visualization requirements. Jview relies on Sun's Java 3D architecture to render an image, and the use of Java Beans enables it to display objects from any data source. Jview-lite utilizes the same display technology but relies on the Web for distribution. The client server architecture, proposed for Jview-lite, has two principal software components: the Simulation Monitor and the Simulation Viewer. The Jview-lite simulation viewer is an applet executing within the client's web browser. On receiving a request from a client's browser, the web server will send the required page and the embedded viewer applet. The browser then initializes the viewer applet, effectively making the browser a 3D-simulation viewer. The viewer, in turn, makes contact with the simulation monitor that is residing on the same machine as the web server. The monitor passes object information to the client. The simulation viewer can then render the simulation based on this information.

Schmid [35 - 40] presents a remote laboratory approach using virtual reality on the Web, where an audio-visual interface together with a collaboration tool gives the feeling of being in the laboratory. Most existing virtual labs offered across the Web include several fully interactive experiments completely written as Java applets, but can't claim to be an equivalent substitute of the real experiment. To solve the problems of this "close-to-reality" environment and to take the full advantage of a remote virtual laboratory, preference is given to an architecture where all computational work is performed locally using some existing computational machines like MATLAB/SIMULINK/MAPLE and the output is linked to the objects of VR scene. One main task to accomplish this is to tie together the computational machines of MATLAB/SIMULINK/MAPLE with Web components to a hypertext-based system, which runs on the Web browser. The proposed client/server architecture uses the following plug-ins for this purpose: MATLAB plug-in integrates MATLAB operation into the Web browser, the Graphics plug-in extends the MATLAB plug-in to view MATLAB or Metafile graphics, the DDE server plug-in, together with the VRML plug-in, is for viewing and acting with VR scenes. The paper concludes with some examples to demonstrate the architecture.

Guru, *et al* [41] introduces a web-based interface for storing and executing simulation models, developed using SIMAN simulation language, over the Internet. This toolkit consists of a World Wide Web interface to SIMAN and a web-accessible database for storing the models. To use the system, a user simply needs access to the Internet and an account on the web server. It allows user to develop, execute and test the functionality of a model through the use of the SIMAN debugger/run controller. The proposed system involves three servers.

a) **Application server:** An Internet user accessing the Web address for the site is directed to this server. The software residing here performs the functions of authenticating the user and providing access to user's models in the database. It gives the user the capability to edit, add or delete models in his/her account. When user wants to compile, execute or debug a model, this server creates an appropriate HTML document (dynamically-generated), which contains the

instruction of the desired operation and pass it to the SIMAN server on user's request.

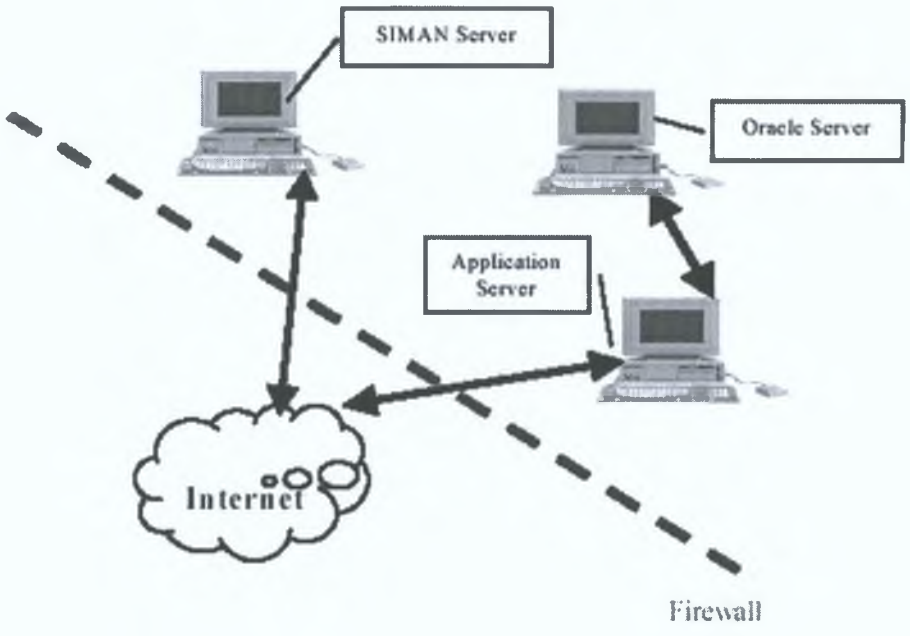


Fig 2.8: Hardware and software layout

b) **Oracle server:** The application server interacts with the Oracle server in order to provide accessibility to the user's models. The models and the user data are stored in a database.

c) **SIMAN server:** This server holds the SIMAN simulation software. The SIMAN simulation software performs the requested operation and returns the results to the user in the form of an HTML document. This done by piping the output of the appropriate SIMAN process that are invoked with the CGI script as a dynamically-generated HTML document. The arguments to the SIMAN process are ASCII text files containing the SIMAN model. These files are temporarily created by CGI scripts with the data posted to them through the HTML forms.

Marr, *et al* [42] discusses a Web-based simulation manager program, called SimManager, which executes simulation study in a parallel-replications mode, utilizing a set of slave processors available to it. Slave (engine) processors need not be

co-located. They can belong either to an intranet, to the Internet, or to a combination thereof. An engine is simply known to SimManager by its IP address. SimManager also knows the performance characteristics of each engine. SimManager would serve as the interface between a customer seeking assistance in simulation modelling and the simulation system. When the customer accesses the web page, he would click on the Catalog of Simulation models and view a description of any models that appear to meet his specific needs. SimManager receives this request for work and places it in the job queue. After determining what engines are available for work it assigns this workload to the engine processor. It also receives back a file containing the statistical results derived from the engine's efforts, and organizes these results into a form that meets the needs of the customer. Once all the jobs associated with the customer request are complete, an email message is sent telling him that the results are ready for pick up.

Rao, *et al* [43] present WESE: A Web Based Environment for Systems Engineering. The design and development of modern systems is complicated by their size and complexity. Furthermore, many of today's complex systems are built using subsystems and components available from third party manufacturers. Using WESE, a developer can analyse different design alternatives involving components from various researchers and manufacturers. The environment provides a flexible framework to develop formal specifications and verify the designs using mechanized theorem proving. WESE also provides the techniques to generate different specifications documents such as cost of the system.

The primary input to WESE is a System Specification Language (SSL) description of the system. The input SSL source is parsed into an intermediate form (SSL-IF) using a parser. SSL-IF forms the input to other modules of WESE SLL-IF is used to construct simulations, generate specifications, and documentation from the distributed factories. The factories play a pivotal role in providing a uniform interface to generate the various components specified in the SSL specifications. The various subsystems of WESE provide necessary support to achieve the different functionalities of environment. The user may employ a hypertext markup language (HTML) or a text based interface to interact with the environment.

The rapid advances in Web technology, most notably Java, enable to execute highly interactive and dynamic simulations/animations on the Web. Java [44] is a portable language because it is compiled into an architecture neutral, byte code format. Hence, a Java application can run on any system, providing that system implements the Java Virtual Machine (JVM). It is this portability, which makes Java so attractive to Internet developers because it allows compiled Java code to be sited on one computer, but downloaded, interpreted and executed on another computer; which isn't necessarily running on the same platform without the need to recompile the software. This is of significant difference to more traditional simulation languages such as C++, where exporting simulation code requires recompilation and installation on each different machine. Although Java is a small core language it does support extensive class libraries. As a result the JVM need only be able to handle the core of the language whilst applications can be developed using the class libraries. These class libraries can be further refined and extended to develop application specific packages (e.g. simulation). All versions of Java come with a set of in-built class libraries which include facilities for developing graphical user interfaces (e.g. the java.awt and javax.swing.awt packages) [45], network application support (java.net package), a number of utility functions (java.util package), remote method invocation (java.rmi package) and 3D animation facility (Java 3D API) [46].

The facilities provided by the Java programming language to support Web technology is the power behind Web-based simulation. Java benefits from being simple, secure, object-oriented, multi-threaded and architecture neutral. More and more researchers develop their simulation packages based on Java. But most of this research is only devoted to discrete event simulation. *SimDynamic* is an attempt to introduce dynamic system simulation on the Web and provides a positive, unique contribution to Web based simulation.

Chapter Three

3 Overview of *SimDynamic*

3.1 What is *SimDynamic*

SimDynamic is a java based simulation package for modelling, simulating and analysing dynamic systems on the Web. A model of moderate complexity can be constructed with minimum of effort and without writing any code. It supports linear and non-linear systems, modelled in continuous time, sampled time or a hybrid of the two. Finally the package will provide both numerical, graphical results and also 3D animation.

SimDynamic is actually a collection of several functional nodes, where each node carries out a specific function such as integration, generation of sine wave, multiplication etc. These nodes serve as the basic building blocks of a model. These are classified into following eight node sets

- Continuous
- Discrete
- Tables
- Math & Logic
- Nonlinear
- Miscellaneous
- Sinks
- Source

The package has been designed with an aim to achieve as much modelling ease as possible and to provide maximum user flexibility in model execution. For modelling the package provides a graphical user interface (GUI) for building models as block diagrams, using simple mouse operations. A block diagram is a pictorial model of a dynamic system. It consists of a set of nodes, interconnected by lines. Each node produces an output either continuously (a continuous node) or at specific points in time (a discrete node). The lines represent connections of node inputs to node outputs.

A block diagram can contain any number of instances of any type of node needed to model a system. This interface will enable the user to draw a block diagram just as to draw with pencil and paper.

Section 3.2 contains a brief description of the GUI, section 3.3 focuses on the manipulation of nodes provided by *SimDynamic*, section 3.4 deals with lines, section 3.5 introduces the simulation parameters, section 3.6 gives a brief description of how to build a model using the package and section 3.7 presents how to make 3D animation.

3.2 Graphical User Interface (GUI)

When *SimDynamic* is started it opens the node set window (figure 3.1). This window is divided into three sections. First section contains the name of the currently selected node set, second section shows all the eight node sets provided by *SimDynamic*, and the final section displays the nodes in the currently selected node set.

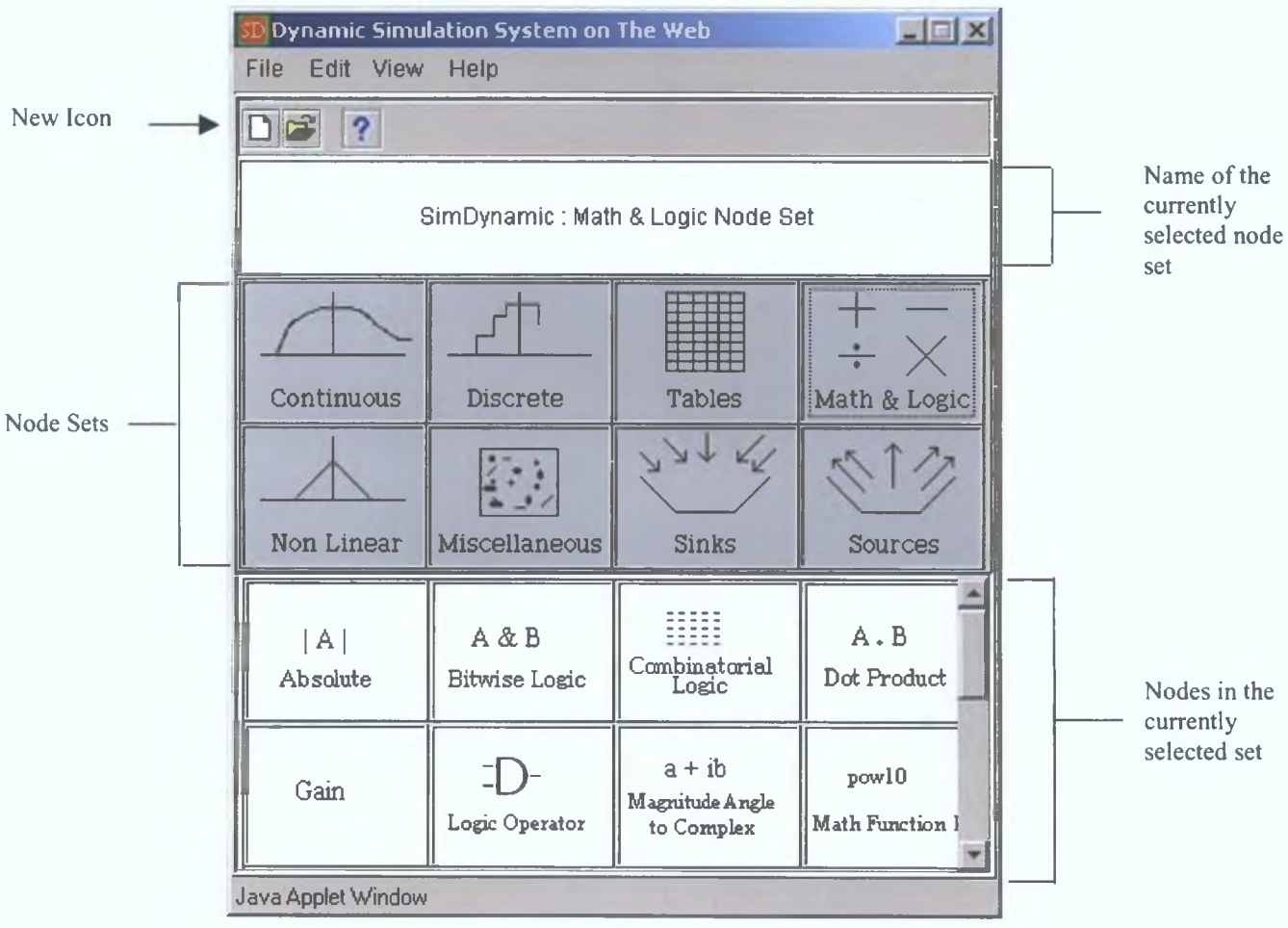


Fig 3.1: Node set window

To start a new model click on the new icon in the toolbar, or select new from the file menu. *SimDynamic* will open the following model sheet window (figure 3.2). A new model will be drawn in the draw area of this blank sheet. Toolbar of this window provides short cut buttons for various menu items. User can use these buttons on the toolbar at his/her convenience. Status-bar at the bottom of the window displays the current state of the simulation model. "Ready " on the status-bar means that *SimDynamic* is idle at the moment and ready for a new simulation run. "Initialising" on the status-bar indicates that *SimDynamic* is initialising the model and "Running" indicates that the model is being executed.

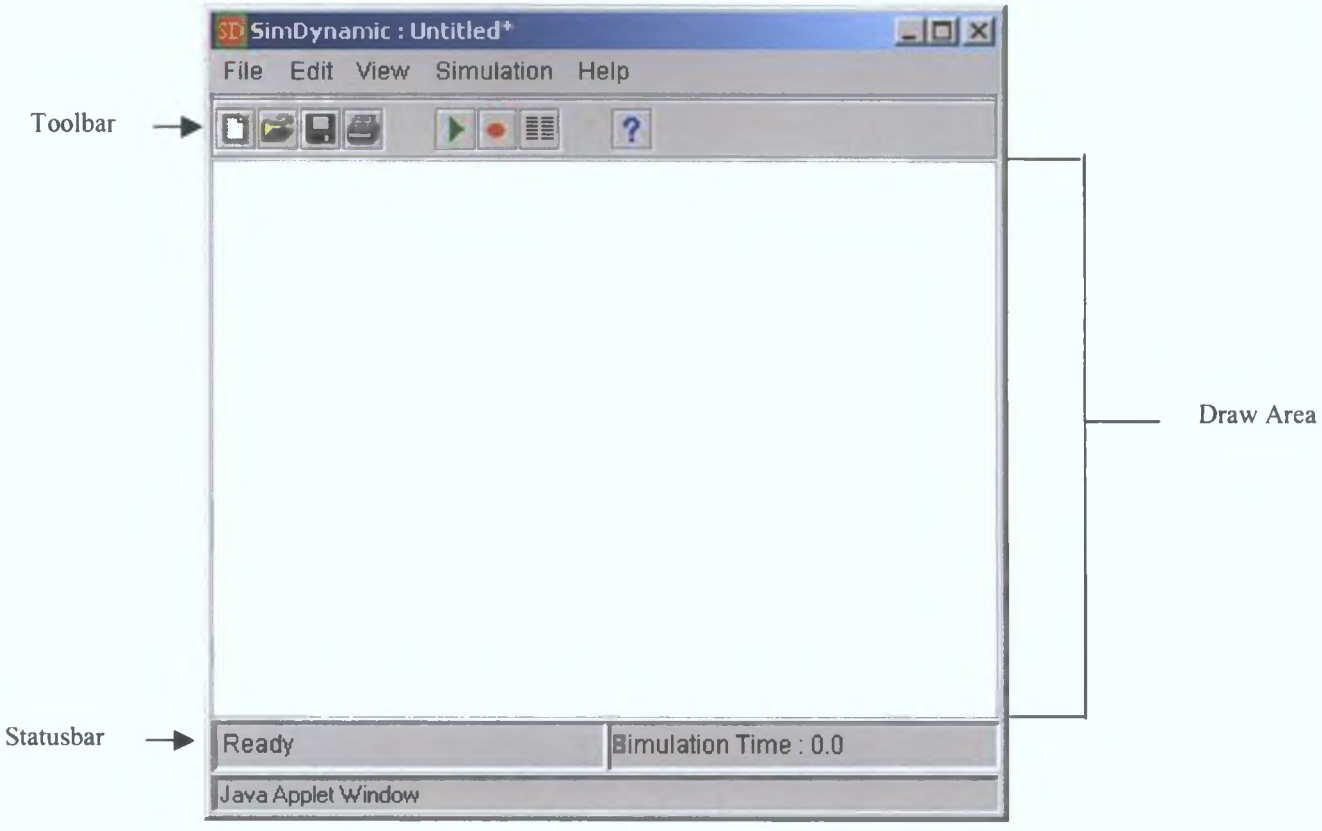


Fig 3.2: Model sheet window

The interface incorporates on-line help for user guidance and assistance. The help provides all the information needed to get along with *SimDynamic*, such as how to build a model, simulation parameters, functional description of nodes etc. Figure 3.3 provides a screenshot of *SimDynamic* on-line help.

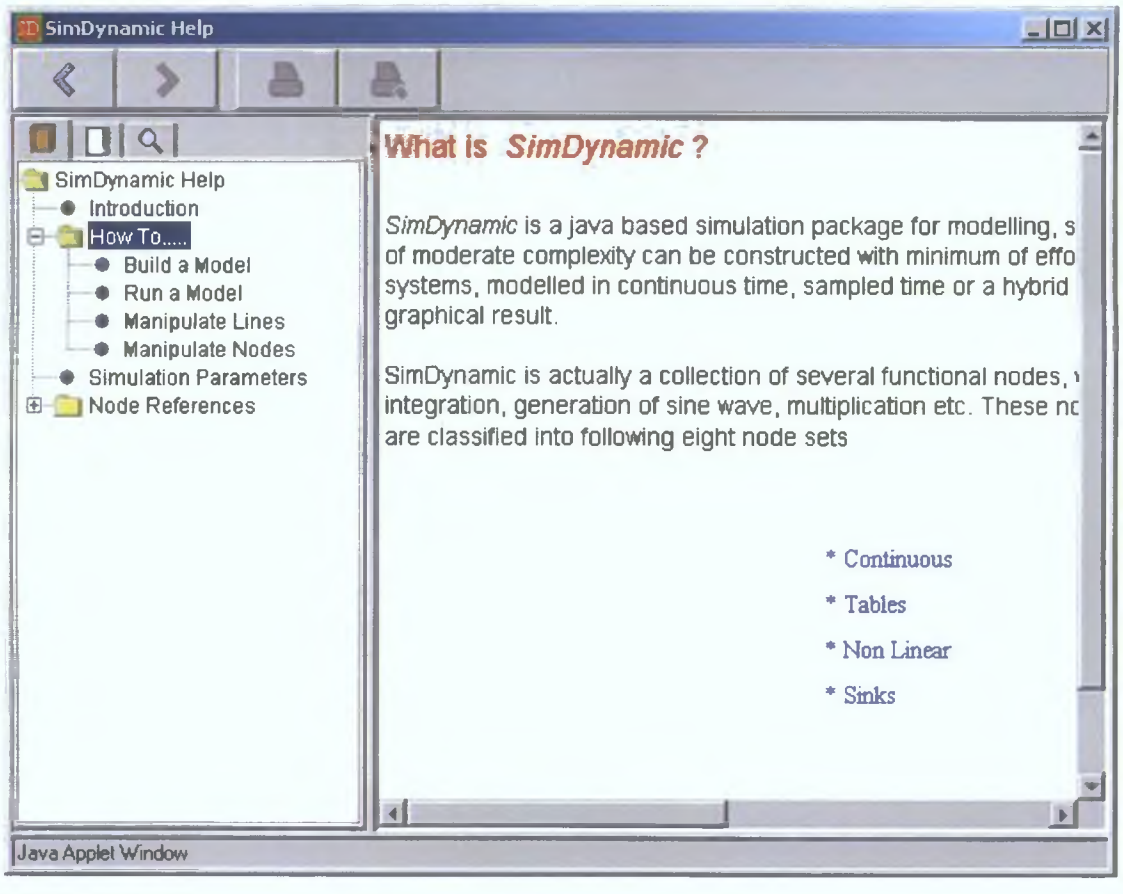


Fig. 3.3: SimDynamic on-line help

3.3 Nodes

Nodes are the basic building block of model. Each node performs a specific task at each simulation time step. A node comprises one or more of the following: a set of inputs, a set of states, and a set of outputs.

Fig 3.4: Logical representation of a node

A node's output is a function of time and the node's inputs and states (if any). The specific function that relates a node's output to its inputs, states, and time depends on the type of node of which the node is an instance.

To have functional description of each node see chapter 5. This section describes the nodes from GUI perspective.

3.3.1 Adding a Node

To add a node in the model sheet, click on the node in the node set window. The mouse cursor will be turned into a hand cursor indicating that the node has been selected. Now click on the draw area of model sheet window. *SimDynamic* will add the selected node in the model sheet. Each node will have one or more input/output port.

3.3.2 Manipulating Nodes

To move a node within the model, place the mouse cursor on it and press the left mouse button. Mouse cursor will turn into a crossed cursor. Now drag the mouse and the node will be moving with the mouse movement. If any line is connected to the node it will be adjusted automatically.

A node can be deleted, rotated or renamed by using node menu. To activate the node menu, position the mouse cursor on the node and click the right mouse button. The node menu will appear like figure 3.5. Now choose appropriate menu item to manipulate the node.

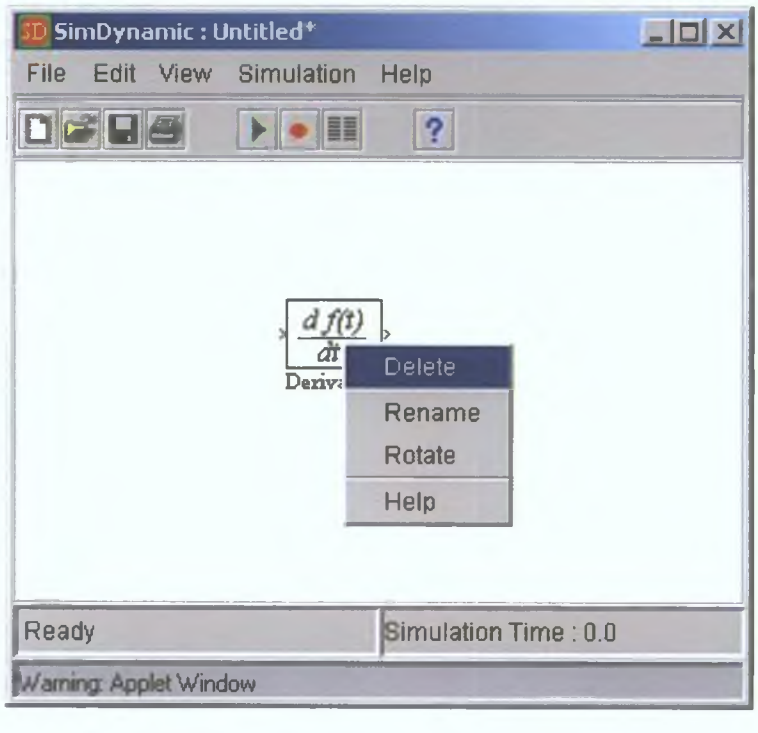


Fig 3.5: Node menu

To delete the node select delete from the menu, the node will be deleted and the connected lines (if any) to the node will be adjusted.

To rename the node choose r*ename* from the menu, *SimDynamic* will open a dilaog box, enter the new name of the node and click ok on the dialog box. The new name will appear at the bottom of the node.
To rotate the node choose *rotate* from the menu, the node will be rotated by 180 degree.

3.3.3 Signals and Data Types

Signals are the streams of values that appear at the outputs of a node when a model is simulated. It is useful to think of signals as travelling along the lines that connect the nodes in a block diagram. But it should be kept in mind that the lines in a model represent logical, not physical, connections among nodes.

The term *data type* refers to the way in which a computer represents numbers in memory. Different machines (with different architectures) allocate different amount of storage and uses special internal codes to keep track of the different types of data it processes. On the other hand, a data type in a programming language is a set of data with values having predefined characteristics. Examples of data types are: integer, floating point, character, string, and pointer. It is a collection of values and operations. Operations do things to values and produce values as a result.

SimDynamic supports two types of signal i) Numerical and ii) Boolean. The primitive data type used by *SimDynamic* for numerical values is Java *double*, which is a 32 bit signed number. So it ranges from $1.7e^{-308}$ to $1.7e^{308}$. *SimDynamic* supports both real and complex numbers, and for logical operations it internally converts the *double* value to *integer*.

Boolean value is used for some logical operations. There are some nodes that support both types of data. It is up to the user to decide whether Boolean value or numerical value is to be used.

3.3.4 Node Parameters

Nodes are added to the models with their default parameters. Each node has its own parameter dialog box to set parameters. To access a node's parameter double click on

it, parameter dialog box of the corresponding node will be opened. Now the user can change parameter values. After changing the parameter values click the ok button on the dialog box. A click on the cancel button will ignore any change in the parameters and will retain the previous parameter values. Parameter Dialog Box for Discrete State Space node is shown here for example.

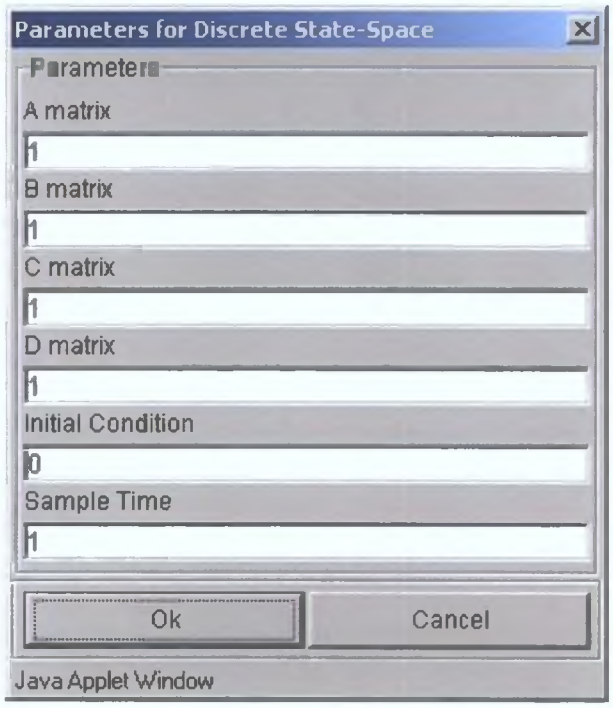


Fig 3.6: Parameter dialog box for Discrete State Space

Most nodes accept vector (1D array) or matrix (2D array) parameter/input. There are some nodes that accept only single number. Vectors may be of two types i) row vectors (2D array contains only one row), ii) column vector (2D array contains only a single column). To enter a row vector as parameter write the numbers separated by spaces. Following dialog box shows, how to enter the row vector 2 3 4 as Gain parameter,

Fig 3.7: Row vector as parameter

To enter column vector as parameter write the numbers separated by commas. Figure 3.8 shows how to input a column vector as gain parameter

3

6

9

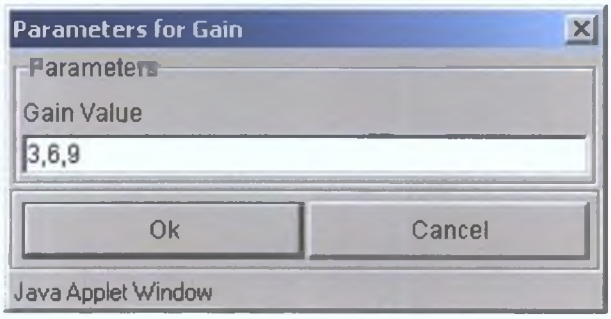


Fig 3.8 : Column vector as parameter

To enter a matrix as parameter enter the rows one by one separated by commas. Following dialog box shows, how to enter the following matrix as Gain parameter

2 3 4
5 6 7
1 3 5

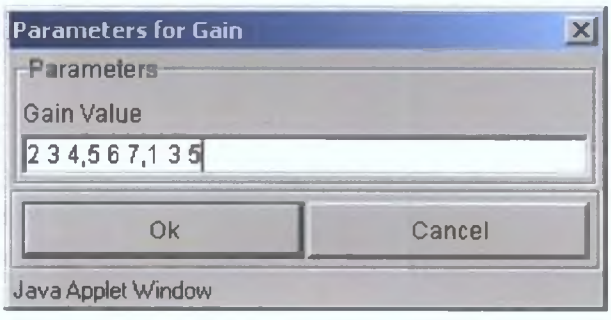


Fig 3.9 : Matrix parameter

Most of the nodes support auto expansion of inputs/parameters wherever applicable. For example 2 3 4 has been entered as gain parameter and the node receives 5 as input. Then Gain node will consider the input as 5 5 5 and output will be 10 15 20. For matrix parameter the node will try to expand the input to a matrix of same dimension as the parameter. For example above matrix has been entered as gain parameter and the node receives 7 as input. Then gain node will expand the input to the following matrix,

$$\begin{matrix} 7 & 7 & 7 \\ 7 & 7 & 7 \\ 7 & 7 & 7 \end{matrix}$$

And the corresponding output from the node will be the following matrix,

$$\begin{matrix} 14 & 21 & 28 \\ 35 & 42 & 49 \\ 7 & 21 & 35 \end{matrix}$$

When using matrix input or parameter, it should be made sure that all the rows of the matrix must have same number of columns.

3.3.5 Direct Feedthrough

There are some nodes whose current outputs depend on their current inputs. These nodes are defined as direct feedthrough nodes. All other nodes are called non-direct feedthrough nodes. Examples of direct feedthrough nodes are Sum, Gain etc. Examples of non-direct feedthrough nodes are Integrator, Memory etc. When using direct feedthrough nodes it should be made sure that the model does not contain any algebraic loop. An algebraic loop generally occurs when an input port with direct feedthrough is driven by the output of the same node, either directly, or by a feedback path through other nodes with direct feedthrough.

3.3.6 Sample Time

The Sample time parameter sets the time interval at which node's states (output) are updated. There are some nodes that have continuous (all the nodes in continuous node set and some other) sample time, some nodes have discrete sample time (all the nodes in discrete node set and some other). There are some nodes, which derive its sample time from its driving node (the node from which it receives its input), these nodes are said to have inherited sample time. When a node has inherited sample time and it receives its input from more than one node, then it sets its sample time to the sample time of the driving node that has the fastest sample time. When setting sample time of

a node it should be ensured that the sample time of each node in a model should be a multiple of step size.

3.4 Lines

3.4.1 Draw Lines

Place the mouse cursor near the input/output port of a node. It is not necessary to position the cursor precisely on the port. Cursor will be turned into a cross hair cursor (figure 3.10).

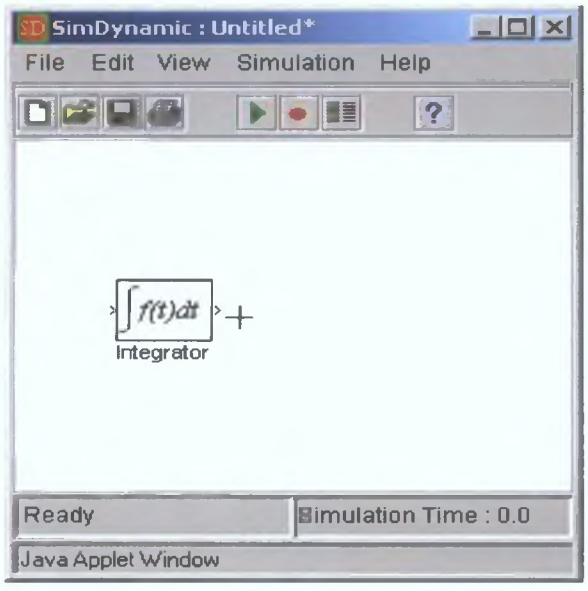


Fig 3.10: Line drawing step 1

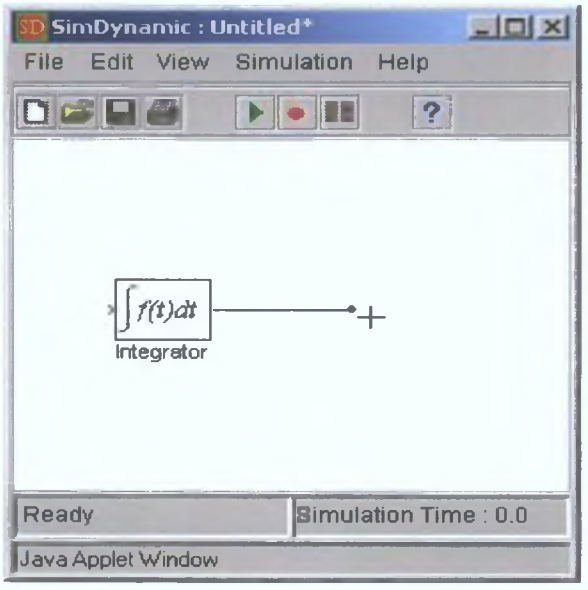


Fig 3.11: Line drawing step 2

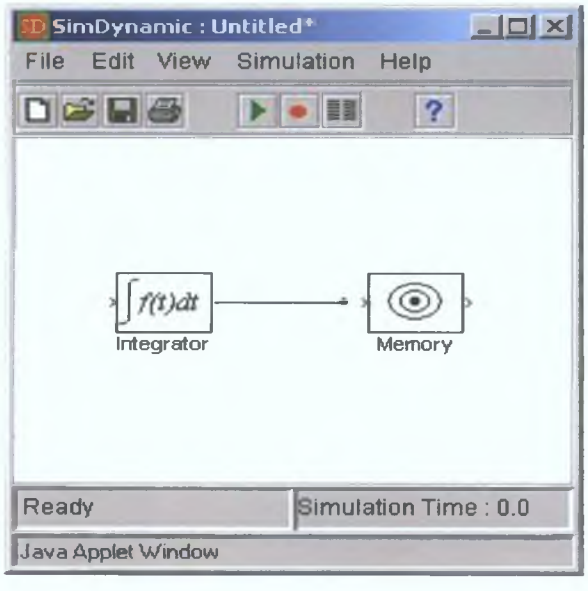


Fig 3.12: Line drawing step 3

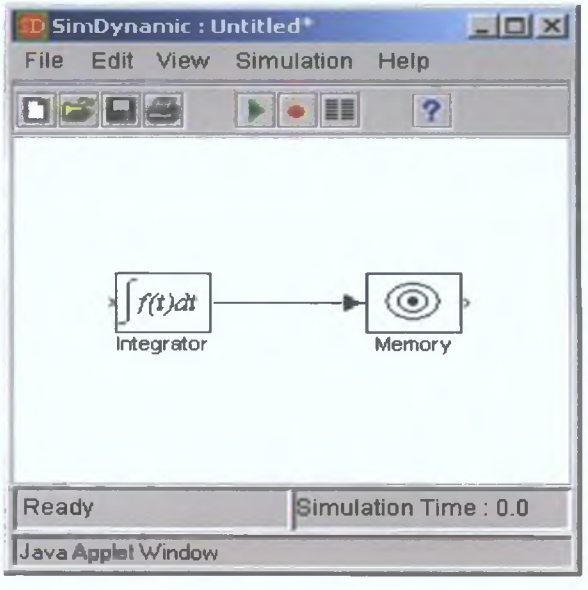


Fig 3.13: Line drawing step 4

Press the left mouse button and drag the mouse to draw line (figure 3.11). To finish the line, extend it near the output/input port of another node and release the mouse (figure 3.12). *SimDynamic* will complete the line by connecting it with the targeted node (figure 3.13).

3.4.2 Manipulating Lines

A line can be deleted, its colour can be changed or a branch line can be drawn from an existing line by using line menu. To activate the line menu, position the mouse tip on the line and click right mouse button. Line menu will appear (figure 3.14). Now choose appropriate menu item to manipulate line.

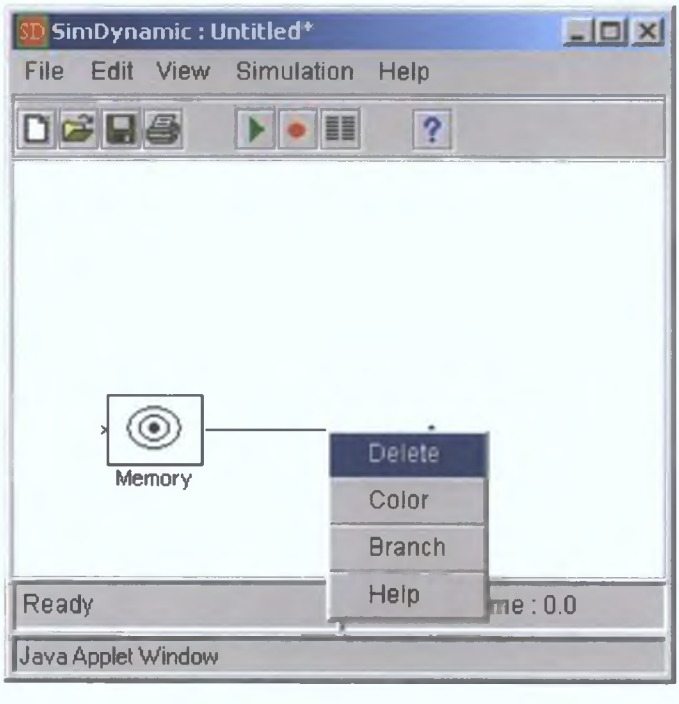


Fig 3.14: Line menu

To delete the line, choose delete from the menu. *SimDynamic* will delete the line and will reset the input/output port of the node to which the line was connected.

To change the colour of the line, select color from the menu. *SimDynamic* will open a colour chooser, select a new colour and click ok button. The line will be redrawn in the new colour.

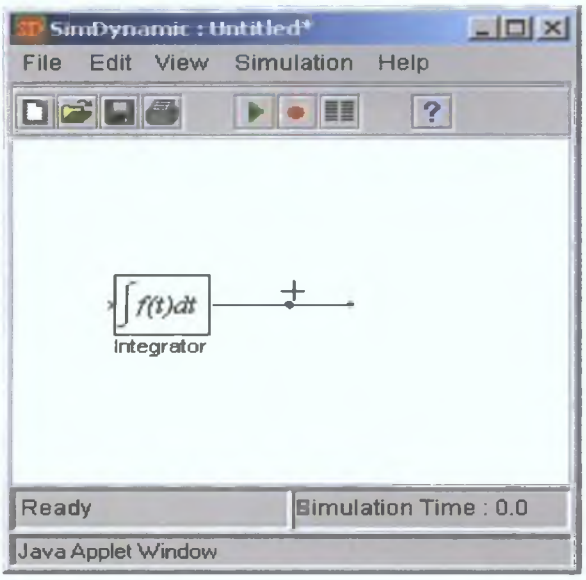


Fig 3.15: Branch line drawing step 1

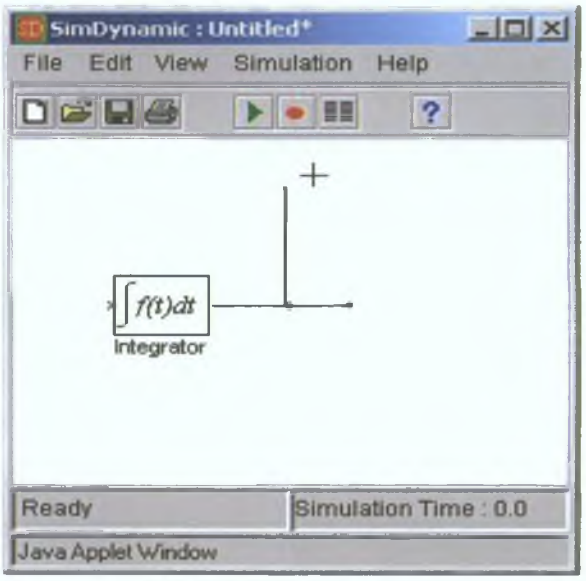


Fig 3.16: Branch line drawing step 2

To draw a branch line, select branch from the line menu. A point will appear on the line. Position the mouse cursor near the point, mouse cursor will be turned into a cross hair cursor (figure 3.15). Now press the left mouse button and drag the mouse to draw branch line (figure 3.16). To finish the branch line, follow the same procedures as with normal lines.

3.5 Simulation Parameters

Before starting a simulation run user has to set simulation parameters. To access the simulation parameters select *parameter* menu item from *Simulation* menu. Following parameter dialog box will appear.

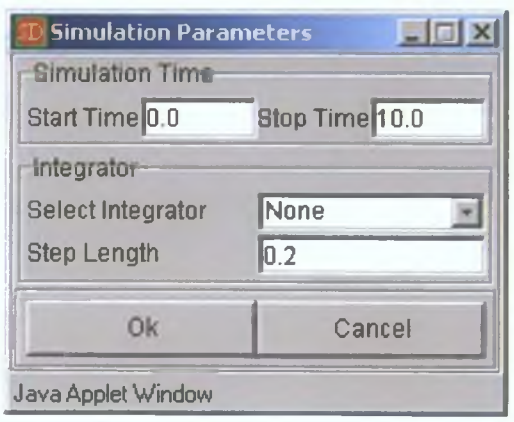


Fig 3.17: Simulation parameters dialog box

3.5.1 Simulation time

Simulation time specifies how long the simulation will continue. This time is measured in seconds. And of course it is in pseudo time. Starting time need not be always 0. Simulation may start from any starting time. The only requirement is that stop time must be greater than the start time.

It should be noted that, simulation time and physical time are not the same. For example, running a simulation for 10 seconds will usually not take 10 seconds. The amount of time it takes to run a simulation depends on many factors, including the model's complexity, the simulation step sizes, and the computer's clock speed, available memory etc.

3.5.2 Integrator

The dynamic models are generally described by differential equations. The solutions of these equations display the operations of the system and this is the fundamental property of dynamic models. This software provides a number of integrators to solve ordinary differential equations (ODE). Because of the diversity of dynamic system behavior, some integrators may be more efficient than others at solving a particular problem. Here is a brief description of each solver that has been implemented. Greater details about integrators can be found in chapter 4.

- RK_DP: It is based on an explicit Runge-Kutta formula, the Dormand-Prince pair.
- RK_O4: It is based on fourth order explicit Runge-Kutta formula.
- RK_BS: It uses an explicit Runge-Kutta pair of Bogacki and Shampine. It may be more efficient than RK_DP in the presence of mild stiffnes.
- RK_HE: It uses Heun's method which also known as modified Euler formula.
- RK_EU: It uses simple Euler's method for integration.

3.5.3 Step Length

SimDynamic starts at the starting time and advances its clock by adding step length to the current clock time until it reaches the stop time. At every time step it calculates the output of each node in the model. Default time step is 0.2. So for a start time 0 and

stop time 10.0, outputs will be calculated at 0, 0.2, 0.4.....10.0. Smaller time steps may ensure more accurate results but at the same time it may increase simulation run time depending on the model complexity and the integrator used.

Step length and sample time of the nodes in a model must be compatible. When setting step length or sample time of nodes, it should be made sure that sample time of all the nodes in a model must be an integer multiple of the step length.

3.6 Run a Model and View Output

Simulating a dynamic system with this package is a two-step process. First, the user has to create a model of the system to be simulated. The model graphically depicts the time-dependent mathematical relationships among the inputs, states and outputs of the system. After the model creation user has to choose an integration method and then can use SimDynamic to simulate the behaviour of the system for a specified period of time.

To start a simulation run choose *run* from the *simulation* menu or click on the run button from the toolbar. First *SimDyanmic* will initialise the model and then it will execute the model. If the model does contain error, simulation will be halted with appropriate error message.

To view result of a simulation run, use the nodes (except terminator node) from *Sink* node set. To view numerical result use *Display* node, for graphical results *Time vs input* or *X vs Y Graph* nodes can be used. To save simulation results in a text file use *To file* node. When simulation run is finished simply double click on any of the sink nodes to view results. *To Dat File* node from the sink node set can be used to save simulation results in a data file for later use by *SimDynamic*.

3.7 3D Animation

To start the 3D model editor, choose *3D animation* from the simulation menu. *SimDynamic* will open the following 3D model editor.

Fig 3.18: 3D model editor window

This editor provides some basic 3D shapes (Box, cone etc.) on the left side. It is assumed that the 3D model will be rendered in a 3D space. At each simulation time step objects of a 3D model will move to a new position in the space. User has to add 3D shapes to the space to construct a model. To add any shape, click on the appropriate button. Each button click will open a dialog box for the selected shape. Here parameter dialog box for a box is shown for example,

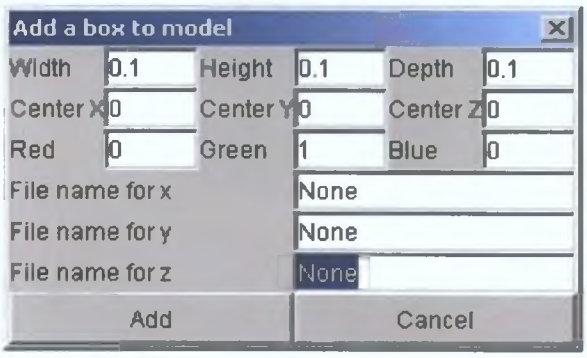


Fig 3.19: Dialog box for 3D box

On the dialog box user has to define the initial position (center X, Center Y, Center Z), colour (in term of RGB value) and dimension of the shape. All the dimensions in the 3D model editor are measured in meter. More importantly, to animate any 3D object at least one data (simulation result from *SimDynamic*) file should be associated with it. 3D animator will read current position of the object from the associated data file(s) and render the 3D animation. Parameter button enables the user to set some 3D parameters. User can set the origin of the 3D space and can also set the time delay during 3D rendering. *Save* button can be used to save a 3D model. Using the *load* an existing 3D model can be opened. To start 3D animation, click the *run* button and a new 3D window will be opened.

Chapter Four

4 The Design of *SimDynamic*

SimDynamic, is a collection of Java classes (objects) which provides basic utilities for constructing simulation models for dynamic systems. In the current implementation, simulation, random number generation, and analysis of simulation result (both numerical and graphical) are built into the library. In the following sections, the design of the *Simdynamic* package will be presented in great detail.

SimDynamic is intended to make model creation an easy task. It is also intended that the model user would be able to understand the model as well as use the simulation easily. This is achieved through providing a user-friendly modelling interface and an easy to view (graphical/numerical) results mechanism. It is believed that *SimDynamic* is an easy to use package and that even a fairly complex simulation model can be constructed without writing any additional code.

The software consists of three packages, each package containing a set of related classes that work together to provide functionality in a key area to support a simulation process. The package hierarchy of this software is shown in figure 4.1.

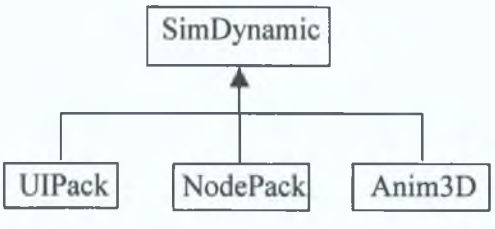


Fig 4.1: SimDynamic package hierarchy

A discussion of each of these packages is presented in this chapter in greater detail. In the class hierarchy diagrams in this chapter foundation classes (java built-in classes) are represented by dashed rectangle and software classes are shown by regular rectangles. The signatures for the public methods of major classes for each package will be given. For brevity, all the classes of each package, and all the methods of classes, are not discussed here. Complete source code of *SimDynamic* is provided in appendix E.

4.1 UIPack

This package contains all the necessary classes to provide the graphical user interface (GUI) of the software. The class hierarchy of this package is as follows

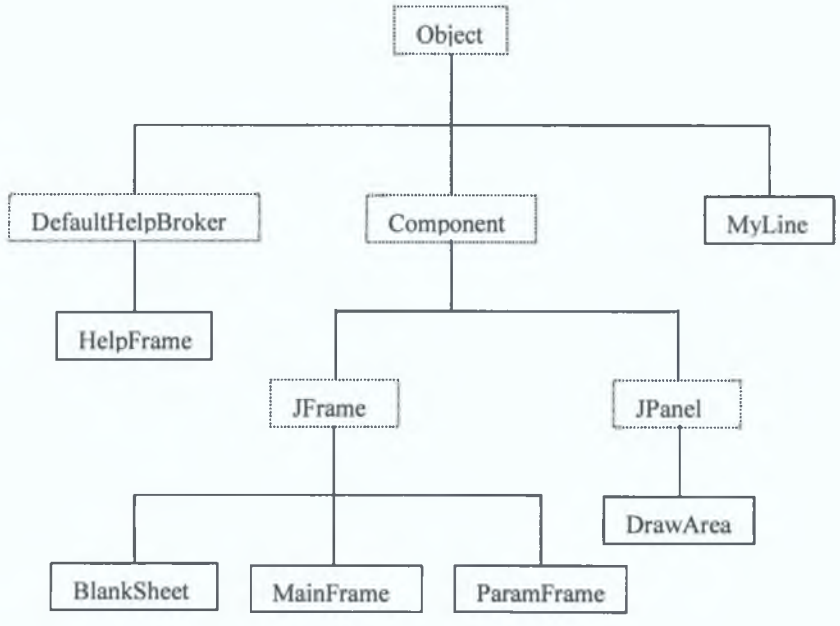


Fig 4.2 : Class hierarchy in UIPack package

4.1.1 MyLine

The MyLine class is responsible for handling all the line drawings in a model, and its definition looks as follows

```
public class MyLine extends Object{
      public Vector points;
      private int lx, ly;

      public MyLine(Container c, Simulator s);
      public void drawL(Point p, int d);
      public void drawC(int w, Color clr);
      public void setArrowPoints(boolean rotated);
      public void drawA();
      public void adjustA(int tx, int ty, boolean rotated);
      public void adjustR(int tx, int ty, boolean rotated);
      public boolean  checkPoint(Point p, Point pt);
}
```

This implementation of MyLine allows only vertical and horizontal lines. A complete line may contain indefinite number of vertical and horizontal segments. MyLine stores end points of every segment in the Vector *points,* lx and ly stores the last drawing point of the line. Constructor needs a *Container* on which the line will be drawn and a *Simulator* which will store the line.

Method *drawL* draws a line segment from (lx, ly) to the current point *p*. This method also determines the direction (vertical or horizontal) of the line.

The drawC method draws a circle at the last point of an incomplete line.

The *setArrowPoints* method calculates the co-ordinates of an arrow for a completed line.

The *drawA* method draws an arrow at the last point of a completed line.

Method *adjustA* and *adjustR* are used to adjust a line connected to a node, when the user moves the node.

4.1.2 HelpFrame

HelpFrame provides the framework for displaying *SimDynamic* online help.

```
HelpFrame extends DefaultHelpBroker{
      public HelpFrame(HelpSet hs);
}
```

HelpFrame simply extends java's DefaultHelpBroker, where the HelpBroker is the default presentation of a HelpSet. And a HelpSet is a collection of help information consisting of a HelpSet file, table of contents (TOC), index, topic files, and Map file. The constructor also customizes the default presentation by enabling and disabling some buttons.

4.1.3 DrawArea

Class *DrawArea* provides the free area on the model sheet window, where user can draw a model. Its signature is as follows,

```
class DrawArea extends JPanel implements ActionListener{
      public DrawArea(Simulator smltr);
      public void painComponent(Graphics g);
      public void actionPerformed(ActionEvent ae);
      public NodeIcon createNode();
```

```
}
```

Constructor *DrawArea* initialises the drawing area and it also implements the *ActionListener* interface to perform the menu actions for a line. This class also includes two inner classes to capture all the user mouse operations.

The *paintComponent* method repaints the draw area when ever it is needed.

The *actionPerformed* method handles all menu actions for the line menu.

The *createNode* method creates a new node, when ever user clicks on the draw area after selecting a node from the node set browser, and returns the node.

4.1.4 BlankSheet

Class *BlankSheet* holds the *DrawArea* and provides the framework for simulation model sheet window. It looks like as follows,

```
public class BlankSheet extends JFrame implements ActionListener{
      public BlankSheet();
      public void processComponent(ComponentEvent ce);
      public void actionPerformed(ActionEvent ae);
}
```

Constructor *BlankSheet* performs all the necessary steps to set the UI. It instantiates a new *DrawArea* and add it to the frame, also instantiates menu and menu items, toolbar, and status bar.

The *processComponent* method is used to store the current size and location of the frame, whenever it is resized or moved by the user.

The *actionPerformed* method handles all the menu actions and toolbar icon actions.

Other than these methods, this class has two important private methods *saveFile()* and *openFile()*. These methods are used to save a model or to open an existing model.

4.1.5 MainFrame

MainFrame class is responsible for the node set browser window, which is the starting window of *SimDynamic*.

```
public class MainFrame extends JFrame implements ActionListener{
      public MainFrame();
```

```
    public void actionPerformed(ActionEvent ae);
}
```

Constructor *MainFrame*, instantiates all the components necessary for the user interface (UI).

The *actionPerformed* method is responsible for all menu actions and toolbar icon actions. This method also creates a new *BlankSheet* whenever user clicks the new menu item.

4.1.6 ParamFrame

Class *ParamFrame* enables the user to set simulation parameters. Its signature is as follows,

```
public class ParamFrame extends JFrame implements ActionListener{
    public ParamFrame(Simulator smltr);
    public void initParams();
    public void actionPerformed(ActionEvent ae);
}
```

Constructor *ParamFrame* instantiate all the components needed for the input such as TextField, ComboBox etc.

The *initParams* method assign the current values of different simulation parameters to the respective input fields before displaying the frame.

The *actionPerform* method is invoked when user clicks any of the buttons (ok/cancel) in the frame. This method parses the input entered by user and checks legitimacy of input. Invalid input will be reported by appropriate error message.

4.1.7 Other Classes in the Package

UIPack also contains following classes. All of these classes (except *CNST* class) extends *JFrame*.

CNST – this class holds some constant values that are used in various places of the software. All members of this class have been declared static. So they are directly accessible by using the class name only.

ColorFrame – *ColorFrame* provides a colour chooser, using that user can change the colour of a line.

ErrorFrame – *SimDynamic* uses *ErrorFrame* to display the appropriate error message, whenever it encounters some error during the runtime of a simulation.

InfoFrame – this class displays the information about *SimDynamic*.

DispFrame – *DispFrame* provides the framework to display numerical results of a simulation. The class *SnkDspl* from NodePack package supplies the numerical results to this class after a simulation run is completed.

TvsXFrame – this class is responsible for graphical output of a simulation. It plots the graph of a simulation result with respect to time. The class *SnkTvsinp* from NodePack package provides necessary data to this class.

XvsYFrame – works in the same way as *TvsXFrame*, but it derives its data from the class *SnkXYGrph* of NodePack package.

4.2 NodePack

NodePack is the core of this software and provides all the functional nodes. This package also contains the main simulation engine (the *Simulator* class) and some other important classes.

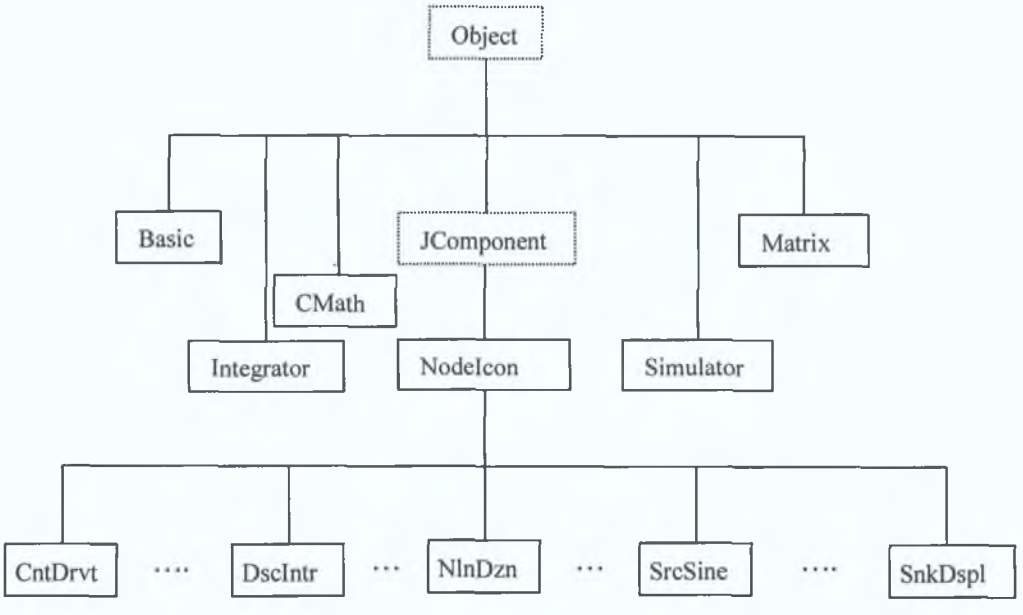


Fig 4.3 : Class hierarchy in NodePack package

4.2.1 Basic

Basic class can be considered as the utility class for *SimDynamic*. This class contains various utility methods to ease the task of other classes.

```
Basic extends Object{
      public static boolean equalTo(double d1, double d2);
      public static boolean grTeqt(double d1, double d2);
      public static boolean lsTeqt(double d1, double d2);
      public static boolean lessTn(double d1, double d2);
      public static boolean greatTn(double d1, double d2);
      public static boolean checkRegion(int x, int y, int cx,
             int cy,  int r);
      public static void VecToArray(double[][] R,double[][] C,
             Vector V);
      public static void VecToArray(double[][] M,Vector V);
      public static void VecToArray(double[] R,Vector V);
      public static byte setOpDataType(Vector opMat);
      public static void dblVect2cmplx(Vector cmplxV, Vector dblV);
      public static void assignVect(Vector V1, Vector V2);
      public static void assignMat(Vector Mat1, Vector Mat2);
      public static boolean isNegative(Vector Mat);
      public static boolean hasGreater(Vector Mat1, Vector Mat2);
      public static Vector matMultByVec(Vector Mat1, Vector Mat2);
      public static void getDim(Vector Mat, MyDim d);
      public static boolean between_include_lower(double n,
             double h, double l);
      public static boolean between_include_higher(double n,
             double h, double l);
      public static boolean between_include_both(double n,
             double h, double l);
      public static Double elementAt(Vector Mat, int row, int col);
      public static Boolean elementAt(Vector Mat, int row,
             int col, byte b);
      public static boolean elementAt(Vector Mat, int row,
             int col, Complex cmplx);
      public static void initMat(Vector Mat, double dbl);
      public static void initMat(Vector Mat, double rel, double img);
      public  static  boolean  canBeResolved(MyDim  dim1,  MyDim  dim2,
      boolean[] expand, MyDim fdim);
      public static boolean canBeResolved(MyDim dim1, MyDim dim2,
```

```
        MyDim dim3, boolean[] expand, MyDim fdim);
    public static boolean canBeResolved(MyDim dim1, MyDim dim2,
        MyDim dim3, MyDim dim4, boolean[] expand, MyDim fdim);
    public static boolean updateParam(String str,Vector mV,
        byte type, String[] err_msg);
    public static void tf2ssAB(Vector A, Vector B, Vector num,
        Vector den, int n);
    public static void tf2ssCD(Vector C, Vector D, Vector num,
        Vector den, int n);
    public static void factToPoly(Vector Mat, Vector gV,
        boolean g);
}
```

All the members in this class have been declared as static. So other classes can access these methods (or variables) without instantiating an object of type Basic. First five methods of this class (*equalTo()*.....*greatTn()*) performs normal relational operations (>, <, >=, = = etc.) on floating point numbers. These methods are written in addition to Java's normal relational operators to attain desired precision in floating point calcualtion.

The *checkRegion()* method returns true if the point (x,y) is within the circle, centred at (cx,cy) with radius r.

The *vecToArray(double[][] R,double[][] C,Vector V)* converts a 2D vector (matrix) of Complex objects to two 2d array containing the real and imaginary parts respectively.

The *vecToArray(double[][] M,Vector V)* method converts a 2D vector (matrix) of Double objects to a 2d array of double.

The *vecToArray(double[] R,Vector V)* converts a 1D vector of Double objects to a 1d array of double.

The *setOpDataType(Vector opMat)* checks each element of the vector opMat. If it has a Complex number then it returns data type complex otherwise it returns data type real.

The *dblVect2cmplx(Vector cmplxV, Vector dblV)* converts a vector of Double objects into a vector of Complex objects, where each complex number's imaginary part is zero.

The *assignVect(Vector V1, Vector V2)* assigns the elements of vector V2 to vector V1.

The *assignMat(Vector Mat1, Vector Mat2)* assigns elements of vector (2D) Mat2 to vector Mat1.

The *isNegative(Vector Mat)* returns true if any element of the vector Mat is negative.

The *hasGreater(Vector Mat1, Vector Mat2)* returns true if any element of vector Mat1 is greater than the corresponding element in vector Mat2.

The *matMultByVec(Vector Mat1, Vector Mat2)* multiply a matrix (Mat1) by a row matrix (Mat2) and return the resultant row matrix.

The *getDim(Vector Mat, MyDim d)* finds the dimension of a matrix (Mat) and assign the dimension to d.

The *between_include_lower*, *between_include_higher*, *between_include_both* these three methods were used to find whether a number is within a given range.

The *elementAt(Vector Mat, int row, int col)* returns a Double object from the matrix (Mat) at (r,c) position.

The *initMat(Vector Mat, double dbl)* and *initMat(Vector Mat, double rel, double img)* were used to initialise a matrix with real number and complex number respectively.

The *updateParam(String str,Vector mV,byte type, String[] err_msg)* parses the string str for numerical parameters and assign the values to the vector mV. If this method encounters error during parsing it returns false and assign appropriate error message to the string err_msg.

The *tf2ssAB(Vector A, Vector B, Vector num, Vector den, int n)* partially converts a transfer function to a state-space. This method calculates the matrices A and B of a state-space from the numerator and denominator of its corresponding transfer function.

The *tf2ssCD(Vector C, Vector D, Vector num, Vector den, int n)* calculates the matrices C and D of a state-space from the numerator and denominator of its corresponding transfer function.

The *factToPoly(Vector Mat, Vector gV, boolean g)* finds the polynomial from its factorial form.

4.2.2 CMath

This class can be considered as a replica of Java's Math class for complex variable. SimDynamic supports both real and complex variables. This class provides all the

complex mathematics. All methods of this class have been defined as static as follows,

```
public class CMath extends Object{
      public static Complex add(Complex C1, Complex C2);
      public static Complex sub(Complex C1, Complex C2);
      public static Complex mult(Complex C1, Complex C2);
      public static Complex div(Complex C1, Complex C2);
      public static Complex pow(Complex C1, Complex C2);
      public static Complex pow10(Complex C1);
      public static Complex exp(Complex C1);
      public static Complex ln(Complex C1);
      public static Complex log10(Complex C1);
      public static Complex log2(Complex C1);
      public static Complex sqr(Complex C1);
      public static Complex sqrt(Complex C1);
      public static Complex conjugate(Complex C1);
      public static Complex inverse(Complex C1);
      public static Complex reciprocal(Complex C1);
      public static double magnitude2(Complex C1);
      public static double abs(Complex C1);
      public static double angle(Complex C1);
      public static Complex sin(Complex C1);
      public static Complex cos(Complex C1);
      public static Complex tan(Complex C1);
      public static Complex asin(Complex C1);
      public static Complex acos(Complex C1);
      public static Complex atan(Complex C1);
      public static Complex sinh(Complex C1);
      public static Complex cosh(Complex C1);
      public static Complex tanh(Complex C1);
      public static Complex asinh(Complex C1);
      public static Complex acosh(Complex C1);
      public static Complex atanh(Complex C1);
      public static Complex ma2cmplx(double d1, double d2);
}
```

All the methods in this class have self-explanatory names. So functional description of the methods has been skipped here.

4.2.3 Matrix

Matrix class contains all the methods required for matrix operations. Its definition is as follows,

```
public class Matrix extends Object{
      public static Vector inverse(Vector mat);
      public static Complex DMinor(Vector a,boolean[] crossed,
             int row,int col);
      public static Vector hermitian(Vector mat);
      public static Vector transpose(Vector mat);
      public static Vector conjugate(Vector mat);
      public static Vector mult(Vector Mat1, Vector Mat2);
      public static Vector add_sub_div(Vector Mat1, Vector Mat2,
             int function);
      public static Complex trace(Vector mat);
}
```

Once again the methods in this class have self-explanatory names. So functional description has been skipped.

4.2.4 NodeIcon

The *NodeIcon* class has been defined as an abstract base class, that encapsulates the features common to the classes that appear as nodes in a model. All the functional nodes (total 83 nodes in this implementation) have been derived from this class. This class looks like as follows,

```
public abstract class NodeIcon extends JComponent implements
ActionListener{
      public NodeIcon(Simulator smltr);
      public void setIpOpX();
      public void adjustName();
      public void alert();
      public void adjust_ipY();
      public void actionPerformed(ActionEvent ae);
      public void reloadImage();
      public void paintComponent(Graphics g);
      public void processMouseEvent(MouseEvent me);
      public void processMouseMotionEvent(MouseEvent me);
```

```
    public boolean checkSampleT(double time);
    public void initDialog(String title, int w, int h);
    public void showDialog();
    public boolean resolveDimension();
    public boolean isOpOK(double simT);
    public void upDateOutput(double simT);
    public void initialize();
    abstract public void makeDialog();
    abstract public void cancel();
    abstract public boolean checkConsistency();
    abstract public boolean action(double simT);
}
```

Constructor *NodeIcon* initialises all common attributes (sample time, number of ports etc.) of a node. It loads the default image and default name for the node.

The method *setIpOpX()* sets the X co-ordinates for the input and output ports of the node.

The method *adjustName()* adjusts the name of the node when user changes the node's default name.

The method *alert()* draws a red rectangle around the node when simulation is halted because of a run time error. It helps the user to identify the node in which error has been occurred.

The method *adjust_ipY()* sets the Y co-ordinate for the input ports of the node.

The method *actionPerformed(ActionEvent ae)* handles all the node popup menu actions.

The method *reloadImage()* reloads the image for the node whenever user change the function of a node.

The method *paintComponent(Graphics g)* repaints the node whenever it is required, such as node has been dragged, or node popup menu has been invoked etc.

The method *processMouseEvent(MouseEvent me)* handles all user mouse actions, such as user double click on the node for the parameter dialog box, or click the right mouse button for node popup menu, or press the mouse to drag the node etc.

The method *processMouseMotionEvent(MouseEvent me)* this method is responsible for node drag operation.

The method *checkSampleT(double time)* it returns true if the time is a valid time-point with respect to the sample time of the node.

The method *initDialog(String title, int w, int h)* initialises the dialog box that will be used as parameter input dialog for the node.

The method *showDialog()* this method displays the parameter dialog box for this node when user double click on the node.

The method *resolveDimension()* expands the input or parameter of the node. It is mentioned in section 3.3.4 that most of the nodes support auto expansion of node input or parameter. This method handles this expansion process. It uses the *canBeResolved* method from the *Basic* class to check which one (parameter or input) should be expanded, then takes apposite steps for expansion.

The method *isOpOK(double simT)* checks that output at the simulation time simT is valid. Sometimes a node may produce invalid (infinity or not a number) output. So at each time step this method checks the validity of the output produced by the node.

The method *upDateOutput(double simT)* updates the output of the node at simulation time simT.

The method *initialize()* initialises the parameters of the node before a simulation run starts.

The method *makeDialog()* prepares the node specific dialog box for parameter input.

The method *initDialog* method initialises a general dialog box for parameters. Every method implements this method to customize the dialog box initialised by *initDialog* method.

The method *cancel()* abstract method is used to perform the cancel operation of the parameter dialog box. If user clicks the cancel button in the parameter dialog box this method restores the previous parameter values.

Abstract method *checkConsistency()* is used to handle the ok action of the parameter dialog box. If user changes the parameter and clicks ok button in the parameter dialog box, this method will parse the new parameters. It also generates error messages if user enters invalid parameter.

action(double simT) is the most important method in the class. It is mentioned in section 3.1 that every node in *SimDynamic* carries out a specific function, such as integration, generation of sine wave etc. at each simulation time step. All the nodes implement this action method to perform its specific function at simulation time simT.

4.2.5 Integrator

The dynamic models are generally described by differential equations. The solutions of these equations display the operations of the system and this is the fundamental property of dynamic models. This class provides a number of numerical methods to solve ordinary differential equations (ODE). Because of the diversity of dynamic system behavior, some methods may be more efficient than others at solving a particular problem.

```
public class Integrator extends Object{
      public Integrator();
      public Vector integrate(Vector A, Vector X, Vector B, Vector Y,
             Vector PRV, int s_p_c, int row, double step);
      public void initialize(int solver, int row, int col);
}
```

Constructor *Integrator* creates a new integrator and allocates memory for co-effiecient tables.

The method *integrate(Vector A, Vector X, Vector B, Vector Y, Vector PRV, int s_p_c, int row, double step)* calculates integral value of a differential equation of the form,

$$\frac{dx}{dt} = AX + BY$$

where A and B may be matrices or one dimensional arrays, X and Y must be one dimensional arrays, at the current simulation time step using the integral values from the previous time step. This method extensively uses various explicit Runge-Kutta formulas to solve the equation. Runge-Kutta formulas used by this method are briefly described in section 4.4.

initialize(int solver, int row, int col) method initializes the tables for the selected Runge-Kutta method.

4.2.6 Simulator

This is the main simulation engine and it controls the whole simulation process of a model. The simulator class makes the nodes to interact as a system and generate desired output. The way it accomplishes this, is described in section 4.5.

```
public class Simulator extends Object{
      public Vector nodeV;
      public Vector lineV;
      public int nodeID;
      public int lineID;

      public Simulator(BlankSheet blsh);
      public void init();
      public boolean simulate();
      public void removeLine(int id);
      public void removeNode(int id);
}
```

Constructor *Simulator* creates a new simulator for the *BlankSheet* blsh. This class is responsible for keeping track of all the lines and nodes in a model. Whenever a new line is drawn in a model, its ID is set equal to *lineID*, and *lineID* is incremented by one. In the same way *nodeID* is used to keep record of the nodes in a model. Vectors *nodeV* and *lineV* are used to store all the nodes and lines in a model respectively. This class also maintains the simulation clock.

The method, *init()* initialises all the variables (such as set *lineID* and *nodeID* to zero, make sure that *nodeV* and *lineV* are empty) at the beginning of a model construction.

The *simulate()* method is the main simulation engine. It carries out the whole simulation run. Its function is described in section 4.5.

The method *removeLine(int id)* removes a line from the vector *lineV*. When user deletes a line int a model this method is invoked with the id of that line. It removes the line from the vector and adjusts ids of the remaining lines.

The method *removeNode(int id)* removes a node from the vector *nodeV*. As user deletes a node from a model this method is invoked with the id of that node. It removes the node from the vector and adjusts ids of the remaining nodes.

4.3 Anim3D

This package contains all the classes responsible for 3D animation. Although Anim3D has been included as a package of *SimDynamic*, this can be used as a separate application to generate 3D animation.

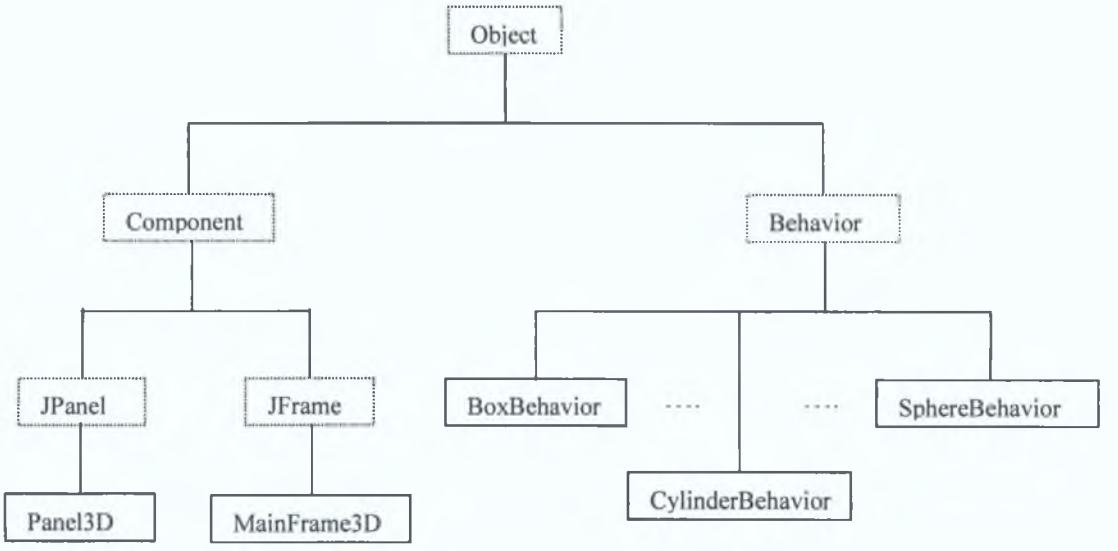


Fig 4.4: Class hierarchy in Anim3D package

4.3.1 MainFrame3D

This class provides the 3D model editor user interface.

```
public class MainFrame3D extends JFrame implements ActionListener{
      public MainFrame3D();
      public void actionPerformed(ActionEvent ae);
}
```

Constructor *MainFrame3D* initialises all the dialog boxes and buttons.
actionPerformed method handles all the button action and invokes appropriate dialog box for input.

4.3.2 Panel3D

Panel3D class constructs the 3D scene and adds the scene into a *Jpanel.* Its definition is as follows,

```
public class Panel3D extends JPanel{
      public BranchGroup createSceneGraph(Vector nodeV,
             UniInfo uniinfo) ;
      public Panel3D(Vector nodeV, UniInfo uniinfo);
}
```

Method *createSceneGraph* creates the whole 3D scene. Vector nodeV contains parameters for every single object of the 3D model. This method goes through the nodeV vector and creates all the 3D objects and adds them to the java 3D BranchGroup [46].

Constructor *Panel3D* creates a SimpleUniverse [46] and adds the BranchGroup, created by *createSceneGraph* method, to it. Finally this method adds the SimpleUniverse to the centre of the *Jpanel*.

4.3.3 Behavior Classes

All the behavior classes in the package like *BoxBehavior* or *CylinderBehavior* are responsible for the movement of the respective object in the 3D animation. Since all the behavior classes have same structure, here *BoxBehavior* class is used to describe the structure

```
public class BoxBehavior extends Behavior {
      public BoxBehavior(TransformGroup tg, Struct3D st3d, UniInfo
         uniinfo);
      public void initialize();
      public void processStimulus(Enumeration criteria);
}
```

Constructor *BoxBehavior* initialises all parameters for the box object.

initialize method allows a Behavior object to initialise its internal state and specify its initial wakeup condition.

processStimulus method is the main method in the class. At each simulation time step this method reads the current position (x, y, z co-ordinate) of the object from the file and move the object to the new position, thereby animates the 3D model.

4.4 Solution of Ordinary Differential Equations

An ordinary differential equation or ODE [47] for short, may be thought of as a differential equality specifying the relationship between a dependent variable, say y, and an independent variable, say x. The *order* of the ODE is the order of the highest derivative of y with respect to x appearing in it. Hence, $dy/dx + x^2 y = 0$ is a first order

ODE whereas $d^2 y/dx^2 + ky = 0$ is a second order ODE. The problem consisting of a differential equation together with an initial value is called an initial value problem (IVP). Here all the y_n are given at some starting value t_n, and it is desired to find the y_{n+1} at some final point t_{n+1} , or at some discrete list of points.

The general form of an IVP is as follows

$$\frac{dy}{dt} = f(y,t), \quad y(0) = b$$

where b is a constant and the function *f(x,y)* is given.

Integrator class uses Euler's method and Runge-Kutta methods to solve this sort of IVPs. A short description of each methods used in *SimDynamic* is presented in the following sections

4.4.1 Euler's Method

In Euler's [48] method y_{n+1} is calculated from y_n using the formula

$$y_{n+1} = y_n + h * f(y_n, t_n)$$

where h is the step size.

This method is called a one-step method, since the information from the single previous step is used to calculate the current approximation. Since the error in Euler's method is of O(h), one has to choose a very small step size to get desired accuracy. So, this method is not preferred over the Runge-Kutta method, as the errors in Runge-Kutta methods are much smaller than those of Euler's method.

4.4.2 Heun's Method

In this [48] method, y_{n+1} is calculated using the following steps:

$$k_1 = f(y_n, t_n)$$
$$k_2 = f(y_n + h * k_1, t_n + h)$$
$$y_{n+1} = y_n + h * \frac{k_1 + k_2}{2}$$

where h is the step size.

Here k_1 is the slope at the point (y_n, t_n), k_2 is the slope at the point $(y_n + h * k_1, t_n + h)$. The average of the two slopes were used to calculate the next approximation y_{n+1} from y_n. The use of mid-point slope cancels out the first order error term making the system second order. It provides much better accuracy over Euler's method.

4.4.3 Runge-Kutta Method

The idea behind Runge-Kutta methods is to compute the value of f(t, y) at several strategically chosen points near the solution curve in the interval $(t_n, t_n + h)$, where h is the step size, and to combine these values in such a way as to get good accuracy in y_{n+1}. Each step of a Runge-Kutta method goes through the following calculations:

$$
\begin{aligned}
k_0 &= h * f(t_n + \alpha_0 * h, \quad y_n) \\
k_1 &= h * f(t_n + \alpha_1 * h, \quad y_n + \beta_{10} * k_0) \\
k_2 &= h * f(t_n + \alpha_2 * h, \quad y_n + \beta_{20} * k_0 + \beta_{21} * k_1) \\
&\cdots\cdots\cdots \\
k_r &= h * f(t_n + \alpha_r * h, \quad y_n + \beta_{r0} * k_0 + \beta_{r1} * k_1 + \cdots\cdots + \beta_{rr-1} * k_{r-1} \\
y_{n+1} &= y_n + w_0 * k_0 + w_1 * k_1 + \cdots\cdots + w_r * k_r
\end{aligned}
$$

The numbers α_i, β_{ij} and w_j are given by the inventors of the method in the form of a table (see appendix B for numerical coefficients)

$$
\begin{array}{l}
\alpha_0 \\
\alpha_1 \beta_{10} \\
\alpha_2 \beta_{20} \beta_{21} \\
\cdots \quad \cdots \quad \cdots \\
\cdots \quad \cdots \quad \cdots \quad \cdots \\
\alpha_r \beta_{r0} \beta_{r1} \cdots\cdots \beta_{rr-1} \\
\quad w_0 w_1 \cdots\cdots w_r
\end{array}
$$

Bogacki and Shampine [49] described a third order Runge-Kutta method with second order error estimate. In this method y is calculated using the following steps,

$$k_1 = y_n + \frac{h}{2} * f(t_n, \quad y_n)$$

$$k_2 = y_n + \frac{3h}{4} * f(t_n + \frac{h}{2}, \quad k_1)$$

$$y_{n+1} = y_n + h * (\frac{2}{9} * f(t_n, \quad y_n) + \frac{1}{3} * f(t_n + \frac{h}{2}, \quad k_1) + \frac{4}{9} * f(t_n + \frac{3h}{4}, \quad k_2))$$

$$\hat{y}_{n+1} = y_n + h * (\frac{7}{24} * f(t_n, \quad y_n) + \frac{1}{4} * f(t_n + \frac{h}{2}, \quad k_1) + \frac{1}{3} * f(t_n + \frac{3h}{4}, \quad k_2) + \frac{1}{8} * f(t_n + h, \quad y_{n+1}))$$

where y_{n+1} is a third order and $\hat{y}_{n+1}$ is a second order estimate of the solution. This requires three function evaluations per step, the term $f(t_n + h, y_{n+1})$ is carried over to the next step where it becomes $f(t_n, y_n)$.

The fourth-order Runge-Kutta [50] method is a commonly used class of Runge-Kutta methods. The error in this method is $O(h^4)$. In this method y_{n+1} is calculated using the following steps,

$$k_1 = f(t_n, \quad y_n)$$

$$k_2 = f(t_n + \frac{h}{2}, \quad y_n + h * \frac{k_1}{2})$$

$$k_3 = f(t_n + \frac{h}{2}, \quad y_n + h * \frac{k_2}{2})$$

$$k_4 = f(t_n + h, \quad y_n + h * k_3)$$

$$y_{n+1} = y_n + (k_1 + 2 * k_2 + 2 * k_3 + k_4) * \frac{h}{6}$$

Dormand-Prince [51] method requires calculation of seven k_j values, k_0, k_1, …, k_6, for each integration step h. From these k values two approximations can be found as follows,

$$y_{n+1} = \sum_{j=0}^{6} w_j * k_j$$

$$\hat{y}_{n+1} = \sum_{j=0}^{6} \hat{w}_j * k_j$$

$\hat{y}_{n+1}$ is the more accurate approximation and has local error of order 5. y_{n+1} is only order 4, and is usually less accurate.

4.5 How *SimDynamic* Works

Simulating a dynamic system is nothing but the process of computing the states and outputs of the system over a particular period of time. In *SimDynamic* each functional node calculates its own output and state at a particular time step through the *action* method. So the only requirement to simulate a model is to co-ordinate the functional nodes to make them interact as a system, and the *Simulator* class does this. When a user clicks the *Start* from the Simulation menu, the *Simulator* class takes over the control and the simulation starts. It accomplishes its task in the following two phases

4.5.1 Initialisation Phase

In this phase *Simulator* performs the following tasks in the order they are described

Checks for Integrator: *Simulator* checks whether the model needs an integrator. If it does need an integrator and no integrator has been selected *Simulator* will generate error message.

Sorts Lines: It should be make sure that all the lines in the model is connected to nodes. If there is any dangling line in the model, error message will be generated. At this stage source node for a branch line is also determined.

Sets input/output nodes of each node: It is mentioned in section 3.3.3, that the lines in a model represent logical, not physical, connections among nodes. And during a simulation run a node receives its input directly from another node. So every node needs to know the address (id) of its predecessor and successor nodes. To find the predecessor/successor nodes of a node, *Simulator* traces each line connected to the node and retrieves the id of the node connected to the other end of the line. It reports error message if fails to resolve any node.

Determines the execution order of nodes: During a simulation, *SimDynamic* updates the states and outputs of a model's nodes once per time step. The order in

which the nodes are updated is therefore critical to the validity of the results. In particular, if a node's outputs are a function of its inputs at the current time step, the node must be updated after the nodes from which it receives its inputs. Otherwise, the node's outputs will be invalid. The order in which nodes are added in a model is not necessarily the order in which they need to be updated during a simulation. Consequently, *SimDynamic* sorts the nodes into the correct order during the initialization phase.

To facilitate this sorting process nodes are classified as follows,

- Direct feedthrough nodes: nodes whose current outputs depend on their current inputs are called *direct feedthrough* nodes.
- Non-direct feedthrough nodes: nodes whose current outputs do not depend on their current inputs are called *non-direct feedthrough* nodes.
- Source nodes: nodes that do not need any input to produce output.
- Sink nodes: nodes that do not produce any output.

With this node classification *Simulator* uses the following rules to determine node execution order

- Source nodes should be executed before any nodes in the model. This ensures that their output will be valid at every time step.
- Sink nodes should be updated after all the nodes in the model have been updated. This ensures that sink nodes will always receive valid input.
- Every other node must be executed before any of the direct feedthrough nodes that it drives. This rule ensures that inputs to direct feedthrough nodes will be valid when they are updated.
- Non-direct feedthrough nodes can be updated in any order as long as they are updated before any direct feedthrough nodes they drive. This can be accomplished without much difficulty by putting all the non-direct feedthrough nodes of a model just after the source nodes in the execution list.

By applying above rules *Simulator* generates a node execution list, in which source nodes appear at the head followed by the non-direct feedthrough nodes in no

particular order. Then comes the direct feedthrough nodes in the order required to supply valid input to the nodes they drive, and the list ends with the sink nodes.

Sets sample time for each node: Nodes in the continuous node set and some other nodes have continuous sample time. Some nodes (nodes in discrete node set) have sample times that are explicitly specified by the user on the node dialog boxes. All other nodes have implicitly defined sample times that are based on the sample times of their inputs. For nodes whose inputs have different sample times, the node is assigned the sample time of the fastest input. *Simulator* also checks that sample time of each node is an integer multiple of the simulation step size.

Resolve dimension for node inputs and parameters: It is mentioned in section 3.3.4, that most of the nodes support auto expansion of inputs and parameters. At this stage *simulator* checks each node in the model for the possibility of input/parameter expansion. If it fails to resolve input/parameter dimension for any node, error message is generated.

4.5.2 Execution Phase

In this phase, *Simulator* successively computes the states and outputs of the system at intervals from the simulation start time to the finish time, using information provided by the model. Simulation starts at the starting time specified by the user on the simulation parameter dialog box and *Simulator* advances its clock by adding step length to the current clock time until it reaches the stop time. At each step *Simulator* computes a node's states and outputs by invoking the *action* method of that node. It passes the current simulation time and step length to the node's *action* method. The node itself derives the input for this simulation time step from its previous node. Using this input, simulation time, step and the node's state (if any) at previous simulation time step, *action* method computes the valid output and state for current simulation step. After the *action* method *Simulator* invokes the node's *isOpOk* method to ensure the validity of currently calculated output. At the end of each step *Simulator* computes the simulation time for the next step and advances its clock to that time. At the end of the simulation time each node holds the final values of the model's output and state.

Chapter Five

5 Functional Description of the Nodes

This chapter provides a brief functional description of each node in *SimDynamic*. Each description also mentiones all the parameters for the node that the user can set on the node parameter dialog box. Nodes are discussed here according to the node set library. For characteristic properties of nodes see appendix A. These nodes have been selected carefully so that the package can be used to model a wide range of complex systems. In some cases ideas have been borrowed from existing non Web-based packages like VisSim and Simulink.

5.1 Continuous Node Set

5.1.1 Derivative

The derivative node calculates the derivative of its input signal by using the following formula

$$\frac{u_t - u_{t-1}}{\Delta t}$$

where, u_t is the input at current simulation time, u_{t-1} is the input at the previous time step, and Δt is the change in time since the previous simulation time step. Initial (at the simulation start time) output from the node is always 0.

Parameters:

- None.

5.1.2 Integrator

The integrator node integrates its input signal and outputs the integral. *SimDynamic* provides a good number of fixed step integrators as described in section 4.2.5. Integrator node uses those integrators to find the integral of its input signal. User can define the initial condition on the node dialog box (internal initial condition) or as input to the node (external initial condition). By selecting one of the external reset choices from the node dialog box, user can reset the current state of the node to

specified initial condition. This implementation provides following three types of reset

Rising – current state of the node will be set to the initial condition, if the signal to the reset port changes from negative to positive.

Falling – current state of the node will be set to the initial condition, if the signal to the reset port changes from positive to negative.

Either – current state of the node will be set to the initial condition, if the signal to the reset port changes from positive to negative or vice versa.

Parameters:

- External Reset: By selecting one of the reset mode (described above) states of the node can be reset to the initial condition. Default is none.
- Initial Condition: Initial value that will be used to solve the IVP. Default value is 0. If user selects external initial condition then the node will accept initial condition as an input.

5.1.3 Memory

The memory node stores the input at current simulation time step and outputs it to the next simulation time step. So this node applies a one-simulation time step delay to its input signal. At the start of the simulation memory node outputs the initial condition parameter value given on the node dialog box.

Parameters:

- Initial Value: This will be the output at the initial simulation time step. Default is 0.

5.1.4 State Space

This node implements a linear state-space [52] system defined by the following equations

$$\frac{dx}{dt} = Ax(t) + Bu(t)$$
$$y = Cx(t) + Du(t)$$

where, x is the state vector of the node, u is the system input, and y is the system output. The matrix coefficients must have the following dimensions:

- A must be an n x n matrix, where n is the number of states.
- B must be an n x m matrix, where m is the number of inputs.
- C must be an r x n matrix, where r is the number of outputs.
- D must be an r x m matrix.

User can define the initial condition on the node parameter dialog box, and initial condition must maintain the dimension provided in the matrix coefficients.

Parameters:

- Coefficient Matrices: A, B, C, and D matrix of the state-space. Default values are all one.
- Initial Condition: Initial state value that will be used to solve the differential equation. Default value is 0.

5.1.5 Transfer Function

The transfer function node finds the value of a transfer function [52] of following form at the current simulation time,

$$G(s) = \frac{n(s)}{d(s)}$$

where,

$$n(s) = n_1 s^{n-1} + n_2 s^{n-2} + \ldots\ldots\ldots\ldots + n_n$$
$$d(s) = d_1 s^{m-1} + d_2 s^{m-2} + \ldots\ldots\ldots\ldots + d_m$$

The user has to set the coefficients for the numerator and denominator on the parameter dialog box. The order of denominator must be greater than or equal to the order of numerator. Numerator can be a vector or matrix, but denominator must be a vector. Width of the output from the node depends on the dimension of the numerator. If the numerator is a vector output will be single number. If the numerator is a matrix output will be a vector with width equal to the number of rows in the numerator.

Parameters:

- Numerator: Numerator coefficients of the transfer function. Default is 1.
- Denominator: Denominator coefficients of the transfer function. Default values is 1 1.

5.1.6 Transport Delay

The transport delay node can be used to delay its input by a specified amount of time. The node outputs the initial output parameter until the simulation time exceeds the time delay parameter. From this point the node starts generating delayed input. The node stores its input and simulation time internally. When simulation time exceeds time delay parameter, the node finds the output at the time that corresponds to the current simulation time minus the delay time. If the time delay parameter is not a multiple of the step size then the node performs linear interpolation or extrapolation on the stored time and input value pairs to calculate the output. Following example will clarify this

Let simulation step size be 0.2, and the node receives following inputs from a clock node

Simulation time	0.0	0.2	0.4	0.6	0.8
Input value	0.0	0.2	0.4	0.6	0.8

and initial output parameter is set to 5. Following table shows the output from the node when time delay parameter is set to 0.05 and 0.45

Simulation time	0.0	0.2	0.4	0.6	0.8
Output (with time delay 0.05)	5.0	0.0	0.35	0.55	0.75
Output (with time delay 0.45)	5.0	5.0	5.0	0.15	0.35

Parameters:

- Initial Output: The node will continue to output this value before simulation time exceeds time delay parameter. Default is 0.
- Time Delay: Input signal will be delayed this amount of time. Default value is 1.

5.1.7 Variable Transport Delay

The variable transport delay node can be used to delay its input by a variable amount of time. It works the same way as the transport delay node. Only difference is that this node accepts the time delay parameter as an input.

Parameters:

- Initial Output: The node will continue to output this value before delayed output is generated.

5.1.8 Zero – Pole

The zero – pole node finds the value of transfer function expressed in zero-pole-gain form (see below) at the current simulation time

$$H(s) = G\frac{Z(s)}{P(s)}$$

where, G is the gain value and

$$Z(s) = (s\text{-}z_1)(s\text{-}z_2)\ldots\ldots(s\text{-}z_n)$$

$$P(s) = (s\text{-}p_1)(s\text{-}p_2)\ldots\ldots(s\text{-}p_m)$$

The user should set the poles (*p1, p2,...*) and zeros (*z1, z2....*) on the parameter dialog box. The number of poles must be greater than or equal to the number of zeros. Zeros may be vector or matrix, but poles must be a vector. Width of the output from the node is determined by the dimension of the zeros. If zeros is a vector then output will be a single number. If zeros is a matrix then output will be a vector with width equal to the number of columns of the matrix. The zero – pole node converts the given zeros and poles into polynomial form and then carries out the calculation like the transfer function node.

Parameters:

- Zeros: Values for the zeros. Default is 1.
- Poles: Values for the poles. Default is 0 –1.
- Gain: Zeros will be multiplied by gain values. Default is 1.

5.2 Discrete Node Set

5.2.1 Discrete Integrator

This node computes discrete time integration [53] of its input signal. For an initial value problem of the following form this node provides following three methods of integration,

$$\frac{dy}{dt} = f(y,t), \quad y(0) = b$$

Forward Euler Method,

$$y_{n+1} = y_n + h * f(y_n, t_n)$$

Backward Euler Method,

$$y_{n+1} = y_n + h * f(y_{n+1}, t_{n+1})$$

Trapezoidal Method,

$$y_{n+1} = y_n + h/2 * f(y_n, t_n)$$

This node allows the user to set initial condition on the parameter dialog box (internal initial condition), or as an input to the node (external initial condition). Like the continuous integrator, discrete time integrator also provides three types of state reset.

Parameters:

- Integrator: One of the integration methods (described above). Default is Forward Euler Method.
- External Reset: By selecting one of the reset modes (rising/falling/either) states of the node can be reset to the initial condition. Default is none.
- Initial Condition: Value that will be used to solve the IVP. Default value is 0. If user selects external initial condition then the node will accept initial condition as an input.
- Sample Time: The time interval at which output will be updated. Default is 1.

5.2.2 Discrete State-Space

The discrete state-space node can be used to describe a system by discrete state-variable as follows

$$x(t+1) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

where, u(t) is the system input at time t, x(t) is the state of the system at time t, and y(t) is the system output at time t. Coefficient matrices A, B, C, D must follow the same dimension constraint as the continuous state-space node.

Parameters:

- Coefficient Matrices: A, B, C, and D matrix of the state-space. Default values are all one.
- Initial Condition: Initial state value that will be used to solve the differential equation. Default value is 0.
- Sample Time: The time interval at which output will be updated. Default is 1.

5.2.3 Discrete Transfer Function

This node implements a z-transform transfer function described by the following equation,

$$H(z) = \frac{n(z)}{d(z)}$$

where,

$$n(z) = n_0 z^n + n_1 z^{n-1} + \ldots\ldots + n_m z^{n-m}$$
$$d(z) = d_0 z^n + d_1 z^{n-1} + \ldots\ldots\ldots\ldots + d_n$$

Numerator can be a vector or a matrix, but denominator must be a vector. The order of denominator must be greater than or equal to the order of numerator. Numerator can be a vector or matrix, but denominator must be a vector. Width of the output from the node depends on the dimension of the numerator. If the numerator is a vector

output will be single number. If the numerator is a matrix output will be a vector with width equal to the number of rows in the numerator.

Parameters:

- Numerator: Numerator coefficients of the transfer function. Default is 1.
- Denominator: Denominator coefficients of the transfer function. Default values is 1 2.
- Sample Time: The time interval at which output will be updated. Default is 1.

5.2.4 Discrete Zero-Pole

The discrete zero-pole node can be used to express a discrete transfer function in terms of poles-zeros-gain form as follows,

$$H(z) = G\frac{Z(z)}{P(z)}$$

where G is gain value and,

$$Z(z) = (z - Z_1)(z - Z_2)\ldots\ldots\ldots(z - Z_m)$$
$$P(z) = (z - P_1)(z - P_2)\ldots\ldots\ldots(z - P_3)$$

The user should set the poles (*p1, p2,...*) and zeros (*z1, z2....*) on the parameter dialog box. The number of poles must be greater than or equal to the number of zeros. Zeros may be vector or matrix, but poles must be a vector. Width of the output from the node is determined by the dimension of the zeros. If zeros is a vector then output will be a single number. If zeros is a matrix then output will be a vector with width equal to the number of columns of the matrix. The discrete zero – pole node converts the given zeros and poles into polynomial form and then carries out the calculation like the discrete transfer function node.

Parameters:

- Zeros: Values for the zeros. Default is 1.
- Poles: Values for the poles. Default is 1 2.

- Gain: Zeros will be multiplied by gain values. Default is 1.
- Sample Time: The time interval at which output will be updated. Default is 1.

5.2.5 Discrete Filter

The discrete filter node can be used to implement digital filters [54] like Infinite Impulse Response (IIR) and Finite Impulse Response (FIR) filters. It actually evaluates a discrete transfer function expressed in terms unit delay operator z^{-1}. So, the coefficients of numerator and denominator should be expressed in ascending powers of z^{-1}.

Parameters:

- Numerator: Numerator coefficients of the transfer function. Default is 1.
- Denominator: Denominator coefficients of the transfer function. Default values is 1 2.
- Sample Time: The time interval at which output will be updated. Default is 1.

5.2.6 Unit Delay

The unit delay node delays and holds its input signal by one sampling interval. Memory node in the continuous node set delays its input signal by one simulation time step, but this node can be used to delay a signal by a multiple of simulation time step. User has to specify the sample time on the parameter dialog box and an initial output value. At the start of the simulation the node outputs the initial value after then it outputs delayed signal.

Parameters:

- Initial Condition: The node outputs this value for the first sample interval. Default value is 0.
- Sample Time: The time interval at which output will be updated. Default is 1.

5.3 Tables Node Set

5.3.1 Direct Look-Up Table (2D)

The direct look-up table enables user to find a single element or a column from a 2D table. For searching a single element node accepts two inputs, a row index and a column index. To output a column this node needs only one input, the index of the column. For example following table is set on the parameter dialog box,

2+3i	5.94	6.21	-7+2i
0	9.34	21	45.32
34.67	-56	5-9i	12.43

If the node is in element selection mode and it receives input 2, 2 then it will output 5-9i. If the node is in column selection mode and it receives input 1, then output from the node will be the second column (5.94, 9.34, -56) of the table.

Parameters:

- Action Type: Selecting an element or column from the table. Default is element selection.
- Table Data: The table to be searched. Default is the following table

2	3	4
56	7-2i	80
2	-5	90+5i

5.3.2 Look – Up Table

The look-up table node maintains two one-dimensional tables, one for input values and one for output values. It tries to map its input to the elements of the input table and outputs the corresponding element from the output table. If the node fails to find a direct mapping then it performs linear interpolation/extrapolation to calculate the desired output. Both tables must be of same size.

For example, let following values are given as the input and output table on the node parameter dialog box,

Input values	4.30	7.22	13.0	18.12

Output values	5.66	3.20	9.31	-7.60

If the node receives 7.22 as input then output will be 3.20. If it receives 3.0 as input then output will be 6.755205, for input value 19, output will be –10.506406, for input 5.0 output will be 5.070274.

Parameters:

- Input Values: Holds the possible node input values. Default input values are

3	5	6	10	12

- Output Values: Holds the possible output values from the node corresponding to the values in the input values. Default values are

8	25	12	30	40

5.3.3 Look – Up Table (2D)

The look-up table (2D) holds the possible output values as a 2D table. It also maintains a row table and a column table that corresponds to the rows and columns of the output table. The node accepts two inputs, a row index and a column index. It then tries to map the row index value to the elements of the row table and column index value to the elements of the column table respectively, and outputs the corresponding element from the output table. If the node fails to find a direct mapping, it performs linear interpolation/extrapolation to find the output. For example following tables are given on the parameter dialog box, and the node receives 1.5 and 4.3 as row and column index respectively then output will be 48.861974. For -3.1 and 2.41 as row and column index respectively, output will be 62.384233.

Col → / Row ↓	9.0	21.43	30.21	32.56
4.32	35.14	15.52	34.22	23.0
10.29	22.31	4.05	53.96	21.53
17.0	12.22	27.06	76.13	33.25
19.11	17.32	-11.5	13.0	26.33

Parameters:

- Row Table: The row values for the table. Default values are,

4	10	17

- Column Table: The column values for the table. Default values are,

9	21	30

- Table Data: The table of output values. Default table is

35	15	34
22	40	53
12	27	76

5.3.4 Polynomial

This node evaluates the value of a polynomial of the following form at current simulation time,

$$p(t) = a_n t^n + a_{n-1} t^{n-1} + \ldots\ldots + a_1 t + a_0$$

Polynomial coefficients can be entered on the parameter dialog box in descending order of power.

Parameters:

- Coefficients: Coefficients of the polynomial in decreasing order of power until the last coefficient, which represents the constant for the polynomial. Default values are, 2 3 4 5

5.3.5 Pre Look – Up Index Search

The pre look-up index search node maintains a table, sorted in ascending order, and finds the index of its input in the table. If it fails to find index directly from the table, it performs linear interpolation/extrapolation to find the index of its input. For each input number the node outputs a (index, fraction) pair, where, index is the index of the number in the table that is less then or equal to the input and fraction is the normalized position of the input between the index and the next index. For example, let following table has been entered on the parameter dialog box

3	5	6	10	12

If the node input is 9.37, output will be (2, 0.8425).

Parameters:

- Table Data: The numbers to search. Default values, 3 5 6 10 12

5.3.6 Interpolation (2D) Using Pre Look – Up

The interpolation node holds possible output values in a 2D table and it accepts two input (row and column) pair (index fraction pair) from pre look-up index search. It then tries to find the output value by using received row and column indices. If it fails to find a direct mapping of the indices, it performs linear interpolation. For example, let following table is the given output table on the parameter dialog box,

10	50	20
30	10	-50
13	21	90

If the node receives (0, 0.75) and (0, -0.45) as row and column index respectively from a pre look-up index search node, then output will be 27.25.

Parameters:

- Table Data: The table of possible output values. Default table is

10	50	20
30	10	-50
13	21	90

5.4 Maths and Logic Node Set

5.4.1 Absolute

This node finds the absolute value of its input. If input is real value x, then output will be |x|. And if input is a complex number x+iy, then output will be $|\sqrt{x^2 + y^2}|$.

Parameters:

- None

5.4.2 Bit-wise Logical Operator

This node can be used to perform bit-wise logical operation on any signal. The node applies one (selected by user) of the following logical operations between the input signal and the second operand provided in the dialog box,

- AND
- OR
- XOR
- NOT
- SHIFT RIGHT
- SHIFT LEFT

User can use hexadecimal or decimal value for the second operand. Since *SimDynamic* considers all real value signals as double, user has to specify in the parameter dialog box how to treat the input signal as an integer.

Parameters:

- Input Data Type: User can use this option to specify input data type. Input can be a 8-bit integer, 16-bit integer, 32-bit integer or 64-bit integer. Default is 64-bit integer.
- Second Operand: This value will be used as the second operand for the bit-wise operation with the node input. User has the option to set this value as a decimal number or hexadecimal number. Default value is decimal 10.
- Operator: Specifies one of the 6 operator applied to the input signal. Default is AND.

5.4.3 Dot Product

The dot product node generates the dot product of its two input vectors using the following formula,

$$y = \sum conjugate(u_i) * v_i$$

where u and v are the input vectors. To have element wise multiplication between two vectors without summing, user has to use product node. For example, let the node receives u = (2-3i, 5+6i) and v = (1+2i, 3-i) as input then output from the node will be, 5-16i.

Parameters:

- None

5.4.4 Gain

The gain node multiplies its input signal by a specified gain factor provided in the parameter dialog box. Let 2+5i has been set as gain parameter and the node input is (4-3i, 7) then output will be (23+14i, 14+35i).

Parameters:

- Gain: The multiplication factor. Default is 1.

5.4.5 Logical Operator

The logical operator node performs one (selected by user) of the following logical operations on the input

- AND
- OR
- NOT
- XOR
- NAND
- NOR

The node accepts both real and boolean values. For real value input the node considers any non-zero (positive/negative) value as true and zero value as false.

Parameters:

- Output Type: Selects the output type (boolean/real) from the node. Default is real.
- Logical Operator: One of the six operators. Default is AND.

5.4.6 Magnitude-Angle to Complex

This node receives a magnitude and a phase angle (in radian) and outputs the corresponding complex number. If the inputs to the node are r and θ, then output will be $x+iy$, where,
$x = r * cos\ \theta$ and $y = r * sin\ \theta$.

Parameters:

- None.

5.4.7 Math Function 1

If u is the input to the node, then this node can apply any (selected by user) of the following functions on u

- Power 10, 10^u
- Exponential, e^u
- Natural log, $\ln u$
- Square, u^2
- Square root, $\sqrt{u}$
- Conjugate(u)

- 10 base log, $\log_{10}u$
- 2 base log, $\log_2 u$
- Magnitude square, $|u|^2$
- Reciprocal, 1/u
- Factorial, u!

Parameters:

- Function: Selects one of the above functions. Default is Power 10.

5.4.8 Math Function 2

Let *n* and *r* be the inputs to the node, then math function 2 can apply any (selected by the user) of the following functions on *n* and *r*,

- Power, n^r
- Hypotenuse, $\sqrt{n^2 + r^2}$
- Modulus, n % r
- Combination, ${}^n c_r$
- Permutation, ${}^n p_r$
- Arc-tan, $\tan^{-1} n/r$

Parameters:

- Function: Selects one of the above functions. Default is Power.

5.4.9 Matrix 1

The matrix 1 node accepts a matrix as its input and can apply one (selected by user) of the following functions on the matrix,

- Conjugate
- Determinant
- Hermitian
- Inverse
- Trace
- Transpose

Parameters:

- Function: Selects one of the above functions. Default is Conjugate.

5.4.10 Matrix 2

The matrix 2 node accepts two matrices as its input and apply one (selected by user) of the following functions on the matrices

- Addition (element wise)
- Subtraction (element wise)
- Multiplication
- Division (element wise)

Parameters:

- Function: Selects one of the above functions. Default is Add.

5.4.11 Min Max

The min max node outputs the minimum or maximum (specified by user) of its input values.

Parameters:

- Function: Selects either minimum or maximum. Default is Maximum.

5.4.12 Product

The product node can be used to have element wise multiplication or division of two signals. The node accepts two inputs (single number/ vector / matrix) and performs the specified operation on them. To perform matrix multiplication, use matrix 2 node.

Parameters:

- Function: Either multiplication or division. Default is multiplication.

5.4.13 Real-Imaginary to Complex

This node accepts two real values as input signal and outputs a complex number treating the input values as real and imaginary part of the complex number.

Parameters:

- None

5.4.14 Relational Operator

This node can be used to compare two signals. It accepts two inputs and performs one (selected by user) of the following relational operation on them

- Equal to, = =
- Not equal to, ! =
- Less than, <
- Less than equal to, <=
- Greater than, >
- Greater than equal to, >=

The output type from the node may be boolean or real (specified by user). If the result is true, the output is 1 for real type output, *true* for boolean type output; if false, it is 0 for real type output, *false* for boolean type output.

Parameters:

- Output Data Type: Selects the data type (boolean/real) of the output from the node. Default is real.
- Relational Operator: One of the six operators. Default is Equal to.

5.4.15 Resolve Complex

The resolve complex node can be used to find the magnitude or phase angle or real part or imaginary part of a complex signal.

Parameters:

- Function: Selects one of the four (magnitude/angle/real/imaginary) outputs. Default is magnitude.

5.4.16 Rounding Function

It performs one (selected by user) of the following rounding operation on its input,

- Round(x) – finds the nearest integer of x.
- Floor(x) – finds the largest integer not greater than x.
- Ceil(x) – finds the smallest integer not smaller than x
- Fix(x) – rounds to the nearest integer towards zero.

Parameters:

- Function: Selects one of the four functions. Default is Round(x).

5.4.17 Sign

The sign node can be used to indicate the sign of a signal. If the input to the node is positive, output will be 1; if it is negative, output will be –1; if it is 0, output will be 0.

Parameters:

- None.

5.4.18 Sum

The sum node adds its inputs element wise. By changing the sign (+/ –) of inputs subtraction can also be performed.

Parameters:

- Sign of First Input: Positive or negative.
- Sign of Second Input: Positive or negative.

5.4.19 Trigonometric Function

This node performs numerous common trigonometric functions. The node accepts real or complex signal and outputs the result of one of the following functions operating on the input.

- sin
- cos
- tan
- asin
- acos
- atan
- sinh
- cosh
- tanh

Parameters:

- Function: Selects one of the above functions. Default is sin(x).

5.5 Non – Linear Node Set

5.5.1 Backlash

The backlash node can be used to model *backlash* [55] effect of gears or similar drive systems. When gears are engaged (figure 5.1) a change in input shaft causes an equal change in output.

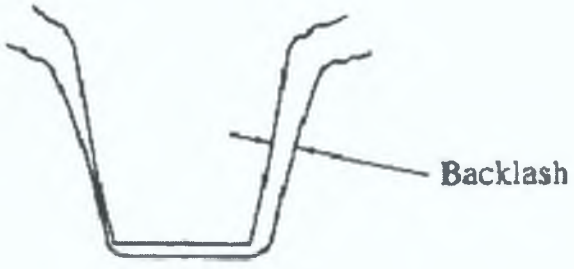


Fig 5.1: Backlash in gears

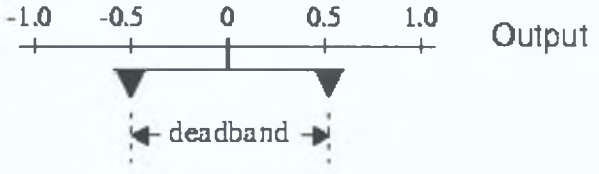


Fig 5.2: Dead-band

This node assumes this behaviour by introducing a parameter *dead-band*, which is equal to the width of the space between the gear teeth (figure 5.2). The dead-band is centred about the output. The output from the node will be one of the following

- If the input is increasing (has a positive slope), output = input – dead-band/2.
- If the input is decreasing (has a negative slope), output = input + dead-band/2
- If input slope is unchanged, output remains constant.

Parameters:

- Dead-Band Width: Width of the dead-band. Default is 1.
- Initial Output: Value of initial output. Default is 0.

5.5.2 Coulomb and Viscous Friction

This node can be used to model static (coulomb) and dynamic (viscous) friction [55]. If u is the input, the node calculates its output using the following formula

$$y = sign(u) * \{magnitude\ of\ viscous\ friction * |u| + magnitude\ of\ coulomb\ friction\}$$

where *sign(u)* is defined as follows,

$$sign(u) = \begin{cases} 1, u > 0 \\ 0, u = 0 \\ -1, u < 0 \end{cases}$$

User has to provide the values for coulomb and viscous friction on the parameter dialog box.

Parameters:

- Coulomb Friction Value: Magnitude of coulomb friction. Default is 2 1 4.
- Coefficients of Viscous Friction: Magnitude of viscous friction. Default is 1.

5.5.3 Dead Zone

The dead zone node generates zero output within a specified region. The lower limit (start of dead zone) and the upper limit (end of dead zone) of the dead zone should be specified on the dialog box. Depending on the input, output from the node will be,

- Output = 0, if the input is greater than the lower limit and less than the upper limit.

- Output = input – upper limit, if input is greater than or equal to upper limit.
- Output = input – lower limit, if input is less than or equal to lower limit.

Parameters:

- Start of Dead Zone: Lower limit of dead zone. Default is –1.
- End of Dead Zone: Upper limit of dead zone. Default is 1.

5.5.4 Manual Switch

The manual switch node enables user to select one of its two inputs on the parameter dialog box, to pass through to the output.

Parameters:

- Input Selection: Select one of two inputs. Default is first input.

5.5.5 Multi-port Switch

The multi-port switch node has two inputs and a control input. Depending on the control input, output from the node will be as follows,

- If control input is 1, output = first input.
- If control input is 2, output = second input.
- If control input less than 1 or greater than 2, ERROR message.

Parameters:

- None.

5.5.6 Quantizer

The quantizer node passes its input signal through a stair-step function. The effect is to quantize a smooth signal into a stair-step output. If u is the input and q is the quantization interval specified on the parameter dialog box, then output from the node will be,

$$y = q * round(u/q)$$

Parameters:

- Quantization Interval: The interval around which the output is quantized. Default is 0.5.

5.5.7 Rate Limiter

Rate limiter restricts the rate of change of a signal within a specified range. The node calculates the rate of change of its input signal using the following formula,

$$rate = \frac{u_t - u_{t-1}}{\Delta t}$$

where, u_t is the input at current simulation time, u_{t-1} is the input at the previous time step, and Δt is the change in time since the previous simulation time step. A rising slew rate and a falling slew rate should be specified on the parameter dialog box. If r is the rising rate and f is the falling rate, then

- If *rate* is greater than r, then output, $y(t) = \Delta t * r + y(t-1)$.
- If rate is less than f, then output, $y(t) = \Delta t * f + y(t-1)$.
- If $f \leq rate \leq r$, then output, $y(t) = u(t)$.

Parameters:

- Rising Slew Rate: Upper limit for the rate. Default is 1.
- Falling Slew Rate: Lower limit for the rate. Default is –1.

5.5.8 Relay

Depending on the input signal relay node switches its output between two constant values. A switch on value, a switch off value, switch on output and switch off output should be specified on the parameter dialog box. Let *son* is the switch on value, *sof* is the switch off value, *opn* is the switch on output value and *opf* is the switch off output value. If u is the input to the node,

- If $u > son$, then output, $y_t = opn$.

- If $u < sof$, then output, $y_t = opf$.
- If $sof \leq u \leq son$, then output, $y_t = y_{t-1}$.

Parameters:

- Switch on Point: Threshold for switch on. Default is 1.
- Switch off Point: Threshold for switch off. Default is –1.
- Output when on: Output from the node when the relay is on. Default is 1.
- Output when off: Output from the node when the relay is off. Default is 0.

5.5.9 Saturation

The saturation node can be used to restrict a signal within a specified rang. User has to define upper limit and lower limit on the parameter dialog box. If the input signal to the node is within the range specified by the lower limit and upper limit, the input signal passes through unchanged. If input is greater than the upper limit output from the node will be the upper limit. If input is less than the lower limit output will be the lower limit.

Parameters:

- Upper Limit: Upper bound of the range. Default is 1.
- Lower Limit: Lower bound of the range. Default is –1.

5.5.10 Switch

The switch node has two inputs and a control input. User has to specify a threshold value on the parameter dialog box. If the control input is greater than or equal to the threshold value then output will be equal to the first input, otherwise output will be equal to the second input.

Parameters:

- Threshold: Holds the threshold value for control input. Default is 0.

5.6 Miscellaneous Node Set

5.6.1 Counter

This node provides a simple counter. An initial value and increment length should be specified on the parameter dialog box. The node will generate output by adding the increment length with its output value at previous time step. At the start of the simulation output from the node will be the initial value provided on the parameter dialog box. For example, if initial value is 3, increment length is 0.5 and step size is 0.2, then output from this node for first few steps will be as follows

Simulation Time	0.0	0.2	0.4	0.6
Output	3	3.5	4	4.5

Parameters:

- Initial Value: Starting value for the counter. Default is 1.
- Increment Length: Counter value will be increased by this value at every simulation time step. Default is 1.

5.6.2 Cumulative Function

This node performs one of the following cumulative functions

- Cumulative Sum: At any simulation step the node will calculate its output using the following formula

$$output = \sum_{i=0}^{n} input_i$$

where n is the total number of simulation steps since simulation started, including the current step.

- Cumulative Average: At any simulation step node output will be according to the following formula,

$$output = \frac{\sum_{i=0}^{n} input_i}{n}$$

where n is the total number of simulation steps since simulation started, including the current step.

- Cumulative RMS: At any simulation time step node output will be according to the following formula,

$$output = \sqrt{\frac{\sum_{i=0}^{n} input_i^{\,2}}{n}}$$

where n is the total number of simulation steps since simulation started, including the current step.

Parameters:

- None.

5.6.3 Data Type Conversion

The data type conversion node provides the option to convert boolean type signal to numerical type signal and vice versa.

For boolean to numerical conversion, the node follows the following rules,

- If input is true, output is 1.
- If input is false, output is 0.

For numerical to boolean conversion following rules are followed,

- If input is non-zero (positive or negative), output will be true.
- If input is zero, output will be false.

Parameters:

- Convert to: Select whether output will be boolean or real. Default is real.

5.6.4 Zero Crossing

The zero crossing node can be used to detect whether a signal crosses a given crossing offset. User has to specify the offset value and the direction of crossing on the parameter dialog box. The node provides three crossing directions. Output from the node may be boolean or real. Let u_t is the current input, u_{t-1} is the input at the

previous step, and h is the crossing offset, then depending on the crossing direction output from the node will be as follows,

- Rising – if $u_t > u_{t-1}$ and $u_{t-1} \leq h \leq u_t$, then output will be true for boolean output, 1 for real output; otherwise output will be false (or 0).
- Falling – if $u_t < u_{t-1}$ and $u_t \leq h \leq u_{t-1}$, then output will be true for boolean output, 1 for real output; otherwise output will be false (or 0).
- Either – output will be true (or 1) if input crosses the zero offset either way; otherwise it will be false (or 0);

Parameters:

- Output Data Type: Selects the data type (boolean/real) of the output from the node. Default is real.
- Crossing Offset: The value that the input will cross. Default is 0.
- Crossing Direction: One of the above crossing directions. Default is Either.

5.6.5 Initial Condition

The initial condition node sets an initial value for a signal. At the simulation start time the output from the node will be the initial value given on the parameter dialog box, after then the node outputs its input without any alteration.

Parameters:

- Initial Value: Output from the node at the first simulation time step. Default is 1.

5.6.6 Matrix Concatenation

The matrix concatenation node can be used to concatenate two matrices. Concatenation can be accomplished in two ways, i) horizontally or ii) vertically. For example, let following two matrices are inputs to the node,

input 1			input 2		
2	-4	2-i	9.05	3	12-6i
-1	-3+2i	5	-12	5.66	-3i
7.3	1	2i	21	7.32	6-9i

If the matrices are concatenated horizontally then the output from the node will be,

2	-4	2-i	9.05	3	12-6i
-1	-3+2i	5	-12	5.66	-3i
7.3	1	2i	21	7.32	6-9i

If the matrices are concatenated vertically then output will be,

2	-4	2-i
-1	-3+2i	5
7.3	1	2i
9.05	3	12-6i
-12	5.66	-3i
21	7.32	6-9i

Parameters:

- Concatenation Method: Selects one of the two concatenation methods. Default is Horizontal concatenation.

5.6.7 Merge

The merge node superimposes two signals on a single signal. An initial output and the offsets for the input signals on this output should be defined on the parameter dialog box. Using these parameters and the input signals the node will find the final output. Following example will clarify the matter,

Let, initial output defined on the parameter dialog box is, -2.2 -5 7+3i 3.2 4.12 1.8

First input is, 2.92 -1.45

Second input is, -3.4 -7i

If offsets are 0 and 4 for first and second input respectively, then output will be

2.92 -1.45 7+3i 3.2 -3.4 -7i

If offsets are 3 and 1 for first and second input respectively, then output will be

-2.2 -3.4 -7i 2.92 -1.45 1.8

Parameters:

- Initial Output: Initial value of output upon which the input signals will be superimposed. Default is 1.
- Input Port Offsets: Offset of each input signal relative to the beginning of the output signal. Default is 0.

5.6.8 Mux

The mux node combines two signals into one signal. This node accepts two signals and simply appends the second signal to the first signal.

Parameters:

- None.

5.6.9 Probe

The probe node enables user to find information about a signal. Using this node user can find one of the following information

- Width – outputs the number of elements in the signal.
- Sample time – outputs the sample time of the signal.
- Complex – outputs 1 if the signal contains complex number, otherwise outputs 0.
- Dimension – outputs the number of rows and columns in the signal.

Parameters:

- Probing: Select one of the information described above. Default is Width.

5.6.10 Reshape

Dimensionality of a signal can be changed using reshape node. Selecting the appropriate output dimension on the parameter dialog box, user can change the

dimension of a matrix. The node provides four possible output dimensions. For example, let following matrix is the input to the node

2-4i	4.55
6	-3i
-12	6.91

Depending on the selected output dimension, output from the node will be,

- 1D array – 2-4i 6 -12 4.55 -3i 6.91
- Column vector

2-4i
6
-12
4.55
-3i
6.91

- Row vector – 2-4i 6 -12 4.55 -3i 6.91
- Custom – say custom dimension is 2 x 3, then output,

2-4i	-12	-3i
6	4.55	6.91

Parameters:

- Output Dimension: Specifies required output dimension. Default is 1D Array.

5.6.11 Selector

This node enables the user to select elements from a vector or from a matrix. The node has two inputs, first input can be a vector or a matrix from where elements will be selected, and second input is the indices of the elements to be selected. For example,

- Vector input – let input to the node is 3.4 5-2i 7.64 12 -9, and index input is 2 3 5, then output will be 5-2i 7.64 -9
- Matrix input – for matrix input, index input must be a matrix with 2 rows. Elements of the first row of index input will select a row (vector) from the matrix and elements of the second row of index input will select elements from the selected vector. For example, let following two matrices are input and index input respectively,

$$\begin{matrix} 3.4 & 5\text{-}2i & 7.64 \\ 12 & -9 & -4i \\ -2.56 & 13 & 11.32 \end{matrix}$$

$$\begin{matrix} 2 & 3 \\ 2 & 3 \end{matrix}$$

then output from the node will be,

$$\begin{matrix} -9 & -4i \\ 13 & 11.32 \end{matrix}$$

Parameters:

- Input Type: Specifies whether the input is a Vector or a Matrix. Default is Vector.

5.6.12 Signal Specification

This node can be used to halt the simulation if a signal does not have desired attributes. On the parameter dialog box user can choose an expected attribute (dimension, data type etc.). If the input signal does not have this attribute then the node will generate an error message and simulation will be halted; otherwise the node outputs its input signal unchanged.

Parameters:

- Dimension: Expected dimension of the input signal. Default is –1. A –1 value for dimension indicates that the node will not check for dimension.
- Data Type: Expected data type of the input signal. Default is booelan.

5.7 Sinks Node Set

5.7.1 Display

The display node accepts all types (numerical/boolean) data as input, and shows the value of its input in a separate window.

Parameters:

- None.

5.7.2 Terminator

The terminator node can be used to manage floating output lines from a node. If any model contains some floating output line, then *SimDynamic* halts the simulation. To avoid such situation terminator node has been provided.

Parameters:

- None.

5.7.3 Time Vs Input

This node enables user to view the change in a signal with respect to the simulation time. The node accepts a signal and plots the signal against the simulation time. Simulation time is shown along x-axis and input signal is shown along y-axis. If the sample time of the received signal is continuous, it produces a point - to - point plot. If the sample time is discrete then it produces a stair-step plot. If the width of the received signal is more than one (vector or matrix signal), it uses different colours in this order: green, yellow, red, orange, cyan, and magenta.

Parameters:

- None.

5.7.4 To File

The To File node accepts all types (numerical/boolean) data as input, and saves the value of its input in a text file. User has to provide a file name on the parameter dialog box. Any text editor like Notepad or MS-Word can be used to view the saved file.

Parameters:

- File Name: Name of the file to be saved. Default is None.

5.7.5 XY Graph

The XY graph node accepts two signals and plots the first signal against the second signal. First input is shown along the y-axis and second input is shown along the x-axis.

Parameters:

- None.

5.7.6 To Data File

The To data file node can be used to save simulation result for later use by *SimDynamic*. This node accepts all types (numeric/boolean) data as input, and saves the value of its input in a data file in a format that can only be used or opened by *SimDynamic*. The most important use of this node is for 3D rendering. *SimDynamic* 3D image generator reads data from files, so this node is the only way to transmit simulation result to the 3D engine.

Parameters:

- File Name: Name of the file to be saved. Default is None.

5.8 Source Node Set

5.8.1 Bool

This node generates a boolean value. User has to specify the value (true/false) on the parameter dialog box and the node generates that value at every simulation time step.

Parameters:

- Boolean Value: The output value from the node. Default is true.

5.8.2 Chirp Signal

The chirp signal node produces a sine wave whose frequency increases at linear rate with time. Initial frequency, target time, and final frequency of the signal should be specified on the parameter dialog box.

Parameters:

- Initial Frequency: Initial frequency of the signal. Default is 0.1 Hertz
- Target Time: The time at which the signal frequency will reach the target frequency. Default value is 20 seconds.
- Target Frequency: Final frequency of the signal. Default is 1 Hertz.

5.8.3 Clock

This node simply outputs the current simulation time at each simulation step. For example let step size is 0.3 and simulation start time is 5.0, then output from this node for first few steps will be 5.0, 5.3, 5.6, 5.9….. This node is useful for other nodes that need the simulation time.

Parameters:

- None.

5.8.4 Constant

This node generates a constant value. User has to specify the value (real/complex) on the parameter dialog box and the node generates that value at every simulation time step.

Parameters:

- Constant Value: The output value from the node. Default is 1.

5.8.5 Digital Clock

It outputs the simulation time at the specified sampling interval. User should set the desired sampling rate on the parameter dialog box. For example, if the simulation step size is 0.2, then output from the clock node will be 0, 0.2, 0.4, 0.6..... But with the same step size and 0.4 sample time, output from the digital clock node will be 0, 0.4, 0.8.....

Parameters:

- Sample Time: The time interval at which output will be updated. Default is 1.

5.8.6 Discrete Pulse Generator

It generates a series of pulses at regular intervals. Amplitude, sample time, period, pulse width and phase delay should be specified on the parameter dialog box. Period, pulse width and phase delay should be expressed in terms of sample time, i.e. period is the number of sample periods the pulse is high and low, pulse width is the number of sample periods the pulse is high, phase delay is the number of sample periods before the pulse starts.

Parameters:

- Amplitude: Amplitude of the pulse. Default is 1.
- Period: Period of the pulse in terms of sample interval. Default is 2.
- Pulse Width: Width of the pulse in terms of sample interval. Default is 1.

- Phase Delay: Delay before each pulse is generated. Expressed in number of samples. Default is 0.
- Sample Time: The time interval at which output will be updated. Default is 1.

5.8.7 From Data File

The from data file node reads data from a file that has been save by the *to data file* node. This node provides the mechanism of using previously generated simulation result. The node compares step size saved in the file with the current simulation step size. If they do not match it will generate error message.

Parameters:

- File Name: Name of the file that contains the data. Default is None.

5.8.8 Ground

SimDynamic halts simulation if any node contains a floating input line. Ground node can be used to connect such floating input line. This node produces a 0 all the time.

Parameters:

- None.

5.8.9 Pulse Generator

It generates a continuous train of pulses. Amplitude, period, duty cycle and start time of the pulse should be specified on the parameter dialog box. Period and start time should be expressed in second, and duty cycle should be expressed as a percentage of period.

Parameters:

- Amplitude: Amplitude of the pulse. Default is 1.
- Period: Period of the pulse, expressed in seconds. Default is 1.
- Duty Cycle: Percentage of the pulse period that the signal is high. Default is 50%.

- Start Time: Delay before each pulse is generated. Expressed in seconds. Default is 0.

5.8.10 Ramp

The ramp node can be used to generate a signal that starts at a specified time and changes by a given rate [56]. The rate of change (slope), start time and initial output should be set on the parameter dialog box. The node continues to output the initial output given on the dialog box as long as simulation time does not exceed start time, then it starts to produce the ramp signal.

Parameters:

- Slope: The rate of change of the generated signal. Default is 1.
- Start Time: Time at which the node will start generating the signal. Default is 0.
- Initial Output: Output from the node before the ramp signal starts. Default is 0.

5.8.11 Random Number

This node generates normally distributed random variates. Mean, variance of the distribution and an initial seed should be specified on the parameter dialog box. This node uses java's built in random number generator to generate two random numbers R_1 and R_2, using the initial seed. From those two random numbers, random variates are generated through following steps [57]

- Generate two independent standard normal variates

$$Z_1 = (-2lnR_1)^{1/2} cos(2\pi R_2)$$
$$Z_2 = (-2lnR_1)^{1/2} sin(2\pi R_2)$$

- If μ is the mean and σ is the variance, then normal variate X_i can be found from the formula

$$X_i = \mu + \sigma Z_i$$

Parameters:

- Mean: Mean of the normal distribution. Default is 0.
- Variance: Variance of the distribution. Default is 1.
- Initial Seed: Starting seed for the random number generator. Default is 0.
- Sample Time: The time interval at which output will be updated. Default is 1.

5.8.12 Repeating Sequence

The repeating sequence node generates a periodic signal defined by the user. The node has two parameters time points and output values. User can define the shape of the signal by specifying these parameters on the dialog box. For example, default values for time points and output values are (0 2) and (0 2) respectively. This means, the node will repeat a signal every 2 seconds from the start of the simulation time that has maximum amplitude of 2. The node uses linear interpolation to calculate the value of the signal between the specified time points. So for these parameter values, and simulation step size 0.2, output from the node, for first 2.2 seconds, will be

Simulation time	0	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0	2.2
Output	0	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	0	0.2

Again if the time points and output values are 1 2 3 and 5 8 13 respectively and simulation step size is 0.2, then output will be

Simulation time	0	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0	2.2
Output	5	5.6	6.2	6.8	7.4	8	9	10	11	12	5	5.6

Parameters:

- Time Values: Time values that determine the period of the generated signal. Default is 0 2.
- Output Values: Values that determine the shape (amplitude) of the generated signal. Default is 0 2.

5.8.13 Signal Generator

This can produce one of the three different waveforms: sine wave, square wave, and sawtooth wave. User has to specified amplitude and frequency on the parameter dialog box for the signal.

Parameters:

- Wave Form: Selects the wave form (sine/square/sawtooth). Default is sine wave.
- Amplitude: Signal amplitude. Default is 1.
- Frequency: Frequency of the generated signal. Default is 1 Hertz.

5.8.14 Sine Wave

It generates a sine wave using the following formula

$$y = amplitude * sin(frequency * simulation\ time + phase\ angle)$$

The user has to specify amplitude, frequency and phase angle on the parameter dialog box. This node can be used to generate a sine wave either in continuous or discrete mode. If the sample time on the parameter dialog box is left 0, then a continuous wave will be generated. For any positive value for sample time will generate a wave at specified interval.

Parameters:

- Amplitude: Amplitude of the generated signal. Default is 1.
- Frequency: Frequency of the generated signal. Default is 1 rad/sec.
- Phase: Phase shift of the signal. Default is 0 rad.
- Sample Time: The time interval at which output will be updated. Default is 0, which means it will generate a continuous sine wave.

5.8.15 Step

It provides a step [56] between two defined values at a particular time. Step time, initial output and final output should be specified on the parameter dialog box. The

node continues to produce the initial output until the simulation time reaches the step time. From this point the node continues to produce the final value provided on the parameter dialog box.

Parameters:

- Step Time: Time at which output will switch to final value. Default is 1 sec.
- Initial Value: Output from the node before it switches to the final value. Default is 0.
- Final Value: Output from the node when simulation time reaches the step time. Default is 1.
- Sample Time: The time interval at which output will be updated. Default is 0.

5.8.16 Uniform Random Number

This node generates uniformly distributed random variates. Minimum value, maximum value of the distribution and an initial seed should be specified on the parameter dialog box. This node uses java's built in random number generator to generate a random numbers R, using the initial seed. If α and β are the minimum and maximum values respectively, then uniform variate X can be found from following formula [57]

$$X = \alpha + (\beta - \alpha)R$$

Parameters:

- Minimum: Minimum value of the distribution. Default is -1.
- Maximum: Maximum of the distribution. Default is 1.
- Initial Seed: Starting seed for the random number generator. Default is 0.
- Sample Time: The time interval at which output will be updated. Default is 1.

Chapter Six

6 Application of *SimDynamic*

SimDynamic can be used to simulate the behaviour of a wide range of real-world systems, including electrical, mechanical, and thermodynamic systems. In this chapter a few examples are presented. Since most dynamic systems are expressed in terms of differential equations, the chapter starts with an example of how to solve a differential equation of second order using *SimDynamic.* More complex and real world systems are presented in the subsequent sections. First example provides all the steps necessary to build a *SimDynamic* model. Similar steps were followed in the subsequent examples, but for brevity the model building steps are omitted from the latter examples. It is recommended that reader must go through chapters 3 and 5 before reading this chapter.

6.1 Solving Ordinary Differential Equation

Consider the following second order differential equation where coefficients are chosen arbitrarily,

$$\frac{d^2y(t)}{dt^2} + 0.5\frac{dy}{dt} + 11y = 9\,|\sin(3.77t)| \qquad 6.1$$

This can be written as two first order differential equations, if two new variables *x1(t)* and *x2(t)* are introduced. Let *x1(t) = y(t)*, then the above equation reduces to the following two first order equations,

$$\frac{dx1(t)}{dt} = x2(t) \qquad 6.2$$

$$\frac{dx2(t)}{dt} = -(0.5x2(t) + 11x1(t)) + 9\,|\sin(3.77t)| \qquad 6.3$$

So, the solution of this equation can be found by using two integrator nodes to integrate the first order derivatives.

6.1.1 Building the Model

Step 1: Add two *Integrator* nodes to the empty model sheet from the Continuous node set. Rename (using the node menu) the nodes as x1 and x2 respectively. Connect the output port of x2 to the input port of x1 (figure 6.1). The input to the left most *Integrator* is the derivative of x2 and its output is the variable x2. The input to the rightmost *Integrator* is the derivative of x1 (i.e. x2) and its output will be the variable x1 (i.e. y, which is the final output).

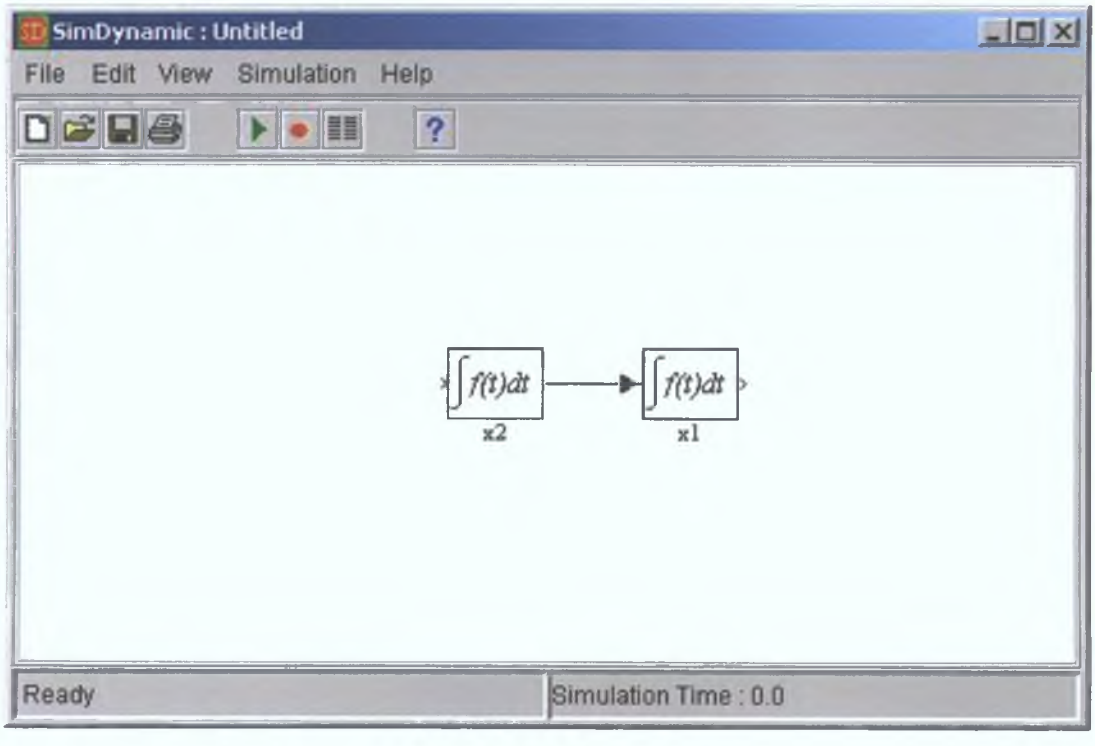


Fig 6.1: Step 1 for ODE model

Step 2: Add two *Gain* nodes from the Math and Logic node set and rotate them (using the rotate option of the node menu). Rename the *Gain* nodes as 0.5 and 11. Set *Gain* node parameters to 0.5 and 11 respectively on the parameter dialog box. Draw a branch line from the output line of x2 *Integrator* and connect it to the input port of 0.5 *Gain* node. Draw a branch line from the output line of x1 *Integrator* and connect it to the input port of 11 *Gain* node.

Step 3: Add two *Sum* nodes from the Math and Logic node set and rotate them. Rename the *Sum* nodes as sum1 and sum2. Connect the outputs of both *Gain* nodes to the inputs of sum1 node. Connect the output of sum1 node to one input of sum2 node. Change the sign of this input to minus. After steps 2 and 3 the model will look like as

in figure 6.2. Now it has almost the complete representation of the differential equation except the input (i.e. sine term in the equation).

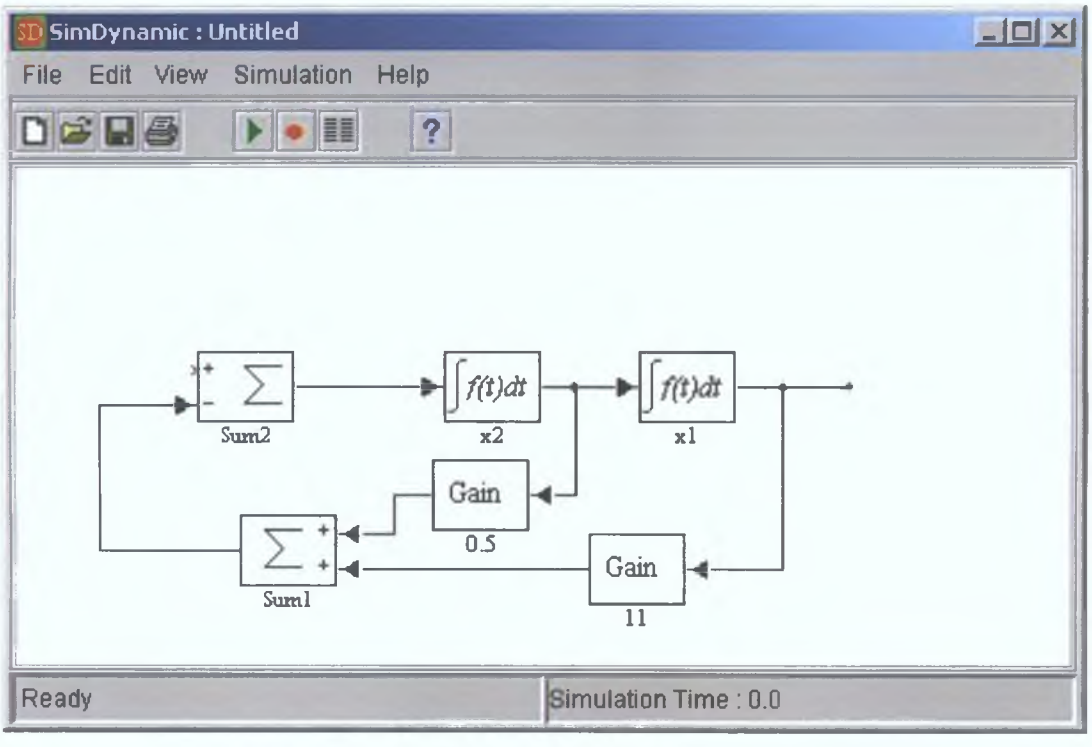


Fig 6.2: Step 2 and 3 for ODE model

Step 4: Add a *Clock* node from the source node set, rotate it and rename it as t. Add a *Gain* node from the Math and Logic node set, rotate it, rename it as 3.77 and set the gain parameter to 3.77 on the parameter dialog box. Connect the output of the clock node to the input of the *Gain* node. Add a *Trigonometric Function* node from the Math and Logic node set and rotate it. Connect the output of the 3.77 *Gain* node to the input of the *Trigonometric Function* node. Select the sine function from the function list on the *Trigonometric Function* node parameter dialog box. So the input to the *Trigonometric Function* node is now 3.77t and output from this node will be sin(3.77t). Add an *Absolute* node from the Math and Logic node set and rotate it. Connect the output of the *Trigonometric Function* node to the input of the *Absolute* node. Add another *Gain* node from the Math and Logic node set, rotate it, rename it as 9 and set the gain parameter to 9 on the parameter dialog box. Connect the output of the *Absolute* node to the input of the 9 *Gain* node. And connect the output of the 9 *Gain* node to the first input of the sum2 node. The model is complete now. To view output add a *Time vs Input* node from the Sink node set and draw branch line from the

output line of x1 *Integrator* and connect it to the input of the *Time vs Input* node. The completed model will look like as figure 6.3.

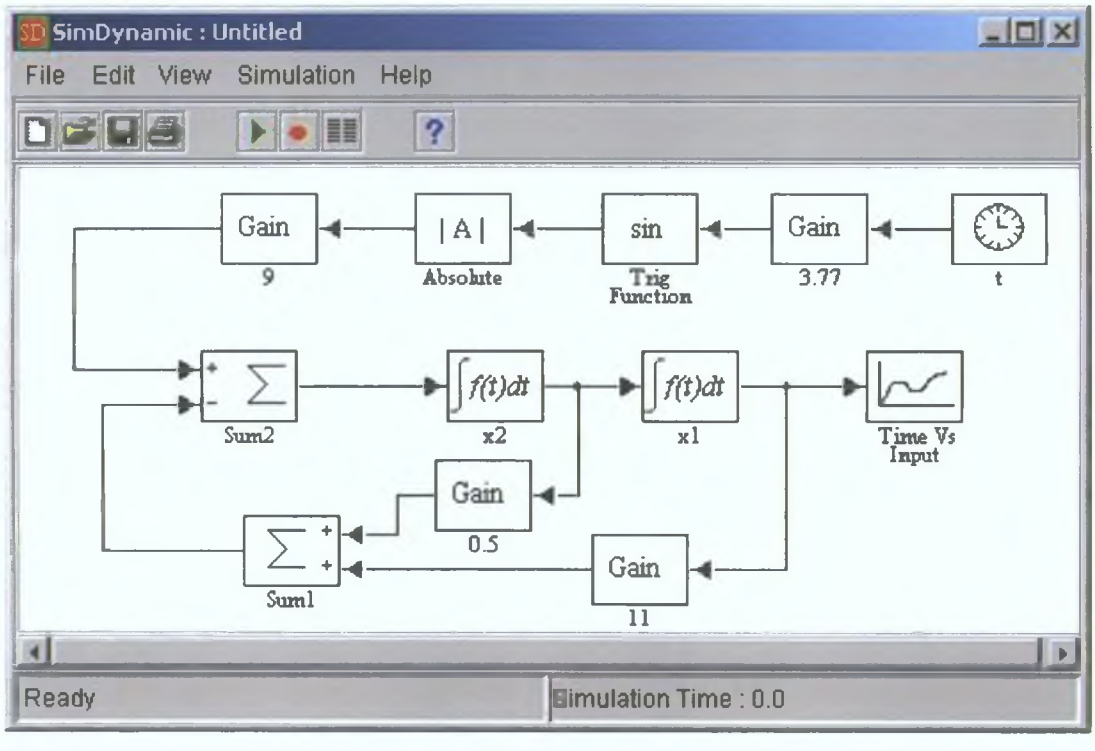


Fig 6.3: Completed ODE model

6.1.2 Running Simulation and Viewing Result

In this example simulation will be carried out for 10 seconds. So click the parameter button from the toolbar (or select parameter from simulation menu) and set simulation start time to 0, and stop time to 10 on the simulation parameter dialog box. Choose RK_DP integrator on the parameter dialog box, set simulation step size to 0.2 and click ok. Now click the run button from the toolbar (or choose run from simulation menu). When simulation is finished ("Ready" appears on the status bar), double click on the *Time vs Input* node. A new window like figure 6.4 will show the simulation result. This window will show graphical result of the simulation. In this graph simulation time is shown along x-axis and the output is shown along y-axis. To view the numerical result, add a *Display* node from the Sink node set. Draw a branch line from the output line of the x1 *Integrator* and connect it to the input of the *Display* node. Run the simulation again. When simulation is completed double click on the *Display* node. Another window like figure 6.5 will show the numerical result of the simulation.

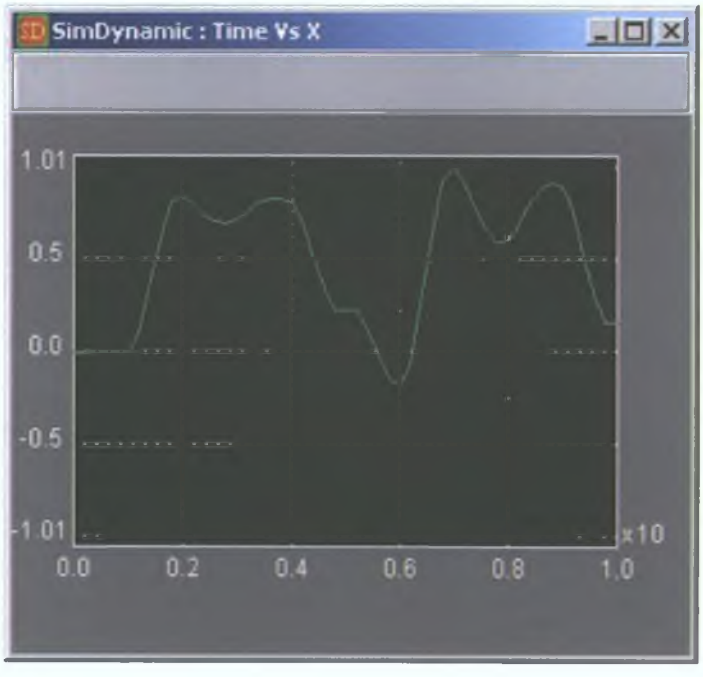


Fig 6.4: Graphical result for ODE model.

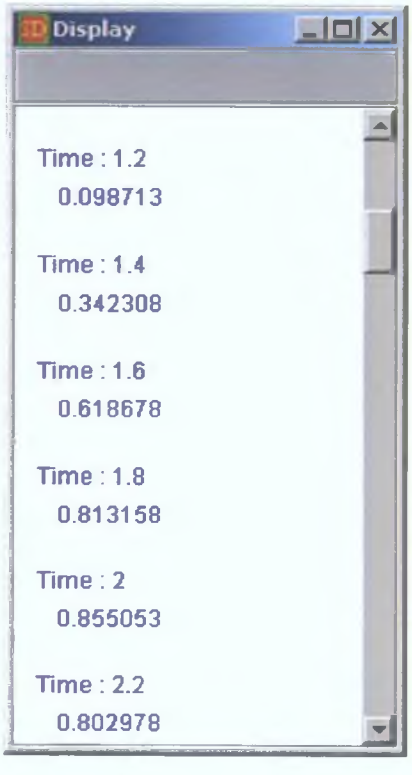


Fig 6.5: Numerical result for ODE model

6.2 Simple Damped Pendulum

Consider the simple damped pendulum shown in figure 6.6. It is assumed that the mass is m kg, length of the weightless rod is L metre, and there is a frictional torque at the pivot point that is related to the angular velocity through the coefficient c N-m-s/rad.

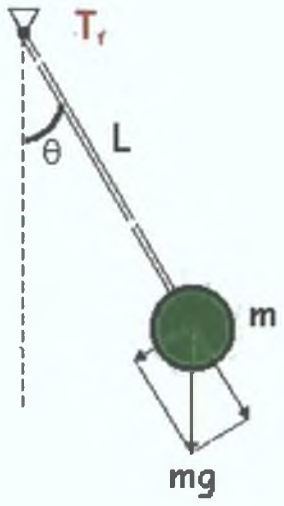


Fig 6.6: A simple pendulum

The equation of motion of this damped pendulum is,

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L}\sin\theta - \frac{c}{mL^2}\frac{d\theta}{dt} \qquad 6.4$$

6.2.1 Building the Pendulum Model

From equation 6.4 following (figure 6.7) model is built using *SimDynamic*. In this model, input to the right most *Integrator* node (renamed as theta) is $d\theta/dt$, and output from this node (green signal line) is θ, on which *Trigonometric Function* node is applied to have $sin\theta$. Two *Product* nodes (labelled as g/L and $g/L*sin$ respectively) are used to perform the multiplication $g/L*sin\ \theta$. Similarly the left most *Integrator* node (renamed as d(0)/dt) outputs (red signal line) $d\theta/dt$, and this is multiplied by c/mL^2 using three *Product* nodes. Four *Constant* nodes are used to supply the values for mass (labelled as Mass), length (labelled as Length), torque (labelled as torque), and acceleration due to gravity (labelled as g). These four nodes in this model enable

the user to change respective parameters and immediately see what happens, for "what if" exploration.

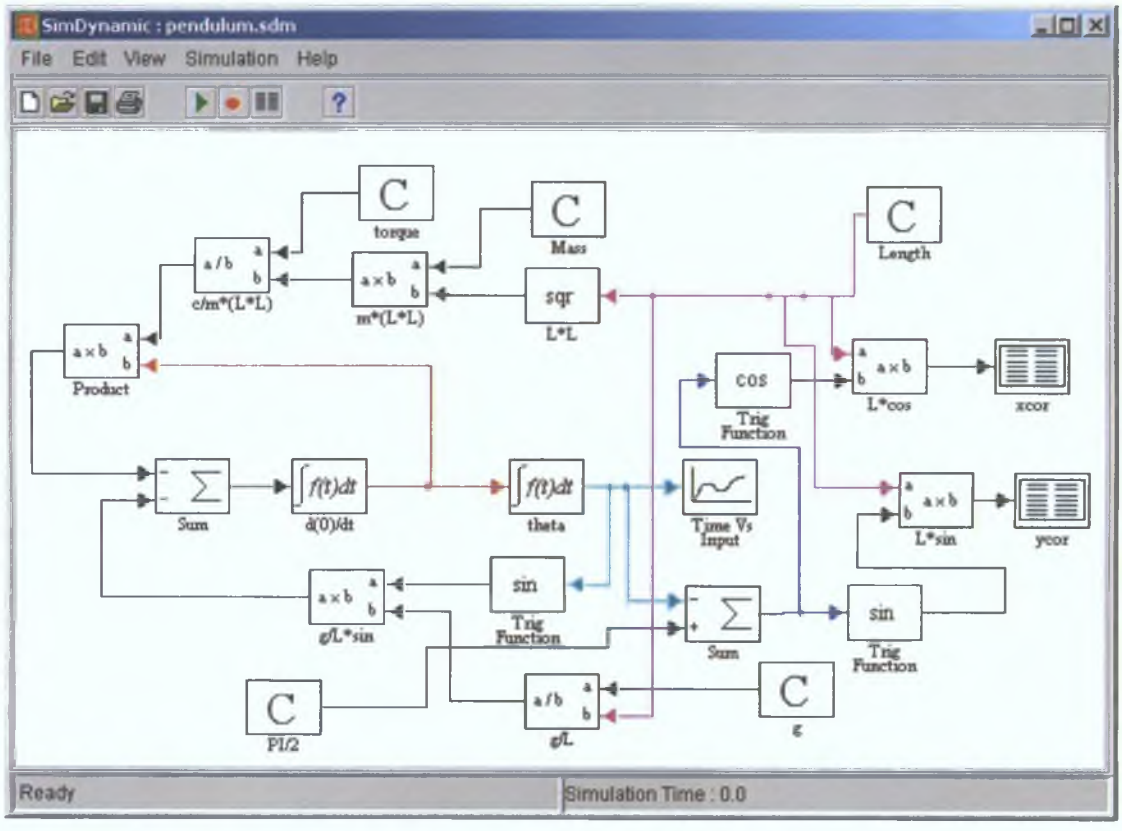


Fig 6.7: Pendulum model in SimDynamic

To view the graphical output from the simulation a *Time vs Input* node is used, and input to this node is θ. So the angular displacement of the pendulum will be plotted against the simulation time. Rest of the nodes in the model are used to calculate the position (x and y co-ordinate) of the bob at each simulation time step. These values are used later for 3D rendering. Two *To Dat File* nodes (labelled as xcor and ycor) are used to save the x co-ordinate and y co-ordinate values respectively in data files that will be used for the 3D rendering.

6.2.2 Running Simulation and Viewing Result

Simulation was carried out for 20 seconds using RK_DP integrator, with an initial displacement *PI/3* and step size 0.2. Following parameter values were used,

- Mass, m = 1 kg
- Length, L = 1 meter
- Torque Coefficient, c = 0.1 N-m-s/rad
- Acceleration due to gravity, g = 9.8 m/s^2

Figure 6.8 shows the graphical result followed by the numerical (for first 10 seconds) result in table 6.1.

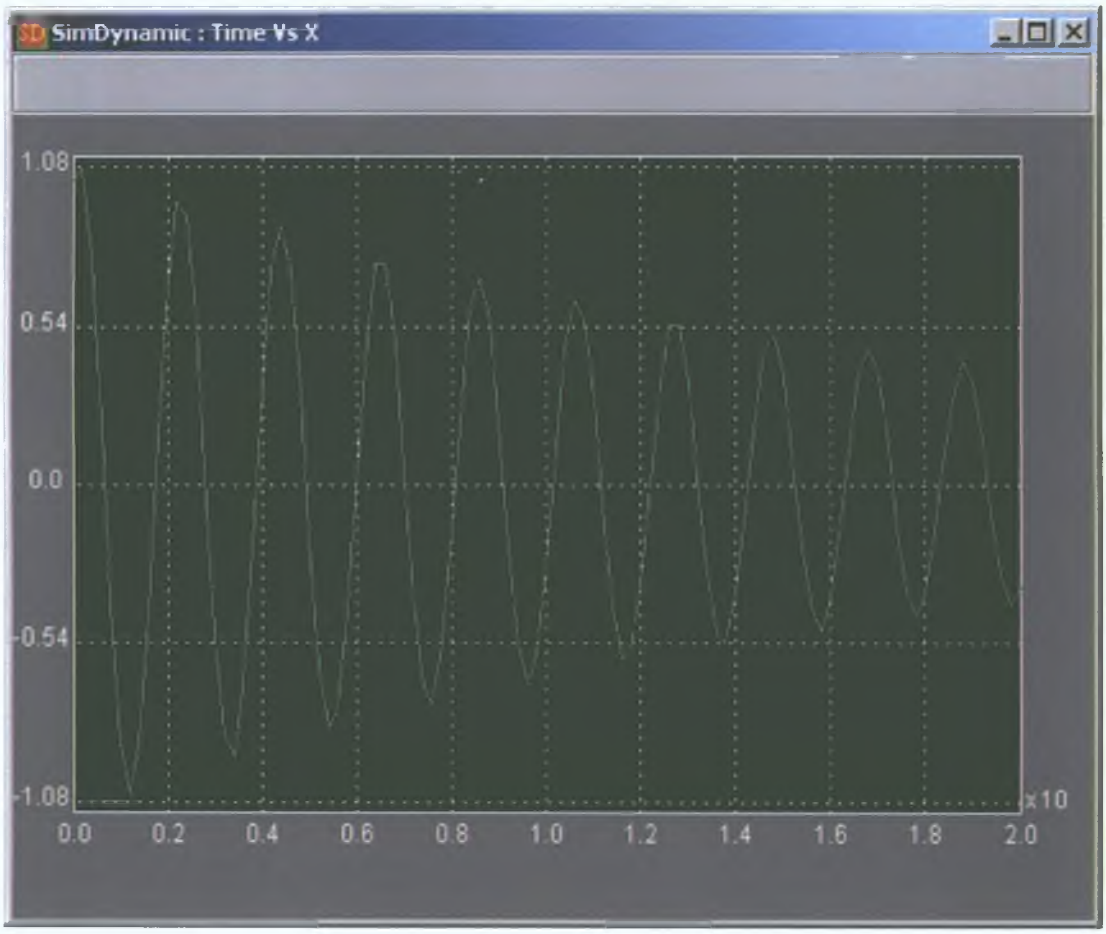


Fig 6.8: Graphical result for pendulum model (rod length 1 meter)
simulation time along x-axis, θ along y-axis

Table 6.1: Numerical result for the Pendulum model (rod length 1 meter)

Time	Output	Time	Output	Time	Output	Time	Output	Time	Output
0.0	1.04719	2.2	0.96670	4.2	0.73782	6.2	0.49669	8.2	0.28935
0.2	1.08213	2.4	0.92649	4.4	0.88519	6.4	0.76006	8.4	0.61154
0.4	0.77975	2.6	0.58400	4.6	0.73560	6.6	0.75761	8.6	0.71130
0.6	0.22108	2.8	0.04356	4.8	0.33655	6.8	0.49530	8.8	0.56206
0.8	-0.40648	3.0	-0.50209	5.0	-0.17666	7.0	0.06085	9.0	0.21571
1.0	-0.87646	3.2	-0.85855	5.2	-0.61489	7.2	-0.38744	9.2	-0.20322
1.2	-1.04774	3.4	-0.92165	5.4	-0.82847	7.4	-0.68627	9.4	-0.53890
1.4	-0.88515	3.6	-0.68083	5.6	-0.75837	7.6	-0.74013	9.6	-0.67523
1.6	-0.43412	3.8	-0.20929	5.8	-0.43005	7.8	-0.53770	9.8	-0.57249
1.8	0.16275	4.0	0.32930	6.0	0.046653	8.0	-0.14752	10.0	-0.26804
2.0	0.68849								

For a "what if" exploration, pendulum rod length was changed to 0.5 meter and simulation was carried out for 10 seconds. The graphical and numerical (first 10

seconds) results for this simulation run are shown in figure 6.9 and table 6.2 respectively.

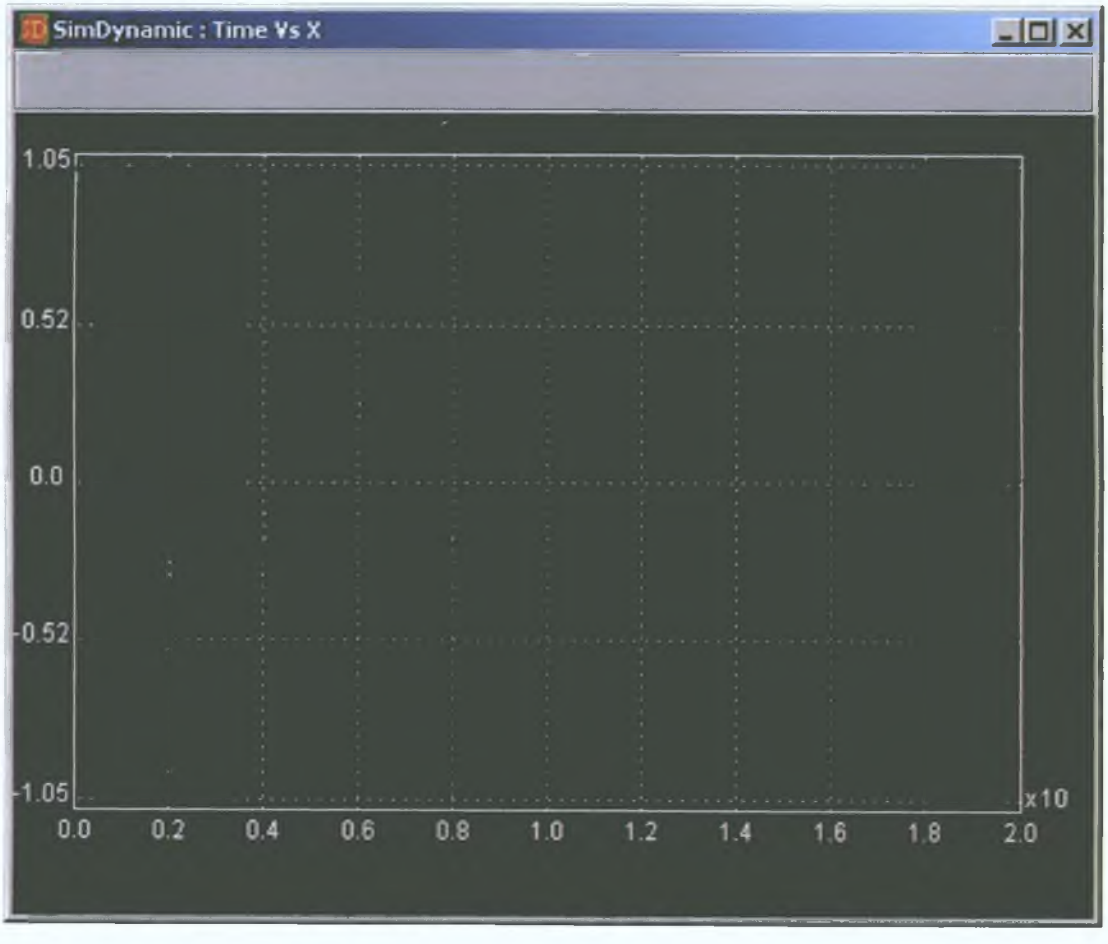


Fig 6.9: Graphical result for pendulum model (rod length 0.5 meter)
simulation time along x-axis, θ along y-axis

Table 6.2: Numerical result for pendulum model (rod length θ.5 meter)

Time	Output	Time	Output	Time	Output	Time	Output	Time	Output
0.0	1.04719	2.2	-0.61732	4.2	0.15672	6.2	0.09345	8.2	-0.16330
0.2	0.91565	2.4	-0.61236	4.4	0.41391	6.4	-0.15891	8.4	-0.00794
0.4	0.23019	2.6	-0.19949	4.6	0.36628	6.6	-0.28040	8.6	0.14103
0.6	-0.55600	2.8	0.31889	4.8	0.06929	6.8	-0.19679	8.8	0.17935
0.8	-0.91581	3.0	0.57817	5.0	-0.25325	7.0	0.01823	9.0	0.08872
1.0	-0.67531	3.2	0.42820	5.2	-0.37452	7.2	0.20386	9.2	-0.05739
1.2	-0.01451	3.4	-0.00256	5.4	-0.22725	7.4	0.23228	9.4	-0.15182
1.4	0.60502	3.6	-0.3983	5.6	0.06721	7.6	0.09585	9.6	-0.13218
1.6	0.77629	3.8	-0.48989	5.8	0.29185	7.8	-0.09746	9.8	-0.02104
1.8	0.43198	4.0	-0.24031	6.0	0.29584	8.0	-0.20733	10.0	0.09637
2.0	-0.17691								

6.2.3 3D Animation for Pendulum Model

This section describes all the steps needed to build a 3D pendulum model using *SimDynamic*. This will provide a general idea of how to use *SimDynamic* to construct a 3D model. These 3D model-building steps have been omitted for other examples described later.

To start the 3D model editor, select 3D-Animation from simulation menu. In the pendulum model three objects will be used. A small box will be used as the hook of the pendulum, a cylinder will be used as the rod, and a sphere will be used as the bob.

Step 1: Click the *box* button. On the box parameter dialog box set box width, height and depth to 0.04 each. Since the box will remain static during 3D rendering, no data files will be added to box. Leave other fields with default values. Click the *Add* button on the dialog box.

Step 2: Click the *cylinder* button. On the cylinder parameter dialog box, set cylinder radius to 0.02, height to 1, and cylinder centre to (0.0, -0.5, 0.0). During the 3D animation one end of the cylinder will remain fixed and other end will move with the bob. So enter the name of the data files for x-coordinate and y-coordinate (that are saved using the xcor and ycor nodes in the pendulum model above) for the top end of the cylinder. Leave other fields with default values. Click the *Add* button on the dialog box.

Step 3: Click the *sphere* button. On the parameter dialog box set the sphere radius to 0.2. Enter the data file names for x-coordinate and y-coordinate (that are saved using the xcor and ycor nodes in the pendulum model) for the centre of the sphere. Leave the z-coordinate filed with default value. Click the *Add* button on the dialog box.

After all three steps, the 3D model editor will look like as in figure 6.10,

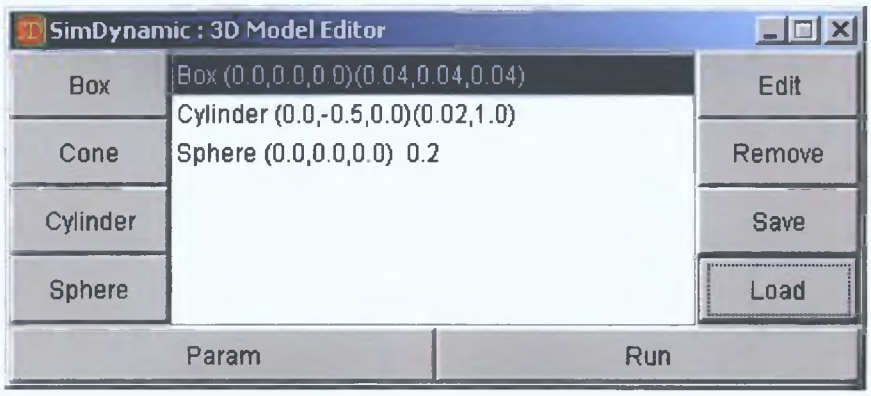


Fig 6.10: Pendulum 3D model

To start 3D animation click on the *run* button. Figure 6.11 shows four different snap shots of the 3D animation for the pendulum model generated by *SimDynamic*

Fig 6.11 (a)

Fig 6.11 (b)

Fig 6.11 (c)

Fig 6.11 (d)

Fig 6.11: 3D animation for pendulum model

6.3 Bouncing Ball

A rubber ball is thrown from a height of y_0 metre with the velocity v_0 meter/second (figure 6.12). Elasticity of the ball is ε ($0 \leq \varepsilon \leq 1$).

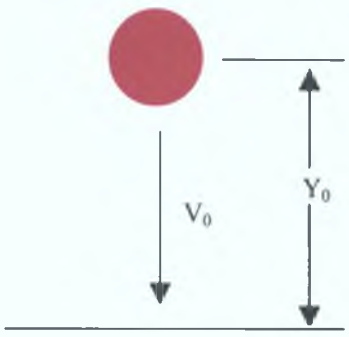


Fig 6.12: A ball is thrown downward with velocity v_0

Considering the downward direction as the positive direction, velocity v and height (from the floor) y of the ball at time t can be found from the equations,

$$v = v_0 + gt \qquad 6.5$$

$$y = y_0 - (vt + \frac{1}{2} gt^2) \qquad 6.6$$

where g is the acceleration due to gravity. At every y position if the ball is above the floor (i.e. y is greater than or equal to zero) then it will continue to fall. If the calculated y value is negative (indicating the ball hits the floor), then velocity is recalculated using the following formula

$$v_r = -\varepsilon\, v_b \qquad 6.7$$

where v_r is the rebound velocity, v_b is the velocity with which the ball hits the floor and minus sign is used since the ball will change direction.

6.3.1 Building the Bouncing Ball Model

From the above equations and assumptions the following (figure 6.13) model was constructed using *SimDynamic*. This model will calculate v and y at every simulation time step. Inputs to the *Sum1* node are v_0 and gt, so the output from this node is the

velocity v ($v = v_0 + gt$) at time step t. Inputs to the *Sum2* nodes are vt and $1/2gt^2$, so the output from this node is the distance from the throwing point at time step t. *Sum3* node calculates the position (i.e. y) of the ball

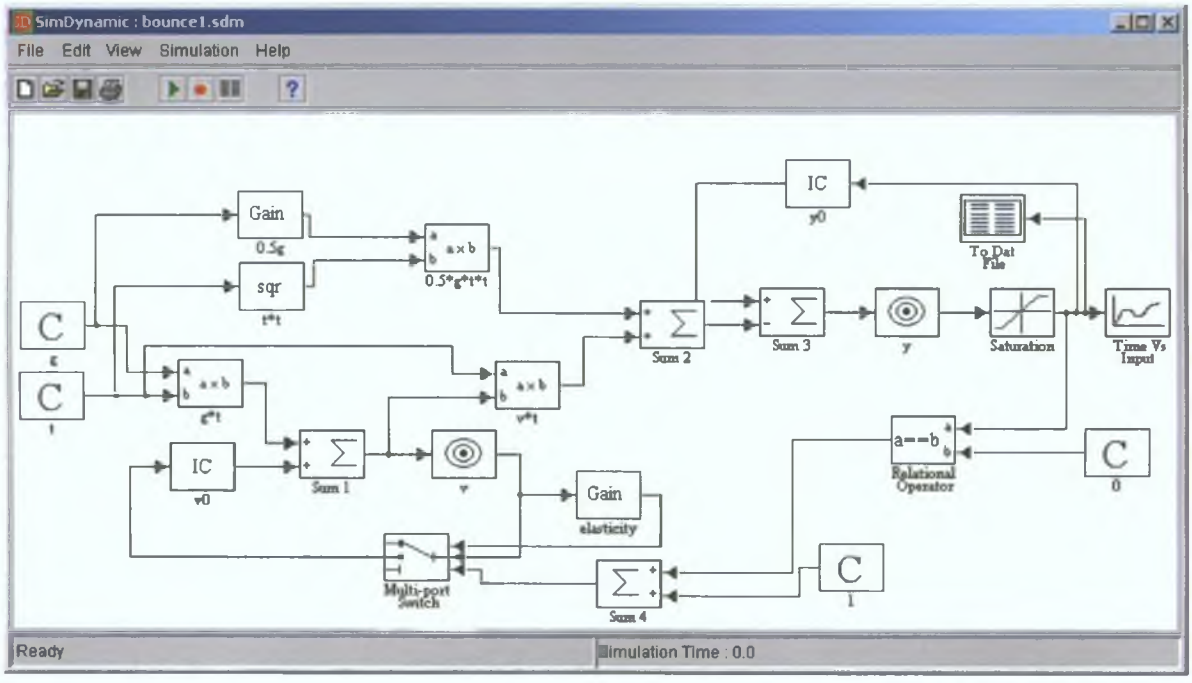


Fig 6.13: Bouncing ball model in SimDynamic

from the ground at time step t. *Saturation* node is used to limit the value of y within the range 0 to 15. *Time vs Input* node is used to draw the position of the ball against the simulation time. The position of the ball is always compared with constant value 0 using a *Relational Operator* node. If the ball position is 0 then velocity will be calculated using equation 6.7, otherwise it will be calculated using equation 6.5. For this switching between equations a *Multi-Port Switch* node has been used. Control input to the switch node is the output from R*elational Operator* (1 has been added to the output for indexing purpose) node. Two *Initial Condition* nodes (labelled as v_0 and y_0) are used to provide initial velocity and initial position of the ball. A *Gain* node (labelled as elasticity) is used for the elasticity of the ball. To save the ball position at every simulation time step (for 3D animation) a *To Dat File* node is used.

6.3.2 Running Simulation and Viewing Result

Simulation was carried out for 20 seconds with step size 0.005. Following parameter values were used,

- Initial Velocity, $V_0 = 10$ m/s
- Initial Position, $Y_0 = 15$ meter
- Elasticity, $\varepsilon = 0.8$

Graphical result for this simulation run is shown in figure 6.14,

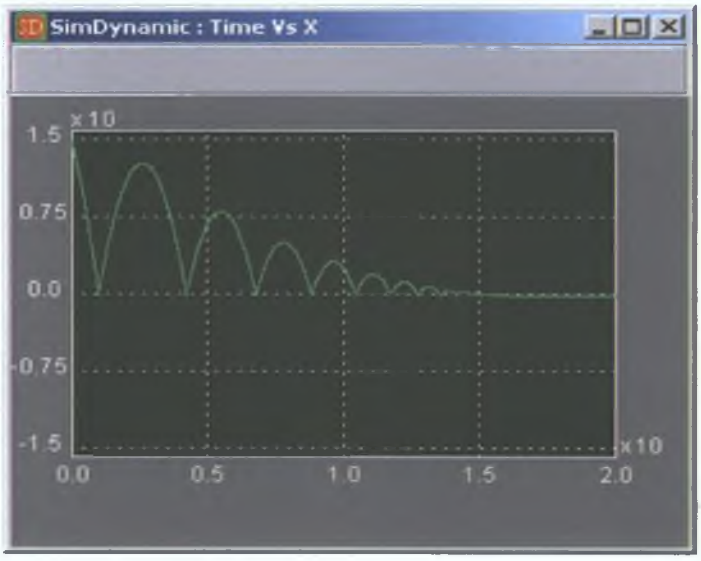


Fig 6.14: Graphical result for bouncing ball model ($\varepsilon = 0.8$)
simulation time along x-axis, ball position along y-axis

The elasticity was changed to 1.0 and simulation was carried out for 20 seconds. Graphical result for this simulation run is shown in figure 6.15,

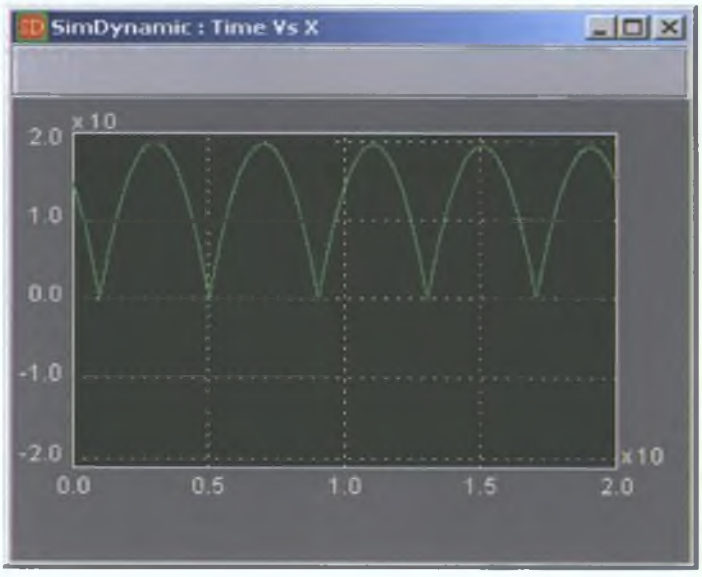


Fig 6.15: Graphical result for bouncing ball model ($\varepsilon = 1.0$)
simulation time along x-axis, ball position along y-axis

6.3.3 3D Animation for Bouncing Ball Model

A sphere is used for the bouncing ball 3D model, and the data file saved in the bouncing ball model is attached to the sphere to move it along y-axis. Figure 6.16 shows four different snap shots of the 3D animation for bouncing ball model generated by *SimDynamic*

Fig 6.16 (a)

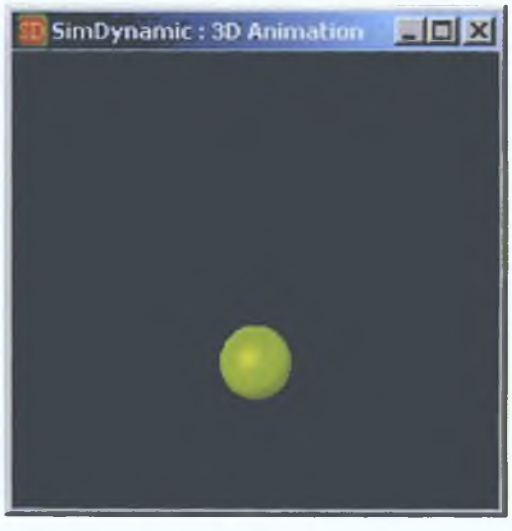


Fig 6.16 (b)

Fig 6.16 (c)

Fig 6.16 (d)

Fig 6.16: 3D animation for bouncing ball model

6.4 Bus Suspension

A bus suspension model can be simplified (considering just one wheel of the bus) to a one-dimensional spring-damper system. A diagram of this system is shown in figure 6.17,

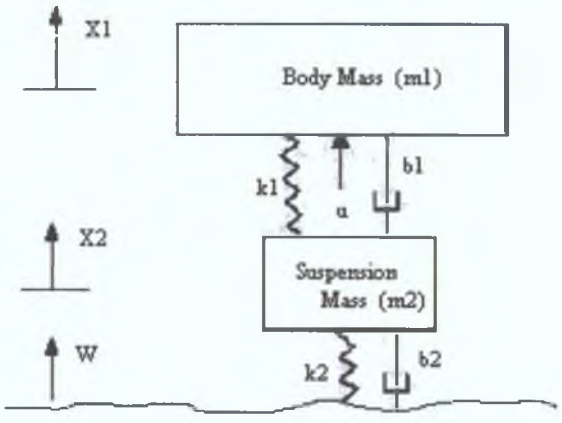


Fig 6.17: Bus suspension system (1/4 bus)

where,

- m1 is body mass
- m2 is suspension mass
- k1 is spring constant of suspension system
- k2 is spring constant of wheel and tire
- b1 is damping constant of the suspension system
- b2 is the damping constant of the wheel and tire
- w is the disturbance from the road
- u is the control force.

When the bus is experiencing any road disturbance, the distance (x1 – w) will be very difficult to measure, and the deformation of the tire (x2 – w) is negligible. So the distance (x1 – x2) will be output from the model. Newton's law for each of these masses can be expressed as:

$$\sum_{1} F = m1 \frac{d^2 x1}{dt^2} \qquad 6.8$$

$$\sum_2 F = m2\frac{d^2x2}{dt^2} \qquad 6.9$$

There are three forces acting on *m1* (one damper, one spring, and control force u) and five forces are acting on *m2* (two dampers, two springs and control force u).

The force from spring 1 is equal to *k1*(x1 – x2)*, and force from damper 1 is equal to *b1*(v1 – v2),* where *v1 = dx1/dt* and *v2 = dx2/dt*. Force from spring 2 is equal to *k2*(w – x2)*, and force from damper 2 is equal to *b2*(v2 – dw/dt)*. So,

$$\sum_1 F = u - k1*(x1 - x2) - b1*(v1 - v2) \qquad 6.10$$

$$\sum_2 F = k1*(x1 - x2) + k2*(w - x2) + b1*(v1 - v2) + b2*(\frac{dw}{dt} - v2) - u \qquad 6.11$$

6.4.2 Building the Bus Suspension Model

Using equations 6.8 – 6.11 following model (fig 6.18) was built. The road disturbance (w) and control force (u) are simulated using two *step* nodes labelled as W and U respectively.

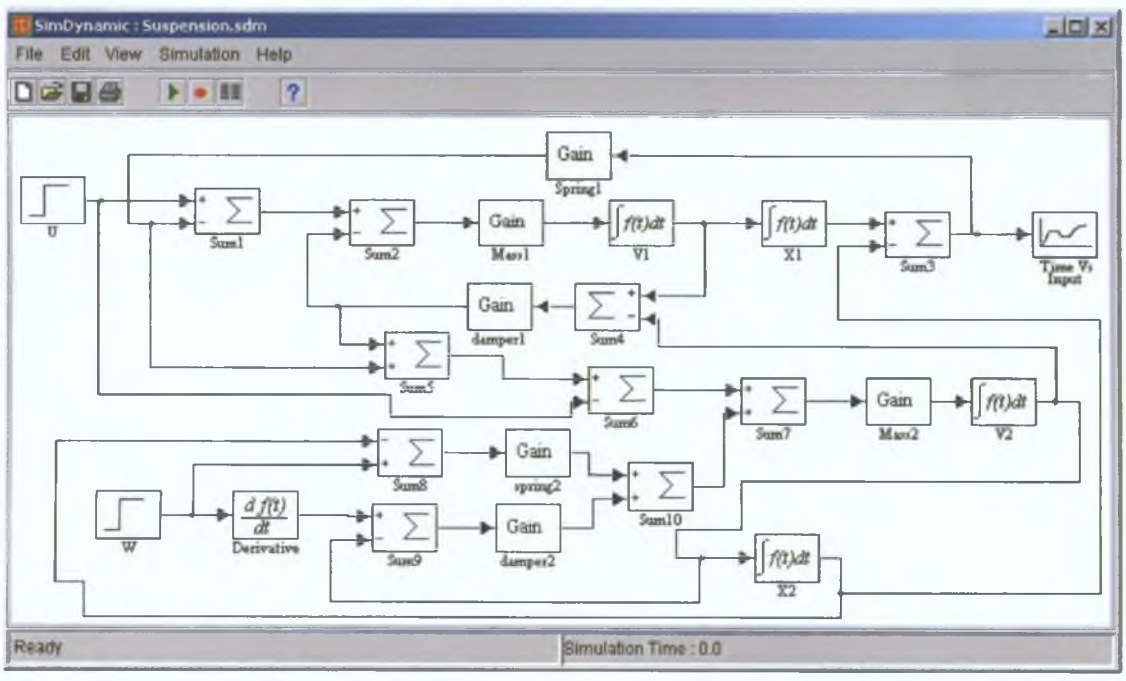


Fig 6.18: Bus suspension model (open loop) in SimDynamic

The output from *Sum3* node is (*x1 – x2*), which is multiplied by *k1* (*Gain* node labelled as spring1). Output from *Sum4* node is (*v1 – v2*), which is multiplied by *b1* (*Gain* node labelled as damper1). *Sum1* and *Sum2* nodes are used to sum up all three forces act upon mass1. The resultant force (output from *Sum2* node) is multiplied *1/m1* (*Gain* node labelled as Mass1) and fed to an *Integrator* node (labelled as V1). The output from this *Integrator* node V1, is fed to a second *Integrator* node (labelled as X1). Similar description can be made for the second mass-spring system. A *Time vs Input* node is used to view the graphical output from the simulation.

This model will only provide open-loop response from the system. To construct a closed loop model (figure 6.19) feedback controller was designed feeding back the following five states (see appendix C for derivation): [*x1 dx1/dt y1=x1-x2 dy1/dt* $\int y1$].

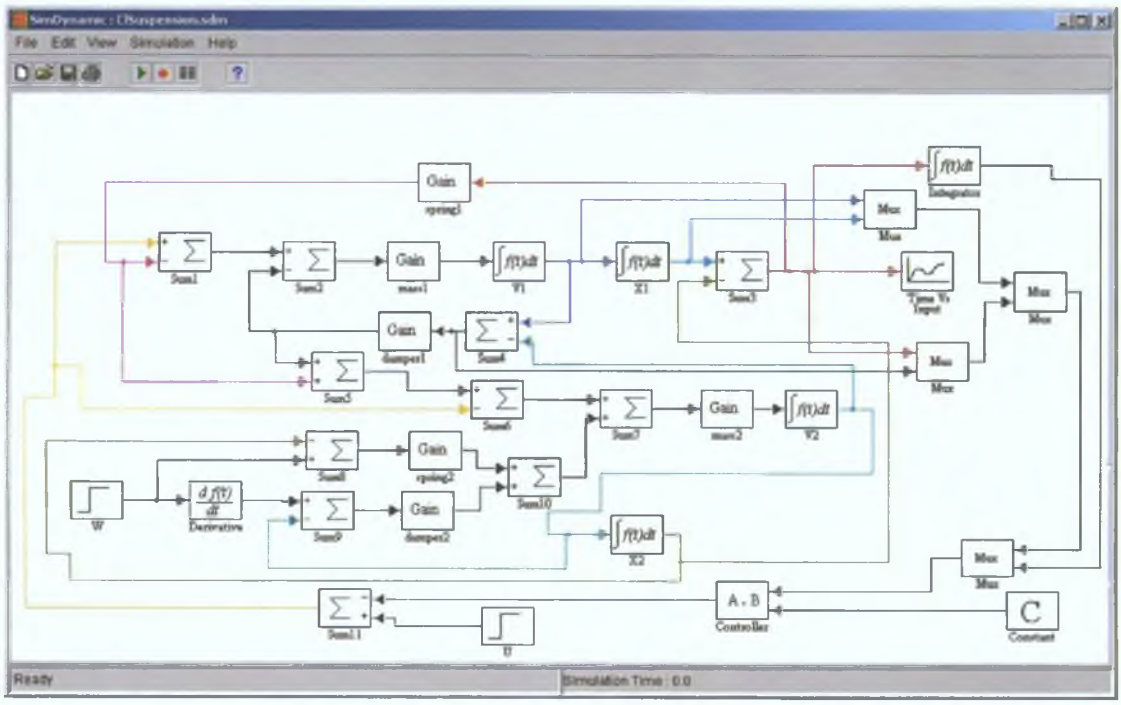


Fig 6.19: Bus suspension model (closed loop) in SimDynamic

The controller used the following gain matrix: [2300000 0 500000000 0 8000000]. An *Integrator* node is used to find $\int y1$ and four *Mux* nodes are used to combine all signals in a vector form. A *Dot Product* node (labelled as controller) is used as the feedback controller.

6.4.3 Running Simulation and Viewing Result

Simulation was carried out for 50 seconds for the open-loop model, using RK_DP integrator and step size 0.01. Following parameter values were use

- Mass1, m1 = 2500 kg
- Spring1 constant, k1 = 80000 N/m
- Damper1 constant, b1 = 350 Ns/m
- Mass2, m2 = 320 kg
- Spring2 constant, k2 = 500000 N/m
- Damper2 constant, b2 = 15020 Ns/m

The open loop response from the bus suspension model is shown in figure 6.20

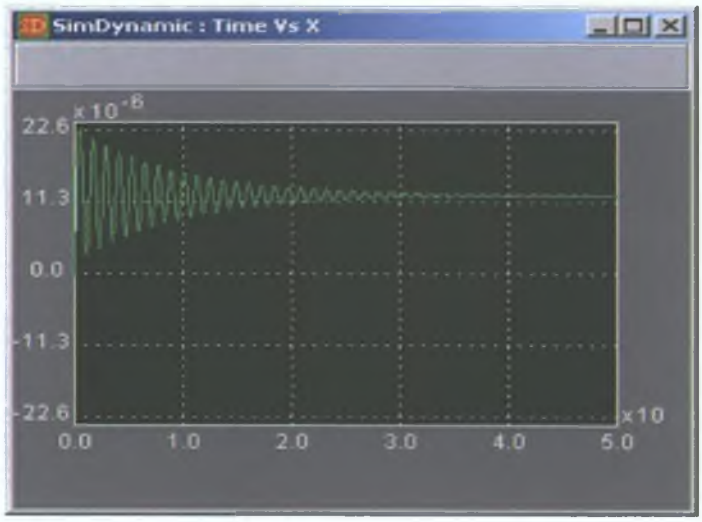


Fig 6.20: Graphical result for bus suspension model (open loop)

For the closed loop model same parameter values were used and simulation was carried out for 2 seconds. Figure 6.21 shows the result for this simulation run,

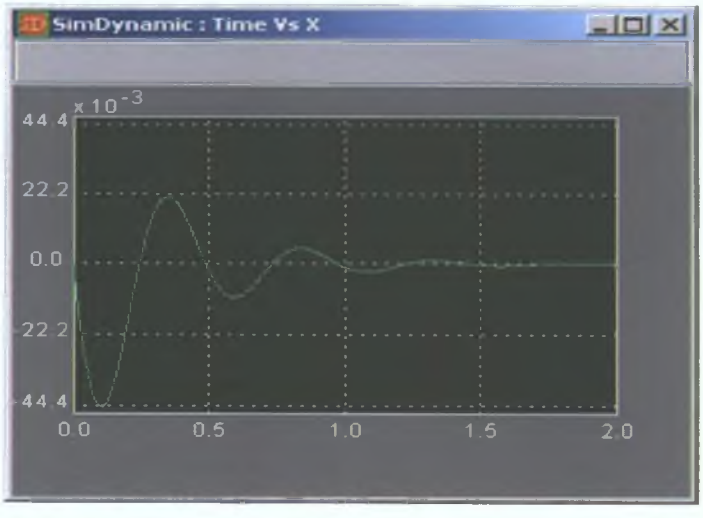


Fig 6.21: Graphical result for bus suspension model (closed loop)

6.4.4 3D Animation for Bus Suspension Model

To make the 3D animation more attractive, instead of a single step input, discrete pulses of various amplitude and sample time are applied to the closed loop model of figure 6.19 as road disturbance. For four wheels of the bus four discrete pulse trains (figure 6.22) are applied to the model and their responses are saved in data files.

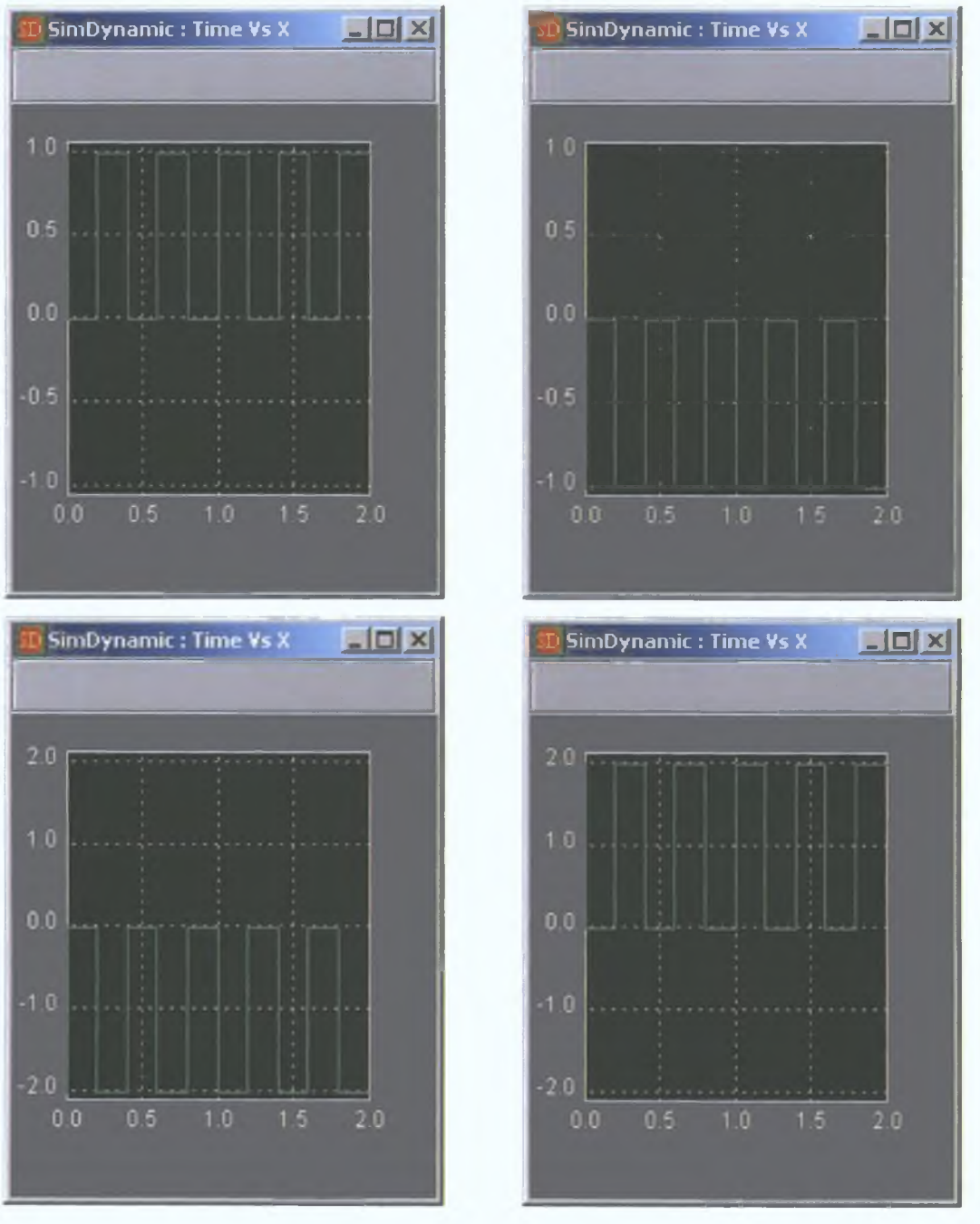


Fig 6:22: Discrete pulses applied to closed-loop bus suspension model

To construct a 3D model of a spring-damper-mass system two cylinders are used. One cylinder will act as the spring-damper and other cylinder will act as a mass on top of it. In this way the 3D model for a four spring-damper-mass is created and placed over a rigid platform. A box is used as the rigid platform. Figure 6.23 presents four different snap shots of 3D animation for the bus suspension model.

Fig 6.23 (a)

Fig 6.23 (b)

Fig 6.23 (c)

Fig 6.23 (d)

Fig 6.23: 3D animation for bus suspension model

6.5 Qualitative Decision Making (QDM)

This section presents an effort to model a qualitative decision making system using *SimDynamic*. Consider the following system,

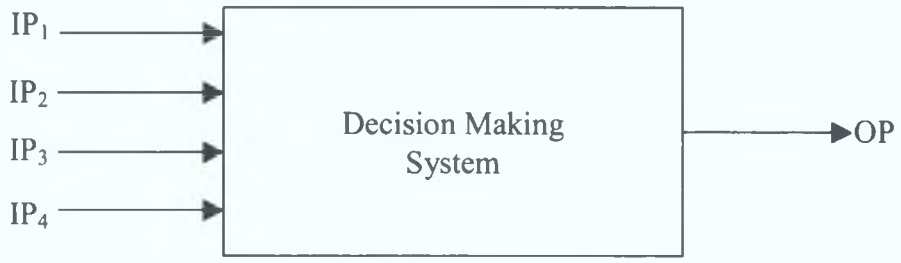


Fig 6.24: Qualitative decision making system

It has four inputs (IP_i) and one output (OP) and the characteristics of the system are as follows,

- All input and output values are measured on a numerical scale ranging from 0 to 1.0.
- If output (OP) value is less than 0.5 then it is unsatisfactory.
- IP_1 and IP_2 have major impact on the system and IP_3 and IP_4 have minor impact on the system.
- Threshold values for IP_1 and IP_2 are 0.55 and 0.6 respectively.
- Threshold values for IP_3 and IP_4 are 0.4 and 0.3 respectively.
- If either IP_1 or IP_2 is less than its threshold value then output must be unsatisfactory (less than 0.5), irrespective of other input values.
- If either IP_1 or IP_2 is at its threshold value, all other inputs must be above or equal to their respective threshold values to produce satisfactory output.
- If any two inputs are below their respective threshold values at the same time, then output must be unsatisfactory.

6.5.1 Building the QDM Model

This system can be modelled easily using logical programming. But here a heuristic approach has been adopted. Following differential equation is used for each of the four inputs,

$$a\frac{d^2x}{dt^2} + b\frac{dx}{dt} + cx = f(t) \qquad 6.12$$

Arithmetic average of the outputs from four equations is used as the final output from the system. The main challenge was to find the coefficients (a, b, c) for each input to develop the above system. A close observation showed that 'c' has the greatest affect on the equation, 'b' has linear affect and 'a' has the least affect on the equation. With lower c values equation output tends to rise and vice versa. Coefficient values were chosen in such a way that all the individual equations produce the minimum satisfactory (0.5) output, so that the final output from the system is satisfactory (0.5) when all the inputs are at their respective threshold values. From these observations and assumptions following coefficient values are set for each equation after numerous trial and error run,

- Input 1 (IP_1): a =1, b = 1, c = 1.1
- Input 2 (IP_2): a = 1, b = 1, c = 1.2
- Input 3 (IP_3): a = 1, b = 1, c = 0.8
- Input 4 (IP_4): a = 1, b = 1, c = 0.6

Following QDM model is built using all above equations,

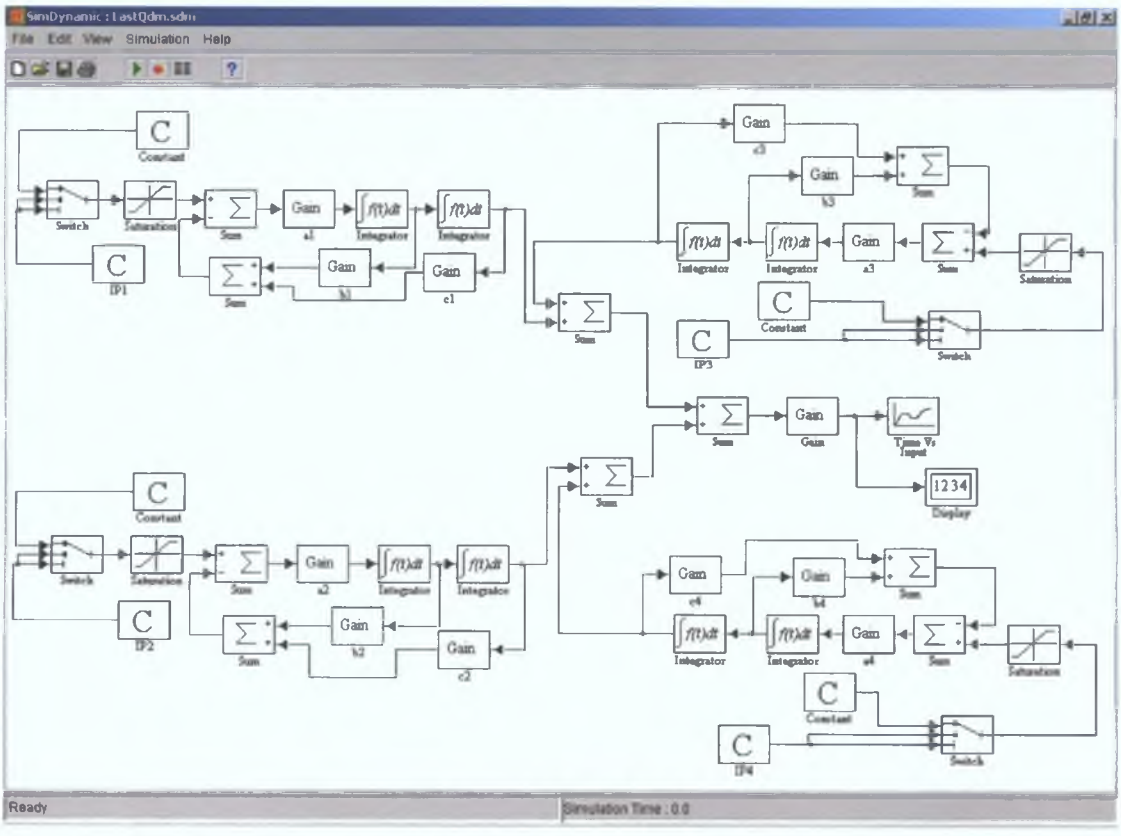


Fig 6.19: QDM model in SimDynamic

Equation for each input is implemented using similar steps as in section 6.1. For each input an additional *switch* node and a *saturation* node are used in series. *Switch* node will force a zero input if the input is below the threshold value, and the *saturation* node is used to limit the input value within the valid input range i.e. 0.0 – 1.0. Outputs from all for equations are summed and then multiplied by 0.25 (to get the arithmetic average) using a *gain* node. A *Time vs Input* node and a *display* node are used for graphical and numerical results respectively.

6.5.2 Running Simulation and Viewing Result

Simulation was carried out for 10 seconds with step size 0.1 and RK_DP integrator. Following table (table 6.3) shows results for some combinations of the inputs,

Table 6.3: Results for QDM model

IP_1	IP_2	IP_3	IP_4	OP
1	1	1	1	0.658
0.55	0.6	0.4	0.3	0.5
0.54	1	1	1	0.458
1	0.59	1	1	0.477
1	1	0.3	0.2	0.438
0	0	0	0	0
0.54	1	0.3	1	0.334
1	0.6	1	0.2	0.478

From the results shown in the table it is quite obvious that the model has implemented the system according to the specification in the problem statements.

6.6 Discussion on *SimDynamic*

6.6.1 Attributes of *SimDynamic*

SimDynamic is entirely written in Java and the package appears as an applet [58] on an HTML page [59]. Therefore it is available over the World Wide Web, taking full advantage of the portability of Java and offers most of the advantages of a Web-based simulation software. Although Web version of *SimDynamic* lacks some features, such as file saving (due to security restriction imposed by Java) or 3D animation, it allows users full interactive capabilities and provides model creation, with both numerical and graphical outputs. Users may create, modify and interact with a simulation model

with as ease as with a non Web-based software. It was mentioned in chapter 1 that *SimDynamic* can also be used as a non Web-based simulation software (Appendix D provides full installation instructions), and in that case it can be considered as a complete professional simulation package. From software engineering point of view some of the aspects of *SimDynamic* are worth mentioning here

- **Functionality/Scope:** The most fundamental property of a simulation tool may be its scope, that is, the set of simulations that it is potentially able to execute. There are many tools designed for simulating a specific type of models. These tools may give very good support for the purpose they are designed for, but they tend to be too restrictive for simulation of complex domains. Because of a rich collection of functional nodes (as a whole 83 in this implementation) *SimDynamic* offers modelling of a wide range of real world dynamic systems, including mechanical, electrical thermodynamic and fluid mechanic systems. Not only real world systems, it can be used to solve mathematical equations or to perform simple matrix operations. A model of moderate complexity can be constructed with minimum of effort and without writing any code. It supports linear and non-linear systems, modelled in continuous time, sampled time or a hybrid of the two. Finally it provides both numerical, and graphical result, and also 3D animation.

- **Maintainability/Expandability:** The object-oriented structure offered by Java and maintained in *SimDynamic* allows easy expansion of the package. The functional nodes have been implemented as individual objects (classes). So extension of the package can be achieved easily by creating additional functional nodes (classes) and incorporating them to the main engine (simulator) of *SimDynamic*.

- **Usability/Ease of Use:** *SimDynamic* presents a simple and friendly graphical user interface (GUI). Users can build models as block diagram, using simple mouse operation. It also provides appropriate error messages whenever an error is encountered in a model during a simulation run.

- **Efficiency:** Object-oriented paradigm, utilized to develop *SimDynamic*, reduces wasteful use of system resources such as memory and processor cycle resulted from reuse of code. Moreover, to store models *SimDynamic* uses Java built in class Vectors that expand themselves as necessary instead of being initially defined with unchanging sizes. As a result, the amount of memory required by the package at a given time depends on the number of nodes and lines within the current model and that ensures the optimal memory management.

- **Reliability:** Generally the reliability of the software comes down to a simple question: does the software provide correct and complete result? Now, implementation of *SimDynamic* involves the use of most authenticated and proved algorithms. Therefore it is desirable that it will produce correct result. Furthermore, some of the *SimDynamic* results were compared to the results from well-established simulation package SIMULINK [60], and in all cases results from *Simdynamic* matched with the results from Simulink.

- **Portability:** Software portability is the ability to use the same program in multiple environments. As it is mentioned before *SimDynamic* is entirely written in Java, which itself is a highly portable language since it translates to machine independent byte-codes, which are interpreted by a Java Virtual Machine (JVM). So to access the Web version of *SimDynamci* user will only need a Java compatible Web browser such as Windows Explorer or Netscape irrespective of the machine architecture. The non Web based version of the package can also be used on multiple platform.

- **Consistency:** The Abstract Window Toolkit in Java aids the creation of graphical user interface in *SimDynamic* and allows the interface to be rendered according to the platform dependent information, ensuring maximum graphical compatibility with the platform. The interface is consistent also in that comparable operations are activated in the same way. For example, the buttons on various dialog boxes have a consistent look and feel and method of operation.

- **User Guidance:** The interface incorporates on-line help for user guidance and assistance. The help provides all the information needed to get along with *SimDynamic,* such as how to build a model, simulation parameters, functional description of nodes etc.

6.6.2 Extension Scopes

The most adequate media for remote access is presently the World Wide Web. Web based simulation is a means to exploit Web technology to support the future of computer simulation. *SimDynamic* is an attempt to introduce dynamic system simulation on the Web. This implementation of *SimDynamic* contains a small number of limitations and these are programming issues that can be remedied with more complex programming techniques than undertaken for the creation of *SimDynamic.* The Current version provides 83 functional nodes to facilitate modelling, but it is extendable because of the object-oriented paradigm utilized to develop *SimDynamic.* Future additions should include nodes for handling more advanced and various type simulation models, particularly discrete event simulation models.

Although *SimDynamic* provides a good number of integrators to solve ordinary differential equations, they are all fixed step integrators and integration of stiff problems has not been considered. In realistic applications the solution is often changing rapidly at certain critical times and it is difficult to approximate accurately. At this particular time short integration steps are needed to maintain accuracy. At other times for the same problem computing time can be saved by using large integration steps and still get sufficient accuracy. It is usually impossible to predict the precise time the small steps are needed. So the software should have the mechanism to handle this step size automatically. And this is not very painstaking to achieve. All the algorithms used to implement *SimDynamic* fixed step integrators have their adaptive (variable step) counterparts. By incorporating variable step integrators it is possible to enhance computing ability of *SimDynamic.*

Another area of interest for future works should be the 3D animation provided by *SimDynamic*, which can be considered in its embryonic stage. This version provides

only four basic 3D shapes (box, cylinder, cone and sphere) for all sorts of 3D modelling that are simply not sufficient to model realistic complex models. Future version of *SimDynamic* should include more and more 3D shapes to provide 3D modelling flexibility. Also at present 3D animation is not available in the Web version of *SimDynamic*, because of the security restrictions imposed by Java on Applets. In this implementation the simulation results are saved in data files and 3D animator reads the results from data files. 3D animation can be made available in the Web version also by avoiding this use of files to communicate between the main simulation engine and the 3D animator.

Chapter Seven

7 Conclusions and Suggested Future Work

7.1 Conclusions

SimDynamic, a simulation package, has been developed for modelling, simulating and analysing dynamic systems on the Web, and it has achieved most of the design goals. Simulation results from *SimDynamic* were compared to the results from Simulink, and in all cases results from both software were identical. Developed entirely in Java, it offers most of the advantages of Web-based simulation described in chapter one. Major advantages are

Wide Availability

- allows access to many platforms without recompiling.
- allows access to distant sites without transporting through hardware or software.

Efficient Maintenance

- enables frequent modifications to be made and instantly distributed
- allows modifications and implementations made through the server

SimDynamic can also be used as a non Web-based simulation software. Some of the salient features of *SimDynamic* are summarised here,

- **Functionality:** The current implementation contains 83 functional nodes, that will enable user to model a wide range of real world dynamic systems including mechanical, electrical and thermodynamic systems.
- **Expandability:** The package can be extended easily by implementing additional functional nodes and incorporating them to the main simulation engine.
- **Portability:** Both the Web-based and no Web-based version of *SimDynamic* can be used on various platforms.

- **User Guidance:** *SimDynamic* user interface provides on-line help for user guidance and assistance.
- **Reliability:** Because of the use of authenticated and proved algorithms in *SimDynamic* development, it produces desirable results.

7.2 Suggested Future work

- Computing power of *SimDynamic* can be enhanced by incorporating more functional nodes to handle advanced and various types of simulation models, particularly discrete event simulation models.
- Better accuracy, especially for stiff differential equations, can be achieved by implementing variable steps integrators.
- 3D animation should be introduced in the Web-based version of *SimDynamic.*
- Model saving option should be enabled in the Web-based version. A database can be maintained to save models separately for each user. Each user should be given unique ID to maintain his/her account.
- Unauthorised access or modification to the models developed using *SimDyanmic* should be protected by imposing password or time limit restriction on the package.

Despite few limitations in the Web-based version, *SimDynamic* encourages users of all levels of understanding to interactively develop models, by providing a high level of user interactivity through simple interfaces. Additionally it enables the users to understand the system under consideration through graphical as well as numerical results of the simulation. Because of wide range modelling ability and tremendous flexibility, *SimDynamic* may be a rendezvous for the simulation practitioners on the Web.

References

[1] Andrew S. Tanenbaum. (1996) "Computer Networks". Prentice-Hall, Inc. USA.

[2] Sun Microsystems, Inc. © 1995 – 2002, "The Source For Java Technology". http://java.sun.com/

[3] Martin Barnes. "Virtual Reality and Simulation". *Proceedings of the 1996 Winter Simulation Conference,* pp 101 – 110, 1996

[4] Quest Software Inc. © 1985 – 2002, http://www.quest.com/

[5] Delmia Corp. http://www.delmia.com

[6] Karen C. Jones, Marc W. Cygnus, Richard L. Storch, Kenneth D. Farnsworth. "Virtual Reality for Manufacturing Simulation". *Proceedings of the 1993 Winter Simulation Conference,*pp 882 – 887, 1993

[7] AutoSimulations, Inc. http://www.prolog-italia.com/Autosimulations.htm

[8] Robert Macredie, Simon J.E. Taylor, Xiaoning Yu and Richard Keeble. "Virtual Reality and Simulation: An Overview". *Proceedings of the 1996 Winter Simulation Conference*, pp 669 – 674, 1996

[9] Hank Grant and Chuen-Ki Lai. "Simulation Modelling with Artificial Reality Technology (SMART): An Integration of Virtual Reality and Simulation Modelling". *Proceedings of the 1998 Winter Simulation Conference,* pp 437 – 441, 1998

[10] Fifth Dimension Technologies. 1996. *5DT Glove – User's Manual*, Pretoria.

[11] Virtual I-O. 1995. *VIO I-Glasses – user's Manual*, Seattle.

[12] Scott Kimbrough. "Control System Development Tools". *ACM SIGAPL APL Quote Quad, Proceedings of the international conference on APL: APL in transition January 1987*, Volume 17 Issue 05-10-1987, 4

[13] Schmid, Chr., "Real-Time Control with CADACS-PC". In M. Jamshidi and C.J. Herget (Ed.): Recent Advances in Computer-Aided Control Systems Engineering. North-Holland, Amsterdam, 1992, S. 337-355.

[14] Schmid, Chr., "KEDDC - A Computer-Aided Analysis and Design Package for Control Systems". In M. Jamshidi und C.J. Herget (Ed.): Advances in Computer-Aided Control Systems Engineering. North-Holland, Amsterdam, 1985, S. 159 180.

[15] Fishwick,P.A., "SIMPACK: Getting Started with Simulation Programming in C and C++", In *1992 Winter Simulation Conference*, December, Arlington, VA, 154-162.

[16] Martin Otter. "The ANDECS Simulation Environment DSSIM". DLR FF-DR-ER Technical report TR R101 – 93, March 1993

[17] DLR - Institute of Robotics and Mechatronics. http://www.robotic.dlr.de/control/andecs/

[18] Hilding Elmqvist, Francois E. Celler and Martin Otter. "Object-Oriented Modeling of Hybrid Systems". Keynote Address, ESS 93, European Simulation Symposium, Delft, The Netherlands, Oct 25 –28, 1993, pp 1 –11.

[19] Dynasim AB. © 1997 – 2002, http://www.dynasim.se/

[20] Miller, D.C., Thorpe, J.A. "SIMNET: The Advent of Simulator Networking." *Proceedings of the IEEE, Vol. 83*, No. 8, pp 1114-1123, 1995

[21] IEEE Standard for Inform. Tech. – Protocols for Distributed Simulation Applications: Entity Information and Interaction. IEEE Standard 1278 – 1993. New York: IEEE Computer Soc., 1993.

[22] Thomas, R., Neilson, I. "Harnessing Simulations in the Service of Education: The Interact Simulation Environment." *Computer Educ. Vol 25,* pp 21-29, 1995

[23] Cole, R., Tooker, S. "Physics To Go: Web-based Tutorials for CoLoS Physics Simulations." *Proceeding of the Frontiers on Education FIE '96 26th Annual Conference.*

[24] Fishwick, P.A. "Web-Based Simulation: Some Personal Observations." *Winter Simulation Conference,* Coronado, CA, pp 772-779, 1996

[25] Buss, A.H., Stork, K.A. "Discrete Event Simulation on the World Wide Web Using Java." *Winter Simulation Conference,* Coronado, CA, pp 780-785, 1996

[26] Miller, J.A., Nair, R.S., Zhang, Z., Zhao, H. "JSIM: A Java-Based Simulation and Animation Environment." *Proceedings of the 30th Annual Simulation Symposium,* Atlanta, Georgia, pp. 31-42. 1997

[27] Howell, F., McNab, R. "SimJava: A Discrete Event Simulation Package for Java With Applications in Computer Systems Modelling." *Proceedings of First International Conference on Web-based Modelling and Simulation,* San Diego CA, Society for Computer Simulation, Jan 1998.

[28] Page, E.H., Moose, R.L., Griffin, S.P. "Web-Based Simulation in SimJava Using Remote Method Invocation." *Winter Simulation Conference,* Atlanta, Georgia, pp 468-474, 1997

[29] Lorenz, P., Dorwarth, H., Ritter, K., Schriber, T.J. "Towards a Web Based Simulation Environment." *Winter Simulation Conference,* Atlanta, Georgia, pp 1338-1344, 1997

[30] Yücessan, E., Chen, C.H., Lee, I. "Web-Based Simulation Experiments." *Winter Simulation Conference,* Washington, pp 1649-1654, 1998

[31] Chen, C.H., Chen, H.C., Dai, L. "A Gradient Approach for Smartly Allocating Computing Budget For Discrete Event Simulation." *Winter Simulation Conference,* San Diego, CA, pp 398-405, 1996

[32] Veith, T.L., Kobza, J.E., Koelling, C.P. "NetSim: Java ™ – Based Simulation for the World Wide Web." *Computer & Operation Research 26* (1999), pp 607-621

[33] Elmaghraby, A.S., Elfayoumy, S.A., Karachiwala, I.S., Graham, J.H., Emam, A.Z., Sleem, A.M. "Web-Based Performance Visualization of Distributed Discrete Event Simulation." *Winter Simulation Conference,* Phoenix, Arizona, pp 1618-1623, 1999

[34] Salisbury, C.F., Farr, D.S., Moore, J.A. "Web-Based Simulation Visualization Using Java 3D." *Winter Simulation Conference,* Phoenix, Arizona, pp 1425-1429, 1999

[35] Schmid, C. "A Remote Laboratory Using Virtual Reality on the Web." *Simulation, Special Issue: On Web-Based Simulation, 73 (1999),* No. 1, S. 13-21.

[36] Schmid, C. "Virtual Control Laboratories and Remote Experimentation in Control Engineering". *Proceedings of 11th EAEEIE Annual Conference on Innovations in Education for Electrical and Information Engineering,* ULM , Germany, S.213-218, April 2000

[37] Schmid, C. "Remote Experimentation Techniques for Teaching Control Engineering". *4th International Scientific - Technical Conference Process Control 2000,* Kouty nad Desnou, Czech Republic, June 2000

[38] Schmid, C. "VCLab - The Virtual Control Engineering Laboratory". Invited paper. *Proceedings of IFAC Symposium on System Identification*, Santa Barbara, California, USA, June 2000

[39] C. Schmid and A. Ali. "A Web-based System for Control Engineering Education". Invited paper. *Proceedings of American Control Conference ACC'2000,* Chicago, June 2000.

[40] C.Schmid. "Remote Experimentation in Control Engineering". *45th International Scientific Colloquium,* Ilmenau Technical University, October 2000

[41] Guru, A., Savory, P., Williams, R. "A Web-Based Interface for Storing and Executing Simulation Models." *Winter Simulation Conference,* Orlando, Florida, pp 1810-1814, 2000

[42] Marr, C., Storey, C., Biles, W.E., Kleijnen, J.P.C. "A Java-Based Simulation Manager for Web-Based Simulation." *Winter Simulation Conference,* Orlando, Florida, pp 1825-1822, 2000

[43] Dhananjai Madhava Rao, Victoria Chernyakhovsky and Philip A. Wilsey. "WESE: A Web-based Environment for Systems Engineering". *Proceedings of the 2000 International Conference on Web-based Modelling and Simulation, WEBSIM-2000.* San Diego, California, October 2000.

[44] Patrick Naughton and Herbert Schildt. (2000) "Java™ 2: The Complete Reference". McGraw-Hill Companies, Inc. USA.

[45] John Zukowski. (1997) "Java AWT Reference". O'Reilly & Associates, Inc. USA.

[46] Henry Sowizral, Kevin Rushforth and Michael Deering. (1998) "The Java™ 3D API Specification". Addison-Wesley Publishing Company. USA.

[47] Shepley L. Ross. (1989) "Introduction to Ordinary Differential Equations". JohnWiley & Sons. USA.

[48] http://www.aero.gla.ac.uk/UGrad/home_pages/nmae/nmae/rkutta/rkutta.html

[49] "Pair of Runge–Kutta formulas". Bogacki P, Shampine LF. A 3(2).Appl Math Lett 1989;2:1–9.

[50] William H. Press, Brian P. Flannery, Saul A. Teukolsky and William T. Vetterling. (1987) "Numerical Recipes: The Art of Scientific Computing". Cambridge University Press. USA.

[51] http://www.maths.uq.edu.au/~gac/math3201/math3201.html

[52] D.K. Anand and R.B.Zmood. (1995) "Introduction to Control Systems". Butterworth-Heinemann Ltd. UK.

[53] Francis Scheid. (1988) "Schaum's Outline of Theory and Problems of Numerical Analysis". McGraw-Hill Companies Inc, USA.

[54] Phillip A. Regalia. (1995) "Adaptive IIR Filtering in Signal Processing and Control. Marcel". Dekker, Inc. USA.

[55] Bernard Friedland. (1996) "Advanced Control System Design". Prentice-Hall, Inc. USA.

[56] Martin Hargeaves. (1996) "Engineering Systems: Modelling and Control". Addison-Wesley Longman Ltd. England.

[57] Jerry Banks, John S. Carson, Barry L. Nelson. (1999) "Discrete – Event System Simulation". Prentice-Hall, Inc. USA.

[58] James Gosling, Frank Yellin, The Java Team. (1996) "The Java™ Application Programming Interface, Volume 2, Window Toolkit and Applets". Addison-Wesley Publishing Company. USA.

[59] http://student.dcu.ie/~mahbubk2/simdynamic.html

[60] The MathWorks Inc. © 1994 – 2002, http://www.mathworks.com/products/simulink

Appendix A

Node Properties

A.1 Continuous Node Set

Node name	**Derivative**
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Continuous
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector / matrix

Node name	**Integrator**
Data type supported	Numerical (real only)
Direct feedthrough	If reset and/or external initial condition are enabled
Sample time	Continuous
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	**Memory**
Data type supported	Numerical (real / complex)
Direct feedthrough	No
Sample time	Continuous
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number/ vector / matrix

Node name	**State Space**
Data type supported	Numerical (real only)
Direct feedthrough	If D is non zero matrix
Sample time	Continuous
Expansion of input/parameter	Expands only the initial condition
Dimension of input/parameter	Input: single number, Parameters: single number / vector / matrix

Node name	**Transfer Function**
Data type supported	Numerical (real only)
Direct feedthrough	If numerator and denominator have same number of columns
Sample time	Continuous
Expansion of input/parameter	Not supported
Dimension of input/parameter	Input: single number Numerator: single / vector/ matrix Denominator: single / vector

Node name	Transport Delay
Data type supported	Numerical (real only)
Direct feedthrough	No
Sample time	Continuous
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	Zero-Pole
Data type supported	Numerical (real only)
Direct feedthrough	If poles and zeros have same number of columns
Sample time	Continuous
Expansion of input/parameter	Not supported
Dimension of input/parameter	Input: single number Zeros: single number / vector / matrix Poles: single number / vector

A.2 Discrete Node Set

Node name	Discrete Filter
Data type supported	Numerical (real only)
Direct feedthrough	If numerator and denominator have same number of columns
Sample time	Discrete
Expansion of input/parameter	Not supported
Dimension of input/parameter	Input: single number Numerator: single / vector / matrix Denominator: single / matrix

Node name	Discrete Integrator
Data type supported	Numerical (real only)
Direct feedthrough	If reset and/or external initial condition are enabled
Sample time	Discrete
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	Discrete State Space
Data type supported	Numerical (real only)
Direct feedthrough	If D is non zero matrix
Sample time	Discrete
Expansion of input/parameter	Expands only the initial condition
Dimension of input/parameter	Input: single number, Parameters: single number / vector / matrix

Node name	Discrete Transfer Function
Data type supported	Numerical (real only)
Direct feedthrough	If numerator and denominator have same number of columns
Sample time	Discrete
Expansion of input/parameter	Not supported
Dimension of input/parameter	Input: single number Numerator: single / vector/ matrix Denominator: single / vector

Node name	Discrete Zero-Pole
Data type supported	Numerical (real only)
Direct feedthrough	If poles and zeros have same number of columns
Sample time	Discrete
Expansion of input/parameter	Not supported
Dimension of input/parameter	Input: single number Zeros: single number / vector / matrix Poles: single number / vector

Node name	Unit Delay
Data type supported	Numerical (real / complex)
Direct feedthrough	No
Sample time	Discrete
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	Zero Order Hold
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Discrete
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

A.3 Tables Node Set

Node name	Direct Look-Up Table (2D)
Data type supported	Input: numerical (real only) Table data: numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not supported
Dimension of input/parameter	Single number / vector / matrix

Node name	Interpolation (2D) with PreLook-up
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported (only for input)
Dimension of input/parameter	Single number / vector / matrix

Node name	Look-Up Table
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not supported
Dimension of input/parameter	Input: single number / vector / matrix Tables: vector

Node name	Look-Up Table (2D)
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported (only for input)
Dimension of input/parameter	Input: single number / vector / matrix Tables: vector / matrix

Node name	Polynomial
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not supported
Dimension of input/parameter	Input: single number / vector / matrix Co-efficient: single number / vector

Node name	Pre Look-up Index Search
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not supported
Dimension of input/parameter	Input: single number / vector / matrix Table: vector

A.4 Maths and Logic Node Set

Node name	Absolute
Data type supported	Numerical (real /complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector / matrix

Node name	Bitwise Logical Operator
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	Combinatorial Logic
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not supported
Dimension of input/parameter	Input: single number / vector Table: matrix

Node name	Dot Product
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not supported
Dimension of input/parameter	Single number / vector

Node name	Gain
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	Logical Operator
Data type supported	Numerical (real only) / boolean
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	Magnitude - Angle to Complex
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	**Math Function 1**
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector / matrix

Node name	**Math Function 2**
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	**Matrix 1**
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Matrix

Node name	**Matrix 2**
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Matrix

Node name	**Min Max**
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	**Product**
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	**Real - Imaginary to Complex**
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited

Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	**Relational Operator**
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	**Resolve Complex**
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector / matrix

Node name	**Rounding Function**
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector / matrix

Node name	**Sign**
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector / matrix

Node name	**Sum**
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	**Trigonometric Function**
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector / matrix

A.5 Non-Linear Node Set

Node name	**Backlash**
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	**Coulomb and Viscous Friction**
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported (only for tables)
Dimension of input/parameter	Input: single number Tables: single number / vector/ matrix

Node name	**Dead Zone**
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	**Manual Switch**
Data type supported	Numerical (real / complex) / boolean
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector / matrix

Node name	**Multi-port Switch**
Data type supported	Numerical (real / complex) / boolean
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	**Quantizer**
Data type supported	Input: numerical (real / complex) Table: numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	Rate Limiter
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	Relay
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	Satuaration
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	Switch
Data type supported	Control input: numerical (real only) Threshold value: numerical(real only) Input: numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

A.6 Miscellaneous Node Set

Node name	Counter
Data type supported	Numerical (real only)
Direct feedthrough	No
Sample time	Continuous
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector / matrix

Node name	Cumulative Function
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector / matrix

Node name	**Data Type Conversion**
Data type supported	Numerical (real only) / boolean
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector / matrix

Node name	**Zero Crossing**
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	**Initial Condition**
Data type supported	Numerical (real only)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Supported
Dimension of input/parameter	Single number / vector / matrix

Node name	**Matrix Concatenation**
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector / matrix

Node name	**Merge**
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector

Node name	**Mux**
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector

Node name	**Probe**
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable

Dimension of input/parameter	Single number / vector / matrix

Node name	**Reshape**
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector / matrix

Node name	**Selector**
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Inherited
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector / matrix

Node name	**Signal Specification**
Data type supported	Numerical (real / complex)
Direct feedthrough	Yes
Sample time	Continuous
Expansion of input/parameter	Not applicable
Dimension of input/parameter	Single number / vector / matrix

A.7 Sink Node Set

Node name	**Display**
Data type supported	Numerical (real / complex) / boolean
Direct feedthrough	Not applicable
Sample time	Inherited
Expansion of input	Not applicable
Dimension of input	Single number / vector / matrix

Node name	**Time vs Input**
Data type supported	Numerical (real only)
Direct feedthrough	Not applicable
Sample time	Inherited
Expansion of input	Not applicable
Dimension of input	Single number / vector / matrix

Node name	**Terminator**
Data type supported	Numerical (real / complex) / boolean
Direct feedthrough	Not applicable
Sample time	Inherited
Expansion of input	Not applicable
Dimension of input	Single number / vector / matrix

Node name	To File
Data type supported	Numerical (real / complex) / boolean
Direct feedthrough	Not applicable
Sample time	Inherited
Expansion of input	Not applicable
Dimension of input	Single number / vector / matrix

Node name	XY Graph
Data type supported	Numerical (real only)
Direct feedthrough	Not applicable
Sample time	Inherited
Expansion of input	Not applicable
Dimension of input	Single number / vector / matrix

Node name	To Data File
Data type supported	Numerical (real / complex) / boolean
Direct feedthrough	Not applicable
Sample time	Inherited
Expansion of input	Not applicable
Dimension of input	Single number / vector / matrix

A.8 Source Node Set

Node name	Bool
Data type supported	Boolean
Direct feedthrough	Not applicable
Sample time	Continuous (step size dependent)
Expansion of parameter	Not applicable
Dimension of parameter	Single number / vector / matrix

Node name	Chirp Signal
Data type supported	Numerical (real only)
Direct feedthrough	Not applicable
Sample time	Continuous
Expansion of parameter	Supported
Dimension of parameter	Single number / vector / matrix

Node name	Clock
Data type supported	Not applicable
Direct feedthrough	Not applicable
Sample time	Continuous (step size dependent)
Expansion of parameter	Not applicable
Dimension of parameter	Not applicable

Node name	Constant
Data type supported	Numerical (real / complex)
Direct feedthrough	Not applicable
Sample time	Continuous (step size dependent)
Expansion of parameter	Not applicable
Dimension of parameter	Single number / vector / matrix

Node name	Digital Clock
Data type supported	Not applicable
Direct feedthrough	Not applicable
Sample time	Discrete
Expansion of parameter	Not applicable
Dimension of parameter	Not applicable

Node name	Discrete Pulse Generator
Data type supported	Numerical (real only)
Direct feedthrough	Not applicable
Sample time	Discrete
Expansion of parameter	Supported
Dimension of parameter	Single number / vector / matrix

Node name	Ground
Data type supported	Not applicable
Direct feedthrough	Not applicable
Sample time	Continuous (step size dependent)
Expansion of parameter	Not applicable
Dimension of parameter	Not applicable

Node name	Pulse Generator
Data type supported	Numerical (real only)
Direct feedthrough	Not applicable
Sample time	Continuous
Expansion of parameter	Supported
Dimension of parameter	Single number / vector / matrix

Node name	Ramp
Data type supported	Numerical (real only)
Direct feedthrough	Not applicable
Sample time	Continuous
Expansion of parameter	Supported
Dimension of parameter	Single number / vector / matrix

Node name	Random Number Generator
Data type supported	Numerical (real only)
Direct feedthrough	Not applicable

Sample time	Discrete
Expansion of parameter	Supported
Dimension of parameter	Single number / vector / matrix

Node name	**Repeating Sequence**
Data type supported	boolean
Direct feedthrough	Not applicable
Sample time	Continuous
Expansion of parameter	Not applicable
Dimension of parameter	Vector

Node name	**Signal Generator**
Data type supported	Numerical (real only)
Direct feedthrough	Not applicable
Sample time	Continuous
Expansion of parameter	Supported
Dimension of parameter	Single number / vector / matrix

Node name	**Sine Wave**
Data type supported	Numerical (real only)
Direct feedthrough	Not applicable
Sample time	Continuous
Expansion of parameter	Supported
Dimension of parameter	Single number / vector / matrix

Node name	**Step**
Data type supported	Numerical (real only)
Direct feedthrough	Not applicable
Sample time	Discrete
Expansion of parameter	Not applicable
Dimension of parameter	Single number / vector / matrix

Node name	**Uniform Random Number**
Data type supported	Numerical (real only)
Direct feedthrough	Not applicable
Sample time	Discrete
Expansion of parameter	Supported
Dimension of parameter	Single number / vector / matrix

Node name	**From Data File**
Data type supported	Retrieved from file
Direct feedthrough	Not applicable
Sample time	Retrieved from file
Expansion of parameter	Not applicable
Dimension of parameter	Not applicable

Appendix B

Runge-Kutta Coefficients

Dormand-Prince Pair

α	β						
0							
1/5	1/5						
3/10	3/40	9/40					
4/5	44/45	-56/15	32/9				
8/9	19372/6561	-25360/2187	64448/6561	-212/729			
1	9017/3168	-355/33	46732/5247	49/176	-5103/18656		
1	35/384	0	500/1113	125/192	-2187/6784	11/84	
w	35/384	0	500/1113	125/192	-2187/6784	11/84	0
	5179/57600	0	7571/16695	393/640	-92097/339200	187/2100	1/40

Fourth Order Runge-Kutta

α	β			
0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
w	1/6	1/3	1/3	1/6

Bogacki-Shampine

α	β			
0				
1/2	1/2			
3/4	0	3/4		
1	0	0	1	
w	2/9	1/3	4/9	0
	7/24	1/4	1/3	1/8

Heun's Method

α	β	
0		
1	1	
w	1/2	1/2

Appendix C

Bus Suspension System (State-Space Controller)

A bus suspension model can be simplified (considering just one wheel of the bus) to a one-dimensional spring-damper system. A diagram of this system is shown below,

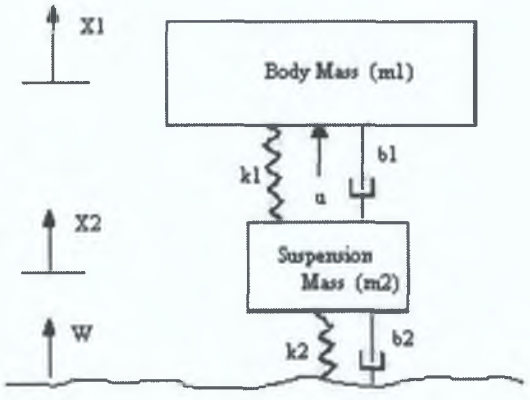


fig 6.19: Bus suspension system (1/4 bus)

where,

- m1 is body mass
- m2 is suspension mass
- k1 is spring constant of suspension system
- k2 is spring constant of wheel and tire
- b1 is damping constant of the suspension system
- b2 is the damping constant of the wheel and tire
- w is the disturbance from the road
- u is the control force.

When the bus is experiencing any road disturbance, the distance (x1 – w) will be very difficult to measure, and the deformation of the tire (x2 – w) is negligible. So the distance (x1 – x2) will be output from the model.

State Space Equation

From the above figure and Newton's law, dynamic equations for this model are as follows,

$$m_1 \frac{d^2 x_1}{dt^2} = u - b_1(\frac{dx_1}{dt} - \frac{dx_2}{dt}) - k_1(x_1 - x_2)$$

$$m_2 \frac{d^2 x_2}{dt^2} = b_1(\frac{dx_1}{dt} - \frac{dx_2}{dt}) + k_1(x_1 - x_2) + b_2(\frac{dw}{dt} - \frac{dx_2}{dt}) + k_2(w - x2) - u$$

Dividing the first and second equation by m_1 and m_2 respectively,

$$\frac{d^2 x_1}{dt^2} = \frac{u}{m_1} - \frac{b_1}{m_1}(\frac{dx_1}{dt} - \frac{dx_2}{dt}) - \frac{k_1}{m_1}\ (x_1 - x_2)$$

$$\frac{d^2 x_2}{dt^2} = \frac{b_1}{m_2}(\frac{dx_1}{dt} - \frac{dx_2}{dt}) + \frac{k_1}{m_2}(x_1 - x_2) + \frac{b_2}{m_2}(\frac{dw}{dt} - \frac{dx_2}{dt}) + \frac{k_2}{m_2}(w - x2) - \frac{u}{m_2}$$

To be a valid state-space representation, the derivative of all states must be in terms of inputs and the states themselves. For this system first three states will be x_1, dx_1/dt, and $y_1 = x_1 - x_2$. Substituting y_1 in the above equations,

$$\frac{d^2 x_1}{dt^2} = \frac{u}{m_1} - \frac{b_1}{m_1}\frac{dy_1}{dt} - \frac{k_1}{m_1}\ y_1$$

$$\frac{d^2 x_2}{dt^2} = \frac{b_1}{m_2}\frac{dy_1}{dt} + \frac{k_1}{m_2} y_1 + \frac{b_2}{m_2}(\frac{dw}{dt} - \frac{dx_2}{dt}) + \frac{k_2}{m_2}(w - x2) - \frac{u}{m_2}$$

Subtracting the second equation from the first equation to get an expression for d^2y/dt^2,

$$\frac{d^2 x_1}{dt^2} - \frac{d^2 x_2}{dt^2} = \frac{d^2 y_1}{dt^2} =$$
$$-b(\frac{1}{m_1} - \frac{1}{m_2})\frac{dy_1}{dt} - k_1(\frac{1}{m_1} + \frac{1}{m_2})y_1 - \frac{b_2}{m_2}(\frac{dw}{dt} - \frac{dx_2}{dt}) - \frac{k_2}{m_2}(w - x_2) + (\frac{1}{m_1} + \frac{1}{m_2})u$$

Integrate this equation to avoid the second order derivative,

$$\frac{dy_1}{dt} = -b_1(\frac{1}{m_1}+\frac{1}{m_2})y_1 - \frac{b_2}{m_2}(w-x_2) + \int(-k_1(\frac{1}{m_1}+\frac{1}{m_2})y_1 - \frac{k_2}{m_2}(w-x_2) + (\frac{1}{m_1}+\frac{1}{m_2})u)dt$$

This equation contains no derivative of the input and dy_1/dt is expressed in terms of states and inputs only, except for the integral. Let the integral is the fourth state of the system and,

$$y_2 = \int(-k_1(\frac{1}{m_1}+\frac{1}{m_2})y_1 - \frac{k_2}{m_2}(w-x_2) + (\frac{1}{m_1}+\frac{1}{m_2})u)dt$$

Then state equation for y_2 is,

$$\frac{dy_2}{dt} = -k_1(\frac{1}{m_1}+\frac{1}{m_2})y_1 - \frac{k_2}{m_2}(w-x_1+y_1) + (\frac{1}{m_1}+\frac{1}{m_2})u$$

And final state equations for y_1 and dx_1/dt are

$$\frac{dy_1}{dt} = -b_1(\frac{1}{m_1}+\frac{1}{m_2})y_1 - \frac{b_2}{m_2}(w-x_1+y_1) + y_2$$

$$\frac{d^2x_1}{dt^2} = \frac{-b_1b_2}{m_1m_2}x_1 + (\frac{b_1}{m_1}(\frac{b_1}{m_1}+\frac{b_1}{m_2}+\frac{b_2}{m_2}) - \frac{k_1}{m_1})y_1 - \frac{b_1}{m_1}y_2 + \frac{u}{m_1} + \frac{b_1b_2}{m_1m_2}w$$

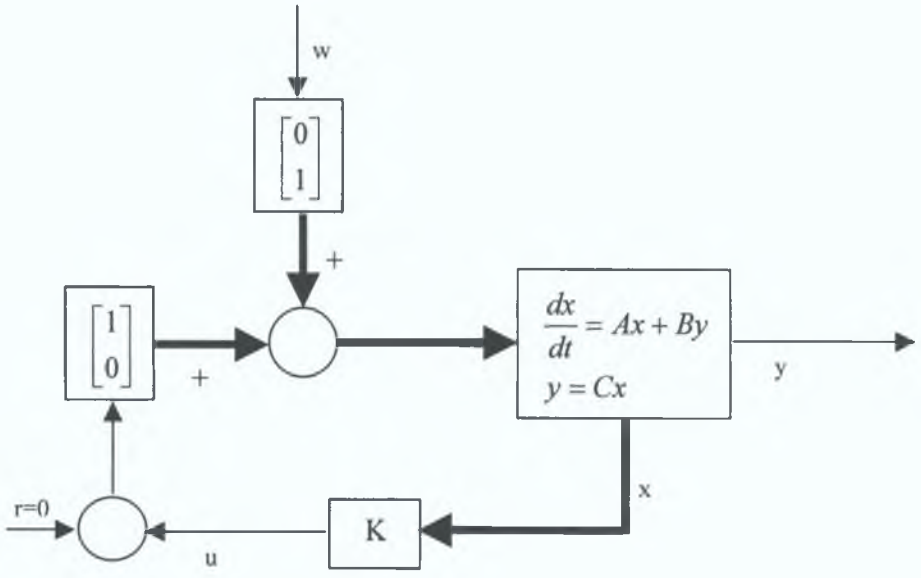
The matrix form of the above equations is,

$$\begin{bmatrix} x_1 \\ \frac{d^2x_1}{dt^2} \\ \frac{dy_1}{dt} \\ \frac{dy_2}{dt} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{-b_1b_2}{m_1m_2} & 0 & \frac{b_1}{m_1}(\frac{b_1}{m_1}+\frac{b_1}{m_2}+\frac{b_2}{m_2}) - \frac{k_1}{m_1} & \frac{-b_1}{m_1} \\ \frac{b_2}{m_2} & 0 & -(\frac{b_1}{m_1}+\frac{b_1}{m_2}+\frac{b_2}{m_2}) & 1 \\ \frac{k_2}{m_2} & 0 & -(\frac{k_1}{m_1}+\frac{k_1}{m_2}+\frac{k_2}{m_2}) & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ \frac{dx_1}{dt} \\ y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{1}{m_1} & \frac{b_1b_2}{m_1m_2} \\ 0 & \frac{-b_2}{m_2} \\ \frac{1}{m_1}+\frac{1}{m_2} & \frac{-k_2}{m_2} \end{bmatrix} \begin{bmatrix} u \\ w \end{bmatrix}$$

$$y = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ \frac{dx_1}{dt} \\ y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} u \\ w \end{bmatrix}$$

Full-State Feedback Controller

Assuming that all the states can be measured, the schematic of the system should be as follows,



The characteristic polynomial for this closed-loop system is the determinant of (sI – (A – B[1,0]'K)). To eliminate the steady-state error, integral control is used here. After adding an extra state $\int (x_1 - x_2) = \int y_1$, the state-space matrices A, B, C and D change to be the following:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{-b_1 b_2}{m_1 m_2} & 0 & \frac{b_1}{m_1}\left(\frac{b_1}{m_1} + \frac{b_1}{m_2} + \frac{b_2}{m_2}\right) - \frac{k_1}{m_1} & \frac{-b_1}{m_1} \\ \frac{b_2}{m_2} & 0 & -\left(\frac{b_1}{m_1} + \frac{b_1}{m_2} + \frac{b_2}{m_2}\right) & 1 \\ \frac{k_2}{m_2} & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 \\ \frac{1}{m_1} & \frac{b_1 b_2}{m_1 m_2} \\ 0 & \frac{-b_2}{m_2} \\ \frac{1}{m_1} + \frac{1}{m_2} & \frac{-k_2}{m_2} \\ 0 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad D = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

Applying trial-and-error method, the matrix for the controller K is found as follows,

$$K = \begin{bmatrix} 0 & 2300000 & 500000000 & 0 & 8000000 \end{bmatrix}$$

Appendix D

SimDynamic Installation Instructions

This appendix provides all the instructions to install and run *SimDynamic*. And also presents a brief manual to handle the built in models that come along with the package. It should be kept in mind that examples are only available to non Web-based version.

D1. Web-based Version

Requirements

User will need a java compatible Web browser irrespective of the operating system (Windows /Unix/Solaries) or hardware. Most browsers (like Internet Explorer or Netscape) now come up with built-in JVM, so user does not need to install anything.

Running *SimDynamic*

Type the address http://student.dcu.ie/~mahbubk2/simdynamic.html in the address bar of the browser. Browser will load the *SimDynamic* homepage. Click the start button in the middle of the page. *SimDynamic* will start, now use the help file if required.

D2. Non Web-based Version

Although *SimDynamic* can be installed on several operating systems (like Windows, Unix or Solaries), this section provides instructions only for Windows.

Requirements

- Pentium III 650 MHz (or higher)
- 128 MB RAM (or higher)
- Java™ 2 Runtime Environment Installed. Can be downloaded from http://java.sun.com/products/jdk/1.2/java2.html
- Java 3D™ API installed. Can be downloaded from http://java.sun.com/products/java-media/3D/

Installation

SimDynamic installer comes with a single self-extracting exe file sd_installer.exe. Run this file (double click in windows explorer). This will install *SimDynamic* in C:\SimDynamic folder. You can change the drive (such as D: or E: etc) but **do not change** the default folder

name, and **do not put** the default folder inside other folder like programme files. After installation *SimDynamic* icon will appear on the desktop.

Running *SimDynamic*

Double click on the icon on the desktop. It will open command prompt window (a small black window, do not close it) and then start *SimDynamic*. Now use the online help file to proceed.

D3. Handling Examples (Only for non Web-based Version)

Pendulum Model

1. Select File → New from main window (figure D1). Model Sheet window will appear.

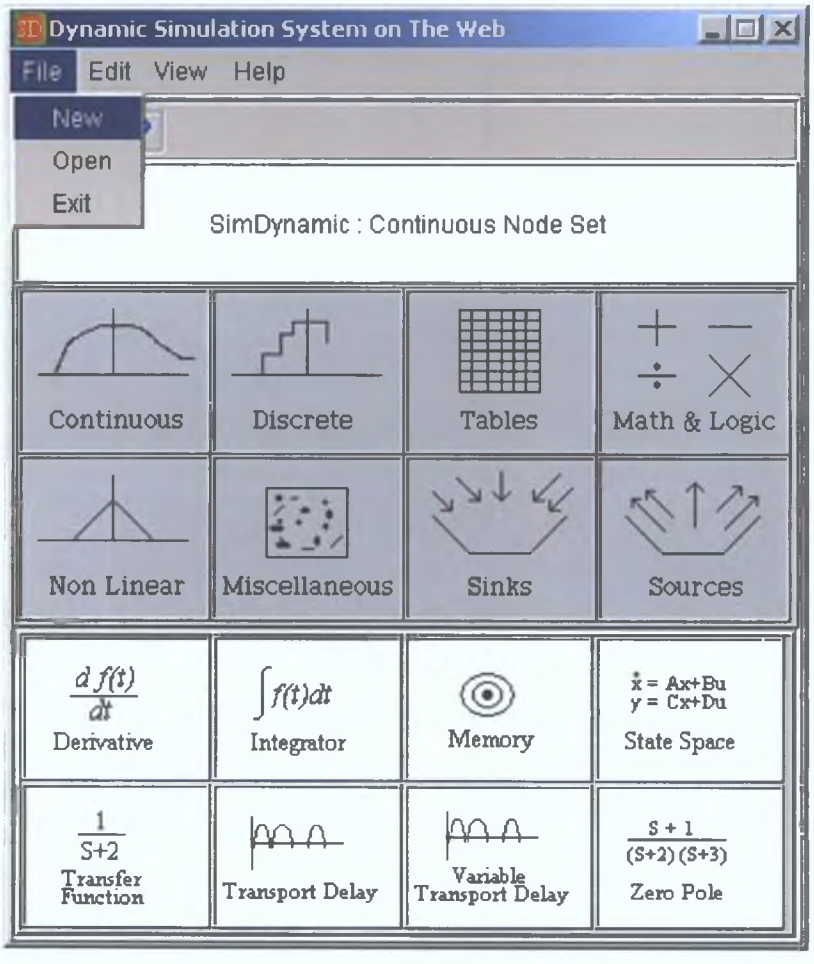


Fig D1

2. Select File → Open from model sheet window (figure D2), file open dialog box will appear.

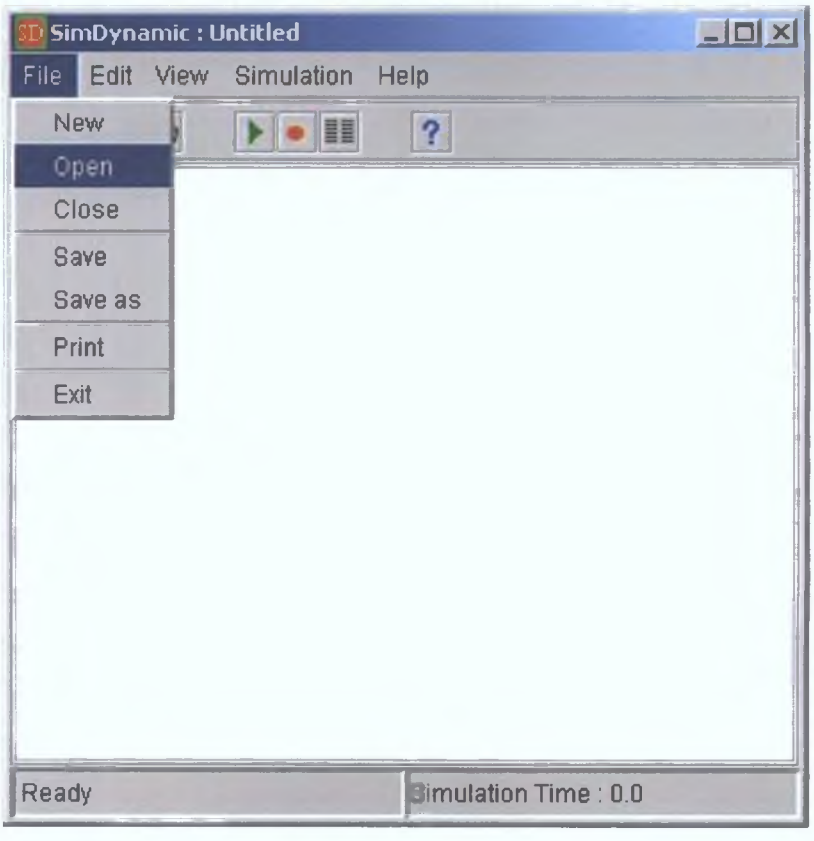


Fig D2

3. Select pendulum.sdm and click OK (or double click on pendulum.sdm). Pendulum model will be loaded (figure D3). Maximize window as needed.

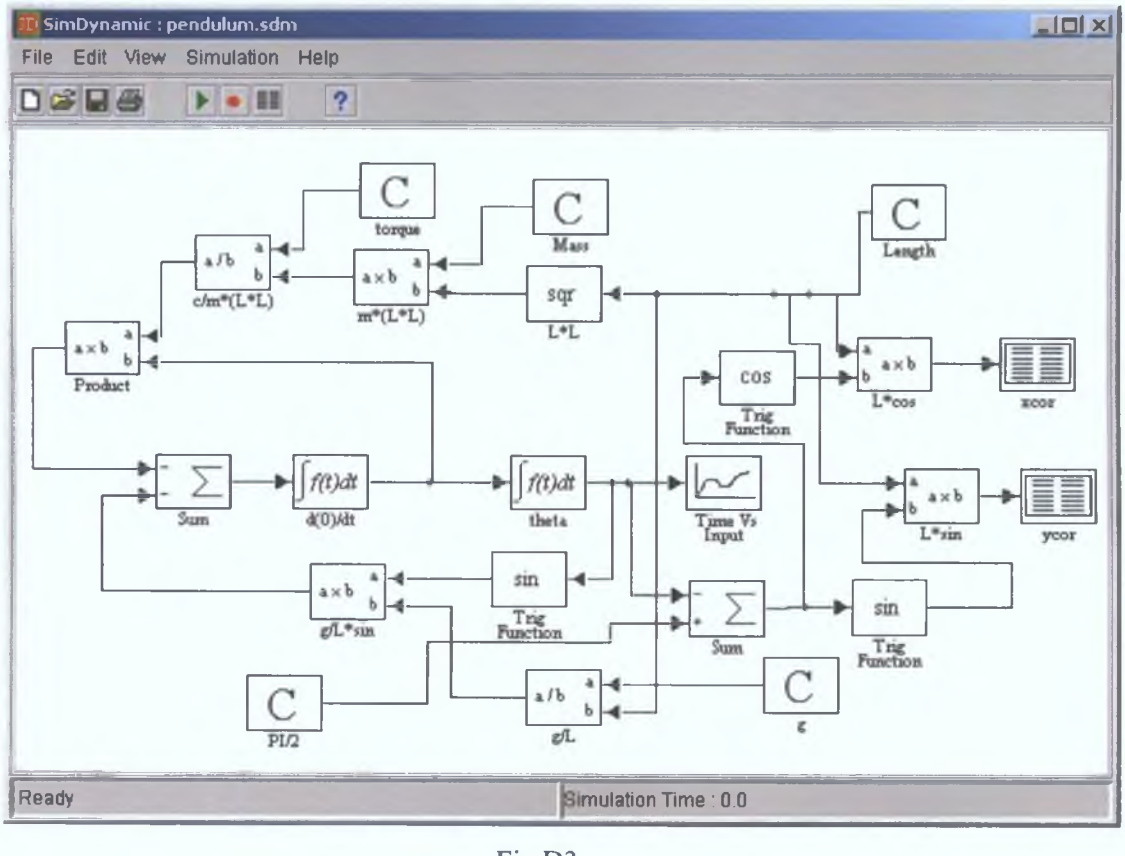


Fig D3

4. To change Simulation parameters select Simulation → Parameter from model sheet window (figure D4). Current settings are RK_DP integrator, step size 0.2 and simulation time from 0 to 10 seconds. Enter new parameters and then click OK.

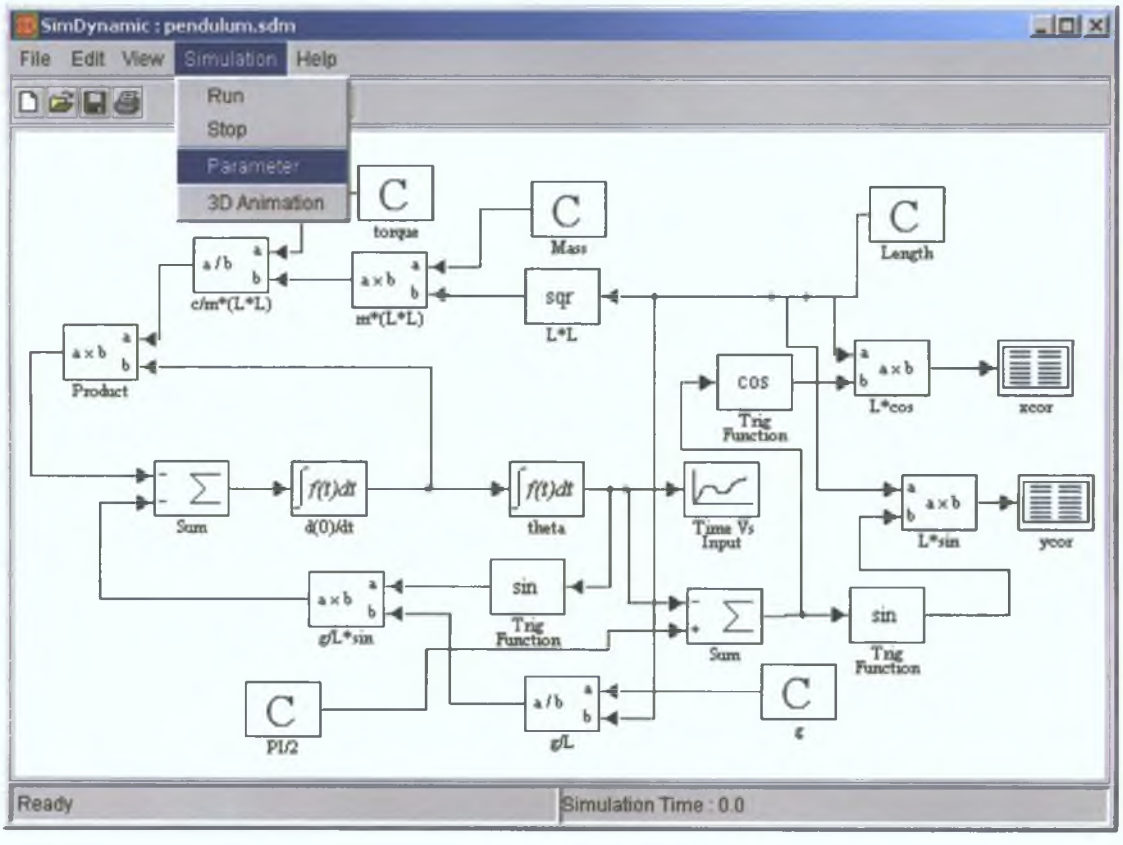


Fig D4

5. To change any model parameter (torque/Mass/Length) double click on appropriate node. Parameter dialog box will appear, set new value and click OK.
6. To run the simulation, select Simulation → Run. Simulation will continue. When Ready appears on the status bar simulation is finshed.
7. To view result, double click on Time vs Input node.
8. To start 3D animation, select Simualtion → 3D animation. 3D Model editor will open. To load the pendulum 3D model, click on load in the model editor, a dialog box will appear, enter pendu.s3d and click on load on the dialog box. Pendulum 3D model will be loaded (figure D5).

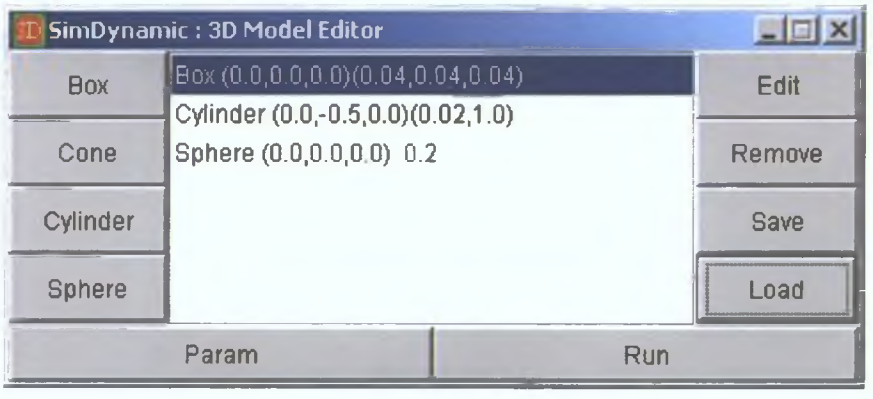


Fig D5

9. Before starting 3D animation save the simulation result in data files. To do so double click on xcor node, a dialog box will appear, in the file name field type xcor and click on save. Similarly double click on ycor and enter ycor and click on save.
10. Now click on Run button (at the bottom of 3D editor). 3D animation will start.
11. Whenever you change any parameter (like torque/Mass/Length) in the simulation model, you have to save the data files xcor and ycor (as in step 9).
12. If you change the length of the pendulum, then the cylinder length of the 3D model should also be changed before 3D animation. To do so, select the cylinder and click on Edit button. Cylinder parameter dialog box will appear (figure D6). On the parameter dialog box set the new length in the height field. Set the Center Y field to minus half of the length. (i.e. if the length is 1.0 then set Center Y to –0.5). DO not change any other value on the Cylinder parameter dialog box. Click OK

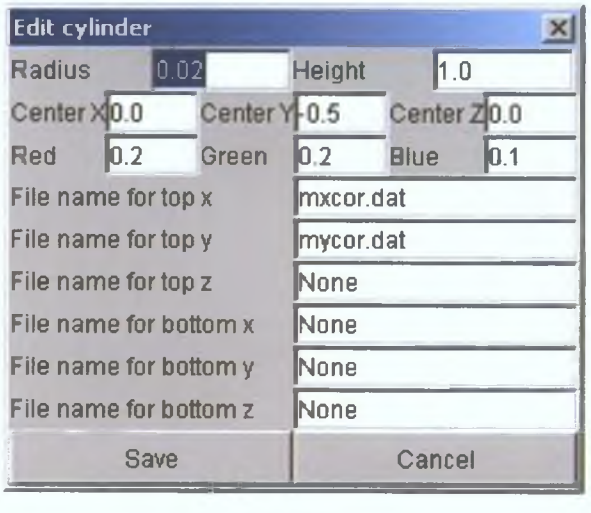


Fig D6

13. Now Click on Run button on the 3D model Editor. 3D animation will start with new length.

14. To close pendulum model close the model sheet window by clicking the cross at the top right of the window or by selecting File → Exit.

Bouncing Ball Model

1. Follow steps 1 – 3 of the previous example to open the file bounce.sdm
2. To change simulation parameter follow step 4. This model doesn't need an integrator. Default simulation time 0 to 20 seconds and step length is 0.005. Keep in mind that a larger step length for this example will produce inaccurate result. If you change step length then set the same value in the node labelled as t in the model.
3. To change any model parameter, like initial velocity (node labelled as v0), initial height (node labelled as y0), elasticity (node labelled as elasticity), click on the appropriate node. Set new parameter value and click OK.
4. Follow steps 6 and 7 from previous example to run and view result.
5. To start 3D animation, select Simualtion → 3D animation. 3D Model editor will open. To load the bouncing ball 3D model, click on load in the model editor, a dialog box will appear, enter bounce.s3d and click on load on the dialog box. Bouncing ball 3D model will be loaded
6. Before starting 3D animation save the simulation result in data files. To do so double click on the node labelled as To Dat File, a dialog box will appear, in the file name field type bounce and click on save.
7. Now click on Run button (at the bottom of 3D editor). 3D animation will start.
8. Whenever you change any parameter (like initial velocity/initial height/elasticity) in the simulation model, you have to save the data file bounce as in step 6.
9. To close bouncing ball model close the model sheet window by clicking the cross at the top right of the window or by selecting File → Exit.