Research on Remote Control of Reconfigurable Modular Robotic System

by

Zhanglei Song

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Applied Science

in

The Faculty of Engineering and Applied Science

Mechanical Engineering Program

University of Ontario Institute of Technology

August, 2009

CERTIFICATE OF APPROVAL

Submitted by: _____Zhanglei Song_____ Student #: __100350351____
First Name, Last Name

In partial fulfillment of the requirements for the degree of:

_____ Master of Applied Science _____ in ____Mechanical Engineering_____
Degree Name in full (e.g. Master of Applied Science) Name of Program

Date of Defense (if applicable): _____

Thesis Title:

Research on Remote Control of Reconfigurable Modular Robotic System.

The undersigned certify that they recommend this thesis to the Office of Graduate Studies for acceptance:

_____	_____	_____
Chair of Examining Committee	Signature	Date (yyyy/mm/dd)
_____	_____	_____
External Examiner	Signature	Date (yyyy/mm/dd)
_____	_____	_____
Member of Examining Committee	Signature	Date (yyyy/mm/dd)
_____	_____	_____
Member of Examining Committee	Signature	Date (yyyy/mm/dd)

As research supervisor for the above student, I certify that I have read the following defended thesis, have approved changes required by the final examiners, and recommend it to the Office of Graduate Studies for acceptance:

_____	_____	_____
Name of Research Supervisor	Signature of Research Supervisor	Date (yyyy/mm/dd)

Abstract

Serial manipulators, which have large work space with respect to their own volume and occupied floor space, are the most common industrial robots by far. However, in many environments the situation is unstructured and less predictable, such as aboard a space station, a nuclear waste retrieval site, or a lunar base construction site. It is almost impossible to design a single robotic system which can meet all the requirements for every task. In these circumstances, it is important to deploy a modular reconfigurable robotic system, which is suitable to various task requirements. Modular reconfigurable robots have a variety of attributes that are well suited to for these conditions, including: the ability to serve as many different tools at once (saving weight), packing into compressed forms (saving space) and having high levels of redundancy (increasing robustness). By easy disassembly and reassembly features, this serial modular robotic system will bring advantages to small and medium enterprise to save costs in the long term.

This thesis focuses on developing such a serial reconfigurable modular robotic system with remote control functionality. The robotic arms are assembled by PowerCube Modules with cubic outward appearance. The control and power electronics are fully integrated on the connector block inside of the modules. Those modules are connected in series by looping through, and can work completely independently. The communication

between robotic arms and PC controller is connected by the Control Area Network bus. CAN protocol detects and corrects transmission errors caused by electromagnetic interference. The local PC can directly control the robotic arm via Visual Basic code, and it can also be treated as server controller. Client PCs can access and control the robotic arm remotely through Socket communication mechanism with certain IP address and port number. A Java3D model is created on the client PC synchronously for customers online monitoring and control. The forward and inverse kinematic analysis is solved by Vector Algebraic Method. The Neutral Network Method is also introduced to improve the kinematic analysis. Multiple-layer networks are capable of approximating any function with finite number of discontinuities. For learning the inverse kinematics neural network needs information about coordinates, joint angles and actuator positions. The desired Cartesian coordinates are given as input to the neural network that returns actuator positions as output. The robot position is simulated using these actuator positions as reference values for each actuator.

Acknowledgments

First, I would like to express my deepest gratitude to my advisor, Dr. Dan Zhang, for his constant support and encouragement over the past two years. Dr. Zhang's broad knowledge and enthusiasm for everything we have worked on have been an inspiration for me. I have also learned from him to be patient and open-minded toward everything. I would like to thank him for his invaluable supervision, full support and critical review of the manuscript. Without his ideas and continuous encouragement, this work would never have been possible.

Specially, I am thankful to Dr. Jianhe Lei, who had a crucial role in my Master dissertation. He was a source of valuable guidance and assistance for conducting research and overcoming its numerous and unexpected obstacles. His insights and experience had a positive influence on my work, particularly, in terms of providing proper research directions and developing ideas for writing this thesis.

Many thanks to members of the robotics and automation laboratory in UOIT for their intellectual advice and the endless encouragement. They are always there to lend a helping hand and keen in teaching me, sharing their ideas with me and approach in problem solving.

Last but not least, I am deeply in debt to all my family members. My dearest parents are the strongest supporters for my graduate student life emotionally and intellectually. I am also greatly indebted to my girlfriend Amanda Shou for her patience and understanding as well as her time spent and effort in assisting me.

List of Figures

List of Tables

List of Acronyms

VB:	Visual Basic
DOF:	Degree of Freedom
CAN:	Controller Area Network
GUI:	Graphic User Interface
ISO:	International Organization for Standardization
OSI:	Open System Interconnection Reference Model
NRZ:	Non Return to Zero
IDE:	Interactive development environment
COM:	Component Object Model
UDP:	User Datagram Protocol
TCP:	Transmission Control Protocol
API:	Application programming interface
SRMR:	Serial Reconfigurable Modular Robotic
FIFO:	First-In-First-Out buffers
SCARA:	Selective-Compliance Assembly Robot Arm

Chapter 1

Introduction and Motivation

According to the Oxford dictionary, the word robot is an “apparently human automation, intelligent and obedient, but impersonal machine”. A robot is defined as an automatic device that executes different functions ordinarily ascribed to human beings. By general agreement, a robot is a reprogrammable, multifunctional manipulator, which is designed to move materials, tools, parts, or specialized devices, by variable programmed motions for the performance of a variety of tasks. Robots can be classified into many types by different criterion. Robots can be classified by size, such as macro-robots whose dimensions are measured in meters. Micro-robots bear dimensions allowing them a reach of a faction of a millimeter. Robots can be also classified by application, such as material-handling, surveillance, surgical operations, rehabilitation and entertainment. The most comprehensive criterion would be by structure, such as parallel manipulators and serial manipulators, which is a more straightforward classification.

A parallel mechanism is a closed-loop mechanism of which the end-effector is connected to the base by a multitude of independent kinematic chains. Generally it comprises two platforms which are connected by joints or legs acting in parallel [1]. Parallel robots have been introduced to various industries and fields, such as manufacturing production configurations [2], assembly robot arms in automotive

applications [3], deep sea exploration [4], etc. A general parallel robot is shown in Fig. 1.1.

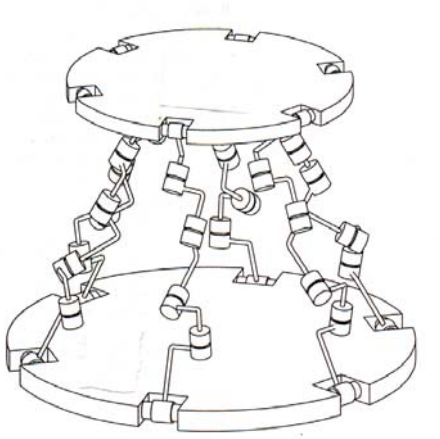


Fig. 1.1 General Parallel Manipulator (J. Angeles, 1997)

However, one of the disadvantages of parallel robots is a relatively large footprint-to-workspace ratio, for example, the hexapod parallel robot: easily take up a sizable work area. The exception is the Triceps' robot [5] which requires less space. Another limitation of the parallel configuration is that it has a small range of motion due to the configuration of the axis when compared to a serial link machine. Therefore, serial manipulators, which have large work space with respect to their own volume and occupied floor space, are still the most common industrial robots by far.

Usually serial robots are built with an anthropomorphic mechanical arm structure, i.e. a serial chain of rigid links, which are connected by prismatic joints and revolute joints. From rigid body motion, a serial robot usually requires six degrees of freedom in order to place a manipulated object in an orientation and an arbitrary position in the workspace of the robot. Therefore, most serial robots have six joints. However, in today's industry, the most popular application for serial robots is for pick-and-place assembly. This application requires less than six degrees of freedom. A special assembly robot of this type is called

SCARA type. SCARA is an acronym standing for Selective-Compliance Assembly Robot Arm, as coined by Hiroshi Makino [6], the inventor of this new class of robots. The class provides motion capabilities to the end-effector that are required by the assembly of printed-board circuits and other electronic devices with a flat geometry. In most general form, a serial robot consists of a number of rigid links connected with joints. Simplicity considerations in manufacturing and control, have led to robots with only revolute or prismatic joints and orthogonal, parallel and/or intersecting joint axis (instead of arbitrarily placed joint axis). Fig. 1.2 shows an example of a serial robot.

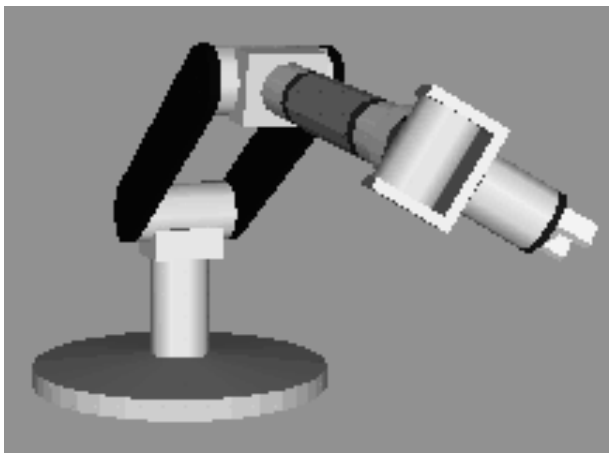


Fig. 1.2 a serial manipulator with 7-DOF in a kinematic chain

However, in many environments, the situation is unstructured and less predictable, such as aboard a space station, a nuclear waste retrieval site, or a lunar base construction site. It is almost impossible to design a single robotic system which can meet all the requirements for every task. In these circumstances, it is important to deploy a modular reconfigurable robotic system, which is suitable to various task requirements.

A robot usually cannot work alone in many situations, since it is unlike human beings, who can think and make a decision by themselves. A robot can only complete tasks with existing programs under certain requirements. It will get stuck or confused when it is

facing various new problems and accidents. Therefore, another goal of this thesis is to develop a real time remote control program to monitor and manipulate, which can load on the serial reconfigurable modular robotic system. An internet based framework can serve real time data from bottom up and can function as a constituent component of e-manufacturing. The framework is designed to use the popular client-server architecture and view-control-model design pattern with secured session control. The proposed solutions for meeting both the user requirements demanding rich data sharing and the real-time constraints are: (1) using interactive Java 3D models instead of bandwidth-consuming camera images for visualization; (2) transmitting only the sensor data and control commands between models and device controllers for monitoring and control; (3) providing users with thin-client graphical interface for navigation; and (4) deploying control logic in an application server [7].

1.1 Literature Survey

Nowadays, modular robots perform a more and more important role in industrial robotic system. Many modular robotic systems have been set up and demonstrated, including the “Reconfigurable Modular Manipulator System” developed by Khosla and co-workers at CMU [8], the several generations of the “Cellular robotic system” developed by Fukuda and co-workers at Nagoya University [9], and modular manipulator systems developed by Cohen et al. at University of Toronto [10], by Wurst at University of Stuttgart [11], and by Tesar and Butler in University of Texas at Austin [12].

The modular robotic system in [10] employs both revolute and prismatic joints. Both of them are actuated by DC motors. The revolute joints use harmonic-drive transmissions and the prismatic joints use ball-screw transmission. The modular robot in [11] consists of a lot of rotational joints driven by AC motors in conjunction with differential gears, and links of square cross-section. The compact joint design allows complicated 2-DOF or 3-DOF rotary motions.

The robotic system of controlling of the HERO 2000 (Fig. 1.3) robot from a PC is developed by Kumar, Sathish [13]. He used the HERO 2000 robot and its remote control unit in the LINK mode. The motion controller is programmed with a serial communication in C program. The remote control unit has a few specified keys with respect to certain built-in functions of the robot. The serial communication program provides additional keys to perform more built-in functions, which support flexibility in programming the robot's motions. The added feature could be developed as camera based monitoring system for the robot.

Fig. 1.3 Heathkit Hero 2000 Robots (by so-cal Hero Group)

An IP network based remote sensing and control system was developed by Zhiwei Gen in 2007 [14]. The operator can control a slave manipulator with sensing capabilities located at a remote site. Control commands and sensory feedback are transmitted through computer networks. The control and sensing data involved in such systems differentiate themselves from other media types in that they require both reliable and smooth delivery. Reliable delivery requires that the transport service have TCP style semantics. By being smooth, the transport service should be able to deliver the control and sensing data with both the average latency and the standard deviation of the latency bounded and reduced. Traditional transport services have great difficulty meeting the latency requirements of delivering sensing and control data over the communication networks.

A lot of research has been done on the remote control with Internet-Based Control, which sets up a connection between the robot and the user through internet. One application is using a software package of MATLAB, SIMULINK, WinCon, Visual C++ and real time workshop. WinCon is an ideal rapid prototyping and hardware-in-the-loop simulation workhorse for control system and signal processing algorithms. A real-time Windows application that runs Simulink models in real-time on a PC, WinCon allows for quick and seamless design iterations without the need to write code by hand. The major advantage for this method is that WinCon provides real-time control. However, WinCon is limited by SIMULINK, which controls the manipulator. SIMULINK is impossible to directly incorporate models for real power semiconductors, and the complexity of the block diagram, which is used to simulate the power circuit, can increase drastically with the number of semiconductors in the circuit. In addition, the user has to pay for WinCon

software in order to access remote control hardware. Fig. 1.4 shows the user interface of WinCon 7.

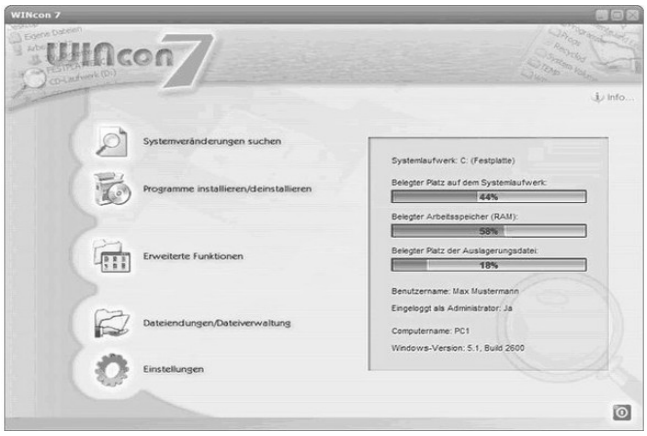


Fig. 1.4 Command window for WinCon 7 (by Quanser)

Another application is using NetMeeting to achieve remote control. The client only needs to call through NetMeeting with administrator's permission, then the client can remotely share the software to control the manipulators. This system doesn't require any java-enabled web browser and NetMeeting is a freeware. However, at the server side, there must be a person to give permission and authorize the client every time, which makes the process inconvenient and complicated. Fig. 1.5 shows the operation window for NetMeeting, which is produced by Microsoft.

Fig. 1.5 Operation Window for NetMeeting (by Microsoft)

1.2 Subject of Study

Robots can work automatically since scientists have preprogrammed them before manufacturing, and they also supervise, edit and improve robots from time to time by direct physical interaction. Robots can also solve a lot of the problems encountered when they are introduced to many small and medium size enterprises. In fact, these companies are not as big as the automotive industry and they have a much larger spectrum of applications and automation environments than found in the automotive final assembly lines. Additionally, the economy is a sensitive problem to small and medium companies for a big investment. They also don't have the permission to access the automation and robot expertise, which are found in the organization of an automotive manufacturer. Therefore, it becomes an urgent affair to develop a robot family at a reasonable and acceptable price level which can satisfy all the applications and requirements of small and medium sized enterprises. In a recent study, developing the modular robots is the best choice for success in this case. With this solution, the industrial robots will be assembled for various work space sizes, for various load weights and for various performance requirements.

However, the modules can be designed and assembled as standard components in many different ways. A set of modules can be built to a lot of unique assembly configurations. It is used as a key for the module assembly planning problem to find an optimal configuration, which can satisfy the requirements, since a modular robot can achieve a large number of objectives by reconfiguring small inventory modules. This module assembly problem has been divided into two sub problems: module assembly

enumeration and task oriented optimal configurations. Fortunately, both of them have been conquered in a systematic way by Dr. Chen in 1994 [15].

A highly modular robot concept is going to increase the difficulties of the configuration, and maintenance. So in this thesis, a virtual environment is developed with configuration tools for robot and cell modules. Thus, customers can remotely control robots through the virtual environment, instead of camera images, this thesis used a Java3D model, which, with behavioral control nodes, can do a better job because of low volume message passing. It largely reduced network traffic, and makes real-time monitoring, on-line inspection and remote control much faster and stable without disconnecting, delay and stuck.

In summary, this thesis focuses on the development of Serial Reconfigurable Modular Robotic System. A single 4 Degrees of Freedom robotic arm is assembled with a certain configuration, which is connected to controller PC through CAN bus. The remote control functionality is also added into this system. Although this thesis is working on a single robotic arm, this arm contains all the functions needed. It is the fundamental research for future development. A work cell will be set up by many robotic arms with same functionality. With internet based remote control technology, engineers can manipulate robots from a distance, in order to work in high risk environments with radiation, research under the deep sea or explore the areas inaccessible to human beings.

1.3 The Organization of the Thesis

The thesis consists of 6 chapters. Chapter 2 presents the general design of serial reconfigurable modular robotic system. The objective of the SRMR system will be described. A detailed analysis of system requirements will be illustrated in two parts: client side and server side with software and hardware development. Real time control and the openness of the system are also discussed in general in this chapter.

Chapter 3 focuses on the hardware setup of the robotic system. CAN (Controller Area Network) serial bus system will be introduced, which includes Industrial applications of the CAN network, how the CAN network functions, and Implementations of the CAN protocol. PowerCube modules will also be introduced with its internal controller. Internet is used as the communication medium.

Chapter 4 deals with the software part of the robotic system. Server is used to control the serial reconfigurable modular robot directly. It is programmed by Visual Basic.net. Client is designed as a user-friendly interface. A 3D model is created by Java3D which is synchronized with synchronized with the motion of serial reconfigurable modular robot. The communication is set up by socket protocol.

Chapter 5 introduces the serial modular reconfigurable robot assembled by PowerCube modules. The nine different conceptual configurations are discussed for different workspaces. Some concepts underlying kinematic structure development will be addressed, such as Cheychev-Grubler-Kutzbach criterion, a basic theory for kinematic chain degree of freedom distribution and criteria for better and practical kinematic structures of machine tools. Upon the conceptual structure, forward and inverse kinematic

problems are developed. Neural network has also been used to improve the kinematic analysis.

Chapter 6 brings together the most important conclusions and observations of the study and suggestions for the work to be done in the future.

Chapter 2

General Design of the SRMR System

2.1 The Objective of SRMR System

The primary purpose to invent serial manipulators is to replace human beings in dangerous work conditions, such as searching for injured people in nuclear radiation environments, refloating sunken ships under the deep sea, and so on. They are also very useful to perform repetitive tasks which are too tedious for human beings, such as assembling tons of screws to modules, pushing and releasing springs to test the elasticity, etc. Recently, Serial manipulators have revolutionized the modern assembly lines. Tasks can be performed faster with higher accuracy and better quality. They also have the advantage of having a large workspace with respect to the amount of space they occupy.

General speaking, serial manipulators consist of two types of joints: revolute or prismatic joints, which have been connected with rigid links. A revolute joint (also called pin joint or hinge joint) is a kinematic pair with one degree of freedom used in mechanisms. Revolute joints provide single-axis rotation function used in many places such as door hinges, folding mechanisms, and other uni-axial rotation devices. An example of revolute joint is shown in Fig. 2.1.

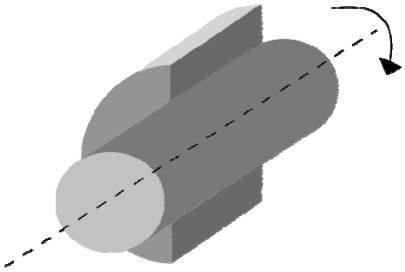


Fig. 2.1 Revolute Joint

A prismatic joint (also called sliders) is a kinematic pair with one degree of freedom used in mechanisms, which performs constitute purely linear motion along the joint axis. Prismatic joints provide single-axis sliding functions, therefore, they are used in places such as hydraulic and pneumatic cylinders. A prismatic joint sample is shown in Fig. 2.2.

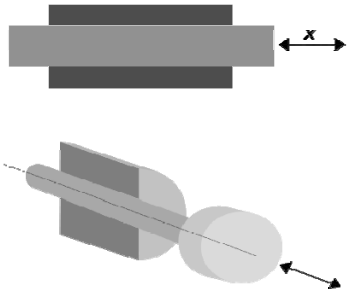


Fig. 2.2 Prismatic Joint

In this thesis, the four degrees of freedom serial manipulator is built by all the revolute joints. This robot arm has one end attached to the ground and it is fixed. The other end is connected with a two finger gripper, which is free to move around in its workspace envelop, as shown in Fig 2.3.

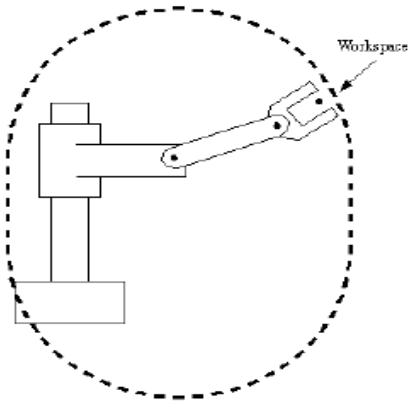


Fig. 2.3 Workspace envelop

A serial manipulator usually has very limited workspace and it can only actualize the task, which is eligible in the workspace. Therefore, many industry factories have multiple serial manipulators to execute various separate tasks due to the workspace limitation. A serial reconfigurable modular robot (SRMR) is designed to solve this workspace problem. It can achieve various tasks by switching configurations. The SRMR consists of links and joints, which are modular and they are able to be recombined to various structures to reach different workspace.

In order to arbitrarily position and orient a robot end-effector in space, a reconfigurable modular serial manipulator with at least 6 degrees of freedom is required. However, for most industrial tasks such as welding and assembly, robots with three to five degrees of freedom are sufficient. The advantages of these simpler robots are less expensive and easier to control. In this thesis, the SRMR is designed to have four degrees of freedom. This manipulator can overcome the shortcoming of the serial manipulators to provide a revolutionized assembly line and can be used for a variety of different applications as required.

2.2 Requirements Analysis

Before designing and implementing the SRMR system, the requirements must be clarified first. Basically, the requirements are analyzed for server and client side.

2.2.1 Requirements at server side

The server side has to be able to control the modular robotic arm. It should be able to debug the SRMR system at local PC controller and execute commands from the client side remotely. Therefore, the graphic user interface at server side should include sections for both local control and remote control. The local control panel should be able to scan modules and initialize their states. It also should be able to control each module separately and independently for debugging. The remote control panel should be able to receive commands from client side and perform required motions. The GUI could have a similar appearance with Fig. 2.4.

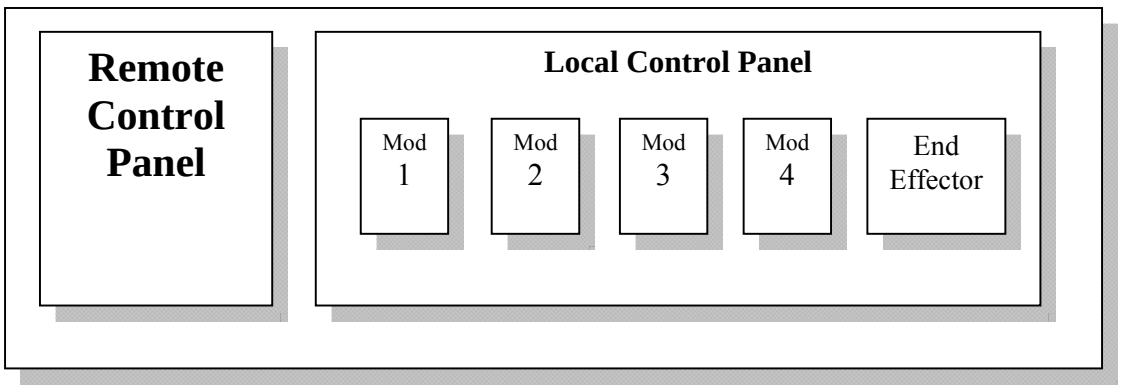


Fig. 2.4 The proposed GUI of server side

The server controller will be able to connect to the modular robotic arm via CAN bus. The robot controlling language program will be checked for syntax error. Once the command codes are valid and obey certain rules which are agreed by the server and the client, the server side will interpret those codes and attempt to control the modular robotic arm. In order to answer the request and execute the command from the client side, the server side should be able to communicate with the client side. The server side has to wait for and answer the connection request from the client side. As soon as the client sends a disconnection request, the server should release the connection. The server also should be able to handle the conflict of multiple connection requests. If the server has already connected to a client, the server will put all later connection requests in a waiting list in the order of arrival. Once the server is free, the first waiting connection request will be handled immediately. If the server receives a disconnection request from a client, which has a connection request in the waiting list, the connection request will be cancelled.

2.2.2 Requirements at client side

The client side should be able to simulate the motion for robotic arm independently, without connecting to the server. This function is designed to test and examine path planning in virtual environment before applying it to the real robotic arm to avoid unnecessary damage. Of course, the main function for client side is to be able to connect the server to the remote control modular robotic arm. In order to transmit commands to the server side, the client side must have connect and disconnect functions with respect to certain IP address of the server. The client side should have a friendly graphic user

interface, which provides a 3D virtual model with a similar appearance of a robotic arm. The model will be displayed on the GUI and consist of four cubic modules and one two finger gripper. The 3D robot model will be able to do the following movements according to the input command: rotating clockwise and count clockwise for rotary modules and picking and releasing for end-effector. In order to control the 3D robot model, customers should be able to input commands via GUI. Those commands will be interpreted by internal codes. A number of buttons will execute these codes and perform corresponding actions. Those buttons include: connecting and disconnecting server, calculating rotate angles, sending commands to server, performing certain actions, and exiting program safely. The GUI could have a similar appearance with Fig. 2.5.

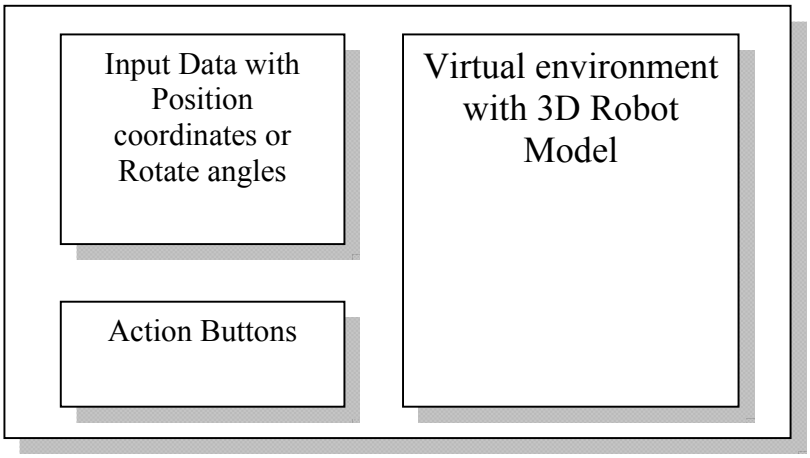


Fig.2.5 The proposed GUI of client side

2.3 Proposed SRMR System Design

According to the design objective, a 4 DOF robotic arm will be assembled as a part of SRMR system. The robotic arm is built by PowerCube modules. The detailed information of these modules will be introduced in Chapter 3. There are a number of

configurations to build the robotic arm in order to reach different workspaces. The design of configurations will be illustrated in Chapter 5. Besides the robotic arms, this system also includes power supply system, CAN bus, terminal block and PC controller. As shown in Fig. 2.6, the robotic arms perform as the core part in this system, which execute picking, holding, releasing, rotating, and translating, and so on. As mentioned at the beginning, the robotic arm is assembled by PowerCube Modules with cubic outward appearance. The control and power electronics are fully integrated on the connector block inside the modules. Those modules are connected in series by looping through, and can also work completely independently. The master control system generates the sequential program and stores the current sequential command in the module. The subsequent command is stored in the buffer, and then the command is sent to the connected modules step by step. The power system is set up according to total modules voltage and current. It supplies both power voltage and logic voltage in order to make all the modules work properly. The communication between robotic arms and PC controller is connected by the CAN bus. One terminal is connected via 9-pin SUB-D port on the terminal block and the other terminal is connected via USB bus on PC. The maximum data transfer rate is 1 Mbit/s, which is constructed and transmitted by the CAN chip. The CAN protocol corresponds to the data link layer in the ISO (International Organization for Standardization) and OSI (Open System Interconnection Reference Model) reference model, which meets the real-time requirements. CAN protocol detects and corrects transmission errors caused by electromagnetic interference. It is also easy to configure the overall system and the central diagnosis.

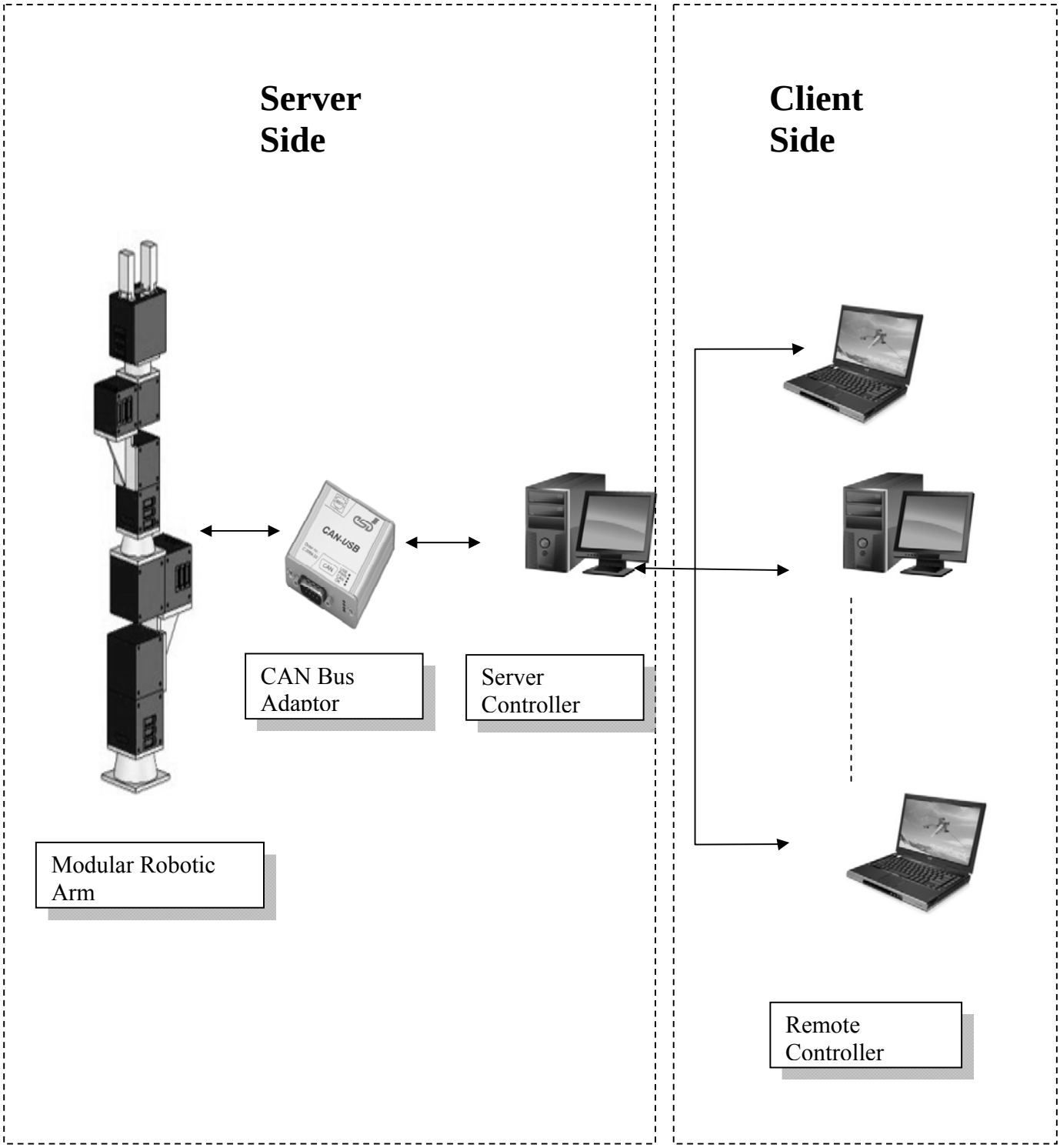


Fig. 2.6 Structure of SRMR system

The control interface is programmed by Visual Basic.net so that the local PC can send commands and retrieve data from PowerCube modules via VB.net code. When open socket, the local PC can be considered as a server, which is waiting for the command from internet with matching IP address and port number. The other PCs can perform as clients and remotely control the robotic arms through a socket. A Java3D model is created on the client side, which executes the same action as the real robotic arms. The Java3D model can also be executed offline. Customers can do simulations to avoid damage in reality. A detailed explanation and calculation is illustrated in two major parts: hardware and software in next two chapters.

2.4 Summary

In this chapter, the objective of building this SRMR system is explained at the beginning. SRMR system can achieve various tasks by switching configurations. The SRMR consists of links and joints, which are modular and they are able to be recombined to various structures to reach different workspace. The requirements of building such a SRMR system are analyzed from the server and the client side. An overview of the proposed system is introduced from two major parts: hardware and software of the SRMR system.

Chapter 3

Hardware Architecture of SRMR System

3.1 Modular Robotic Arm Assembling

The modular robotic arm is assembled with PowerCube modules in this research. The modules of the PowerCube series provide the basis for flexible combination in automation. The modules are manufactured with a built-in Servo motor, incremental encoder for positioning, and velocity control, limit switches monitoring, voltage monitoring, current monitoring and temperature monitoring. The cube geometry provides diverse possibilities for creating individual solutions from the modular system. The control and power electronics are fully integrated in the modules. The integrated high-end microcontroller is good for rapid data processing. The decentralized control system is also useful for digital signal processing. There are six different modules in PowerCube series: PG for servo-electric 2-Finger Parallel Gripper, PR for servo-electric Rotary Actuators, PW for servo-electric Rotary Pan Tilt Actuators, PSM for servo-motors with

integrated position control, PDU for servo-positioning motor with precision gears, and PLS for servo-electric Linear Axes with ball and screw spindle drive.

In this thesis, the robotic arm is assembled by four PR modules and one PG module as end effector. The PR module is equipped with a harmonic dive precision gear, which is driven directly by a brushless DC servo-motor, as shown in Fig. 3.1. This rotary actuator is electrically actuated by the fully integrated control and power electronics. It has brushless DC servo-motor as drive, in order to perform high versatility due to the controlled position, speed and torque. High torques and speeds can also provide rapid acceleration and short cycle times. Its versatile actuation option is compatible with existing popular servo-controlled concepts via CAN bus, RS-232 and Profibus DP.

Fig. 3.1 PowerCube PR module

Inside the PowerCube module, there are six assemblies, which are shown clearly in Fig. 3.2. Area 1 is the control electronics, which integrate control and power electronics. Area 2 is the encoder for position evaluation. Area 3 is the motor for maximum torque at

46 NM. Area 4 is the harmonic drive gear, which is typically used for gearing reduction, but may also be used to increase rotational speed or for differential gearing. Very high gear reduction ratios are possible in a small volume. Area 5 is the brake for holding function when the unit is stationary and on power failure. Area 6 is damp-proof cap linking to the customer's system.

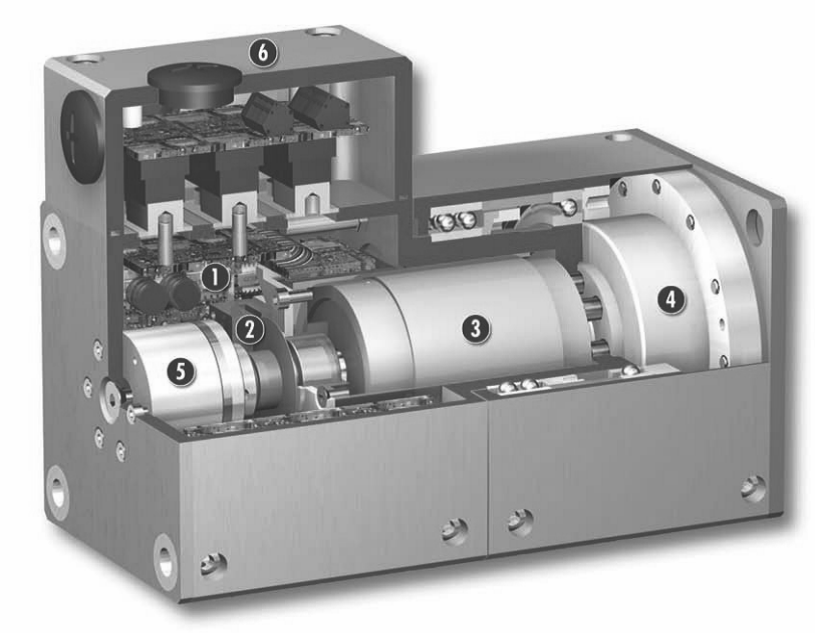


Fig. 3.2 Sectional diagram of PowerCube module

The end effector is built with PG 70, which is a servo-electric two-finger parallel gripper, and shown in Fig. 3.3. A wide variety of different fingers can be attached for parallel grippers due to design of the gripper base jaws. There is a magnetic brake which is actuated immediately if power fails. This gripper can perform gripping, holding and releasing actions.

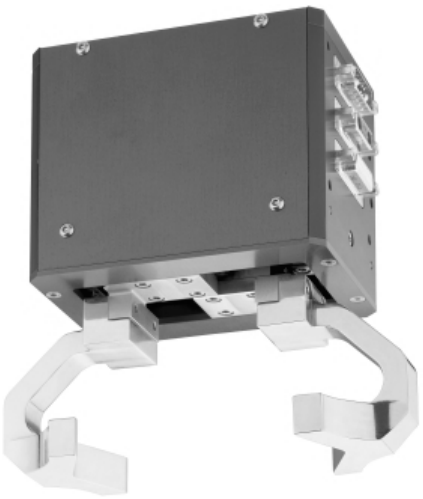


Fig. 3.3 End effector: PG 70

The internal logic receives the parameters from the operation software, in order to control the actuator. The mechanical movement of the actuator is made by the gripper jaws. The position is constantly monitored. Sensors transmit the data back to the internal logic. The gripper movement is linear, and is controlled via the user interface.

Each PowerCube Module is connected with the help of the connector-block (Fig. 3.4), which is located above the control electronics. There are six terminal blocks embedded on the connector block. They are used for logic voltage, motor voltage 1 (24V), motor voltage 2 (48V), CAN bus, Profibus DP and RS232. Those terminal blocks are connected by PowerCube cables, which contain seven different colored wires. The white and brown wires are connected to UL- and UL+ to supply 24V for logic voltage. The blue and red wires are connected to VM1- and VM1+ to supply 24V for motor voltage. The yellow, green, and shield are connected to CANL, CANH and SHD accordingly for CAN communication. Last module has to keep jumper J1 active for CAN terminator.

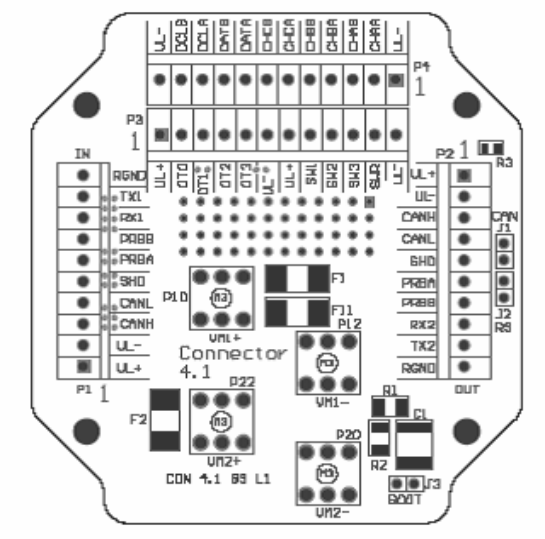


Fig. 3.4 Connector Block

3.2 Power System Design

The power supply to PowerCube is daisy chain type. Although all the modules are connected in series, the electrical circuits inside the modules are parallel connected. Each module has the same voltage and the total current is the summation of the current from each module. From the electrical operating data, the PR70 rotary module has 24V DC nominal voltage and 4A nominal current. The PR90 rotary module has 24V DC nominal voltage and 6A nominal current. The PG70 gripper module has 24V DC nominal voltage and 2.5A nominal current. The power for each module can be calculated as follows:

Power for PR70:

$P_1 = V \times I = 24V \times 4A = 96W$

Power for PR90:

P_2 = V x I

= 24V x 6A

= 144W

Power for PG70:

P_3 = V x I

= 24V x 2.5A

= 60W

There are two PR70, two PR90 modules and one PG70 module, so the total power is:

P_{total} = 2 x P_1 + 2 x P_2 + 1 x P_3

= 2 x 96W + 2 x 144W + 60W

= 540W

MeanWell SE-600 (Fig. 3.5) has been selected as DC Power Supply in experiments. SE-600 series is a 600 Watt high power density AC/DC single output power supply. Standard features include: built in remote sensor function, built in DC fan, output voltage adjustment, short circuit protection, overload protection, over temperature protection, and over-voltage protection. Two power supplies supply motor voltage and logic voltage individually.

Fig. 3.5 SE 600 – 24V

3.3 Control Area Network Bus and Protocol

3.3.1 Types of Field Bus

There are many standard connection types that can be used as control equipment for PowerCube technology. RS-232 (Fig. 3.6) is one of the most well known interface standards, which is a single-ended standard. The data transmission rate can be up to 20kbps. Since the maximum load capacitance is defined at 2500 pF, the maximum cable length is 15m. It is possible to lower the capacitance of cable in order to increase the length. Voltage levels with respect to a common ground represent the signals. A binary 0 is called a space. A binary 1 is called a mark. For transmission, the space is +5 to +15 Vdc and the mark is -5 to -15Vdc. For receiving, the space is +3 to +15Vdc and the mark is -3 to -15Vdc. The major advantages of RS-232 are commonly available, easy to wire and relatively low cost, however, RS-232 has obvious disadvantages: short length and slow data transmission rates. It is also easily affected by noise.

Fig. 3.6 RS232 interface

Profibus (Fig. 3.7) is another widely used communication field bus in the industry. This work is concerned in its variation Profibus DP (Distributed Peripherals) designed for communication between control units (like PLC or industrial computers) and distributed peripherals (sensors, actuators) over shared connection. Profibus is master-slave protocol with deterministic medium access control (stations can transmit only with permission), where one connection can be used by multiple masters and each master has granted maximum time delay for acquisition of the transmit permission. With arising use of PCs' for control purposes in the industry, Profibus access is also demanded from them. So far it was necessary to have a special hardware card, which was able to handle Profibus access. These cards are based on their own communication processor, customer's circuit or another hardware solution with enough power to meet high Profibus requirements [16]. Disadvantage of these Profibus cards is their high price. This brought the motivation to make a realization of Profibus as cheap and simple as possible.

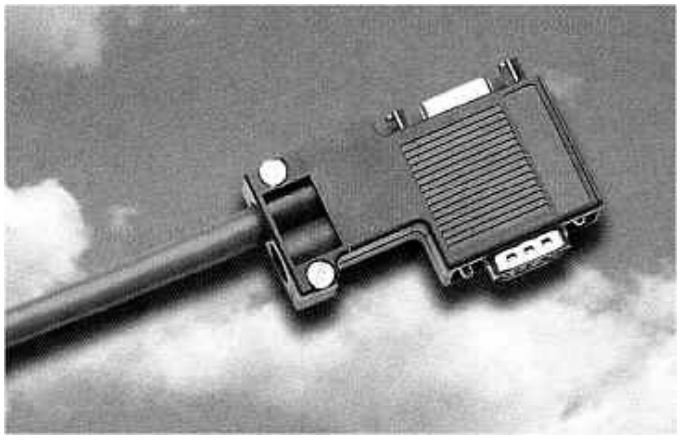


Fig. 3.7 Profibus interface

The Controller Area Network (CAN) bus is a balanced (differential) 2-wire interface running over a shielded twisted pair (STP), un-shield twisted pair (UTP), or Ribbon cable. Each node uses a Male 9-pin D connector (Fig. 3.8). Pin 2 and 7 are connected to CAN

low and CAN high signal lines. Pin 5 is connected to shield which is connected with case of the 9-pin DSUB connector. Pin 3 and 6 are the reference potential of local CAN physical layer. All the other pins are reserved for future applications.

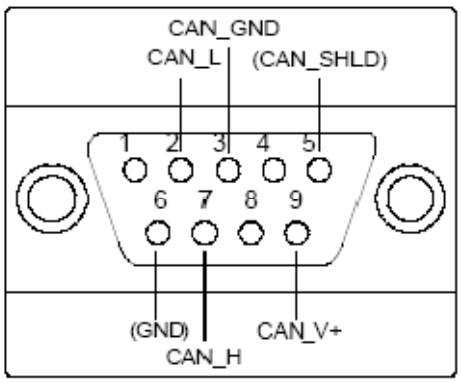


Fig. 3.8 Pin Position

Compared to the two connection types above, CAN Bus has its own advantages with linear bus topology so that the bus is still operational for all the other stations even when one station breaks down. The wiring complexity is low. This factor plays an especially large role in automobiles. An economical and easy to manage twisted wire pair serves as the transmission medium. CAN stations can be subsequently added to and removed from the existing CAN bus relatively easily. Only the connection to the bus line must be made or disconnected. This aspect plays a significant role, especially with trouble shooting and repairs. The breakdown of a CAN station has no immediate impact on the CAN bus. All the other stations can communicate unconstrained [17].

3.3.2 Hardware of CAN bus Module

CAN bus has various types of modules for specified usages with respect to different requirements. The module CAN-ISA/331 is a PC board designed for the ISA bus. It uses a 68331 micro controller, which cares for local CAN data management. The CAN data is stored in the local SRAM. The module CAN-Bluetooth is an intelligent CAN-interface with a PowerPC-micro controller for local data management. CAN-Bluetooth enables a wireless transmission of fieldbus data across a range of up to 10 meters. The CAN-interface is ISO 11898-compatible and permits a maximum data-transfer rate of 1 Mbit/s. Via the optionally available accumulator, CAN-Bluetooth can be operated without stationary power supply. In this project, CAN-USB-Mini is used due to the USB hot swapping property, which allows devices to be connected and disconnected without rebooting the computer or turning off the devices. The CAN-USB-Mini module is an intelligent CAN interface with a MB90F543 micro controller for local CAN data management. The block circuit diagram is shown in Fig. 3.9. The maximum data is 1 Mbit/s, and it can be set by means of software. CAN interface and other voltage potentials are electrically insulate by means of optical couplers and DC/DC converters. The power supply is fed via USB bus at 5V and 500mA.

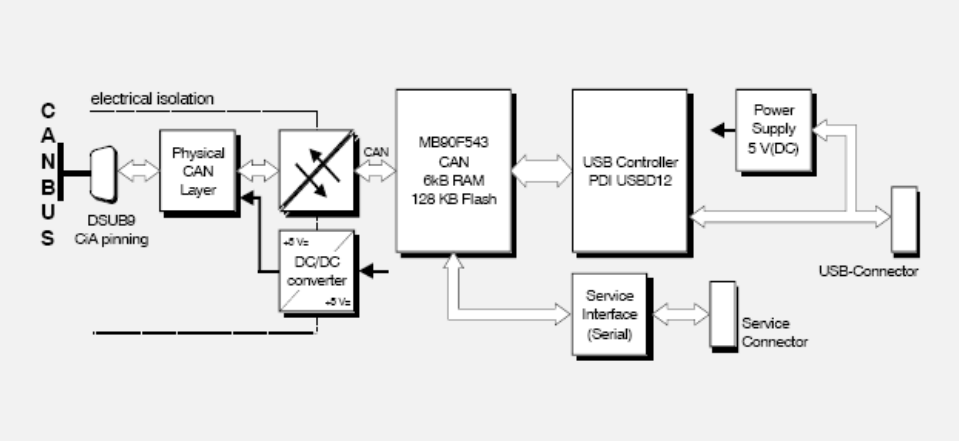


Fig. 3.9 Block-circuit diagram of CAN-USB-Mini module

The bus physical connection consists of three parts: transceiver, CAN controller, and microcontroller. The transceiver is a device that has both a transmitter and a receiver which are combined and share common circuitry or a single housing. It is the first chip, which is connected directly to the bus line and ensures that the specified voltage is held on the bus line. Additionally, it prevents the CAN station from voltage surges. The CAN Controller is a synthesizable IP block providing Controller Area Network (CAN) functionality, compliant with the CAN Specification Revision 2.0 Part B. It is set up in the middle and connected to the transceiver, which receives the signals from the bus line. It filters out the unrelated data for CAN station and sends only the related data along for processing. It can detect errors in the bus and react properly. In addition, it ensures that the CAN protocol is held, when transmitting data. The microcontroller is a small computer on a single integrated circuit consisting of a relatively simple CPU, combined with support functions. This microcontroller has a processing program. According to this, it can evaluate the data from the CAN controller, which can be inquired from the connected sensor system and actuators. Fig. 3.10 illustrates the physical CAN connection.

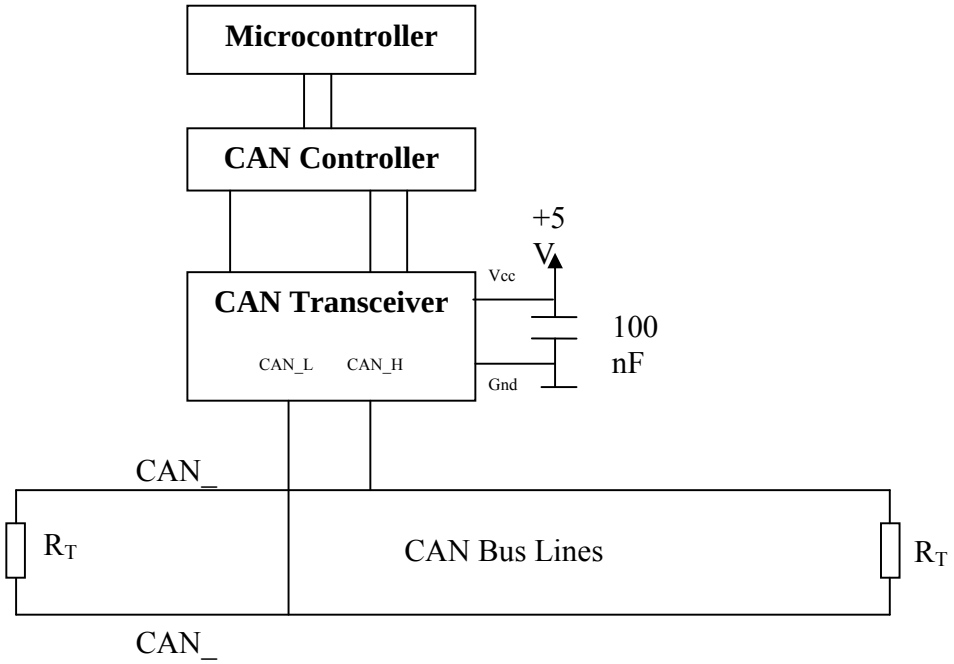


Fig. 3.10 Physical Connection for CAN bus

3.3.3 CAN Data Transmit and Frame protocol

The Bit Encoding used for CAN bus is: Non Return to Zero (NRZ) encoding (with bit-stuffing) for data communication on a differential two wire bus. Non-return to zero encoding（Fig. 3.11） is used in slow speed synchronous and asynchronous transmission interface. With NRZ, a logic 1 bit is sent as a high value and a logic 0 bit is sent as low value. The use of NRZ encoding ensures compact messages with a minimum number of transitions and high resilience to external disturbance.

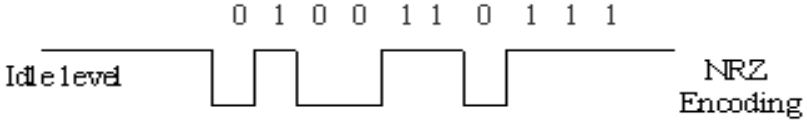


Fig. 3.11 Non-return to zero encoding

Message transfer is manifested and controlled by four different frame types: Data Frame, Remote Frame, Error Frame and Overload Frame. A Data Frame (Fig. 3.12) carries data from a transmitter to the receivers. It contains seven different bit fields: Start of Frame, Arbitration Field, Control Field, Data Field, CRC field, ACK field and End of Frame. The Start of Frame marks the beginning of data frame with a single dominant bit. All the stations have to synchronize to the leading edge before transmission. The Arbitration Field is composed of the Identifier and the RTR-Bit. The Identifier has 11 bits and transmitted in the order from ID-10 to ID-0. RTR Bit is the bit request in remote transmission and has to be dominant. The Control Field contains six bits. It includes the data length code which indicates the number of bytes in the data field. The Data Field consists 0 to 8 bytes which is transferred in Data Frame. CRC Field consists of the CRC sequence followed by CRC delimiter, which includes a single recessive bit. The ACK Field is two bits long and contains an ACK slot and an ACK delimiter, which also includes a single recessive bit.

Bus idle	SOF	Arbitration Field	Control Field	Data Field	CRC Field	ACK Field	EOF	Inter Mission
	1 bit	12 / 32 bit	6 bit	0 to 8 byte	16 bit	2 bit	7 bit	3 bit

Fig. 3.12 Structure of Data Frame

A Remote Frame (Fig. 3.13) is transmitted by a bus unit to request the transmission of the Data Frame with the same Identifier. Contrary to Data Frame, the RTR bit of Remote Frame is recessive and there is no Data Field. It consists of the other six bit fields: Start of Frame, Arbitration Field, Control Field, CRC field, ACK field and End of Frame. The function of each bit field is corresponding to Data Frame.

Bus idle	SOF	Arbitration Field	Control Field	CRC Field	ACK Field	EOF	Inter Mission
	1 bit	12 / 32 bit	6 bit	16 bit	2 bit	7 bit	3 bit

Fig. 3.13 Structure of Remote Frame

The Error Frame (Fig. 3.14) is transmitted when any unit detects a bus error. It consists of two different fields: Error Flag and Error Delimiter. The Error Flag has two forms: Active Error Flag which consists of six consecutive dominant bits and Passive Error Flag which consists of six consecutive recessive bits. The Error Delimiter consists of eight recessive bits.

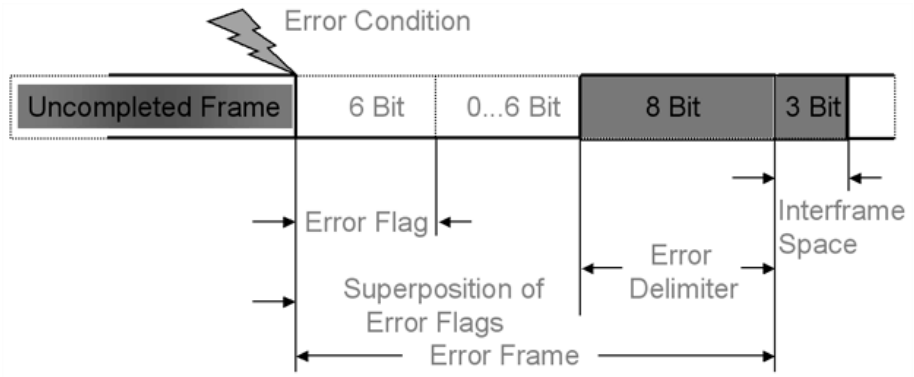


Fig. 3.14 Structure of Error Frame (by Softing Company)

The Overload Frame (Fig. 3.15) contains two bit fields: Overload Flag and Overload Delimiter. An Overload Frame can be generated by a node if, due to internal conditions, the node is not yet able to start reception of the next message, or, if during Intermission one of the first two bits is dominant. Another overload condition is that a message is valid for receivers, even when the last bit of EOF is received as dominant. An overload Frame is used to supply for an extra delay between the progressing and succeeding Data or Remote Frames.

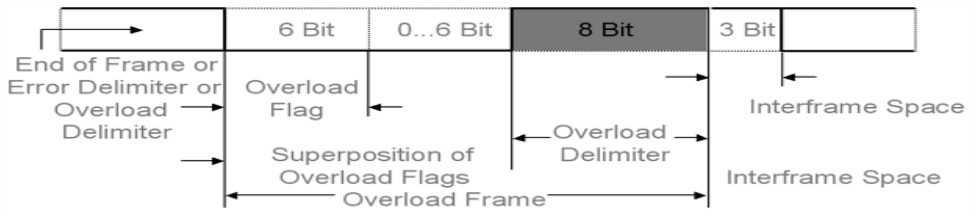


Fig. 3.15 Structure of Overload Frame (by Softing Company)

3.4 Debugging the Hardware of SRMR System

The robotic arm is assembled by PowerCube modules and provided by 24V DC power supply. The local controller connects to the robotic arm via CAN bus. A block diagram shown in Fig. 3.16 illustrates of the hardware of SRMR system.

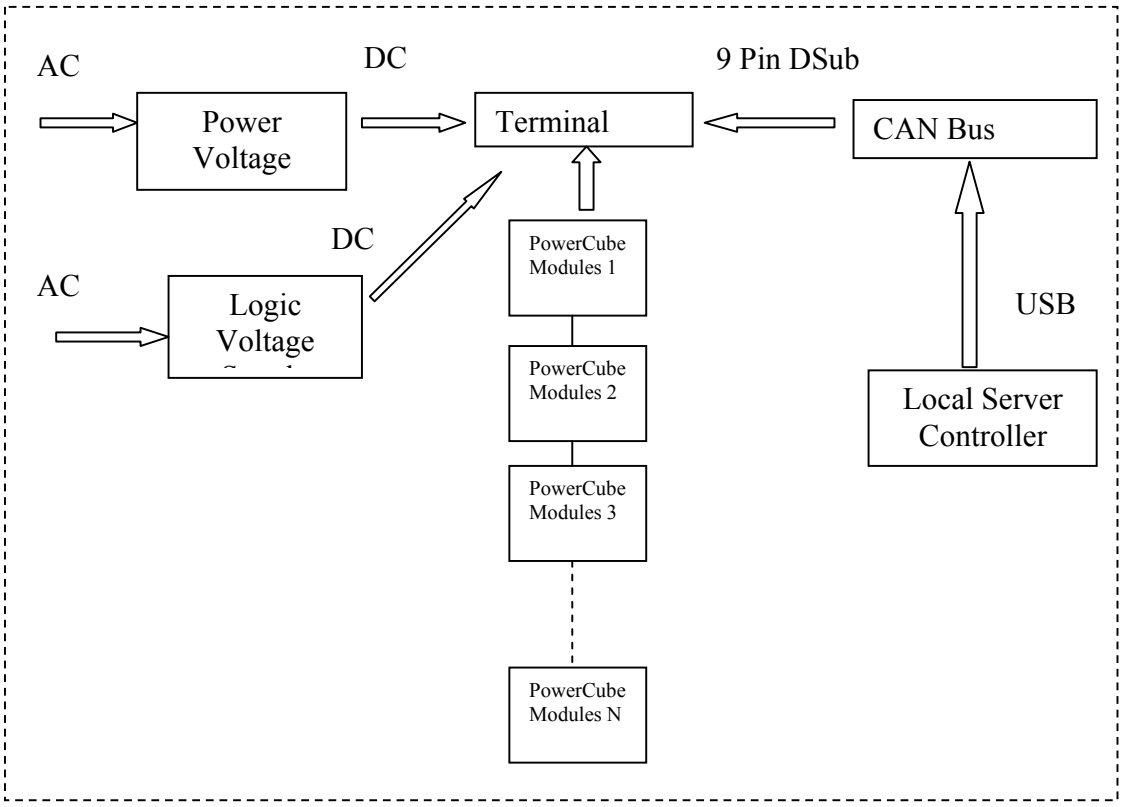


Fig. 3.16 Block Diagram of the Hardware of SRMR System

At this point the SRMR system with a single robotic arm is completely set up. The newest PowerCube software version 5073 is used to test the functionality of this SRMR system. The new module version includes complete newly designed control hardware. New included algorithm allows smooth driving in position mode. The current, voltage and temperature control can be tested in this version.

The Figure 3.17 shows the main dialog, which provides controls for scanning the bus and finds modules. Modules with ID from 08 to 12 are detected. The button “Options” opens a dialog box. In the new box, users are able to setup and configure the InitString depending on the type of field network and baud rate. In telecommunications and electronics, baud rate is the transmission medium per second in a digitally modulated signal.

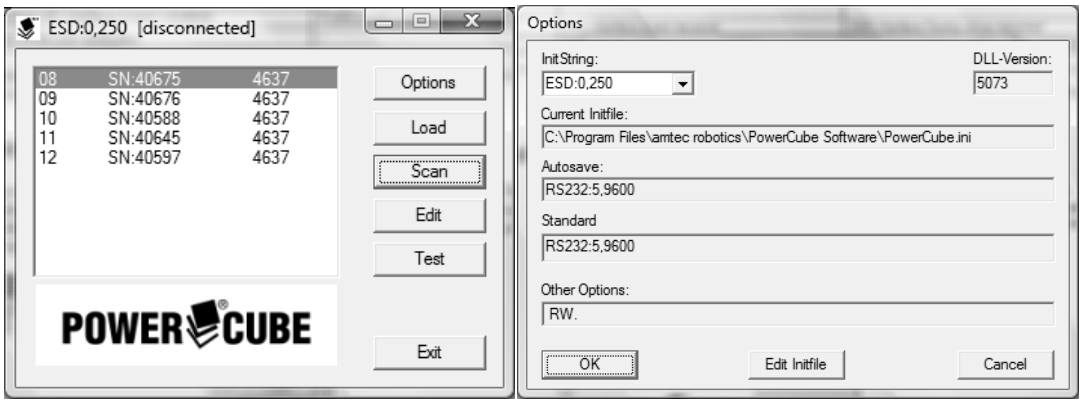


Fig. 3.17 PowerCube Main Dialog

As long as software can detect the modules, users are allowed to open and configure the properties. In the module dialog, there are five property sheets to configure and customize the modules. In the first sheet “Identification” (Fig. 3.18), the user can specify the identification values. The linear screw and belt option is for a linear module with ball

screw spindle transmission and toothed belt transmission. In this experiment, the rotary module is used for all the PowerCube modules. As default, the maximum rpm's of the motor is set to 4025 and the power electronics are selected as 250/500W type, and gear ratio is 161. The nominal current of the motor is used as 9.98. Number of encoder impulses is 2000. Home offset is offsetting to calibrate the zero position of the module. Minimum module position is -160 and the maximum module position is 160. The maximum delta position is the maximum tolerated following error. Module address identifies the module on the bus with its own unique address. All the specifications are applied to module ID from 08 to 12.

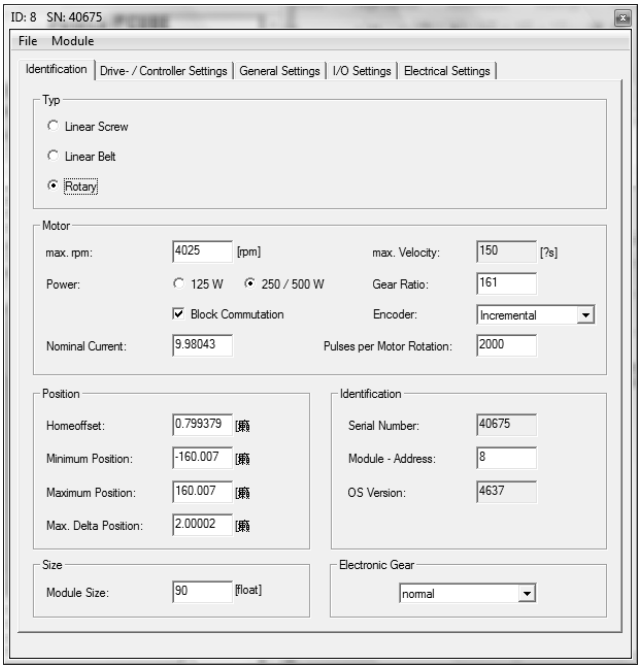


Fig. 3.18 Page of Identification

On the page "Drive-/controller Settings" (Fig. 3.19), the user can specify the drive and controller values. They are able to define the Maximum module speed and acceleration. However, the minimum value is fixed as default. Home velocity is for the module homing procedure. Users can enable the brake option to protect the modules.

Amplification CO, Damp and Nominal Acceleration AO is digital servo loop parameter. Temporary configuration can activate the synchronized transmission, motion and allow the maximum value in current mode.

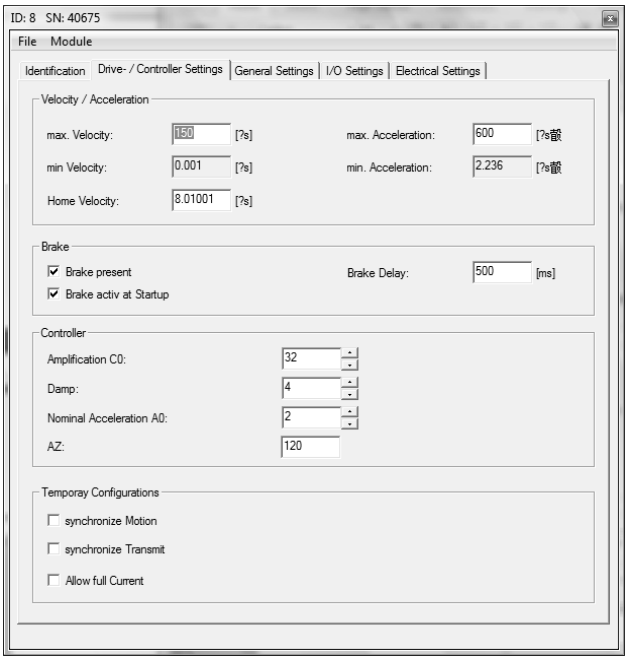


Fig. 3.19 Page of Drive-/controller Settings

On page “General Settings” (Fig. 3.20), the user can configure the communication and special settings. According to the initial setting, CAN bus is set with baud rate 250 for the module. The M3 and M4 Can bus identifiers also have been activated. CANopen is a communication protocol according to DS402. The DS402 is used to provide drives in a CAN network with an understandable and consistent behavior. The profile is built on top of a CAN communication profile, called CANopen, which describes the basic communication mechanisms common to all devices in the CAN network. The purpose of the drive units is to connect axle controllers or other motion control products to the CAN bus. They usually receive configuration information via service data objects for I/O configurations, limit parameters for scaling, or application-specific parameters. The

motion control products use process-data object mapping for real-time operation, which may be configured using service data objects (SDOs). This communication channel is used to interchange real-time data-like set-points or actual values such as position actual values. In the special setting, M3 compatible can activate M3 compatible behavior (Step motion, homing, State information). Release Motor on HALT can switches off the motor control loop on occurrence of an error condition. Zero Moves after HOK can automatically move the module to its zero position after homing.

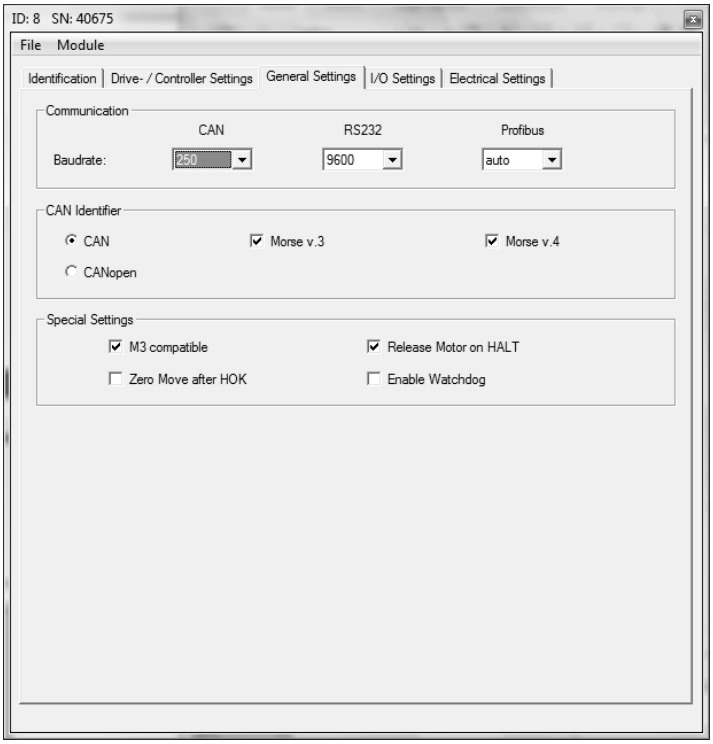


Fig. 3.20 Page of General Settings

On page “I/O Settings” (Fig. 3.21), the user can configure the digital inputs and outputs. Home with encoder-zero selects a homing procedure using the encoder index signal. Falling edge enables homing on the falling edge of SWR. Convert to limit switch enables the home switch SWR as limit switch (except during homing). Enable SWR

enables the home switch SWR. Low active SWR selects the home switch SWR to be low active. External SWR enables the external home switch SWR.

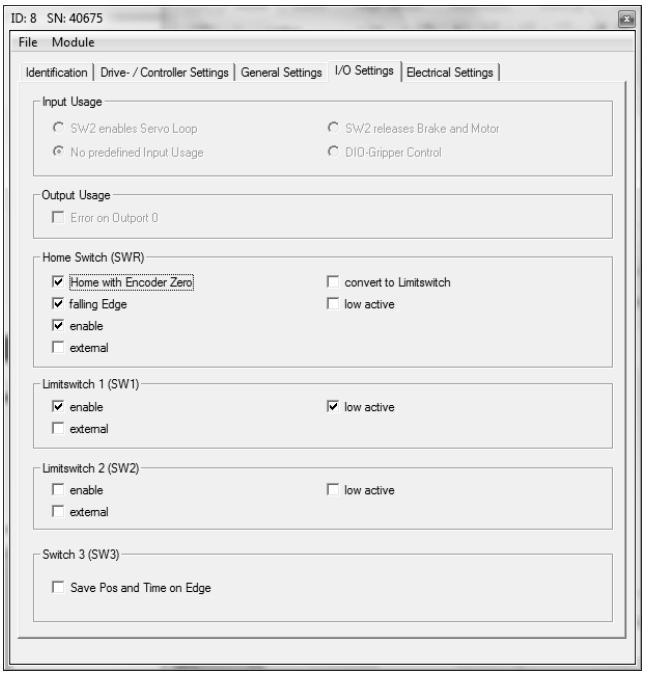


Fig. 3.21 Page of I/O Settings

On page “Electrical Settings” (Fig. 3.22), the user can configure the electrical parameter of the module. These parameters include six voltage data such as Min./Max. voltage supply for logic electronic, Min./Max. voltage supply for motor electronic, Max. Time that Min. logic voltage/motor voltage can be undershot and Max. Time that Min. logic voltage/motor voltage can be overshot. There are also five current parameters, such as nominal current, absolute current limit and Max. current, Max. Time that the nominal/Max. current can be overshot.

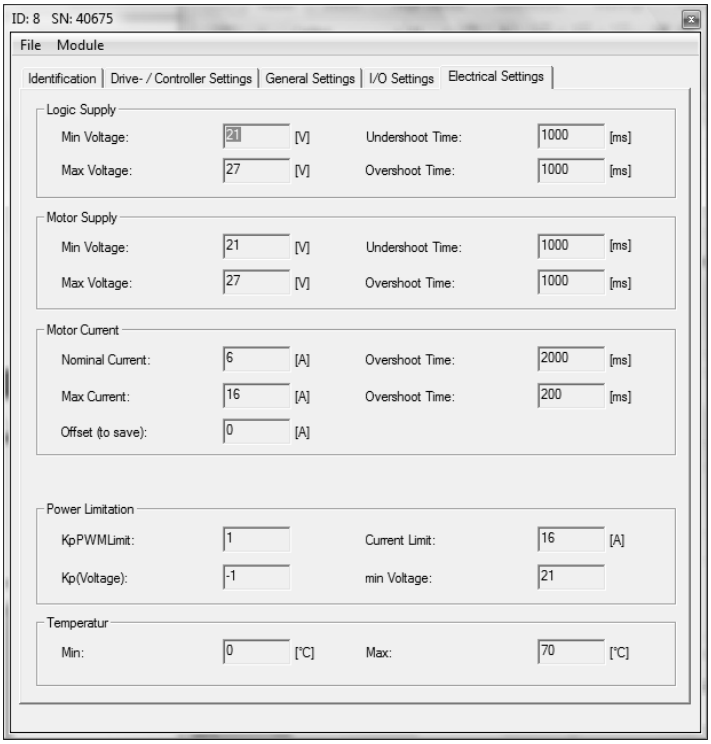


Fig. 3.22 Page of Electrical Settings

After setting up all the module robot configurations, users are able to test them with specified motions. All state data of the module can be achieved from the page ” Module test” (Fig. 3.23) and the PowerCube modules can be moved in position, velocity, current and loop mode. The data collection is activated if the belonging checkbox is enabled. Users can choose each data between actual, minimal and maximal values due to different experiment purposes. With the button “clear MIN/MAX” the maximum and minimum values can be reset. Actual data can be recorded, if a log file were created with the Write log button. Additionally, Home command initializes the original position of PowerCube modules, Reset Command is used to clear all the previous motion, and Halt command can be sent to stop modules during the motion.

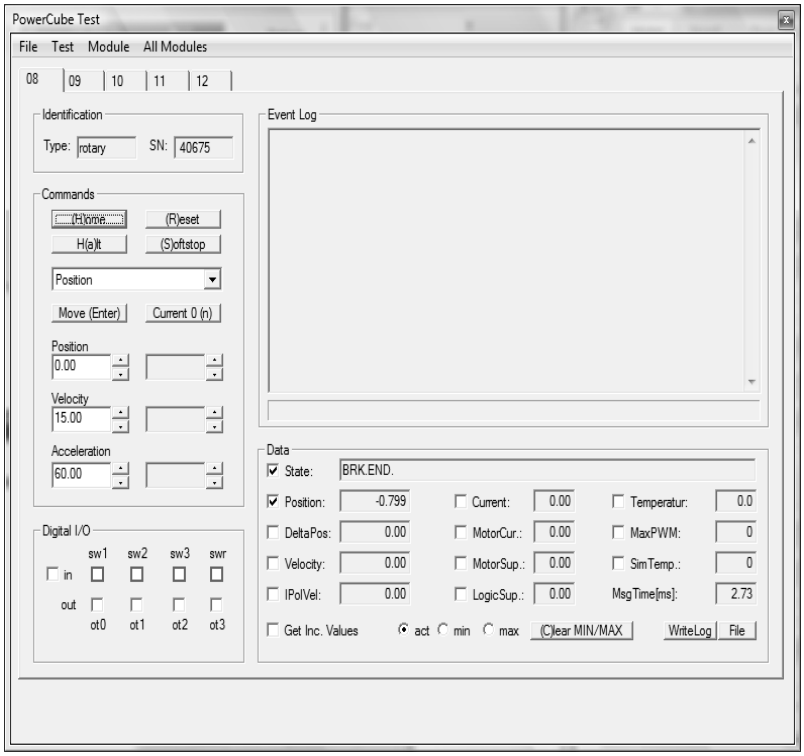


Fig. 3.23 Page of Module Test

3.5 Summary

In this chapter, it mainly discusses about the hardware of SRMR system. PowerCube modules are introduced as the parts of forming the robotic arm. Wire connection is showed via the connect block. Power system is built according to the voltage and current of each module. CAN bus is mainly introduced by comparing with RS232 and Profibus. At last, the hardware including the completed robotic arm is debugged by the demon test software.

Chapter 4

Software of SRMR System Design

For the development of software, the SRMR system consists of a server side and a client side. Based on the requirements, the server side consists of three modules: a connection control module, a local controller module and a data transaction module. The client side can be designed to consist of three modules: a connection control module, a 3D robot model module, and a data transaction module. Fig. 4.1 shows the designed structure for the software of the system. The specific description of each module will be presented in the following sections.

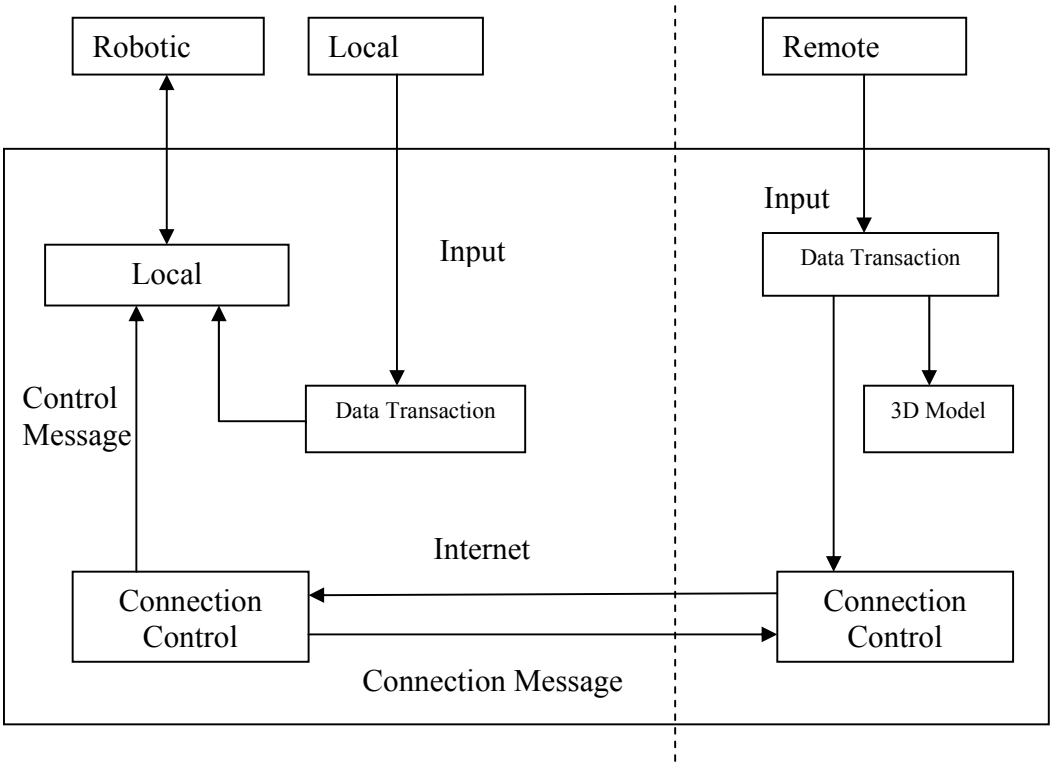


Fig. 4.1 The structure of the Software of SRMR System

4.1 Development of Server Control platform

4.1.1 Server Programming Language

According to the previous chapter, all the hardware parts have been working properly, which are controlled by PowerCube demo software in the SRMR system. However, the demo software is not an open source and only used for hardware examination. Customers cannot develop and continue on researches with limited function by demo version; therefore, it is necessary to deploy a new server control platform to configure the serial robotic arm under developer's commands. QuickStep is one of the programming software introduced by AMTech. QuickStep was designed for beginner programmers as a simple programming tool for PowerCube modules. It enables the user to process motion commands, to use diagnostic functions and to read and set parameters. Furthermore, with an integrated development environment the QuickStep allows the user to generate motion sequences. The function for positioning is provided by the QuickStep programming language as well as mathematical functions, therefore, it is possible to build more or less complex test sequences for robotic arms. The integrated trace-back utility enables the user to record motion parameters like position, velocity, and acceleration. The result can be exported to other windows programs for user analysis later on. The QuickStep state language emulates the way a designer views a machine and designs an automation program. When programming complex machines, designers can use multitasking to split up the program into several independent tasks. The QuickStep flow chart is shown in Fig.

4.2. Up to 84 tasks may be run simultaneously, each operating completely asynchronously, as though a separate controller were executing each program. Tasks may intercommunicate, however, using any of the controller's shared resources: numeric registers, flags, etc, this modular approach to development helps clarify programs and simplifies debugging and maintenance.

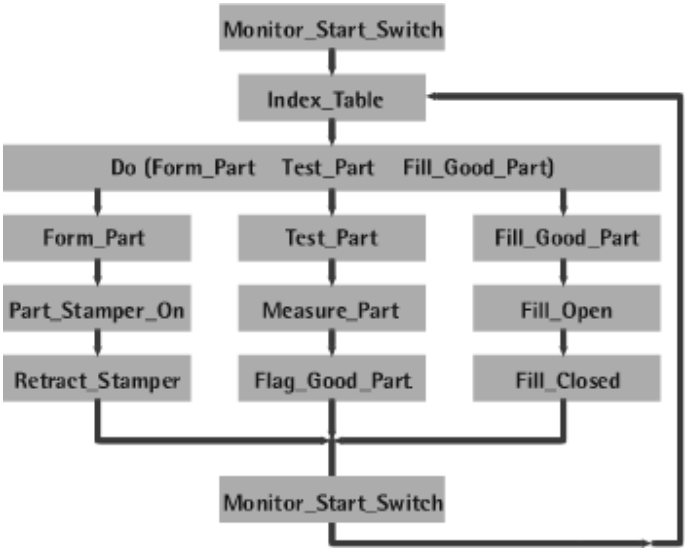


Fig. 4.2 QuickStep Program Flow chats

Compared to other programming language, the QuickStep is relatively elementary. It was not designed as a trajectory control and path planning system for complex robotic applications. Additionally QuickStep has to be started only after power on of all connected modules. This limitation prevents designers from programming independently away from the robotic equipments. After many researches and comparison on Visual C++, Visual Basic.net etc, the server control panel was designed to be programmed with Visual Basic.net since the PowerCube Company provides the programming library for VB.net and also easy to implement the socket communication mechanism.

VB.net is a computer programming system developed and owned by Microsoft. The language not only allows programmers to create simple GUI applications, but can also develop complex applications. There are quite a number of reasons for the enormous success of VB.net:

- The structure of the Basic programming language is very simple, particularly as to the executable code. VB.net is not only a language, but also primarily an integrated, interactive development environment ("IDE"). The VB.net-IDE has been highly optimized to support rapid application development ("RAD"). It is particularly easy to develop graphical user interfaces and to connect them with handler functions which are provided by the application. The graphical user interface of the VB.net-IDE provides intuitively appealing views for the management of the program structure in the large and the various types of entities (classes, modules, procedures, forms…).
- VB.net is a component integration language which matches Microsoft's Component Object Model ("COM") in tune. COM components can be written in different languages and then integrated through VB.net. Interfaces of COM components can be easily called remotely via Distributed COM ("DCOM"), which makes it easy to construct distributed applications. COM components can be embedded in the user interface of your application. Similarly, it can be linked to stored documents (Object Linking and Embedding "OLE", "Compound Documents"). There is a wealth of readily available COM components for many diverse purposes.

4.1.2 Server Control Panel GUI Design

There are three modules on the server side: a connection control module, a local controller module and a data transaction module. The connection control module is mainly used for controlling the connection from client side. It offers the following functions:

- Listening to a connection and disconnection request from client.
- Queuing the connection request in a waiting list when server is busy.
- Removing the cancelled connection request or Releasing the invalid connection.
- Receiving command from client and transmitting to local controller module.

The data transaction module takes the responsibility for validating and executing the robot controlling language program. It allows users to input data and analyze the input. The robot controlling language is written based on VB.net, since M5APIW32.BAS is provided which holds functions and constant declarations for PowerCube modules. The control codes can be divided into four parts: Open / close function, motion operation, and feedback function. Before controlling the robotic arm, the interface has to be open by specifying an InitString Result is a valid deviceID. Here, CAN protocol is used with baud rate 250.

```
PCube_openDevice (dev, "ESD: 0,250")
PCube_closeDevice (dev, "ESD: 0,250")
```

If the hardware has been installed and connected to the server controller properly, then all the PowerCube modules can be found with module ID and serial number. Homing is the first step to start to control the robot. The homing procedure can set the modules to the original position, which is specified by deviceID and moduleID.

```
PCube_homeModule (dev, modId1)
```

The movement of each module is controlled by ramp motion with specification of target position, speed and acceleration. It starts a ramp motion profile of the module specified by deviceID and moduleID. The target position is given in rad resp. m, target speed in rad/s resp. m/s and target acceleration in rad/s^2 resp. m/s^2.

```
PCube_moveRamp(dev, modId1, pos1, vel, acc)
```

An emergency stop function is performed by Halt. It issues a quick stop of the module specified by deviceID and moduleID in order to avoid a dangerous situation.

```
PCube_haltModule(dev, modId1)
```

A reset button is used when users want to clear previous actions. It issues a reset of the module specified by deviceID and moduleID. A reset can clear error flags in the module state. If an error is permanent, reset is ignored.

```
PCube_resetModule(dev, modId1)
```

The feedback function allows users to retrieve data from modules such as Max./Min. speed, Max./Min. acceleration, Max./Min. current, Module state, temperature and so on.

However, not all of them are important to this research, only position and serial number are required. The retrieved position can be compared to the input position to find the absolute error between actual value and theoretical value, or the actual position can be sent back to client side and let remote users view the actual motion of robots in virtual environment. The serial number can be used to verify the related module in order to avoid the mixture.

```
PCube_getPos (dev, modID5, x1)

PCube_getModuleSerialNo(dev, modID5, x2)
```

The local controller module is mainly used for operating the real robotic arm. It provides the following functions:

- Connecting to the real modular robotic arm via CAN bus connection.
- Receiving commands from the local server controller or remote controller
- Executing it on the modular robotic arm via CAN bus connection.

The Fig. 4.3 shows the process flow of server side. When the server side is running, it will be on the statue of awaiting connecting request. Once it receives a connecting request, this process flow will be gone through as shown in the figure.

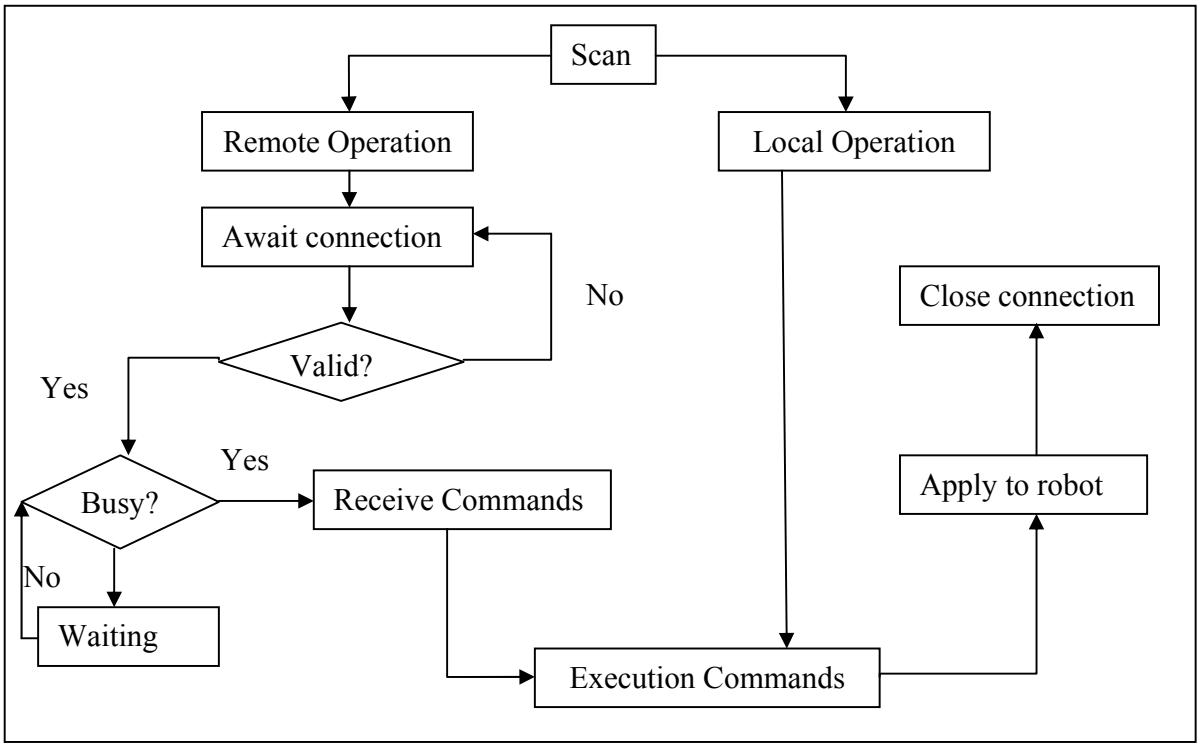


Fig. 4.3 Process flow of server side

The graphic user interface at server side is designed as Fig. 4.4. Area 1 is the Connection module. The scan button allows user to search PowerCube modules via CAN bus. Connect button opens the waiting command and gets ready for remote users to connect. Area 2, 3, & 5 are control module. Area 2 allows users to input rotation angles in radius for each module. Area 3 has all the motion operation to deal with initializing home position, executing input commands, resetting present states, interrupting current motion, and picking/releasing. Those operations have an effect on the whole robotic arm. All the modules can make a movement synchronously in order to finish a required motion completely and efficiency. Area 5 has the same functions as Area 3; however, by clicking buttons in Area 5, users are able to control each module independently. This function could be used for debugging a single module or doing specified experiments. Area 4

shows the feedback data for position of modules. Users can get a straightforward comparison on theoretical and actual values.

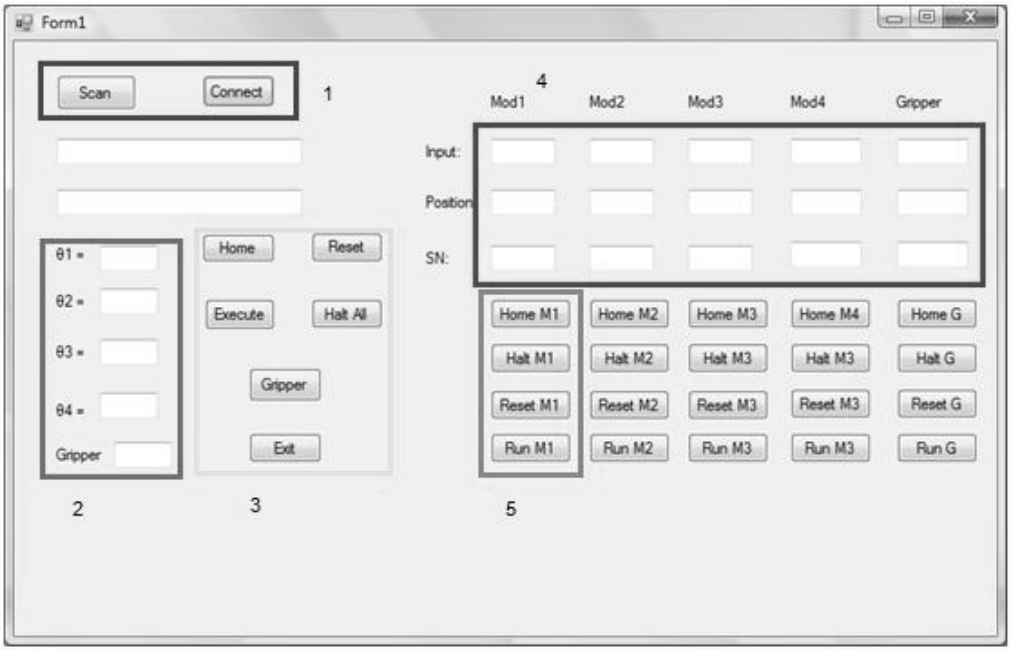


Fig. 4.4 GUI for Server control pannel

In order to apply the command message to the robotic arm, the NTCAN-API always uses the interface made available by the operating system to communicate with the device driver. The NTCAN-API is based on so-call message queues which are also known as First-In-First-Out buffers, which contain sequences of CAN messages. Each handle is assigned to a receive and a transmit FIFO whose size is defined when the handle is opened. In the receive FIFO CAN messages are stored in the chronological order of their reception. By calling a read operation (canRead() or canTake()) one or more CAN messages are copied from the handle FIFO into the address space of the application. By calling a write operation (canWrite() or canSend()) one or more CAN messages are copied from the address space of the application into the transmit FIFO and the CAN messages are transmitted on the CAN bus in their chronological order.

4.2 Development of Client Control platform

4.2.1 Client Programming Language

The main function at the client side is providing a friendly GUI and a 3D model for users to remote control robots. Remote control via Internet has an important practical value to extend traditional use of Internet media from information exchanges to control of physical devices. Internet limitations on communication bandwidth and data exchange rate make robot remote control based on camera images inefficient because of the slow responses made by system to the operator's actions. Using Java3D model of the robot is a good way to provide suitable control conditions for the operator. Java3D based virtual environment in which graphic models of the robot and objects are used to provide real time visualization of the current state of the robot site. Moreover, the use of a 3D virtual environment significantly simplifies the remote performance of operations as it allows changes of viewpoint, zooming, the use of semitransparent images, etc. The use of open technology, Java3D, allows the virtual control environment to run on any type of computer platform [20]. In order to integrate Java3D model to GUI perfectly, the GUI is programmed by Java. Compared to C++ and C, java has thrown out many of complex problems, such as pointers, unions and enumerations. Unlike C and C++, Java is not compiled down to a platform-specific machine language. Instead, Java programs are compiled down to a platform independent language called bytecode. Bytecode is similar

to machine language, but it is not designed to run on any real, physical computer. Instead, bytecode is designed to be run by a program called Java Virtual Machine. JVM is an interpreter that translates Java bytecode into real machine language instructions that are executed on the underlying, physical machine. Java is a single root, single inheritance object-oriented language. It has a built in support for multithreading. Programs written in Java are platform independent. The "write once, run anywhere" feature made Java become the best choice for client programming language.

4.2.2 Java3D Model

Java 3D is an addition to Java for displaying three-dimensional graphics. Programs written in Java 3D can be run on several different types of computer and over the internet. The Java 3D class library provides a simpler interface than most other graphics libraries, but has enough capabilities to produce good games and animation. Java 3D builds on existing technology such as DirectX and OpenGL so the programs do not run as slowly as one might expect. Also, Java 3D can incorporate objects created by 3D modeling packages such as True Space and VRML models. The Java3D application programmer's interface (API) provides a very flexible platform for building a wide range of graphic applications, and is becoming one of the most attractive tools for creating 3D user interfaces, 3D visualizations and virtual environments. The Java3D API is a collection of Java classes providing an interface to the rendering of 3D graphics programs. Java3D is a standard extension to the Java 2 SDK and is made up of the Java3D core classes and the Java3D utility. A large number of applications based on Java3D API libraries have been

developed. Java3D provides not only a new level of dynamics and interactivity but also very good integration with previous Java components [19]. The program structure can be set as Fig. 4.5.

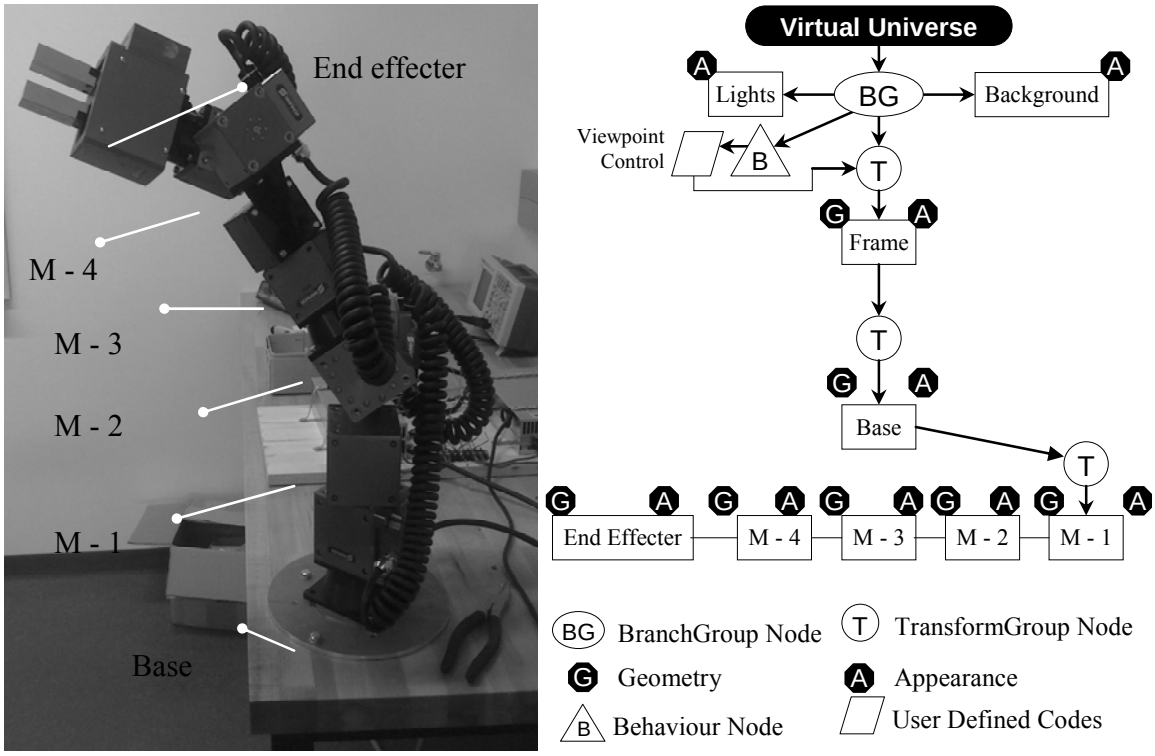


Fig. 4.5 Structure of 3D model programming

The Java3D model of the robotic arm is formed by many cubes, therefore, cubes have been built with the following code and the graph is shown in Fig. 4.6.

```
new Box (.2f, .2f, .2f, app)
```

The dimension of each cube can be set up with length, width, and height. The RBG color can be configured with appearance by mixture of the three primary colors: Red, Green, and Blue. Appearance defines the characteristic of cuboids. It has to be defined in

Appearance class. Each object in 3D space should include geometry property and appearance property, such as color, texture, material, etc.

```
Appearance app = new Appearance ();
Material material = new Material ();
material.setEmissiveColor (new Color3f (1.0f, 0.0f, 0.0f));
app.setMaterial (material);
```

The background bound and color can be set as follows:

```
BoundingSphere bounds =
new BoundingSphere (new Point3d (0.0, 0.0, 0.0), 100.0);
Color3f bicolor = new Color3f (.05f, 0.00f, 0.0f);
```

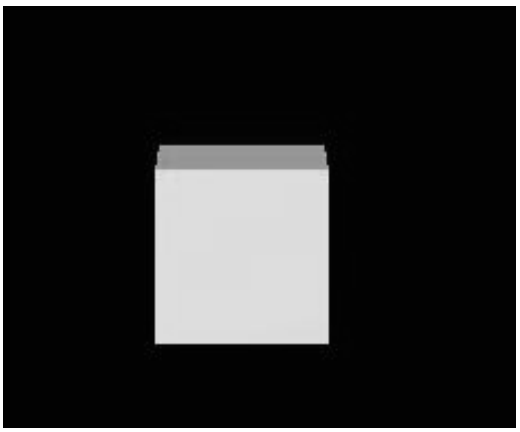


Fig. 4.6 Basic component of Cube

Those cubes can be placed anywhere in the virtual environment according to their coordinates. Vector3f defines a Vector for a three float value tuple. Vector3f can represent any three-dimensional value, such as a vertex, a normal, etc. Utility methods are also included to aid in mathematical calculations.

With calculated position and scale, the 3D model of robotic arm can be created as Fig. 4.7.

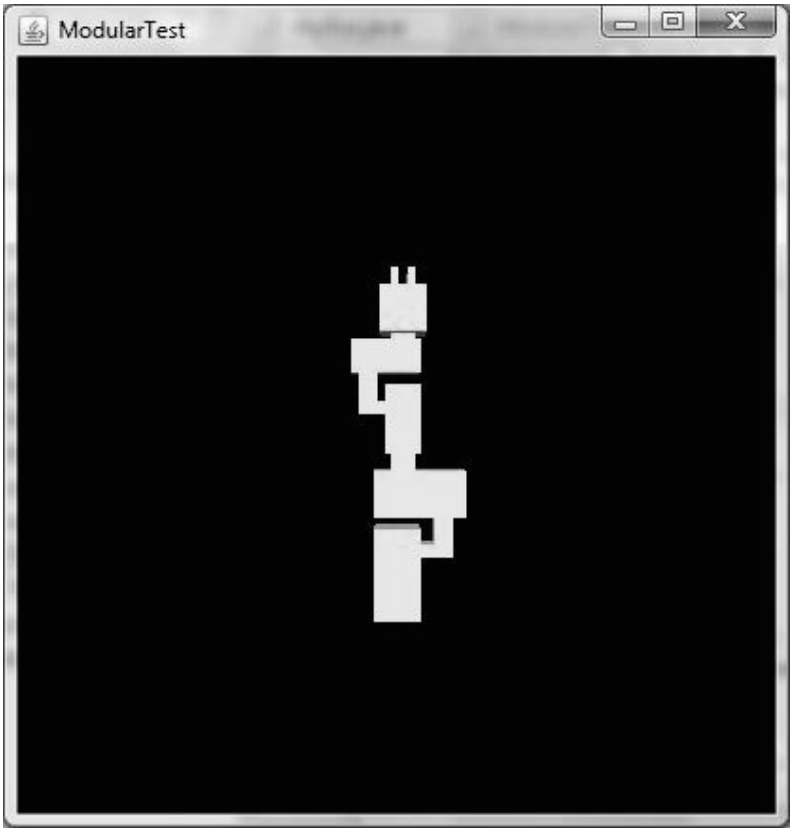


Fig. 4.7 the 3D model of Robotic Arm

One of the biggest challenges of building 3D model is adding the motion to each module. It is very simple to make a single module rotate independently. In Java 3D, there is a group of functions to rotate objects.

rotX (float angle);

rotY (float angle);

rotZ (float angle);

Those methods are based on the viewer point. In this research the viewer point is set along the Z axis, therefore, the coordinate system in Java 3d is shown in Fig. 4.8:

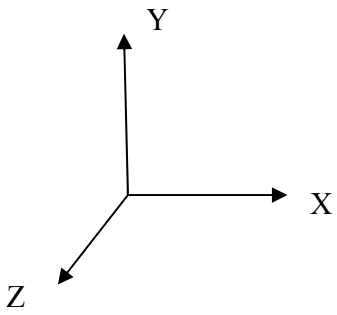


Fig. 4.8 Coordinate system in Java3D

If users want to make an object rotate around X axis, it is not as simple as writing rotX. As users stand at the Z direction, they should make the object rotate towards Z axis by 90 degrees. Then graphically, the object is rotating around the X axis. So the code should be:

```
rotZ (Math.Pi / 2d);
```

Alternatively, if users want to make the object rotate around Y axis, they should set the object turn towards Z axis by 0, pi or 2 pi degrees. The code is as following:

```
rotZ (Math.Pi / 0.5d)
```

In most cases, the module is not rotating around global X, Y, or Z axis directly. Instead, it may have to rotate around the nearby module's local X, Y, or Z axis. So a method called setTranslation is introduced. The 3D coordinate of the new axis has to be determined first, and then this data needs to be transferred into the method to tell the program how to rotate. With this method, a module is able to rotate around any axis in any direction. The following code shows the rotation has been made around the global Y axis then translated to the local Y axis at the 3D coordinate (x1, y1, z1).

```
Axis2.rotZ (Math.PI/.5f);
Axis2.setTranslation (new Vector3f(x1, y1, z1));
```

In order to add the rotation to cube modules, the class rotationInterpolator is used to achieve this goal. This class defines a behavior that modifies the rotational component of its target TransformGroup by linearly interpolating between a pair of specified angles (using the value generated by the specified Alpha object). The interpolated angle is used to generate a rotation transform about the local Y-axis of this interpolator. We can use the following method to construct a new rotational interpolator that varies the target transform node's rotational component.

```
RotationInterpolator (Alpha alpha, TransformGroup target, Transform3D
axisOfTransform, float minimumAngle, float maximumAngle)
```

Alpha is the alpha generator to be used in the rotation computation. Target is the TransformGroup node affected by the interpolator. axisOfTransform is the transform that defines the local coordinate system in which this interpolator operates. The minimunAngle is the starting angle in radians. The maximumAngle is the ending angle in radians. By setting the axisOfTransform to either rotX or rotY, one can achieve the motion rotate around X and Y axis. By setting the minimumAngle and maximumAngle, users can make the object rotate at the angle as she or he wants.

The two-finger gripper does the parallel motion. In Java 3D, users can use PositionInterpolator to achieve this goal. This class defines a behavior that modifies the translational component of its target TransformGroup by linearly interpolating between a pair of specified positions (using the value generated by the specified Alpha object). The interpolated position is used to generate a translation transform along the local X-axis or this interpolator. Users can use the following method to construct a new position interpolator that varies the target TransformGroup's translational component (startPosition and end Position).

PositionInterpolator (Alpha alpha, TransformGroup target, Transform3D axisOfTransform, float startPosition, float endPosition)

Alpha is the alpha object for this interpolator. Target is the transformgroup node affected by this positionInterpolator. axisOfTransform is the transform that defines the local coordinate system in which this interpolator operates. The translation is done along

the X-axis of this local coordinate system. startPosition is where the moving object starts. endPosition is where the moving object stops.

In virtual environment, there is no linked force between each module, therefore, if module A wants to move with the nearby module B then rotate around another module C, module A needs to inherit the motion of module B first which makes it looks like module A is linked to module B. Then a rotation motion will be added to both modules to complete the full motion. In this program, several motion classes were designed and assigned to corresponding modules. The construct of these motion classes is shown as follows:

```
private TransformGroup createObject1(BranchGroup b1, float xpos1, float
ypos1, float zpos1, float x1, float y1, float z1,float r    ){
// add rotation direction
//add rotation speed
//add rotation axis
//add rotation angles
}
```

In the motion class, the first x, y, z coordinates are determined in global system and used to set the location of each module. The following x, y, z coordinates are used to add the motion to the corresponding axis. In the constructor, the rotating directions, speed, angles, and loop times can be specified to each module. The following figures show the

motion of modules in the virtual environment. All modules stay still in Fig. 4.9. The second module starts to rotate around X axis in Fig. 4.10. The second module is rotating around Y axis with the first module in Fig. 4.11. In this case, the two modules look like they are linked together since they have the same motion and speed. The second module is also able to rotate around Y axis with the first module and in the meantime the second module is also able to rotate around its own X axis, which is shown in Fig. 4.12. Finally, by assembling all the modules and adding motions, the completed 3D model can rotate as the real robotic arm, which is shown in Fig. 4. 13.

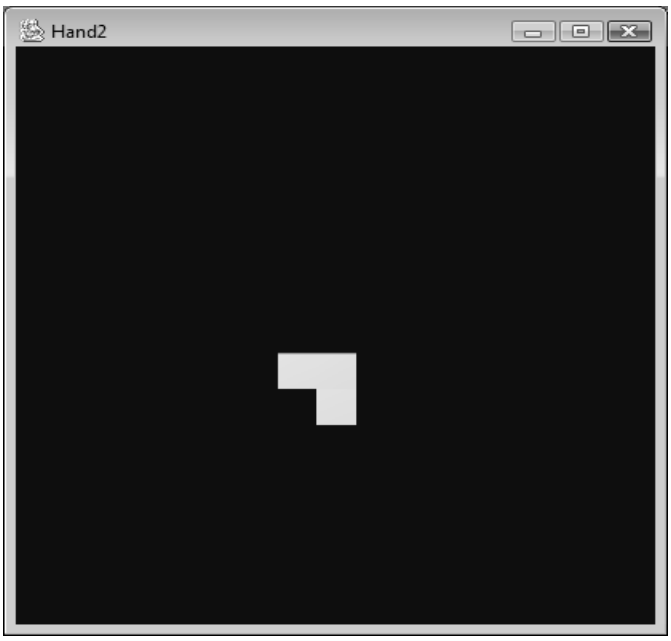


Fig. 4.9 motion of stillness

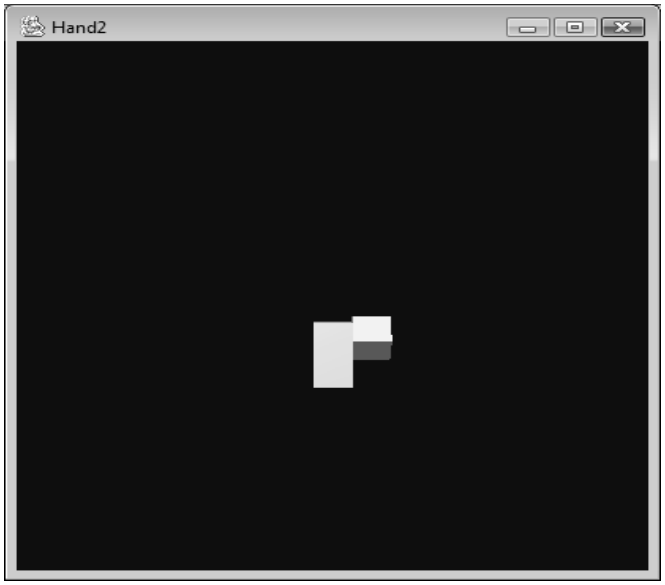


Fig. 4.10 motion of one module

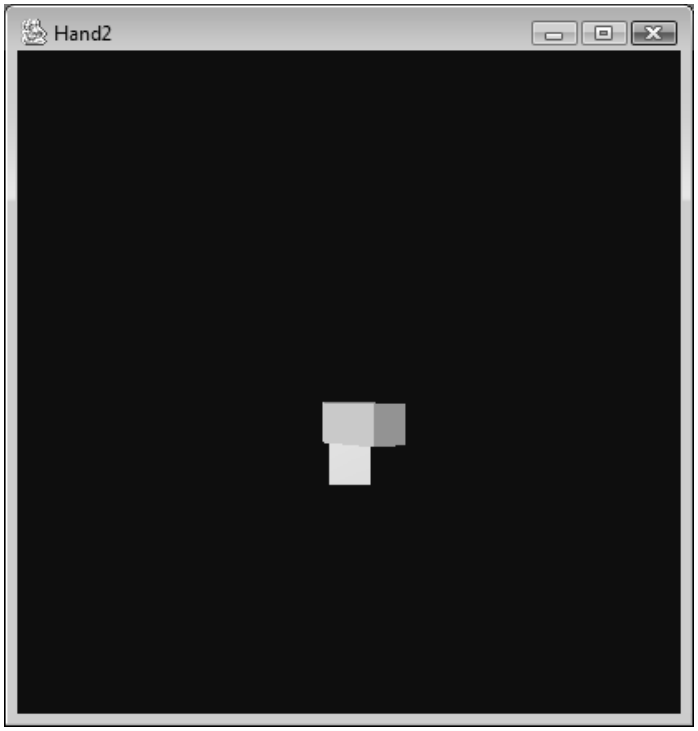


Fig. 4.11 motion of two modules

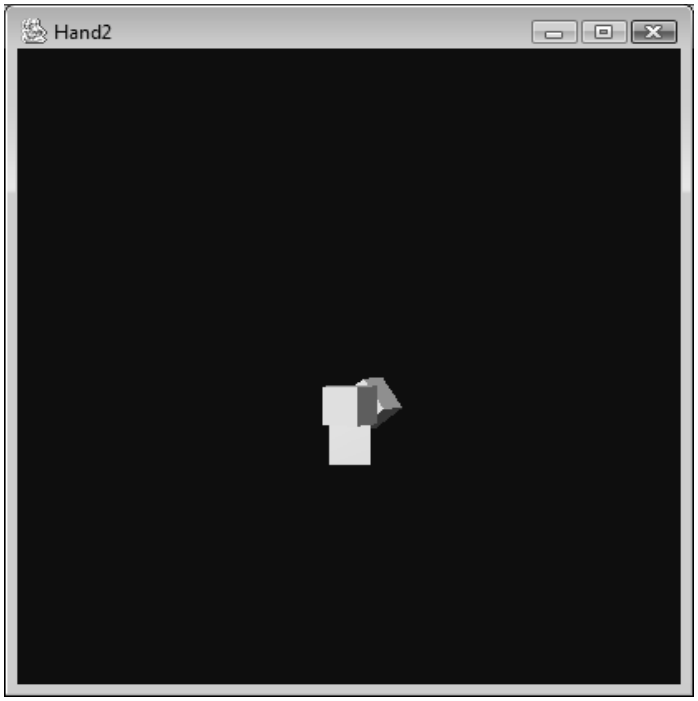


Fig. 4.12 motion of two modules with their own direction

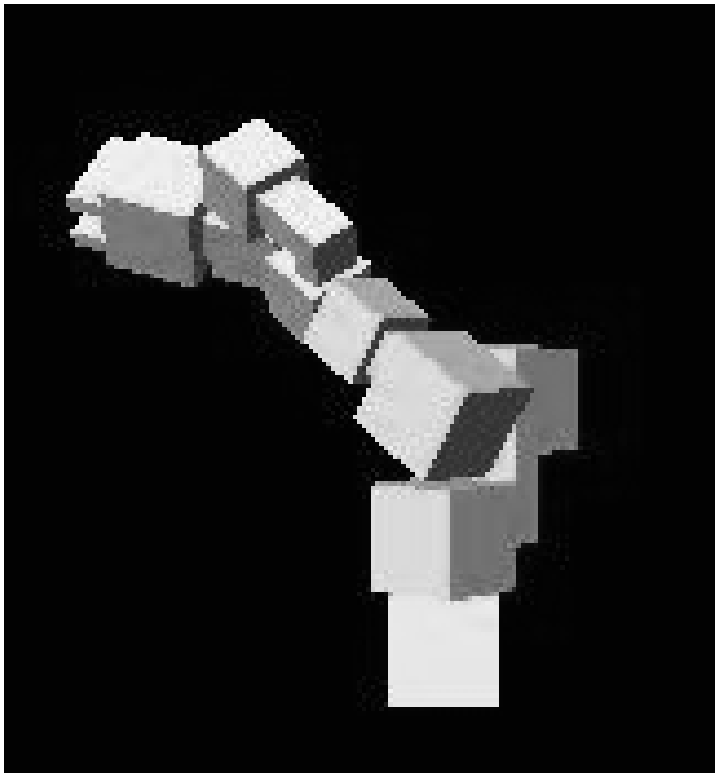


Fig. 4.13 motion of 3D robotic arm model

4.2.3 Client Control Panel GUI Design

As mentioned in the beginning of this chapter, the client side contains three modules: a connection control module, a 3D robot model module, and a data transaction module. The connection control module is mainly used for control the connection to server side. It offers the following functions:

- Sending connection request to server.
- Sending disconnection request to server.
- Keeping the connection.
- Sending users inputted program to server.

The 3D model module provides a view of 3D virtual robot model and offers the following functions:

- Displaying the 3D model of robotic arm in the virtual environment.
- Receiving the operation message from the program editor.
- Operating the motion according to the received control message.

The data transaction module takes the responsibility for validating and executing the robot-controlling language program. It has the following functions:

- Allowing users to input data and calculate the inverse kinematics.
- Checking error of the input.
- Applying the valid operation code to the 3D robotic arm model and simulating at the client side.
- Transferring the operation message to server side.

Based on the above modules, people can enter the data by choosing rotational angles or position coordinate. With pressing calculate button, the results can be calculated by vector algebraic method. Users can simulate the results with 3D model in virtual environment. In the meantime, the results will be sent to server side to control the real robotic arm. The process flow chart of client side is shown in Fig. 4.14.

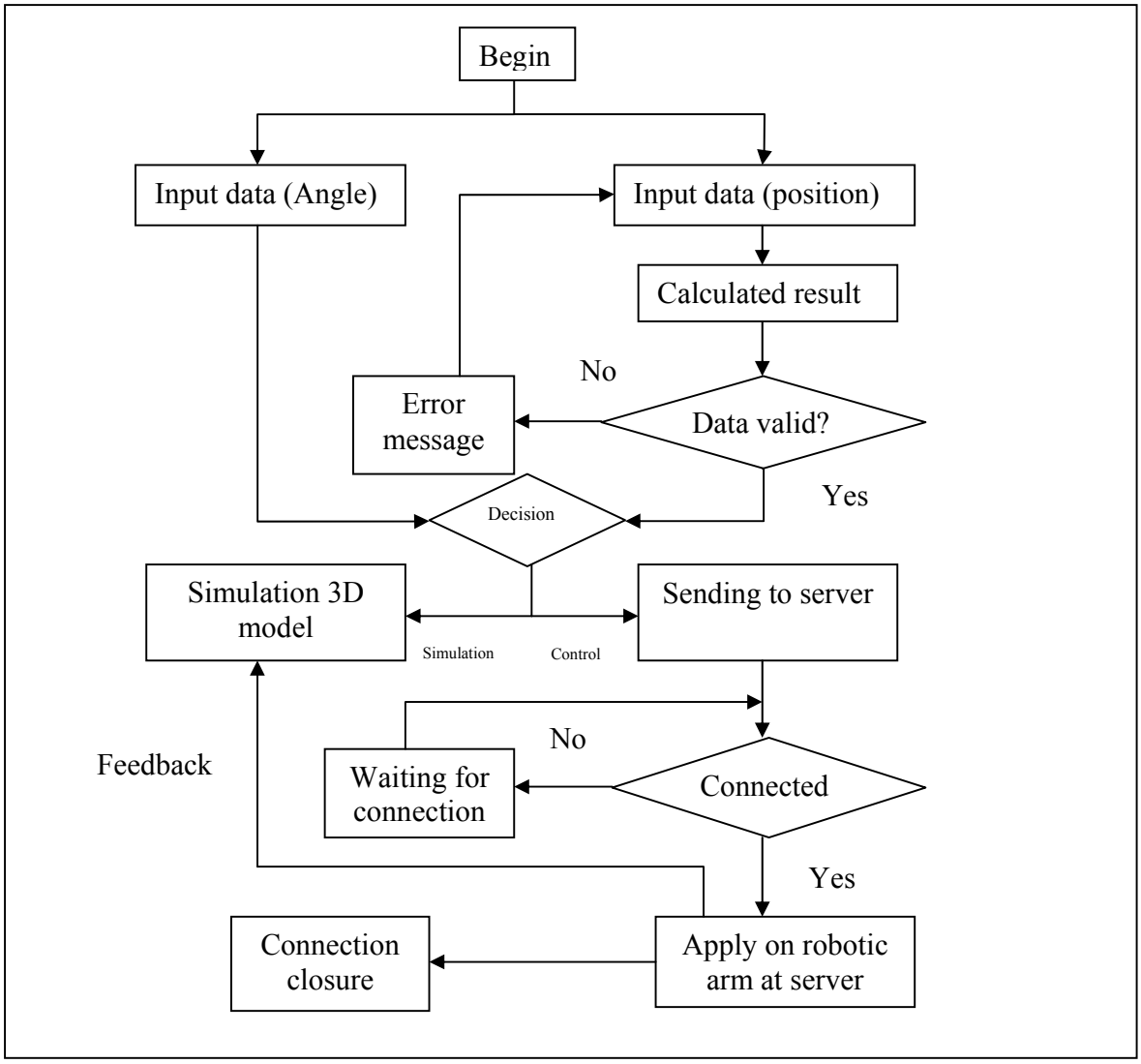


Fig. 4.14 Process flow of client side

The graphic user interface of client side should include an operation window and a 3D robot model window. In the corresponding textbox, users can input the rotation angle directly or input the position coordinate and solved by inverse kinematic by clicking the calculate button. Once users retrieve the rotational angle for each module, they can simulate on the Java3D model by clicking the execute button and/or send the data to server side to control the robotic arm by clicking the send button. Reset button can be used to clear all the current status. The action button controls the 2 finger gripper directly and instructs it to perform to perform picking, holding and releasing actions. The Exit button allows users to close the connection and exit the program safely. The real graphic user interface at client side is designed as Fig. 4.15.

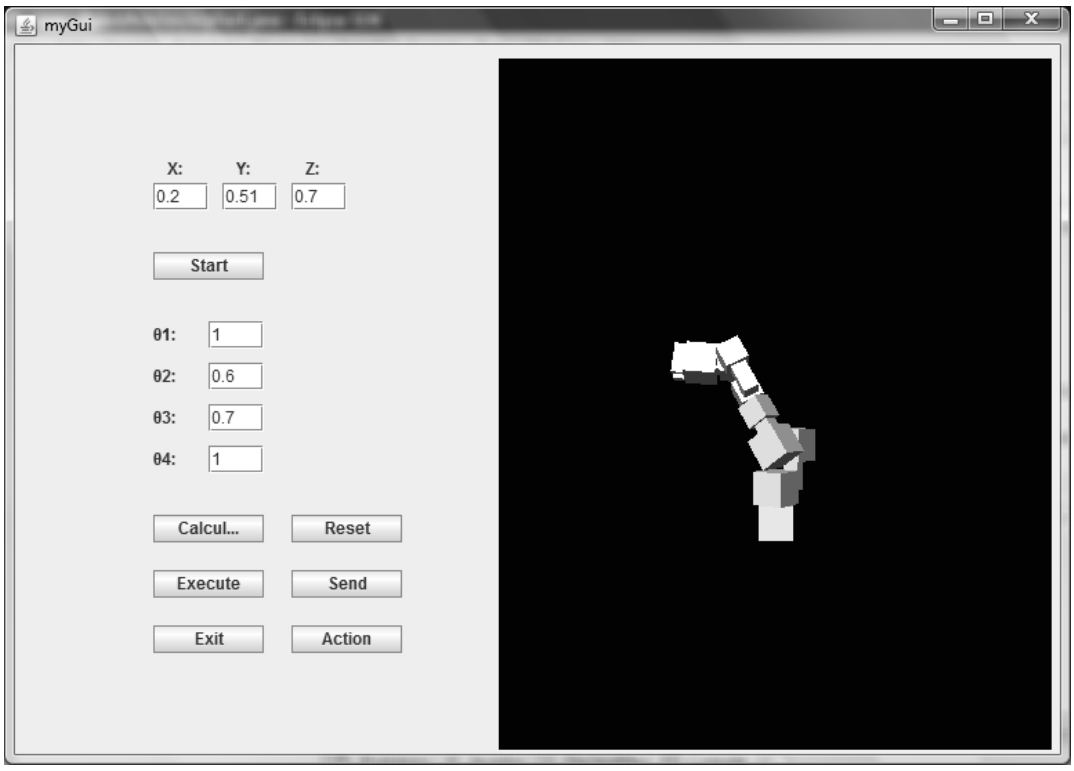


Fig. 4.15 GUI at client side

4.3 Development of Communication via Internet

There are several communication protocols in the world such as User Datagram Protocol, Transmission Control Protocol etc. UDP uses a simple transmission model without hand-shaking dialogues; therefore it cannot guarantee reliability, ordering, or data integrity. Thus, UDP provides an unreliable service and datagram may appear duplicated, arrive out of order, or go missing without notice. Time-sensitive applications often use UDP. The reason is that dropping packets is preferable to using delayed packets. UDP applications use datagram sockets to create host-to-host communications. Sockets bind the application to service ports and perform as the endpoints of data transmission. TCP operates at a higher level. It is concerned only with the two end systems such as a Web browser and a Web server. In particular, unlike UDP, TCP provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer. Among its other management tasks, TCP controls message size, the rate at which messages are exchanged, and network traffic congestion. Therefore, in this project, the communication is set up by the TCP. TCPSocket provides a socket interface, using an Orbited daemon to proxy TCP streams to the web browser. In computer networking, an internet socket is the endpoint of bidirectional communication flow across an internet protocol based computer network. Internet sockets are an application programming interface (API) in an operating system, used for inter-process communication. Internet sockets constitute a mechanism for delivering incoming data packets to the appropriate application process or thread. They are beads on a combination

of local and remote IP address and port numbers. Each socket is mapped by the operational system to a communicating application process or thread.

4.3.1 Communication at Server

At the server side, the communication program is written by VB.net. A socket address combines an IP address (the computer's location) and a port (the type of a communication service) into a single Identity. The following code at the server side opens the address of server and selects a port, so that the client can hook on it.

```
Const portNumber As Integer = 8000
Dim localAddr As IPAddress =IPAddress.Parse ("10.129.13.66")
```

Now the server is waiting on the client's message. The client is able to send message to server. However, the data is not usable right away, since all the data is sent in string type, so the following code is created for converting data types.

```
Dim clientdata As String = Encoding.ASCII.GetString (bytes)
DataArray (i) = Val (clientdata)
```

As soon as the server gets data from the client, it will convert the data to double type which is required for this program. It will restore the converted data into an Array automatically for the future usage. In the meanwhile, the server returns a signal back to

client as a response, so that the client will understand the previous message arrived successfully and continue to send next message until the client finishes sending the entire message.

4.3.2 Communication at Client

At the client side, the communication program is written by Java. The following code creates a stream socket and connects it to the specified port number at the specified IP address.

```
Socket (InetAddress Address, int port)
```

A PrintStream adds functionality to another output stream, namely the ability to print representations of various data values conveniently. Two other features are provided as well. Unlike other output streams, a PrintStream never throws an IOException; instead, exceptional situations merely set an internal flag that can be tested via the checkError method. Optionally, a PrintStream can be created so as to flush automatically; this means that the flush method is automatically invoked after a byte array is written, one of the println methods is invoked, or a newline character or byte ('\n') is written. All characters printed by a PrintStream are converted into bytes using the platform's default character encoding. The following code sends command to the server side.

```
new PrintStream (socket.getOutputStream ())
```

In order to receive response from the server, the following code is used:

```
new BufferedReader (new InputStreamReader (socket.getInputStream ()))
```

The bufferedReader class reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines. The buffer size may be specified, or the default size may be used. The default is large enough for most purposes. In general, each read request made of a Reader causes a corresponding read request to be made of the underlying character or byte stream. It is therefore advisable to wrap a BufferedReader around any Reader whose read () operations may be costly, such as FileReaders and InputStreamReaders.

4.4 Summary

In this chapter, the structure of software for SRMR system is described at the beginning. The server side is programmed by VB.net. GUI at the server side is created with various predefined functions supplied by PowerCube. The Client side is programmed by JAVA. A 3D model is created by JAVA3D for simulation, which performs the exactly same motion as the real robotic arm. The GUI at the client side allows users to control robot remotely and do the simulation on/off line. The communication between the server and the client is set up by socket.

Chapter 5:

Case Study

5.1 Modular Robot Kinematics

5.1.1 Geometric Structure

Although a reconfigurable modular serial manipulator can be created by various ways, not all of configurations are eligible or worth to produce in fact. Lacking of design feasibility eliminates many configurations. Unpredictable workspace is another reason for failure. The configuration in Fig. 5.1 is eliminated because of lacking design merit. This configuration does not utilize the manipulators modularity and reconfigurable deign to the fullest extent. The extra joint in the configuration does not improve the structure.

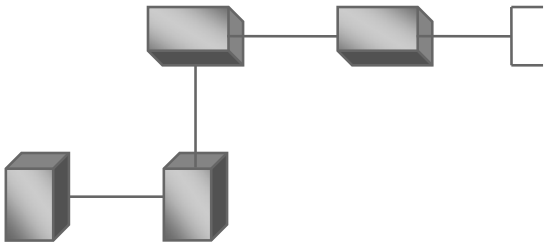


Fig. 5.1 Failure design 1

Another configuration is eliminated shown in Fig. 5.2. Although it used a different configuration of joint modules and links, it reaches the same workspace as the

other configuration. There is no reason to create different configurations with the same functions, which may cause unnecessary confusion.

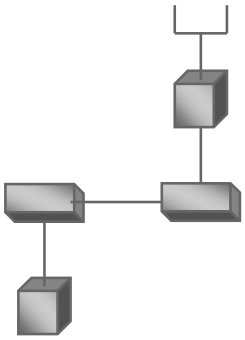
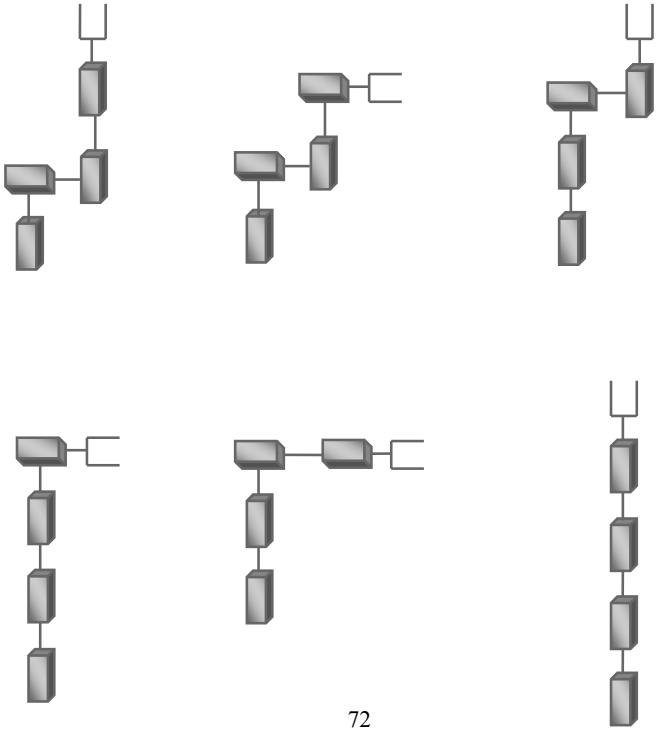


Fig. 5.2 Failure design 2

After many evaluations, nine configurations are selected and shown in Fig. 5.3, because each configuration is able to reach different workspaces. Each block represents a module with revolute joint. The straight lines represent links connecting these joint modules. The-end effector is hooked at the end of last module. They can be assembled with four joint modules and links.



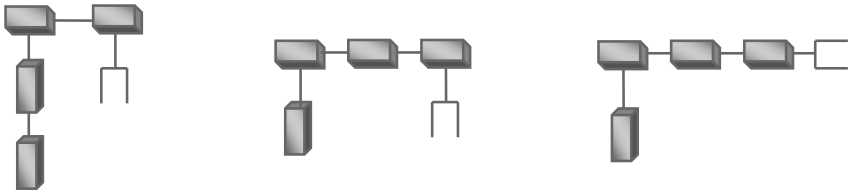


Fig. 5.3 nine unique configurations of the 4 DOF modular robot

In this research, one configuration is selected to study in detail. According to the configuration of the real robotic arm, the geometric structure can be sketched in Fig. 5.4.

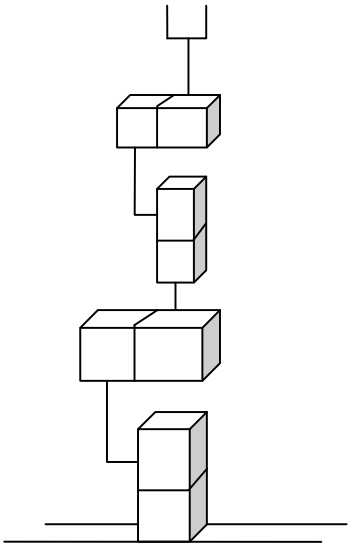


Fig. 5.4 Geometric structure of Serial Modular Robot

The Degree of Freedom for the robot is examined with the Chebychev-Grubler-Kutzbach's criterion:

$$M = d\,(n - g - 1) + \sum_{i=1}^{g} f_i$$

In the expression, the M stands for the system mechanism. The d is the order of the system. For planar motion, d = 3, and for spatial motion d = 6. The n is the number of

links including the frames. The g stands for the number of joints. Finally, f_i is the number of DOFs for i_{th} joint.

In this research, all the joints are using revolute joint with 1 DOF. There are 4 joints and 5 links. The motion belongs to planar motion. Therefore, the DOF of this robot can be calculated as follows:

$$M = d\,(n - g - 1) + \sum_{i=1}^{g} f_i$$

$$M = 3\,(5 - 4 - 1) + 1 + 1 + 1 + 1$$

$$= 4$$

Therefore, the calculation result shows that the modular robotic arm has four degrees of freedom, which agrees with the design result in realism.

5.1.2 Forward Kinematics Analysis

Robot kinematics, which classified as forward kinematics and inverse kinematics, is one of the fundamental issues in robotics. It studies the motion of robots for programming, control, and design purposes. Many effective kinematic modeling methods have been developed for conventional robots such as the Denavit-Hartenberg method [21], the screw coordinates method [22], and the zero reference position method [23]. All of them are based on the geometric relationships of links and joints. Among them, the D-H representation is one significant milestone in robot kinematics. With D-H notation, links are numbered 0, 1, …, n, the i_{th} pair being defined as that coupling the (i - 1)st link with

the i_{th} link, where link 0 is the base, and the link n is the end effector. Every link gets its own coordinate system with origin O_i and axis X_i, Y_i,and Z_i. In its frame, Z_i is the axis of the i_{th} pair and represents the direction of the joint axis. X_i is defined as the common perpendicular to Z_{i-1} and Z_i, directed from the former to the latter. The distance between Z_i and Z_{i+1} is defined as a_i, which is nonnegative. The Z_i coordinate of the intersection of Z_i with X_{i+1} is denoted by b_i. Since this quantity is a coordinate, it can be either positive or negative. The absolute value of b_i is the distance between X_i and X_{i+1}. The angle between Z_i and Z_{i+1} is defined as α_i and is measured about the positive direction of X_{i+1}. It is also called twist angle between successive pair axes. The angle between X_i and X_{i+1} is defined as θ_i and is measured about the positive direction of Z_i.

In order to find the D-H table, the geometric model of robotic arm can be analyzed as Fig. 5.5

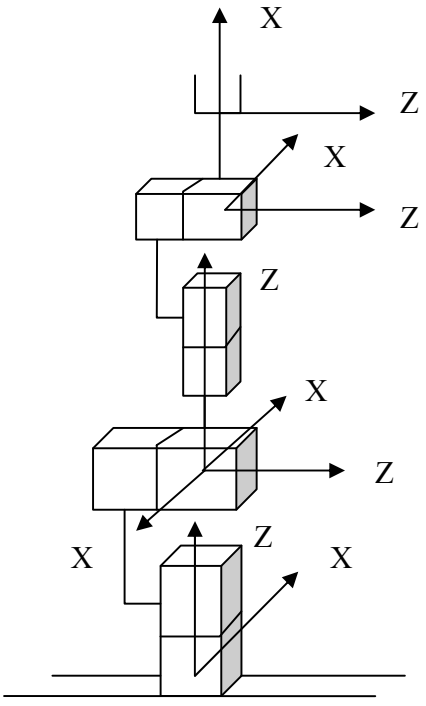


Fig. 5.5 Analysis of 4DOF Modular Robot

Therefore, the D-H table can be setup as follows, where L_i is the distance between each joint:

	a_i	b_i	α_i	θ_i
1	0	L_1	90^o	θ_1
2	0	0	90^o	θ_2
3	0	$L_3 + L_4$	90^o	θ_3
4	L_5	0	0^o	θ_4

Table 5.1 D-H Table

Forward kinematics is computation of the position and orientation of robot's end effector as a function of its joint angles. It is widely used in robotics, computer games, and animation. With the help of D-H table, the rotation matrices and vectors can be written as follows:

$$Q_i = \begin{bmatrix} \cos\theta_1 & -\cos\alpha_1\sin\theta_1 & \sin\alpha_1\sin\theta_1 \\ \sin\theta_1 & \cos\alpha_1\cos\theta_1 & -\sin\alpha_1\cos\theta_1 \\ 0 & \sin\alpha_1 & \cos\alpha_1 \end{bmatrix} \quad (5.1)$$

$$Q_2 = \begin{bmatrix} \cos\theta_2 & -\cos\alpha_2\sin\theta_2 & \sin\alpha_2\sin\theta_2 \\ \sin\theta_2 & \cos\alpha_2\cos\theta_2 & -\sin\alpha_2\cos\theta_2 \\ 0 & \sin\alpha_2 & \cos\alpha_2 \end{bmatrix} \quad (5.2)$$

$$Q_3 = \begin{bmatrix} \cos\theta_3 & -\cos\alpha_3\sin\theta_3 & \sin\alpha_3\sin\theta_3 \\ \sin\theta_3 & \cos\alpha_3\cos\theta_3 & -\sin\alpha_3\cos\theta_3 \\ 0 & \sin\alpha_3 & \cos\alpha_3 \end{bmatrix} \quad (5.3)$$

$$Q_4 = \begin{bmatrix} \cos\theta_4 & -\cos\alpha_4\sin\theta_4 & \sin\alpha_4\sin\theta_4 \\ \sin\theta_4 & \cos\alpha_4\cos\theta_4 & -\sin\alpha_4\cos\theta_4 \\ 0 & \sin\alpha_4 & \cos\alpha_4 \end{bmatrix} \quad (5.4)$$

$$a_1 = \begin{bmatrix} a_1\cos\theta_1 \\ a_1\sin\theta_1 \\ b_1 \end{bmatrix} \tag{5.5}$$

$$a_2 = \begin{bmatrix} a_2\cos\theta_2 \\ a_2\sin\theta_2 \\ b_2 \end{bmatrix} \tag{5.6}$$

$$a_3 = \begin{bmatrix} a_3\cos\theta_3 \\ a_3\sin\theta_3 \\ b_3 \end{bmatrix} \tag{5.7}$$

$$a_4 = \begin{bmatrix} a_4\cos\theta_4 \\ a_4\sin\theta_4 \\ b_4 \end{bmatrix} \tag{5.8}$$

The position vector P of the operation point of end effector can be calculated as follows:

$$P = a_1 + Q_1a_2 + Q_1Q_2a_3 + Q_1Q_2Q_3a_4$$

$$P = \begin{bmatrix} (\cos\theta_1\sin\theta_2 + (\cos\theta_1\cos\theta_2 - \sin\theta_1)\sin\theta_3)L_5\cos\theta_4 \\ (\sin\theta_1\sin\theta_2 + (\sin\theta_1\cos\theta_2 - \cos\theta_1)\sin\theta_3)L_5\cos\theta_4 \\ L_1 - \cos\theta_2(L_3 + L_4) + \sin\theta_2\cos\theta_3\cos\theta_4L_5 - \cos\theta_2\sin\theta_4L_5 \end{bmatrix} \tag{5.9}$$

5.1.3 Inverse Kinematics Analysis base on VA Method

Inverse kinematics is the process of determining the parameters of a jointed flexible object in order to achieve a desired pose. It is a type of motion planning. The purpose of an inverse kinematics algorithm is to determine the joint angles that cause a robot to reach a desired pose of the end effector. Because a modular robot has unfixed assembly configurations and the number of DOFs in the robot varies, it is hard to find a closed-form inverse kinematics solution. Solving the inverse kinematics problem for serial robots is a difficult task. The complexity in the solution arises from the nonlinear equations occurring during transformation between joint and Cartesian spaces. The common approach for solving the IK problem is using the D-H matrix method; the inverse kinematics is analyzed by performing a series of matrix transformation and equating the corresponding matrix elements by Manseur and Doty [24]. According to the past research, the matrix method will cause much unavoidable extraneous work since less than half equations are useful. Therefore, the vector algebraic method [25] is deployed in this thesis, which is free of the drawback of the matrix method. Before doing the inverse kinematics analysis, a number of vector formulas have to be described first. In Fig. 5.6, unit vector $\vec{w}$ is perpendicular to both unit vector $\vec{u}$ and $\vec{v}$, the angle θ is the right-hand rotation angle from $\vec{u}$ to $\vec{v}$. This rotation can be represented by the vector formulas as follows:

$$\vec{u} \times \vec{v} = \vec{w} \sin\theta \tag{5.10}$$

$$\vec{v} = \cos\theta\vec{u} + \sin\theta\vec{w} \times \vec{u} \tag{5.11}$$

$$\vec{u} = \cos\theta\vec{v} - \sin\theta\vec{w} \times \vec{v} \tag{5.12}$$

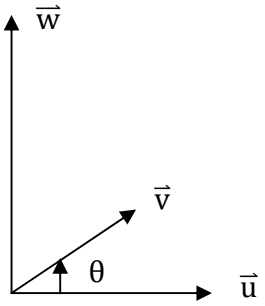


Fig. 5.6 Vectors Representations

The general form of the 4 DOFs serial robot is shown in Fig. 5.7. $\overrightarrow{x_i}$ and $\overrightarrow{z_i}$ are unit vectors. The offset is expressed by b_i. I is defined as the input vector of the robot loop. J is the output vector of the robot loop. F is the floating vector. The initial state has to be setup before solve the IK problem. $\overrightarrow{z_1}, \overrightarrow{z_5}, \overrightarrow{x_5}$ are unit vectors, which can be assigned according to the local coordinate system, $\theta_5, R, \rho_i, b_i, \alpha_i (i = 1 - 4)$ are physical data, which can be measured or obtained from DH table. Therefore, it is possible to find the joint angles: $\theta_1, \theta_2, \theta_3, \theta_4$ with the initial conditions.

By the vector calculation, the vector $\vec{I}$ $\vec{J}$ and $\vec{F}$ can be determined from

$$\vec{J} = \rho_3 \overrightarrow{x_3} - b_3 \overrightarrow{z_3} \tag{5.13}$$

$$\vec{I} = -b_4 \overrightarrow{z_4} + \rho_4 \overrightarrow{x_4} + b_5 \overrightarrow{z_5} - \vec{R} \tag{5.14}$$

$$\vec{F} = -\rho_1 \overrightarrow{x_1} + b_2 \overrightarrow{z_2} - \rho_2 \overrightarrow{x_2} \tag{5.15}$$

$$\overrightarrow{z_3} = \cos\alpha_4 \overrightarrow{z_4} - \sin\alpha_4 \overrightarrow{x_3} \times \overrightarrow{z_4} \tag{5.16}$$

$$\overrightarrow{x_3} = \cos\theta_4 \overrightarrow{x_4} - \sin\theta_4 \overrightarrow{z_4} \times \overrightarrow{x_4} \tag{5.17}$$

$$\overrightarrow{z_4} = \cos\alpha_5 \overrightarrow{z_5} - \sin\alpha_5 \overrightarrow{x_4} \times \overrightarrow{z_5} \tag{5.18}$$

$$\overrightarrow{x_4} = \cos\theta_5 \overrightarrow{x_5} - \sin\theta_5 \overrightarrow{z_5} \times \overrightarrow{x_5} \tag{5.19}$$

The vector loop equation formed by the dash line can be written as follows:

$$\vec{I} + \vec{J} = \vec{F} \tag{5.20}$$

Substituting (5.13) and (5.15) into (5.20):

$$\vec{I} + \rho_3\overrightarrow{x_3} - b_3\overrightarrow{z_3} = -\rho_1\overrightarrow{x_1} + b_2\overrightarrow{z_2} - \rho_2\overrightarrow{x_2} \tag{5.21}$$

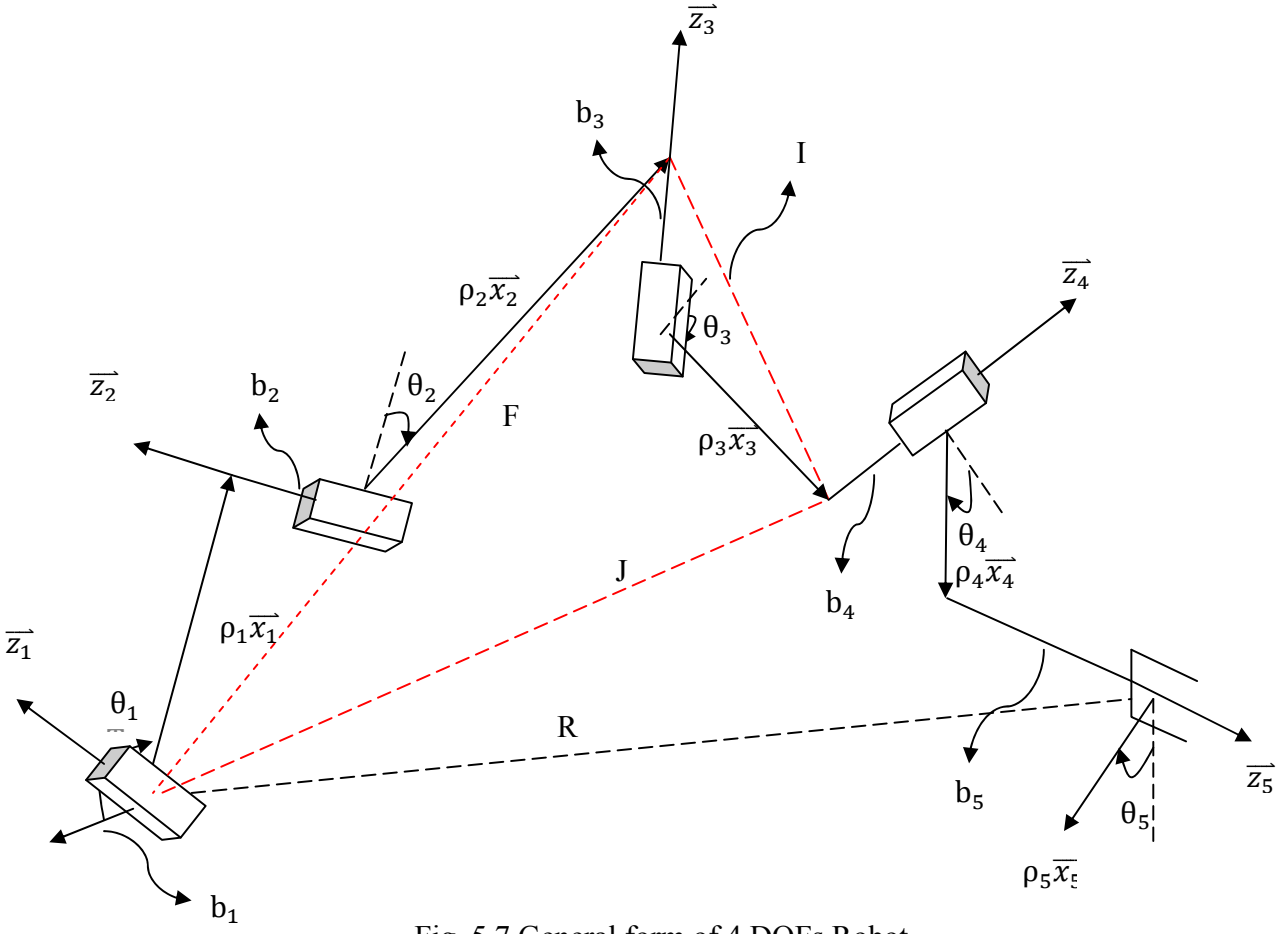


Fig. 5.7 General form of 4 DOFs Robot

In order to solve this kind of mechanism, the basic mathematical operations are: 1) dot product $\overrightarrow{z_1}$ and $\overrightarrow{z_3}$ with both sides of the vector loop equation; 2) square both sides of the vector loop equation; 3) evaluate $\overrightarrow{z_3} \cdot \overrightarrow{z_3}$ from the loop equation. Dot product $\overrightarrow{z_1}$ with both sides of (5.21) to get:

$$\overrightarrow{z_1} \cdot \vec{I} + \left(\rho_3\overrightarrow{z_1}\right) \cdot \overrightarrow{x_3} - \left(b_3\overrightarrow{z_1}\right) \cdot \overrightarrow{z_3} = b_2\cos\alpha_1 - \rho_2\sin\alpha_1\sin\theta_2 \tag{5.22}$$

Dot product $\overrightarrow{z_3}$ with both sides of (5.21) to get:

$$\vec{I} \cdot \overrightarrow{z_3} - (b_3) = b_2\cos\alpha_2 - \rho_1\sin\alpha_2\sin\theta_2 \tag{5.23}$$

Eliminating θ_2 from (5.22) and (5.23)

$$\left(\rho_1\rho_3\sin\alpha_2\overrightarrow{z_1}\right)\cdot\overrightarrow{x_3}-\left(\rho_1 b_3\sin\alpha_2\overrightarrow{z_1}+\rho_2\sin\alpha_1\vec{I}\right)\cdot\overrightarrow{z_3}$$

$$=\ \rho_1\sin\alpha_2\left(b_2\cos\alpha_1-\overrightarrow{z_1}\cdot\vec{I}\right)-\rho_2\sin\alpha_1(b_3+\mathrm{b}_2\cos\alpha_2) \tag{5.24}$$

Substituting (5.16) into (5.24):

$$U\cdot\overrightarrow{x_3}=V \tag{5.25}$$

Where:

$$U=\ \rho_1\rho_3\sin\alpha_2\overrightarrow{z_1}+\sin\alpha_2\overrightarrow{z_4}\times(\rho_1 b_3\sin\alpha_2\overrightarrow{z_1}+\rho_2\sin\alpha_1\vec{I}) \tag{5.25a}$$

$$V=\rho_1\sin\alpha_2\left(b_2\cos\alpha_1-\overrightarrow{z_1}\cdot\vec{I}\right)-\rho_2\sin\alpha_1(b_3+\mathrm{b}_2\cos\alpha_2)$$

$$+\cos\alpha_3\overrightarrow{z_4}(\rho_1 b_3\sin\alpha_2\overrightarrow{z_1}+\rho_2\sin\alpha_1\vec{I}) \tag{5.25b}$$

Substituting (5.17) into (5.25):

$$A\cos\theta_4+\mathrm{B}\sin\theta_4=\mathrm{C} \tag{5.26}$$

Where:

$$A=U\cdot\overrightarrow{z_4} \tag{5.26a}$$

$$B=U\cdot\overrightarrow{x_4}\times\overrightarrow{z_1} \tag{5.26b}$$

$$C=\ V \tag{5.26c}$$

Equating $\overrightarrow{z_1}\cdot\overrightarrow{z_3}$ to get:

$$\overrightarrow{z_1}\cdot\overrightarrow{z_3}=\cos\alpha_1\cos\alpha_2-\sin\alpha_1\sin\alpha_2\cos\theta_2 \tag{5.27}$$

Squaring both sides of (5.21):

$$\left(\vec{I}^2+{\rho_3}^2+{b_3}^2\right)+\left(2\rho_3\vec{I}\right)\cdot\overrightarrow{x_3}-\left(2b_3\vec{I}\right)\cdot\overrightarrow{z_3}=({\rho_1}^2+{\rho_2}^2+{b_2}^2+2\rho_1\rho_2 cos\theta_2) \tag{5.28}$$

Eliminating θ_2 from (5.27) and (5.28):

$$\left(2\rho_3 sin\alpha_1 sin\alpha_2\vec{I}\right)\cdot\overrightarrow{x_3}+2\left(\rho_1\rho_2\overrightarrow{z_1}-b_3\sin\alpha_1\sin\alpha_2\vec{I}\right)\cdot\overrightarrow{z_3}$$

$$=2\rho_1\rho_2\cos\alpha_1\cos\alpha_2-\sin\alpha_1\sin\alpha_2(\vec{I}^2+{\rho_3}^2+{b_3}^2-{\rho_1}^2-{\rho_2}^2-{b_2}^2) \tag{5.29}$$

Substituting (5.16) into (5.29):

$$U' \cdot \overrightarrow{x_3} = V' \tag{5.30}$$

Where:

$$U' = 2\rho_3 sin\alpha_1 sin\alpha_2 \vec{I} - 2sin\alpha_3 \overrightarrow{z_4} \times (\rho_1 \rho_2 \overrightarrow{z_1} - b_3 \sin\alpha_1 \sin\alpha_2 \vec{I}) \tag{5.30a}$$

$$V' = 2\rho_1 \rho_2 \cos\alpha_1 \cos\alpha_2 - 2\cos\alpha_3 \overrightarrow{z_4} \cdot \left(\rho_1 \rho_2 \overrightarrow{z_1} - b_3 \sin\alpha_1 \sin\alpha_2 \vec{I}\right)$$

$$-\sin\alpha_1 \sin\alpha_2 (\vec{I}^2 + {\rho_3}^2 + {b_3}^2 - {\rho_1}^2 - {\rho_2}^2 - {b_2}^2) \tag{5.30b}$$

Substituting (5.17) into (5.30):

$$A' cos\theta_4 + B' sin\theta_4 = C' \tag{5.31}$$

Where:

$$A' = U' \cdot \overrightarrow{z_4} \tag{5.31a}$$

$$B' = U' \cdot \overrightarrow{x_4} \times \overrightarrow{z_4} \tag{5.31b}$$

$$C' = V' \tag{5.31c}$$

In order to solve the θ_4, two linear equations (5.26) and (5.31) need to be solved:

$$cos\theta_4 = -\mathrm{M}_2 / M_3 \tag{5.32}$$

$$sin\theta_4 = \mathrm{M}_1 / M_3$$

Where:

$$M_1 = (\mathrm{AC}' - \mathrm{A'C}) \tag{5.32a}$$

$$M_2 = (\mathrm{BC}' - \mathrm{B'C}) \tag{5.32b}$$

$$\mathrm{M}_3 = (\mathrm{AB}' - \mathrm{A'B}) \tag{5.32c}$$

θ_2 can be determined by (5.23) and (5.27):

$$cos\theta_2 = (cos\alpha_1 cos\alpha_2 - \overrightarrow{z_1} \cdot \overrightarrow{z_3}) / (sin\alpha_1 sin\alpha_2) \tag{5.33}$$

$$sin\theta_2 = (b_3 + b_2 cos\alpha_2 - \vec{I} \cdot \overrightarrow{z_3}) / (\rho_1 sin\alpha_2)$$

Where:

$$\overrightarrow{z_3} = \rho_1\overrightarrow{z_5} + \rho_2\overrightarrow{x_5} + \rho_1\overrightarrow{z_5} \times \overrightarrow{x_5} \quad (5.33a)$$

$$\rho_1 = (cos\alpha_3 cos\alpha_4 - sin\alpha_3 cos\theta_4 sin\alpha_4) \quad (5.33b)$$

$$\rho_2 = (cos\alpha_3 sin\alpha_4 + sin\alpha_3 cos\theta_4 cos\alpha_4) sin\theta_5 + sin\alpha_3 sin\theta_4 cos\theta_5 \quad (5.33c)$$

$$\rho_3 = (cos\alpha_3 sin\alpha_4 + sin\alpha_3 cos\theta_4 cos\alpha_4) cos\theta_5 - sin\alpha_3 sin\theta_4 sin\theta_5 \quad (5.33d)$$

In order to solve θ_1, $\overrightarrow{z_3}$ has to be rewritten as follows:

$$\overrightarrow{z_3} = n_1 cos\theta_1 + n_2 sin\theta_1 + \beta_3\overrightarrow{z_1} \quad (5.34)$$

Where:

$$n_1 = \beta_1\overrightarrow{x_1} + \beta_2\overrightarrow{x_1} \times \overrightarrow{z_1} \quad (5.34a)$$

$$n_2 = \beta_2\overrightarrow{x_1} - \beta_1\overrightarrow{x_1} \times \overrightarrow{z_1} \quad (5.34b)$$

$$\beta_1 = sin\theta_2 cos\alpha_2 \quad (5.34c)$$

$$\beta_2 = sin\alpha_1 cos\alpha_2 + cos\theta_2 cos\alpha_1 sin\alpha_2 \quad (5.34d)$$

$$\beta_3 = cos\alpha_1 cos\alpha_2 - cos\theta_2 sin\alpha_1 sin\alpha_2 \quad (5.34e)$$

Merging (5.33a) and (5.34):

$$n = n_1 cos\theta_1 + n_2 sin\theta_1 \quad (5.35)$$

$$n = \rho_1\overrightarrow{z_5} + \rho_2\overrightarrow{x_5} + \rho_3\overrightarrow{z_5} \times \overrightarrow{x_5} - \beta_3\overrightarrow{z_1} \quad (5.36)$$

Taking the dot product of n_1 and n_2 with both sides of (5.35):

$$cos\theta_1 = (n_1 \cdot n)/(\beta_1{}^2 + \beta_2{}^2) \quad (5.37)$$

$$sin\theta_1 = (n_2 \cdot n)/(\beta_1{}^2 + \beta_2{}^2)$$

Finally, in order to determine θ_3, $\overrightarrow{x_3}$ need to be solved first:

$$\overrightarrow{x_3} = cos\theta_3\overrightarrow{x_2} + sin\theta_3\overrightarrow{z_3} \times \overrightarrow{x_2} \quad (5.38)$$

Taking the dot product of $\overrightarrow{x_2}$ and $\overrightarrow{z_3} \times \overrightarrow{x_2}$ with both sides of (5.38):

$$cos\theta_3 = \overrightarrow{x_3} \times \overrightarrow{x_2} \quad (5.39)$$

$$sin\theta_3 = \overrightarrow{x_3} \cdot \overrightarrow{z_3} \times \overrightarrow{x_2}$$

With the algorithm derived above, a general theoretical inverse kinematic model has been built successfully. With the position vector of end effector as input data, all the angles for joints can be calculated and control the robotic arm to perform a desired action. However, the vector algebraic method is a theoretical method. There are modeling errors. The dimension of modules may be different between CAD model and real modules, which may influence the accuracy of the calculation results. Furthermore, solving such a number of non linear functions still persecutes people in practice. Therefore, Artificial Neural Networks (ANN) has been introduced as an excellent assistance tool due to the learning ability. The ANN method is sensor based, which can be used in real time manner in the future. ANN can also reflect the real parameters from modules, which is able to eliminate the modeling errors.

5.1.4 Inverse Kinematics Analysis base on ANN Method

An artificial neural network (ANN), usually called "neural network" (NN), is a mathematical model or computational model that tries to simulate the structure and/or functional aspects of biological neural networks. It consists of an interconnected group of artificial neurons and processes information using a connectionist approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase. In more practical terms neural networks are non-linear statistical data modeling tools. They can be used to model complex relationships between inputs and outputs or to

find patterns in data. ANN is a network of processing neurons, which can exhibit complex global behavior, determined by the connections between the processing elements and element parameters. ANN is a parallel-distributed information processing system. This system consists of operators interconnected via one-way signal flow channels. ANN stores the samples with a distributed coding, in order to a trainable nonlinear system. ANN usually contains three layers: input layer, hidden layer and output layer as shown in Fig. 5.8. Given the inputs and desired outputs, it will be self-adaptive to the environment so as to respond different inputs rationally. However, it has a complex internal structure, so the ANN's realized to date are composite systems imitating basic biological functions of neurons.

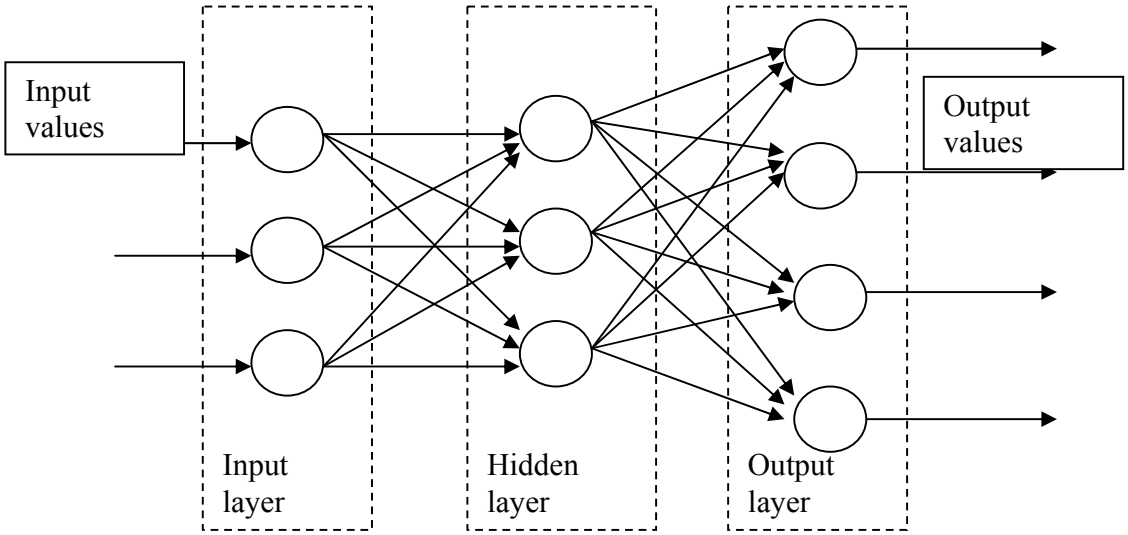


Fig.5.8 Structure of Neural Networks

Among many neural network learning algorithms, the backpropagation (BP) algorithm is one of the most popular algorithms and has been widely used, since it can map nonlinear processes. BP neural network is a feed-forward multi-layer network,

which is generalized by the Windrow-Hoff learning rule to multiple-layer networks and nonlinear differentiable transfer functions. The BP algorithm is composed of forward propagation of data flow and back propagation of error signal. Each input data is starting with FP from input layer to output layer through hidden layer. If the weight value between input layer and hidden layer is v_{ki} and the weight value between hidden layer and output layer is w_{jk}, then the transfer function for the hidden layer and output layer with n inputs, q hidden neurons and m outputs can be derived as follows [26]:

$$z_k = f_1\left(\sum_{i=0}^{n} v_{ki} x_i\right) \quad k=1,2,\cdots\cdots, q \tag{5.40}$$

$$y_j = f_2\left(\sum_{k=0}^{q} w_{jk} z_k\right) \quad j=1,2,\cdots\cdots, m \tag{5.41}$$

However, if the outputs do not match with the expected output, then back propagation will be used until the network error reaches the minimum. The expected outputs are set as t_j. Then for the pth learning data, the mean of the squared error is:

$$E_p = \frac{1}{2} \sum_{j=1}^{m} (t_j^p - y_j^p)^2 \tag{5.42}$$

In order to reduce the global error, w_{jk} has to be readjusted by the accumulative error BP algorithm as follows:

$$\Delta w_{jk} = -\mu \frac{\partial E}{\partial w_{jk}} = -\mu \frac{\partial}{\partial w_{jk}} \left(\sum_{p=1}^{p} E_p\right) = \sum_{p=1}^{p} \left(-\mu \frac{\partial E_p}{\partial w_{jk}}\right) \tag{5.43}$$

where μ is the learning rate.

The error signal is defined as follows:

$$\delta_{yj} = -\frac{\partial E_p}{\partial S_j} = -\frac{\partial E_p}{\partial y_j} \cdot \frac{\partial y_j}{\partial S_j} \tag{5.44}$$

By chain theorem, the formula for weight value at the output layer can be readjusted as follows:

$$\Delta w_{jk} = \sum_{p=1}^{p} \sum_{j=1}^{m} \mu \left(t_j^p - y_j^p \right) f_2'(S_j) z_k \tag{5.45}$$

With the similar approach, the formula for weight value at the hidden layer can be readjusted as follows:

$$\Delta v_{ki} = \sum_{p=1}^{p} \sum_{j=1}^{m} \mu \left(t_j^p - y_j^p \right) f_2'(S_j) w_{jk} f_1'(S_k) x_i \tag{5.46}$$

In this thesis, the ANN is programmed with Matlab. The following constructor is mainly used to create the feed-forward BP neural network.

```
net = newff(PN,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)
```

PN is the N x 2 matrix of min and max values for N input elements. Alternatively, minmax(input) can simply replace this function. Si is the size of each layer. There are 3 inputs, 18 hide layers and 4 outputs in this network. TFi is the transfer function of each layer. Tansig is used in this program, which is a hyperbolic tangent sigmoid transfer function as shown in Fig. 5.9. BTF is the BP network traning function. Trainingdx is used in this program, which updates weight and bias values according to gradient descent momentum and an adaptive rate. BLF is the BP network learning function. Learngdm is used in this program which is the gradient descent with momentum weight and bias learning function. Finally, PF is the performance function. Mse is used in this program, which measures the network's performance according to the mean of squared errors. Therefore, the completed code is as follows:

```
net = newff(minmax(pp'),[3 20 4],{'tansig','tansig','tansig'},'trainlm')
[net1,tr]=train(net1,pp',t)
```

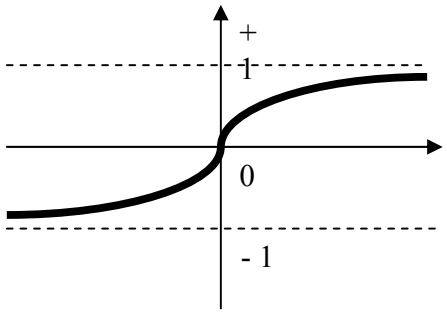


Fig. 5.9 Graph of Tan-Sigmoid Transfer Function

Additionally, neural network training can be made more efficient by scaling the inputs and expected outputs [27]. Therefore, these data can always fall within a specified range. The Matlab function premnmx() can be used to scale inputs and targets so that they fall in the range [-1,1]. In the same way, the postmnmx() can used to convert the network output back into the original units.

The following data in table 5.2 and 5.3 is collected from experiments via forward kinematics. The different combinations of joint angles are selected and substitute into forward kinematics in order to obtain the coordinates of the end effector. The joint angle is set as input with unit degree and the end effector coordinates are set as expected value with unit meter. By training the network, the relationship between input and output has been discovered finally. Fig 5.10 shows the training process at 300 epochs, which is way off the task goal. Fig 5.11 shows the training process at 9850 epochs, which meets the performance goal.

θ_1	θ_2	θ_3	θ_4	x	y	z
0	0	0	0;	0	0	0.841;
0	0	0	20;	0.112	0	0.804;
0	0	0	30;	0.148	0	0.779;
0	0	90	0;	0	0	0.841;
0	0	90	10;	0.072	0	0.819;
0	0	90	20;	0.112	0	0.804;
0	0	90	30;	0.151	0	0.779;
0	0	30	10;	0.045	0.03	0.821;
0	0	45	10;	0.0601	0.07	0.822;
0	0	60	10;	0.048	0.055	0.821;
0	30	0	10;	-0.191	0	0.781;
0	45	0	10;	-0.281	0	0.729;
0	60	0	10;	-0.392	0	0.651;
0	30	30	10;	-0.165	0.028	0.789;
0	30	45	10;	-0.205	0.042	0.785;
0	30	60	10;	-0.198	0.055	0.771;
0	45	30	10;	-0.278	0.034	0.721;
0	45	45	10;	-0.272	0.025	0.712;
0	45	60	10;	-0.333	0.033	0.701;
0	60	30	10;	-0.385	0.023	0.619;
0	60	45	10;	-0.402	0.022	0.601;
0	60	60	10;	-0.422	0.033	0.595;
30	30	30	10;	-0.208	-0.042	0.792;
30	60	30	10;	-0.358	-0.158	0.568;
60	30	30	30;	-0.144	-0.092	0.792;
60	60	30	30;	-0.228	-0.225	0.662;

Table 5.2 Data for ANN Training

θ_1	θ_2	θ_3	θ_4	x	y	z
0	0	0	10;	0.073	0	0.819;
0	0	0	25;	0.134	0	0.791;
0	0	0	35;	0.172	0	0.771;
0	0	90	5;	0.051	0	0.832;
0	0	90	15;	0.089	0	0.812;
0	0	90	25;	0.128	0	0.791;
0	0	90	35;	0.172	0	0.771;
0	0	30	30;	0.095	0.05	0.781;
0	0	45	30;	0.112	0.085	0.789;
0	0	60	30;	0.087	0.085	0.779;
0	30	0	30;	-0.098	0	0.812;
0	45	0	30;	-0.183	0	0.761;
0	60	0	30;	-0.301	0	0.671;
0	30	30	30;	-0.135	0.035	0.789;
0	30	45	30;	-0.161	0.072	0.771;
0	30	60	30;	-0.171	0.078	0.765;
0	45	30	30;	-0.234	0.022	0.735;
0	45	45	30;	-0.238	0.065	0.718;
0	45	60	30;	-0.236	0.095	0.693;
0	60	30	30;	-0.386	0.042	0.659;
0	60	45	30;	-0.365	0.054	0.643;
0	60	60	30;	-0.394	0.089	0.622;
30	30	60	10;	-0.228	-0.058	0.784;
30	60	60	10;	-0.408	-0.128	0.558;
60	30	60	30;	-0.195	-0.062	0.772;

Table 5.3 Data for trained ANN validation

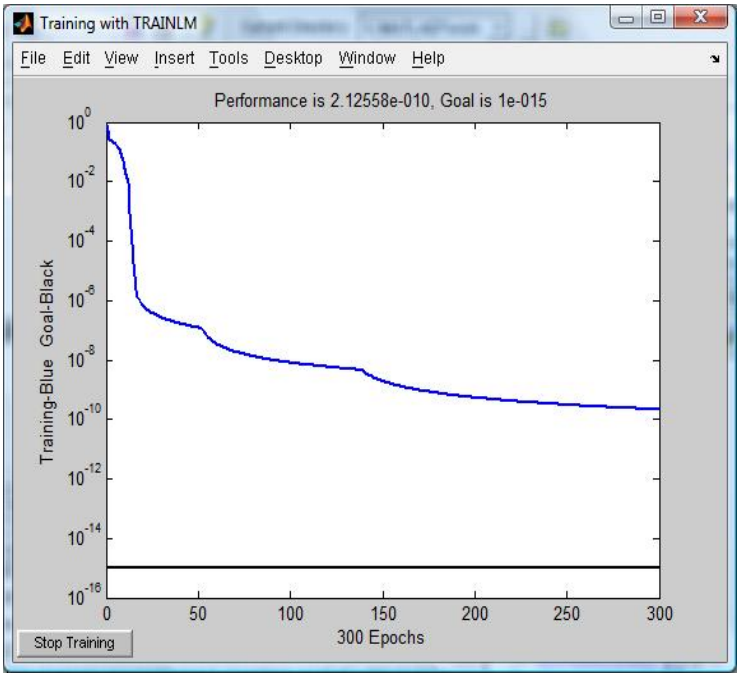


Fig. 5.10 Training process at 300 epochs

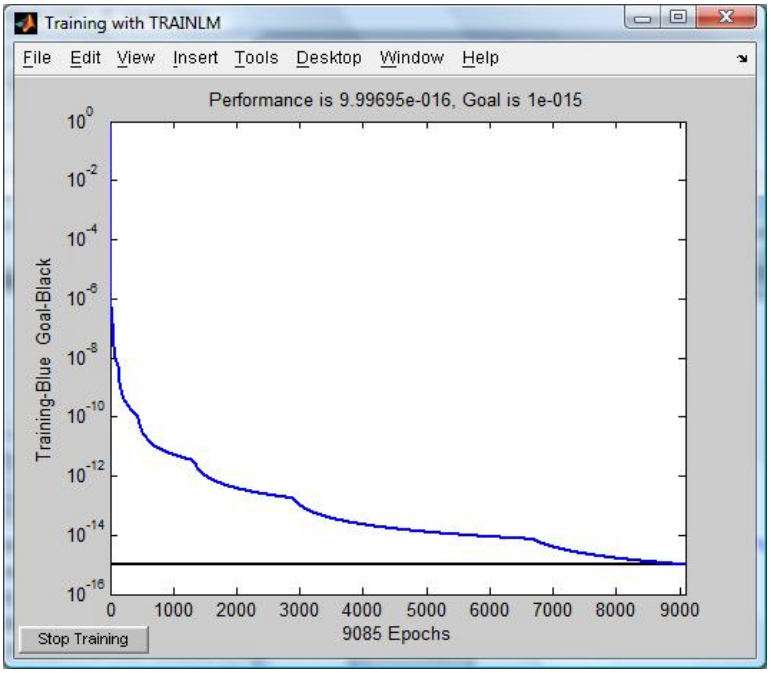


Fig. 5.11 Training process at 9085 epochs

5.2 Motion Trajectory Planning

Pick and place operation [28] is one of the typical tasks for trajectory planning techniques. In PPO, a robot is designed to take a workpiece from a given initial pose to final pose specified by the position and orientation in certain coordinate frame. The Point to Point motion trajectory planning is used in this thesis. The main idea of this approach is calculating the rotate angle of each joint by given the coordinate of some points in space, which is passed through by the robot. By analyzing these joint angles, a smooth function will be derived which accommodates with all the points passed through by the robot. The smooth function could be divided into many parts, but the time of each part is consistent for all joints. In order to achieve the PTP motion, the smooth function [29] of each part has to satisfy 8 conditions: requirements of position, velocity, and acceleration of initial and final poses, requirements of position coordinates of rising and falling points. A 7 order polynomial is employed to represent the motion of each joint.

$$p_i(t) = a_{i0} + a_{i1}t + a_{i2}t^2 + \cdots + a_{i7}t^7 \qquad (i = 1, \cdots N) \tag{5.47}$$

Since it is a high order polynomial, it requires a lot of work on calculation. So separating the motion trajectory to reduce the order is necessary. If only two middle points are considered, the polynomial can be divided with the 3-5-3 trajectory division method. This method is dividing the motion of each joint into 3 parts. The first section is from the initial point to the rising point, which is expressed by a 3 orders polynomial. The second section is from the rising point to the falling point, which is described by a 5 orders polynomial. The last section is from the falling point to the final point, which is

represented by a 3 orders polynomial. The functions of each subsection trajectory are shown as follows:

$$U_1(t) = a_{10} + a_{11}t + a_{12}t^2 + a_{13}t^3 \tag{5.48}$$

$$U_2(t) = a_{20} + a_{21}t + a_{22}t^2 + a_{23}t^3 + a_{24}t^4 + a_{25}t^5 \tag{5.49}$$

$$U_3(t) = a_{30} + a_{31}t + a_{32}t^2 + a_{33}t^3 \tag{5.50}$$

Therefore, the velocity function of each joint can be expressed as follows by taking the first derivative of position function:

$$V_i(t) = \frac{dU_i(t)}{d\tau} = \frac{1}{\Delta\tau_i}\frac{dU_i(t)}{dt} \tag{5.51}$$

Where τ_i is the real time for ith trajectory, and t is the dimensionless time.

The acceleration function of each joint can be determined by taking the second derivative of position function:

$$a_i(t) = \frac{d^2U_i(t)}{d\tau^2} = \frac{1}{(\Delta\tau_i)^2}\frac{d^2U_i(t)}{dt^2} \tag{5.52}$$

In order to solve these functions amount of initial conditions have to be setup such as the position, velocity and acceleration of the joint at the initial point:

$$p_0 = p(t_0), V_0 = \dot{p}(t_0), a_0 = \ddot{p}(t_0) \tag{5.53}$$

The position of the joint at the rising and falling points:

$$p_1 = p(t_1), p_2 = p(t_2) \tag{5.54}$$

The position, velocity and acceleration of the joint at the final point:

$$p_f = p(t_f), V_f = \dot{p}(t_f), a_f = \ddot{p}(t_f) \tag{5.55}$$

As soon as the coefficients of each polynomial are determined, the motion of all the joints can be marked out.

Base on the above path trajectory method, a pick and place test has been done showed in the following Fig. 5.12. The required initial data is collocated in the table 5.4.

Fig. 5.12 Test on Pick and Place motion

	θ1	θ2	θ3	θ4	Vel	Acc
initial	-45	95	0	-85	0	0
rising	-50	90	0	-80	15	60
falling	-135	75	0	-65	15	60
final	-150	80	0	-70	0	0

Table 5.4 motion planning data

5.3 Summary

In this chapter, geometric structure is studied at first. Nine unique configurations are introduced due to different workspace. Forward and inverse kinematic analysis is studied on a chosen configuration by vector algebraic method. Neural network is used to improve the inverse kinematic analysis. A pick and place experiment is done by the Point to Point motion trajectory planning.

Chapter 6

Conclusion and Future Work

6.1 Contributions of the Thesis

The Internet-based SRMR System designed in this thesis has the potential applications in industrial, military and medical fields. The contributions of this thesis can be summarized as follows.

- An Internet-based robot control system, with client-server architecture is developed. This system can be used to control various physical devices over the Internet.
- A Java3D model is created at the client side, which can perform the same action as the real robotic arm.
- An inverse kinematics modeling based on Artificial Neural Network is developed. It improves the accuracy of the robotic system. The forward kinematics is also studied.
- A motion trajectory planning algorithm is developed by using spline function approach. The motion of each joint of robot is specified by a set of spline functions. The method is simple and has less computation amount.

Overall, an internet-based serial reconfigurable modular robotic system is developed with advanced techniques, such as Java3D, CAN Bus. Many current robotic systems do not use these advanced techniques and has these features.

6.2 Conclusion

The serial manipulator is assembled by PowerCube modules with 4-DOF, and can be reassembled easily into many other configurations to reach different workspace. With this feature, the reconfigurable robot can suit many unstructured and less predictable environments. The additional remote control ability allows the robot to work in high risk environment with radiation, research under the deep sea or explore the inaccessible area for human beings. An overview of the history of robots is presented. Different types of robots are compared. Serial robots are introduced specially due to its simple structure, decoupled control system and large workspace. The idea of reconfigurable modular robot system is proposed, which brings the motivation on research. Past efforts on remote control are also studied. An overview of the SRMR system is described. Requirements are analyzed at the server side and the client side. GUI models are proposed with certain functions. The architecture of SRMR system is designed theoretically. The hardware of SRMR system is introduced. The features and functionally of PowerCube modules are studied. Power system is built successfully. Controller Area Network serial bus system is mainly introduced with its API protocol and physical equipments. Amount of programming is explained such as GUI creation at the server and the client side, 3D

model simulation, and internet communication. The server is programmed with Visual Basic.net. The client is programmed with Java and Java3D. The communication is set up by TCP/IP socket protocol to achieve the remote control ability. Finally a special configuration of serial robot is studied. Forward and Inverse Kinematics and path planning are worked on it. A number of equations are derived to solve those problems.

6.3 Future Work

In this thesis, only one of the configurations of the modular robotic arm has been studied. Similar work can be done to the other configurations with the general methods and equations, which were derived in this thesis. The mobility could bring many interesting issues to the modular robot. The mobile base could be connected to the base unit of the modular robotic arm. Thus, a robot assembly configuration can perform a variety of tasks just by changing the base location such as executing the search and rescue mission, exploring the risky unknown environment, and working in the unreachable situation for human beings. The standard function of a single modular robotic arm has been fully developed in this thesis. In the future, a group of similar modular robotic arms can be gathered to form a work cell, which can perform more complex and various tasks.

Reference

[1] H. Bruyninckx, and J. Hallam, “The Robotics WebBook”, 2005.

[2] D. –S. Necsulescu, “Stochastic Modelling of Robotic Manufacturing Systems”, 1985.

[3] G.A. Mintchell, “Industrial Robots: Fast, Nimble at 30”, Control Engineering 49, no. 11 pp.32, 2002.

[4] R. Saltaren, “Field and Service Applications – Exploring Deep Sea by Teleoperated Robot – an Underwater Parallel Robot with High Navigation Capabilities”, IEEE Robotics Automation Magazine 14, no. 3 pp. 65, 2007.

[5] Overview of Parallel Robots, http://www.robotmatrix.org/ParallelRobotic.htm.

[6] SCARA, http://en.wikipedia.org/wiki/SCARA.

[7] Dan Zhang, Lihui Wang and Ebrahim Esmailzadeh, “Web-Based Remote Manipulation of Parallel Robot in Advanced Mnufactruing Systems”

[8] D. Schmitz, P. Khosla and T. Kanade, “The CMU reconfigurable Modular Manipulator System,” Carnegie Mellon Univ., CMU-RI-TR-88-7, 1988.

[9] T. Fukuda and S. Nakagawa, “Dynamically Reconfigurable Robotic System,” in Pro. IEEE Int. Conf. Robotics and Automation. Pp. 1581-1586, 1988.

[10] R. Cohen, M. G. Lipton, M. Q. Dai and B. Benhabib, “Conceptual Design of a Modular Robot,” ASME J. Mechanical Design, 114, pp. 117-125, March, 1992.

[11] K. H. Wurst, “The Conception and Construction of a Modular Robot System,” in Proc. 16th Int. Sym. Industrial Robotics (ISIR). Pp. 37-44, 1986

[12] D. Tesar and M. S. Butler, “A Generalized Modular Architecture for Robot Structures,” ASME J. of Manufacturing Review, 2, no. 2, pp. 91-117, 1989.

[13] Sathish Kumar, “Design and development of a system to control the HERO 2000 robot from a PC”, University of Louisville, 1995.

[14] Zhiwei Gen, “Supporting remote sensing and control over IP networks”, Michigan State University, 2007, 178 pages; AAT 3264151

[15] I-Ming Chen, “Theory and applications of modular reconfigurable robotic systems”, California Institute of Technology, 1994.

[16] Pavel Trnka and Petr Smolik, “Profibus DP Master for PC”, Czech Technical University

[17] Original Marken Partner Lernsysteme, http://lernsystem.original-marken-partner.de/Advantages-and-disadvantages.442.0.html?&L=1

[18] Igor Belousov, “Sidebar – Java3D for Web-Based Robot Control”, Russia

[19] C.C. Ko, Ben M. Chen, C.M. Loke, C.D. Cheng, X. Xiang, A.K. Eapen and T.S. Lim, “Automation in creating Java 3D-based Virtual Instruments”, National University of Singapore, Singapore 117576

[20] G.A. Mintchell, “Industrial Robots: Fast, Nimble at 30”, Control Engineering 49, no. 11 pp.32, 2002

[21] J. Denavit and R.S. Hartenberg. “Kinematic notation for lower-pair mechanisms based on matrices”, ASME Journal of Applied Mechanics, 2:215-211, 1955.

[22] J. Duffy, “Analysis of mechanisms and robot manipulators”, 1980.

[23] K.C. Gupta, “Kinematic analysis of manipulator using the zero reference position description”, In J.M. McCarthy, editor, “The Kinematics of Robot Manipulators”,, pages 3-11, Cambridge, MA, 1987, MIT Press.

[24] R. Manseur and K. L. “Doty, mech. Mach. Theory” 27, 575-586, 1992

[25] Y.B. Zhou, R. O. Buchal and R.G. Fenton, “Analysis of the General 4R and 5R Robots Using a Vector Algebraic Approach”, Mech. Mach. Theory. Vol. 30, No. 3, pp.421-432, 1995

[26] Philippe Crochat and Daniel Franklin, Back-Progagation Neural Network, http://pcrochat.online.fr/webus/tutorial/BPN_tutorial.html

[27] Jianhe Lei, “Application of neural network to nonlinear static decoupling of robot wrist force sensor”, the 6th world congress on Intelligent Control and Automation, 2006.

[28] Jorge Angeles, “Fundamentals of Robotic Mechanical Systems”, 189-202, 2003

[29] Xiangrong Xu, Xiaogang Wang and Feng Qin, “Trajectory Planning of Robot Manipulators by Using Spline Function Approach”, the 3rd world congress on intelligent control and automation, 2000.

Appendix A: Server VB.net Program

```
Imports System.Net.Sockets
Imports System.Net
Imports System.Threading
Imports System.Text

Public Class Form1

    Dim ret, ret1, ret2, ret3, ret4, ret5 As Long
    Dim dev As Long
    Dim numOfModules As Long
    Dim modId1, modId2, modId3, modId4, modID5 As Long
    Dim state As Integer
    Dim pos1, pos2, pos3, pos4, pos5 As Single
    Dim vel As Single
    Dim acc As Single
    Dim restStr1, restStr2, restStr3, restStr4 As String
    Dim onoff As Single
    Dim x1, x2, x3, x4 As Single
    Dim y1, y2, y3, y4 As VariantType


    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        Do
            ' Must listen on correct port- must be same as port client wants to connect
on.
            ' Const portNumber As Integer = 8000
            'Dim localAddr As IPAddress = IPAddress.Parse("10.129.13.66")
            Const portNumber As Integer = 8000
            Dim localAddr As IPAddress = IPAddress.Parse("10.129.13.64")
            Dim tcpListener As New TcpListener(localAddr, portNumber)

            tcpListener.Start()
            TextBox1.Text = "Waiting for connection..."
            Try
                'Accept the pending client connection and return                    'a
TcpClient initialized for communication.
                Dim tcpClient As TcpClient = tcpListener.AcceptTcpClient()
                TextBox2.Text = "Connection accepted."

                ' Get the stream
```

```
Dim networkStream As NetworkStream = tcpClient.GetStream()
' Read the stream into a byte array
Dim bytes(tcpClient.ReceiveBufferSize) As Byte
networkStream.Read(bytes, 0, CInt(tcpClient.ReceiveBufferSize))

' Return the data received from the client to the console.
Dim clientdata As String = Encoding.ASCII.GetString(bytes)

If Val(clientdata) = -1 Then

    TextBox7.Text = "Grab"
    pos5 = -10
    vel = 0.05
    acc = 0.01
    modID5 = 12
    dev = 0
    ret5 = PCube_moveRamp(dev, modID5, pos5, vel, acc)

ElseIf Val(clientdata) = 1 Then

    TextBox7.Text = "Release"
    pos5 = 10
    vel = 0.05
    acc = 0.01
    modID5 = 12
    dev = 0
    ret5 = PCube_moveRamp(dev, modID5, pos5, vel, acc)

ElseIf Val(clientdata) = 0 Then

    modId1 = 8
    dev = 0
    ret1 = PCube_homeModule(dev, modId1)
    If ret1 <> 0 Then
        End
    End If

    modId2 = 9
    dev = 0
    ret2 = PCube_homeModule(dev, modId2)
    If ret2 <> 0 Then
        End
    End If

    modId3 = 10
    dev = 0
```

```
        ret3 = PCube_homeModule(dev, modId3)
        If ret3 <> 0 Then
            End
        End If

        modId4 = 11
        dev = 0
        ret4 = PCube_homeModule(dev, modId4)
        If ret4 <> 0 Then
            End
        End If

        modID5 = 12
        dev = 0
        PCube_resetModule(dev, modID5)
        ret5 = PCube_homeModule(dev, 12)
        If ret5 <> 0 Then
            End
        End If

    Else

        For i = 1 To Len(clientdata)
            If Mid(clientdata, i, 1) = "/" Then
                TextBox3.Text = Mid(clientdata, 1, i - 1)
                restStr1 = Mid(clientdata, i + 1, Len(clientdata))
                Exit For
            End If
        Next i

        For i = 1 To Len(restStr1)
            If Mid(restStr1, i, 1) = "/" Then
                TextBox4.Text = Mid(restStr1, 1, i - 1)
                restStr2 = Mid(restStr1, i + 1, Len(restStr1))
                Exit For
            End If
        Next i

        For i = 1 To Len(restStr2)
            If Mid(restStr2, i, 1) = "/" Then
                TextBox5.Text = Mid(restStr2, 1, i - 1)
                restStr3 = Mid(restStr2, i + 1, Len(restStr2))
                Exit For
            End If
        Next i
```

```
                    For i = 1 To Len(restStr3)
                        If Mid(restStr3, i, 1) = "/" Then
                            TextBox6.Text = Mid(restStr3, 1, i - 1)
                            restStr4 = Mid(restStr3, i + 1, Len(restStr3))
                            Exit For
                        End If
                    Next i

                    onoff = Val(restStr4)

                    If Val(TextBox3.Text) = 0.0 And Val(TextBox4.Text) = 0.0
And Val(TextBox5.Text) = 0.0 And Val(TextBox6.Text) = 0.0 Then

                        modId1 = 8
                        dev = 0
                        ret1 = PCube_homeModule(dev, modId1)
                        If ret1 <> 0 Then
                            End
                        End If

                        modId2 = 9
                        dev = 0
                        ret2 = PCube_homeModule(dev, modId2)
                        If ret2 <> 0 Then
                            End
                        End If

                        modId3 = 10
                        dev = 0
                        ret3 = PCube_homeModule(dev, modId3)
                        If ret3 <> 0 Then
                            End
                        End If

                        modId4 = 11
                        dev = 0
                        ret4 = PCube_homeModule(dev, modId4)
                        If ret4 <> 0 Then
                            End
                        End If

                        modID5 = 12
                        dev = 0
                        PCube_resetModule(dev, modID5)
                        ret5 = PCube_homeModule(dev, 12)
                        If ret5 <> 0 Then
```

```
                    End
                End If

            Else
                pos1 = Val(TextBox3.Text)
                vel = 0.5
                acc = 1
                modId1 = 8
                dev = 0
                ret1 = PCube_moveRamp(dev, modId1, pos1, vel, acc)

                pos2 = Val(TextBox4.Text)
                vel = 0.5
                acc = 1
                modId2 = 9
                dev = 0
                ret2 = PCube_moveRamp(dev, modId2, pos2, vel, acc)

                pos3 = Val(TextBox5.Text)
                vel = 0.5
                acc = 1
                modId3 = 10
                dev = 0
                ret3 = PCube_moveRamp(dev, modId3, pos3, vel, acc)

                pos4 = -Val(TextBox6.Text)
                vel = 0.5
                acc = 1
                modId4 = 11
                dev = 0
                ret4 = PCube_moveRamp(dev, modId4, pos4, vel, acc)
            End If

        End If

        tcpClient.Close()
        tcpListener.Stop()

    Catch exp As Exception
        'Console.WriteLine(e.ToString())
        'Console.ReadLine()
    End Try
Loop While onoff = 1

End Sub
```

```
    Private Sub TextBox3_TextChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles TextBox3.TextChanged

    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        dev = 0
        ret = PCube_openDevice(dev, "ESD:0,250")
        If ret <> 0 Then
            End
        End If
    End Sub

    Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button5.Click
        dev = 0
        ret = PCube_closeDevice(dev)
        End
    End Sub

    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click

        modId1 = 8
        dev = 0
        ret1 = PCube_homeModule(dev, modId1)
        If ret1 <> 0 Then
            End
        End If

        modId2 = 9
        dev = 0
        ret2 = PCube_homeModule(dev, modId2)
        If ret2 <> 0 Then
            End
        End If

        modId3 = 10
        dev = 0
        ret3 = PCube_homeModule(dev, modId3)
```

```
        If ret3 <> 0 Then
            End
        End If

        modId4 = 11
        dev = 0
        ret4 = PCube_homeModule(dev, modId4)
        If ret4 <> 0 Then
            End
        End If

        modID5 = 12
        dev = 0
        PCube_resetModule(dev, modID5)
        ret5 = PCube_homeModule(dev, 12)
        If ret5 <> 0 Then
            End
        End If

    End Sub

    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click

        pos1 = Val(TextBox3.Text)
        vel = 0.5
        acc = 1
        modId1 = 8
        dev = 0
        ret1 = PCube_moveRamp(dev, modId1, pos1, vel, acc)

        pos2 = Val(TextBox4.Text)
        vel = 0.5
        acc = 1
        modId2 = 9
        dev = 0
        ret2 = PCube_moveRamp(dev, modId2, pos2, vel, acc)

        pos3 = Val(TextBox5.Text)
        vel = 0.5
        acc = 1
        modId3 = 10
        dev = 0
        ret3 = PCube_moveRamp(dev, modId3, pos3, vel, acc)

        pos4 = -Val(TextBox6.Text)
```

```
        vel = 0.5
        acc = 1
        modId4 = 11
        dev = 0
        ret4 = PCube_moveRamp(dev, modId4, pos4, vel, acc)

    End Sub

    Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button6.Click

        pos5 = Val(TextBox7.Text)
        vel = 0.05
        acc = 0.01
        modID5 = 12
        dev = 0
        ret5 = PCube_moveRamp(dev, modID5, pos5, vel, acc)

    End Sub

    Private Sub Button7_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button7.Click
        dev = 0
        PCube_resetAll(dev)
    End Sub

    Private Sub Button8_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button8.Click
        dev = 0
        PCube_haltAll(dev)
    End Sub

    Private Sub Button9_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button9.Click

        pos1 = Val(TextBox10.Text)
        vel = 0.5
        acc = 1
        modId1 = 8
        dev = 0
        ret1 = PCube_moveRamp(dev, modId1, pos1, vel, acc)

        For i = 1 To 1000000000

        Next i
```

```
        For i = 1 To 1000000000

        Next i

        For i = 1 To 1000000000

        Next i

        y1 = PCube_getPos(dev, modId1, x1)

        TextBox8.Text = x1

        y2 = PCube_getModuleSerialNo(dev, modId1, x2)
        TextBox9.Text = x2

    End Sub

    Private Sub Button10_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button10.Click
        modId1 = 8
        dev = 0
        ret1 = PCube_homeModule(dev, modId1)
        If ret1 <> 0 Then
            End
        End If
    End Sub

    Private Sub Button11_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button11.Click
        modId1 = 8
        dev = 0
        PCube_haltModule(dev, modId1)
    End Sub

    Private Sub Button12_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button12.Click
        modId1 = 8
        dev = 0
        PCube_resetModule(dev, modId1)
        TextBox8.Text = ""
        TextBox9.Text = ""
        TextBox10.Text = ""
    End Sub

    Private Sub Button13_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button13.Click
```

```
        modId2 = 9
        dev = 0
        ret2 = PCube_homeModule(dev, modId2)
        If ret2 <> 0 Then
            End
        End If
    End Sub

    Private Sub Button14_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button14.Click
        modId2 = 9
        dev = 0
        PCube_haltModule(dev, modId2)
    End Sub

    Private Sub Button15_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button15.Click
        modId2 = 9
        dev = 0
        PCube_resetModule(dev, modId2)
        TextBox11.Text = ""
        TextBox12.Text = ""
        TextBox13.Text = ""
    End Sub

    Private Sub Button16_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button16.Click
        pos2 = Val(TextBox11.Text)
        vel = 0.5
        acc = 1
        modId2 = 9
        dev = 0
        ret2 = PCube_moveRamp(dev, modId2, pos2, vel, acc)

        For i = 1 To 1000000000

        Next i

        For i = 1 To 1000000000

        Next i

        For i = 1 To 1000000000

        Next i
```

```
        y1 = PCube_getPos(dev, modId2, x1)

        TextBox12.Text = x1

        y2 = PCube_getModuleSerialNo(dev, modId2, x2)
        TextBox13.Text = x2
    End Sub

    Private Sub Button17_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button17.Click
        modId3 = 10
        dev = 0
        ret3 = PCube_homeModule(dev, modId3)
        If ret3 <> 0 Then
            End
        End If
    End Sub

    Private Sub Button18_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button18.Click
        modId3 = 10
        dev = 0
        PCube_haltModule(dev, modId3)
    End Sub

    Private Sub Button19_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button19.Click
        modId3 = 10
        dev = 0
        PCube_resetModule(dev, modId3)
        TextBox14.Text = ""
        TextBox15.Text = ""
        TextBox16.Text = ""
    End Sub

    Private Sub Button20_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button20.Click
        pos3 = Val(TextBox14.Text)
        vel = 0.5
        acc = 1
        modId3 = 10
        dev = 0
        ret3 = PCube_moveRamp(dev, modId3, pos3, vel, acc)

        For i = 1 To 1000000000
```

```
        Next i

        For i = 1 To 1000000000

        Next i

        For i = 1 To 1000000000

        Next i

        y1 = PCube_getPos(dev, modId3, x1)

        TextBox15.Text = x1

        y2 = PCube_getModuleSerialNo(dev, modId3, x2)
        TextBox16.Text = x2
    End Sub

    Private Sub Button21_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button21.Click
        modId4 = 11
        dev = 0
        ret4 = PCube_homeModule(dev, modId4)
        If ret4 <> 0 Then
            End
        End If
    End Sub

    Private Sub Button22_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button22.Click
        modId4 = 11
        dev = 0
        PCube_haltModule(dev, modId4)
    End Sub

    Private Sub Button23_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button23.Click
        modId4 = 11
        dev = 0
        PCube_resetModule(dev, modId4)
        TextBox17.Text = ""
        TextBox18.Text = ""
        TextBox19.Text = ""
    End Sub

    Private Sub Button24_Click(ByVal sender As System.Object, ByVal e As
```

```
System.EventArgs) Handles Button24.Click
        pos4 = Val(TextBox17.Text)
        vel = 0.5
        acc = 1
        modId4 = 11
        dev = 0
        ret4 = PCube_moveRamp(dev, modId4, pos4, vel, acc)

        For i = 1 To 1000000000

        Next i

        For i = 1 To 1000000000

        Next i

        For i = 1 To 1000000000

        Next i

        y1 = PCube_getPos(dev, modId4, x1)

        TextBox18.Text = x1

        y2 = PCube_getModuleSerialNo(dev, modId4, x2)
        TextBox19.Text = x2
    End Sub

    Private Sub Button25_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button25.Click
        modID5 = 12
        dev = 0
        ret5 = PCube_homeModule(dev, modID5)
        If ret5 <> 0 Then
            End
        End If
    End Sub

    Private Sub Button26_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button26.Click
        modID5 = 12
        dev = 0
        PCube_haltModule(dev, modID5)
    End Sub

    Private Sub Button27_Click(ByVal sender As System.Object, ByVal e As
```

```
System.EventArgs) Handles Button27.Click
        modID5 = 12
        dev = 0
        PCube_resetModule(dev, modID5)
        TextBox20.Text = ""
        TextBox21.Text = ""
        TextBox22.Text = ""
    End Sub

    Private Sub Button28_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button28.Click
        pos5 = Val(TextBox20.Text)
        vel = 0.05
        acc = 0.01
        modID5 = 12
        dev = 0
        ret5 = PCube_moveRamp(dev, modID5, pos5, vel, acc)

        For i = 1 To 1000000000

        Next i

        For i = 1 To 1000000000

        Next i

        For i = 1 To 1000000000

        Next i

        y1 = PCube_getPos(dev, modID5, x1)

        TextBox21.Text = x1

        y2 = PCube_getModuleSerialNo(dev, modID5, x2)
        TextBox22.Text = x2
    End Sub
End Class
```

Appendix B: Client Java Program

```
import javax.swing.*;
import javax.vecmath.Color3f;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.applet.Applet;
import java.awt.BorderLayout;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.geometry.Box;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.loaders.Scene;
import com.sun.j3d.loaders.objectfile.ObjectFile;
import java.awt.*;
import java.text.DecimalFormat;
import java.util.Enumeration;
import java.math.BigDecimal;
import java.net.*;
import java.io.*;

public class clientGui extends JFrame {

    //Modular move rotation in radius unit
    public float r1=0, r2, r3, r4;
    int onoff = 0;

    //Gripper move position in distance unit
    public float r5=0, r6=0 ;
    public static final long serialVersionUID = 1L;

    Socket socket;
    PrintStream ps;

    public JTextField
        txt1 = new JTextField(10),
        txt2 = new JTextField(10),
        txt3 = new JTextField(10),
        txt4 = new JTextField(10),
```

```
        txt5 = new JTextField(10),
        txt6 = new JTextField(10),
        txt7 = new JTextField(10),
        txt8 = new JTextField(10);

private JButton
        b1 = new JButton("Start" ),
        b2 = new JButton("Reset"),
        b3 = new JButton("Exit"),
        b4 = new JButton("Execute"),
        b5 = new JButton("Send"),
        b6 = new JButton("Calculate"),
        b7 = new JButton("Action");

private JLabel
        lb1 = new JLabel("X:"),
        lb2 = new JLabel("Y:"),
        lb3 = new JLabel("Z:"),
        lb4 = new JLabel("θ1:"),
        lb5 = new JLabel("θ2:"),
        lb6 = new JLabel("θ3:"),
        lb7 = new JLabel("θ4:"),
        lb8 = new JLabel("GP:");

private ActionListener bl1 = new ActionListener(){
        public void actionPerformed(ActionEvent e){
                txt4.setText("50");
                txt5.setText("60");
                txt6.setText("70");
                txt7.setText("80");
        }
};

private ActionListener bl2 = new ActionListener(){
        public void actionPerformed(ActionEvent e){
                txt1.setText("0.0");
                txt2.setText("0.0");
                txt3.setText("0.0");
                txt4.setText("0.0");
                txt5.setText("0.0");
                txt6.setText("0.0");
                txt7.setText("0.0");
        }
};

private ActionListener bl3 = new ActionListener(){
```

```
        public void actionPerformed(ActionEvent e){
            System.exit(0);
        }
    };

    public ActionListener bl4 = new ActionListener(){
        public void actionPerformed(ActionEvent e){
            String r11 = txt4.getText();
            r1 = Float.parseFloat(r11);
            String r21 = txt5.getText();
            r2 = Float.parseFloat(r21);
            String r31 = txt6.getText();
            r3 = Float.parseFloat(r31);
            String r41 = txt7.getText();
            r4 = Float.parseFloat(r41);
            GraphicsConfiguration config =
SimpleUniverse.getPreferredConfiguration();
            Canvas3D c = new Canvas3D(config);
            getContentPane().add(c);
            c.repaint();
            u = new SimpleUniverse(c);
            BranchGroup scene = createSceneGraph(u);
            // This will move the ViewPlatform back a bit so the objects in the scene can
be viewed.
            u.getViewingPlatform().setNominalViewingTransform();
            u.addBranchGraph(scene);
            c.setBounds(350, 10, 400, 500);
        }
    };

    public ActionListener bl5 = new ActionListener(){
        public void actionPerformed(ActionEvent e){
            try {
                socket = new Socket("10.129.13.64",8000);
                if(socket != null)
                {
                    StringBuffer msg = new StringBuffer();
                    msg.append(txt4.getText()+ "/" + txt5.getText()+ "/" +
txt6.getText()+ "/" + txt7.getText()+ "/" +"1");
                    ps = new PrintStream(socket.getOutputStream());
                    ps.println(msg);
                    ps.flush();
                    socket.close();
                }
            } catch (UnknownHostException e1) {
                // TODO Auto-generated catch block
```

```
                e1.printStackTrace();
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
        }
};

public ActionListener bl6 = new ActionListener(){
    public void actionPerformed(ActionEvent e){
        float x = Float.parseFloat(txt1.getText());
        float y = Float.parseFloat(txt2.getText());
        float z = Float.parseFloat(txt3.getText());

        if (x>0&&y>0&&x>y){
            txt4.setText("-2.4");
            txt5.setText("1.8");
            txt6.setText("-0.5");
            txt7.setText("1.1");
        }

        else if (x>0&&y>0&&x<y){
            txt4.setText("-1.9");
            txt5.setText("1.8");
            txt6.setText("0.1");
            txt7.setText("1.1");
        }

        else if (x<0&&y>0&&Math.abs(x)<y){
            txt4.setText("-1.3");
            txt5.setText("1.8");
            txt6.setText("0.2");
            txt7.setText("1.1");
        }

        else if (x<0&&y>0&&Math.abs(x)>y){
            txt4.setText("-0.7");
            txt5.setText("1.8");
            txt6.setText("0.2");
            txt7.setText("1.9");
        }

        else if (x<0&&y<0&&Math.abs(x)>Math.abs(y)){
            txt4.setText("-0.1");
            txt5.setText("1.8");
            txt6.setText("0.5");
```

```
            txt7.setText("1.3");
        }

        else if (x<0&&y<0&&Math.abs(x)<Math.abs(y)){
            txt4.setText("1.1");
            txt5.setText("2");
            txt6.setText("1.1");
            txt7.setText("0.7");
        }

        else if (x>0&&y<0&&x<Math.abs(y)){
            txt4.setText("2");
            txt5.setText("1.8");
            txt6.setText("0.1");
            txt7.setText("1.1");
        }

        else if (x>0&&y<0&&x>Math.abs(y)){
            txt4.setText("2");
            txt5.setText("1.8");
            txt6.setText("0.6");
            txt7.setText("1.3");
        }
    }
};

public ActionListener bl7 = new ActionListener(){
    public void actionPerformed(ActionEvent e){
        try {
            socket = new Socket("10.129.13.64",8000);
            if(socket != null)
            {
                StringBuffer msg = new StringBuffer();
                msg.append(txt8.getText());
                ps = new PrintStream(socket.getOutputStream());
                ps.println(msg);
                ps.flush();
                socket.close();
            }
        } catch (UnknownHostException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        } catch (IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
```

```
        }
    };


    private SimpleUniverse u = null;

    public BranchGroup createSceneGraph(SimpleUniverse u){
        BranchGroup objRoot = new BranchGroup();
        TransformGroup objScale = new TransformGroup();
        Transform3D t3d = new Transform3D();
        t3d.setScale(0.3);
        objScale.setTransform(t3d);
        objRoot.addChild(objScale);
        BoundingSphere bounds = new BoundingSphere(new Point3d(0.0,0.0,0.0),
100.0);
        Color3f bgColor = new Color3f(.05f, 0.00f, 0.0f);
        Background bg = new Background(bgColor);
        bg.setApplicationBounds(bounds);
        objRoot.addChild(bg);
        Color3f light1Color = new Color3f(1.0f, 1.0f, 0.9f);
        Vector3f light1Direction= new Vector3f(4.0f, -7.0f, -12.0f);
        DirectionalLight light1 = new DirectionalLight(light1Color, light1Direction);
        light1.setInfluencingBounds(bounds);
        objRoot.addChild(light1);
        Appearance app = new Appearance();
        Material material = new Material();
        material.setEmissiveColor(new Color3f(1.0f,0.0f,0.0f));
        app.setMaterial(material);

          BranchGroup scene1 = new BranchGroup();                  //first modular1
           scene1.addChild(new Box(.2f,.2f,.2f,app));
          BranchGroup scene2 = new BranchGroup();                     //first
modular2
           scene2.addChild(new Box(.2f,.2f,.2f,app));

          BranchGroup scene3 = new BranchGroup();                  //second
modular1
           scene3.addChild(new Box(.2f,.2f,.2f,app));
          BranchGroup scene4 = new BranchGroup();                     //second
modular2
           scene4.addChild(new Box(.2f,.2f,.2f,app));

          BranchGroup scene5 = new BranchGroup();                     //third
modular2
           scene5.addChild(new Box(.15f,.15f,.15f,app));
          BranchGroup scene6 = new BranchGroup();                     //third
```

```
modular2
            scene6.addChild(new Box(.15f,.15f,.15f,app));

            BranchGroup scene7 = new BranchGroup();                    //fourth
modular1
            scene7.addChild(new Box(.15f,.15f,.15f,app) );
           BranchGroup scene8 = new BranchGroup();                     //fourth
modular2
            scene8.addChild(new Box(.15f,.15f,.15f,app));

           BranchGroup scene9 = new BranchGroup();                     //end effector
            scene9.addChild(new Box(.2f,.2f,.2f,app));

           BranchGroup scene10 = new BranchGroup();                    //support1
            scene10.addChild(new Box(.1f,.1f,.1f,app));

           BranchGroup scene11 = new BranchGroup();                    //support2
            scene11.addChild(new Box(.1f,.1f,.1f,app));

           BranchGroup scene12 = new BranchGroup();                    //gripple1
            scene12.addChild(new Box(.035f,.1f,.2f,app));

           BranchGroup scene13 = new BranchGroup();                    //gripple2
            scene13.addChild(new Box(.035f,.1f,.2f,app));

            BranchGroup scene14 = new BranchGroup();                 //support between
Modular 1 and 2
            scene14.addChild(new Box(0.1f, 0.06f, 0.1f, app));

            BranchGroup scene15 = new BranchGroup();
            scene15.addChild(new Box(0.08f, 0.18f, 0.1f, app));

            BranchGroup scene16 = new BranchGroup();                 //support between
Modular 3 and 4
            scene16.addChild(new Box(0.1f, 0.06f, 0.1f, app));

            BranchGroup scene17 = new BranchGroup();
            scene17.addChild(new Box(0.08f, 0.18f, 0.1f, app));

          //First Modular_11 stay still
          objScale.addChild(createObject1 (scene1, 0.0f , -1.4f, 0.0f , 0.0f , 0.0f , 0.0f ,
0.0f    ));

          //First Modular_12 rotate around Global Y axis
          objScale.addChild(createObject1 (scene2, 0.0f , -1.0f, 0.0f , 0.0f , 0.0f , 0.0f ,
r1 ));
```

```
        //Second Modular_21 move with First Modular_12 around Global Y axis
        //place Modular_21
        TransformGroup tf21 = createObject1 (scene4, 0.4f , -0.5f, 0.0f , 0.0f , 0.0f ,
0.0f , 0 );
        //set a rotation around Global Y axis
        objScale.addChild(createObject1x(tf21, 0.0f , 0.0f, 0.0f , 0.0f , -1.0f, 0.0f ,r1 ));

        //Second Modular_22
        //place Modular_22 with its own rotation about X-axis
        TransformGroup tf22 = createObject2 (scene3, 0.0f , -0.5f, 0.0f , 0.0f , 0.0f ,
0.0f , r2 );
        //move with Second Modular_21
        objScale.addChild(createObject1x (tf22, 0.0f , 0.0f, 0.0f , 0.0f , -0.5f, 0.0f , r1 ));

        //Third Modular_31
        //rotate around Second Modular_2's X axis
        TransformGroup tf31    = createObject2 (scene5, 0.05f , 0.0f, 0.0f , 0.0f ,-0.5f ,
0.0f , r2 );
        //move with Second Second Modular_22
        objScale.addChild(createObject1x (tf31, 0.0f , 0.0f, 0.0f , 0.0f , -0.4f , 0.0f ,
r1 ));

        //Third Modular_32
        //place Modular_32 with its own rotation about Y-axis
        TransformGroup tf32    = createObject1 (scene6, 0.05f , 0.3f, 0.0f , 0.0f ,0.0f ,
0.0f , r3);
        //move with Third Modular_31
        TransformGroup tf321 = createObject2x (tf32, 0.0f , 0.0f, 0.0f , 0.0f , -0.5f ,
0.0f , r2 );
        objScale.addChild(createObject1x (tf321, 0.0f , 0.0f, 0.0f , 0.0f , -0.4f , 0.0f ,
r1 ));

        //Forth Modular_41
        //place Modular_41
        TransformGroup tf41    = createObject1 (scene7, -0.25f , 0.7f, 0.0f ,    0.0f , 0.0f,
0.0f , 0);
        //rotate around Modular_32's Y axis
        TransformGroup tf411    = createObject1x (tf41, 0.0f , 0.0f, 0.0f ,    0.05f , 0.3f,
0.0f , r3);
        //move with Modular_32
        TransformGroup tf4111 = createObject2x (tf411, 0.0f , 0.0f, 0.0f , 0.0f , -0.5f ,
0.0f , r2 );
        objScale.addChild(createObject1x (tf4111, 0.0f , 0.0f, 0.0f , 0.0f , -0.4f , 0.0f ,
r1 ));
```

```
            //Forth Modular_42
            //place Modular_42 and rotate around its own X axis
            TransformGroup tf42 = createObject2 (scene8, 0.05f , 0.7f, 0.0f ,    0.0f , 0.0f,
0.0f , r4);
            //move with Modular_41
            TransformGroup tf421    = createObject1x (tf42, 0.0f , 0.0f, 0.0f ,    0.05f , 0.3f,
0.0f , r3);
            TransformGroup tf4211 = createObject2x (tf421, 0.0f , 0.0f, 0.0f , 0.0f , -0.5f ,
0.0f , r2 );
            objScale.addChild(createObject1x (tf4211, 0.0f , 0.0f, 0.0f , 0.0f , -0.4f , 0.0f ,
r1 ));

            //End Effector
            //place the End Effector
            TransformGroup tf51    = createObject2 (scene9, 0.05f , 1.1f, 0.0f ,    0.0f , 0.0f,
0.0f , 0);
            //rotate around Modular_42's X axis
            TransformGroup tf511    = createObject2x (tf51, 0.0f , 0.0f, 0.0f ,    0.05f , 0.7f,
0.0f , r4);
            //move with Modular_42
            TransformGroup tf5111    = createObject1x (tf511, 0.0f , 0.0f, 0.0f ,    0.05f ,
0.3f, 0.0f , r3);
            TransformGroup tf51111 = createObject2x (tf5111, 0.0f , 0.0f, 0.0f , 0.0f , -0.5f ,
0.0f , r2 );
            objScale.addChild(createObject1x (tf51111, 0.0f , 0.0f, 0.0f , 0.0f , -0.4f , 0.0f ,
r1 ));

            //Connection between Modular_22 and Modular_31
            TransformGroup con1    = createObject2 (scene10, 0.05f , -0.2f, 0.0f , 0.0f ,0.f ,
0.0f , 0 );
            //move with Second Second Modular_22
            TransformGroup con11    = createObject2x(con1, 0.0f , 0.0f, 0.0f , 0.0f ,-0.5f ,
0.0f , r2 );
            objScale.addChild(createObject1x (con11, 0.0f , 0.0f, 0.0f , 0.0f , -0.4f , 0.0f ,
r1 ));

            //Connection between Modular_42 and End Effector
            //place the connection2
           TransformGroup con21 = createObject2 (scene11, 0.05f , 0.9f, 0.0f ,    0.0f , 0.0f,
0.0f , 0);
            //move with Modular_42
            TransformGroup con211 = createObject2x (con21, 0.0f , 0.0f, 0.0f ,    0.05f ,
0.7f, 0.0f , r4);
            TransformGroup con2111    = createObject1x (con211, 0.0f , 0.f, 0.0f ,    0.05f ,
0.3f, 0.0f , r3);
            TransformGroup con21111 = createObject2x (con2111, 0.0f , 0.0f, 0.0f , 0.0f ,
```

```
-0.5f , 0.0f , r2 );
          objScale.addChild(createObject1x (con21111, 0.0f , 0.0f, 0.0f , 0.0f , -0.4f , 0.0f ,
r1 ));

          //connection between Modular_12 and 21
          //place connetion3
          TransformGroup con31 = createObject1 (scene14, 0.3f , -1.0f, 0.0f , 0.0f , 0.0f ,
0.0f , 0 );
          //set a rotation around Global Y axis
          objScale.addChild(createObject1x(con31, 0.0f , 0.0f, 0.0f , 0.0f , -1.0f,
0.0f ,r1 ));

          TransformGroup con32 = createObject1 (scene15, 0.4f , -0.88f, 0.0f , 0.0f , 0.0f ,
0.0f , 0 );
          //set a rotation around Global Y axis
          objScale.addChild(createObject1x(con32, 0.0f , 0.0f, 0.0f , 0.0f , -1.0f,
0.0f ,r1 ));

          //connection between Modular_32 and 41
          //place connection 4
          TransformGroup con41    = createObject1 (scene16, -.2f , 0.25f, 0.0f ,    0.0f ,
0.0f, 0.0f , 0);
          //rotate around Modular_32's Y axis
          TransformGroup con411    = createObject1x (con41, 0.0f , 0.0f, 0.0f ,    0.05f ,
0.3f, 0.0f , r3);
          //move with Modular_32
          TransformGroup con4111 = createObject2x (con411, 0.0f , 0.0f, 0.0f , 0.0f ,
-0.5f , 0.0f , r2 );
          objScale.addChild(createObject1x (con4111, 0.0f , 0.0f, 0.0f , 0.0f , -0.4f , 0.0f ,
r1 ));

          TransformGroup con42    = createObject1 (scene17, -.25f , 0.37f, 0.0f ,    0.0f ,
0.0f, 0.0f , 0);
          //rotate around Modular_32's Y axis
          TransformGroup con412    = createObject1x (con42, 0.0f , 0.0f, 0.0f ,    0.05f ,
0.3f, 0.0f , r3);
          //move with Modular_32
          TransformGroup con4112 = createObject2x (con412, 0.0f , 0.0f, 0.0f , 0.0f ,
-0.5f , 0.0f , r2 );
          objScale.addChild(createObject1x (con4112, 0.0f , 0.0f, 0.0f , 0.0f , -0.4f , 0.0f ,
r1 ));

          //Gripple1
          //place the Gripper1 and move to left
          TransformGroup gp11 = createObject3 (scene12, -0.05f , 1.35f, 0.0f ,    0.0f ,
0.0f, 0.0f , r5);
```

```
            //move with End Effector
            TransformGroup gp111 = createObject2x (gp11, 0.0f , 0.0f, 0.0f ,    0.05f , 0.7f,
0.0f , r4);
            TransformGroup gp1111 = createObject1x (gp111, 0.0f , 0.f, 0.0f ,    0.05f , 0.3f,
0.0f , r3);
            TransformGroup gp11111 = createObject2x (gp1111, 0.0f , 0.0f, 0.0f , 0.0f ,
-0.5f , 0.0f , r2 );
            objScale.addChild(createObject1x (gp11111, 0.0f , 0.0f, 0.0f , 0.0f , -0.4f , 0.0f ,
r1 ));

            //Gripper2
            //place the Gripper2 and move to right
           TransformGroup gp21 = createObject3 (scene13, 0.15f , 1.35f, 0.0f ,    0.0f , 0.0f,
0.0f , r6);
            //move with End Effector
            TransformGroup gp211 = createObject2x (gp21, 0.0f , 0.0f, 0.0f ,    0.05f , 0.7f,
0.0f , r4);
            TransformGroup gp2111 = createObject1x (gp211, 0.0f , 0.f, 0.0f ,    0.05f , 0.3f,
0.0f , r3);
            TransformGroup gp21111 = createObject2x (gp2111, 0.0f , 0.0f, 0.0f , 0.0f ,
-0.5f , 0.0f , r2 );
            objScale.addChild(createObject1x (gp21111, 0.0f , 0.0f, 0.0f , 0.0f , -0.4f , 0.0f ,
r1 ));

            objRoot.compile();
            return objRoot;
      }

      private TransformGroup createObject1(BranchGroup b1, float xpos1, float ypos1,
float zpos1,
                  float x1, float y1, float z1,float r    ){

            BoundingSphere bounds =    new BoundingSphere(new Point3d(0.0,0.0,0.0),
100.0);

            TransformGroup objTrans1 = new TransformGroup();
            objTrans1.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

            Transform3D t = new Transform3D();

            Vector3f lPos1 =    new Vector3f(xpos1,ypos1,zpos1);
            t.set(lPos1);
            TransformGroup l1Trans = new TransformGroup(t);
            objTrans1.addChild(l1Trans);

            TransformGroup objTrans2 = new TransformGroup(t);
```

```
        objTrans2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        l1Trans.addChild(objTrans2);
        objTrans2.addChild(b1);
        ////////////////////////////////
        Transform3D Axis2 = new Transform3D();
        Axis2.rotZ(Math.PI/.5f);
        Axis2.setTranslation(new Vector3f(x1, y1, z1));
        Alpha rotor2Alpha = new Alpha(1, Alpha.INCREASING_ENABLE ,2000,
0,4000, 0, 0, 0, 0, 0);
       RotationInterpolator rotator2 = new RotationInterpolator(rotor2Alpha, objTrans2,
Axis2, 0.0f, r);
        rotator2.setSchedulingBounds(bounds);
        objTrans2.addChild(rotator2);

        return objTrans1;
    }

    private TransformGroup createObject1x(TransformGroup b1, float xpos1, float
ypos1, float zpos1,
            float x1, float y1, float z1,float r    ){
        BoundingSphere bounds =    new BoundingSphere(new Point3d(0.0,0.0,0.0),
100.0);

        TransformGroup objTrans1 = new TransformGroup();
        objTrans1.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

        Transform3D t = new Transform3D();

        Vector3f lPos1 =    new Vector3f(xpos1,ypos1,zpos1);
        t.set(lPos1);
        TransformGroup l1Trans = new TransformGroup(t);
        objTrans1.addChild(l1Trans);

        TransformGroup objTrans2 = new TransformGroup(t);
        objTrans2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        l1Trans.addChild(objTrans2);
        objTrans2.addChild(b1);
        ////////////////////////////////
        Transform3D Axis2 = new Transform3D();
        Axis2.rotZ(Math.PI/.5f);
        Axis2.setTranslation(new Vector3f(x1, y1, z1));
        Alpha rotor2Alpha = new Alpha(1, Alpha.INCREASING_ENABLE ,2000,
0,4000, 0, 0, 0, 0, 0);
       RotationInterpolator rotator2 = new RotationInterpolator(rotor2Alpha, objTrans2,
Axis2, 0.0f,r);
        rotator2.setSchedulingBounds(bounds);
```

```
        objTrans2.addChild(rotator2);

        return objTrans1;
    }


    //Create second module，and x1,y1,z1 stand for the distance moved by the axis
    private TransformGroup createObject2(BranchGroup b1, float xpos1, float ypos1,
float zpos1,
                                float x1, float y1, float z1,float r    ){
        BoundingSphere bounds =    new BoundingSphere(new Point3d(0.0,0.0,0.0),
100.0);

        TransformGroup objTrans1 = new TransformGroup();
        objTrans1.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

        Transform3D t = new Transform3D();

        Vector3f lPos1 =    new Vector3f(xpos1,ypos1,zpos1);
        t.set(lPos1);
        TransformGroup l1Trans = new TransformGroup(t);
        objTrans1.addChild(l1Trans);

        TransformGroup objTrans2 = new TransformGroup(t);
        objTrans2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        l1Trans.addChild(objTrans2);
        objTrans2.addChild(b1);
        ////////////////////////////////
        Transform3D Axis2 = new Transform3D();
        Axis2.rotZ(Math.PI/2.0f);
        Axis2.setTranslation(new Vector3f(x1, y1, z1));
        Alpha rotor2Alpha = new Alpha(1, Alpha.INCREASING_ENABLE ,2000,
0,4000, 0, 0, 0, 0, 0);
       RotationInterpolator rotator2 = new RotationInterpolator(rotor2Alpha, objTrans2,
Axis2, 0.0f, r);
        rotator2.setSchedulingBounds(bounds);
        objTrans2.addChild(rotator2);

        return objTrans1;
    }

    //add one motion to the previous motion
    private TransformGroup createObject2x(TransformGroup b1, float xpos1, float
ypos1, float zpos1,
                                float x1, float y1, float z1,float r    ){
        BoundingSphere bounds =    new BoundingSphere(new Point3d(0.0,0.0,0.0),
```

```
100.0);

        TransformGroup objTrans1 = new TransformGroup();
        objTrans1.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

        Transform3D t = new Transform3D();

        Vector3f lPos1 =    new Vector3f(xpos1,ypos1,zpos1);
        t.set(lPos1);
        TransformGroup l1Trans = new TransformGroup(t);
        objTrans1.addChild(l1Trans);

        TransformGroup objTrans2 = new TransformGroup(t);
        objTrans2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        l1Trans.addChild(objTrans2);
        objTrans2.addChild(b1);
        ////////////////////////////////
        Transform3D Axis2 = new Transform3D();
        Axis2.rotZ(Math.PI/2.0f);
        Axis2.setTranslation(new Vector3f(x1, y1, z1));
        Alpha rotor2Alpha = new Alpha(1, Alpha.INCREASING_ENABLE ,2000,
0,4000, 0, 0, 0, 0, 0);
       RotationInterpolator rotator2 = new RotationInterpolator(rotor2Alpha, objTrans2,
Axis2, 0.0f,r);
        rotator2.setSchedulingBounds(bounds);
        objTrans2.addChild(rotator2);

        return objTrans1;
    }

    //create the horizontal motion
    private TransformGroup createObject3(BranchGroup b1, float xpos1, float ypos1,
float zpos1,
                                    float x1, float y1, float z1,float r    ){
        BoundingSphere bounds =    new BoundingSphere(new Point3d(0.0,0.0,0.0),
100.0);

        TransformGroup objTrans1 = new TransformGroup();
        objTrans1.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

        Transform3D t = new Transform3D();

        Vector3f lPos1 =    new Vector3f(xpos1,ypos1,zpos1);
        t.set(lPos1);
        TransformGroup l1Trans = new TransformGroup(t);
        objTrans1.addChild(l1Trans);
```

```
        TransformGroup objTrans2 = new TransformGroup(t);
        objTrans2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        l1Trans.addChild(objTrans2);
        objTrans2.addChild(b1);
        ////////////////////////////////
          Transform3D Axis3 = new Transform3D( );

       Alpha tranAlpha3 = new Alpha(-1, Alpha.INCREASING_ENABLE, 0, 0, 10000,
0, 0, 10000, 0, 0 );
        PositionInterpolator pos4 = new PositionInterpolator(tranAlpha3, objTrans2,
Axis3, 0.0f, r);
        pos4.setSchedulingBounds(bounds);
        objTrans2.addChild(pos4);

        return objTrans1;
    }


    public clientGui(){

        getContentPane().setLayout(null);
        getContentPane().add(txt1);
        getContentPane().add(txt2);
        getContentPane().add(txt3);
        getContentPane().add(txt4);
        getContentPane().add(txt5);
        getContentPane().add(txt6);
        getContentPane().add(txt7);
        getContentPane().add(txt8);

        b1.addActionListener(bl1);
        b2.addActionListener(bl2);
        b3.addActionListener(bl3);
        b4.addActionListener(bl4);
        b5.addActionListener(bl5);
        b6.addActionListener(bl6);
        b7.addActionListener(bl7);

        getContentPane().add(b1);
        getContentPane().add(b2);
        getContentPane().add(b3);
        getContentPane().add(b4);
        getContentPane().add(b5);
        getContentPane().add(b6);
        getContentPane().add(b7);
```

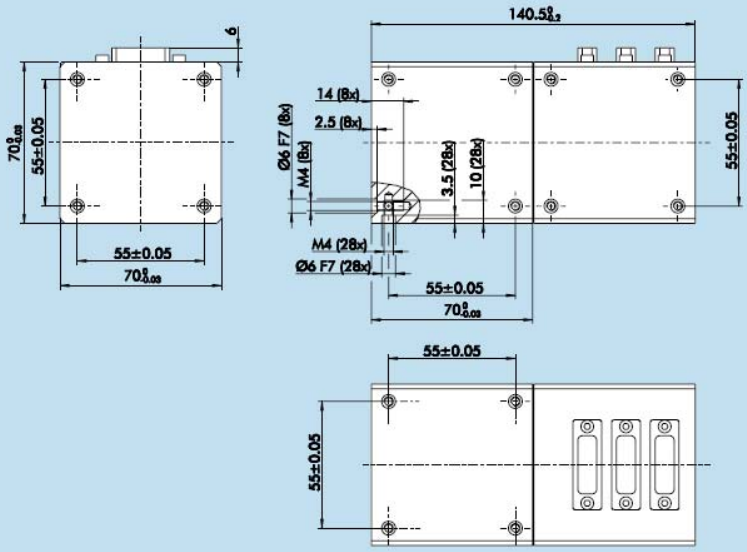
```
        getContentPane().add(lb1);
        getContentPane().add(lb2);
        getContentPane().add(lb3);
        getContentPane().add(lb4);
        getContentPane().add(lb5);
        getContentPane().add(lb6);
        getContentPane().add(lb7);
        getContentPane().add(lb8);

        txt1.setBounds(100, 100, 40, 20);
        txt2.setBounds(150, 100, 40, 20);
        txt3.setBounds(200, 100, 40, 20);
        txt4.setBounds(140, 200, 40, 20);
        txt5.setBounds(140, 230, 40, 20);
        txt6.setBounds(140, 260, 40, 20);
        txt7.setBounds(140, 290, 40, 20);
        txt8.setBounds(140, 320, 40, 20);

        b1.setBounds(100, 150, 80, 20);
        b2.setBounds(200, 370, 80, 20);
        b3.setBounds(100, 450, 80, 20);
        b4.setBounds(100, 410, 80, 20);
        b5.setBounds(200, 410, 80, 20);
        b6.setBounds(100, 370, 80, 20);
        b7.setBounds(200, 450, 80, 20);

        lb1.setBounds(110, 80, 40, 20);
        lb2.setBounds(160, 80, 40, 20);
        lb3.setBounds(210, 80, 40, 20);
        lb4.setBounds(100, 200, 40, 20);
        lb5.setBounds(100, 230, 40, 20);
        lb6.setBounds(100, 260, 40, 20);
        lb7.setBounds(100, 290, 40, 20);
        lb8.setBounds(100, 320, 40, 20);

    }
    public static void main(String[] args){
        clientGui f = new clientGui();
        f.setTitle("clientGui");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(800, 600);
        f.setVisible(true);
    }

}
```

Appendix C: PowerCube PR70

PR 070

Maße/Technische Daten
Dimensions/Technical data

amtec robotics



PR 070

Modultyp	PR 070		Module type
Motordaten			**Motor data**
Nenndrehmoment	0.181 Nm	0.181 Nm	Nominal torque
Max. Drehmoment	0.565 Nm	0.565 Nm	Max. torque
Nenndrehzahl	5000 min^{-1}	5000 min^{-1}	Nominal R.P.M.
Spannung	24 ± 1 V	24 ± 1 V	Voltage
Max. Strom	15 A	15 A	Max. current
Encoder			**Encoder**
Auflösung [Inc/Umdrehung]	2000	2000	Resolution [Inc/Rotation]
Getriebedaten			**Gear data**
Übersetzung	161:1	101:1	Ratio
Nenndrehmoment	40 Nm	40 Nm	Nominal torque
Durchschnittliches Drehmoment	49 Nm	49 Nm	Average torque
Wiederholbares Spitzenmoment	92 Nm	87 Nm	Repeatable peak torque
Max. Antriebsdrehzahl	4000 min^{-1}	4000 min^{-1}	Max. input R.P.M.
Wirkungsgrad	0.8	0.8	Efficiency
Moduldaten			**Module data**
Abtriebs-Nennmoment	23 Nm	15 Nm	Nominal output torque
Max. Abtriebsmoment	73 Nm	46 Nm	Max. output torque
Max. Winkelgeschwindigkeit	149 °/sec	238 °/sec	Max. angular velocity
Resolution [Winkelsec/Inc]	4	6	Resolution [Arcsec/Inc]
Lageerfassung	894 Inc/°	561 Inc/°	Resolution
Wiederholgenauigkeit der Positionierung	± 0,02°	± 0,02°	Repeatability of positioning
Aktionsbereich mit Endlagenschalter	± 160°	± 160°	Positioning w/ limit switches
Aktionsbereich ohne Endlagenschalter	3335 ± n*360°	5316 ± n*360°	Positioning w/o limit switches
Masse	1.7 kg	1.7 kg	Weight

Weitere Getriebeübersetzungen auf Anfrage — **Other gear ratio on request**

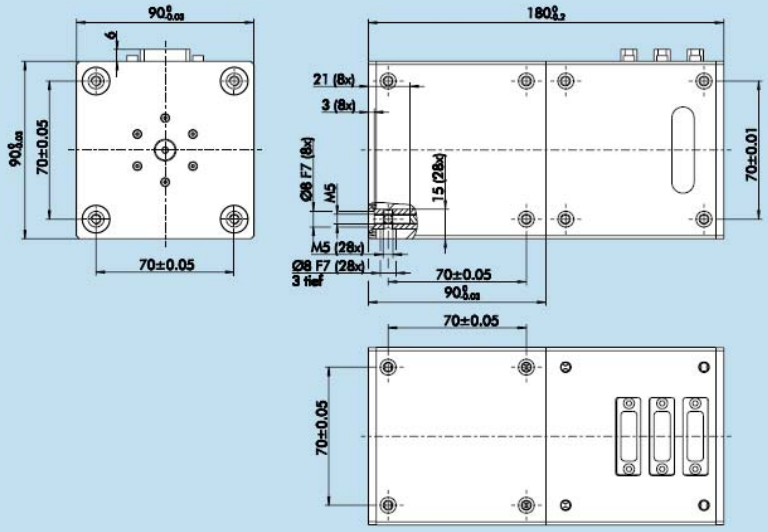
Appendix D: PowerCube PR90

PR 090

Maße/Technische Daten
Dimensions/Technical data

POWERCUBE

PR 090



Modultyp	PR 090		Module type
Motordaten			**Motor data**
Nenndrehmoment	0.558 Nm	0.558 Nm	Nominal torque
Max. Drehmoment	1.6 Nm	1.6 Nm	Max. torque
Nenndrehzahl	4300 min^{-1}	4300 min^{-1}	Nominal R.P.M.
Spannung	24 ± 1 V	24 ± 1 V	Voltage
Max. Strom	30 A	30 A	Max. Current
Encoder			**Encoder**
Auflösung [Inc/Umdrehung]	2000	2000	Resolution [Inc/Rotation]
Getriebedaten			**Gear data**
Übersetzung	161:1	101:1	Ratio
Nenndrehmoment	67 Nm	67 Nm	Nominal torque
Durchschnittliches Drehmoment	108 Nm	108 Nm	Average torque
Wiederholbares Spitzenmoment	176 Nm	167 Nm	Repeatable peak torque
Max. Antriebsdrehzahl	4000 mirr^{-1}	4000 mirr^{-1}	Max. input R.P.M.
Wirkungsgrad	0.8	0.8	Efficiency
Moduldaten			**Module data**
Abtriebs-Nennmoment	72 Nm	45 Nm	Nominal output torque
Max. Abtriebsmoment	206 Nm	129 Nm	Max. output torque
Max. Winkelgeschwindigkeit	149 °/sec	238 °/sec	Max. angular velocity
Resolution [Winkelsec/Inc]	4	6	Resolution [Arcsec/Inc]
Lageerfassung	894 Inc/°	561 Inc/°	Resolution
Wiederholgenauigkeit der Positionierung	± 0,02°	± 0,02°	Repeatability of positioning
Aktionsbereich mit Endlagenschalter	± 160°	± 160°	Positioning w/ limit switches
Aktionsbereich ohne Endlagenschalter	3335 ± n*360°	5316 ± n*360°	Positioning w/o limit switches
Masse	3.2 kg	3.2 kg	Weight

Weitere Getriebeübersetzungen auf Anfrage — **Other gear ratio on request**

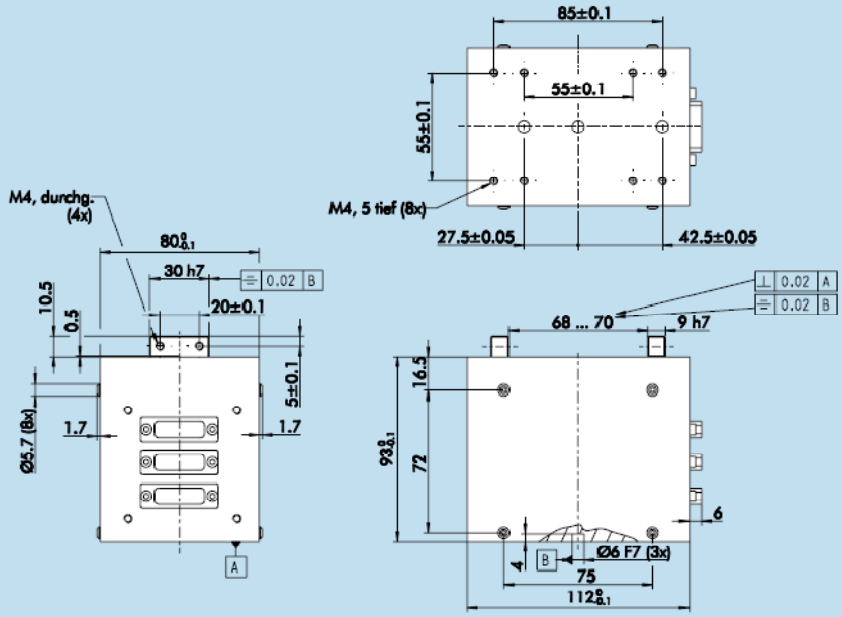
Appendix D: PowerCube PG70

PG 070

Maße/Technische Daten
Dimensions/Technical data

amtec robotics

PG 070



Modultyp	PG 070	Module type
Motordaten		**Motor data**
Nenndrehmoment	0.028 Nm	Nominal torque
Max. Drehmoment	0.1 Nm	Max. torque
Nenndrehzahl	14400 min^{-1}	Nominal R.P.M.
Encoder		**Encoder**
Auflösung [Inc/Umdrehung]	2000	Resolution [Inc/Rotation]
Moduldaten		**Module data**
Max. Greifkraft	200 N	Max. gripping force
Max. Geschwindigkeit	70 mm/sec	Max. velocity
Auflösung	0.25 µm/Inc	Resolution
Wiederholgenauigkeit der Position	0.05 mm	Repeatability of positioning
Hub pro Backe	35 mm	Stroke per jaw
Masse	1.4 kg	Weight