



Graduate Theses, Dissertations, and Problem Reports

2000

Java-based MIDI interface for robot control

Gaurav Subhash Harode
West Virginia University

Follow this and additional works at: <https://researchrepository.wvu.edu/etd>

Recommended Citation

Harode, Gaurav Subhash, "Java-based MIDI interface for robot control" (2000). *Graduate Theses, Dissertations, and Problem Reports*. 1019.
<https://researchrepository.wvu.edu/etd/1019>

This Thesis is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Thesis has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.

JAVA BASED MIDI INTERFACE FOR ROBOT CONTROL

Gaurav S. Harode

**Thesis Submitted to the
College of Engineering and Mineral Resources
At West Virginia University
In Partial fulfillment of the Requirements for
the Degree of**

**Master of Science
In
Industrial Engineering**

**Dr.Rashpal Ahluwalia, Ph.D., Chair
Dr.William Dodrill, Ph.D.
Dr.Ralph Plummer, Ph.D.**

Department of Industrial and Management Systems Engineering

**Morgantown, West Virginia
2000**

Keywords: MIDI, User Interface, Java

ABSTRACT

JAVA Based MIDI Interface for Robot Control

Gaurav S. Harode

Robot offers an excellent means of utilizing high level technology to make a given manufacturing operation more profitable and competitive. Apart from manufacturing, robots are also being utilized in the areas of medicine and agriculture with great returns. Robotic applications now require multiple robots working in a coordinated manner, something similar to an orchestra. This lead to the concept of a Robotic Instrument Digital Interfacing (RIDI) system for real time coordinated control of robotic devices. It provides a novel runtime environment that supports the building and deployment of distributed robotic devices.

The objective of this research is to design and develop a Graphical User Interface (GUI). The GUI aims at providing extensive support for monitoring multiple device activity in real time. With the exception of code for device-specific interfaces, the prototype RIDI-GUI implementation will be coded in Java. This thesis details on the various aspects involved in the design and implementation of the GUI software for two robot arms.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and gratitude to Dr. Rashpal Ahluwalia, my committee chair for his continuous support through research. He was always available to answer my questions and explicate problems that I encountered.

I also appreciate the time and effort of Dr. William Dodrill, my committee member who offered numerous constructive suggestions. Further, I would like to thank Dr. Ralph Plummer for his constant encouragement and support throughout the research.

Finally, I would like to thank my parents, my brother, Priyal and all my friends for their support all through my studies.

TABLE OF CONTENTS

ABSTRACT	ii
ACKNOWLEDGEMENTS	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES	vii
LIST OF TABLES	ix
Chapter 1	1
INTRODUCTION	1
1.1 Overview of ROBOTICS	1
1.2 Industrial Robots	2
1.2.1 Off-Line Programming of a Control Computer	4
1.2.2 On-Line Programming of a Control Computer	5
1.2.3 Applications of Industrial Robots	5
1.3 Need for Research	6
1.3.1 Real Time Environment	7
1.3.2 Real Time Robotic Systems	8
1.3.3 Extensions to the field of MIDI	9
1.4 Problem Statement	10
Chapter 2	13
LITERATURE REVIEW	13
2.1 General Overview of Robot Control	13
2.2 Overview of MIDI Discipline	14
2.2.1 MIDI Interfaces	14
2.2.2 MIDI System	15
2.3 MIDI User Interfaces	19
2.3.1 MIDI Data	22
2.3.2 MIDI Protocol	24
2.4 MIDI Show Control & MIDI Machine Control	25

Chapter 3	27
Robotic Instrument Digital Interfacing System	27
3.1 Introduction to RIDI	27
3.1.1 Key Features of RIDI System	27
3.1.2 RIDI Graphical User Interface	31
3.2 The Robot	32
3.3 Extension of MIDI Capabilities to Robotic Applications	35
3.4 RIDI Feedback	39
3.5 Application Areas for RIDI System	40
Chapter 4	43
RIDI - GUI SOFTWARE	43
4.1 Introduction to RIDI-GUI	43
4.1.1 Structure of the GUI Screens	43
4.1.2 GUI - Data Flow	44
4.2 The User Interface	46
4.2.1 RIDI Kinematics Screen	46
4.2.2 RIDI Forward Kinematics Input Screen	49
4.2.3 RIDI Reverse Kinematics Input Screen	54
4.2.4 GUI Activity Screen	56
4.3 Layered Structure of the GUI Activity Area	60
4.4 Thread Synchronization to achieve Real Time Control	61
Chapter 5	64
MODEL EXAMPLES	64
5.1 Real World Application	64
5.2 GUI Example	64
Chapter 6	73
CONCLUSIONS AND FUTURE WORK	73

6.1	Conclusions	73
6.2	Future Work	74
REFERENCES		76
APPENDIX		79

LIST OF FIGURES

Figure 1.1: Industrial Robot System	3
Figure 2.1: MIDI Connectors	15
Figure 2.2: A Simple MIDI System	16
Figure 2.3: An Expanded MIDI System	18
Figure 2.4: MIDI User Interface (Cool Edit Pro)	19
Figure 2.5: Single Instrument Activity	21
Figure 2.6: MIDI Event List	23
Figure 3.1: RIDI Implementation Environment Overview	28
Figure 3.2: Simplified Proposed System	32
Figure 3.3: Mitsubishi RM501 Robot Arm	33
Figure 3.4: Schematic Representation of RM501	34
Figure 4.1: Layered Structure for Developing Screens	43
Figure 4.2: GUI Data Flow	45
Figure 4.3: Orientation of Robotic Arm at a Common Point (x1, y1, z1)	45
Figure 4.4: RIDI Kinematics Screen	47
Figure 4.5: Layered View of the RIDI Kinematics Screen	48
Figure 4.6: RIDI Forward Kinematics Input Screen	49
Figure 4.7: Functionality of Robot Arm	50
Figure 4.8: Functionality of Combo Box	52
Figure 4.9: Checking on Non-Numeric Entries	53
Figure 4.10: Layered Structure for Forward Kinematics Screen	53

Figure 4.11: RIDI Reverse Kinematics Input Screen	55
Figure 4.12: Layered Structure of Reverse Kinematics Input Screen	55
Figure 4.13: RIDI GUI Activity Screen	56
Figure 4.14: Control Panel of RIDI Activity Screen	57
Figure 4.15: Layered Structure of the Control Panel	57
Figure 4.16: Graphical Representation of Each Joint	58
Figure 4.17: Specification of the Joint	58
Figure 4.18: Prismatic Joint Activity Panel	59
Figure 4.19: Scrollable Coordinates Activity Area	60
Figure 4.20: Layered Structure of the Main Activity Screen	61
Figure 5.1: Assembly Task Layout	65
Figure 5.2: RIDI Input Screen for Assembly Example	66
Figure 5.3: Initial Activity Screen for Robot 1	67
Figure 5.4: Initial Activity Screen for Robot 2	68
Figure 5.5: Intermediate Activity Screen for Robot 1	69
Figure 5.6: Intermediate Activity Screen for Robot 2	70
Figure 5.7: Final Activity Screen for Robot 1	71
Figure 5.8: Final Activity Screen for Robot 2	72

LIST OF TABLES

Table 3.1: Range of Different Joints

35

Chapter 1

INTRODUCTION

1.1 Overview of ROBOTICS

In the past decade the problem of productivity has attracted international attention. One of the possible means of increasing productivity is to modernize manufacturing facilities by means of automation. With pressing need for productivity and quality end products, industry is turning more and more toward computer-based automation, specifically the use of computer controlled robots. Most automated manufacturing tasks are done by special purpose machines designed to perform predetermined functions in a manufacturing process. The inflexibility of these machines makes the computer-controlled manipulators more attractive and cost effective in various manufacturing and assembly tasks. Basically there are three distinct types of manufacturing processes

1. Continuous process- high degree of automation with minimum flexibility.
2. Mass production- mainly utilizing transfer lines to handle high volume discrete part manufacturing.
3. Batch production- handles mid to low volume manufacturing of discrete parts.

The efficiency level of batch production is above average. In order to improve the efficiency of batch manufacturing, programmable automation is utilized. The industrial robots are often utilized [1]. A robot is a reprogrammable multi-functional manipulator designed to move material, parts, tools or specialized devices, through variable programmed motions for the performance of variety of tasks [2]. With this

definition, robots must possess intelligence, which is normally due to the mini-micro computer associated with its control unit.

Robotics is the study of basic organization and operation of intelligent computer-based robots. Within the next 10 to 15 years, with the increasing use of computers, we can expect further practical application of computer-controlled robots in manufacturing and other industries.

1.2 Industrial Robots

An industrial robot is a general-purpose manipulator consisting of several rigid links connected in series by revolute or prismatic joints. One end of the chain is attached to a supporting base while the other end is free and attached with a tool to manipulate objects or perform assembly tasks. The motion of the joints results in relative motion of the links. Mechanically, a fixed robot is composed of an arm (or main frame) and a wrist subassembly plus a tool. It is designed to reach a workpiece located within its work volume. The work volume is the sphere of influence of a robot whose arm can deliver the wrist subassembly unit to any point within the work volume. Today's industrial robots, though controlled by mini/micro computers, are basically simple positional machines. They execute a given task by playing back prerecorded or preprogrammed sequences of motions that have been guided or taught by a user with a hand held control/tech box. Moreover the robot is equipped with few or no external sensors (both contact and noncontact) for obtaining the information vital to its working environment. The robot manipulator task can be represented by the system shown in Figure 1.1. The figure exhibits a diagram of a closed loop

system, wherein the feedback from the robot is obtained for tracking the ongoing task.

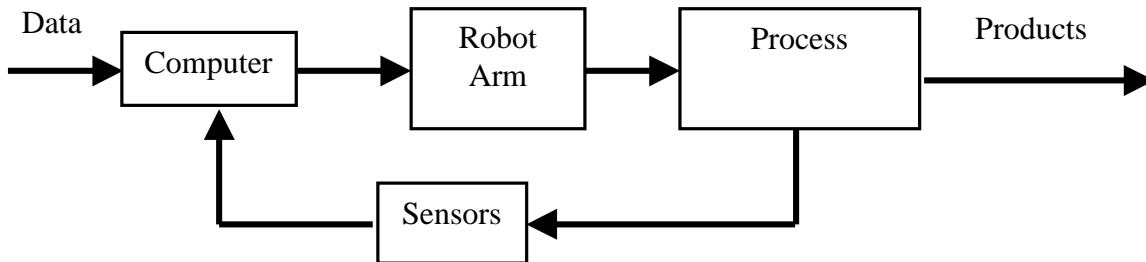


Figure 1.1: Industrial Robot System

One of the key issues in the field of robotics is that of controlling the robots. Repetitive jobs in labor intensive operations are performed by computer controlled machines supervised by a highly trained group of operators who set up and program each job, modify procedures to fit the particular circumstances, change over for new batches or models, maintain the equipment, and cope with breakdown and stoppages. The capabilities of such programmable systems differ sharply from those of conventional "fixed" automation systems, in which special purpose machines are designed to perform specific repetitive tasks. This is particularly important where production runs are small and where different models may have to be produced frequently [3]. Manipulation of workpieces and tools for material handling and assembly jobs requires many basic operations to be performed. In a completely structured environment it may be possible to perform all these operations in a feed-forward manner with no sensory control or correction needed. But in reality this is not the case. It therefore, becomes unavoidable for the operations to provide feedback to the controller, where this feedback assists the user to control the robot in

case of an unexpected situation. With the availability of the computer the operator can program the operation of the robot in either an off-line mode or an on-line mode.

1.2.1 Off-Line Programming of a Control Computer

Off-line programming can be defined as the task of programming the robot through the use of remotely generated point coordinated data, function data, and cycle logic. In this case the operator can develop a program off-line and then enter it into the control computer for operational use. When off-line programming is used, the robot system remains in operation while a new program is being generated. Through off-line programming, robot becomes more closely integrated into the total manufacturing system, since information from the CAD/CAM database is shared with other elements of the system. Off-line programming defines and documents robot instructions better than manual teaching. Advanced robot languages permit the user to specify sequences of movements and operations at a keyboard terminal connected to the computer. Software aids available with many of the languages make programming faster and more accurate. Programs are readily modified at the keyboard by using editing routines and symbolic data references. Off-line programming languages provide for more flexible use of sensors and adaptive control. Off-line programming languages are classified into two types as either implicit or explicit. Explicit types of languages are VAL, EMILY, SIGLA, and WAVE. Implicit types are AL, ROBOT APT, AUTOPASS, RAPT, and MAL [4]. Explicit types permit detailed control over the manipulator actions with direct

commands such as open and close, whereas in implicit types user describes the tasks to be performed rather than detailed robotic motions.

1.2.2 On-Line Programming of a Control Computer

Many robots have been equipped for on-line programming. In on-line programming the operator is able to program the robot using teach or an on-line mode, running the robot through the desired movements and then recording the moves for later reproduction or playback. It is necessary for the operator to teach the arm by guiding it through its assigned tasks. Generally this teaching or programming is defined into three tasks. The first task is the identification of the coordinates of points at which some action is to occur. The second task involves stating the functions to be performed at a point or along a path. The third task involves defining the logic of the operation or cycle. This involves determining what path the arm should take under specified conditions. Conditions can be represented by status of external signal or internal flags, or by the magnitude of the internal or external variables. An on-line mode is satisfactory for many applications but it becomes tedious when hundreds of points must be individually programmed.

1.2.3 Applications of Industrial Robots

Many commercially available industrial robots are widely used in manufacturing and assembly tasks such as simple material handling, spot/arc welding, parts assembly, spray painting, and loading and unloading numerically controlled machines, in space and under sea explorations, in prosthetic arm research, and in

other related areas. The robot offers an excellent means of utilizing high level technology to make a given manufacturing operation more profitable and competitive. In U.S. metal working industry, machine loading and unloading appears to be the biggest single area of application for robots [4].

But what drives this work is the growing use of robotics in assembly operations. Assembly robot is getting intense attention in the U.S. and Japan. In assembly, the driving impetus is economics, and multi-shift capabilities of robots revolutionize one-shift assembly operations. Assembly of manufacturing goods accounts for 53% of total production time. Usually one-third of a manufacturing firm's workforce is involved in assembly tasks [5]. A robot is ideally suited for light assembly tasks that are capable of good repeatability and accuracy over long operating times [6]. Mid volume batches of a product and mixed batches of similar models of a product can be assembled profitably by machines that are adaptable and programmable (an industrial robot). Apart from these applications in the field of manufacturing and assembly, robots can also be used for other non-manufacturing tasks in various technical and scientific fields. Robots are also being utilized in the areas of robot surgery, minimum invasive surgery, laparoscopic surgery, and remote surgery [7]. Robotic applications now require multiple robots working in a coordinated manner, something similar to an orchestra.

1.3 Need for Research

In case of on-line programming, the task becomes tedious if we want to program for hundred of points individually. For example in the aerospace industry there are

hundreds of holes that must be drilled in sheet metal and many rivets that must be properly placed [8]. It is inefficient for the operator to manually program these points using current on-line teaching techniques. With off-line programming this drawback can be overcome, but then off-line mode has its own unique concerns. They are [9]:

- A need to adjust for inaccuracies associated with the arm.
- A way of aligning various coordinated reference frames.
- A verification procedure.

In addition to above drawbacks, one must synchronize the operations of multiple robots. This work presents a unique approach and coordinated control of multiple robot arms in a real time. It will be possible to control the robots or robotic devices from a microcomputer with a greater accuracy, better control and proper verification.

1.3.1 Real Time Environment

Most of us are familiar with the systems in which data needs to be processed at regular and timely intervals. For example [10], an aircraft uses a sequence of stream of accelerometer pulses to determine its current position. In addition, other systems require a rapid response to events that occur at irregular rates, such as a temperature failure in a nuclear plant. Such events require real time processing. Some of the key points with regards to real time system are

- Real time system are those in which timeliness is as important as the correctness of the outputs.
- Performance estimation and reduction are crucial in real time system
- Real time systems do not have to be "fast systems."

The focus of this research is to develop a Robotic Instrument Digital Interfacing (RIDI) system for real time coordinated control of robotic devices. The control of multiple robotic devices needs to be accomplished in real time. The RIDI system should be able to respond to the commands given by the user through the Graphical User Interface (GUI), within a certain time or the user might lose track of the robot motions and result in a failure to accomplish the required task. Depending on the application for which the RIDI system is used, it can be classified as soft real-time or a hard real-time system. For instance, if the RIDI system is used for assembly operations, the failure to meet performance requirements will result in its degradation and not the total destruction of the system, it is said to be a soft real-time system. On the contrary, the failure of the RIDI system to meet the performance requirements when used in the field of robotic surgery or laparoscopic surgery can result in total system failure and might even result in loss of life. In this case, the RIDI system falls into the category of hard real-time system. Since the RIDI system, though classified when used in different situations, follows the same architecture to accomplish the designated tasks, it becomes imperative on our part to treat it as a hard real-time system. The main purpose to develop RIDI was to establish a general purpose framework for controlling multiple robotic devices.

1.3.2 Real Time Robotic Systems

Keeping in mind the timeliness, correctness, and other salient features that real-time systems have to offer, there is significant ongoing research into the real-time control of robotic devices. In order to isolate and define the fundamental research

issues in coordinated multiple robot systems, and to recommend areas for future work, a workshop was organized by A. J. Koivo (Purdue University) and G. A. Bekey (USC) under the support of National Science Foundation [11]. In this workshop stress was laid on the planning, control, and applications in coordinated control of multiple robot manipulators. This work however, did not discuss the real time coordinated control of robotic devices or manipulators. Another work in the area that dealt with real time control of robot manipulators in presence of obstacles was conducted by S. Kheradpir and S.J. Thorp [12]. This work included real time control of manipulators and applied the concepts of obstacle avoidance strategy (OAS) and state dependent control constraints (SDCC). This work presents an alternative approach to the problem of obstacle avoidance for a Cartesian Coordinate Robot. These two applications though involve the real time coordinated control of robots in a real time environment, they are quite different from RIDI. There is a need to achieve a coordinated control of multiple robotic devices, may it be manipulators or different robotic joints or even different robots, in a real-time environment.

1.3.3 Extensions to the field of MIDI

This work focuses on the development of a MIDI (Musical Instrument Digital Interfacing) based discipline for the control and application of multiple robots. MIDI is a proven and phenomenally successful discipline. Its capabilities provide opportunities far beyond the world of music. There are many areas other than music where the concept of MIDI has been the basis of development. The field of robotics presents particularly attractive potential. Just as MIDI has had a profound influence

in the world of music, a similar discipline for robot devices offers extraordinary opportunities. Intrinsic values achievable through this work include novel approaches to interconnecting robotic devices, coordinated control of multiple robotic devices, and a unified implementation environment. This unified application environment would be for tasks with demanding real time coordination and control. Such an environment could be used for areas including music, robotics, surgery, process control, instrumentation, etc.

1.4 Problem Statement

RIDI provides a novel runtime environment that supports the building and deployment of distributed robotic devices. One of the major components of the prototype RIDI system is the user interface. This interface is aimed at providing extensive support for monitoring multiple device activity in a real time system. The aim of this work is to develop the user interface. Software developers face many difficult decisions when building new applications, not the least of which is the design of the graphical user interfaces [13].

With the exception of code for the device-specific interfaces, the prototype RIDI implementation will be in coded in JAVA. JAVA was selected for developing this interface application because of the various features it provides for building effective User Interface (UI) and also the application can be made to run on any machine irrespective of the platform that the development machine supports. The UI is aimed to receive data from the user regarding a particular coordinated set of motions of multiple robots or robotic joints and display the motion of the robot in a

real time environment. This study will test the UI using simulators rather than actual robots. The robotic devices will be simulated to receive data from the simulator and provide feedback to the UI on a real time basis. The user will be able to see the motion of all robots, each on a different channel, in a graphical manner. The user can also view the activity on an individual channel. The user interface is on the lines similar to the MIDI user interfaces already available in the market. Of all the MIDI UI present, two were referred for this work. One of them was developed by Cool Edit Pro. It is a product of Syntrillium Software. The software was downloaded from the web and was studied for development features [14]. Apart from this integrated digital audio and MIDI products developed by Twelve-Tone Systems Inc referred to, as Cakewalk products was also studied [15]. The major difference in the RIDI system than that of the MIDI system lies in the area of feedback. RIDI requires high level feedback, so that the user can know when the task is accomplished or at what position the activity is at a particular moment in time. Since RIDI deals with robots it becomes necessary for this feature to be included since its absence can lead to catastrophic results. This is not the case when dealing with music. An open loop control system will be utilized. Whereas what we aim to achieve in RIDI is a closed loop environment where online feedback is provided to the user.

The problem can be stated as:

1. Study the UI developed for MIDI (Musical Instrument Digital Interfacing).
2. Evaluate key feasibility considerations related to the development, implementation and use of GUI for RIDI.

3. Implement and demonstrate a RIDI User Interface along with its application to control two robot arms.

Chapter 2

LITERATURE REVIEW

2.1 General Overview of Robot Control

The task of robot control system is to execute the planned sequence of motions and forces correctly, in the presence of unforeseen errors [16]. Errors can arise from inaccuracies in the model of the robot, mechanical compliance in linkages, limitations in the precision of computation, etc. At robot level and action is decomposed into a sequence of robot motions and forces in the Cartesian space. At the joint level, these motions and forces are decomposed into parallel joint motions and joint torques. In both Cartesian and joint space, we require precise control of position, velocity, force and torque. To obtain this control, we use feedback and control laws. In case of the RIDI system the orchestrator handles the feedback where as the robot controller deals with the control laws. The different approaches as regards robot control are open loop control, adaptive control, feedforward control and feedback control. RIDI achieves feedback control for obtaining an accurate control of a process during execution of an action. In feedback control the parameter that is being controlled is continually measured, compared to a reference (error calculation), and the action modified according to control law to overcome the error. In the past, there has been extensive research in this area and many different methods have been used to achieve feedback control. One such work aims at using object oriented programming approach, ROBOOP, a robotic manipulator simulation package which is both platform and vendor independent [17]. In an another related work an experimental

dual-arm space teleoperation system was constructed for complex and difficult tasks [18].

2.2 Overview of MIDI Discipline

MIDI is a very efficient method of representing musical performance information, and, it is an attractive protocol not only for composers and performers, but also for computer applications which produce sound, such as multimedia presentations or computer games [19]. The fact that MIDI transmits instruction for a musical instrument to play a particular piece of music rather than the music itself, saves quite a lot of storage space and hence helps in real time coordination of musical instruments. The MIDI data stream is a unidirectional asynchronous bit stream at 31.25 Kbits/sec with 10 bits transmitted per byte. MIDI information is transmitted in MIDI messages, which can be thought of as instructions which tells a music synthesizer how to play a piece of music.

2.2.1 MIDI Interfaces

MIDI interfaces, the hardware that makes MIDI available to an instrument or computer, appear on keyboards, sound cards, computer motherboards, guitars, tape recorders, saxophones, violins etc. there are various MIDI interfaces available in the market today. A site on the web called *Midifarm* lists all the major manufacturers of MIDI interfaces [20]. Among the major manufacturers of MIDI interfaces are MIDIMAN and OCTETDESIGN [21], [22]. The MIDI interface on a MIDI instrument will generally include three different MIDI connectors, labeled IN, OUT, and THRU. The MIDI IN port on the device is for receiving the MIDI messages at

the port. The MIDI OUT connector on the MIDI instrument is for transmitting changed data from that instrument and passed on to other MIDI device. The THRU port is used for transmitting unchanged MIDI messages from that device to other MIDI device. This is illustrated in Figure 2.1. It can be seen that the data coming out from the OUT port is different as the one entering the device from the IN port but the data coming out of the THRU port is same as that goes into the IN port of the device. The MIDI THRU port does not appear on all the instruments. MIDI cables always connect a MIDI output port to a MIDI input port.

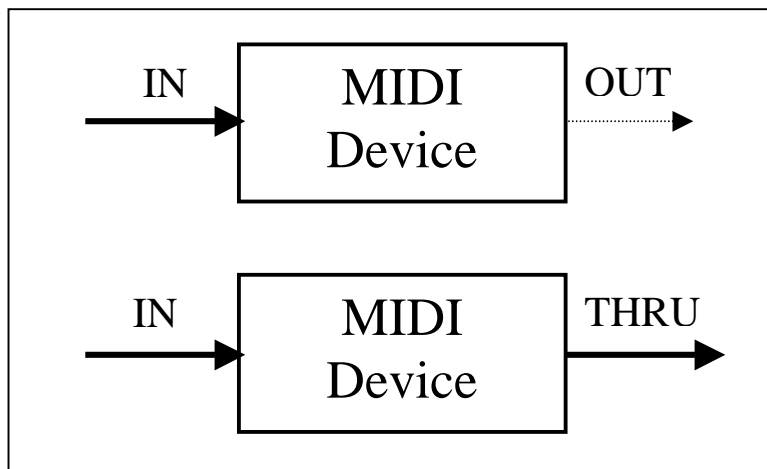


Figure: 2.1 MIDI Connectors

2.2.2 MIDI System

The MIDI data stream is usually originated by a MIDI controller, such as a musical instrument keyboard, or by a MIDI sequencer. A MIDI controller is a device, which is played by an instrument, and it translates the performance into a MIDI data stream in real time. A MIDI sequencer is a device, which allows MIDI data sequences to be captured, stored, edited, combined, and replayed. The recipient

of this MIDI data stream, transmitted by the MIDI controller or sequencer, is commonly a MIDI sound generator or sound module, which will receive MIDI messages at its MIDI IN connector and respond to these messages by playing sounds. Many keyboard instruments include both the keyboard controller and the MIDI sound module functions within the same unit. In these units there is an internal link between the keyboard and the sound module which may be enabled or disabled by setting the local control function of the instrument to ON or OFF respectively. Figure 2.2 shows a simple MIDI system where in the MIDI message from the controller is passed on to a MIDI sound generator. In the system depicted in Figure 2.2, the sound module would have to be set to receive the channel, which the keyboard controller is transmitting on in order to play the particular sound.

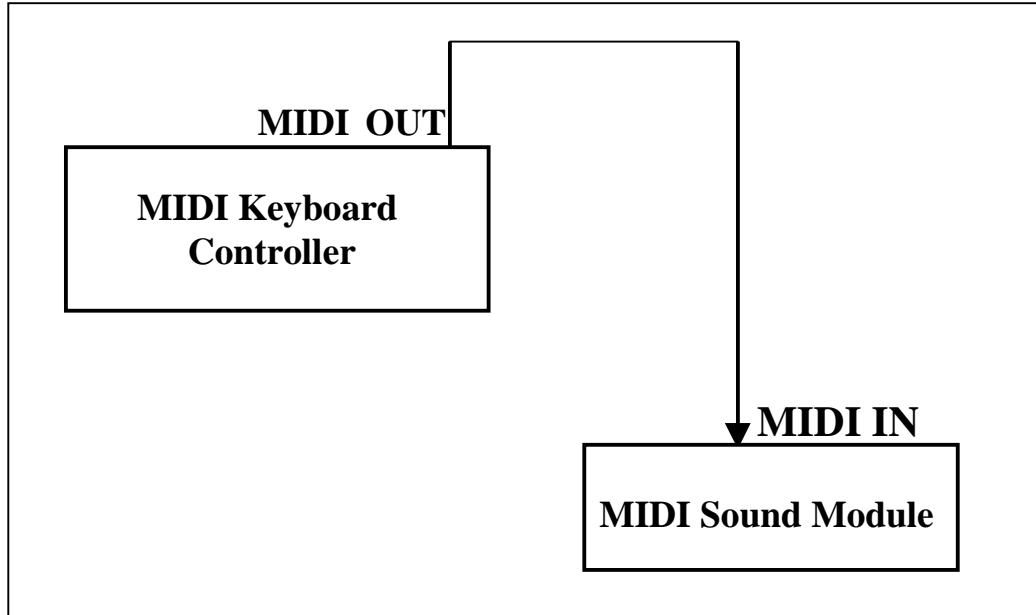


Figure: 2.2 A Simple MIDI System

A single physical MIDI channel is divided into 16 logical channels by the inclusion of a 4-bit channel number within many of the MIDI messages. A musical instrument keyboard can generally be set to transmit on any of the sixteen MIDI channels. A sound module or sound generator can be set to receive one of the specific MIDI channel or channels. In the system illustrated by Figure 2.2 the sound module would have to be set to receive the channel, which the keyboard controller is transmitting on, in order to play a particular sound. Since a MIDI THRU connector transmits a repeated message that was received on the IN connector of that device, it is quite possible to daisy chain the sound modules by connecting the THRU output of one device to the IN connector of next device downstream in the chain. This is illustrated in Figure 2.3. In this case a keyboard controller is used as an input device to a MIDI sequencer, and there are several sound modules connected to the sequencers MIDI OUT port. A composer might utilize such a system to play a particular piece of music involving several different parts where each part is written for a different instrument. The composer would play the individual part on the keyboard one at a time and these individual parts would be captured by the sequencer. The sequencer would then play the parts back together through the sound modules. Each part would be played on different MIDI channel and the sound modules would be set to receive different channels. For example, sound module number 1 might be set to play part received on channel 1 using a piano sound, while module 2 plays the information received on channel 3 using an acoustics bass sound. So when the sound module # 1 receives parts played on all the channels, it will take only the part which is played on channel 1 since it is set for that one. While the remaining parts on other

channels will be sent unchanged through the THRU port to the sound modules down the order.

In this example a different sound module is used to play each part. However sound modules, which are multitimbral, are capable of playing several different parts simultaneously [19]. A single multitimbral sound module may be configured to receive the piano part on channel 1, the bass part on channel 6, and the drum part on channel 10, and would play all three parts simultaneously. The communication method is high speed, asynchronous serial data transmission. Data is transmitted and received as a stream of bytes. For MIDI data, each data item is referred to as an event. The data stream includes all event data for all interconnected devices.

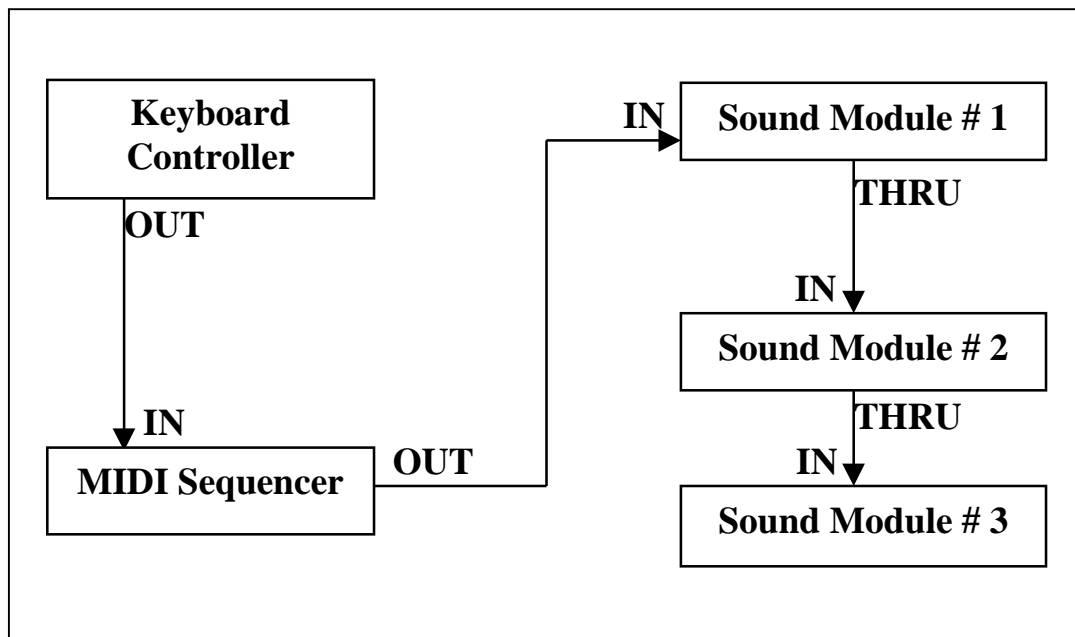


Figure 2.3: An Expanded MIDI System

As illustrated in the Figure 2.3, data for all ports and all channels are transmitted to all network devices. Each specific device, however, responds to only events for its port and channels ignoring the other event data.

2.3 MIDI User Interfaces

The user interface provides interaction between the system and a user. An efficient user interface should be intuitive, have options, should provide interactive access etc. For the successful application of the MIDI system different user interfaces are available in the market. Numerous musical applications have been developed and are readily available for critical evaluation. Figure 2.4 shows a screen printout from the digital audio package Cool Edit Pro, a product of Syntrillium Software [14].

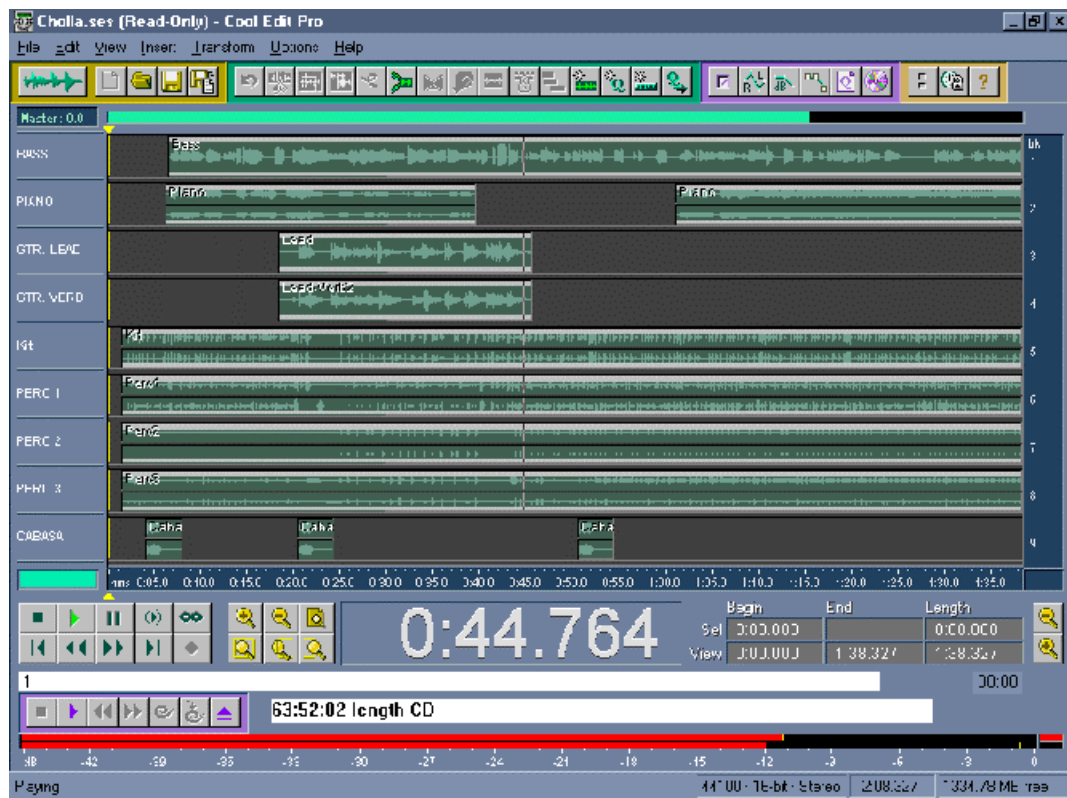


Figure 2.4 MIDI User Interface (Cool Edit Pro)

The above figure includes summary control information for nine musical instrument parts (bass, piano, guitar, etc.). Elapsed time from the start of coordinated rendition is shown by the continuously updated counter, which currently shows 0 min. 44.764 seconds. The moving vertical line and its associated scrolling scale also shows the elapsed time. On the screen are different options and tool bars which help to edit a current rendition, record the rendition, retrieve an old played rendition etc. Numerous pull down menus and toolbar buttons are also included. From this screen one can increase the speed at which the rendition is being played while receiving the feedback right on the screen. The user interface in this screen is dynamic. It changes continuously in real time. In addition to monitoring control and response data for multiple devices in real time, the display can be frozen at any point in time to allow detailed analysis of interactions. Both backward and forward panning is possible. It is possible to isolate data for individual devices and to expand and contract time scales. Control characteristics can be modified, and the impact of changes can be monitored. By simple mouse click on a control button of any of the musical instrument, a user can switch from the display shown in Figure 2.4 to Figure 2.5.

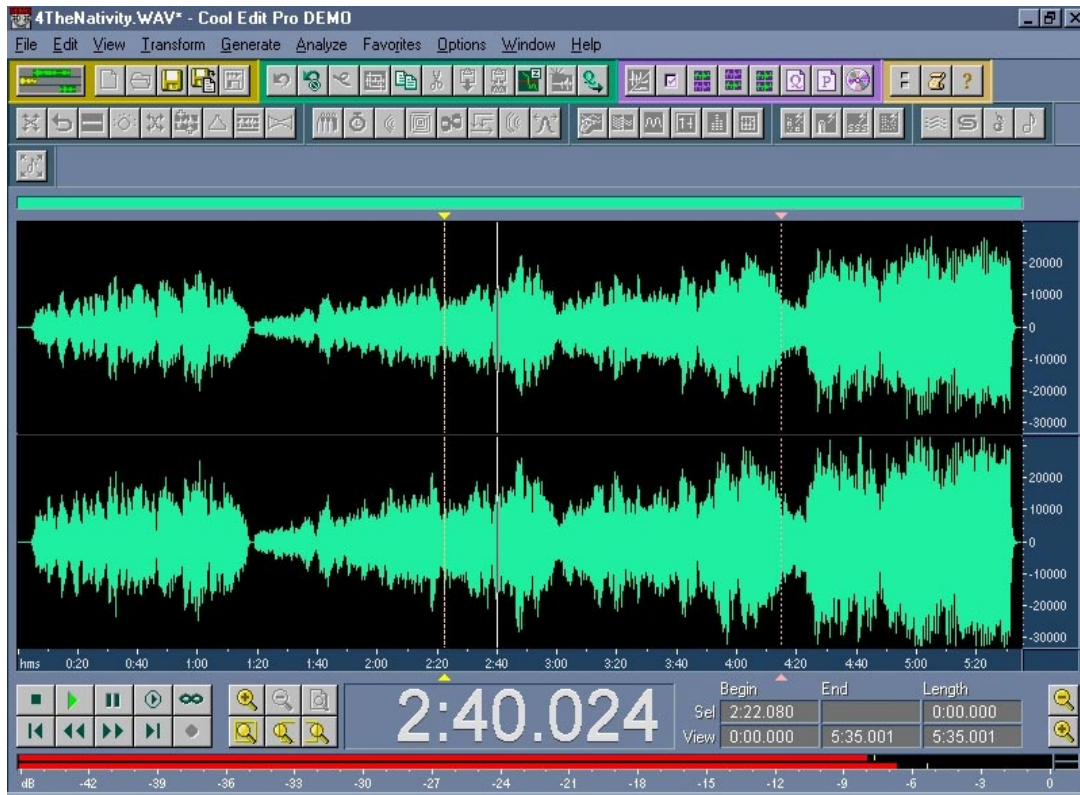


Figure 2.5: Single Instrument Activity

This figure shows summary data as a function of time for a single instrumental part. So Figure 2.4 provides information for multiple musical instruments where as Figure 2.5 is displayed for a selected musical instrument. Thus the user can monitor input and output for multiple robots, and at any point the user can switch the control to an individual instrument. All the different operations that we could perform on the multiple robots on the screen shown in Figure 2.4 can be carried out on the single instrument to which the context is switched. There are other auxiliary displays that are shown by the software, like the spectral analysis display where in the wave frequencies of the audio wave are displayed as function of color. These types of

display options have proven to be quite effective for other various types of analysis on the data transmitted in real time environment.

2.3.1 MIDI Data

As mentioned earlier each interconnected device may receive, respond to, modify, generate, and transmit MIDI data. The transmission method used for MIDI is similar to the one used for computer modem communication. Data are transmitted and received as a stream of bytes. Individual data items may consist of number of bytes. For MIDI data, each data item is referred to as an event. The data stream includes all event data for all interconnected devices. Each encoded event includes data for channel identification and a port identification. Port identification provides a mechanism for simultaneous communication involving multiple network configurations, and channel identifications are used to specify specific networked devices to receive specific event data. As seen earlier, data for all ports and all channels are transmitted to all networked devices. Each specific device, however, responds to only events for its port and channel. All other data are ignored. MIDI event data is very vital in achieving coordinated control many interconnected devices. An overall feel of the MIDI event data can be gained through the study of what is called the MIDI event list. The entire event list includes all of the data needed to produce an orchestrated rendition for the different instrumental parts. Each line of the event list represents data for a single event. Figure 2.6 illustrates one such event list provided by the Cakewalk Pro application software package. Cakewalk Pro is the product of Cakewalk^{CAKE} [23]. This particular figure shows a portion of the event list

for a particular rendition. There are some other softwares that are also available in the market.

Trk	Hr:Mn:Sc:Fr	Meas:Beat:Tick	Chn	Kind	Values
3	00:01:39:10	38:2:000	3	Control	11-Expressio 75
6	00:01:39:10	38:2:000	6	Note	E 4 72 105
7	00:01:39:10	38:2:000	7	Note	A 5 98 1:105
9	00:01:39:10	38:2:000	9	Note	E 5 72 2:105
10	00:01:39:10	38:2:000	11	Note	E 4 72 105
11	00:01:39:10	38:2:000	12	Note	E 3 72 105
6	00:01:40:00	38:3:000	6	Note	E 5 72 105
10	00:01:40:00	38:3:000	11	Note	E 5 72 105
11	00:01:40:00	38:3:000	12	Note	E 4 72 105
6	00:01:40:20	38:4:000	6	Note	E 4 72 105
7	00:01:40:20	38:4:000	7	Note	G#5 98 105
10	00:01:40:20	38:4:000	11	Note	E 4 72 105
11	00:01:40:20	38:4:000	12	Note	E 3 72 105
3	00:01:41:10	39:1:000	3	Note	C#5 111 52
7	00:01:41:10	39:1:000	7	Note	C#6 98 52
3	00:01:41:20	39:1:060	3	Note	D 5 111 52
7	00:01:41:20	39:1:060	7	Note	D 6 98 52
3	00:01:42:00	39:2:000	3	Control	11-Expressio 75
3	00:01:42:00	39:2:000	3	Note	F#5 111 52
6	00:01:42:00	39:2:000	6	Note	A 4 72 1:105
7	00:01:42:00	39:2:000	7	Note	F#6 98 52
8	00:01:42:00	39:2:000	8	Note	C#6 72 1:105
9	00:01:42:00	39:2:000	9	Note	A 5 72 1:105
10	00:01:42:00	39:2:000	11	Note	A 4 72 1:105
11	00:01:42:00	39:2:000	12	Note	A 3 72 1:105
3	00:01:42:10	39:2:060	3	Note	E 5 111 52
7	00:01:42:10	39:2:060	7	Note	E 6 98 52
3	00:01:42:20	39:3:000	3	Note	E 5 111 78
7	00:01:42:20	39:3:000	7	Note	E 6 98 78
3	00:01:43:05	39:3:090	3	Note	E 5 111 26
7	00:01:43:05	39:3:090	7	Note	E 6 98 26

Figure 2.6: MIDI event list

Figure 2.6 illustrates the list of the events that occur to play a particular portion of a rendition. During a rendition of the work, data for all events are transmitted sequentially to all participating instrumental parts. Here the data for individual parts are identified by the channel number, which is shown in the table as Chn. The cursor indicated by the square box shows the current event, and the slide bar on the right indicates the current position in the event data file. As the rendition is generated or played in real time the cursor and the sliding bar along with the window contents scroll downwards. The second column shows the time at which that particular event is taking place. Apart from the information regarding to what channel is being played it also provides additional information as to whether the particular message is a control change message or a note message.

2.3.2 MIDI Protocol

MIDI is a serial protocol. That is, each bit of information flows along a wire and one bit follows the next bit in sequence. Each byte is sent at 31,250 baud and occupies 10 bits. Among these 10 bits the first bit is the start bit, the last bit is the stop bit and the remaining 8 are the data bits [19]. A MIDI message is a package of data that fully specifies an event. Most messages are one, two or three bytes in length, although some of them may be longer. Generally a three-byte message takes nearly a millisecond to transmit. MIDI messages are broadly classified into Channel messages and System messages. Channel messages are those which apply to a specific channel and the channel number is included in the status byte of these messages. System messages are the ones, which are not channel specific, and no channel number is indicated in their status byte. Each MIDI message regardless of its length contains a single status byte and zero or more data bytes. The status byte specifies what kind of event it is. There is considerable redundancy in a typical stream of MIDI messages. For example Note Ons on given channel are sent, followed some time later by corresponding Note Offs. There are number of events which are sent simultaneously and since each byte is sent serially, it is required to squeeze out some of the redundant data. MIDI protocol provides a simple and effective way to do just that by a feature called *running status* [24]. Running status is a type of data compression so that we can omit the redundant status bytes. It works on the principle that a new status byte must be sent whenever the status changes, either because of a new message type or the same message type on a new channel. Running status is used only for channel messages and is terminated when the status

byte changes. However in practice, its a good idea to refresh the status byte periodically, even when it is not strictly required [25]. This is so because if a receiving device were to come on line in the middle of the sequence, or lose track of the status byte, it would refuse to make any sound until the status was refreshed. Generally updating once every 16 or 32 events is adequate.

2.4 MIDI Show Control & MIDI Machine Control

Apart from musical data the MIDI messages also send data not pertaining to musical information. This data is related to the devices, which are aimed for controlling along with the MIDI messages. These devices are the ones related to features like pyrotechnique displays, sound, lighting etc which support the musical rendition going on. This feature of MIDI is referred to as MIDI show control (MSC). Like these features it is quite possible to control the robots along with the musical information send along different channels. In this case the robotic information will be the auxiliary data send along with the musical data and the end task accomplished in such a manner will be referred to as MIDI machine control (MMC). But this work does not aim at accomplishing the coordinated control of robotics devices by sending MIDI messages along different channels. The primary objective of this work is to develop a system, which will control devices (robotic) on a real time basis. Hence instead of sending MIDI data to control the robots the final system will require the user to enter robotic information into the user interface and receive the feedback of the robotic motions on the user interface. So one doesn't need the knowledge of the fundamentals of MIDI show control.

As MIDI is an established discipline in itself and has great potential for musicians and composers in the same way the coordinated control of robotic devices through RIDI will have extensive use in the various fields like medicine, automation etc. The proposed system aims at controlling multiple robotic devices using the concepts used in MIDI. This work aims at developing the user interface for the system. In order to validate the user interface simulator logic will also be developed. The interface will be developed using JAVA.

Chapter 3

Robotic Instrument Digital Interfacing System

3.1 Introduction to RIDI

The overall objective of this work is based on the concept of developing a discipline of Robotic Instrument Digital Interfacing (RIDI) similar to MIDI. Aspects of the RIDI interface include the capture and use of robots control action sequences, action sequence creation and evaluation, control of single and multiple robotic instruments, integration, real time considerations, and application implementation support. This work aims to lead to the definition, implementation, demonstration, and evaluation of a prototype discipline for the control and coordination of robotic instruments. The RIDI system is primarily responsible for coordinated control of multiple robotic devices in a run time environment. An overview of the prototype system is shown in Figure 3.1. Included in this figure are the robots being controlled and the system that controls the robots.

3.1.1 Key Features of RIDI System

The following are the key features of the RIDI System.

RIDI Data Flow

The Orchestrator

The Communications Network

The Robot Interface

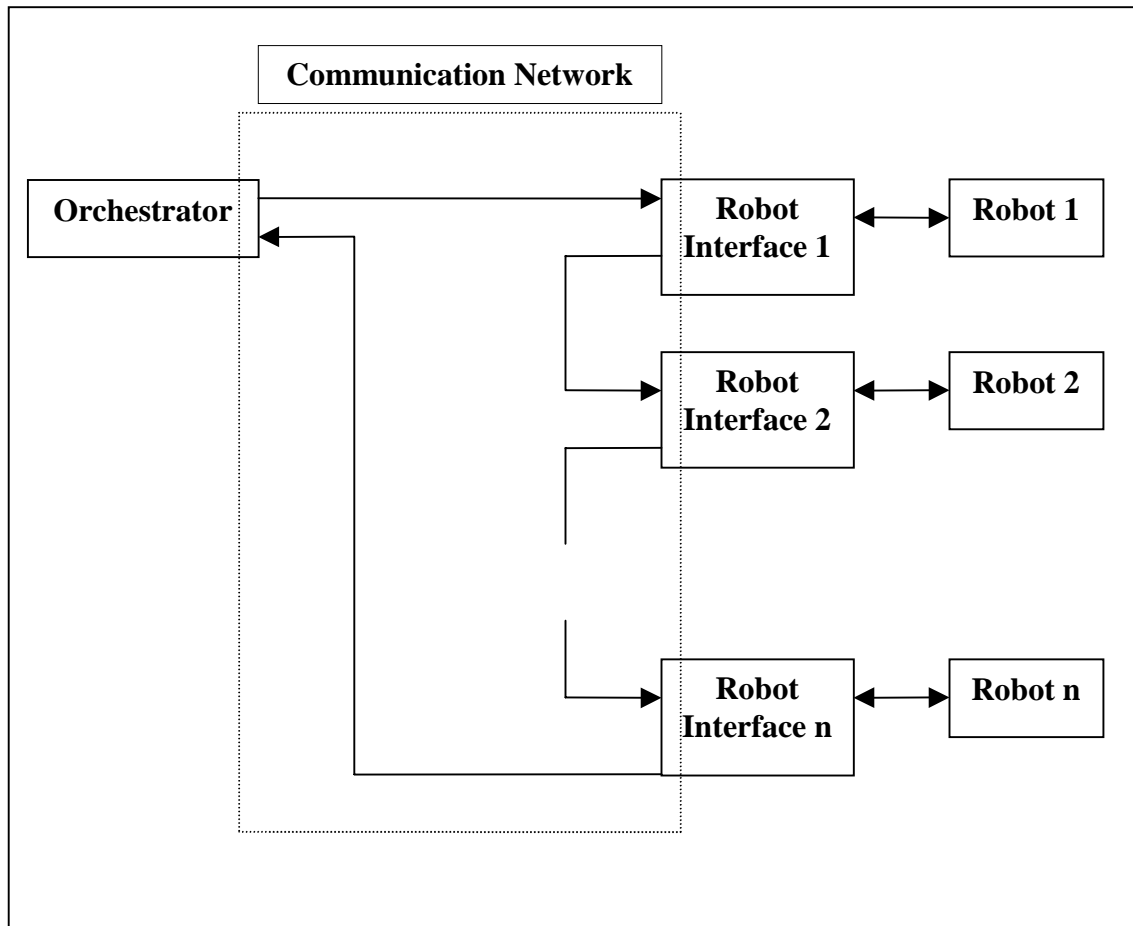


Figure 3.1: RIDI Implementation Environment Overview

The RIDI Data Flow

As shown in the Figure 3.1 the data from the orchestrator pertaining to different robotic devices is transmitted to the robot interfaces via the communication network. The initial robotic interface accepts only that data that is pertaining to the device for which it is the interface. The rest of the data is passed on to the interfaces of the other robotic devices down the line. So eventually each of the robotic interface will receive data pertaining to the device it is connected to. The robot will send this

received information to the robot controller, which will eventually run the robot. The robot will give feedback to the robotic interface about its motion through the robot controller. Each robot interface will receive feedback from its respective robot. This feedback from each robot will finally be transmitted back to the orchestrator. So the user interface which is the responsibility of the orchestrator will show the movement of the robot on receiving the feedback.

The Orchestrator

As was used in previous discussions related to MIDI capabilities, the term orchestration refers to the combined performances of many instrumental parts coordinated in real time [7]. A similar meaning applies in relation to robotic instruments. Completion of comprehensive tasks may depend on many tasks performed by various specialized robotic instruments. A simple example to explain this will be that of a robot picking up parts from a conveyor and placing them in a box [26]. This task decomposes into the following actions.

1. Place an empty box at the packing location,
2. Repeat the following till there are no more parts on the conveyor,
 - When the part reaches the end of the conveyor, pick the part and place it in the box.
 - When the box is full, remove the full box and get a new empty box and place it at the packing location.

Next each action is decomposed into a sequence of robotic operations.

In case of RIDI system the MIDI event data encoding is directly applicable. Data for all participating robotic instruments are intermingled and all of the data is available to all the participating robotic instruments. The orchestrator is the area where the system takes input from the user through a console. The console interacts with the user through a graphical user interface developed for coordinated control for multiple robotic devices. The data submitted by the user related to all the robotic instruments is then analyzed and edited accordingly, and forwarded to the next phase of the system. For example, if a simple pick and place task involves three motions from robot 1 and two motions from robot 2. Then the corresponding movements of the joints of these two robots will be analyzed according to the activity specified on each channel number and sent to the networking phase. So the orchestrator provides the user interface for the RIDI control system. It also provides the logic for coordinated control of the robots. For this implementation, the orchestrator is an IBM compatible PC computing system [27]. A RIDI interface connection is used for connection with the communication network. Also the code for network communications support runs on the orchestrator PC.

The Communications Network

This is the phase, which comes after the orchestrator and before the robot interfaces. The role of a communications network is to provide data communication between the orchestrator and the robot interfaces. RIDI cables are used to daisy chain connect the orchestrator with each of the robot interfaces. For this implementation, standard MIDI cables will be used. All of the robot interfaces will receive data

transmitted by the orchestrator. Feedback from the robots to the orchestrator is supported by closure of the daisy chain to form a ring-structured network. The orchestrator provides the master link connection for the ring.

The Robot Interfaces

A separate interface is used for each robot connected to the communications network. For this implementation, each robot controller is an IBM compatible PC. Each robot controller includes RIDI interface for connection with the communications network and a parallel connection for communication with the robot controller. Extension of the boundary between the communication network to include a small portion of each robot interface indicates that each robot interface PC includes code for network communications. Each robot interface executes code, which provides the logic for interaction between the RIDI data stream and the participating robot. The code for each robot interface is specific for that particular robot and possible data for control of each robot are a characteristic of the robot.

3.1.2 RIDI Graphical User Interface

The RIDI GUI resides in the orchestrator of the RIDI system and is responsible for interaction between the user and the robotic instruments, which are the subject of control. The GUI is an important part of the system as it is the means for the user to give instructions to the robot and also to receive the feedback from the robot about the execution of the instruction in a real time mode. The GUI is developed in JAVA and is currently being developed for only two robots each with 5 degrees of freedoms. Though a part of the whole system this interface is validated for

its effectiveness independently. For that purpose a simulator is developed along with the user interface. The user interface transmits information to the simulator, which provides feedback to the user interface. Figure 3.2 shows the simplified diagram of this work.

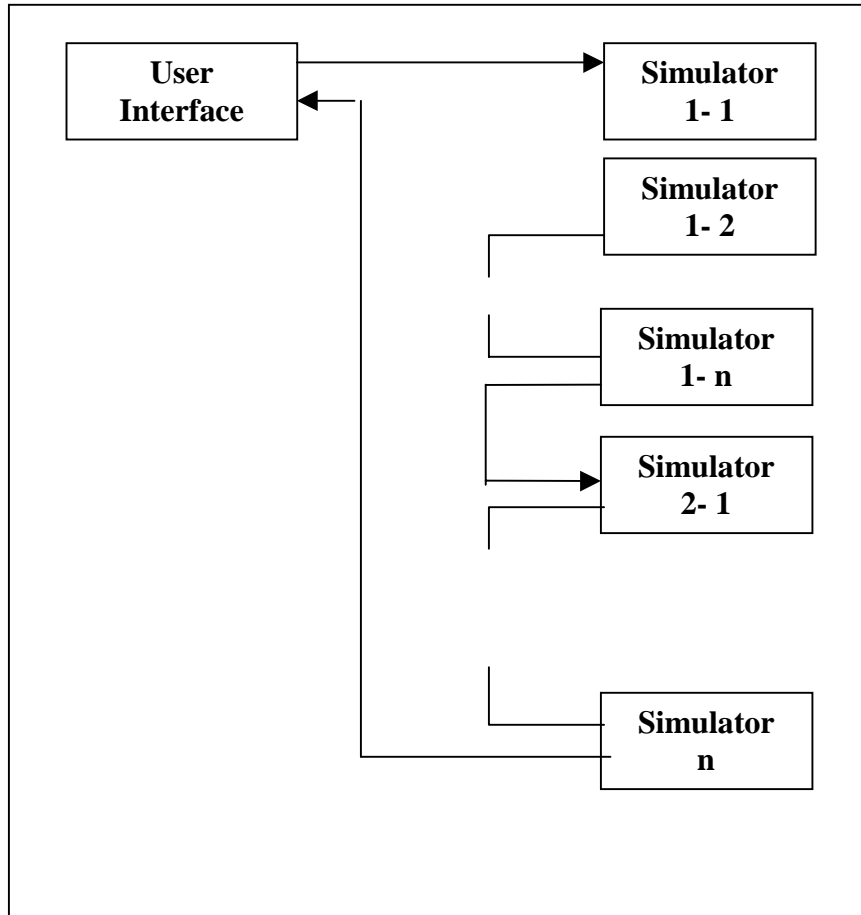


Figure 3.2: Simplified Proposed System

In this simplified system, the receiver of the data is not a robot but a simulated program, there is no need to have a communication network to transmit the data. Also, in this case there is one simulator for each joint of each of the robots. For

example simulator 1 is for the shoulder joint 1 of robot 1. This is supported by the fact that though a particular joint for both the robots are functionally similar in nature they will be operating independently. Apart from exhibiting real time motions of different robotic joints in a real time mode the GUI also indicates the position of the end effector in the coordinate space system.

3.2 The Robot

The Mitsubishi RM501 is a five degrees of freedom bench top robot, suitable for handling small lightweight tasks. It consists of a body (waist) mounted on a base, a shoulder, an elbow, a wrist and a gripper as the end effector, Figure 3.3.

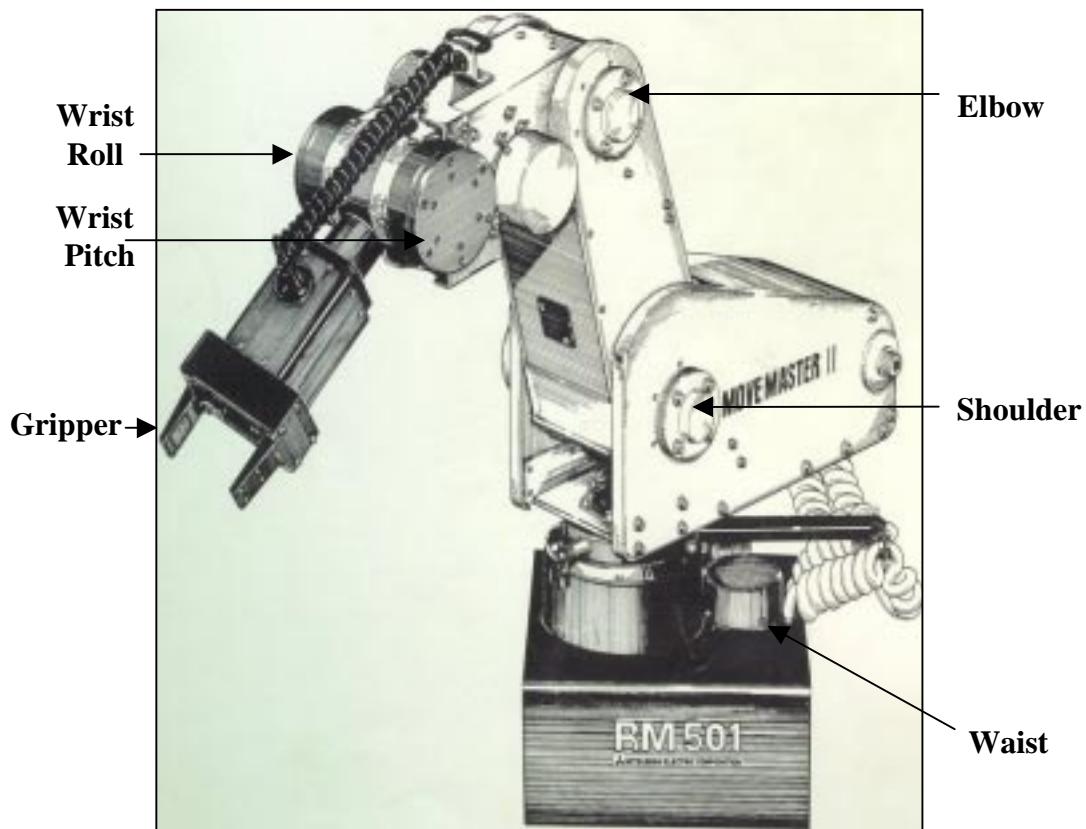


Figure 3.3: Mitsubishi RM501 Robot Arm

The robot is controlled by a drive unit, which can be interfaced to a personal computer. Motion control can be achieved under program control or via the teach pendant. Interface to an external device can be implemented through a serial or parallel port [28]. The movements in each joint in the robot are controlled by stepper motor pulses specified by each joint. The joint parameters denoted by θ_1 , θ_2 , θ_3 , θ_4 and θ_5 corresponding to the waist, shoulder, elbow, wrist pitch and wrist roll respectively as shown in Figure 3.4. Programming of RM501 is done by outputting the appropriate ASCII character string over the communication port.

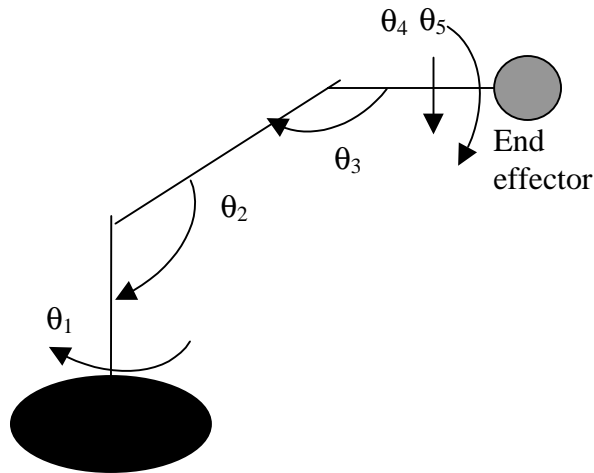


Figure 3.4: Schematic Representation of RM501

ITEM	RANGE
Structure	Five dof vertical multi-joint type
Waist	300⁰
Shoulder	130⁰
Elbow	90⁰
Wrist Roll	360⁰
Wrist Pitch	180⁰

Table 3.1 Range for Different Joints

Table 3.1 shows the range of RM501 joints. The primary reason for the use of RM501 for this study is its availability. However the RIDI system will be developed in such a manner that RM501 could be replaced by another, more accurate, five degree of freedom robot arm without causing any significant changes to the overall system. The RM501 has a gripper as its end effector. This has a simplest type of grippers, which resembles a pair of pliers, with two jaws rotating around a common pivot.

3.3 Extension of MIDI Capabilities to Robotic Applications

As a realistic approach to evaluating the significance of this research opportunity, the following discussions centers on relating a limited number of fundamental MIDI capabilities to robotic application opportunities. In relating MIDI capabilities to robotic application opportunities, a conceptual relationship between

MIDI instruments and robotic instruments exist. Each instrument, whether it is a MIDI or a robotic instrument, is characterized by its functional capabilities. The functional capabilities of an organ differ from those of a drum set. The functional capabilities of a welding robot differ from those of a surgical robot. Each instrument, however, is capable of performing a defined set of functions. Also, the functioning of each instrument depends on precise control sequences, which may involve demanding real-time requirements. Just as there is a distinction between MIDI instruments and MIDI devices, a distinction between robotic instruments and robotic devices can be envisioned. A MIDI instrument is responsible for producing sounds associated with a particular instrument part. MIDI devices include MIDI instruments as well as devices such as storage and retrieval units, sequencers, and controllers. Similarly, robotic instruments perform real-time robotic functions, and robotic devices include robotic instruments as well as devices, which provide various types of control and coordination functions such as for motion control. The following discussion relates only a representative few of MIDI's extensive capabilities to potential robotics application opportunities. Even though limited in number, the specific capabilities discussed do identify some of the many potential opportunities offered by the proposed RIDI system.

Digital Encoding of Event Data

A central feature of the MIDI discipline is the capability of encoding instrument performance data. In essence, this capability constitutes a language that serves many functions. It provides a means for capturing performance data, for

generating performance sequences, for editing and modifying event sequences, and for controlling devices. Performance data is encoded digitally as event sequences. Each encoded event is characterized by the identification of a particular event that can be executed by a particular instrument, attributes for the event, and a channel number. This feature is essential for such functions such as performance data storage, retrieval, communication, and use.

Single Instrument Performance

The feature of capturing and use of event sequence data for a single musical instrument is applicable to robotic instruments. A robot instrument (such as welding or a surgical robot) is fitted with the RIDI system. As an operator performs a task using the robotic instrument, all of the discrete events that comprise the task are captured and encoded in a digital form by the RIDI interface. The captured RIDI data can then be used to drive the robotic instrument and thus repeat the original tasks. Approaches other than the capturing of operator performance sequences and the use of RIDI interface are feasible. Most robotic instruments have computer control interfaces. For such instruments, a convenient and practical approach would be to implement a software interpreter that interfaces between RIDI data and the robot controller.

Orchestrated Rendition

Orchestrated rendition refers to the performance of an orchestration consisting of many instrumental parts. This requires interconnection and communication among

participating instruments. Intercommunication requirements may involve various complexities. The MIDI discipline includes amazingly successful interconnection and communication capabilities. Even if not directly applicable to an interconnected robotic instrument environment, these capabilities provide a detailed functional model for development. Apart from interconnection, communication, and real time coordination and control, an additional aspect of orchestrated rendition is overall monitoring with interactive modifications. The MIDI discipline is exceptionally well suited to orchestrated rendition. A coordinated performance can be controlled and monitored interactively. Individual instrumental parts can be added and/or deleted. Extensive methods needed to meet demanding orchestrated rendition requirements are an integral part of the MIDI discipline, and these are directly applicable to the functioning of interconnected robotic instruments.

Support Applications

Support applications are applications that provide specialized, and often unique, support within an overall discipline. These can include specialized software packages, hardware devices, implementation methods, and novel configurations. Extensive support applications have been developed along with, and are now an integral part of, the MIDI discipline. Many of these may be directly useful as a part of a RIDI discipline. Other applications may offer ideas as well as models for development opportunities.

3.4 RIDI Feedback

The most vital function of the RIDI system is the feedback handling. The potential of the systems rests on this functionality. The user can exercise control on the task to be performed by receiving the feedback from the RIDI system. There is no other avenue for the user to keep track of the ongoing task. The feedback mechanism in RIDI system is similar to that of the MIDI feedback mechanism. In a given set of robotic motions there are various important parameters which are required by the computing system for carrying out the task. Based on this information the future working of the system depends. In the GUI, the simulated robotic environment provides the feedback to the GUI through method calls. In an actual scenario it may not be this easy. Currently there is only one machine and so there is no issue of feedback being transmitted from one machine to another before submitting it to the orchestrator through the communications network. In this system the feedback is handled in a much more simple and straightforward manner. For example when the shoulder joint of the robot is moving through a given angle, numeric data is sent back to the GUI about the position of the robot and the GUI checks for the end position or end angle for the robot. When the current angle and the end angle match, the GUI stops further movement of the shoulder joint. The feedback is also generated in a real time mode to make the system more secure and reliable for complex tasks.

3.5 Application Areas for RIDI System

Robots are used in many diverse applications, from turtle robots in classroom, to welding robots in car manufacturing factories, to the teleoperated arm on the space shuttle. Each application has its own set of problems, and consequently, its own set of robotics requirements [29]. Introducing robot technology into different areas like automation, industry, laboratories, nuclear industry, agriculture, medicine etc. has had a major impact in these fields. The RIDI system can be applied to all of these fields but its feasibility lies in the areas, which require coordinated control of different robotic instrument in a real time system for completion of the task. The three major areas, which can be seen as having potential opportunities for the implementation of the RIDI system, are

- Assembly
- Medicine
- Agriculture

Assembly

One long-term goal of manufacturing technology is the totally automated factory, where a design is conceived at a computer graphics terminal and no further human intervention is required to manufacture the article. Manufacturing in a totally automated environment though feasible is far from reality. Industries require the help of human at different stages of automation. One of these is the assembly stage. There are many tasks in the assembly area which are still quite labor intensive and require constant monitoring by the user while the process is carried on. A system like

RIDI can be very useful at such a stage where a user can control different assemblies on the shop floor from his computer. By RIDI a user can constantly monitor the ongoing assembly process through the feedback that is received by the user interface of the RIDI system. If any unexpected situation arises then the user can very well control the motions of the robot to manage that particular situation. Assembly involves many different processes, inserting one part into another, placing one part over another, tightening up the nut, driving a screw, or snapping fastener into place. In its simplest form assembly means bringing two parts into contact in the correct position, in the correct orientation, and in the correct sequence so that they stay together. Using RIDI one can ensure more accurate assembly operation where the user can have a total control over the operation from his PC.

Medicine

Opportunities for the application of RIDI system can develop in many directions but the ones in the field of medicine are especially inviting. Possible specific application areas include robot surgery, minimum invasive surgery, laparoscopic surgery, and remote surgery. Examples of current work in this and other related areas are numerous [27]. They include work now in progress at development centers such as the Center for Medical Robotics and Computer Assisted Surgery (MRCAS) within the Robotic Institute at Carnegie Mellon University, the Project on Image Guided Surgery at MIT, and the Medical Robotics Project at UC Berkley [27] [31] [32] [33]. Some examples are from the current work which is part of the ARTEMIS (Advanced Robot and Telem manipulator System for Invasive Surgery)

project at the Forshungszentrum Karlsruhe, IAI in Karlsruhe, Germany [30]. The examples include various aspects of laparoscopic surgical procedures. In these examples the surgeon controls the surgery externally without any physical intervention. The robot is used to hold the tools for performing surgery on the patient. The surgery is performed real time and provides a potential opportunity for implementation of RIDI system.

Agriculture

Other area, which provides potential opportunity for RIDI applications, is agriculture. As far use of robots in agriculture is concerned, one of the most successful projects so far has been the development of sheep shearing-robot in Australia [34]. Over 200 sheep have been shorn with fewer injuries than those, which have, occurred with human shearers. Since this shearing is done on live sheep, it becomes quite important for the motion of the robot hand to be accurate in order to minimize the injuries while shearing. RIDI system can be implemented in this area where the robotic instrument used to do the shearing activity will be controlled externally by an operator. He will get the feedback of the activity being carried out on the sheep and in case of any unexpected event he can very well control the robot and save the sheep from injury.

Chapter 4

RIDI - GUI SOFTWARE

4.1 Introduction to RIDI-GUI

The Graphical User Interface for Robotic Instrument Digital Interface was developed on a Windows 98 platform. The GUI is written in pure Java (Version 2. 2) code. No use of Integrated Development Environment is made. This pure Java nature of the code makes the software more portable and platform independent. The software is developed using Swing features in Java, enhancing the software's user friendliness and flexibility. The software is developed in the form of Java frames which are embedded in the web to provide Internet access to the users.

4.1.1 Structure of the GUI Screens

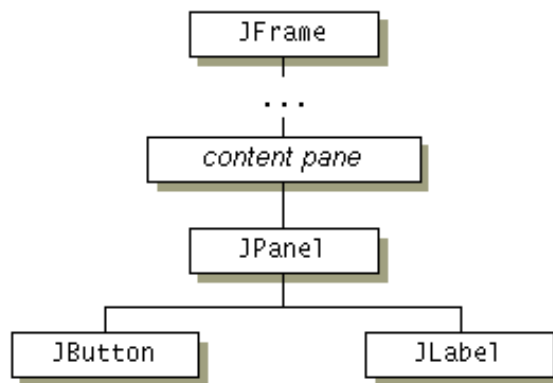


Figure 4. 1: Layered Structure for Developing Screens

The most important feature of the RIDI GUI Software is its layered structure. Figure 4. 1 shows the layered structure used to develop the frames of the software. This diagram shows each container created or used by the program along with the component it contains. This figure gives an idea of how the containers and

components are placed in the specific order for making the screens that appear on the monitor. Even the simplest Swing programs employed for developing the software has multiple levels in its containment hierarchy. The root of this hierarchy is always a top-level container, which in this case is a Swing Frame. All the screens of the RIDI GUI software have the Swing Frames as the top-level containers. The top-level containers provide a place for its descendant Swing Components to paint themselves. Every top-level container indirectly contains an intermediate container know as content pane. All the containers and components of the screen are placed into this intermediate content pane. In the figure above the Swing Panel is placed into the get content pane and the components, a button and a label, go into the Panel. In this way all the GUI screens are developed by adding appropriate components at required places enhancing the look and feel of the software.

4.1.2 GUI - Data Flow

The input data to the GUI is either the θ value of each joint angle or the coordinates of the end effector in a three dimensional coordinate system. The user is required to enter the values for the initial and final states of the robotic arm. The values are entered into the input screen and the user can visualize the motions of the different joints on the main activity screen. If the user enters the forward kinematics values, the control is directly shifted to the activity screen. If the user enters the space coordinates of the end effector as input, then the input is converted into θ values for each of the 5 angles. These θ values are then passed to the activity screen for carrying out the specified task. Figure 4. 2 shows the flow of data in the RIDI-GUI.

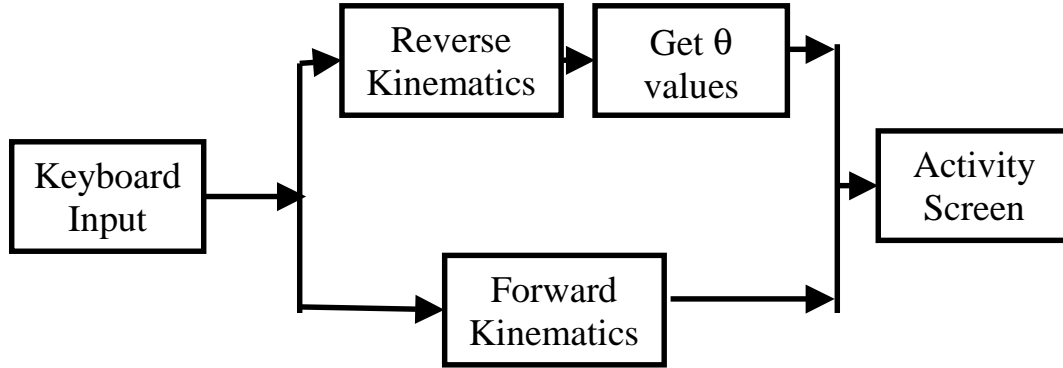


Figure 4. 2: GUI Data Flow

The logic running behind the screens for showing each joint motion though robot specific can be easily extended to other robots with little modifications. Also the logic for converting space coordinates into individual θ angles is robot specific. Apart from giving the three coordinates of the end effector in space, the user is also required to provide the values of joint angles for wrist pitch and wrist roll, θ_4 and θ_5 , to specify the orientation in which the user wants the robotic arm to approach that joint. This concept is explained by Figure 4. 3.

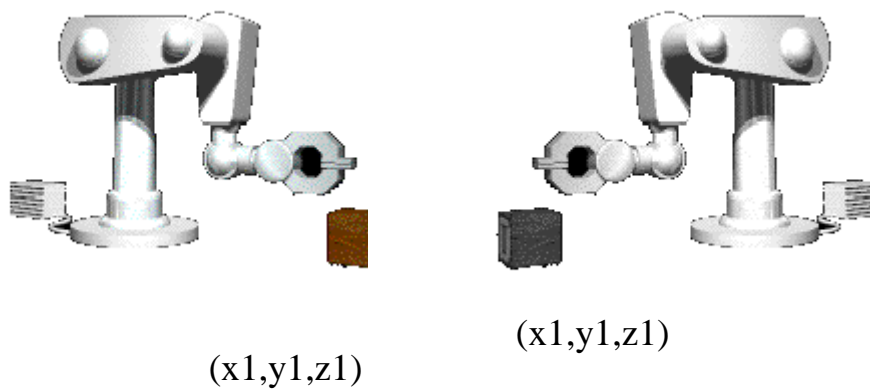


Figure 4. 3: Orientation of Robotic Arm at a Common Point (x_1, y_1, z_1)

Figure 4. 3 shows a robot approaching a same point in 3D-space coordinate system in two different ways. This makes it imperative for the user to provide the values of θ_4 and θ_5 .

4.2 The User Interface

There are four major screens in the RIDI Graphical User Interface. They are

1. The RIDI Kinematics Screen
2. The RIDI Forward Kinematics Input Screen
3. The RIDI Reverse Kinematics Input Screen
4. The RIDI Activity Screen

The user encounters the RIDI kinematics screen and depending on whether he wants to follow the reverse or forward kinematics approach, the screen#2 or screen#3 come up. Once the user has provided the input values to the system, the activity screen is shown to carry out the specified task.

4.2.1 RIDI Kinematics Screen

The RIDI Kinematics Screen is meant for the user to make choice between the two kinematics approaches. The screen is shown in Figure 4. 4. The screen is divided into two distinct sections. The top is meant for the user to select the approach he wants to follow. The two approaches are

- Forward Kinematics, used for calculating the position of the end effector using the individual theta angles as input.
- Reverse Kinematics, used for calculating the individual joint angles using the coordinates of the end effector in space as input.

A Radio Button is used to help user make the selection. Radio Buttons are group of buttons in which, by convention, only one button can be selected at one time. The default selection is set to forward kinematics. There are two buttons which help the user to proceed with the software or terminate it. The next button takes the user to the corresponding input screen. The exit button takes the user out of the system.

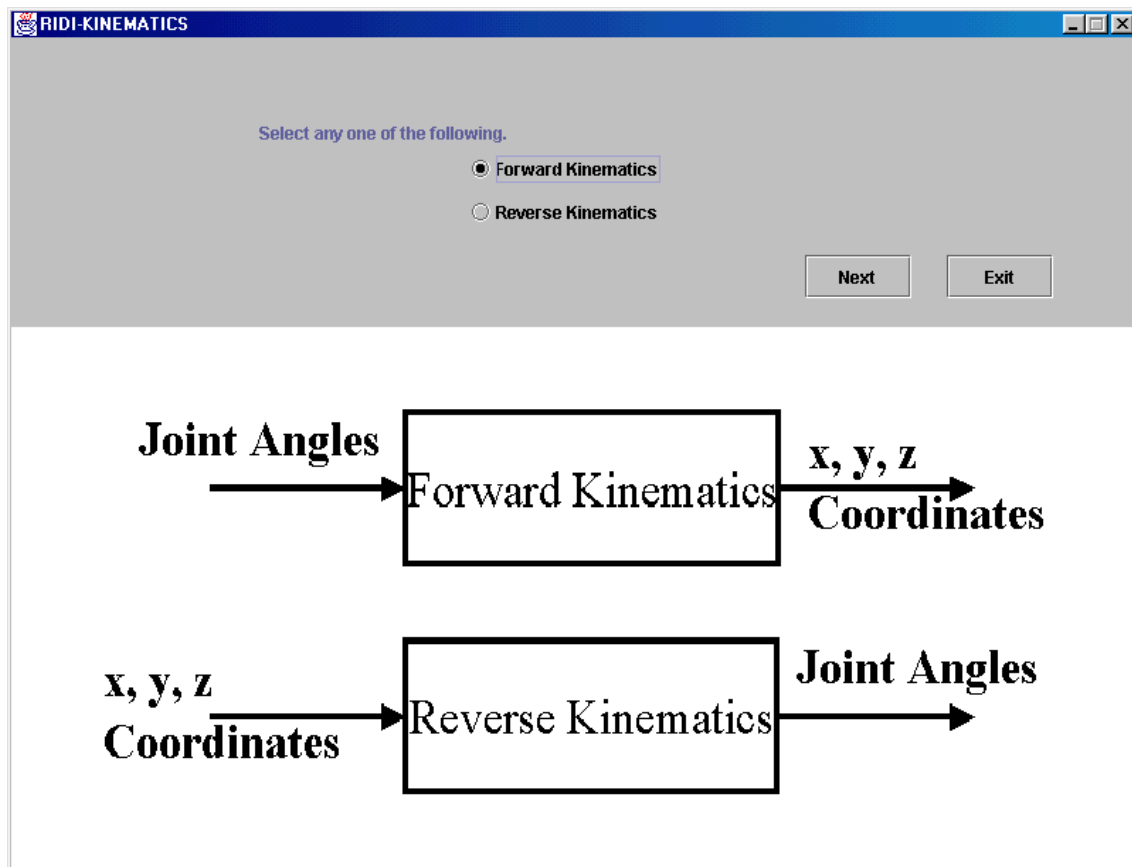


Figure 4. 4: RIDI Kinematics Screen

The bottom section has figures exhibiting the general idea of the two approaches. Figure 4. 5 shows the hierarchical architecture of the containers and components which make up the screen.

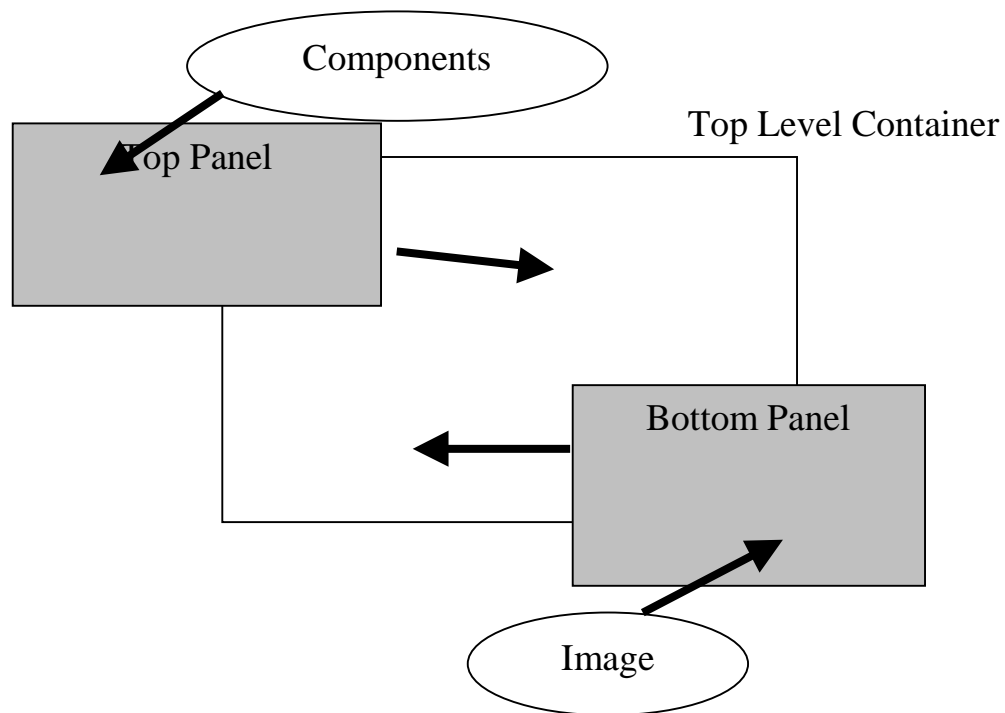


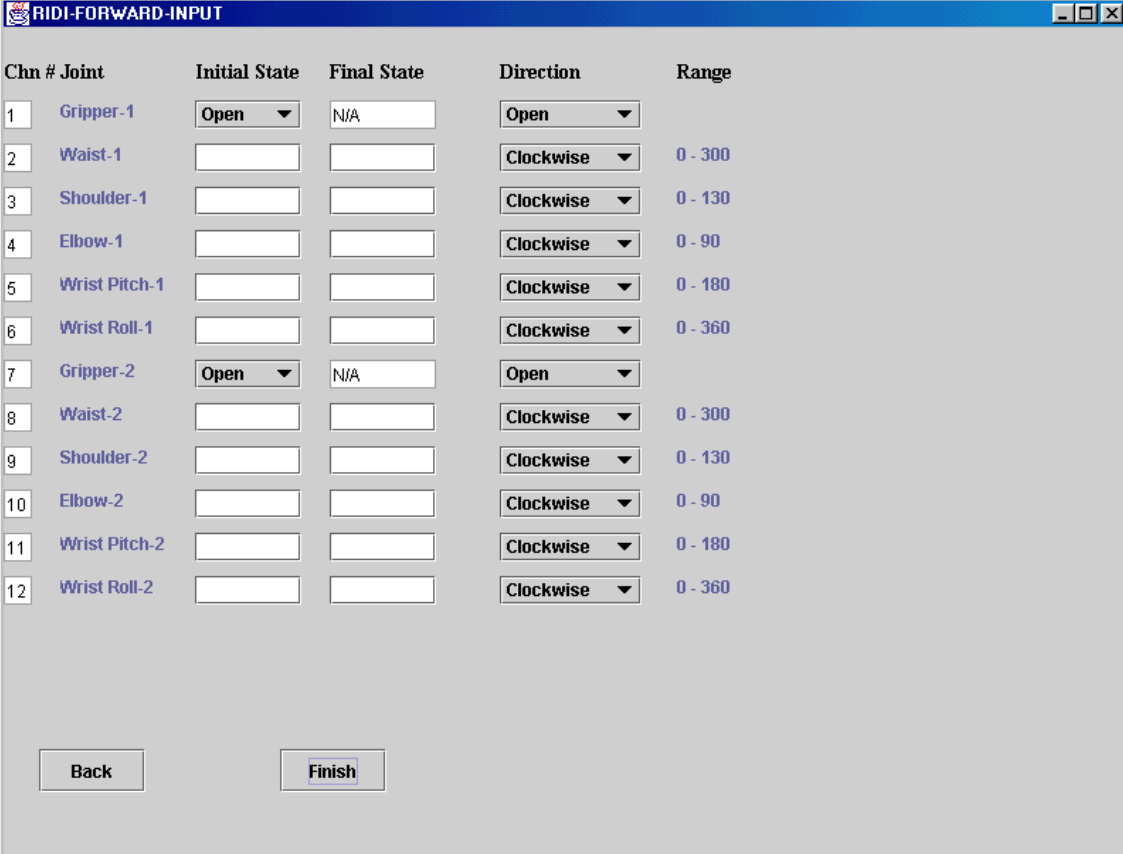
Figure 4. 5: Layered View of the RIDI Kinematics Screen

There is a top-level container at the root of the hierarchy. The top panel goes into the top portion of the swing frame where as the bottom area is occupied by the bottom panel. The images exhibiting the two approaches go into the bottom panel as shown in the figure. The top panel encapsulates the following components

1. Swing Label, asking the user to select any approach.
2. Swing Radio Button, for making the selection
3. Two Swing Buttons, for navigating to next screen or terminate the program

4. 2. 2 RIDI Forward Kinematics Input Screen

This screen is retrieve input from the user. The user has to enter initial and final values for individual theta angles along with the direction in which the user wants each of the joints to move. Figure 4. 6 shows the RIDI forward kinematics



The screenshot shows a software window titled "RID-FORWARD-INPUT". It contains a table with 5 columns: "Chn #", "Joint", "Initial State", "Final State", "Direction", and "Range". There are 12 rows of data. The first and seventh rows have "Open" in the "Initial State" and "N/A" in the "Final State" columns. The "Direction" column contains "Open" for rows 1 and 7, and "Clockwise" for all other rows. The "Range" column shows values like "0 - 300", "0 - 130", "0 - 90", "0 - 180", and "0 - 360". At the bottom of the window are two buttons: "Back" and "Finish".

Chn #	Joint	Initial State	Final State	Direction	Range
1	Gripper-1	Open	N/A	Open	
2	Waist-1			Clockwise	0 - 300
3	Shoulder-1			Clockwise	0 - 130
4	Elbow-1			Clockwise	0 - 90
5	Wrist Pitch-1			Clockwise	0 - 180
6	Wrist Roll-1			Clockwise	0 - 360
7	Gripper-2	Open	N/A	Open	
8	Waist-2			Clockwise	0 - 300
9	Shoulder-2			Clockwise	0 - 130
10	Elbow-2			Clockwise	0 - 90
11	Wrist Pitch-2			Clockwise	0 - 180
12	Wrist Roll-2			Clockwise	0 - 360

input screen.

Figure 4. 6: RIDI Forward Kinematics Input Screen

The user is expected to enter values on the specified fields of channel numbers on which he wants the activity to be carried out. There are 12 channels, which are responsible for motions of 6 joints of two different robots. The first 6 channels are for robot 1 while the other 6 are for robot 2. The channel numbers for each robot are

specified in the leftmost column as shown in the figure above. The second field is for identification of the joint, which specifies the joint name followed by the corresponding robot number. For e. g. Waist-1 implies waist joint of robot 1. Both the channel number and the joint identification field are uneditable.

Initial and Final States

These editable fields are for the user to enter the joint specific values. One can see that there is a difference between the way these fields have been put up for prismatic joints and rotatory joints. This is because of the difference in the way the values of these two different joints is specified. For e. g. suppose he wants to rotate the shoulder of robot 2 from the initial state of 10° to the final state of 100° . But one cannot specify such exact values for the gripper. It can either be a close command or open command. There is no intermediate stage. To incorporate this behavior of the prismatic joint a swing combo box has been added to the initial state field. So the user will select whether he wants the initial state of the gripper to be closed or open. Figure 4. 7 shows how a combo box works in this screen.

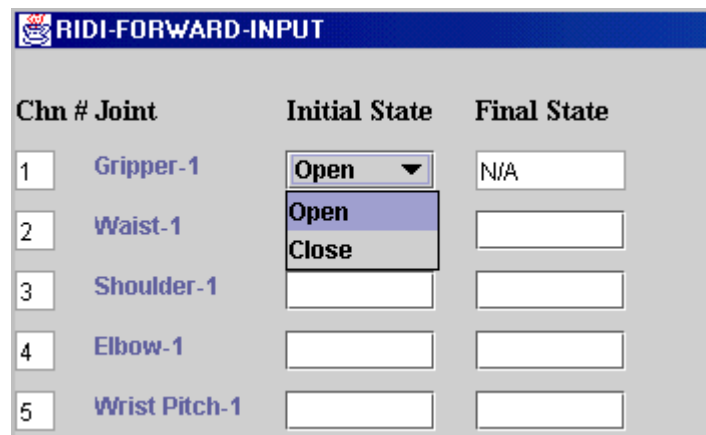


Figure 4. 7: Functionality of ComboBox

The combo box in Figure 4. 7 has a scroll down button, which gives the options available for the user to choose from. The options are "Open" and "Close". The selected option is highlighted. The default values for the combo box is "Open". The final state field is not applicable for the grippers because there is no specific value to be entered. This field is not applicable for the prismatic joints and is marked "N/A". This is field also uneditable.

Direction

This field is provided for the user to specify direction in which each individual joint should move. For instance, if the initial and final θ for a elbow joint of robot 1 are 20 and 65, then the user has to specify that the joint should trace that path in a clockwise manner. Similarly for the prismatic joint, if the initial position of the gripper is open and the user wants to close it, then he should specify that information in this field. This field also employs combo boxes for retrieving information from the user. Same options are used for the prismatic joint as shown in Figure 4. 7. For the rotatory joints, "Clockwise" and "Anti- Clockwise" options are used. Figure 4. 8 shows the combo boxes for the roatatory joints. The default value for these combo boxes is "Clockwise". Similar to Figure 4. 7 here also the selected option is highlighted. In the figure below, "Clockwise" option is highlighted.

-INPUT			
Initial State	Final State	Direction	Range
Open ▼	N/A	Open ▼	
<input type="text"/>	<input type="text"/>	Clockwise ▼	0 - 300
<input type="text"/>	<input type="text"/>	Clockwise ▼	0 - 130
<input type="text"/>	<input type="text"/>	Clockwise	0 - 90
<input type="text"/>	<input type="text"/>	Anti-Clockwise	
<input type="text"/>	<input type="text"/>	Clockwise ▼	0 - 180
<input type="text"/>	<input type="text"/>	Clockwise ▼	0 - 360

Figure 4. 8: Functionality of Combo Boxes for Rotatory Joints

Range

This is an uneditable field. This field helps the user to enter values within the specific range for each particular joint. The primary purpose of this field is to avoid any malfunctioning of the system. Figure 4. 8 shows the 5 ranges for the 5 joints of robot 1. The user should adhere to these ranges while entering values. For example for the elbow joint the user cannot expect the angle of this joint to be 100°, since its range is from 0 to 90 degrees. Entering such out of range value will give the user wrong results.

Numeric Check

It is imperative for the user to enter only numbers in the editable fields, such as the initial and final values for rotatory joint. This system provides a check on this condition and will not allow the user to enter any value into the field except numbers. Figure 4. 9 demonstrates this check. On an invalid entry into any numeric field this message appears on the screen.

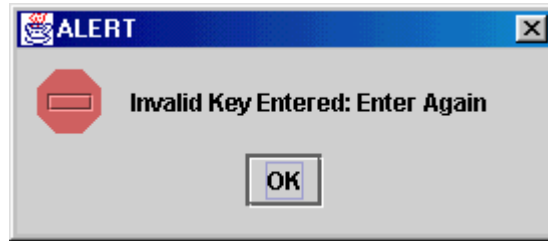


Figure 4. 9: Checking on Non Numeric Entries

The user cannot have control over the GUI screen unless he clicks the OK button. On clicking the OK button, the field where this error occurs becomes empty and it is required by the user to input values again in that particular field. The layered structure of this screen has much more components and containers than the kinematics screen. Figure 4. 10 shows the layered structure of the RIDI forward kinematics input screen.

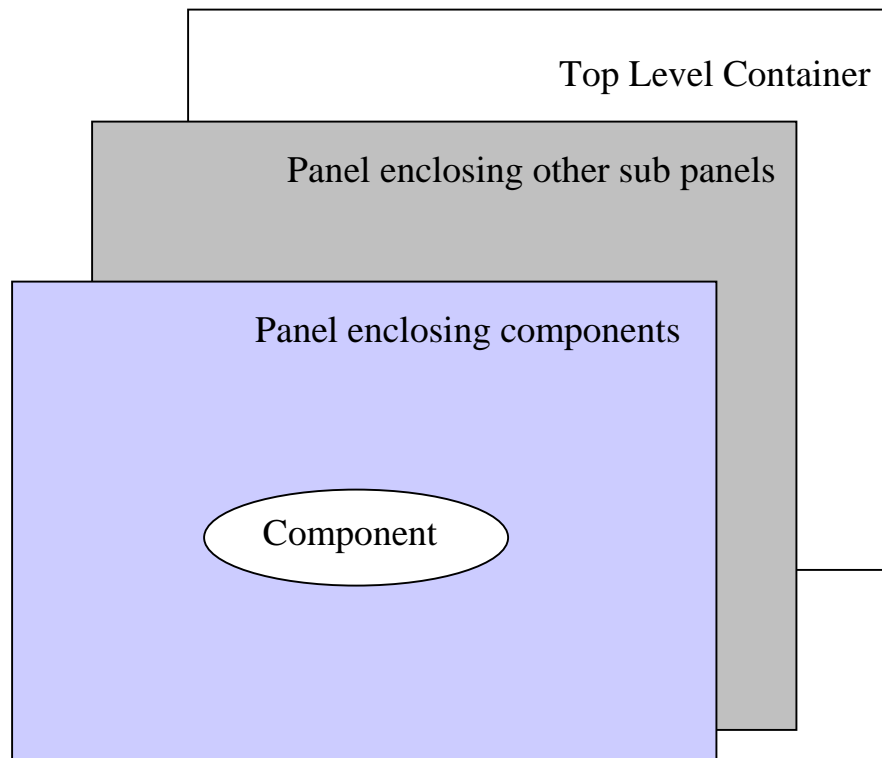


Figure 4. 10: Layered Structure for Forward Kinematics Screen

The figure above shows how the components are layered down in the outer panel, which in turn is encapsulated by an outer swing panel. The swing panel itself is laid out in the top-level container, the swing frame. The components laid down in the outer most panel are

1. Combo Boxes, for making choices between available options.
2. Text Fields, for entering numeric values.
3. Labels, for joint identification and range specifications.

4. 2. 3 RIDI Reverse Kinematics Input Screen

This comes up on the monitor if the user follows the reverse kinematics approach. The user is expected to enter the initial and final x , y , z coordinates of the end effector. The user is also required to enter individual theta values for wrist roll and wrist pitch for managing the orientation. Figure 4. 11 shows the RIDI reverse kinematics input screen. The input to the text fields is converted into individual theta angles using a reverse kinematics program. The values are then passed to the activity screen for getting the desired results. This screen has a simple hierarchical structure as shown in Figure 4. 12. Only one container is included in the top-level container. All other components are placed into this container. The components laid in the outer most panel are

1. Text Fields, for the user to enter values.
2. Labels, for identification of each text field.
3. Buttons, for termination and navigation.

All the text fields are editable and similar numeric check is applied on each text field.

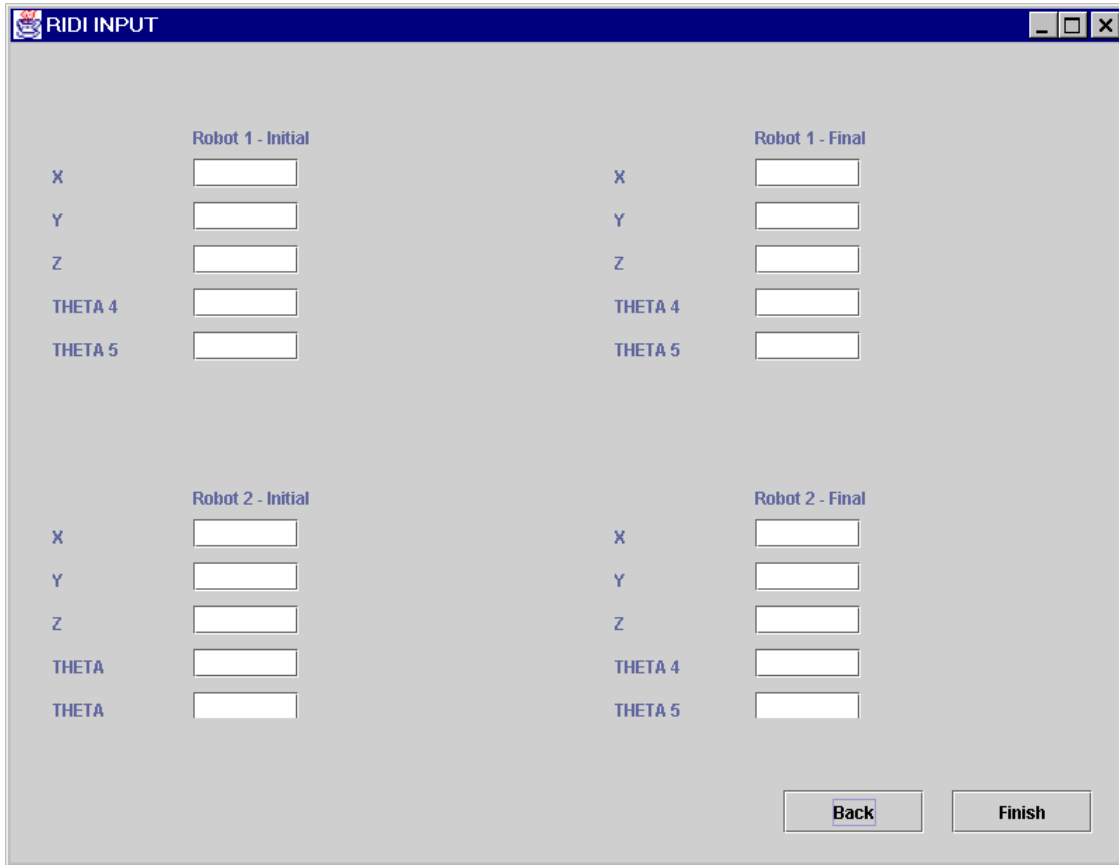


Figure 4. 11: RIDI Reverse Kinematics Input Screen

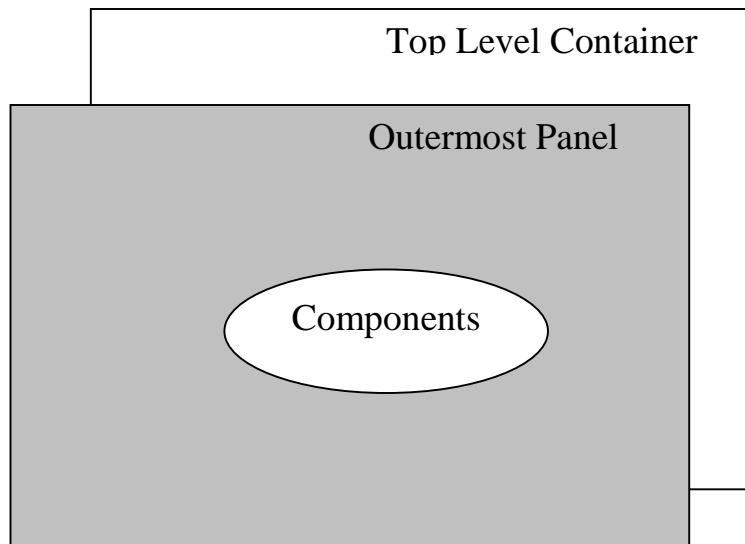
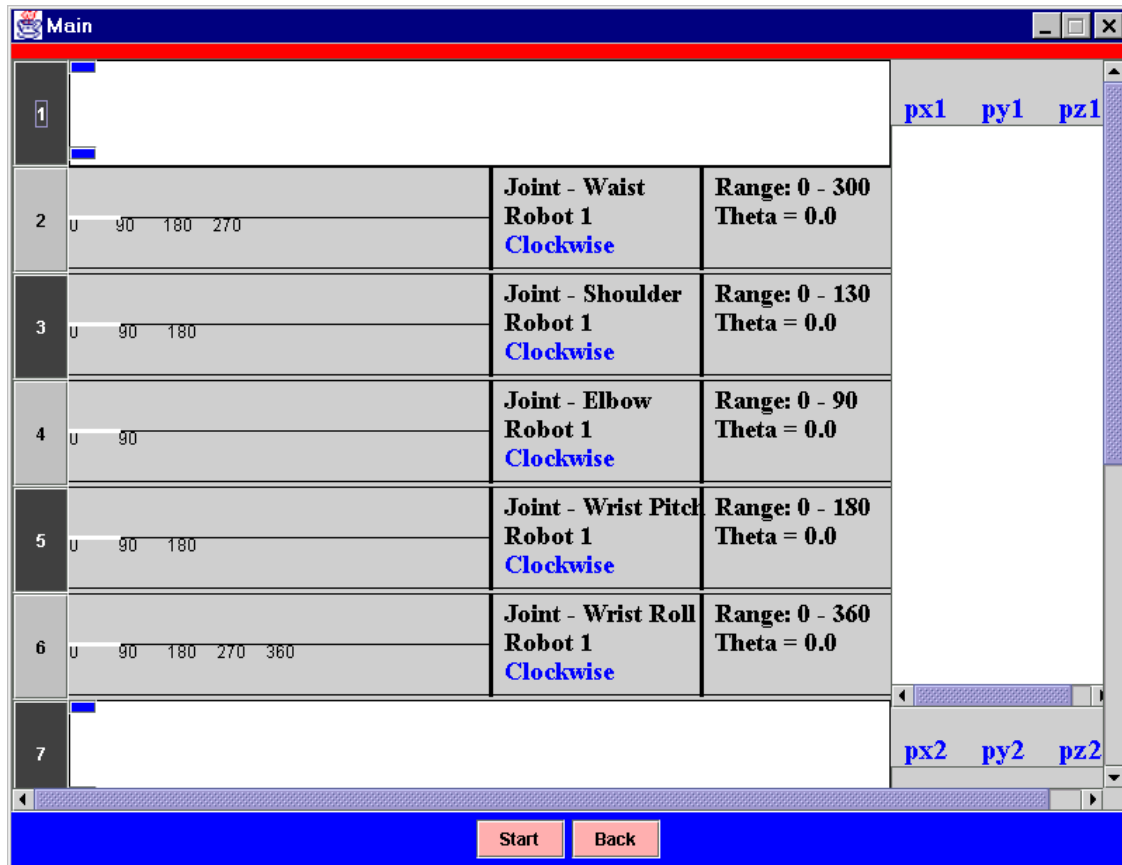


Figure 4. 12: Layered Structure of Reverse Kinematics Input Screen

4. 2. 4 GUI Activity Screen

The GUI activity screen is responsible for the output of the software. The user sees the specified tasks, which he wants to carry out on each channel corresponding to each individual joint. This screen incorporates variable features to accomplish the real time motion of different joint activities. Figure 4. 13



demonstrates this screen.

Figure 4. 13: RIDI GUI Activity Screen

Following are the areas of operation in this screen.

1. Control Panel
2. Rotational Joint Activity
3. Prismatic Joint Activity

4. Coordinate Points Activity Area

Control Panel

This is the area at the bottom of the screen in blue color. This area is not dynamic during the execution of the joint activities. The role of this area is to lay out the buttons for accessing control over the screen and for navigation. There is no exit screen in this area. The user can exit the system only by using the window options. There is a start button, which simultaneously starts activities on each screen. The back button is used for navigating to the RIDI Kinematics Screen. Figure 4. 14 shows this area separately and Figure 4. 15 explains its layered structure.

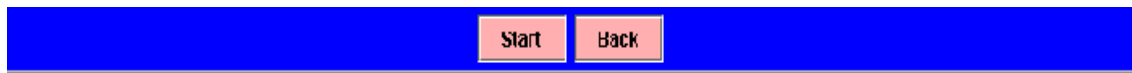


Figure 4. 14: Control Panel of RIDI Activity Screen

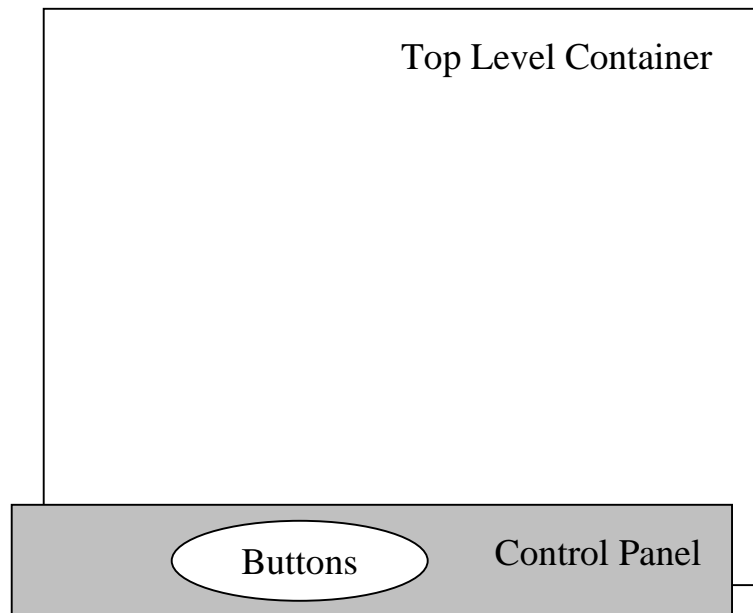


Figure 4. 15: Layered Structure of the Control Panel

Rotational Joint Activity

This area refers to the different channels corresponding to each rotatory joint. There are ten different panels, five for each robot. This area is responsible for showing the graphical representation of the changing individual θ values. It also provides information to the user regarding the specifications about each channel. This area is divided into sections. One area shows the actual motion of the links while the other shows the numerical representation of the changing θ values along with the joint specifications. Figure 4. 16 shows the actual activity section and Figure 4. 17 shows the specification section of this area. Though these two sections are laid out on one single panel, their way of representing information is different.

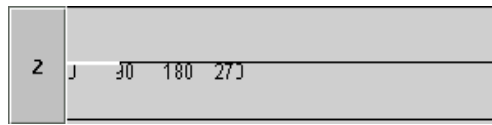


Figure 4. 16: Graphical Representation of Each Joint

Joint - Waist Robot 1 Clockwise	Range: 0 - 300 Theta = 0.0
--	---

Figure 4. 17: Specifications of the Joint Motion

Figure 4. 16 shows the motion of the link depending upon the initial and final θ values corresponding to that joint. If both the initial and final values are same, then there will be no motion on that particular panel. Also the range of degrees shown on the motion panel depend on the range of the corresponding joint. For e. g. Figure 4. 17 shows the range of the waist joint between 0 to 300 degrees. Accordingly the values in Figure 4. 16 are shown as 0, 90, 180& 270. In this case 360 is excluded

because, the range is till 300 only. The specification panel shows the direction of each joint. In Figure 4. 17 the direction is shown “Clockwise” in blue color. Also this section shows the value of theta on a real time basis. As the link starts moving, the theta values start incrementing or decrementing accordingly. Other information provided by this section is the joint name and the corresponding robot number. This area is a swing panel added to the scrollbar panel. No components are added to the panel. All the information is painted on the panel using 2D Graphics features in Java.

Prismatic Joint Activity

The function of this area is to show the graphical representation of the closing and opening operations of the grippers of two robots. Figure 4. 18 shows the panel for prismatic joint. The two blue buttons represent the gripper plates.



Figure 4. 18 Prismatic Joint Activity Panel

Coordinate Points Activity Area The coordinate points activity area is shown by a white scroll bar panel at the right of the GUI activity screen. It gives the coordinates of the end effector of each robot. There are two scrollable areas, which are updated as the activities on each robot start. Figure 4. 19 shows this area separately. The values shown in this area are in millimeters.



Figure 4. 19: Scrollable Coordinates Activity Area

The panel showing the indication of px , py and pz doesn't scroll down. Only the white area is scrollable. Similar area is added exactly below this area for retrieving coordinates of robot 2 end effector.

4.3 Layered Structure of the GUI Activity Area

Figure 4. 20 shows the layered structure of the GUI activity screen. At the root of the hierarchy is a swing frame, which is the top-level container. The top level container comprises of two containers, the Scroll Pane and the Panel. The panel goes to the bottom of the frame and comprises of components. The scroll pane occupies the remaining area of the pane comprises of two panels. One panel comprises of panels to show activity on each screen, where as the other panel takes in further containers to show activity of the space coordinate points. This is the general structure of the

activity screen. Though specific to the robot it can be extended to other robots without much difficulty.

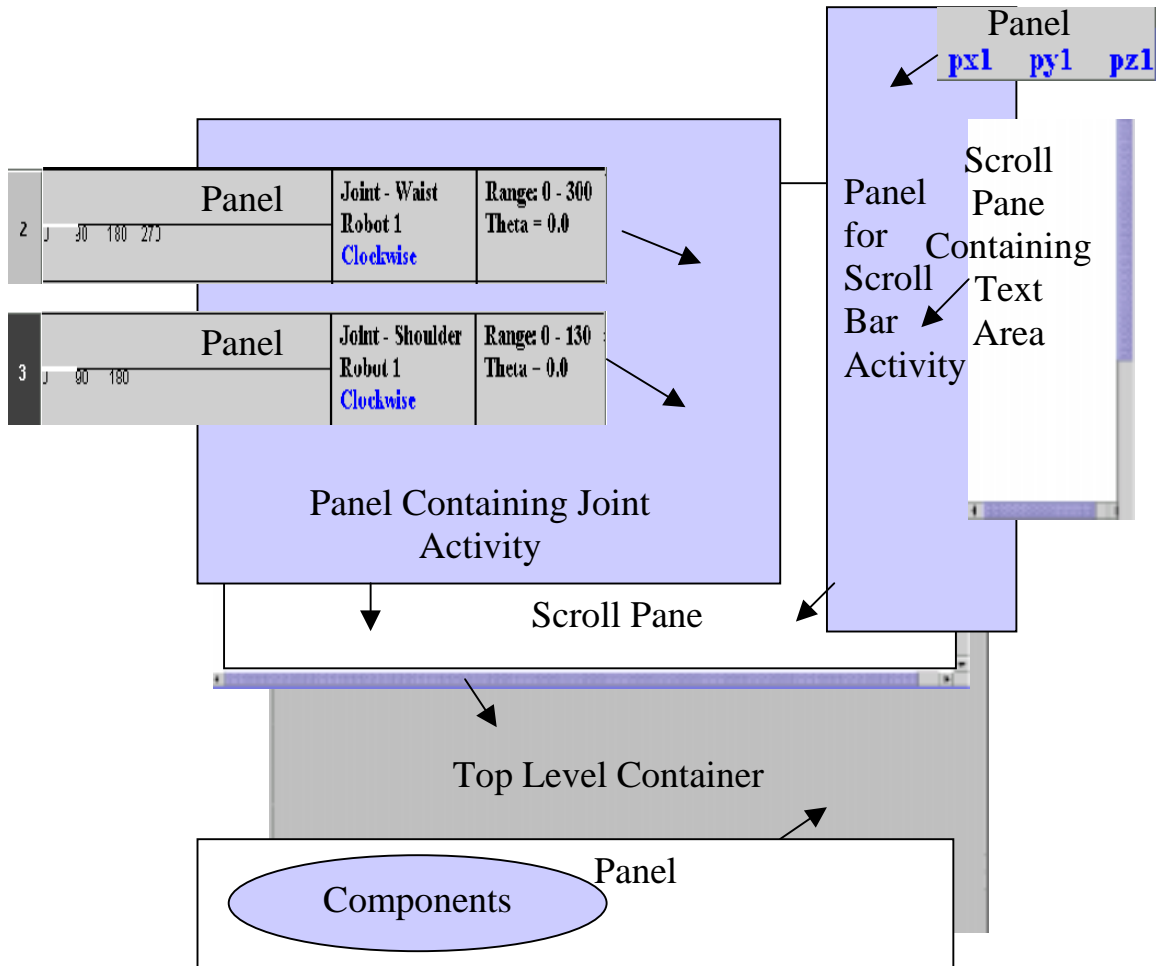


Figure 4. 20: Layered Structure of Main Activity Screen

4.4 Thread Synchronization to achieve Real Time Control

The most crucial phase of the program is its run time behavior. When the θ values are provided as input to the system, the user should update each of the individual joint angle in a synchronized manner. The elbow angle of robot 1 once incremented to its

higher value should wait for all other joints to carry out their respective updating. Till then the elbow joint should not update its value. Also the calculation of space coordinate system should be synchronized with the running threads. Once theta value of each individual angle is passed for computation, no further incrementation should be carried out on any joint until the coordinates are updated. This feature of synchronization is achieved by using Java's synchronized method. A sample code executing this feature is shown below.

Sample Code:

```
public class get_theta
{
int j = 4;
txt_rl tr1 = new txt_rl();
public double theta[] = new double[6];
public int i=0;
public synchronized void setValue(double value, int index, double f)
{
theta[index] = value;
if(value == f)
j = j-1;
i = i + 1;
while(i<=j)
{
try
{
wait();
return;
}
catch(InterruptedException e)
{
}
}
tr1.draw_theta(theta);
i = 0;
notifyAll();
}
}
```


The `get_theta` class is responsible for synchronization between the threads running on each joint and to update the scroll area for coordinate points. The `wait()` method makes all threads running on this class to wait until the value of variable `i` is not greater than `j`. When this condition is met, the `notifyAll()` method, releases all the threads running on this method. The `draw_theta` method is used to pass values of `theta` to the class updating the point coordinate area.

Chapter 5

MODEL EXAMPLES

5.1 Real World Application

There are several opportunities for the RIDI system to be utilized for real world application. This chapter will discuss the use of RIDI interface to conduct robot assisted assembly. The assembly task can be conducted by guiding the robot via a PC. Let the assembly task be that of picking up two different parts from two known locations and aligning them for an assembly. To accomplish this two robotic arms are used in a 3-D space. The coordinate points in 3D space for the present position of the two parts and the final position of the parts to be aligned is provided as input. This set of task is analogous to the set of musical notes to be played by each instrument in a musical rendition. The operator can monitor the robot motions on the GUI. He will have the ability to control the motion of the robots and can change the course of their motions if need arises. This system can be extended to handle more complex robotic tasks.

5.2 GUI Example

Figure 5.1 shows a prototype 3D space coordinate system where in two robots are assigned the task of picking up two different parts and align them for a manufacturing assembly. The initial position of both the robots and the final position required for the alignment of the parts is input to the system.

Example Figure

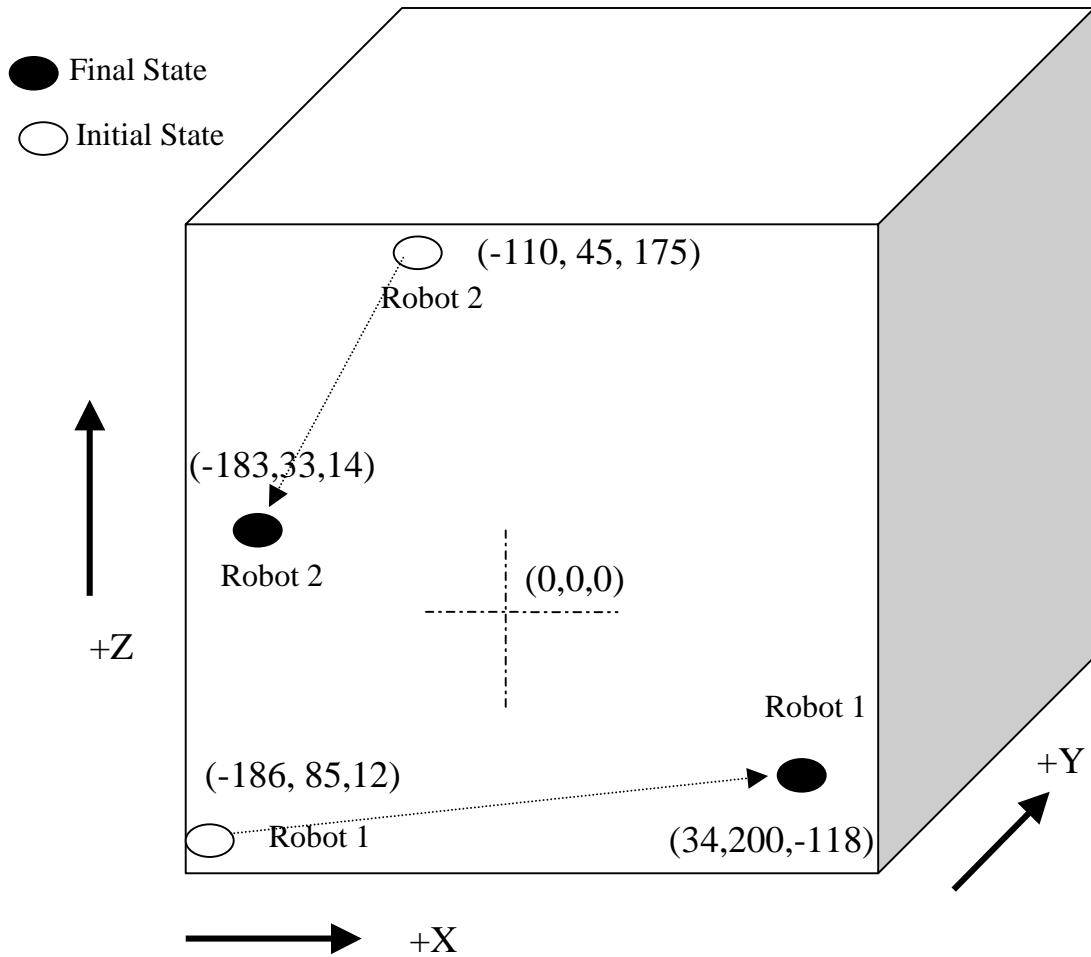


Figure 5.1: Assembly Task Layout Example

The current position of Robot1 is at $X = -186$, $Y = 85$ and $Z = 12$. Its orientation is given by the angles $\theta_4 = 27$ and $\theta_5 = 59$. The user wants the robot to move to the final position of $X = 34$, $Y = 200$ and $Z = -118$ with the theta values as $\theta_4 = 48$ and $\theta_5 = 22$.

Robot 2 has its present position as $X = -110$, $Y = 45$ and $Z = 175$ with $\theta_4 = 72$ and $\theta_5 = 39$. The user wants to move the robot 2 arm to the position $X = -183$, $Y = 33$ and $Z = 14$ and with the orientation as $\theta_4 = 77$ and $\theta_5 = 79$. All these values are in

mm. The dotted lines represent the path traced by both robot end effectors to accomplish the stated task.

The screenshot shows a window titled "RID I INPUT" with a dark blue header. The main area is light gray and contains two columns of input fields. The first column is for "Robot 1 - Initial" and the second is for "Robot 1 - Final". Below these are two columns for "Robot 2 - Initial" and "Robot 2 - Final". Each field contains a numerical value. At the bottom right, there are two buttons: "Back" and "Finish".

Robot	Parameter	Initial Value	Final Value
Robot 1	X	-190	35
	Y	90	200
	Z	10	-120
	THETA 4	27	48
	THETA 5	59	22
Robot 2	X	-110	-185
	Y	45	30
	Z	175	15
	THETA 4	72	77
	THETA 5	39	79

Figure 5.2: RIDI Input Screen for Assembly Example

GUI Input Screen

From the given data, it is concluded that one needs to conduct reverse kinematics approach. Accordingly, the user will select the reverse kinematics option and the RIDI Reverse Kinematics Input Sheet will come up on the monitor. Figure 5.2 shows the input screen for the stated problem. After the user has entered the values he will click on the

finish button to navigate to the GUI activity screen. The GUI activity screens contain 12 channels and hence cannot be viewed in entirety. So each screen is explained with the help of two figures.

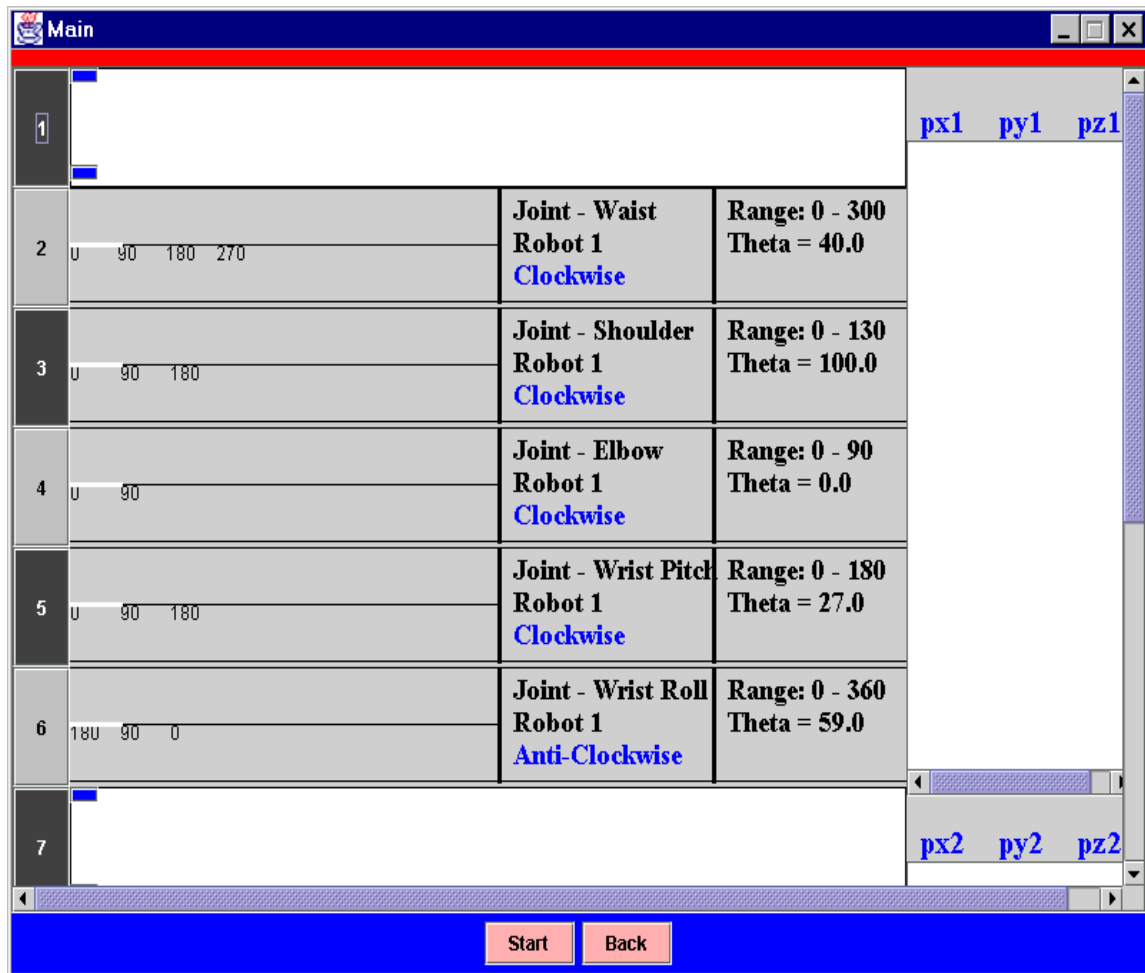


Figure 5.3: Initial Activity Screen for Robot 1

GUI Activity Screen – Initial State

Figure 5.2 shows the initial state of the activity screen for robot 1. For the initial state coordinates entered by the user as input, we get the initial values of individual θ angles. For example we can deduce that the initial state of robot 1 is θ_1

= 40.0; $\theta_2 = 100.0$; $\theta_3 = 0$; $\theta_4 = 27$; $\theta_5 = 59$. Similarly Figure 5.3 provides information about robot 2.

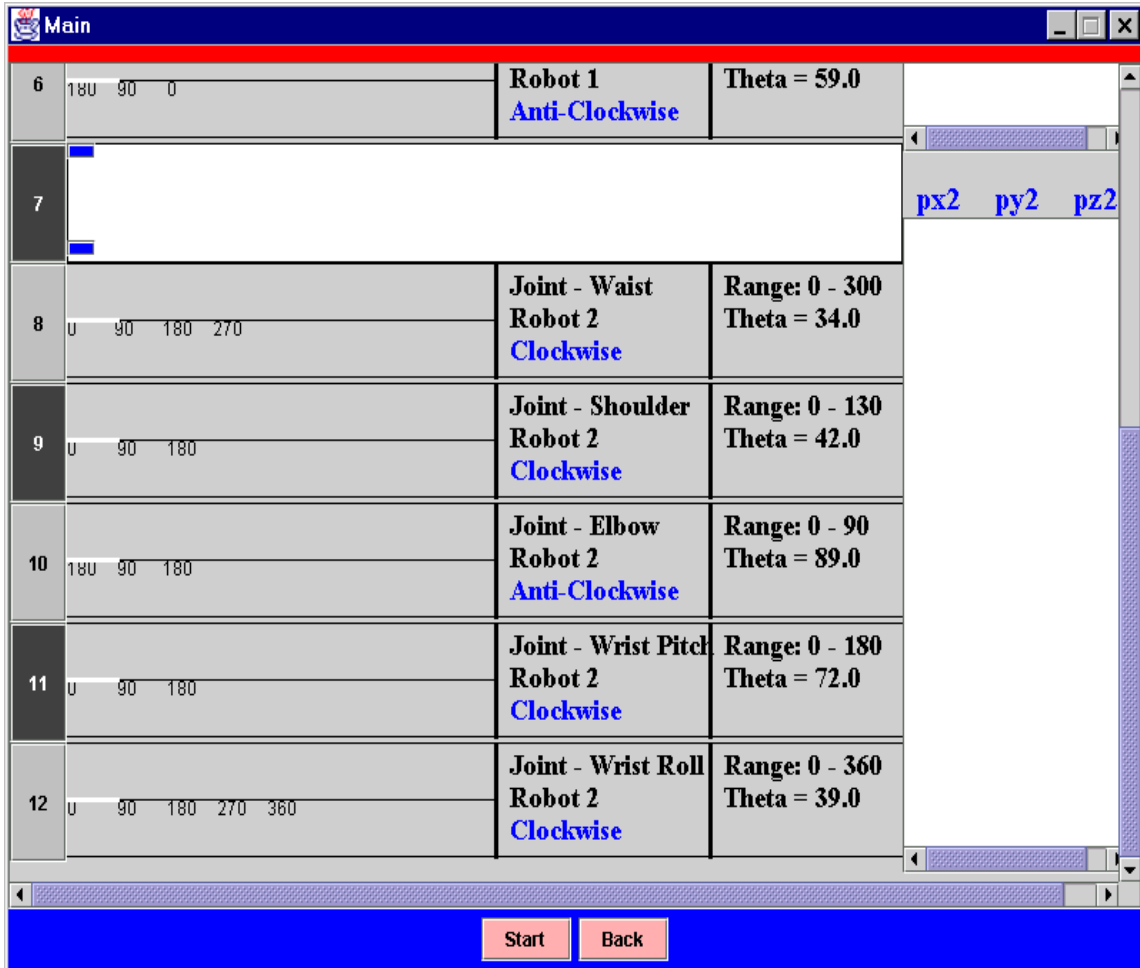


Figure 5.4: Initial Activity Screen for Robot 2

The initial state θ values for robot 2 are $\theta_1 = 34.0$; $\theta_2 = 42.0$; $\theta_3 = 89$; $\theta_4 = 72$; $\theta_5 = 39$. The user after getting this screen is now required to click on the start button to see the real time motion of each joint.

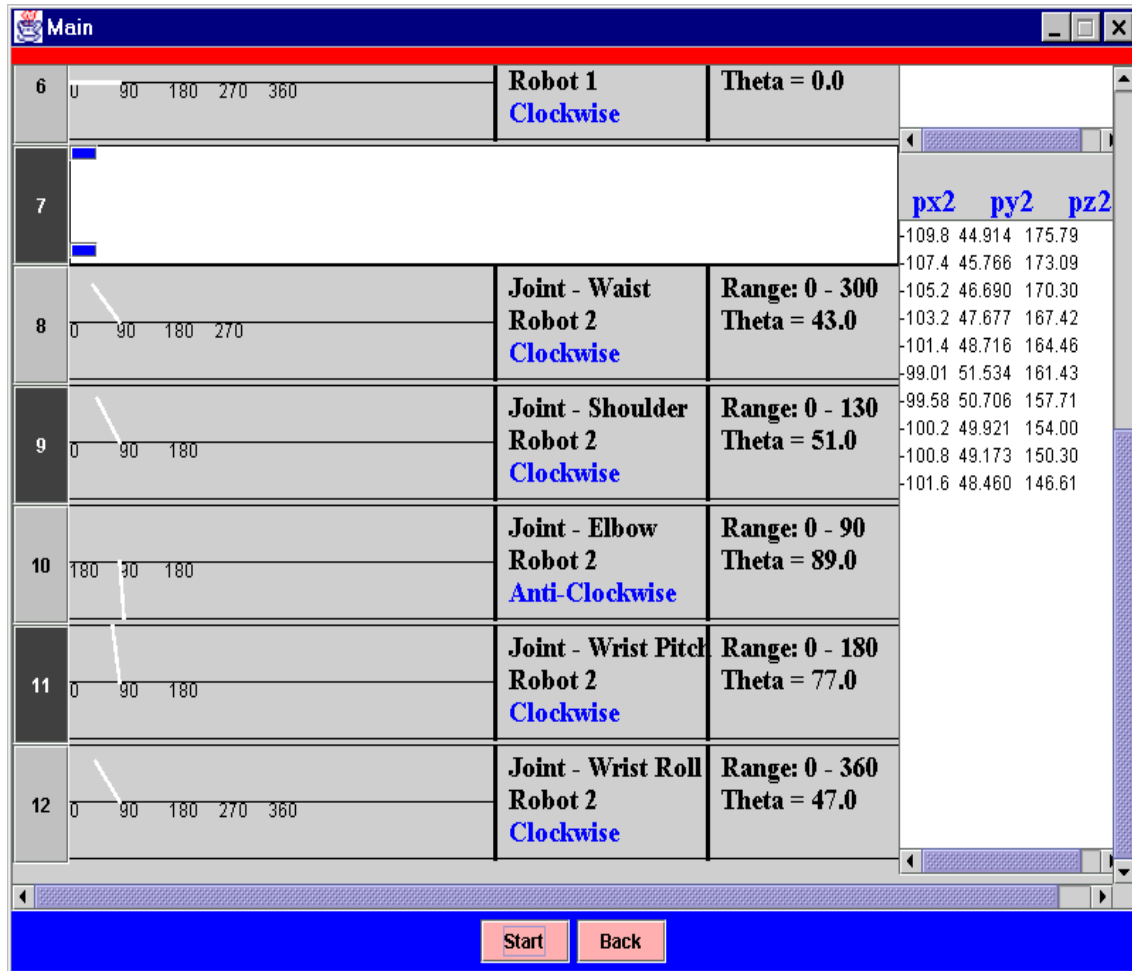


Figure 5.6: Intermediate Screen for Robot 2

The intermediate state in the above figure refers to the position. $px = -101.6$;
 $py = 48.460$; $pz = 146.61$.

The corresponding θ values are

$$\theta_1 = 43.0; \theta_2 = 51.0; \theta_3 = 89.0; \theta_4 = 77.0; \theta_5 = 47.0$$

For both robots 1 and 2 the links as well as the θ values along with the space coordinate points can be observed in an intermediate state screen. This gives the user a fair idea about the execution of the task.

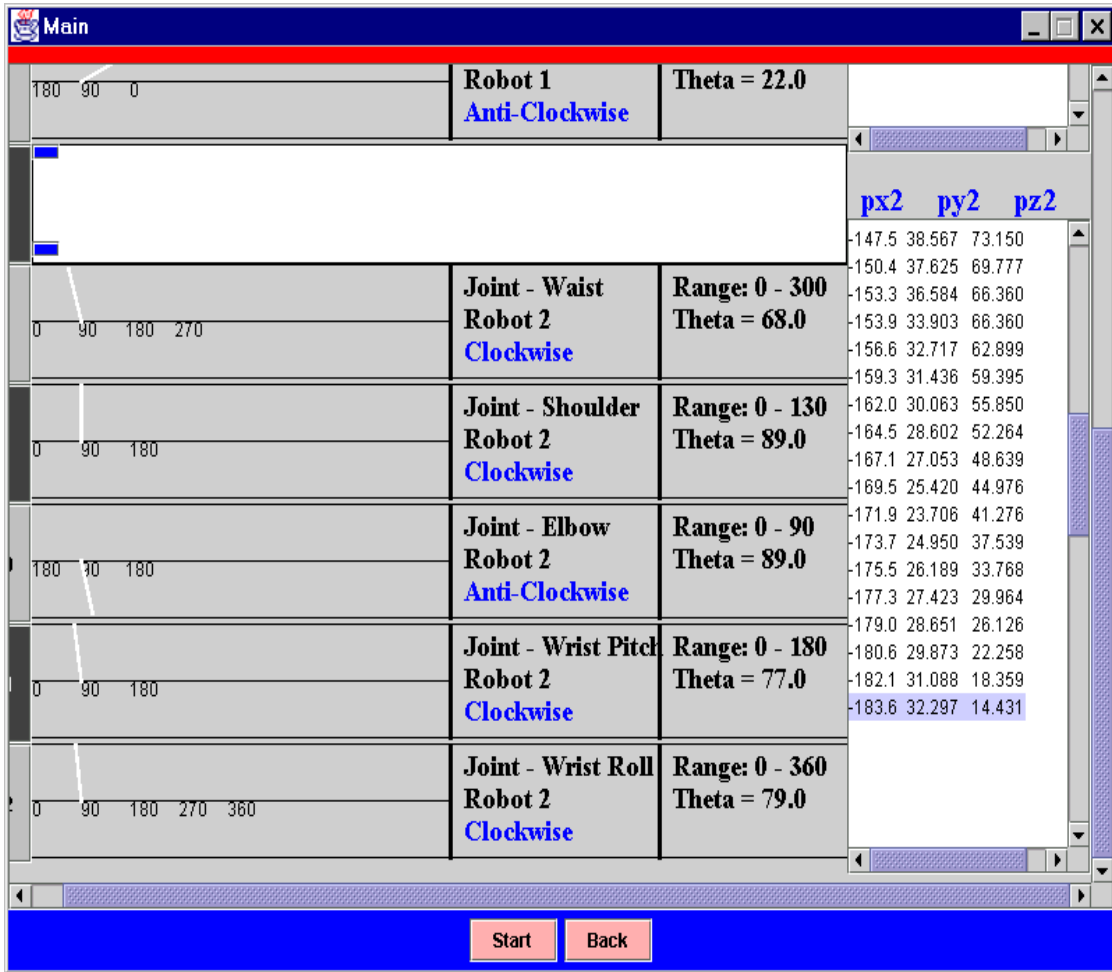


Figure 5.8: Final Activity Screen for Robot 2

The point coordinates of the end effector do not match exactly with the specified one because of the round off and truncation.

Chapter 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The only medium for the user to interact with RIDI system is through the computer monitor and keyboard. This makes the development of Graphical User Interfaces a critical issue for the success of the system. Though in its early stages, RIDI GUI offers a basis for the functionality of a complex RIDI system. Developing the GUI of the RIDI system was the first step towards the RIDI project and an important one. Details of the tasks accomplished are as follows.

- The MIDI User Interfaces available in the market were studied and some of the features of these user interfaces were incorporated into the RIDI GUI. For example, the feature of using channels for showing data on different channels was incorporated from the MIDI Cool Edit Pro Software. Also, the scrolling nature of the coordinates of the end effector was on the lines similar to the MIDI Event List.
- Key feasibility issues were studied and evaluated for developing the GUI. Many features that go into MIDI User Interfaces were not included into the RIDI GUI because of the difference in the nature of environment, in which the two systems operate.
- The GUI was successfully implemented in Java and demonstrated for a practical example for validating its functionality. Though the developed GUI works best when used in conjunction with the RM 501 MoveMaster, it can be extended to other similar robots with few modifications.

- The GUI was tested and demonstrated for two robots. Additional robots can be easily incorporated into the system. The logic behind each robot is the same, making it feasible for other similar automated system.

Though in its primitive stage this GUI provides in roads into the control and manipulation of robotic devices in a real time environment.

6.2 Future Work

The RIDI GUI offers great opportunities for further expansion. The GUI is developed to control simple robotic motions, leaving opportunities for many complex tasks that can be done through the RIDI system. The developed GUI is user friendly and compatible for conveying user the information of the robotic motions. The RIDI GUI looks forward to more practical scenario where in the input given by the user will be used to actually move parts from one place to another, rather than just represent joint links moving in specified direction and to a specified value. The GUI in its primitive stages provides a building block for more complex and intricate real time motions or real time systems. Some of the areas which have not been covered in this work, but provide a challenge are

- One of the major works to be done in this project is establishing the communication between computer input and the robot controller. This area is responsible for converting the data provided by the user in a robot understandable form. It should not only convert the data but also be responsible for serial transmission of that data through MIDI cables in a real time mode. The manner in which relevant data is handled by the individual robot controller and irrelevant

data is passed on to the next controller down the line is based on how well the data is serially transmitted through the cables and how well it is controlled

- The other important feature this project offers is the feedback of data from the robot to the user in a timely fashion. Considering the applications, this work can be implemented for, it is inevitable to overlook the importance of feedback in RIDI. Currently, this is accomplished in the GUI with not many intricacies because only one computer is used. In cases where more than one computer is used, retrieving feedback can be an area of concern. Hence to make this system flawless as far as feedback in multiple PC environment is concerned, future work needs to be done on this area.
- More flexibility can be added to the RIDI system by incorporating some of the features, which have not been considered in the GUI. The GUI can very well support the change in number of robots and joints with no major changes in the code. But not the entire range of robots is covered by the GUI. So there is a possibility to have a menu driven GUI which considers all the popular robots used in industry, and accordingly the requirements of the GUI can be changed.

REFERENCES

1. Luh, J. Y. S., " An Anatomy of Industrial Robots and Their Controls", IEEE Transactions on Automatic Control, Vol. AC-28, No. 2, pp. 5, February 1983.
2. Lee C. S. G., Gonzalez R. C., Fu K. S., "Tutorial On Robotics", IEEE Computer Society, pp. 1, 1986.
3. Rosen C. A., Nitzan D., "Use of Sensors in Programmable Automation," Computer, Vol. 10, Number 12, pp. 12, December 1977.
4. Dorf, R. C., "Robotics and Automated Manufacturing", pp. 119, 1983.
5. Nevins J. L., Whitney D. E., " Computer Controlled Assembly ", *Scientific American*, pp. 62-74, February 1982.
6. Thompson., T., " Robots for Assembly ", *Assembly Engineering* , pp. 32-36, July 1981.
7. Dodrill W. H., Ahluwalia R. S., "Project Summary-Robotic Instrument Digital Interfacing", NASF Proposal, pp. 1-5, 1998.
8. Tarvin., R. L., " An Off-Line Programming Approach ", *Robotics Today*, pp.30-35, Summer 1981.
9. Dorf, R. C., "Robotics and Automated Manufacturing", pp. 112, 1983.
10. Laplante., P. A., " Real Time Systems Design and Analysis: An Engineers Handbook ", pp. 1, 1997.
11. Koivo A. J., Bekey G. A., "Report of Workshop on Coordinated Multiple Robot Manipulators: Planning, Control, and Applications", IEEE Journal of Robotics and Automation, Vol. 4, No. 1., February 1988, pp. 91-93.
12. Kheradpir S., Thorp J. S., "Real-Time Control of Robot Manipulators in the Presence of Obstacles", IEEE Journal of Robotics and Automation, Vol. 4, No. 6, December 1988, pp. 687-692.
13. Valaer L., Babb G., "Choosing a User Interface Development Tool", IEEE Software, July/August 1997, pp. 29 – 39.
14. Cool Edit Pro, <http://www.syntrillium.com/cep/prodemo.htm>.

15. Cakewalk, <http://www.cakewalk.com/products/index.html>, Twelve Tone Systems Inc.
16. McKerrow, P. J., "Introduction to Robotics", pp. 640, 1991.
17. Gourdeau, R., " Object Oriented Programming for Robotic Manipulator Simulation ", IEEE Robotics & Automation Magazine, pp. 21-28, September 1997.
18. An Experimental Dual Arm Space Teleoperation System: P - ARM, <http://www.space.mech.tohoku.ac.jp/research/parm/p-arm.html>
19. Heckroth, J., <http://www.harmony-central.com/midi/doc/tutorial.html>, "Tutorial on MIDI and Music Synthesis", The MIDI Manufacturers Association, pp. 1, 1995.
20. Midifarm, <http://www.midifarm.com/midifarm/hardware.asp>.
21. Midiman Products, <http://www.midiman.net/product.HTM>.
22. Octetdesign, <http://www.octetdesign.com/products/pianomidi.shtml>.
23. CakewalkPro, <http://www.cakewalk.com/products/PA/PA8.html>.
24. Lipscomb, E., <http://www.eeb.ele.tue.nl/midi/intro.html>, "Introduction Into MIDI", North Texas Computing Center Newsletter, "Benchmark", October 1989.
25. Borg International Services, <http://www.borg.com/~jglatt/tutr/whatmidi.html>, Suite 403, 4th Floor, 1001 Broad Street, Utica, New York, 13501.
26. McKerrow, P. J., "Introduction to Robotics", pp. 486, 1991.
27. Dodrill W. H., Ahluwalia R. S., Famouri P., "Robotic Instrumental Digital Interfacing", DARPA Information Technology Office, pp. 1-11, 1998.
28. Mitsubishi RM501 Movemaster II Instruction Manual, Mitsubishi Electric Corp., Nagoya, Japan, 1987.
29. McKerrow, P. J., "Introduction to Robotics", pp. 23, 1991.
30. Advanced Robot Telemanipulator System for Minimal Invasive Surgery, <http://iregt1.iai.fzk.de>, Forshugzentrum Karlsruhe GmbH, Institut fuer Agewandte Informatik, Postfach 3640, Karlsruhe, Germany.

- 31.** Center for MRCAS, <http://www.mrcas.ri.cmu.edu>, Carnegie Mellon University Robotic Institute, 130 Smith Hall, 5000 Forbes Avenue, Pittsburgh, PA 15213.
- 32.** Project on Image Guided Surgery, http://www.ai.mit.edu/projects/vision-surgery/surgery_home_page.html, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139-4307 USA.
- 33.** Medical Robotics Project, <http://www.eecs.berkeley.edu/~mcenk/medical>, Robotics Lab, The University of California, Berkeley, Berkeley, CA 94720.
- 34.** McKerrow, P. J., "Introduction to Robotics", pp. 43, 1991.

APPENDIX

```

//ridiKinematics.java

//Program for the RIDI Kinematics Screen ... It is the class which contains the main method.

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.Graphics;

public class ridiKinematics extends JFrame
{
    public static int rb_f = 1, rb_r = 0;

    //Defining Panel which goes into the upper part of the screen
    JPanel panel_choice = new JPanel();

    //Image definition which goes into the label on the bottom of the frame
    Icon img_kinematics = new ImageIcon("kinematics.gif");
    JLabel label_kinematics = new JLabel(img_kinematics);

    //Defining Radio Button and label for making choice between the two approaches
    JRadioButton rb_forward = new JRadioButton("Forward Kinematics",true);
    JRadioButton rb_reverse = new JRadioButton("Reverse Kinematics");
    ButtonGroup group_kinematics = new ButtonGroup();
    JLabel label_choice = new JLabel("Select any one of the following.");

    //Defining buttons for navigation and termination of the program
    JButton next_choice = new JButton("Next");
    JButton exit_choice = new JButton("Exit");
    Insets ins = new Insets(2,2,2,2);

    //Constructor for the class
    public ridiKinematics()
    {
        //Setting properties of the Frame
        super("RIDI-KINEMATICS");
        setSize(800,600);
        setResizable(false);
        getContentPane().setBackground(Color.white);
        getContentPane().setLayout(new BorderLayout());

        //Setting properties of the upper panel
        panel_choice.setPreferredSize(new Dimension(800,200));
        panel_choice.setBackground(Color.lightGray);
        panel_choice.setLayout(null);

        //Setting bounds for the label, buttons and radio button
        rb_forward.setBounds(325,75,150,30);
        rb_reverse.setBounds(325,105,150,30);
        label_choice.setBounds(175,50,250,30);
        exit_choice.setBounds(660,150,75,30);
        next_choice.setBounds(560,150,75,30);

        next_choice.setMargin(ins);
        exit_choice.setMargin(ins);
    }
}

```

```

//Setting background colors for radio buttons, label and buttons
rb_forward.setBackground(Color.lightGray);
rb_reverse.setBackground(Color.lightGray);
label_choice.setBackground(Color.lightGray);
next_choice.setBackground(Color.lightGray);
exit_choice.setBackground(Color.lightGray);

//adding the radio buttons to the radio group
group_kinematics.add(rb_forward);
group_kinematics.add(rb_reverse);

//adding listeners to the radio buttons and buttons
rb_forward.addActionListener(new RadioListener());
rb_reverse.addActionListener(new RadioListener());
next_choice.addActionListener(new ButtonHandler());
exit_choice.addActionListener(new ButtonHandler());

//adding the button , radio button and the label to the upper panel
panel_choice.add(rb_forward);
panel_choice.add(rb_reverse);
panel_choice.add(label_choice);
panel_choice.add(next_choice);
panel_choice.add(exit_choice);

//adding the panel and the label to the upper and lower portion respectively
getContentPane().add(panel_choice, BorderLayout.NORTH);
getContentPane().add(label_kinematics, BorderLayout.CENTER);

setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        int n = JOptionPane.showConfirmDialog(null,"Are you sure you want to
exit?", "RIDI_GUI",JOptionPane.YES_NO_OPTION);
        if(n == JOptionPane.YES_OPTION)
        {
            System.exit(0);
        }
    }
});

setVisible(true);
}
public static void main(String args[])
{
    ridiKinematics rki = new ridiKinematics();
}

//class to handle radio button actions
class RadioListener implements ActionListener
{
    public void actionPerformed(ActionEvent er)
    {
        String rad = er.getActionCommand();
    }
}

```

```

if(rad == "Forward Kinematics")
{
ridiKinematics.rb_f = 1;
ridiKinematics.rb_r = 0;
}
else
{
ridiKinematics.rb_f = 0;
ridiKinematics.rb_r = 1;
}
}
}

//class to handle button actions
class ButtonHandler implements ActionListener
{
public void actionPerformed(ActionEvent eb)
{
String s = eb.getActionCommand();
if(s == "Next")
{
if(ridiKinematics.rb_f == 1)
{
//if forward choice selected , go to the RIDI Forward Kinematics Input Screen
ridiInput rinp = new ridiInput();
setVisible(false);
}
else
{
//if reverse choice selected , go to the RIDI Reverse Kinematics Input Screen
revInput revi = new revInput();
setVisible(false);
}
}
if(s == "Exit")
{
int n = JOptionPane.showConfirmDialog(null,"Are you sure you want to
exit?","RIDI_GUI",JOptionPane.YES_NO_OPTION);
if(n == JOptionPane.YES_OPTION)
{
System.exit(0);
}
}
}
}
}
}

```

```

//ridiInput.java
//This class is called to bring up the Forward Kinematics Input Screen.

//program for defining the frame for RIDI Forward Kinematics Input Screen
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ridiInput
{
    public final static JFrame frame_input = new JFrame("RIDI-FORWARD-INPUT");
    public ridiInput()
    {
        //defining an object of the ridiInput1 class and addin properties to it
        ridiInput1 rinp1 = new ridiInput1();
        frame_input.getContentPane().setLayout(new BorderLayout());
        frame_input.setSize(800,600);

        //add the North and Center Panel
        frame_input.getContentPane().add(rinp1, BorderLayout.CENTER);
        frame_input.getContentPane().add(rinp1.pc, BorderLayout.CENTER);

        frame_input.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
        frame_input.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                {
                    int n = JOptionPane.showConfirmDialog(ridiInput.frame_input, "Are you sure you want to
exit?", "RIDI_GUI", JOptionPane.YES_NO_OPTION);
                    if(n == JOptionPane.YES_OPTION)
                    {
                        System.exit(0);
                    }
                }
            }
        });
        frame_input.setVisible(true);
    }
}

```

```

//ridiInput1.java

/*Program for defining the Panel which takes in the repaint method and goes
into the North part of ridiInput.java frame*/

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.Graphics;
import java.awt.Toolkit.*;

public class ridiInput1 extends JPanel
{
    /*defining the string variables to store values for diff. joints in initial
    value...*/
    public static String str_grp1_init;
    public static String str_wst1_init;
    public static String str_sho1_init;
    public static String str_elb1_init;
    public static String str_wrr1_init;
    public static String str_wrp1_init;
    public static String str_grp2_init;
    public static String str_wst2_init;
    public static String str_sho2_init;
    public static String str_elb2_init;
    public static String str_wrr2_init;
    public static String str_wrp2_init;

    /*defining the string variables to store values for diff. joints related to
    final value...*/
    public static String str_wst1_final;
    public static String str_sho1_final;
    public static String str_elb1_final;
    public static String str_wrr1_final;
    public static String str_wrp1_final;
    public static String str_wst2_final;
    public static String str_sho2_final;
    public static String str_elb2_final;
    public static String str_wrr2_final;
    public static String str_wrp2_final;

    // defining the string variables for direction of each joint...
    public static String str_grp1_dir;
    public static String str_wst1_dir;
    public static String str_sho1_dir;
    public static String str_elb1_dir;
    public static String str_wrr1_dir;
    public static String str_wrp1_dir;
    public static String str_grp2_dir;
    public static String str_wst2_dir;
    public static String str_sho2_dir;
    public static String str_elb2_dir;
    public static String str_wrr2_dir;
    public static String str_wrp2_dir;

    //String definitions for fields of combo boxes...

```

```

String[] jointPris = {"Open","Close"};
String[] jointRot = {"Clockwise","Anti-Clockwise"};

//define buttons for navigation
JButton finish_init = new JButton("Finish");
JButton kinem_init = new JButton("Back");

Insets ins = new Insets(2,2,2,2);

//define text components for channel numbers
JTextField chn1_input = new JTextField("1");
JTextField chn2_input = new JTextField("2");
JTextField chn3_input = new JTextField("3");
JTextField chn4_input = new JTextField("4");
JTextField chn5_input = new JTextField("5");
JTextField chn6_input = new JTextField("6");
JTextField chn7_input = new JTextField("7");
JTextField chn8_input = new JTextField("8");
JTextField chn9_input = new JTextField("9");
JTextField chn10_input = new JTextField("10");
JTextField chn11_input = new JTextField("11");
JTextField chn12_input = new JTextField("12");

//defining labels for the joint titles...
JLabel l_grp1_input = new JLabel("Gripper-1");
JLabel l_wst1_input = new JLabel("Waist-1");
JLabel l_sho1_input = new JLabel("Shoulder-1");
JLabel l_elb1_input = new JLabel("Elbow-1");
JLabel l_wrr1_input = new JLabel("Wrist Pitch-1");
JLabel l_wrp1_input = new JLabel("Wrist Roll-1");
JLabel l_grp2_input = new JLabel("Gripper-2");
JLabel l_wst2_input = new JLabel("Waist-2");
JLabel l_sho2_input = new JLabel("Shoulder-2");
JLabel l_elb2_input = new JLabel("Elbow-2");
JLabel l_wrr2_input = new JLabel("Wrist Pitch-2");
JLabel l_wrp2_input = new JLabel("Wrist Roll-2");

//Labels for range of individual joint
JLabel rws1 = new JLabel("0 - 300");
JLabel rsh1 = new JLabel("0 - 130");
JLabel rel1 = new JLabel("0 - 90");
JLabel rwr1 = new JLabel("0 - 180");
JLabel rwp1 = new JLabel("0 - 360");
JLabel rws2 = new JLabel("0 - 300");
JLabel rsh2 = new JLabel("0 - 130");
JLabel rel2 = new JLabel("0 - 90");
JLabel rwr2 = new JLabel("0 - 180");
JLabel rwp2 = new JLabel("0 - 360");

//defining the combo boxes and text field for initial values of joints
JComboBox grp1_init = new JComboBox(jointPris);
JTextField wst1_init = new JTextField(str_wst1_init);
JTextField sho1_init = new JTextField(str_sho1_init);

```

```

JTextField elb1_init = new JTextField(str_elb1_init);
JTextField wrp1_init = new JTextField(str_wrp1_init);
JTextField wrp1_init = new JTextField(str_wrp1_init);
JComboBox grp2_init = new JComboBox(jointPris);
JTextField wst2_init = new JTextField(str_wst2_init);
JTextField sho2_init = new JTextField(str_sho2_init);
JTextField elb2_init = new JTextField(str_elb2_init);
JTextField wrp2_init = new JTextField(str_wrp2_init);
JTextField wrp2_init = new JTextField(str_wrp2_init);

// define text fields for final values...
JTextField grp1_final = new JTextField("N/A");
JTextField wst1_final = new JTextField(str_wst1_final);
JTextField sho1_final = new JTextField(str_sho1_final);
JTextField elb1_final = new JTextField(str_elb1_final);
JTextField wrp1_final = new JTextField(str_wrp1_final);
JTextField wrp1_final = new JTextField(str_wrp1_final);
JTextField wrp1_final = new JTextField(str_wrp1_final);
JTextField grp2_final = new JTextField("N/A");
JTextField wst2_final = new JTextField(str_wst2_final);
JTextField sho2_final = new JTextField(str_sho2_final);
JTextField elb2_final = new JTextField(str_elb2_final);
JTextField wrp2_final = new JTextField(str_wrp2_final);
JTextField wrp2_final = new JTextField(str_wrp2_final);

//defining combo boxes for getting directions....
JComboBox grp1_choice = new JComboBox(jointPris);
JComboBox wst1_choice = new JComboBox(jointRot);
JComboBox sho1_choice = new JComboBox(jointRot);
JComboBox elb1_choice = new JComboBox(jointRot);
JComboBox wrp1_choice = new JComboBox(jointRot);
JComboBox wrp1_choice = new JComboBox(jointRot);
JComboBox wrp1_choice = new JComboBox(jointRot);
JComboBox grp2_choice = new JComboBox(jointPris);
JComboBox wst2_choice = new JComboBox(jointRot);
JComboBox sho2_choice = new JComboBox(jointRot);
JComboBox elb2_choice = new JComboBox(jointRot);
JComboBox wrp2_choice = new JComboBox(jointRot);
JComboBox wrp2_choice = new JComboBox(jointRot);

/*Panel which will take these different components and will go in the center
of the ridiInput.java*/
JPanel pc = new JPanel();

public ridiInput1()
{
//defining the properties of the panel
setLayout(null);
setBackground(Color.lightGray);
pc.setLayout(null);
setSize(800,40);

//set the textfield for channel numbers uneditable
chn1_input.setEditable(false);
chn2_input.setEditable(false);
chn3_input.setEditable(false);

```



```

chn4_input.setEditable(false);
chn5_input.setEditable(false);
chn6_input.setEditable(false);
chn7_input.setEditable(false);
chn8_input.setEditable(false);
chn9_input.setEditable(false);
chn10_input.setEditable(false);
chn11_input.setEditable(false);
chn12_input.setEditable(false);

//set background for channel number text fields...
chn1_input.setBackground(Color.white);
chn2_input.setBackground(Color.white);
chn3_input.setBackground(Color.white);
chn4_input.setBackground(Color.white);
chn5_input.setBackground(Color.white);
chn6_input.setBackground(Color.white);
chn7_input.setBackground(Color.white);
chn8_input.setBackground(Color.white);
chn9_input.setBackground(Color.white);
chn10_input.setBackground(Color.white);
chn11_input.setBackground(Color.white);
chn12_input.setBackground(Color.white);

//set bounds for buttons
finish_init.setBounds(195,500,75,30);
kinem_init.setBounds(25,500,75,30);
finish_init.setMargin(10);
kinem_init.setMargin(10);

// add item listeners to combo boxes for initial values...
wst1_init.addKeyListener(new KeyHandler(wst1_init));
sho1_init.addKeyListener(new KeyHandler(sho1_init));
elb1_init.addKeyListener(new KeyHandler(elb1_init));
wrr1_init.addKeyListener(new KeyHandler(wrr1_init));
wrp1_init.addKeyListener(new KeyHandler(wrp1_init));
wst2_init.addKeyListener(new KeyHandler(wst2_init));
sho2_init.addKeyListener(new KeyHandler(sho2_init));
elb2_init.addKeyListener(new KeyHandler(elb2_init));
wrr2_init.addKeyListener(new KeyHandler(wrr2_init));
wrp2_init.addKeyListener(new KeyHandler(wrp2_init));

//add key listeners to combo boxes to take only numbers...
wst1_final.addKeyListener(new KeyHandler(wst1_final));
sho1_final.addKeyListener(new KeyHandler(sho1_final));
elb1_final.addKeyListener(new KeyHandler(elb1_final));
wrr1_final.addKeyListener(new KeyHandler(wrr1_final));
wrp1_final.addKeyListener(new KeyHandler(wrp1_final));
wst2_final.addKeyListener(new KeyHandler(wst2_final));
sho2_final.addKeyListener(new KeyHandler(sho2_final));
elb2_final.addKeyListener(new KeyHandler(elb2_final));
wrr2_final.addKeyListener(new KeyHandler(wrr2_final));
wrp2_final.addKeyListener(new KeyHandler(wrp2_final));

//set bounds for textfield for channel numbers

```

```
chn1_input.setBounds(0,50,20,20);
chn2_input.setBounds(0,80,20,20);
chn3_input.setBounds(0,110,20,20);
chn4_input.setBounds(0,140,20,20);
chn5_input.setBounds(0,170,20,20);
chn6_input.setBounds(0,200,20,20);
chn7_input.setBounds(0,230,20,20);
chn8_input.setBounds(0,260,20,20);
chn9_input.setBounds(0,290,20,20);
chn10_input.setBounds(0,320,20,20);
chn11_input.setBounds(0,350,20,20);
chn12_input.setBounds(0,380,20,20);
```

```
//setting bounds for the joint title labels...
l_grp1_input.setBounds(40,42,75,30);
l_wst1_input.setBounds(40,72,75,30);
l_sho1_input.setBounds(40,102,75,30);
l_elb1_input.setBounds(40,132,75,30);
l_wrr1_input.setBounds(40,162,75,30);
l_wrp1_input.setBounds(40,192,75,30);
l_grp2_input.setBounds(40,222,75,30);
l_wst2_input.setBounds(40,252,75,30);
l_sho2_input.setBounds(40,282,75,30);
l_elb2_input.setBounds(40,312,75,30);
l_wrr2_input.setBounds(40,342,75,30);
l_wrp2_input.setBounds(40,372,75,30);
```

```
//set bounds for text boxes for initial values..
grp1_init.setBounds(135,50,75,20);
wst1_init.setBounds(135,80,75,20);
sho1_init.setBounds(135,110,75,20);
elb1_init.setBounds(135,140,75,20);
wrr1_init.setBounds(135,170,75,20);
wrp1_init.setBounds(135,200,75,20);
grp2_init.setBounds(135,230,75,20);
wst2_init.setBounds(135,260,75,20);
sho2_init.setBounds(135,290,75,20);
elb2_init.setBounds(135,320,75,20);
wrr2_init.setBounds(135,350,75,20);
wrp2_init.setBounds(135,380,75,20);
```

```
//setting the gripper fields uneditable for final Values...
grp1_final.setEditable(false);
grp2_final.setEditable(false);
```

```
//set background for textfields of final values...
grp1_final.setBackground(Color.white);
wst1_final.setBackground(Color.white);
sho1_final.setBackground(Color.white);
elb1_final.setBackground(Color.white);
wrr1_final.setBackground(Color.white);
wrp1_final.setBackground(Color.white);
grp2_final.setBackground(Color.white);
```

```

wst2_final.setBackground(Color.white);
sho2_final.setBackground(Color.white);
elb2_final.setBackground(Color.white);
wrr2_final.setBackground(Color.white);
wrp2_final.setBackground(Color.white);

//set bounds for textfields of final values...
grp1_final.setBounds(230,50,75,20);
wst1_final.setBounds(230,80,75,20);
sho1_final.setBounds(230,110,75,20);
elb1_final.setBounds(230,140,75,20);
wrr1_final.setBounds(230,170,75,20);
wrp1_final.setBounds(230,200,75,20);
grp2_final.setBounds(230,230,75,20);
wst2_final.setBounds(230,260,75,20);
sho2_final.setBounds(230,290,75,20);
elb2_final.setBounds(230,320,75,20);
wrr2_final.setBounds(230,350,75,20);
wrp2_final.setBounds(230,380,75,20);

//set bounds for JComboBox....
grp1_choice.setBounds(350,50,100,20);
wst1_choice.setBounds(350,80,100,20);
sho1_choice.setBounds(350,110,100,20);
elb1_choice.setBounds(350,140,100,20);
wrr1_choice.setBounds(350,170,100,20);
wrp1_choice.setBounds(350,200,100,20);
grp2_choice.setBounds(350,230,100,20);
wst2_choice.setBounds(350,260,100,20);
sho2_choice.setBounds(350,290,100,20);
elb2_choice.setBounds(350,320,100,20);
wrr2_choice.setBounds(350,350,100,20);
wrp2_choice.setBounds(350,380,100,20);

//set Bounds for labels...
rws1.setBounds(475,72,75,30);
rsh1.setBounds(475,102,75,30);
rel1.setBounds(475,132,75,30);
rwr1.setBounds(475,162,75,30);
rwp1.setBounds(475,192,75,30);
rws2.setBounds(475,252,75,30);
rsh2.setBounds(475,282,75,30);
rel2.setBounds(475,312,75,30);
rwr2.setBounds(475,342,75,30);
rwp2.setBounds(475,372,75,30);

//add buttons to the panel...
pc.add(finish_init);
pc.add(kinem_init);

//add channel numbers on the panel...
pc.add(chn1_input);
pc.add(chn2_input);
pc.add(chn3_input);
pc.add(chn4_input);

```

```

pc.add(chn5_input);
pc.add(chn6_input);
pc.add(chn7_input);
pc.add(chn8_input);
pc.add(chn9_input);
pc.add(chn10_input);
pc.add(chn11_input);
pc.add(chn12_input);

// adding joint titles on the panel...
pc.add(l_grp1_input);
pc.add(l_wst1_input);
pc.add(l_sho1_input);
pc.add(l_elb1_input);
pc.add(l_wrr1_input);
pc.add(l_wrp1_input);
pc.add(l_grp2_input);
pc.add(l_wst2_input);
pc.add(l_sho2_input);
pc.add(l_elb2_input);
pc.add(l_wrr2_input);
pc.add(l_wrp2_input);

//add initial value text fields and combo boxes on the panel...
pc.add(grp1_init);
pc.add(wst1_init);
pc.add(sho1_init);
pc.add(elb1_init);
pc.add(wrr1_init);
pc.add(wrp1_init);
pc.add(grp2_init);
pc.add(wst2_init);
pc.add(sho2_init);
pc.add(elb2_init);
pc.add(wrr2_init);
pc.add(wrp2_init);

//add final value textfields on the panel...
pc.add(grp1_final);
pc.add(wst1_final);
pc.add(sho1_final);
pc.add(elb1_final);
pc.add(wrr1_final);
pc.add(wrp1_final);
pc.add(grp2_final);
pc.add(wst2_final);
pc.add(sho2_final);
pc.add(elb2_final);
pc.add(wrr2_final);
pc.add(wrp2_final);

// add combo boxes for the direction on the panel...
pc.add(grp1_choice);
pc.add(wst1_choice);
pc.add(sho1_choice);

```

```

pc.add(elb1_choice);
pc.add(wrr1_choice);
pc.add(wrp1_choice);
pc.add(grp2_choice);
pc.add(wst2_choice);
pc.add(sho2_choice);
pc.add(elb2_choice);
pc.add(wrr2_choice);
pc.add(wrp2_choice);

//add labels for ranges
pc.add(rws1);
pc.add(rsh1);
pc.add(rel1);
pc.add(rwr1);
pc.add(rwp1);
pc.add(rws2);
pc.add(rsh2);
pc.add(rel2);
pc.add(rwr2);
pc.add(rwp2);

//add listener to the back button ...
kinem_init.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent ev)
    {
        ridiInput.frame_input.dispose();
        ridiKinematics rkin = new ridiKinematics();
    }
});

//ad listener to the finish button...
finish_init.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent ev)
    {
        // Getting the initial string values into the static variables defined..
        ridiInput1.str_grp1_init = (String) grp1_init.getSelectedItem();
        ridiInput1.str_wst1_init = (String) wst1_init.getText();
        ridiInput1.str_sho1_init = (String) sho1_init.getText();
        ridiInput1.str_elb1_init = (String) elb1_init.getText();
        ridiInput1.str_wrr1_init = (String) wrr1_init.getText();
        ridiInput1.str_wrp1_init = (String) wrp1_init.getText();
        ridiInput1.str_grp2_init = (String) grp2_init.getSelectedItem();
        ridiInput1.str_wst2_init = (String) wst2_init.getText();
        ridiInput1.str_sho2_init = (String) sho2_init.getText();
        ridiInput1.str_elb2_init = (String) elb2_init.getText();
        ridiInput1.str_wrr2_init = (String) wrr2_init.getText();
        ridiInput1.str_wrp2_init = (String) wrp2_init.getText();

        // Getting the final string values into the static variables defined...
        ridiInput1.str_wst1_final = (String) wst1_final.getText();
        ridiInput1.str_sho1_final = (String) sho1_final.getText();
        ridiInput1.str_elb1_final = (String) elb1_final.getText();
        ridiInput1.str_wrr1_final = (String) wrr1_final.getText();
    }
});

```

```

ridiInput1.str_wrp1_final = (String) wrp1_final.getText();
ridiInput1.str_wst2_final = (String) wst2_final.getText();
ridiInput1.str_sho2_final = (String) sho2_final.getText();
ridiInput1.str_elb2_final = (String) elb2_final.getText();
ridiInput1.str_wrr2_final = (String) wrr2_final.getText();
ridiInput1.str_wrp2_final = (String) wrp2_final.getText();

// Getting the directions string values for each static variables defined...
ridiInput1.str_grp1_dir = (String) grp1_choice.getSelectedItem();
ridiInput1.str_wst1_dir = (String) wst1_choice.getSelectedItem();
ridiInput1.str_sho1_dir = (String) sho1_choice.getSelectedItem();
ridiInput1.str_elb1_dir = (String) elb1_choice.getSelectedItem();
ridiInput1.str_wrr1_dir = (String) wrr1_choice.getSelectedItem();
ridiInput1.str_wrp1_dir = (String) wrp1_choice.getSelectedItem();
ridiInput1.str_grp2_dir = (String) grp2_choice.getSelectedItem();
ridiInput1.str_wst2_dir = (String) wst2_choice.getSelectedItem();
ridiInput1.str_sho2_dir = (String) sho2_choice.getSelectedItem();
ridiInput1.str_elb2_dir = (String) elb2_choice.getSelectedItem();
ridiInput1.str_wrr2_dir = (String) wrr2_choice.getSelectedItem();
ridiInput1.str_wrp2_dir = (String) wrp2_choice.getSelectedItem();

//go to the main activity screen
ridiChange rch = new ridiChange();
ridiInput.frame_input.setVisible(false);

}
});
repaint();
}

//paint method for the parent panel class
public void paint(Graphics g)
{
Font f_input = new Font("Serif",Font.BOLD,14);
Graphics2D g2 = (Graphics2D) g;
g2.setFont(f_input);
g2.drawString("Chn #",0,35);
g2.drawString("Joint",40,35);
g2.drawString("Initial State",135,35);
g2.drawString("Final State",230,35);
g2.drawString("Direction",350,35);
g2.drawString("Range",475,35);
}
}

//Key handler class
class KeyHandler implements KeyListener
{
JTextField abc1;
public KeyHandler(JTextField abc)
{
abc1 = abc;
}
public void keyTyped(KeyEvent ek)
{
}
}

```

```

public void keyReleased(KeyEvent ekp)
{
}
public void keyPressed(KeyEvent ek)
{
    function_init(ek);
}
public void function_init(KeyEvent ekp)
{
    int nk = ekp.getKeyCode();
    if((nk > KeyEvent.VK_9) || (nk < KeyEvent.VK_0))
    {
        if(!((nk == KeyEvent.VK_BACK_SPACE) || (nk == KeyEvent.VK_DELETE)))
        {
            JOptionPane.showMessageDialog(ridiInput.frame_input, "Invalid Key Entered: Enter
Again", "ALERT", JOptionPane.ERROR_MESSAGE);
            abc1.setText("");
        }
    }
}
}
}

```

```

//revInput.java

//Program for RIDI Reverse Kinematics Input Screen

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.Graphics;

public class revInput
{
    //Defining the Frame for Ridi Reverse Kinemaics Input Screen
    JFrame frame_revInp = new JFrame("RIDID-REVERSE-INPUT");

    public static String str_x1_init;
    public static String str_y1_init;
    public static String str_z1_init;
    public static String str_t4_init;
    public static String str_t5_init;

    public static String str_x1_fin;
    public static String str_y1_fin;
    public static String str_z1_fin;
    public static String str_t4_fin;
    public static String str_t5_fin;

    public static String str_x2_init;
    public static String str_y2_init;
    public static String str_z2_init;
    public static String str_t42_init;
    public static String str_t52_init;

    public static String str_x2_fin;
    public static String str_y2_fin;
    public static String str_z2_fin;
    public static String str_t42_fin;
    public static String str_t52_fin;

    //Text Field for initial values of Robot1
    JTextField x_init = new JTextField();
    JTextField y_init = new JTextField();
    JTextField z_init = new JTextField();
    JTextField th4_init = new JTextField();
    JTextField th5_init = new JTextField();

    //Text Field for final values of Robot1
    JTextField x_fin = new JTextField();
    JTextField y_fin = new JTextField();
    JTextField z_fin = new JTextField();
    JTextField th4_fin = new JTextField();
    JTextField th5_fin = new JTextField();

    //Text Field for initial values of Robot2
    JTextField x2_init = new JTextField();

```



```

JTextField y2_init = new JTextField();
JTextField z2_init = new JTextField();
JTextField th42_init = new JTextField();
JTextField th52_init = new JTextField();

//Text Field for final values of Robot2
JTextField x2_fin = new JTextField();
JTextField y2_fin = new JTextField();
JTextField z2_fin = new JTextField();
JTextField th42_fin = new JTextField();
JTextField th52_fin = new JTextField();

JLabel l_r1_ini = new JLabel("Robot 1 - Initial");
JLabel l_r2_ini = new JLabel("Robot 2 - Initial");
JLabel l_r1_fin = new JLabel("Robot 1 - Final");
JLabel l_r2_fin = new JLabel("Robot 2 - Final");

// Label for initial values of robot 1
JLabel x_r1_init = new JLabel("X");
JLabel y_r1_init = new JLabel("Y");
JLabel z_r1_init = new JLabel("Z");
JLabel t4_r1_init = new JLabel("THETA 4");
JLabel t5_r1_init = new JLabel("THETA 5");

//Label for final values of robot1
JLabel x_r1_fin = new JLabel("X");
JLabel y_r1_fin = new JLabel("Y");
JLabel z_r1_fin = new JLabel("Z");
JLabel t4_r1_fin = new JLabel("THETA");
JLabel t5_r1_fin = new JLabel("THETA");

//Label for initial values of robot 2
JLabel x_r2_init = new JLabel("X");
JLabel y_r2_init = new JLabel("Y");
JLabel z_r2_init = new JLabel("Z");
JLabel t4_r2_init = new JLabel("THETA 4");
JLabel t5_r2_init = new JLabel("THETA 5");

//Label for final values of robot2
JLabel x_r2_fin = new JLabel("X");
JLabel y_r2_fin = new JLabel("Y");
JLabel z_r2_fin = new JLabel("Z");
JLabel t4_r2_fin = new JLabel("THETA 4");
JLabel t5_r2_fin = new JLabel("THETA 5");

//Buttons for navigation
JButton back = new JButton("Back");
JButton finish = new JButton("Finish");

//Defining Panels
JPanel panel_main = new JPanel();
JPanel panel_south = new JPanel();

public revInput()

```

```

{
//Defining the Frame and specifying its properties
frame_revInp.getContentPane().setLayout(new BorderLayout());
frame_revInp.setSize(800,600);

//adding properties to panels defined
panel_main.setPreferredSize(new Dimension(600,500));
panel_south.setPreferredSize(new Dimension(800,100));
panel_main.setLayout(null);
panel_south.setLayout(null);

//Setting bounds to the buttons
back.setBounds(550,50,100,30);
finish.setBounds(670,50,100,30);

//setting bounds to the labels defined
l_r1_ini.setBounds(130,50,100,30);
l_r1_fin.setBounds(530, 50, 100,30);
l_r2_ini.setBounds(130,300,100,30);
l_r2_fin.setBounds(530,300,100,30);

//Setting bounds to the text fields defined
//robot 1 initial values
x_init.setBounds(130,80,75,20);
y_init.setBounds(130,110,75,20);
z_init.setBounds(130,140,75,20);
th4_init.setBounds(130,170,75,20);
th5_init.setBounds(130,200,75,20);
x_r1_init.setBounds(30,80,75,25);
y_r1_init.setBounds(30,110,75,25);
z_r1_init.setBounds(30,140,75,25);
t4_r1_init.setBounds(30,170,75,25);
t5_r1_init.setBounds(30,200,75,25);

//robot 1 final values
x_fin.setBounds(130,330,75,20);
y_fin.setBounds(130,360,75,20);
z_fin.setBounds(130,390,75,20);
th4_fin.setBounds(130,420,75,20);
th5_fin.setBounds(130,450,75,20);
x_r1_fin.setBounds(30,330,75,25);
y_r1_fin.setBounds(30,360,75,25);
z_r1_fin.setBounds(30,390,75,25);
t4_r1_fin.setBounds(30,420,75,25);
t5_r1_fin.setBounds(30,450,75,25);
//robot 2 initial values
x2_init.setBounds(530,80,75,20);
y2_init.setBounds(530,110,75,20);
z2_init.setBounds(530,140,75,20);
th42_init.setBounds(530,170,75,20);
th52_init.setBounds(530,200,75,20);
x_r2_init.setBounds(430,80,75,25);
y_r2_init.setBounds(430,110,75,25);
z_r2_init.setBounds(430,140,75,25);
t4_r2_init.setBounds(430,170,75,25);
t5_r2_init.setBounds(430,200,75,25);

```

```
//robot 2 final values
x2_fin.setBounds(530,330,75,20);
y2_fin.setBounds(530,360,75,20);
z2_fin.setBounds(530,390,75,20);
th42_fin.setBounds(530,420,75,20);
th52_fin.setBounds(530,450,75,20);
x_r2_fin.setBounds(430,330,75,25);
y_r2_fin.setBounds(430,360,75,25);
z_r2_fin.setBounds(430,390,75,25);
t4_r2_fin.setBounds(430,420,75,25);
t5_r2_fin.setBounds(430,450,75,25);
```

```
//adding labels to the panel
panel_main.add(l_r1_ini);
panel_main.add(l_r1_fin);
panel_main.add(l_r2_ini);
panel_main.add(l_r2_fin);
```

```
//adding text field to the panel
```

```
//robot1
panel_main.add(x_init);
panel_main.add(y_init);
panel_main.add(z_init);
panel_main.add(x_fin);
panel_main.add(y_fin);
panel_main.add(z_fin);
panel_main.add(th4_init);
panel_main.add(th4_fin);
panel_main.add(th5_init);
panel_main.add(th5_fin);
panel_main.add(x_r1_init);
panel_main.add(y_r1_init);
panel_main.add(z_r1_init);
panel_main.add(t4_r1_init);
panel_main.add(t5_r1_init);
panel_main.add(x_r1_fin);
panel_main.add(y_r1_fin);
panel_main.add(z_r1_fin);
panel_main.add(t4_r1_fin);
panel_main.add(t5_r1_fin);
```

```
//robot 2
panel_main.add(x2_init);
panel_main.add(y2_init);
panel_main.add(z2_init);
panel_main.add(x2_fin);
panel_main.add(y2_fin);
panel_main.add(z2_fin);
panel_main.add(th42_init);
panel_main.add(th42_fin);
panel_main.add(th52_init);
panel_main.add(th52_fin);
panel_main.add(x_r2_init);
panel_main.add(y_r2_init);
```

```

panel_main.add(z_r2_init);
panel_main.add(t4_r2_init);
panel_main.add(t5_r2_init);
panel_main.add(x_r2_fin);
panel_main.add(y_r2_fin);
panel_main.add(z_r2_fin);
panel_main.add(t4_r2_fin);
panel_main.add(t5_r2_fin);

//add buttons to the bottom panel
panel_south.add(back);
panel_south.add(finish);

//add panels to the main frame
frame_revInp.getContentPane().add(panel_main, BorderLayout.CENTER);
frame_revInp.getContentPane().add(panel_south, BorderLayout.SOUTH);

//add action listener to the buttons
back.addActionListener(new ButtonHandler(frame_revInp));
finish.addActionListener(new ButtonHandler(frame_revInp));

frame_revInp.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);

frame_revInp.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        int n = JOptionPane.showConfirmDialog(frame_revInp,"Are you sure you want to
exit?","RIDI_GUI",JOptionPane.YES_NO_OPTION);
        if(n == JOptionPane.YES_OPTION)
        {
            System.exit(0);
        }
    }
});

frame_revInp.setVisible(true);
}

//Class for handling Button Action events
class ButtonHandler implements ActionListener
{
    JFrame rn;
    ButtonHandler(JFrame ri)
    {
        rn = ri;
    }
    String s;
    public void actionPerformed(ActionEvent ev)
    {
        s = ev.getActionCommand();
        if(s == "Back")
        {

```

```

//Go to the RIDI Kinematics Screen
ridiKinematics.rb_f = 1;
ridiKinematics.rb_r = 0;
rn.dispose();
ridiKinematics rki = new ridiKinematics();
}
if(s== "Finish")
{
revInput.str_x1_init = x_init.getText();
revInput.str_y1_init = y_init.getText();
revInput.str_z1_init = z_init.getText();
revInput.str_t4_init = th4_init.getText();
revInput.str_t5_init = th5_init.getText();

revInput.str_x1_fin = x_fin.getText();
revInput.str_y1_fin = y_fin.getText();
revInput.str_z1_fin = z_fin.getText();
revInput.str_t4_fin = th4_fin.getText();
revInput.str_t5_fin = th5_fin.getText();

revInput.str_x2_init = x2_init.getText();
revInput.str_y2_init = y2_init.getText();
revInput.str_z2_init = z2_init.getText();
revInput.str_t42_init = th42_init.getText();
revInput.str_t52_init = th52_init.getText();

revInput.str_x2_fin = x2_fin.getText();
revInput.str_y2_fin = y2_fin.getText();
revInput.str_z2_fin = z2_fin.getText();
revInput.str_t42_fin = th42_fin.getText();
revInput.str_t52_fin = th52_fin.getText();

ridiReverse1 r1 = new ridiReverse1();
ridiReverse1f r1f = new ridiReverse1f();
ridiReverse2 r2 = new ridiReverse2();
ridiReverse2f r2f = new ridiReverse2f();

revChange reinp = new revChange();

}

}
}
}

```

```

//ridiChange.java

//This is the program for the main activity screen in forward mode

//Program for GUI Main Activity Screen

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.BorderFactory;
import javax.swing.border.Border;

public class ridiChange
{
    //Defining panel which goes into the scroll pane
    JPanel panel_sp_main = new JPanel();

    JScrollPane sp_main = new JScrollPane(panel_sp_main);

    public static final JFrame frame_main = new JFrame("Main");

    Insets ins = new Insets(2,2,2,2);

    JPanel panel1_main = new JPanel();
    JPanel panel2_main = new JPanel();

    JButton b1 = new JButton("1");
    JButton b2 = new JButton("2");
    JButton b3 = new JButton("3");
    JButton b4 = new JButton("4");
    JButton b5 = new JButton("5");
    JButton b6 = new JButton("6");
    JButton b7 = new JButton("7");
    JButton b8 = new JButton("8");
    JButton b9 = new JButton("9");
    JButton b10 = new JButton("10");
    JButton b11 = new JButton("11");
    JButton b12 = new JButton("12");

    JButton start_sp_main = new JButton("Start");
    JButton back_sp_main = new JButton("Back");

    JPanel jp1;

    Box b_main = new Box(BoxLayout.Y_AXIS);

    panel_gr1 pgr1 = new panel_gr1();
    panel_ws1_internal pws1 = new panel_ws1_internal();
    panel_sh1_internal pshi1 = new panel_sh1_internal();
    panel_el1_internal peli1 = new panel_el1_internal();
    panel_wr1_internal pwri1 = new panel_wr1_internal();
    panel_wp1_internal pwpi1 = new panel_wp1_internal();
    panel_gr2 pgr2 = new panel_gr2();
    panel_ws2_internal pws2 = new panel_ws2_internal();
    panel_sh2_internal pshi2 = new panel_sh2_internal();

```

```

panel_el2_internal peli2 = new panel_el2_internal();
panel_wr2_internal pwri2 = new panel_wr2_internal();
panel_wp2_internal pwpi2 = new panel_wp2_internal();

//panels for anti-clockwise kinematics
panel_ws1 pws1 = new panel_ws1();
panel_sh1 psh1 = new panel_sh1();
panel_el1 pel1 = new panel_el1();
panel_wr1 pwr1 = new panel_wr1();
panel_wp1 pwp1 = new panel_wp1();
panel_ws2 pws2 = new panel_ws2();
panel_sh2 psh2 = new panel_sh2();
panel_el2 pel2 = new panel_el2();
panel_wr2 pwr2 = new panel_wr2();
panel_wp2 pwp2 = new panel_wp2();

// Constructor for class ridiChange

public ridiChange()
{

    sp_main.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

    ridiChange.frame_main.setResizable(false);
    ridiChange.frame_main.getContentPane().setLayout(new BorderLayout());
    ridiChange.frame_main.setSize(800,600);

    ridiChange.frame_main.getContentPane().add(sp_main, BorderLayout.CENTER);
    ridiChange.frame_main.getContentPane().add(panel1_main, BorderLayout.NORTH);
    ridiChange.frame_main.getContentPane().add(panel2_main, BorderLayout.SOUTH);

    //setting panel_sp_main to border layout...
    panel_sp_main.setLayout(new BorderLayout());

    //set sizes for the panels ...
    panel_sp_main.setPreferredSize(new Dimension(800,900));

    start_sp_main.setBackground(Color.pink);
    back_sp_main.setBackground(Color.pink);
    start_sp_main.setForeground(Color.black);
    back_sp_main.setForeground(Color.black);

    jp1 = new JPanel();
    jp1.setPreferredSize(new Dimension(600,70));
    jp1.setLayout(new BorderLayout());
    b1.setMargin(ins);
    b1.setBackground(Color.darkGray);
    b1.setForeground(Color.white);
    b1.setPreferredSize(new Dimension(40,70));
    jp1.add(b1, BorderLayout.WEST);
    jp1.add(pgr1, BorderLayout.CENTER);

```

```
b_main.add(jp1);
```

```
jp1 = new JPanel();  
jp1.setPreferredSize(new Dimension(600,70));  
jp1.setLayout(new BorderLayout());  
b2.setMargin(ins);  
b2.setBackground(Color.lightGray);  
b2.setPreferredSize(new Dimension(40,70));  
jp1.add(b2, BorderLayout.WEST);  
if(ridiInput1.str_wst1_dir == "Clockwise")  
{  
    jp1.add(pws1, BorderLayout.CENTER);  
}  
else  
{  
    jp1.add(pws1, BorderLayout.CENTER);  
}  
b_main.add(jp1);
```

```
jp1 = new JPanel();  
jp1.setPreferredSize(new Dimension(600,70));  
jp1.setLayout(new BorderLayout());  
b3.setMargin(ins);  
b3.setBackground(Color.darkGray);  
b3.setForeground(Color.white);  
b3.setPreferredSize(new Dimension(40,70));  
jp1.add(b3, BorderLayout.WEST);  
if(ridiInput1.str_sho1_dir == "Clockwise")  
{  
    jp1.add(psh1, BorderLayout.CENTER);  
}  
else  
{  
    jp1.add(psh1, BorderLayout.CENTER);  
}  
b_main.add(jp1);
```

```
jp1 = new JPanel();  
jp1.setPreferredSize(new Dimension(600,70));  
jp1.setLayout(new BorderLayout());  
b4.setMargin(ins);  
b4.setBackground(Color.lightGray);  
b4.setPreferredSize(new Dimension(40,70));  
jp1.add(b4, BorderLayout.WEST);  
if(ridiInput1.str_elb1_dir == "Clockwise")  
{  
    jp1.add(peli1, BorderLayout.CENTER);  
}  
else  
{  
    jp1.add(pel1, BorderLayout.CENTER);  
}  
b_main.add(jp1);
```

```
jp1 = new JPanel();
```



```

jp1.setPreferredSize(new Dimension(600,70));
jp1.setLayout(new BorderLayout());
b5.setMargin(ins);
b5.setBackground(Color.darkGray);
b5.setForeground(Color.white);
b5.setPreferredSize(new Dimension(40,70));
jp1.add(b5, BorderLayout.WEST);
if(ridiInput1.str_wrr1_dir == "Clockwise")
{
    jp1.add(pwri1, BorderLayout.CENTER);
}
else
{
    jp1.add(pwr1, BorderLayout.CENTER);
}
b_main.add(jp1);

```

```

jp1 = new JPanel();
jp1.setPreferredSize(new Dimension(600,70));
jp1.setLayout(new BorderLayout());
b6.setMargin(ins);
b6.setBackground(Color.lightGray);
b6.setPreferredSize(new Dimension(40,70));
jp1.add(b6, BorderLayout.WEST);
if(ridiInput1.str_wrp1_dir == "Clockwise")
{
    jp1.add(pwpi1, BorderLayout.CENTER);
}
else
{
    jp1.add(pwp1, BorderLayout.CENTER);
}
b_main.add(jp1);

```

```

jp1 = new JPanel();
jp1.setPreferredSize(new Dimension(600,70));
jp1.setLayout(new BorderLayout());
b7.setMargin(ins);
b7.setBackground(Color.darkGray);
b7.setForeground(Color.white);
b7.setPreferredSize(new Dimension(40,70));
jp1.add(b7, BorderLayout.WEST);
jp1.add(pgr2, BorderLayout.CENTER);
b_main.add(jp1);

```

```

jp1 = new JPanel();
jp1.setPreferredSize(new Dimension(600,70));
jp1.setLayout(new BorderLayout());
b8.setMargin(ins);
b8.setBackground(Color.lightGray);
b8.setPreferredSize(new Dimension(40,70));
jp1.add(b8, BorderLayout.WEST);
if(ridiInput1.str_wst2_dir == "Clockwise")
{
    jp1.add(pwsi2, BorderLayout.CENTER);
}

```

```

else
{
jp1.add(pws2, BorderLayout.CENTER);
}
b_main.add(jp1);

jp1 = new JPanel();
jp1.setPreferredSize(new Dimension(600,70));
jp1.setLayout(new BorderLayout());
b9.setMargin(10);
b9.setBackground(Color.darkGray);
b9.setForeground(Color.white);
b9.setPreferredSize(new Dimension(40,70));
jp1.add(b9, BorderLayout.WEST);
if(ridiInput1.str_sho2_dir == "Clockwise")
{
jp1.add(pshi2, BorderLayout.CENTER);
}
else
{
jp1.add(psh2, BorderLayout.CENTER);
}
b_main.add(jp1);

jp1 = new JPanel();
jp1.setPreferredSize(new Dimension(600,70));
jp1.setLayout(new BorderLayout());
b10.setMargin(10);
b10.setBackground(Color.lightGray);
b10.setPreferredSize(new Dimension(40,70));
jp1.add(b10, BorderLayout.WEST);
if(ridiInput1.str_elb2_dir == "Clockwise")
{
jp1.add(peli2, BorderLayout.CENTER);
}
else
{
jp1.add(pel2, BorderLayout.CENTER);
}
b_main.add(jp1);

jp1 = new JPanel();
jp1.setPreferredSize(new Dimension(600,70));
jp1.setLayout(new BorderLayout());
b11.setMargin(10);
b11.setBackground(Color.darkGray);
b11.setForeground(Color.white);
b11.setPreferredSize(new Dimension(40,70));
jp1.add(b11, BorderLayout.WEST);
if(ridiInput1.str_wrr2_dir == "Clockwise")
{
jp1.add(pwri2, BorderLayout.CENTER);
}
else
{
jp1.add(pwr2, BorderLayout.CENTER);
}

```

```

    }
    b_main.add(jp1);

    jp1 = new JPanel();
    jp1.setPreferredSize(new Dimension(600,70));
    jp1.setLayout(new BorderLayout());
    b12.setMargin(10);
    b12.setBackground(Color.lightGray);
    b12.setPreferredSize(new Dimension(40,70));
    jp1.add(b12, BorderLayout.WEST);
    if(ridiInput1.str_wrp2_dir == "Clockwise")
    {
        jp1.add(pwpi2, BorderLayout.CENTER);
    }
    else
    {
        jp1.add(pwp2, BorderLayout.CENTER);
    }
    b_main.add(jp1);

    pan2 p2 = new pan2();
    panel_sp_main.add(b_main, BorderLayout.CENTER);
    panel_sp_main.add(p2, BorderLayout.EAST);
    panel2_main.setBackground(Color.blue);
    panel2_main.add(start_sp_main);
    panel2_main.add(back_sp_main);

    panel1_main.setBackground(Color.red);

    ridiChange.frame_main.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);

    ridiChange.frame_main.addWindowListener(new WindowAdapter()
    {
        public void windowClosing(WindowEvent e)
        {
            {
                int n = JOptionPane.showConfirmDialog(ridiChange.frame_main,"Are you sure you want to
                exit?","RIDI_GUI",JOptionPane.YES_NO_OPTION);
                if(n == JOptionPane.YES_OPTION)
                {
                    System.exit(0);
                }
            }
        }
    });
    start_sp_main.addActionListener(new ButtonHandler());
    back_sp_main.addActionListener(new ButtonHandler());
    frame_main.setVisible(true);
}

class ButtonHandler implements ActionListener
{
    public void actionPerformed(ActionEvent ev1)
    {
        b1Thread bt1;
    }
}

```

```

b2Thread bt2;

String sps = ev1.getActionCommand();

if(sps == "Start")
{
    bt1 = new b1Thread();
    bt2 = new b2Thread();
    new Thread(pwsi1).start();
    new Thread(pshi1).start();
    new Thread(peli1).start();
    new Thread(pwri1).start();
    new Thread(pwpi1).start();
    new Thread(pwsi2).start();
    new Thread(pshi2).start();
    new Thread(peli2).start();
    new Thread(pwri2).start();
    new Thread(pwpi2).start();
    new Thread(pws1).start();
    new Thread(psh1).start();
    new Thread(pel1).start();
    new Thread(pwr1).start();
    new Thread(pwp1).start();
    new Thread(pws2).start();
    new Thread(psh2).start();
    new Thread(pel2).start();
    new Thread(pwr2).start();
    new Thread(pwp2).start();
}
if(sps == "Back")
{
    ridiInput rinp2 = new ridiInput();
    ridiChange.frame_main.dispose();
}
}
}
}
}

```

```

//revChange.java

//Main activity Screen for Reverse mode

//Program for GUI Main Activity Screen as related to reverse kinematics

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.BorderFactory;
import javax.swing.border.Border;

public class revChange
{
    //Defining panel which goes into the scroll pane
    JPanel panel_sp_main = new JPanel();

    JScrollPane sp_main = new JScrollPane(panel_sp_main);

    public static final JFrame frame_rmain = new JFrame("Main");

    Insets ins = new Insets(2,2,2,2);

    JPanel panel1_main = new JPanel();
    JPanel panel2_main = new JPanel();

    JButton b1 = new JButton("1");
    JButton b2 = new JButton("2");
    JButton b3 = new JButton("3");
    JButton b4 = new JButton("4");
    JButton b5 = new JButton("5");
    JButton b6 = new JButton("6");
    JButton b7 = new JButton("7");
    JButton b8 = new JButton("8");
    JButton b9 = new JButton("9");
    JButton b10 = new JButton("10");
    JButton b11 = new JButton("11");
    JButton b12 = new JButton("12");

    JButton start_sp_main = new JButton("Start");
    JButton back_sp_main = new JButton("Back");

    JPanel jp1;

    Box b_main = new Box(BoxLayout.Y_AXIS);

    panel_gr1 pgr1 = new panel_gr1();
    rev_ws1_internal rws1 = new rev_ws1_internal();
    rev_sh1_internal rshi1 = new rev_sh1_internal();
    rev_el1_internal reli1 = new rev_el1_internal();
    rev_wr1_internal rwri1 = new rev_wr1_internal();
    rev_wp1_internal rwp1 = new rev_wp1_internal();
    panel_gr2 pgr2 = new panel_gr2();
    rev_ws2_internal rws2 = new rev_ws2_internal();
    rev_sh2_internal rshi2 = new rev_sh2_internal();

```

```

rev_el2_internal reli2 = new rev_el2_internal();
rev_wr2_internal rwri2 = new rev_wr2_internal();
rev_wp2_internal rwpi2 = new rev_wp2_internal();

//panels for anti-clockwise kinematics
rev_ws1 rws1 = new rev_ws1();
rev_sh1 rsh1 = new rev_sh1();
rev_el1 rel1 = new rev_el1();
rev_wr1 rwr1 = new rev_wr1();
rev_wp1 rwp1 = new rev_wp1();
rev_ws2 rws2 = new rev_ws2();
rev_sh2 rsh2 = new rev_sh2();
rev_el2 rel2 = new rev_el2();
rev_wr2 rwr2 = new rev_wr2();
rev_wp2 rwp2 = new rev_wp2();

// Constructor for class ridiChange

public revChange()
{

    sp_main.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

    revChange.frame_rmain.setResizable(false);
    revChange.frame_rmain.getContentPane().setLayout(new BorderLayout());
    revChange.frame_rmain.setSize(800,600);

    revChange.frame_rmain.getContentPane().add(sp_main, BorderLayout.CENTER);
    revChange.frame_rmain.getContentPane().add(panel1_main, BorderLayout.NORTH);
    revChange.frame_rmain.getContentPane().add(panel2_main, BorderLayout.SOUTH);

    //setting panel_sp_main to border layout...
    panel_sp_main.setLayout(new BorderLayout());

    //set sizes for the panels ...
    panel_sp_main.setPreferredSize(new Dimension(800,900));

    start_sp_main.setBackground(Color.pink);
    back_sp_main.setBackground(Color.pink);
    start_sp_main.setForeground(Color.black);
    back_sp_main.setForeground(Color.black);

    jp1 = new JPanel();
    jp1.setPreferredSize(new Dimension(600,70));
    jp1.setLayout(new BorderLayout());
    b1.setMargin(10);
    b1.setBackground(Color.darkGray);
    b1.setForeground(Color.white);
    b1.setPreferredSize(new Dimension(40,70));
    jp1.add(b1, BorderLayout.WEST);
    jp1.add(pgr1, BorderLayout.CENTER);

```

```
b_main.add(jp1);
```

```
jp1 = new JPanel();  
jp1.setPreferredSize(new Dimension(600,70));  
jp1.setLayout(new BorderLayout());  
b2.setMargin(10);  
b2.setBackground(Color.lightGray);  
b2.setPreferredSize(new Dimension(40,70));  
jp1.add(b2, BorderLayout.WEST);  
if(ridiReverse1.t1 >= ridiReverse1f.t1f)  
{  
    jp1.add(rws1, BorderLayout.CENTER);  
}  
else  
{  
    jp1.add(rws1, BorderLayout.CENTER);  
}  
b_main.add(jp1);
```

```
jp1 = new JPanel();  
jp1.setPreferredSize(new Dimension(600,70));  
jp1.setLayout(new BorderLayout());  
b3.setMargin(10);  
b3.setBackground(Color.darkGray);  
b3.setForeground(Color.white);  
b3.setPreferredSize(new Dimension(40,70));  
jp1.add(b3, BorderLayout.WEST);  
if(ridiReverse1.t2 >= ridiReverse1f.t2f)  
{  
    jp1.add(rsh1, BorderLayout.CENTER);  
}  
else  
{  
    jp1.add(rsh1, BorderLayout.CENTER);  
}  
b_main.add(jp1);
```

```
jp1 = new JPanel();  
jp1.setPreferredSize(new Dimension(600,70));  
jp1.setLayout(new BorderLayout());  
b4.setMargin(10);  
b4.setBackground(Color.lightGray);  
b4.setPreferredSize(new Dimension(40,70));  
jp1.add(b4, BorderLayout.WEST);  
if(ridiReverse1.t3 >= ridiReverse1f.t3f)  
{  
    jp1.add(re11, BorderLayout.CENTER);  
}  
else  
{  
    jp1.add(re11, BorderLayout.CENTER);  
}  
b_main.add(jp1);
```

```
jp1 = new JPanel();
```

```

jp1.setPreferredSize(new Dimension(600,70));
jp1.setLayout(new BorderLayout());
b5.setMargin(ins);
b5.setBackground(Color.darkGray);
b5.setForeground(Color.white);
b5.setPreferredSize(new Dimension(40,70));
jp1.add(b5, BorderLayout.WEST);
if(ridiReverse1.t4 >= ridiReverse1f.t4f)
{
    jp1.add(rwri1, BorderLayout.CENTER);
}
else
{
    jp1.add(rwr1, BorderLayout.CENTER);
}
b_main.add(jp1);

```

```

jp1 = new JPanel();
jp1.setPreferredSize(new Dimension(600,70));
jp1.setLayout(new BorderLayout());
b6.setMargin(ins);
b6.setBackground(Color.lightGray);
b6.setPreferredSize(new Dimension(40,70));
jp1.add(b6, BorderLayout.WEST);
if(ridiReverse1.t5 >= ridiReverse1f.t5f)
{
    jp1.add(rwpi1, BorderLayout.CENTER);
}
else
{
    jp1.add(rwp1, BorderLayout.CENTER);
}
b_main.add(jp1);

```

```

jp1 = new JPanel();
jp1.setPreferredSize(new Dimension(600,70));
jp1.setLayout(new BorderLayout());
b7.setMargin(ins);
b7.setBackground(Color.darkGray);
b7.setForeground(Color.white);
b7.setPreferredSize(new Dimension(40,70));
jp1.add(b7, BorderLayout.WEST);
jp1.add(pgr2, BorderLayout.CENTER);
b_main.add(jp1);

```

```

jp1 = new JPanel();
jp1.setPreferredSize(new Dimension(600,70));
jp1.setLayout(new BorderLayout());
b8.setMargin(ins);
b8.setBackground(Color.lightGray);
b8.setPreferredSize(new Dimension(40,70));
jp1.add(b8, BorderLayout.WEST);
if(ridiReverse2.th1 >= ridiReverse2f.th2f)
{
    jp1.add(rwsi2, BorderLayout.CENTER);
}

```



```

else
{
jp1.add(rws2, BorderLayout.CENTER);
}
b_main.add(jp1);

jp1 = new JPanel();
jp1.setPreferredSize(new Dimension(600,70));
jp1.setLayout(new BorderLayout());
b9.setMargin(ins);
b9.setBackground(Color.darkGray);
b9.setForeground(Color.white);
b9.setPreferredSize(new Dimension(40,70));
jp1.add(b9, BorderLayout.WEST);
if(ridiReverse2.th2 >= ridiReverse2f.th2f)
{
jp1.add(rshi2, BorderLayout.CENTER);
}
else
{
jp1.add(rsh2, BorderLayout.CENTER);
}
b_main.add(jp1);

jp1 = new JPanel();
jp1.setPreferredSize(new Dimension(600,70));
jp1.setLayout(new BorderLayout());
b10.setMargin(ins);
b10.setBackground(Color.lightGray);
b10.setPreferredSize(new Dimension(40,70));
jp1.add(b10, BorderLayout.WEST);
if(ridiReverse2.th3 >= ridiReverse2f.th3f)
{
jp1.add(rei2, BorderLayout.CENTER);
}
else
{
jp1.add(rel2, BorderLayout.CENTER);
}
b_main.add(jp1);

jp1 = new JPanel();
jp1.setPreferredSize(new Dimension(600,70));
jp1.setLayout(new BorderLayout());
b11.setMargin(ins);
b11.setBackground(Color.darkGray);
b11.setForeground(Color.white);
b11.setPreferredSize(new Dimension(40,70));
jp1.add(b11, BorderLayout.WEST);
if(ridiReverse2.th4 >= ridiReverse2f.th4f)
{
jp1.add(rwri2, BorderLayout.CENTER);
}
else
{
jp1.add(rwr2, BorderLayout.CENTER);
}

```

```

    }
    b_main.add(jp1);

    jp1 = new JPanel();
    jp1.setPreferredSize(new Dimension(600,70));
    jp1.setLayout(new BorderLayout());
    b12.setMargin(10);
    b12.setBackground(Color.lightGray);
    b12.setPreferredSize(new Dimension(40,70));
    jp1.add(b12, BorderLayout.WEST);
    if(ridiReverse2.th5 >= ridiReverse2f.th5f)
    {
        jp1.add(rwpi2, BorderLayout.CENTER);
    }
    else
    {
        jp1.add(rwp2, BorderLayout.CENTER);
    }
    b_main.add(jp1);

    pan2 p2 = new pan2();
    panel_sp_main.add(b_main, BorderLayout.CENTER);
    panel_sp_main.add(p2, BorderLayout.EAST);
    panel2_main.setBackground(Color.blue);
    panel2_main.add(start_sp_main);
    panel2_main.add(back_sp_main);

    panel1_main.setBackground(Color.red);

    revChange.frame_rmain.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);

    revChange.frame_rmain.addWindowListener(new WindowAdapter()
    {
        public void windowClosing(WindowEvent e)
        {
            {
                int n = JOptionPane.showConfirmDialog(revChange.frame_rmain, "Are you sure you want to
                exit?", "RIDI_GUI", JOptionPane.YES_NO_OPTION);
                if(n == JOptionPane.YES_OPTION)
                {
                    System.exit(0);
                }
            }
        }
    });
    start_sp_main.addActionListener(new ButtonHandler());
    back_sp_main.addActionListener(new ButtonHandler());
    frame_rmain.setVisible(true);
}

class ButtonHandler implements ActionListener
{
    public void actionPerformed(ActionEvent ev1)
    {
        b1Thread bt1;
    }
}

```

```

b2Thread bt2;

String sps = ev1.getActionCommand();

if(sps == "Start")
{
    bt1 = new b1Thread();
    bt2 = new b2Thread();
    new Thread(rwsi1).start();
    new Thread(rshi1).start();
    new Thread(re11).start();
    new Thread(rwri1).start();
    new Thread(rwpi1).start();
    new Thread(rwsi2).start();
    new Thread(rshi2).start();
    new Thread(re12).start();
    new Thread(rwri2).start();
    new Thread(rwpi2).start();
    new Thread(rws1).start();
    new Thread(rsh1).start();
    new Thread(re11).start();
    new Thread(rwr1).start();
    new Thread(rwp1).start();
    new Thread(rws2).start();
    new Thread(rsh2).start();
    new Thread(re12).start();
    new Thread(rwr2).start();
    new Thread(rwp2).start();
}
if(sps == "Back")
{
    revInput renp2 = new revInput();
    revChange.frame_rmain.dispose();
}
}
}
}
}

```

```
//ridiReverse1.java
```

```
//This class holds the logic for conversion from coordinate points to theta angles in case of Reverse Kinematics.
```

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.lang.Math;
import java.lang.Integer;
```

```
public class ridiReverse1
```

```
{
    int i_rev;
    double T[][] = new double[4][4];
    double theta[] = new double[10];
    double rtd = 180./3.14159;
    double dtr = 3.14159/180.;
    double px = Integer.parseInt(revInput.str_x1_init);
    double py = Integer.parseInt(revInput.str_y1_init);
    double pz = Integer.parseInt(revInput.str_z1_init);
    double y1, y2, z2, temp1, temp2, T5;
    public static int t1, t2, t3, t4, t5;
```

```
    public ridiReverse1()
```

```
    {
        theta[4] = Integer.parseInt(revInput.str_t4_init);
        theta[5] = Integer.parseInt(revInput.str_t5_init);
```

```
        T5 = theta[5]*dtr;
```

```
        if(px>=0 && py>0)
```

```
        {
            theta[1] = Math.atan(px/py)*rtd;
            y1 = py/Math.cos(theta[1]*dtr);
        }
```

```
        if(px<=0 && py>0)
```

```
        {
            theta[1] = Math.atan(-px/py)*rtd;
            y1 = py/Math.cos(theta[1]*dtr);
        }
```

```
        if(px<=0 && py<0)
```

```
        {
            theta[1] = Math.atan(px/py)*rtd;
            if(theta[1] < 30)
            {
                //System.out.println("ERROR1");
            }
            y1 = py/Math.cos(theta[1]*dtr);
        }
```

```
        if(px>0 && py<0)
```

```

{
theta[1] = Math.atan(-px/py)*rtd;
if((90-theta[1]) > 60)
{
//System.out.println("ERROR2");
}
y1 = py/Math.cos(theta[1]*dtr);
}

if(px>=0 && py == 0)
theta[1] = 60;

if(px<0 && py==0)
{
theta[1] = 240;
y1 = - px;
}

if(pz<41.25)
{
z2 = -pz - 208.75 * Math.cos(T5) + 250.0;
y2 = y1 - 208.75 * Math.sin(T5);
temp1 = (220.0 * 220.0 + 160.0 * 160.0 - y2 * y2 - z2 * z2)/(2.0 * 220.0 * 160.0);
if((temp1*temp1) > 1.0)
{
//System.out.println("ERROR3");
}
else
{
temp2 = (220.0 * 220.0 + y2*y2 + z2*z2 - 160.0*160.0)/(2.0*220.0*Math.sqrt(y2*y2+z2*z2));
if((temp2*temp2)>1.0)
{
//System.out.println("ERROR4");
}
else
{
theta[1] = Math.atan2(Math.sqrt(1-temp1*temp1), temp1)*rtd;
if(theta[1]<90)
{
//System.out.println("ERROR5");
}
else
{
theta[2] = Math.atan2(z2,y2)*rtd;
theta[3] = Math.atan2(Math.sqrt(1-temp2*temp2),temp2)*rtd;
theta[5] = 180 - theta[1] - theta[3];
theta[4] = theta[5] + theta[2] + T5;
}
}
}
System.out.println(theta[1]+ " , " + theta[2] + " , " + theta[3] + " , " + theta[4] + " , " + theta[5]);
}
else
{
z2 = pz + 208.75 * Math.cos(T5) - 250.0;
y2 = y1 - 208.75 * Math.sin(T5);

```

```

temp1 = (220.0 * 220.0 + 160.0 * 160.0 - y2 * y2 - z2 * z2)/(2.0 * 220.0 * 160.0);
if((temp1*temp1) > 1.0)
{
//System.out.println("ERROR6");
}
else
{
temp2 = (220.0 * 220.0 + y2*y2 + z2*z2 - 160.0*160.0)/(2.0*220.0*Math.sqrt(y2*y2+z2*z2));
if((temp2*temp2)>1.0)
{
//System.out.println("ERROR7");
}
else
{
theta[1] = Math.atan2(Math.sqrt(1-temp1*temp1), temp1)*rtd;
if(theta[1]<90)
{
//System.out.println("ERROR8");
}
else
{
theta[2] = Math.atan2(z2,y2)*rtd;
theta[3] = Math.atan2(Math.sqrt(1-temp2*temp2),temp2)*rtd;
theta[4] = theta[5] + theta[2] + T5;
}
}
}
}
}
}
ridiReverse1.t1 = (int) theta[1];
ridiReverse1.t2 = (int) theta[2];
ridiReverse1.t3 = (int) theta[3];
ridiReverse1.t4 = (int) theta[4];
ridiReverse1.t5 = (int) theta[5];
}
}
}

```

```

//ridiReverse1f.java

//This class is responsible for conversion of points into theta angles for robot 1

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.lang.Math;
import java.lang.Integer;

public class ridiReverse1f
{
    int i_rev;
    double T[][] = new double[4][4];
    double theta[] = new double[10];
    double rtd = 180./3.14159;
    double dtr = 3.14159/180.;
    double px = Integer.parseInt(revInput.str_x1_fin);
    double py = Integer.parseInt(revInput.str_y1_fin);
    double pz = Integer.parseInt(revInput.str_z1_fin);
    double y1, y2, z2, temp1, temp2, T5;
    public static int t1f, t2f, t3f, t4f, t5f;

    public ridiReverse1f()
    {
        theta[4] = Integer.parseInt(revInput.str_t4_fin);
        theta[5] = Integer.parseInt(revInput.str_t5_fin);

        T5 = theta[5]*dtr;

        if(px>=0 && py>0)
        {
            theta[1] = Math.atan(px/py)*rtd;
            y1 = py/Math.cos(theta[1]*dtr);
        }

        if(px<=0 && py>0)
        {
            theta[1] = Math.atan(-px/py)*rtd;
            y1 = py/Math.cos(theta[1]*dtr);
        }

        if(px<=0 && py<0)
        {
            theta[1] = Math.atan(px/py)*rtd;
            if(theta[1] < 30)
            {
                //System.out.println("ERROR1");
            }
            y1 = py/Math.cos(theta[1]*dtr);
        }

        if(px>0 && py<0)
        {
            theta[1] = Math.atan(-px/py)*rtd;

```

```

if((90-theta[1]) > 60)
{
//System.out.println("ERROR2");
}
y1 = py/Math.cos(theta[1]*dtr);
}

if(px>=0 && py == 0)
theta[1] = 60;

if(px<0 && py==0)
{
theta[1] = 240;
y1 = - px;
}

if(pz<41.25)
{
z2 = -pz - 208.75 * Math.cos(T5) + 250.0;
y2 = y1 - 208.75 * Math.sin(T5);
temp1 = (220.0 * 220.0 + 160.0 * 160.0 - y2 * y2 - z2 * z2)/(2.0 * 220.0 * 160.0);
if((temp1*temp1) > 1.0)
{
//System.out.println("ERROR3");
}
else
{
temp2 = (220.0 * 220.0 + y2*y2 + z2*z2 - 160.0*160.0)/(2.0*220.0*Math.sqrt(y2*y2+z2*z2));
if((temp2*temp2)>1.0)
{
//System.out.println("ERROR4");
}
else
{
theta[1] = Math.atan2(Math.sqrt(1-temp1*temp1), temp1)*rtd;
if(theta[1]<90)
{
//System.out.println("ERROR5");
}
else
{
theta[2] = Math.atan2(z2,y2)*rtd;
theta[3] = Math.atan2(Math.sqrt(1-temp2*temp2),temp2)*rtd;
theta[5] = 180 - theta[1] - theta[3];
theta[4] = theta[5] + theta[2] + T5;
}
}
}
System.out.println(theta[1]+ " , " + theta[2] + " , " + theta[3] + " , " + theta[4] + " , " + theta[5]);
}
else
{
z2 = pz + 208.75 * Math.cos(T5) - 250.0;
y2 = y1 - 208.75 * Math.sin(T5);
temp1 = (220.0 * 220.0 + 160.0 * 160.0 - y2 * y2 - z2 * z2)/(2.0 * 220.0 * 160.0);
if((temp1*temp1) > 1.0)

```



```

{
//System.out.println("ERROR6");
}
else
{
temp2 = (220.0 * 220.0 + y2*y2 + z2*z2 - 160.0*160.0)/(2.0*220.0*Math.sqrt(y2*y2+z2*z2));
if((temp2*temp2)>1.0)
{
//System.out.println("ERROR7");
}
else
{
theta[1] = Math.atan2(Math.sqrt(1-temp1*temp1), temp1)*rtd;
if(theta[1]<90)
{
//System.out.println("ERROR8");
}
else
{
theta[2] = Math.atan2(z2,y2)*rtd;
theta[3] = Math.atan2(Math.sqrt(1-temp2*temp2),temp2)*rtd;
theta[4] = theta[5] + theta[2] + T5;
}
}
}
}
}
ridiReverse1f.t1f = (int) theta[1];
ridiReverse1f.t2f = (int) theta[2];
ridiReverse1f.t3f = (int) theta[3];
ridiReverse1f.t4f = (int) theta[4];
ridiReverse1f.t5f = (int) theta[5];
}
}

```

```

//ridiReverse2.java

//Conversion for robot 2

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.lang.Math;
import java.lang.Integer;

public class ridiReverse2
{
    int i_rev;
    double T[][] = new double[4][4];
    double theta[] = new double[10];
    double rtd = 180./3.14159;
    double dtr = 3.14159/180.;
    double px = Integer.parseInt(revInput.str_x2_init);
    double py = Integer.parseInt(revInput.str_y2_init);
    double pz = Integer.parseInt(revInput.str_z2_init);
    double y1, y2, z2, temp1, temp2, T5;
    public static int th1, th2, th3, th4, th5;

    public ridiReverse2()
    {
        theta[4] = Integer.parseInt(revInput.str_t42_init);
        theta[5] = Integer.parseInt(revInput.str_t52_init);

        T5 = theta[5]*dtr;

        if(px>=0 && py>0)
        {
            theta[1] = Math.atan(px/py)*rtd;
            y1 = py/Math.cos(theta[1]*dtr);
        }

        if(px<=0 && py>0)
        {
            theta[1] = Math.atan(-px/py)*rtd;
            y1 = py/Math.cos(theta[1]*dtr);
        }

        if(px<=0 && py<0)
        {
            theta[1] = Math.atan(px/py)*rtd;
            if(theta[1] < 30)
            {
                //System.out.println("ERROR1");
            }
            y1 = py/Math.cos(theta[1]*dtr);
        }

        if(px>0 && py<0)
        {
            theta[1] = Math.atan(-px/py)*rtd;
        }
    }
}

```

```

if((90-theta[1]) > 60)
{
//System.out.println("ERROR2");
}
y1 = py/Math.cos(theta[1]*dtr);
}

if(px>=0 && py == 0)
theta[1] = 60;

if(px<0 && py==0)
{
theta[1] = 240;
y1 = - px;
}

if(pz<41.25)
{
z2 = -pz - 208.75 * Math.cos(T5) + 250.0;
y2 = y1 - 208.75 * Math.sin(T5);
temp1 = (220.0 * 220.0 + 160.0 * 160.0 - y2 * y2 - z2 * z2)/(2.0 * 220.0 * 160.0);
if((temp1*temp1) > 1.0)
{
//System.out.println("ERROR3");
}
else
{
temp2 = (220.0 * 220.0 + y2*y2 + z2*z2 - 160.0*160.0)/(2.0*220.0*Math.sqrt(y2*y2+z2*z2));
if((temp2*temp2)>1.0)
{
//System.out.println("ERROR4");
}
else
{
theta[1] = Math.atan2(Math.sqrt(1-temp1*temp1), temp1)*rtd;
if(theta[1]<90)
{
//System.out.println("ERROR5");
}
else
{
theta[2] = Math.atan2(z2,y2)*rtd;
theta[3] = Math.atan2(Math.sqrt(1-temp2*temp2),temp2)*rtd;
theta[5] = 180 - theta[1] - theta[3];
theta[4] = theta[5] + theta[2] + T5;
}
}
}
System.out.println(theta[1]+ " , " + theta[2] + " , " + theta[3] + " , " + theta[4] + " , " + theta[5]);
}
else
{
z2 = pz + 208.75 * Math.cos(T5) - 250.0;
y2 = y1 - 208.75 * Math.sin(T5);
temp1 = (220.0 * 220.0 + 160.0 * 160.0 - y2 * y2 - z2 * z2)/(2.0 * 220.0 * 160.0);
if((temp1*temp1) > 1.0)

```

```

{
//System.out.println("ERROR6");
}
else
{
temp2 = (220.0 * 220.0 + y2*y2 + z2*z2 - 160.0*160.0)/(2.0*220.0*Math.sqrt(y2*y2+z2*z2));
if((temp2*temp2)>1.0)
{
//System.out.println("ERROR7");
}
else
{
theta[1] = Math.atan2(Math.sqrt(1-temp1*temp1), temp1)*rtd;
if(theta[1]<90)
{
//System.out.println("ERROR8");
}
else
{
theta[2] = Math.atan2(z2,y2)*rtd;
theta[3] = Math.atan2(Math.sqrt(1-temp2*temp2),temp2)*rtd;
theta[4] = theta[5] + theta[2] + T5;
}
}
}
}
}
ridiReverse2.th1 = (int) theta[1];
ridiReverse2.th2 = (int) theta[2];
ridiReverse2.th3 = (int) theta[3];
ridiReverse2.th4 = (int) theta[4];
ridiReverse2.th5 = (int) theta[5];
}
}

```

```

//ridiReverse2f.java

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.lang.Math;
import java.lang.Integer;

public class ridiReverse2
{
    int i_rev;
    double T[][] = new double[4][4];
    double theta[] = new double[10];
    double rtd = 180./3.14159;
    double dtr = 3.14159/180.;
    double px = Integer.parseInt(revInput.str_x2_init);
    double py = Integer.parseInt(revInput.str_y2_init);
    double pz = Integer.parseInt(revInput.str_z2_init);
    double y1, y2, z2, temp1, temp2, T5;
    public static int th1, th2, th3, th4, th5;

    public ridiReverse2()
    {
        theta[4] = Integer.parseInt(revInput.str_t42_init);
        theta[5] = Integer.parseInt(revInput.str_t52_init);

        T5 = theta[5]*dtr;

        if(px>=0 && py>0)
        {
            theta[1] = Math.atan(px/py)*rtd;
            y1 = py/Math.cos(theta[1]*dtr);
        }

        if(px<=0 && py>0)
        {
            theta[1] = Math.atan(-px/py)*rtd;
            y1 = py/Math.cos(theta[1]*dtr);
        }

        if(px<=0 && py<0)
        {
            theta[1] = Math.atan(px/py)*rtd;
            if(theta[1] < 30)
            {
                //System.out.println("ERROR1");
            }
            y1 = py/Math.cos(theta[1]*dtr);
        }

        if(px>0 && py<0)
        {
            theta[1] = Math.atan(-px/py)*rtd;
            if((90-theta[1]) > 60)
            {

```

```

//System.out.println("ERROR2");
}
y1 = py/Math.cos(theta[1]*dtr);
}

if(px>=0 && py == 0)
theta[1] = 60;

if(px<0 && py==0)
{
theta[1] = 240;
y1 = - px;
}

if(pz<41.25)
{
z2 = -pz - 208.75 * Math.cos(T5) + 250.0;
y2 = y1 - 208.75 * Math.sin(T5);
temp1 = (220.0 * 220.0 + 160.0 * 160.0 - y2 * y2 - z2 * z2)/(2.0 * 220.0 * 160.0);
if((temp1*temp1) > 1.0)
{
//System.out.println("ERROR3");
}
else
{
temp2 = (220.0 * 220.0 + y2*y2 + z2*z2 - 160.0*160.0)/(2.0*220.0*Math.sqrt(y2*y2+z2*z2));
if((temp2*temp2)>1.0)
{
//System.out.println("ERROR4");
}
else
{
theta[1] = Math.atan2(Math.sqrt(1-temp1*temp1), temp1)*rtd;
if(theta[1]<90)
{
//System.out.println("ERROR5");
}
else
{
theta[2] = Math.atan2(z2,y2)*rtd;
theta[3] = Math.atan2(Math.sqrt(1-temp2*temp2),temp2)*rtd;
theta[5] = 180 - theta[1] - theta[3];
theta[4] = theta[5] + theta[2] + T5;
}
}
}
System.out.println(theta[1]+ " , " + theta[2] + " , " + theta[3] + " , " + theta[4] + " , " + theta[5]);
}
else
{
z2 = pz + 208.75 * Math.cos(T5) - 250.0;
y2 = y1 - 208.75 * Math.sin(T5);
temp1 = (220.0 * 220.0 + 160.0 * 160.0 - y2 * y2 - z2 * z2)/(2.0 * 220.0 * 160.0);
if((temp1*temp1) > 1.0)
{
//System.out.println("ERROR6");
}
}
}

```

```

}
else
{
temp2 = (220.0 * 220.0 + y2*y2 + z2*z2 - 160.0*160.0)/(2.0*220.0*Math.sqrt(y2*y2+z2*z2));
if((temp2*temp2)>1.0)
{
//System.out.println("ERROR7");
}
else
{
theta[1] = Math.atan2(Math.sqrt(1-temp1*temp1), temp1)*rtd;
if(theta[1]<90)
{
//System.out.println("ERROR8");
}
else
{
theta[2] = Math.atan2(z2,y2)*rtd;
theta[3] = Math.atan2(Math.sqrt(1-temp2*temp2),temp2)*rtd;
theta[4] = theta[5] + theta[2] + T5;
}
}
}
}
}
ridiReverse2.th1 = (int) theta[1];
ridiReverse2.th2 = (int) theta[2];
ridiReverse2.th3 = (int) theta[3];
ridiReverse2.th4 = (int) theta[4];
ridiReverse2.th5 = (int) theta[5];
}
}

```

```

//ridiForward_init.java

//Algorithm for converting the theta values to coordinate points.

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.lang.Math;
import java.lang.Math;

public class ridiForward_init
{
    int i_for, dof = 5;
    double A[][] = new double[4][4];
    double B[][] = new double[4][4];
    double T[][] = new double[4][4];
    double C[][] = new double[4][4];
    double dtr = 3.14159/180.;
    double d[] = {0.,250.,0.,0.,160.,0.,0.,0.,0.,0.};
    double a[] = {0.,0.,0.,220.,0.,0.,0.,0.,0.,0.};
    double alpha[] = {0.,0.,-90.*dtr,0.,-90.*dtr,-90.*dtr,0.,0.,0.,0.};
    double theta[] = {0,-
panel_ws1_internal.intg_wst1_init,panel_sh1_internal.intg_sho1_init,panel_el1_internal.intg_elb1_init,pan
el_wr1_internal.intg_wrr1_init,panel_wp1_internal.intg_wrp1_init};
    public ridiForward_init()
    {
        idtA();
        idtB();
        idtT();
        System.out.println(T[0][3] + "," + T[1][3] + "," + T[2][3] + "," + T[3][3]);
    }
    public void idtA()
    {
        int i_idt, j_idt;
        for(i_idt = 0; i_idt<=3;i_idt++)
        {
            for(j_idt = 0;j_idt <=3;j_idt++)
            {
                if(i_idt == j_idt)
                    A[i_idt][j_idt] = 1.0;
                else
                    A[i_idt][j_idt] = 0.0;
            }
        }
    }

    public void idtB()
    {
        int i_idt, j_idt;
        for(i_idt = 0; i_idt<=3;i_idt++)
        {
            for(j_idt = 0;j_idt <=3;j_idt++)
            {
                if(i_idt == j_idt)
                    B[i_idt][j_idt] = 1.0;
                else
            }
        }
    }
}

```



```

    B[i_idt][j_idt] = 0.0;
    }
    }
}
public void idtT()
{
    for(i_for = 1;i_for <= dof;i_for++)
        theta[i_for] = theta[i_for] * dtr;

    for(i_for = 1;i_for <= dof; i_for++)
    {
        A[0][0] = Math.cos(theta[i_for]);
        A[1][0] = Math.sin(theta[i_for]);
        A[2][0] = 0.0;
        A[3][0] = 0.0;

        A[0][1] = -Math.sin(theta[i_for])*Math.cos(alpha[i_for]);
        A[1][1] = Math.cos(theta[i_for])*Math.cos(alpha[i_for]);
        A[2][1] = Math.sin(alpha[i_for]);
        A[3][1] = 0.0;

        A[0][2] = Math.sin(theta[i_for])*Math.sin(alpha[i_for]);
        A[1][2] = -Math.cos(theta[i_for])*Math.sin(alpha[i_for]);
        A[2][2] = Math.cos(alpha[i_for]);
        A[3][2] = 0.0;

        A[0][3] = Math.cos(theta[i_for])*a[i_for];
        A[1][3] = Math.sin(theta[i_for])*a[i_for];
        A[2][3] = d[i_for];
        A[3][3] = 1.0;

        cal_c();

        for(int i_c = 0;i_c<=3;i_c++)
        {
            for(int j_c = 0;j_c<=3;j_c++)
            {
                T[i_c][j_c] = C[i_c][j_c];
            }
        }
        copy_t();
    }
}
public void cal_c()
{
    for(int i_t = 0;i_t<=3;i_t++)
    {
        for(int j_t = 0;j_t<=3;j_t++)
        {
            C[i_t][j_t] = 0.0;
            for(int k_t = 0;k_t<=3;k_t++)
                C[i_t][j_t] = C[i_t][j_t] + A[i_t][k_t] * B[k_t][j_t];
        }
    }
}
public void copy_t()

```

```
{
int i , j;
for(i = 0; i<=3; i++)
{
for(j= 0;j<=3; j++)
B[i][j] = T[i][j];
}
}
}
```

```

/*Sample Programs for the functioning of the Waist Joint of the robot in:
Reverse and Forward mode*/
//Program for outer panel of the waist joint band in forward kinematic mode.

```

```

import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.BorderFactory;
import java.awt.Graphics;
import java.awt.*;
import java.awt.event.*;
import java.lang.Integer;
import java.util.*;

public class panel_ws1 extends JPanel implements Runnable
{
    double theta;
    int v = 0;
    int u = 0;
    int x = 0;
    public static int intg_wst1_init;
    public static int intg_wst1_final;
    String st;
    double init ;
    double fin ;

    public panel_ws1()
    {
        setPreferredSize(new Dimension(560,70));
        try
        {
            panel_ws1.intg_wst1_init = Integer.parseInt(ridiInput1.str_wst1_init);
        }
        catch(NumberFormatException e)
        {
            panel_ws1.intg_wst1_init = 0;
        }

        try
        {
            panel_ws1.intg_wst1_final = Integer.parseInt(ridiInput1.str_wst1_final);
        }
        catch(NumberFormatException e)
        {
            panel_ws1.intg_wst1_final = 0;
        }
        init = (double) panel_ws1.intg_wst1_init;
        fin = (double) panel_ws1.intg_wst1_final;
        st = String.valueOf(init);
    }
    public void run()
    {
        double t;
        for( theta = init; theta>=fin; theta--)
        {
            try
            {

```

```

if(init>180)
{
if(theta == 180)
{
x += 70;
}
}
if(init<=180)
t = 180 - theta;
else
t = 360 - theta;
st = String.valueOf(theta);
u = (int) (35 * Math.sin(Math.toRadians(360-theta)));
repaint();
Thread.sleep(1500);
v = (int) (Math rint(t * 7/18));
panel_ws1_internal.get.setValue(theta,1,fin);
}
catch(InterruptedException ev){}
}
}
public void paint(Graphics g)
{
Graphics2D g2 = (Graphics2D) g;
BasicStroke stroke = new BasicStroke(2.5f);
g2.setColor(Color.black);
g2.drawLine(0,0,740,0);
g2.drawLine(0,35,300,35);
if(init>180)
{
g2.drawString("360",0,45);
g2.drawString("270",35,45);
g2.drawString("180",70,45);
g2.drawString("90",105,45);
g2.drawString("0",140,45);
}
if(init<=180)
{
g2.drawString("180",0,45);
g2.drawString("90",35,45);
g2.drawString("0",70,45);
}
g2.drawLine(0,70,740,70);
g2.setColor(Color.white);
g2.setStroke(stroke);
g2.drawLine(35+x,35,v,35+u);

g2.setColor(Color.black);
g2.drawLine(300,0,300,70);
g2.drawLine(450,0,450,70);

Font f = new Font("Serif", Font.BOLD, 18);
g2.setFont(f);
g2.drawString("Joint - Waist", 310,20);
g2.drawString("Robot 1", 310, 40);
g2.setColor(Color.blue);

```

```

g2.drawString("Anti-Clockwise", 310,60);

g2.setColor(Color.black);
g2.drawString("Theta = "+st, 460,40);
g2.drawString("Range: 0 - 300", 460,20);
}
}

```

//Program for the internal panel for waist joint of robot 1 in Forward kinematics mode.

```

import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.BorderFactory;
import java.awt.Graphics;
import java.awt.*;
import java.awt.event.*;
import java.lang.Integer;
import java.util.*;

public class panel_ws1_internal extends JPanel implements Runnable
{
    double theta;
    int v = 0;
    int u = 0;
    int x = 0;
    public static int intg_wst1_init;
    public static int intg_wst1_final;
    public static get_theta get = new get_theta();
    String st;
    double init ;
    double fin ;

    public panel_ws1_internal()
    {
        setPreferredSize(new Dimension(560,70));
        try
        {
            panel_ws1_internal.intg_wst1_init = Integer.parseInt(ridiInput1.str_wst1_init);
        }
        catch(NumberFormatException e)
        {
            panel_ws1_internal.intg_wst1_init = 0;
        }

        try
        {
            panel_ws1_internal.intg_wst1_final = Integer.parseInt(ridiInput1.str_wst1_final);
        }
        catch(NumberFormatException e)
        {
            panel_ws1_internal.intg_wst1_final = 0;
        }
    }
}

```

```

init = (double) panel_ws1_internal.intg_wst1_init;
fin = (double) panel_ws1_internal.intg_wst1_final;
st = String.valueOf(init);
}
public void run()
{
double t;
for(theta = init; theta<=fin; theta++)
{
try
{
st = String.valueOf(theta);
if(init<180)
{
if(theta == 180)
{
x += 70;
}
}
if(init>=180)
t = theta-180;
else
t = theta;
u = (int) (35 * Math.sin(Math.toRadians(t)));
repaint();
Thread.sleep(1500);
v = (int) (Math rint(t * 7/18));
get.setValue(theta, 1, fin);
}
catch(InterruptedException ev){}
}
}
public void paint(Graphics g)
{
Graphics2D g2 = (Graphics2D) g;
BasicStroke stroke = new BasicStroke(2.5f);
g2.setColor(Color.black);
g2.drawLine(0,0,740,0);
g2.drawLine(0,70,740,70);
if(init<180)
{
g2.drawString("0",0,45);
g2.drawString("90",33,45);
g2.drawString("180",67,45);
g2.drawString("270",102,45);
}
if(init>=180)
{
g2.drawString("180",0,45);
g2.drawString("270",33,45);
}
g2.drawLine(0,35,300,35);
g2.setColor(Color.white);
g2.setStroke(stroke);
g2.drawLine(35+x,35,v,35-u);
g2.setColor(Color.black);

```

```

g2.drawLine(300,0,300,70);
g2.drawLine(450,0,450,70);

Font f = new Font("Serif", Font.BOLD, 18);
g2.setFont(f);
g2.drawString("Joint - Waist", 310,20);
g2.drawString("Robot 1", 310, 40);
g2.setColor(Color.blue);
g2.drawString("Clockwise", 310,60);

g2.setColor(Color.black);
g2.drawString("Theta = "+st, 460,40);
g2.drawString("Range: 0 - 300", 460,20);
}
}

```

//Program for outer panel of waist joint of robot 1 in Reverse mode.

```

import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.BorderFactory;
import java.awt.Graphics;
import java.awt.*;
import java.awt.event.*;
import java.lang.Integer;
import java.util.*;

public class rev_ws1 extends JPanel implements Runnable
{
    double theta;
    int v = 0;
    int u = 0;
    int x = 0;
    public static int intg_wst1_in;
    public static int intg_wst1_fin;
    String st;
    double init ;
    double fin ;

    public rev_ws1()
    {
        setPreferredSize(new Dimension(560,70));
        rev_ws1.intg_wst1_in = ridiReverse1.t1;
        rev_ws1.intg_wst1_fin = ridiReverse1f.t1f;
        init = (double) rev_ws1.intg_wst1_in;
        fin = (double) rev_ws1.intg_wst1_fin;
        st = String.valueOf(init);
    }
    public void run()
    {
        double t;
        for( theta = init; theta>=fin; theta--)
        {
            try

```

```

{
if(init>180)
{
if(theta == 180)
{
x += 70;
}
}
if(init<=180)
t = 180 - theta;
else
t = 360 - theta;
st = String.valueOf(theta);
u = (int) (35 * Math.sin(Math.toRadians(360-theta)));
repaint();
Thread.sleep(1500);
v = (int) (Math rint(t * 7/18));
rev_ws1_internal.get.setValue(theta,1,fin);
}
catch(InterruptedException ev){}
}
}
public void paint(Graphics g)
{
Graphics2D g2 = (Graphics2D) g;
BasicStroke stroke = new BasicStroke(2.5f);
g2.setColor(Color.black);
g2.drawLine(0,0,740,0);
g2.drawLine(0,35,300,35);
if(init>180)
{
g2.drawString("360",0,45);
g2.drawString("270",35,45);
g2.drawString("180",70,45);
g2.drawString("90",105,45);
g2.drawString("0",140,45);
}
if(init<=180)
{
g2.drawString("180",0,45);
g2.drawString("90",35,45);
g2.drawString("0",70,45);
}
g2.drawLine(0,70,740,70);
g2.setColor(Color.white);
g2.setStroke(stroke);
g2.drawLine(35+x,35,v,35+u);

g2.setColor(Color.black);
g2.drawLine(300,0,300,70);
g2.drawLine(450,0,450,70);

Font f = new Font("Serif", Font.BOLD, 18);
g2.setFont(f);
g2.drawString("Joint - Waist", 310,20);
g2.drawString("Robot 1", 310, 40);

```



```

g2.setColor(Color.blue);
g2.drawString("Anti-Clockwise", 310,60);

g2.setColor(Color.black);
g2.drawString("Theta = "+st, 460,40);
g2.drawString("Range: 0 - 300", 460,20);
}
}

```

//Program for the internal panel of waist joint of robot 1 in Reverse mode.

```

import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.BorderFactory;
import java.awt.Graphics;
import java.awt.*;
import java.awt.event.*;
import java.lang.Integer;
import java.util.*;

public class rev_ws1_internal extends JPanel implements Runnable
{
    double theta;
    int v = 0;
    int u = 0;
    int x = 0;
    public static int intg_wst1_in;
    public static int intg_wst1_fin;
    public static get_theta get = new get_theta();
    String st;
    double init ;
    double fin ;

    public rev_ws1_internal()
    {
        setPreferredSize(new Dimension(560,70));
        rev_ws1_internal.intg_wst1_in = ridiReverse1.t1;
        rev_ws1_internal.intg_wst1_fin = ridiReverse1f.t1f;
        init = (double) rev_ws1_internal.intg_wst1_in;
        fin = (double) rev_ws1_internal.intg_wst1_fin;
        st = String.valueOf(init);
    }
    public void run()
    {
        double t;
        for(theta = init; theta<=fin; theta++)
        {
            try
            {
                st = String.valueOf(theta);
                if(init<180)
                {
                    if(theta == 180)
                    {
                        x += 70;
                    }
                }
            }
        }
    }
}

```

```

    }
    }
    if(init>=180)
    t = theta-180;
    else
    t = theta;
    u = (int) (35 * Math.sin(Math.toRadians(t)));
    repaint();
    Thread.sleep(1500);
    v = (int) (Math rint(t * 7/18));
    get.setValue(theta, 1, fin);
    }
    catch(InterruptedException ev){ }
    }
}
public void paint(Graphics g)
{
    Graphics2D g2 = (Graphics2D) g;
    BasicStroke stroke = new BasicStroke(2.5f);
    g2.setColor(Color.black);
    g2.drawLine(0,0,740,0);
    g2.drawLine(0,70,740,70);
    if(init<180)
    {
        g2.drawString("0",0,45);
        g2.drawString("90",33,45);
        g2.drawString("180",67,45);
        g2.drawString("270",102,45);
    }
    if(init>=180)
    {
        g2.drawString("180",0,45);
        g2.drawString("270",33,45);
    }
    g2.drawLine(0,35,300,35);
    g2.setColor(Color.white);
    g2.setStroke(stroke);
    g2.drawLine(35+x,35,v,35-u);
    g2.setColor(Color.black);
    g2.drawLine(300,0,300,70);
    g2.drawLine(450,0,450,70);

    Font f = new Font("Serif", Font.BOLD, 18);
    g2.setFont(f);
    g2.drawString("Joint - Waist", 310,20);
    g2.drawString("Robot 1", 310, 40);
    g2.setColor(Color.blue);
    g2.drawString("Clockwise", 310,60);

    g2.setColor(Color.black);
    g2.drawString("Theta = "+st, 460,40);
    g2.drawString("Range: 0 - 300", 460,20);
}
}

```

//Sample Programs for panel showing robot 1 gripper motions.

```
import javax.swing.*;
import java.awt.*;
import java.awt.Graphics;
import java.awt.event.*;
import javax.swing.BorderFactory;
import javax.swing.border.Border;

public class panel_gr1 extends JPanel
{
    public static JButton bc1 = new JButton();
    public static JButton bc2 = new JButton();

    //Constructor of the panel...
    public panel_gr1()
    {
        setPreferredSize(new Dimension(560,70));
        setLayout(null);
        setBackground(Color.white);
        setBorder(BorderFactory.createLineBorder(Color.black));
        bc1.setBackground(Color.blue);
        bc2.setBackground(Color.blue);
        if(ridiInput1.str_grp1_init == "Open")
        {
            bc1.setBounds(0,0,20,10);
            bc2.setBounds(0,60,20,10);
        }
        else
        {
            bc1.setBounds(0,25,20,10);
            bc2.setBounds(0,35,20,10);
        }
        add(bc1);
        add(bc2);
    }
}
```

```
//Sample programs for the threads on which the gripper operates of robot 1.  
//b1Thread.java
```

```
import javax.swing.*;  
import java.awt.event.*;  
import java.awt.*;  
  
public class b1Thread implements Runnable  
{  
    public static Thread t1;  
    int k = 0;  
    public static int i, j;  
  
    public b1Thread()  
    {  
        t1 = new Thread(this,"Gripper-1");  
        t1.start();  
    }  
    public void run()  
    {  
        try  
        {  
            if(ridiInput1.str_grp1_init == ridiInput1.str_grp1_dir)  
            {  
            }  
            else  
            {  
                if(ridiInput1.str_grp1_init == "Close")  
                {  
                    i = 25;  
                    j = 35;  
                    while(i>=0)  
                    {  
                        panel_gr1.bc1.setBounds(k,i,20,10);  
                        panel_gr1.bc2.setBounds(k,j,20,10);  
                        Thread.sleep(1500);  
                        j+=1;  
                        i-=1;  
                        k+=2;  
                    }  
                }  
            }  
            else  
            {  
                int i = 0;  
                int j = 60;  
                while(i<=25)  
                {  
                    panel_gr1.bc1.setBounds(k,i,20,10);  
                    panel_gr1.bc2.setBounds(k,j,20,10);  
                    Thread.sleep(1500);  
                    j-=1;  
                    i+=1;  
                    k+=2;  
                }  
            }  
        }  
    }  
}
```

```

    }
    catch(InterruptedException ei)
    {
    }
}
}

```

/*Sample Program for robot 1 for obtaining the theta values from each individual joint and passing on for calculation of the the final coordinates when all the 5 values have been extracted */

```

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class get_theta
{
    int j = 4;
    txt_r1 tr1 = new txt_r1();
    public double theta[] = new double[6];
    public int i=0;
    public synchronized void setValue(double value, int index, double f)
    {
        theta[index] = value;
        if(value == f)
        j = j-1;
        i = i + 1;
        while(i<=j)
        {
            try
            {
                wait();
                return;
            }
            catch(InterruptedException e)
            {
            }
        }
        tr1.draw_theta(theta);
        i = 0;
        notifyAll();
    }
}

```

//Program for updating the values of each joint on the panel and calculating each individual theta.

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.lang.Math;

public class txt_r1 extends JTextArea //implements Scrollable
{
    int i_for , dof = 5;
    double A[][] = new double[4][4];
    double B[][] = new double[4][4];
    double T[][] = new double[4][4];
    double C[][] = new double[4][4];
    double dtr = 3.14159/180.;
    double d[] = {0.,250.,0.,0.,160.,0.,0.,0.,0.,0.};
    double a[] = {0.,0.,0.,220.,0.,0.,0.,0.,0.,0.};
    double alpha[] = {0.,0.,-90.*dtr,0.,-90.*dtr,-90.*dtr,0.,0.,0.,0.};
    double theta[] = new double[6];
    //Rectangle rect = new Rectangle(0,0,175,395);

    String s1, s2, s3;
    public txt_r1()
    {
        setPreferredSize(new Dimension(175,3600));
    }

    public synchronized void draw_theta(double t[])
    {
        int fla = 0;
        for (int m = 1; m<=5; m++)
        {
            theta[m] = t[m];
        }
        idtA();
        idtB();
        idtT();
        s1 = String.valueOf(T[0][3]).substring(0,6);
        s2 = String.valueOf(T[1][3]).substring(0,6);
        s3 = String.valueOf(T[2][3]).substring(0,6);
        fla = pan2.str(s1+" " + s2 + " " + s3);
        if (fla == 0)
        {
            try
            {
                wait();
                return;
            }
            catch(InterruptedException e)
            {}
        }
        else
        {
            fla = 0;
            notifyAll();
        }
    }
}
```

```

}

public void idtA()
{
int i_idt, j_idt;
for(i_idt = 0; i_idt<=3;i_idt++)
{
for(j_idt = 0;j_idt <=3;j_idt++)
{
if(i_idt == j_idt)
A[i_idt][j_idt] = 1.0;
else
A[i_idt][j_idt] = 0.0;
}
}
}

public void idtB()
{
int i_idt, j_idt;
for(i_idt = 0; i_idt<=3;i_idt++)
{
for(j_idt = 0;j_idt <=3;j_idt++)
{
if(i_idt == j_idt)
B[i_idt][j_idt] = 1.0;
else
B[i_idt][j_idt] = 0.0;
}
}
}

public void idtT()
{
for(i_for = 1;i_for <= dof;i_for++)
theta[i_for] = theta[i_for] * dtr;

for(i_for = 1;i_for <= dof; i_for++)
{
A[0][0] = Math.cos(theta[i_for]);
A[1][0] = Math.sin(theta[i_for]);
A[2][0] = 0.0;
A[3][0] = 0.0;

A[0][1] = -Math.sin(theta[i_for])*Math.cos(alpha[i_for]);
A[1][1] = Math.cos(theta[i_for])*Math.cos(alpha[i_for]);
A[2][1] = Math.sin(alpha[i_for]);
A[3][1] = 0.0;

A[0][2] = Math.sin(theta[i_for])*Math.sin(alpha[i_for]);
A[1][2] = -Math.cos(theta[i_for])*Math.sin(alpha[i_for]);
A[2][2] = Math.cos(alpha[i_for]);
A[3][2] = 0.0;

A[0][3] = Math.cos(theta[i_for])*a[i_for];
A[1][3] = Math.sin(theta[i_for])*a[i_for];
A[2][3] = d[i_for];
}
}

```

```

A[3][3] = 1.0;

cal_c();

for(int i_c = 0;i_c<=3;i_c++)
{
for(int j_c = 0;j_c<=3;j_c++)
{
T[i_c][j_c] = C[i_c][j_c];
}
}
copy_t();
}
public void cal_c()
{
for(int i_t = 0;i_t<=3;i_t++)
{
for(int j_t = 0;j_t<=3;j_t++)
{
C[i_t][j_t] = 0.0;
for(int k_t = 0;k_t<=3;k_t++)
C[i_t][j_t] = C[i_t][j_t] + A[i_t][k_t] * B[k_t][j_t];
}
}
}
public void copy_t()
{
int i , j;
for(i = 0; i<=3; i++)
{
for(j= 0;j<=3; j++)
B[i][j] = T[i][j];
}
}
}

```