

11-22-2004

Design and development of a one-degree-of-freedom force-reflecting manual controller prototype for teleoperation

Chandrasekar Reddy Puligari

Florida International University, chandu_puligari@yahoo.com

DOI: 10.25148/etd.FI08081538

Follow this and additional works at: <https://digitalcommons.fiu.edu/etd>

Recommended Citation

Puligari, Chandrasekar Reddy, "Design and development of a one-degree-of-freedom force-reflecting manual controller prototype for teleoperation" (2004). *FIU Electronic Theses and Dissertations*. 39.

<https://digitalcommons.fiu.edu/etd/39>

This work is brought to you for free and open access by the University Graduate School at FIU Digital Commons. It has been accepted for inclusion in FIU Electronic Theses and Dissertations by an authorized administrator of FIU Digital Commons. For more information, please contact dcc@fiu.edu.

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

DESIGN AND DEVELOPMENT OF A ONE-DEGREE-OF-FREEDOM
FORCE- REFLECTING MANUAL CONTROLLER PROTOTYPE FOR
TELEOPERATION

A thesis submitted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

in

MECHANICAL ENGINEERING

by

Chandrasekar Reddy Puligari

2004

To: Dean Vish Prasad
College of Engineering

This thesis, written by Chandrasekar Reddy Puligari, and entitled Design and Development of a One-Degree-of-Freedom Force-Reflecting Manual Controller Prototype for Teleoperation, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this thesis and recommend that it be approved.

Ibrahim N. Tansel

Diana M. Rincón

Sabri Tosunoglu, Major Professor

Date of Defense: November 22, 2004

The thesis of Chandrasekar Reddy Puligari is approved.

Dean Vish Prasad
College of Engineering

Dean Douglas Wartzok
University Graduate School

Florida International University, 2004

DEDICATION

I dedicate this thesis to my parents Mohan Reddy and Sukeshini without whom this work would not have been completed and to my brother Rajashekar Reddy who has been encouraging and supporting me. Without their patience, support, and most of all love, the completion of this work would not have been possible. I love you all.

ACKNOWLEDGMENTS

I wish to thank my committee members Drs. Sabri Tosunoglu, Ibrahim Tansel and Diana Rincon for their support and patience. Especially, I would like to thank Professor Sabri Tosunoglu for advising and supporting me throughout the course of my research. Their guidance has been most helpful and appreciated.

My thanks are also extended to Dr. Daniel W. Repperger, Program Manager and Technical Contact for the U.S. Air Force grant (Grant No: 571859500 and Panther Soft Grant No: 212600507) supporting me throughout my graduate studies.

Furthermore, I would also like to thank my friends and peers Jordi Blanch, Andre Senior, Can (John) Dede, Ricardo Garcia, Satya, and Ravi K. Dowluri for their support and help in developing the software and conducting experiments.

ABSTRACT OF THE THESIS

DESIGN AND DEVELOPMENT OF A ONE-DEGREE-OF-FREEDOM FORCE-REFLECTING MANUAL CONTROLLER PROTOTYPE FOR TELEOPERATION

by

Chandrasekar Reddy Puligari

Florida International University, 2004

Miami, Florida

Professor Sabri Tosunoglu, Major Professor

The present research is carried out from the viewpoint of primarily space applications where human lives may be in danger if they are to work under these conditions. This work proposes to develop a one-degree-of-freedom (1-DOF) force-reflecting manual controller (FRMC) prototype for teleoperation, and address the effects of time delays commonly found in space applications where the control is accomplished via the earth-based control stations.

To test the FRMC, a mobile robot (PPRK) and a slider-bar were developed and integrated to the 1-DOF FRMC. The software developed in Visual Basic is able to telecontrol any platform that uses an SV203 controller through the internet and it allows the remote system to send feedback information which may be in the form of visual or force signals. Time delay experiments were conducted on the platform and the effects of time delay on the FRMC system operation have been studied and delineated.

TABLE OF CONTENTS

CHAPTER	PAGE
I: INTRODUCTION	
1.1	Introduction..... 1
1.2	Teleoperation..... 2
1.3	Telerobot..... 3
1.4	Time delay..... 4
1.5	Thesis objective..... 5
II: BACKGROUND AND DESIGN OF MANUAL CONTROLLERS	
2.1	History..... 7
2.2	Force-reflecting manual controller (FRMC) system..... 10
2.3	Conceptual designs of manual controllers..... 15
2.3.1	FRMC with ball screw mechanism..... 16
2.3.2	FRMC with gear system..... 17
2.3.3	FRMC with belt system..... 18
2.3.4	FRMC with direct drive mechanism..... 19
2.3.5	FRMC with drive roller and belt system..... 21
III: ACTUATORS AND SENSORS	
3.1	Actuators 22
3.1.1	Hydraulic..... 22
3.1.2	Pneumatic..... 24
3.1.3	Electric..... 25
3.1.4	Selection..... 27
3.2	Sensors..... 29
3.2.1	Proximity..... 30
3.2.2	Range..... 30
3.2.3	Touch..... 31
3.2.4	Force/torque..... 31
3.2.5	Potentiometer..... 32
3.2.6	Vision..... 32
3.2.7	Tachometer..... 33
3.2.8	Selection..... 33
IV: MICROCONTROLLERS	
4.1	Mini board..... 37
4.2	Handy board..... 38
4.3	Bot board..... 39

4.4	MTJPro11.....	40
4.5	Basic stamp.....	41
4.6	Other microcontrollers.....	42
4.7	Servo controllers.....	43
4.7.1	Brainstem servo controller.....	43
4.7.2	Pololu servo controller.....	44
4.7.3	USB servo controller.....	45
4.7.4	Selection: Pontech SV203 servo controller.....	46
V: FRMC AND PLATFORM DESIGN		
5.1	Force-reflecting manual controller system design.....	48
5.2	Power supply and pin out.....	51
5.3	Housing.....	52
5.4	Slider bar platform design.....	53
5.5	Palm pilot robot kit platform design.....	54
5.5.1	Platform design modification.....	55
VI: ROBOT CONTROL SOFTWARE DEVELOPMENT AND SYSTEM INTEGRATION		
6.1	Robot control software.....	57
6.2	1-DOF FRMC Prototype simulation with mouse.....	59
6.3	1-DOF FRMC Prototype simulation with SV203 controller.....	60
6.4	Time delay in space applications.....	62
6.5	Time delay experiments conducted with 1-DOF FRMC.....	64
6.6	System integration.....	66
6.7	Winsock control.....	68
6.7.1	TCP.....	68
6.7.2	UDP.....	69
6.7.3	Winsock properties.....	69
6.7.4	Winsock methods.....	70
6.8	Operating the system software.....	71
6.8.1	Video streaming.....	73
VII: RESULTS AND DISCUSSIONS		
7.1	Conclusions.....	76
7.2	Recommendations.....	78
7.3	Future work.....	78
REFERENCES.....		80
APPENDICES.....		83

LIST OF FIGURES

FIGURE	PAGE
Figure 1.1 Telerobotic system block diagram.....	3
Figure 1.2 Telerobot working on mars.....	4
Figure 2.1 FRMC concept utilizing ball screw mechanism.....	16
Figure 2.2 FRMC concepts with gear reduction.....	18
Figure 2.3 FRMC concepts with belt system.....	19
Figure 2.4 FRMC concepts utilizing direct drive system.....	20
Figure 2.5 Roller drive and belt joystick.....	21
Figure 3.1 Hydraulic actuator.....	23
Figure 3.2 View of pneumatic actuator.....	24
Figure 3.3 Servo timing diagram.....	28
Figure 3.4 Futaba servo motor.....	28
Figure 3.5 Non-linear graph.....	34
Figure 3.6 GP2D12 sensor.....	34
Figure 3.7 Distance through triangulation.....	35
Figure 3.8 Internal block diagram.....	35
Figure 4.1 PIC microcontroller.....	36
Figure 4.2 Mini board.....	38
Figure 4.3 Handy board.....	38
Figure 4.4 Bot board.....	39
Figure 4.5 MTJPro11 microcontroller.....	40
Figure 4.6 Basic stamp.....	41

Figure 4.7	AT89C2051 microcontroller.....	42
Figure 4.8	OOBOARD	43
Figure 4.9	Brainstem controller.....	44
Figure 4.10	Pololu servo controller.....	45
Figure 4.11	USB servo controller.....	45
Figure 4.12	SV203 controller.....	46
Figure 5.1	Direct drive mechanism.....	49
Figure 5.2	Connection of servo with SV203.....	50
Figure 5.3	Pin out diagram.....	51
Figure 5.4	Connection of potentiometer with SV203.....	52
Figure 5.5	1-DOF FRMC developed.....	52
Figure 5.6	Slider bar platform.....	53
Figure 5.7	PPRK designed at CMU.....	54
Figure 5.8	PPRK modified mobile platform	56
Figure 6.1	Form view when robot on the left is away from the obstacle represented by the plane icon.....	59
Figure 6.2	Form view when robot is close to the obstacle.....	60
Figure 6.3	GUI developed to control SV203.....	61
Figure 6.4	GUI of visual basic program.....	62
Figure 6.5	Round-trip communication time delays between the earth and planets...	63
Figure 6.6	Time delay trend for different users identified as series 1 through 5.....	65
Figure 6.7	Process of the system developed.....	72
Figure 6.8	Camera attached to the PPRK.....	74

Figure 6.9 Webcam at client.....	75
Figure 6.10 Webcam at server.....	75

CHAPTER 1

1.0 INTRODUCTION

1.1 Introduction

The term robot is defined as a mechanical device that sometimes resembles a human being and is capable of performing a variety of often complex human tasks on command or by being programmed in advance. The term robota or rather robot means tedious labour in czech and was first used in Czechoslovakia by Karel Capek in a play called Rossum's Universal Robots (R.U.R) around 1920 [1]. Americans were the first to develop a programmable robot which was used to spray paint. This programmable robot was developed by Willard Pollard and Harold Roselund for a company named Devilbiss in 1938. In 1942, Isaac Asimov published three laws of robotics in a short story called "Runaround" [2]. The three laws of robotics are stated as follows:

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey orders given it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

The first teleoperated articulated arm was developed by Raymond Goertz in France around 1951 for the Atomic Energy Commission. The design is based entirely on mechanical coupling between the master and slave arms (using steel cables and pulleys). Derivatives of this design are still seen in places where handling of small nuclear samples

is required. This is generally regarded as the major milestone in force feedback technology. The pioneering work by George Devol with the help of Joseph Engelberger led to the development of a programmable robot for industrial applications through the company named Unimation in 1954. This robot model was developed only for the basic use like transfer objects from one point to another, less than a dozen feet apart. The first industrial robot from Unimation was installed in the production line of General motors company around 1962. Day by day the use of robots became more popular but they also have a disadvantage in 60's and 70's that they are a bit expensive, as the time passed by, they gradually became inexpensive due to their use in daily life. Car manufacturing companies have used these robots more than anyone else in those days for improving their production. In the present world the applications of the robots have been extended to laboratories, research and space exploration sites, energy plants, hospitals, toy industries and even to outer space.

1.2 Teleoperation

Teleoperation plays a vital role in the field of robotics. Teleoperation system usually consists of two robot manipulators that are connected in such a way as to allow the human operator controls one of the master arm (one of the manipulator) to generate commands which sends commands to the slave arm (remote manipulator) which are kept at some distance apart. The commands from the master robot are sent to the slave robot by many different ways. Teleoperation has very wide range of application in the field of medicine, space exploration, military, undersea and hazardous applications etc. Perhaps the most common application of this technique is in the field of space

exploration. In a typical teleoperation system, the operator (master robot) receives feedback information from the slave robot in any of the forms like audio, visual, and force [3]. Fig. 1.1 shows the basic principle behind a telerobotic system. This teleoperation is very useful in dangerous applications like nuclear waste cleanup sites where a human cannot fulfill the desired task.

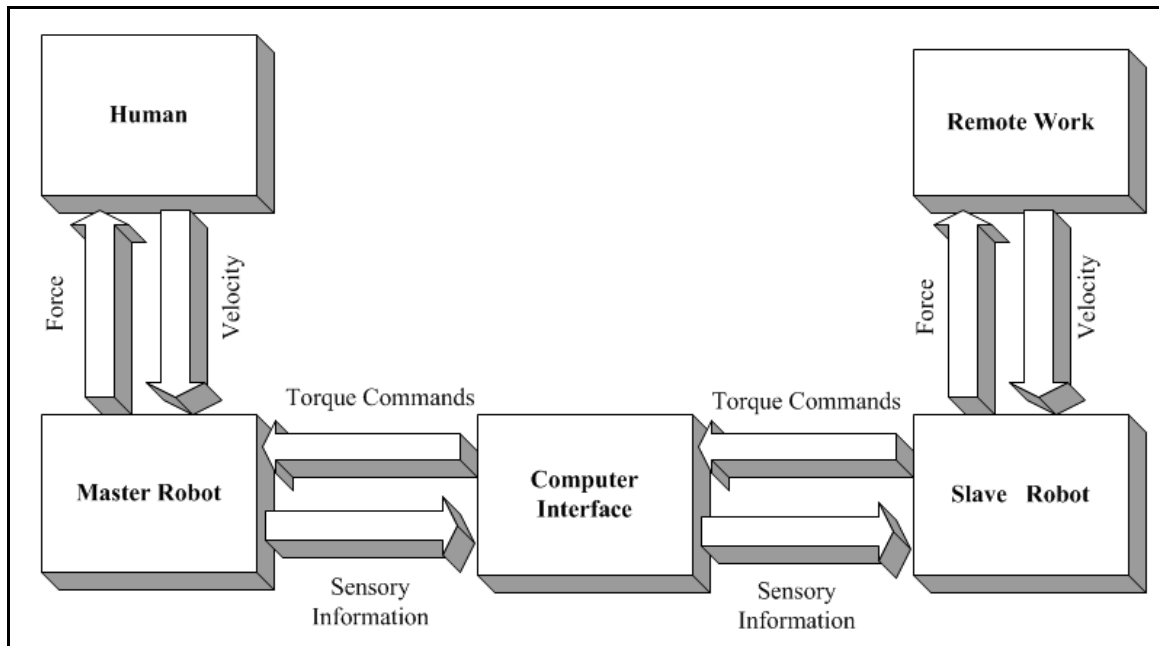


Figure 1.1 Telerobotic system block diagram

1.3 Telerobot

Latest technology in the field of robotics and automation has replaced many jobs that are repetitive in nature. However, there are many tasks that are non-repetitive, unpredictable and hazardous to human health. These tasks have to be performed by a remote manipulator, otherwise known as a telerobot. Telerobotics refers to the use of

robots “at a distance”. Latest development in the field of telerobot is the exploration of mars by National Aeronautics and Space Administration (NASA) is shown in fig 1.2. The applications of telerobots are wide spread in today’s world such as remote control for televisions, music systems and also remote controls for garage doors etc.



Figure 1.2 Telerobot working on mars

1.4 Time delay

In a teleoperation system there are two robots. A human operator moves and controls the local robot, also known as the input or master joystick. Motion commands are measured on this device and transmitted to remote location. The second robot, called slave, executes these commands and tries to track the input device. In many of the cases the time taken to execute the commands of the master robot by the slave robot are delayed due to some intermediate delays, these are called constant time delays. When Internet is used as the communication medium connecting the master and slave manipulators, where

transmission delays are variable. For teleoperation over the Internet the delay varies with such factors as congestion, bandwidth, or distance, and these varying delays may severely degrade performance or even result in an unstable system. There has to date been relatively little research on this problem [18].

1.5 Objectives of the current research

One of the desired objectives in the development of the teleoperators systems is to design and develop a system that provides the operator with the sensation feedback. A force-reflecting manual controller is one of the components in the system that provides the operator with force feedback. Unfortunately, most of the manual controllers are large, bulky, complicated and expensive. For the system to be used in practical situation, the system must be portable, compact, lightweight, easy to use and easy to manufacture. Other requirements include large workplace and sufficient force reflection.

In this work, the design and development of a portable force reflecting manual controller is addressed as a design problem. This includes conceptual design of the system, construction of the testbed, software development, testing, and laboratory demonstration.

In order to satisfy above requirements, the previous works have been investigated for mechanical and control design. In addition, a survey of mechanical components such as actuator systems and sensors has been conducted to identify compact size, light weight, and high performance components. The proposed conceptual designs for a one degree of freedom (DOF) force reflecting manual controller (FRMC) have been shown and the developed prototype has been demonstrated on a 1-DOF testbed.

The contents of each chapter are summarized as follows:

Chapter 1: History of robots is briefly described and the concept of teleoperators and the telesensation systems is introduced. Then, the objectives and the scope of the research work are outlined.

Chapter 2: Background on force-reflecting teleoperation systems is presented by first listing the history of the previous work. Conceptual designs of 1-DOF FRMC developed at FIU are discussed.

Chapter 3: The components of FRMC like actuators, sensors and potentiometers are discussed and the selection of these components are tabulated.

Chapter 4: The important component of a FRMC teleoperator system SV203 microcontroller and the other microcontrollers considered for this project are briefly described.

Chapter 5: The design and development of 1-DOF FRMC for teleoperation prototype and a platform (Palm Pilot Robot Kit) was developed to test the 1-DOF FRMC is discussed.

Chapter 6: The software developed to control this 1-DOF FRMC prototype is discussed. The graphical user interface and the simulation results are shown. The software developed is updated to control the joystick over the internet to test the teleoperation. The system networking is presented in this chapter.

Chapter 7: The conclusions and recommendations derived from this work, and future work to be conducted are presented.

CHAPTER 2

2.0 BACKGROUND AND DESIGN OF MANUAL CONTROLLERS

Telem manipulator is a device which allows an operator to perform a task at a distance. Usually, a teleoperator system consists of one or more telem manipulators, task and environmental sensing systems, and a human machine interface. The applications of these telem manipulators include hazardous radioactive environment, underwater, space, military applications where human access is impossible.

In this chapter, previous work done by the researchers in the field of telem manipulator design will be described, and some of the previous teleoperation system designs are reviewed.

2.1 History

- In 1945, Ray Goertz developed the first manual controller at Argonne National Laboratory (ANL). The first developed manual controller does not include force reflecting feature in it but later they have implemented it.
- In 1949, the first Master-Slave Manipulator (MSM) was developed by Ray Goertz and his team at ANL of the U.S. Atomic Energy Commission. It was given the name Model 1; this bilateral manual controller uses electrical and mechanical systems to penetrating through the ceiling of a shielded cell. Maximum handling capacity was only 5 kgs.
- In 1953, there was a need to design a manipulator with high handling capacities, which inturn resulted in developing Model 8 manipulator with a handling capacity of

9 kgs. This was developed by Central Research Laboratory (CRL) under the supervision of ANL. CRL has manufactured 60 – 70% of the existing Model 8 manipulators.

- In 1954, Goertz and Thompson have replaced the direct mechanical linkage and cable linkages with electrical servomechanisms and developed a closed circuit television, which resembles a teleoperation where the operator could be an arbitrary distance away.
- In 1960, CRL has developed Model D manipulator. It is the first manipulator used for some industrial application with higher load capacity after all the models developed for only laboratory application. The improved versions of these Model D manipulators are Model E and Model F.
- In 1960, R. Mosher developed a Handy man at General Electric. This is a force reflecting manual controller which has two electro hydraulic arms with ten degrees of freedom (each finger has two degrees of freedom).
- In 1961, Aaron Kobrinskii in Moscow was the first to develop the new servomechanism to human limb prostheses. He developed a lower-arm prosthesis driven by minute electric signals picked up from muscles in the stump or upper arm.
- In mid 1960's, Researches in United States and Europe have made similar developments as Aaron Kobrinskii. They developed teleoperators attached to the wheelchairs of quadriplegics which can be commanded by tongue.
- In 1966, Strickler was the first to develop teleoperator with teletouch.
- In 1966, telemanipulators and video cameras were attached to submarines by US navy.

- In 1967, Ferrell and Sheridan developed the first teleoperator with time delay.
- In 1969, NASA developed a space teleoperator for space shuttle, remote manipulator system (RMS) which is 20 mts long.
- In late 1960's, Telediagnosis was developed between a general hospital in Boston and Logan Airport by Murphy, R.L. Jr. and Bird K.T.
- In 1970's, Remotely Operated Vehicles have been developed for undersea applications by Vadus (1976), Yastrebov and Stepanov (1978), and Busby (1979).
- In 1981, M-2 Maintenance system and Advanced Servo manipulator (ASM) master-slave force reflecting manual controller for a nuclear plant was developed by Oak Ridge National Laboratory.
- In 1980's, Whitney at Draper Laboratory at Massachusetts Institute of Technology has developed handy controller.
- In 1985, Tesar and Tosunoglu at University of Texas, Austin have developed both 6-DOF force reflecting manual controller and a 3-DOF spherical force reflecting shoulder controller.
- In 1993, Massachusetts Institute of Technology Artificial Intelligence Laboratory has developed a virtual force feedback master controller PHANToM which was made commercially available.
- In 1993, EXOS Inc. developed commercially available force reflecting manual controller named SAFiRE (Sensing and Force Reflecting Exoskeleton).
- In 1996, Virtual Technologies Inc. developed a force feedback glove, where force reflection was experienced by each finger.

- In 1998, Batsomboon and Tosunoglu have developed a portable force-reflecting manual controller for teleoperation at Florida International University.
- In 1999, a pen based force feedback device PenCAT was developed by Haptic Technologies Inc.
- In 2000, Michael Goldfarb at Vanderbilt University has developed 3-DOF haptic interface in Centre for Intelligent Mechatronics.
- In 2002, Howe and Brockett at Harvard Robotics Lab have developed a HRL manipulator which has 2 two link fingers, each with 3-DOF.
- In 2004, Kamerkar and Kesavdas at University of Buffalo developed a touch based interactive NURBS modeler using a force/position input glove.
- In 2004, Niemeyer, and Kuchenbecker at Stanford University developed a THUMP force reflecting device.

2.2 Force-reflecting manual controller system

Force Reflecting Manual controllers are used in telerobotic applications to control remote robots by local human operators. The applications include space operations, undersea activities, nuclear site cleanup, and microsurgery. Usually the force-reflecting manual controller system consists of two robots, slave and a master. The operator uses the manual controller (master) to control the other robot (slave). The slave robot receives the signal from the master through various communication channels. When the master and slave robots are operated in two different locations visual information from a video image and/or graphical information from the computer screen or the sensory feedback are used to improve the efficiency of the manual controller. When the input

commands are fed to the master, remote system moves accordingly and the force experienced by the system are reflected at the manual controller, which shows that the system is a force reflecting teleoperating system [15].

These force reflecting teleoperating systems are more helpful in a situation where human cannot complete the task due to some hazardous environments, and where the visual information of the remote site cannot be obtained. In such environments, this manual controller can be used effectively to complete the required task. The operator with force reflection tends to make fewer attempts to complete the given task than the operator without force reflection [4].

Some of the well known force reflecting manual controllers or Haptics developed as discussed below:

- **PHANToM**

PHANToM is the force reflecting desktop device developed by the MIT researcher Massie, a 3D input device which can be operated by the finger tip [5]. This is a 3-DOF force reflecting device that provides force reflection to the human operator when it's connected to the computer interface. This FRMC enables the user to feel the virtual reality environment. Different models of this PHANToM have designed like Desktop PHANToM device and PHANToM Omni device. These are considered to be high precision instruments and also some of these are 6-DOF controllers. This PHANToM offers high fidelity, stronger forces and low friction. This device produces a maximum force reflection of 7.9 N and a continuous force of 1.75 N [6].

- **Rutgers master II ND**

This device is a force reflecting haptic interface which uses a glove to cover the fingers of a human instead of using fingers directly. This haptic interface has all the sensing devices on the glove avoiding routing wires to fingertips as in earlier devices. This uses the direct drive mechanism to feel the force reflection. This device uses the pneumatic actuator which is arranged in direct drive mechanism to the fingers. The glove which is covering the fingers will feel the force reflection after receiving the commands from the sensors attached to it, resulting in better force control [7].

- **GROPE-III**

The University of North Carolina (UNC) researches have modified the mechanical controller of a robotic arm previously used in radioactive material handling. Motors were added to the Argonne remote manipulator (ARM). The motors would be activated by the virtual environment to create forces on the controller. UNC used this in their GROPE-III system to help chemist "feel" the attractive and resistive forces of molecules reacting and bonding to each other. The chemist could use the forces and torques to learn how to make new chemical compounds.

- **Exoskeleton**

In 1988, University of Utah with collaboration with EXOS Company has developed a device for telerobotic application where [8], an operator can strap their arm into a large 50 pound exoskeleton developed to deliver force feedback. The systems are used as a kind of master-slave combination, and forces are applied by motors at the joints. The operator's arm and hand can move in 10 different ways at the same time.

The computer constantly changes the force output of the motors and hydraulic actuators on the exoskeleton so that it can feel essentially weightless. However, when the operator touches something, the virtual forces become actual forces felt through the exoskeleton. They could feel the increasing resistance of compressing a virtual spring or their arm would stop hard when it reached a virtual wall. . Unfortunately, these devices are usually very heavy; therefore they can also be used in special applications. EXOS, Inc. has developed a ``light" version for the NASA, but this system does not have any force feedback.

- **Force-reflecting hand controller**

A force reflective hand controller (FRHC) is being developed to control a robot arm. When the robot arm picks up an object or pushes something, those forces and torques are felt by the operator. The operator can now tell how much strain is placed on the robot's motors, gears, and structure. Also this controller gives the operator a partial sense of touch that the robot would have.

- **Touch based interactive NURBS modeler**

The creation of complex NURBS surfaces in design environments is a tedious process because very few tools exist, that allow a designer to design intuitively in real-time; to overcome this problem researchers at University at Buffalo (UB) have developed a device. Here in this design the human hand is perhaps the most useful and diverse tool used to interact with the environment and us. The CAD Modeling Glove (ModelGlove), originally developed by Mayrose (Mayrose et al. 2000) in the Virtual Reality Lab. At UB was adapted as an innovative new interface between the real world and the 3D NURBS model, by capturing the action of the user's hand including pressure and position of the

fingers. This input device is equipped with force and position sensors for quantifying touch and intent of the designer. The goal behind the development of the ModelGlove is to provide designers with a tool, which will allow them to touch, push, and manipulate virtual objects, just as they would model real clay models or sculptures [9].

- **Pen based force display**

The pen based force display is a direct drive mechanism haptic device developed by University of Washington researchers. This device is a parallel, actuation redundant, two degree-of-freedom haptic device, designed to provide force feedback information generated by either a master-slave system or a virtual simulation. The operator interacts with it using either the fingertip or a freely held pen-like tool. This device could provide force feedback for applications such as micro-surgery and telemanipulation, or serve as an input device to characterize the human's finger impedance, or serve as a generic virtual reality mechanical interface [10].

- **Two-hand universal master project (THUMP)**

THUMP is a 3D manipulator used to operate the slave robot for precise robotic applications, developed by the Stanford researchers. This device consists of two hands operated structures resembling a human hand and fingers working in a 3D environment, which will be helpful to be considered as a robot working in virtual reality environment. Each arm of the robot manipulator has seven degrees of freedom to serve as a haptic interface. The two handed haptic capability will provide force and torque feedback identical to those experienced from the remote site. This device is will be much helpful in surgical applications [11].

- **Portable dextrous master**

The Portable Dextrous Master (PDM) is a force feedback system for delicate hand motions. It was designed and manufactured through a collaboration between Sarcos Inc., the Center of Engineering Design at Utah and the Artificial Intelligence Lab at MIT. It is designed to be used with the DataGlove. The DataGlove detects finger movement to the computer which then sends signals to the PDM which, in turn, supplies force feedback in the thumb, forefinger, and middle fingers [12].

2.3 Conceptual designs of one-degree-of-freedom force-reflecting manual controller

Many design constraints arise from consideration of the mechanical interaction of the human with the haptic interface. If the device is designed to be manipulated by the user's fingers, then the design parameters will be related to the capacity of a typical human finger. Interface size and appearance will also play vital roles in haptic interface design. In particular we want users to be instantly comfortable with our design. We believe that we can satisfy this goal with a 1 D.O.F device whose size and motion are similar to that of a PC mouse. The next section will consider how four interface proposals accomplish the design goals mentioned above.

Interface designs

2.3.1 FRMC Concept utilizing ball screw mechanism

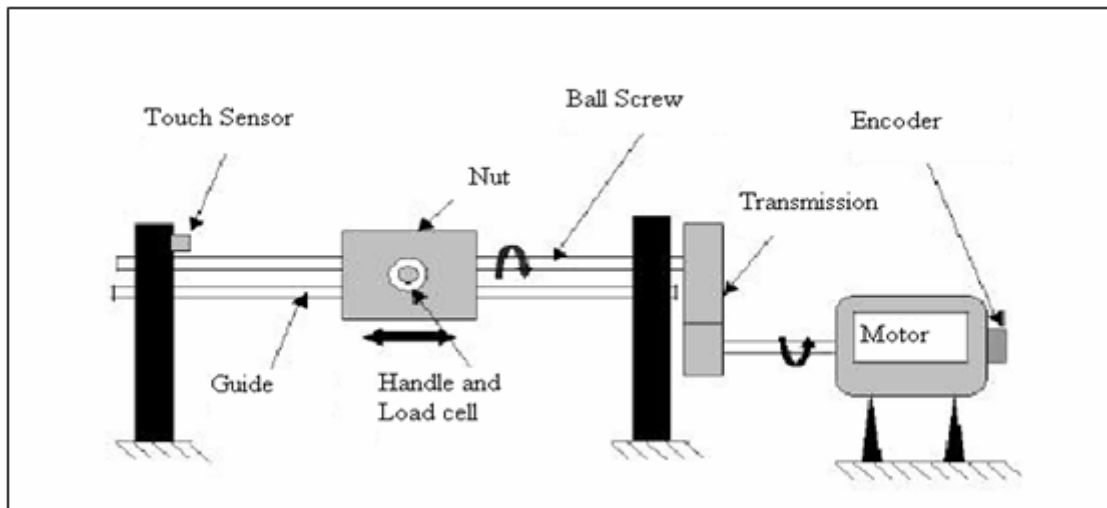


Figure 2.1 FRMC concept utilizing ball screw mechanism

A schematic of the ball screw joystick design is shown in Fig. 2.1. It consists of a ball screw (9.5 mm in diameter, 12.7 mm lead), a fingertip handle attached to the nut, and a motor. A touch sensor is used to initialize the system and an encoder to measure the position of the handle. Ball Screws replace the sliding friction of the conventional power screw with the rolling friction of the bearing balls thereby reducing the friction between the screw threads and the mating nut. The low friction inherent in a ball screw makes them virtually non-self locking and back drivable.

With a desired linear speed of 0.3048 m/sec of the nut, the rpm and the torque of the motor were calculated as 1440 rpm and 0.068 Nm, respectively. Therefore Maxon EC ϕ 40 mm, brushless 120 Watt motor with maximum continuous torque of 0.101 Nm was

considered. To determine the dynamic range of the actuator, the maximum and minimum forces are calculated as follows:

With the efficiency (E) of the lead screw of 78.91 %, lead (L) of the screw and maximum continuous torque (T) of the motor, the maximum force (F_{\max}) of 39.2351 N was calculated using the relation $F_{\max} = (2*3.1416*E)*T/L$. Using the frictional torque of the motor of 0.0096 Nm and breakaway torque of ball screw of 0.0212 Nm, the minimum force (F_{\min}) was 12.0158 N.

The low dynamic range (F_{\max}/F_{\min}) of 3.3 is a result of the large amount of friction inherent in the ball screw. For example, the ball screws reported breakaway torque is almost 2 times the friction torque of the motor. Since the low dynamic range would adversely affect the range of stimuli for the display, the ball screw joystick was taken out of consideration for our master controller.

2.3.2 FRMC Concept with gear system

The process of power transmission here is as follows: When the motor is switched on, the motor shaft rotates which is coupled to the pinion, which transmits power to the gear. The handle of the controller (joystick) is attached to the shaft of the pinion, which receives the transmitted power from the first gear. Fig 2.2 above describes the gear transmission system.

For this FRMC concept to be compact, we have to build the system by taking the gear ratio into account. For the FRMC to be successful, it should have an acceptable level of force reflection capability. Reducing the speed, which is accomplished by selecting a suitable gear ratio, will increase the torque output of the gear system.

Our 1-DOF FRMC prototype will be capable of providing a maximum force reflection of about 5 lb. To be compact, the gears are selected with a gear ratio of 284:1, which reduces the output speed by 284 times which in turn magnifies the output torque of the gear system. Maximum speed of the motor shaft is 15052 rpm and the speed at the output shaft will be reduced to a maximum speed of 53 rpm.

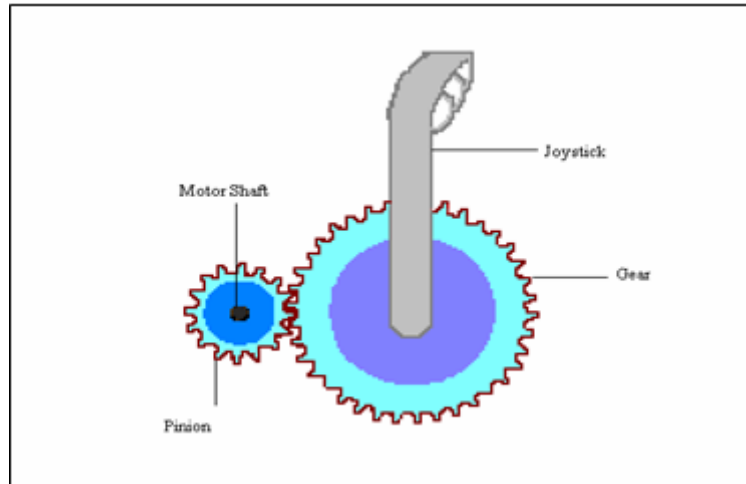


Figure 2.2 FRMC concepts with gear reduction

If we use spur gears, then the backlash will be more and the friction will be less relative to helical gears. If we use helical gears, we will have low backlash and the friction will be more when compared to the spur gears, there will not be any difficulty of back drivability in the case of spur gears.

2.3.3 FRMC Concept with belt system

The process of power transmission here is as follows: When the motor is switched on, the motor shaft rotates which is coupled to the pulley and it transmits power to the other pulley, which is connected to the joystick of the FRMC. The handle of the controller

experiences the torque and the maximum force reflection that can be felt will be about 5 lb.

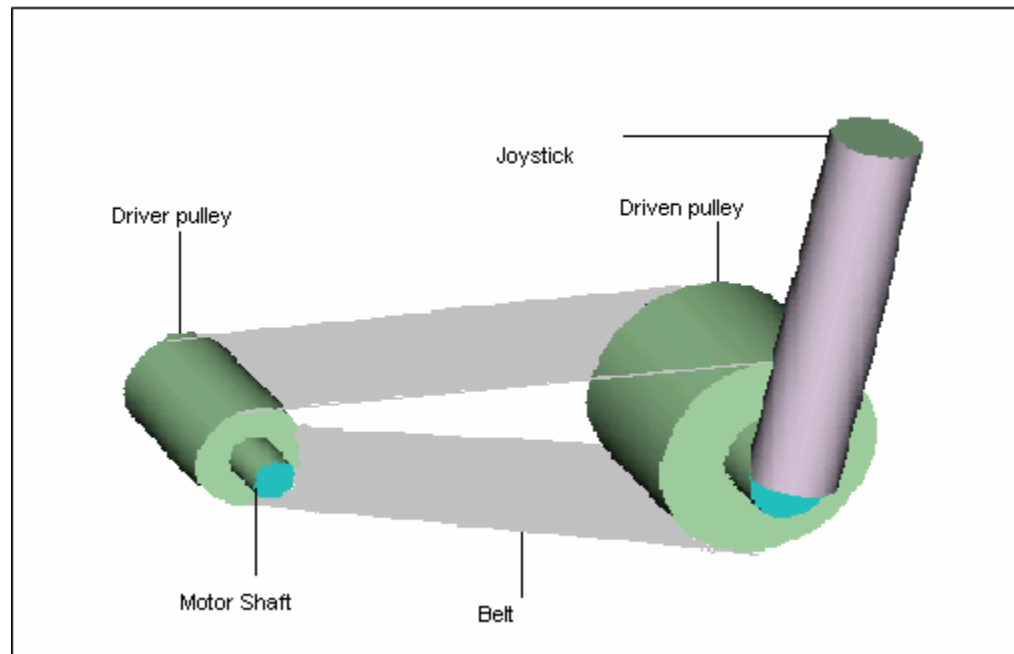


Figure 2.3 FRMC concepts with belt system

The main disadvantage with this concept of FRMC is that the slip factor plays a major role in reducing the torque and the force reflection will also be reduced do this slip factor. Distance between the pulley shafts is also a major limitation in this case. Belt drives are normally used to transmit power between relatively long distances. The Fig 2.3 shows the belt transmission system.

2.3.4 FRMC Concept utilizing direct drive system

In this concept, the joystick is directly attached to the actuator's shaft. By using this concept of direct drive method in FRMC makes the FRMC more compact and also more

efficient as it directly couples to the joystick. The maintenance of this FRMC is inexpensive and the total cost for building this FRMC is also less compared to the other methods. By directly coupling the motor shaft to the joystick we can have acceptably high torque and also smooth operation. The only disadvantage here is that a more complex amplifier design may be required. A FRMC model with direct drive mechanism is shown in fig 2.4.

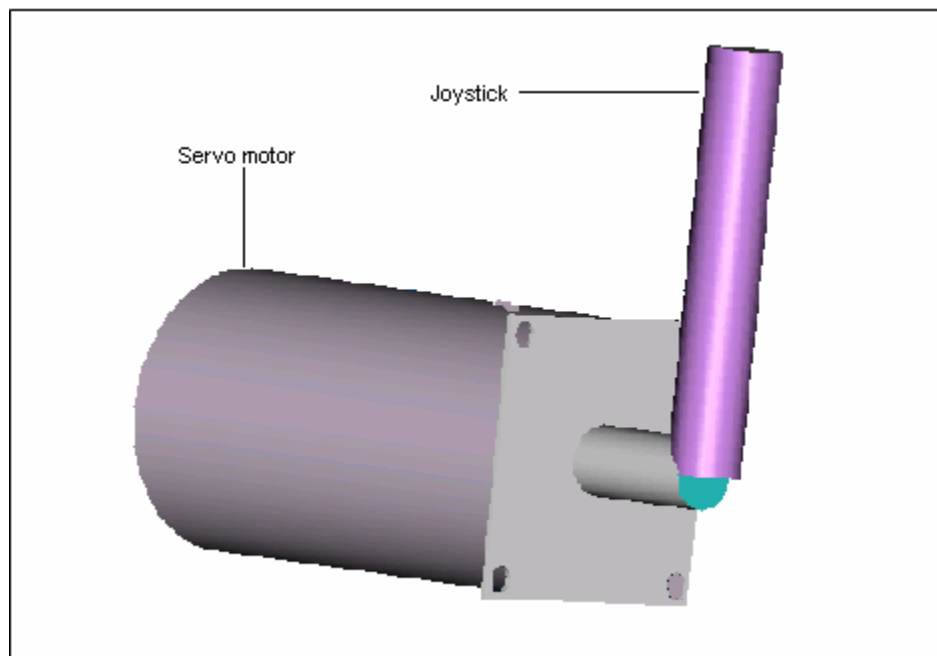


Figure 2.4 FRMC concepts utilizing direct drive system

Due to the direct linking of joystick to the motor shaft we don't have any intermediate losses. We also don't have any frictional losses or backlashes in this case. Unlike the gear transmission and belt transmission systems, direct drive system doesn't reduce speed and

also there is no torque amplification. So we have to select such a motor that is suitable for this configuration; i.e., which has less speed and high torque.

2.3.5 FRMC Concept with drive roller and belt system

To combine direct drive with a more ergonomic design, a linear manipulator was also considered. As shown below in fig 2.5, the design consists of a handle, a linear bearing, motor, and two timing belt pulleys (2.32 cm in diameter, 25 teeth, and 22.0 cm between pulleys). The motor chosen is the Maxon EC $\phi 45$ mm brushless 250 W motor with a maximum continuous torque of 0.306 Nm

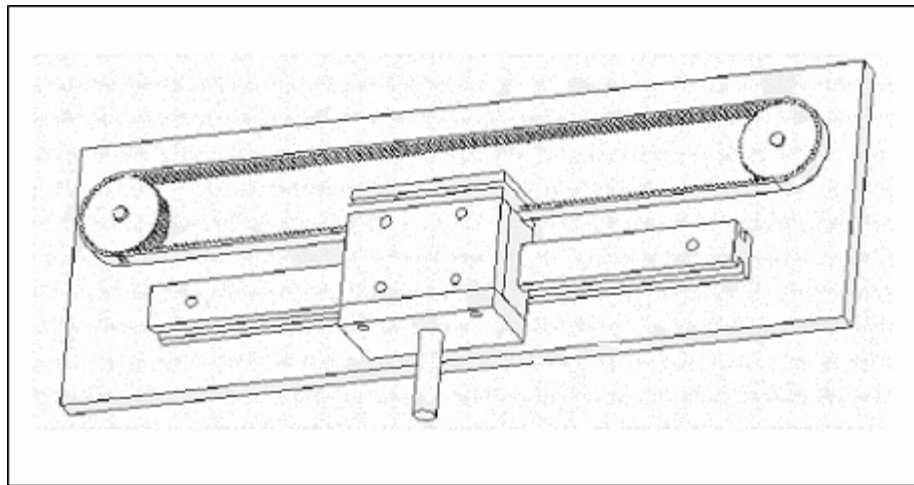


Figure 2.5 Roller drive and belt joystick

If a frictional (minimum) torque of 0.0206 Nm and a maximum continuous torque of 0.306 Nm of the motor are used, then a dynamic range of 14.8 is obtained. Since this design is suitable for ergonomic desktop manipulation and has a reasonable dynamic range, it may be a viable option for our haptic interface design.

CHAPTER 3

3.0 FORCE-REFLECTING TELEOPERATION SYSTEM COMPONENTS

This chapter briefly describes the major components used in a force-reflecting teleoperation system. The components listed in this chapter are actuators, various sensors including potentiometers, sonars, IRs and tactile sensors. The following chapter is then dedicated to microprocessors used in robot design.

3.1 Actuators

Actuator is a device that produces a change in its shaft position (displacement) when a signal is applied to it. The input power for a force-reflecting teleoperation system is provided by the actuator. Usually there are two types of motion, linear and rotary depending on the actuator [14]. Different types of actuators are briefly described below.

3.1.1 Hydraulic actuators

Hydraulic actuators convert fluid power into motion for many robot applications and are normally used when a large amount of payload capacity is required. Although hydraulic actuators come in many designs, piston types are most common. A typical piston-type hydraulic actuator is shown in fig 3.1.

It consists of a cylinder, piston, spring, hydraulic supply and return line. The piston slides up and down inside the cylinder and separates the cylinder into two chambers. The upper chamber contains the spring and the lower chamber contains hydraulic oil. The hydraulic supply and return line is connected to the lower chamber and allows hydraulic fluid to flow to and from the lower chamber of the actuator. The stem

transmits the motion of the piston to rest of the machinery. Initially, with no hydraulic fluid pressure, the spring force holds the valve in the closed position. As the fluid enters the lower chamber, pressure in it increases.

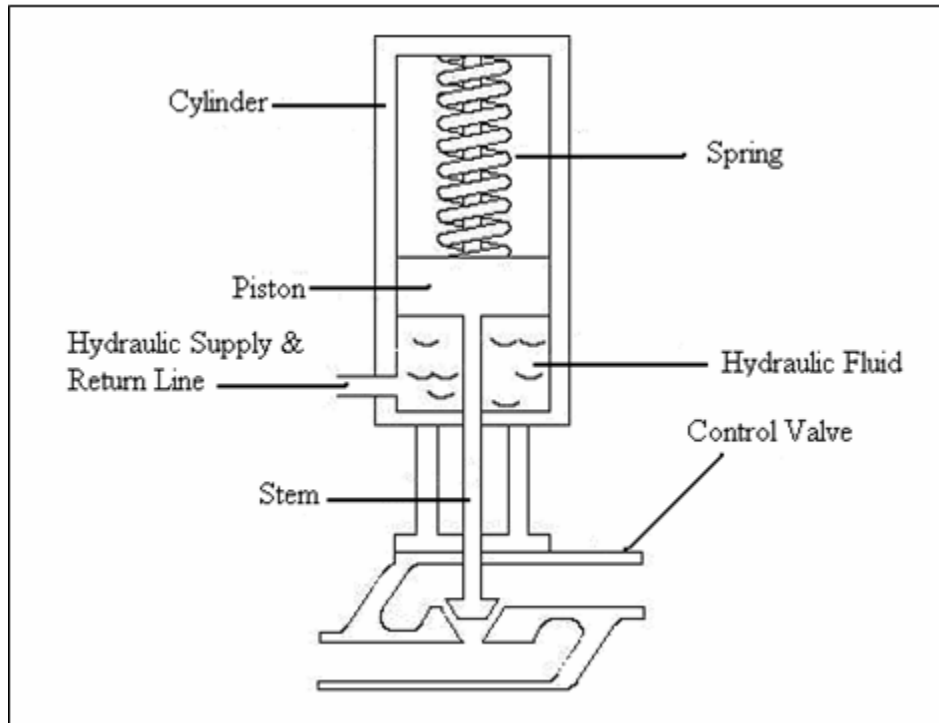


Figure 3.1 Hydraulic actuator

This pressure results in a force on the bottom of the piston opposite to the force caused by the spring. When the hydraulic force is greater than the spring force, the piston begins to move upward, the spring compresses. As the hydraulic pressure increases, the valve continues to open. Conversely, as hydraulic oil is drained from the cylinder, the hydraulic force becomes less than the spring force, the piston moves downward, and the valve closes. By regulating amount of oil supplied or drained from the actuator, the valve can be positioned between fully open and fully closed. The main advantages of these

hydraulic actuators are self lubricating, self cooling, not flammable and smooth in operation while the main disadvantages are that they need maintenance, not good at high speeds, oil leakage problems, large space requirements and not back drivable [2].

3.1.2 Pneumatic actuators

Pneumatic drive systems are found in approximately 30 percent of today's robots. These systems use compressed air as the medium of energy transmission. With pneumatic actuators, the pressure within the chambers is lower than that of hydraulic systems resulting in lower force capabilities. In fig 3.2 there is a cut-away view of the basic pneumatic actuator. It is quite similar to the hydraulic counterpart; however, there are no return hydraulic lines for fluid. In a typical actuator of this type, the fluid, namely the air, is simply exhausted through the outlet valve in the actuator.

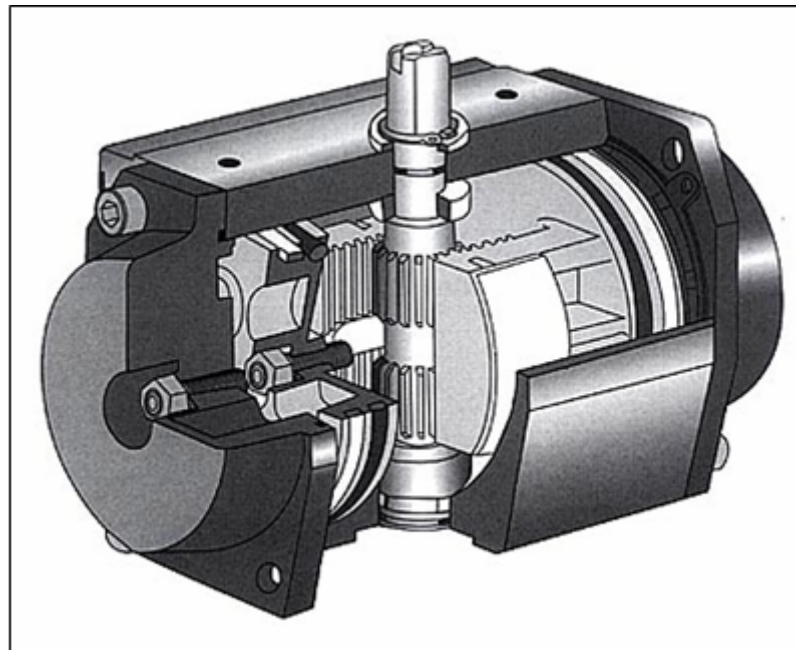


Figure 3.2 View of pneumatic actuator

Pneumatic systems have been used in robotic systems when lightweight, small size systems are needed with relatively high payload-to-weight ratio. The main advantages of these hydraulic actuators are that they are non-flammable, suitable for clean environments, and easy to operate and maintain while the main disadvantages are that they do not have self lubricating properties and not suitable for harsh environments. Also, compressibility of air is seen as a disadvantage in view of the compressibility of air since it prevents easy control of speed and position of a robotic device, which are considered to be essential ingredients for any successful robot operation.

3.1.3 Electrical actuators

Electrical actuators are found in approximately 70 percent of the robotic applications. The principle used behind an electric motor is simple: A magnetic field is applied to a ferrous core, which induces motion. These electrical motors are used where the conversion of electrical energy into mechanical energy is essential. The main advantages of electrical actuators are their high reliability, low friction, fast, accurate, clean, and quiet operation, low maintenance, and low cost. They can also be easily controlled by a microprocessor or computer, which makes them a very attractive alternative.

While the main disadvantages are their high speeds with relatively low torques, which necessitates the use of gear trains or other power transmission systems. Another problem may arise if electric motors are over heated which may cause electrical arcing. Hence, they may not be suitable for environments where flaming is a concern. There are different types of electrical motors depending upon the way they are controlled [2]. These motor types are briefly reviewed below.

- **DC motors:** when the voltage is applied to the terminals of the motor through armature then the motor begins to rotate. A DC motor is intended to work from a direct power supply. As the armature picks up speed, a voltage is induced in the winding that tries to oppose the current flowing in them. The speed at which balance is established (stabilization of motor speed) is a function of the applied voltage and the motor characteristics. The amount of turning force, torque that the motor can produce is a function of the current through the windings. DC motors respond quickly to changes in control signals due to the DC motor's high ratio of torque to inertia.
- **Stepper motors:** Stepper motor consists of two parts stator (fixed part) and rotor (rotating part). These motors are driven by a train of electrical pulses. The stator is wound as two separate coils, these coils are pulsed alternatively to produce a rotating magnetic field. Each pulse turns the rotor through fixed angle; hence angular position change is proportional to the number of pulses. Stepper motors have a fixed number of magnetic poles that determine the number of steps per revolution. Some of the important features of the stepper motors are large torques, excellent rotational accuracy, compact size and works well for range of speeds.
- **Servo motors:** Servo motors have feedback mechanism which enables it to be closed loop operation. These actuators have an optical encoder which measures position and velocity of the motor shaft. These motors continuously monitors the position and velocity information and compares it with the desired values and makes necessary

corrections to reach those desired values. These motors are bulkier and expensive. These motors have longer life and better heat dissipation properties.

- **RC servos:** These servos are very small in size and are inexpensive when compared to other motors. These Servos use potentiometer as feedback to determine their position, developed by hobby industry. The operating principle of these servos is simple, the servo compares its current position with the input Pulse Width Modulated (PWM) signal and will adjust itself to reach that position. These servos are extensively used in mobile robot applications because of its size and availability.

3.1.4 Actuator selection

The actuator selection plays a major role in deciding the size of the FRMC prototype. So the actuator selected should have all the important features like less speed, high torque, compact in size, inexpensive, readily available and easy to control. In the above mentioned all the motors RC servos will be suitable for our project because of its small size, inexpensive and easily available. RC servo has a built-in potentiometer to measure the shaft position of the servo. This servo compares the position of the motor with the input PWM signal and moves its position until that matches the input PWM signal. The pulse repeats every 14 to 20 ms (milliseconds). If the pulse width lasts for approximately 0.6 ms, the servo will rotate to a maximum position. If the pulse width is increased to approximately 2.4 ms, the servo will rotate to the opposite maximum position. A 1.5 ms pulse will set the servo in the middle (neutral) position [16, 27]. Fig 3.3 and fig 3.4 represent the servo timing diagram and sectional view of the Futaba servo motor.

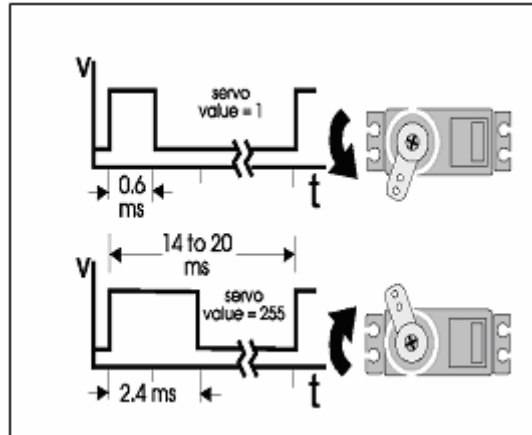


Figure 3.3 Servo timing diagram

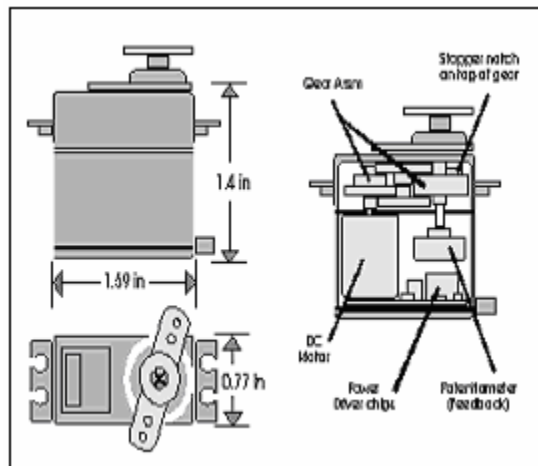


Figure 3.4 Futaba servo motor

These RC servos are used in model airplanes and cars, but when used with the servo controller SV203, it can be made to perform in various systems such as joysticks, mini-robotics, animatronics, and computer motion control. These RC servos are designed in such a way that, it turns only 90 degrees of angle in either direction, which comes to 180 degrees of motion. In order to make the servos to use in a situation that require

continuous rotation they have to be hacked. This modification is done by de-coupling the feedback potentiometer from the output gear and set at constant 90 degrees mark. Thus, when the signal less than 90 degrees is given to the motor, the motor turns at full speed in one direction and vice versa. Since the feedback is removed the servo will continue to rotate as long as the signal remains. The properties of Futaba RC servo selected for the design of FRMC are shown in table 3.1.

Table 3.1 Listing of specifications for futaba electric gear motor

Futaba electric motor model number: S3004 [13]		
Motor Model Specifications	Units	Values
Output Torque	Oz-in	56.9
Maximum Speed	Rpm	53
Size	in	1.59x0.78x1.42
Weight	Grams	38
Cost Estimate	\$	15
Operating Temperatures	Celsius	0 to 40

3.2 Sensors

Measurement of robot parameters is very essential for a successful industrial robot. Today, most of the robots use a component called sensor which is one of the important component of the robot for its survival. Sensor is defined as a device that converts some physical quantity into electrical quantity. A typical sensor consists of a transducer and an

electronic circuit. In robotics, depending on their characteristics these sensors are divided into two categories; internal sensors and external sensors. Internal sensors are those sensors which are used to measure the robot parameters relative to the reference frame of the robot such as wrist force, joint angle, force, position and mass etc., while the external sensors are those sensors which are used to measure the environment properties. Some of the internal and external sensors used in today's robotics field are discussed below [2]

3.2.1 Proximity sensors: Sensor that detects whether an obstacle or object is close to it without touching it. Optical proximity sensors have a light emitter and receiver through which they sense whether the object is within the sensitive range. If the object is within the sensitive region the light beam after emitting is received back otherwise the light beam is not collected by the receiver. These sensors will work in the range of 0 and 500 mm with beam angle of 10 degrees. Some of the proximity sensors also use electromagnetic induction. Inductive proximity sensors are used for a range of 1 to 20 mm. These sensors also use a technique of modifying the signal in case of air flow and fibre optic sensors. Airflow sensors use the flow of air to detect the object. If the object is present the air flow is blocked causing back pressure near the orifice from where air is injected. Magnetic field sensors detect the changes in the magnetic field. These proximity sensors are best suited for environments where the robot and object should not touch each other by preventing collisions and damages to the robot.

3.2.2 Range sensors: Sensor that detects an object at a distance called range or depth. These sensors use two physical principles: the time of flight of a pulse and

triangulation. This time of flight sensors emit a pulse and will measure time between pulse emitted and received. Here both transmitter and receiver lie on the same axis. To calculate the distance of the object we use

$$\text{Distance} = (\text{Time of flight} \times \text{Speed in medium}) / 2$$

Radar is the most common example of the time of flight pulse. Ultrasonic range sensors are used to find the distance of the obstacle. Time of flight measurements over short distances is feasible with ultrasonics as speed of sound is much slower than light. Triangulation sensor uses the geometry to find the distance of the object, which requires detection of the object from two different view points. When the light reflected from the object is detected, the emitter and detector angles are recorded, and the distance or range is calculated from those angles through geometry. Laser sensors use a laser diode as a light source, and a linear photo diode as a detector. From the angle of reflection the distance of the object is calculated. This sensor is more effective when the object is in the range of 5 to 60 mm.

3.2.3 Touch sensors: By using touch sensors we can find two characteristics of the object: its presence and its shape. Most tactile sensors uses switch to sense the object and its shape. A whisker sensor emulates a human hair and detects the object before it collides. The whisker will detect the object by touching the object but this will not damage the robot as it is flexible. When compared to vision sensor, touch sensor provide data which is directly related to the variable being measured; shape position, orientation.

3.2.4 Force/Torque sensors: The force/torque sensor is particularly suitable for applications requiring simultaneous measurement of several forces and moments, or measurements of forces that change direction and position over time. Common applications for this sensor include research and development in robotics, production processes, biomechanics, and dynamics. A waterproof version is available for use in tow tanks, ocean engineering, and other underwater applications.

3.2.5 Potentiometer: A Rotary potentiometer is an instrument which is used to measure the position of the shaft when attached to motors by measuring the difference of electric potential between the two points. Linear potentiometers are common contact transducers in the form of variable resistors with three leads. Two leads connect to the ends of the resistor, so the resistance between them is fixed. The third lead connects to a slider or wiper that travels along the resistor and the resistance between it and each of the other two connections varies.

3.2.6 Vision sensors: Those sensors which see and recognize the object by collecting the light reflected by the object into an image and processing that image. Some electronic devices or computers will process the data collected and will analyze the image of the object. By using these sensors the robot and the object will not touch each other, and are operated with some distance apart; which makes it to be used in hazardous environments where human cannot perform the task. In teleoperation, these vision sensors play very important role. Cameras can be used as the vision sensors in teleoperation and still research is being conducted on what type of vision sensors are to

be used for teleoperation to solve the vision related problems. Most of the image processing by these vision sensors requires complex algorithms that need to be run on the microchip at real time.

3.2.7 Tachometers: Tachometers are used for measuring the speed of the motor shaft. These are classified into two categories: contact type tachometers and non-contact type tachometers. Contact type tachometers are those which require contact with the surface whose speed has to be determined. Non-contact type tachometers are those which do not require contact with the surface whose speed has to be determined instead the speed is determined by focusing a beam of light on the surface. Laser tachometers are the most commonly used non-contact type tachometers.

3.2.8 Sensor selection

The sensor which is readily available, compact, inexpensive and effective should be chosen for our project. Keeping all these factors in mind we have selected to use Sharp GP2D12 analog infrared ranger [13, 17]. These sharp IR rangers use triangulation technique and small linear CCD array to determine the distance of the object from the robot. The sensors which use triangulation techniques are much more efficient than the sensors which use time of flight technique. This sensor has connector with 3 pins; power, ground and the output voltage. This sensor works effectively when the object is within a range of 10 cm to 80 cm. The operating supply voltage is between 4.5 to 5 volts. As shown from the graph below, the output voltage that is generated from the sensor represents a value of a trigonometric function, because of this, the output of these sensors

is non-linear with respect to the distance being measured as shown in Fig 3.5. Fig 3.6 shows the GP2D12 sensor [27].

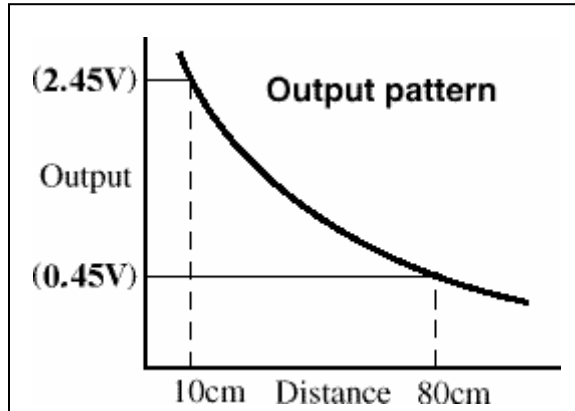


Figure 3.5 Non-linear graph

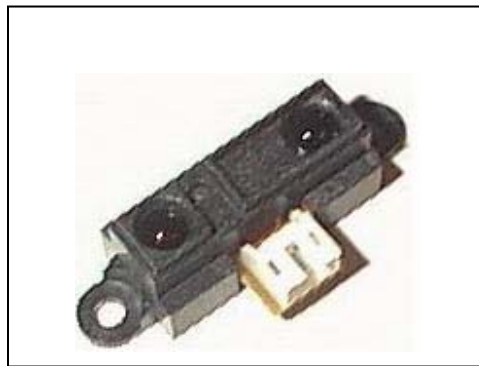


Figure 3.6 GP2D12 sensor

In case of no object, the light emitted is never reflected back. If the light emitted is received by the detector then a triangle is formed between the point of reflection, the emitter and the detector, which is shown in fig 3.7 below. From that triangle the angle of reflection is calculated and thereby the distance of the object.

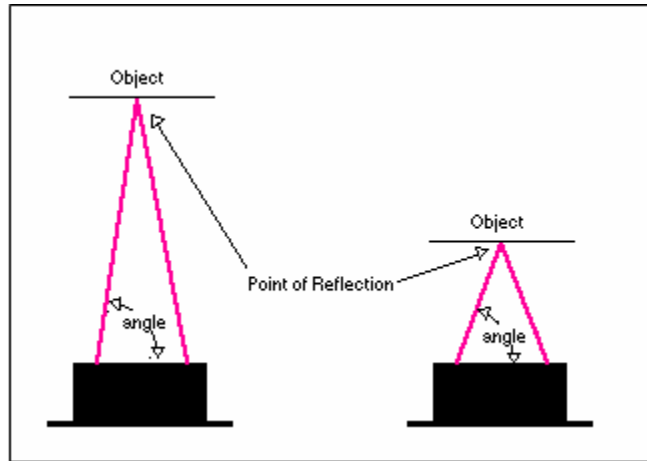


Figure 3.7 Distance through triangulation

The internal block diagram of the SHARP GP2D12 IR sensor is as shown in the fig 3.8 below:

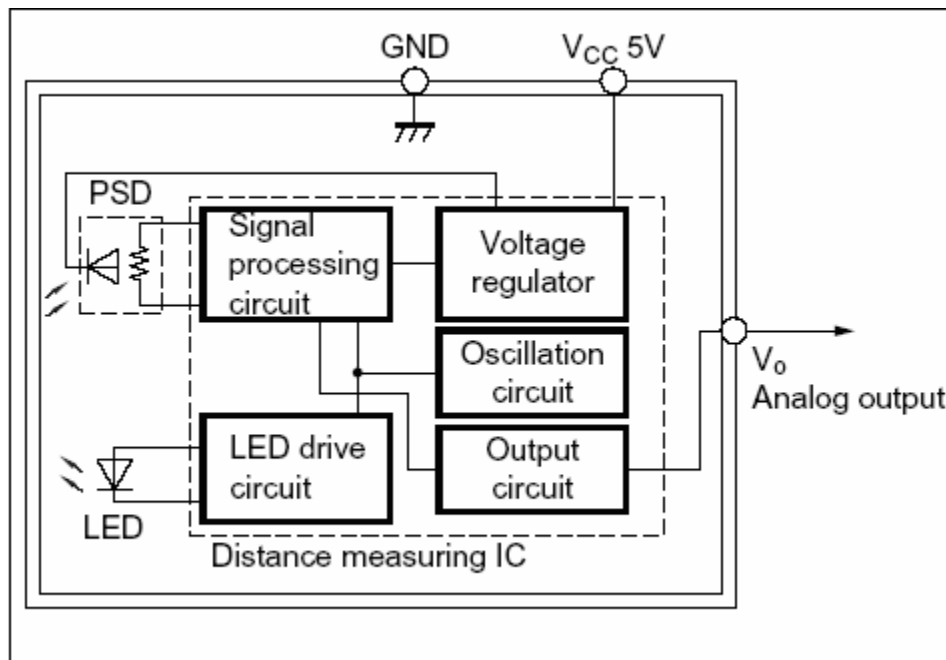


Figure 3.8 Internal block diagram

CHAPTER 4

4.0 MICROCONTROLLERS

Microcontrollers are considered to be the heart of a system wherever it is used. A microcontroller usually consists of microcontroller chip, voltage regulator, memory and input / output ports. There are various microcontroller producing companies in the market. When designing a small manual controller prototype (1-DOF FRMC), we need to consider the shape, size, cost, availability and its functions before deciding which microcontroller should be used. Every microcontroller has to be programmed to use it for some application.

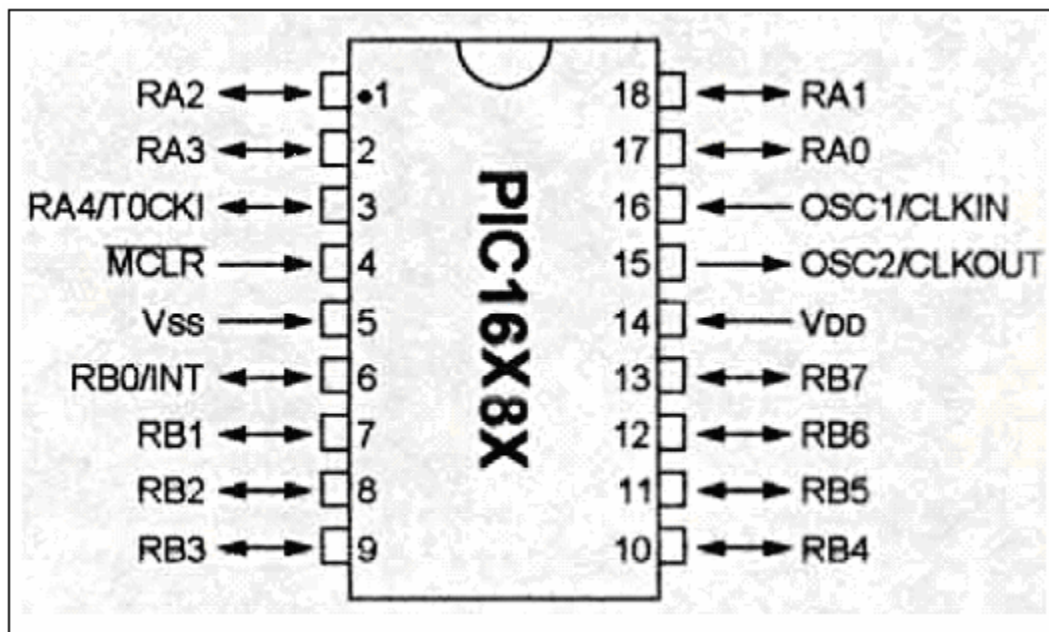


Figure 4.1 PIC microcontroller

In robotic applications, based on the robot's input, the internal programming in the microcontroller can command the motors in such a way that the robot works efficiently.

Of all the microcontrollers Programmable Integrated circuits (PIC), are the simplest microcontroller chips produced by Microchip Technology. Fig 4.1 shows the PIC microcontroller.

This PIC microcontroller should be programmed to be used and it can be programmed by using Basic, C or Assembly language. Some of the microcontrollers and boards that are being used in the robotic applications are described below.

4.1 Mini board:

The Mini Board was developed at Massachusetts Institute of Technology (MIT) for a robot course and design project. It is a small and inexpensive design for a controller board which uses 68HC11 micro-controller. The MINI BOARD 2.1 is a complete embedded computer board for robotic applications. A typical Mini board is shown in fig 4.2 below [25]. It can directly supply power to four DC motors and receive inputs from many sensors. Its small size makes it suited well for mobile applications as well as other embedded control. It can be programmed in 6811 assembler code or C for stand-alone operation, or it can serve as a serial-line based controller operated by a desktop computer. The Mini Board 2.1 Extended is the latest version of the Mini Board 2.0. Mini Board uses Motorola 68hc811e2 microprocessor with 2048 bytes of internal, EE PROM and 256 bytes of RAM which stores approximately 2000 instructions. This board can control four DC motors bidirectional at voltages 5.6 to 36 volts and up to 600 mA current. It has eight analog inputs, eight digital inputs or outputs, several timer and counter I/O pins. It uses RS-232 compatible RJ-11 port for communication/program download between host computers.

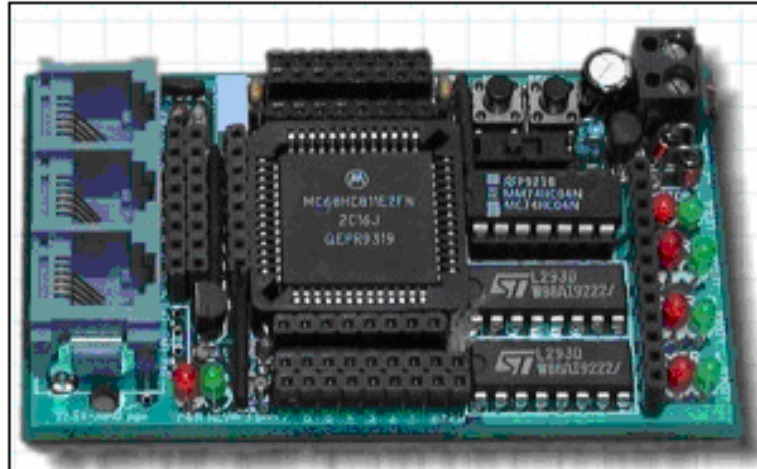


Figure 4.2 Mini board

4.2 Handy board:

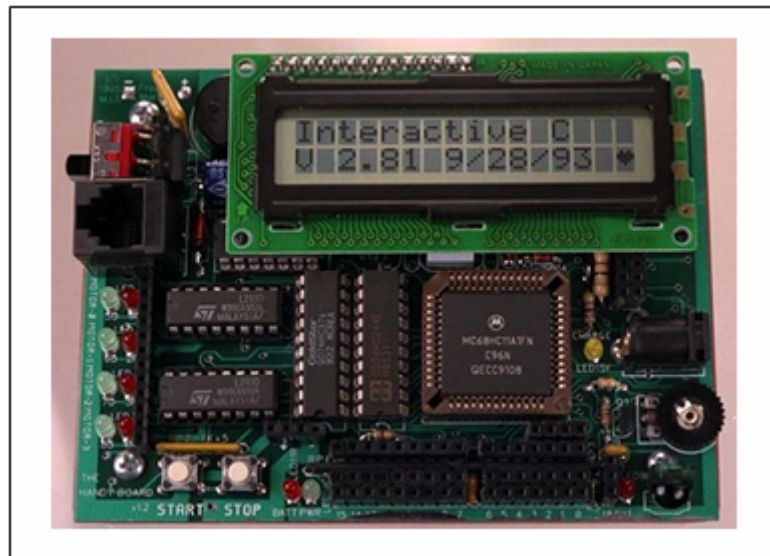


Figure 4.3 Handy board

A new microcontroller board ideal for experimental and educational robotic projects called handy board was developed by Fred Martin at MIT shown in fig 4.3 [24,28]. This handy board is the extended version of the mini board. Some of the important features of

this handy board are; Small board size (4.25 x 3.15 in), uses 52 pin Motorola 6811 microprocessor at 2 MHz, has 16 x 2 character LCD screen, has inputs for 7 analog and 9 digital sensors, has 38 KHz IR receiver and transmitter, connection to the desktop compute COM port (Rs-232), requires separate serial interface board. Since it is used in educational institutions, the software used should be readily available, so interactive C is commonly used to program these handy boards. Its small size and software choices make it used for commercial purposes.

4.3 Bot board:



Figure 4.4 Bot board

Bot board uses popular 68HC11 microcontroller with a minimum configuration designed for general purpose and also for the robotic applications. Some of the features of this Bot board are; it has a small size of 5 x 7.5 cm, is capable of controlling up to 4 R/C servos, powered with RS-232 for serial port communication, has multiple input / output ports, gives more flexibility for least cost. Fig 4.4 shows the generally used Bot board.

4.4 MTJPro11 microcontroller:

This microcontroller is one of the best 68HC11 microcontrollers developed by Mekatronics shown in fig 4.5 below. It is very small in size and is very inexpensive when compared with other microcontrollers with same features. Some of the important features of this microcontroller are; can control up to 5 servo motors, has 32 KB of memory, has a serial communication interface, it has 8 analog input channels and 3 digital input / input capture, it has output of 40 KHz clock signal to module IR &sonar, 8 digital output pins. The only main disadvantage when compared to other microcontrollers is that it does not have an LCD screen [28].

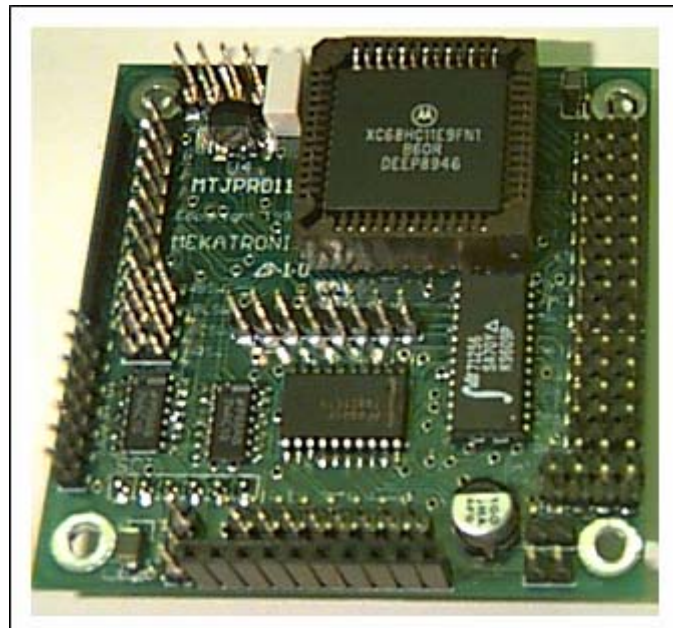


Figure 4.5 MTJPro11 microcontroller

4.5 Basic stamp:

Basic Stamp is a small microcontroller where a PIC microcontroller is embedded inside its design, and is developed by Parallax. This microcontroller with its operating system embedded within requires only a program to run it. The language used to run this microcontroller is the Basic language. Fig 4.6 below shows Basic stamp 2 [23].

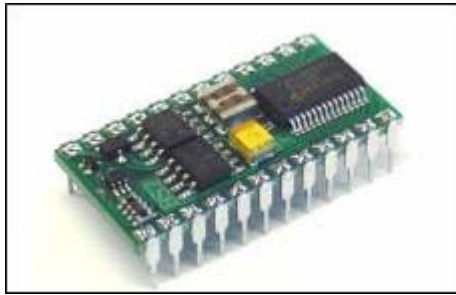


Figure 4.6 Basic stamp

Some of the important features of the Basic stamp include an ability to count cycles on a pin; 16 input / output lines, which are usually used to connect to LED's, potentiometers, push buttons, shift generators and speakers; generating PWM signals and sine waves; PBASIC interpreter; non-volatile EEPROM; resonator; and a 5-volt regulator.

When 5-to-15 volts are applied, the interpreter reads and executes the PBASIC instructions from the EEPROM. BASIC stamps are used in educational institutions, industrial applications, and hobbyist robotics projects. Usually BASIC stamps are able to execute up to 10,000 interpreted PBASIC instructions per second.

4.6 Other microcontrollers:

The JS2051A1 CPU BOARD is a very small CPU board that uses an ATMEL AT89C2051 CPU. The AT89C2051 is an INTEL 8051 CPU compatible microcontroller. It has an extra analog comparator and internal 2K-byte flash memory. Some of the features of this controller are listed as its small size of 36.8 x 20 mm, use of an LCD display and a 5 V DC supply. Fig 4.7 shows a typical AT89C2051 microcontroller below [24].



Figure 4.7 AT89C2051 microcontroller

The OOBoard is based on the OOPIC-R processor. The OOBoard with the OOPIC-R can be programmed by using any of the computer languages such as C, Java, Basic, or an object-orientated language. The OOBoard shown in fig 4.8 has the following features such as dual power supply connectors, dual operating modes selectable by a switch, three user-controllable switches, three user-controllable push buttons, a speaker, a serial LCD connector, TTL serial connector, RS232 serial port, three servo connectors and an h-bridge connector [24]. In order to power the board, two regulators are installed: Ultra-Low Drop-Out regulator for the microprocessor, and a 5A-5V accessory regulator for

motors, sensors, etc. The board also contains a constant current (C/10) battery charger for NiCad packs.

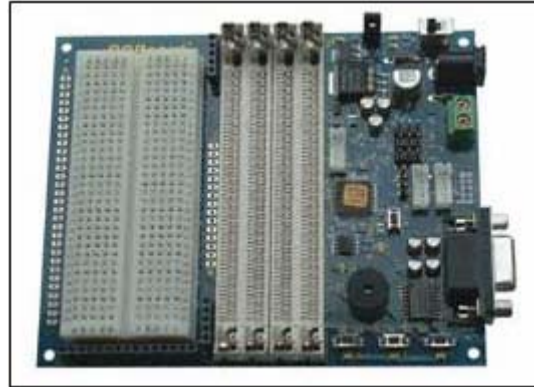


Figure 4.8 OOBOARD

4.7 Servo controllers:

Servo controller boards have been designed to control servo motors. In general, they can drive more servomotors than a single microcontroller can. Microcontrollers have an advantage of input options for sensors; whereas some servo controllers don't have this feature. Our project mainly deals with controlling servos so that we will be able to make a study on servo controllers as well. Some of the commonly used servo controllers in the robotic applications are discussed below.

4.7.1 Brainstem controller:

This BrainStem Module offers 2 channels of high-resolution motion control. These channels offer flexible PWM or PID control of motors with various types of feedback including encoders, quadrature encoders, analog input, and Back-EMF speed control. The

main features of this controller are, it has; 40 MHz RISC processor, one 10 bit A/D, status LED, 368 bytes of RAM, RS232 serial port, small size (2.5 x 2.5 in). The Brainstem controller is as shown in the fig 4.9 below [26].

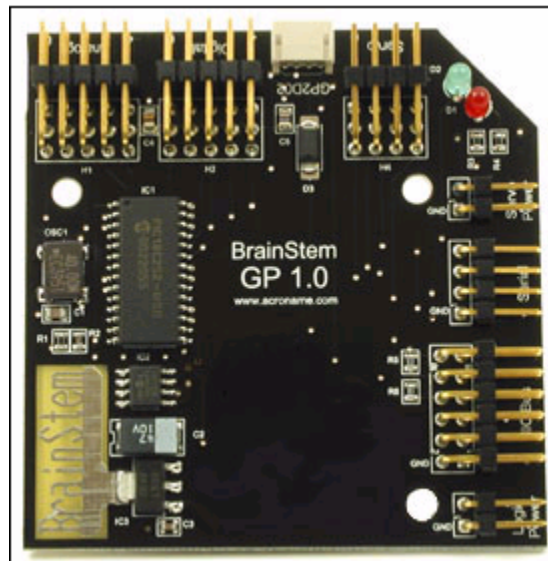


Figure 4.9 Brainstem controller

4.7.2 Pololu servo controller:

The Pololu servo controller shown in fig 4.10 can control up to 8 R/C servos [22]. This Pololu servo controller comes in two different configurations, where one can control a maximum of eight R/C servos and the other controls up to sixteen R/C servos. The interface with the computer to the servo controller is usually through an RS232 serial port or a TTL serial line at baud rates (maximum number of bits of information, including control bits, that are transmitted per second) of 1200 to 38400 baud. Some of the features of this servo controller include its small size of 1.45 x 1.7 in, eight servo ports, and a

power supply with a range from 5.6 to 25 V. The only disadvantage of this servo controller is that it does not have a port to connect any sensors.

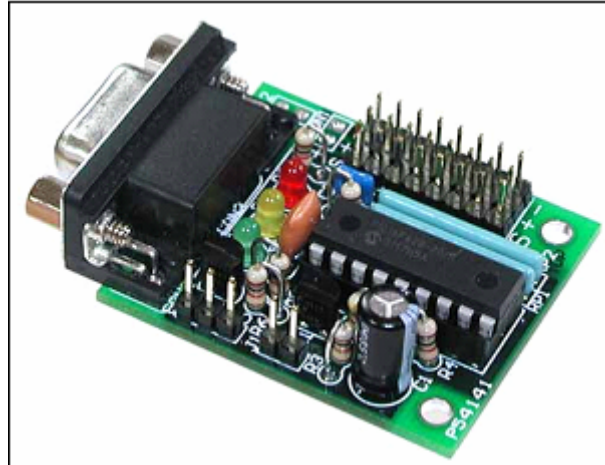


Figure 4.10 Pololu servo controller

4.7.3 USB servo controller:

USB Servo controller was developed by Lynxmotion to control R/C servos using the USB port attached to a computer. Fig 4.11 below shows how USB servo controller looks like [21]. This servo control can control up to eight servos.

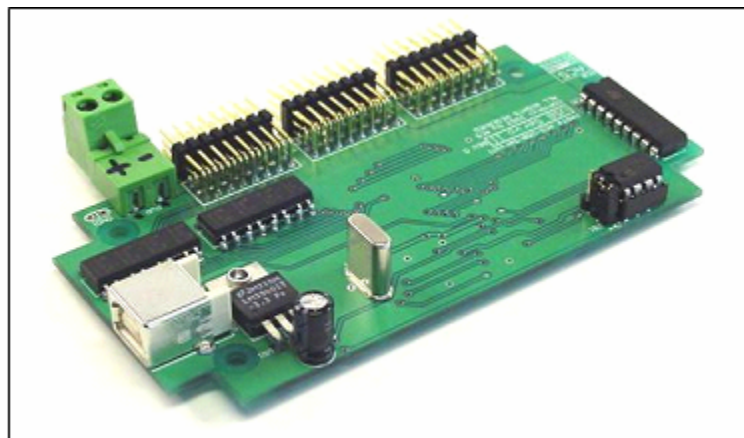


Figure 4.11 USB servo controller

It has eight optically-isolated inputs and also outputs. Power to the servo motors has to be supplied from the external power source. Lynxmotion has also developed two other types of servo controllers. Absence of sensor input is the main disadvantage in all the three servo controllers developed by the Lynxmotion.

4.7.4 Pontech SV203 servo controller:

SV203 Servo controller is used to control R/C servos when connected to a computer through RS232 serial port. This SV203 servo controller was developed by Pontech and can control up to eight servo motors. It has a PIC16C73 microchip embedded in it. This controller accepts serial data from an RS232 port and outputs PWM signal to control R/C servos. Unused servo pins can be reconfigured to digital output in order to drive on/off devices. Fig 4.12 shows the SV203 controller [28].



Figure 4.12 SV203 controller

This servo controller has eight R/C servo inputs and five A/D inputs. The board requires a power supply of 7-15 V and the A/D input power supply has a range of 0-to-5 V. This servo controller is very small in size compared to other controllers (1.4 x 1.7 in). The SV203B/C has the added feature of being able to run a standalone BASIC program on board through an 8K EEPROM (Electrically Erasable Programmable Read Only Memory). An optional IR (Infra Red) Receiver/Transmitter (IR 100) is also available to allow infrared serial communications. A 5-channel, 8-bit A/D input is available to read analog voltages between 0 and 5 Volts. Devices such as analog joysticks or potentiometers can be connected to this port and the position can be read by the PC and sent back to the board to control the servo position

CHAPTER 5

5.0 FRMC AND PLATFORM DESIGN

5.1 Force-reflecting manual controller system design

In the design of FRMC the motor selection, controller selection and the type of transmission system used are important criteria to be considered while designing the system. With this 1-DOF FRMC design, any one-degree-of-freedom system can be controlled; therefore, the research results are expected to be broader than a 1-DOF system might imply. In order to design the 1-DOF FRMC described above, six factors should be considered: (1) to accomplish a very low weight in the device, (2) to have a compact robotic system, i.e., overall size the robot should be small, (3) to transmit the power to the haptic device with very low power loss (torque), (4) to select a suitable motor and a controller, (5) to program the robot in an efficient way, and (6) to assemble and demonstrate the final performance of the entire system. The design parameters also include selection of a potentiometer, selection of the sensor, and developing the necessary software programs to accomplish remote control with the force reflection feature. The process of design will be carried out by using popular engineering software packages; namely, Pro/ENGINEER drawing tool, and Visual Basic programming tool. Performance of the system will also be demonstrated as part of this work.

Among the developed 1-DOF FRMC concepts, we have chosen the direct drive mechanism, which is judged to be more efficient than others. We believe that we can satisfy the goal of a compact design with a 1-DOF device whose size and motion are similar to that of a PC mouse. Since fingers will manipulate the device, it should be

capable of producing a maximum force reflection of about 2 to 4 lb. In this concept, the joystick is directly attached to the actuator's shaft. The direct drive mechanism is as shown in fig 5.1.

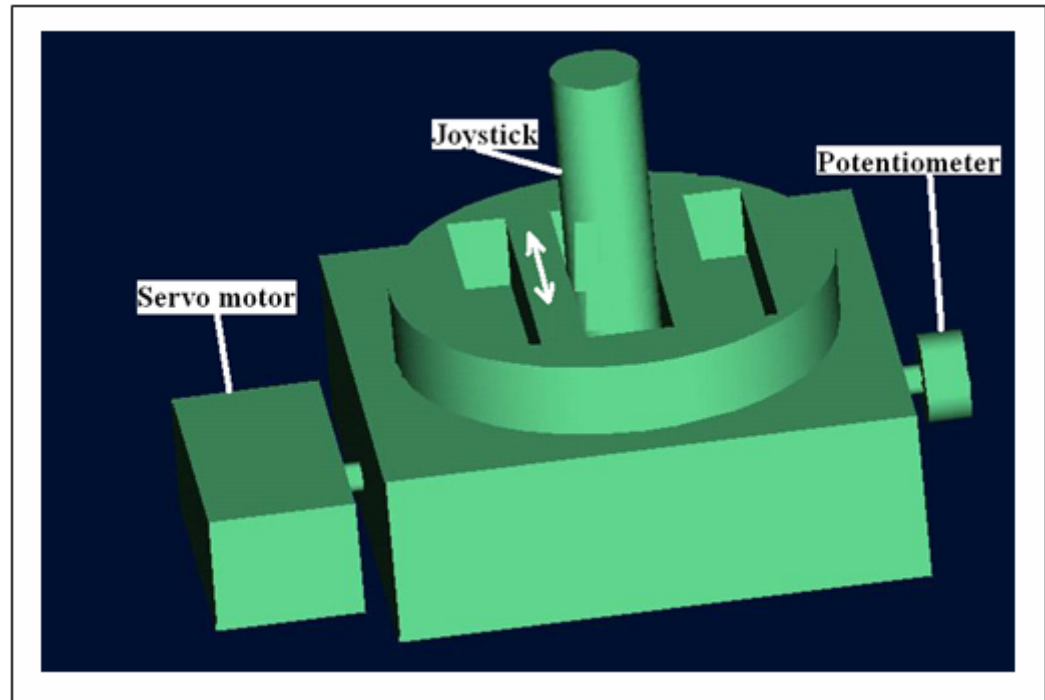


Figure 5.1 Direct drive mechanism

By using this concept of direct drive method in FRMC makes the design more compact and also more efficient as it directly couples to the joystick. The maintenance of this FRMC is inexpensive and the total cost for building it is also less costly compared to the other methods. By directly coupling the motor shaft to the joystick, we can have acceptably high torque and also smooth operation. Due to the direct linking of joystick to the motor shaft we don't have any intermediate losses. We also don't have any frictional losses or backlashes in this case. However, unlike the gear transmission and belt

transmission systems, a direct drive system doesn't reduce speed and also there is no torque amplification.

Finally the manual controller was made from the following components:

- Microcontroller board
- Servo motor
- Power supply
- Infrared sensor
- Potentiometer

We have selected SV203 servo motor controller which uses PIC16C73 microchip developed by Pontech as the microcontroller for our case, and the advantages of using this micro controller in our case is described in chapter 4. We have selected Futaba S3004 as the required servo motor and GP2D12 IR as the required infrared sensor for our design. The selection of these two parameters is described in chapter 3. The connection of the servo motor with the SV203 board is as shown in fig 5.2 below.

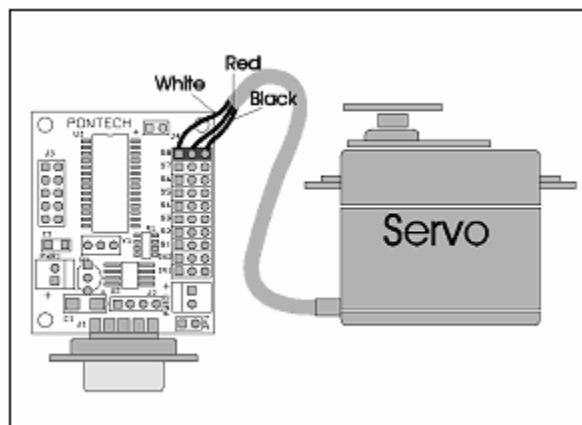


Figure 5.2 Connection of servo with SV203

5.2 Power supply and pin-out

A 6-Volt DC source powers the SV203 micro controller board, either from 4 alkaline batteries or 5 NiCad (Nickel Cadmium) cells. An AC adapter can also be used: 6VDC, at 300mA. If using NiCad, a 4-cell pack might be easier to find than a 5-cell pack. The board will operate fine with 4 cells, but may not last as long as 5 cells can. Given below in Figure 5.3 is the pin out diagram, which explains in detail the entire circuit board terminals and the respective pin outs.

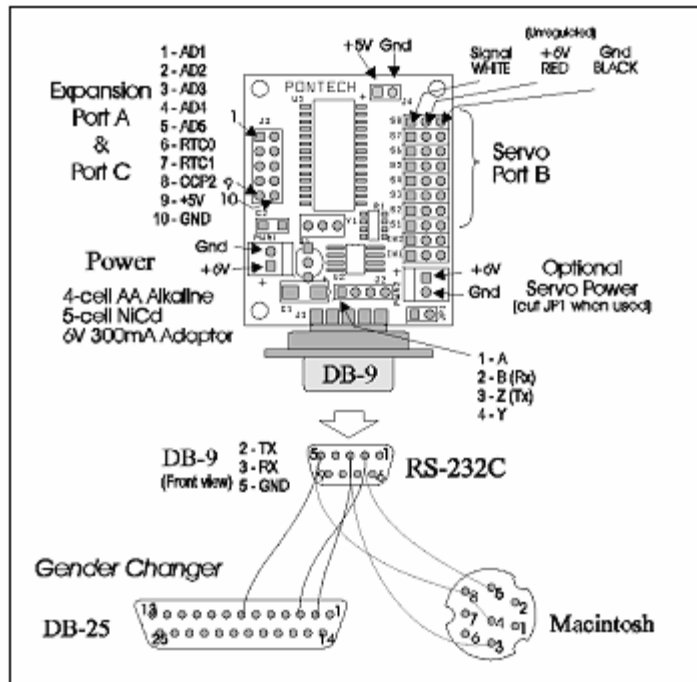


Figure 5.3 Pin out diagram

The potentiometer is used to measure the shaft position of the servomotor and sends it to the compute through SV203 board. The potentiometer is attached to the servomotor as shown in the fig 5.4 below.

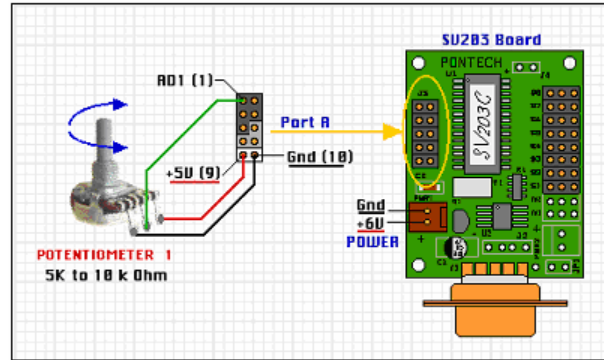


Figure 5.4 Connection of potentiometer with SV203

5.3 Housing

The main objective is to make the structure as simple as possible and at the same time capable of holding as many as components as possible. The housing structure was made of plexi glass to mount the robot components, along with fastening facilities such as screws and nut housing for mounting the components. Plexi glass was chosen to reduce the overall weight of the robot. The other components are plastic angles for holding and mounting the servo motor and servo controller. Fig 5.5 shows the 1-DOF FRMC developed.

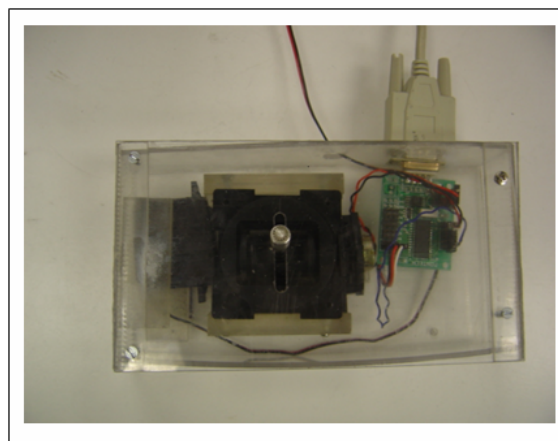


Figure 5.5 1-DOF FRMC developed

5.4 Slider bar platform design

In order to test the 1-DOF FRMC, a platform has to be developed. So initially we have developed a simple 1-DOF sliding bar platform which is built by using a RC servo motor, infrared range sensor, sliding bar, SV203 controller. The system operates at 7 V of power supply. When the joystick is moved in forward/backward direction, the potentiometer reads the position of the joystick and will send the signal to the SV203 controller, according to this potentiometer feedback SV203 controller produces the signal to activate the slider servo motor to move it in forward/backward direction accordingly. The sensor on one end of the slider bar monitors the path of the moving part and will send the signal to the SV203 controller. Once the moving part (plate) reaches the boundary of the slider bar then the signal from the infrared sensor is sent to the servo controller which in turn moves the servo motor coupled to the joystick to the other side and will produce force reflection if the user still moves the joystick in the same direction and the servo motor which is driving the slider bar will rotate in reverse direction until it reaches the boundary on the other side. Fig 5.6 shows the slider bar platform developed to test 1-DOF FRMC.

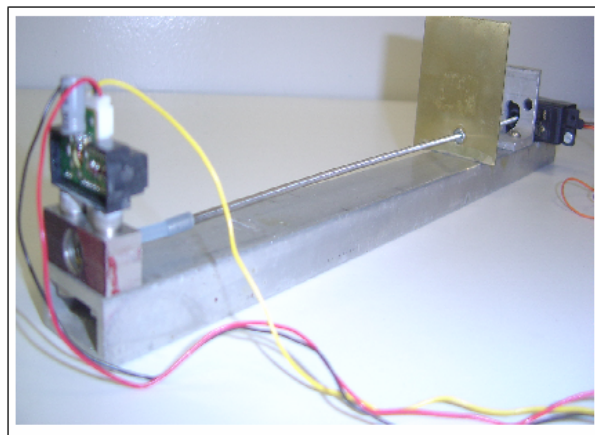


Figure 5.6 Slider bar platform

5.5 Palm pilot robot kit (PPRK) platform design

A platform named palm pilot robot kit was developed by freshman Grigoriy Reshko and Dr. Matt Mason at Carnegie Mellon University (CMU). We have decided to use this PPRK to test our 1-DOF FRMC. Fig 5.7 shows the PPRK designed at CMU. The PPRK is a small robot that uses a palm as brain of the kit. The main purpose of designing PPRK was to make mobile robots available to everyone.



Figure 5.7 PPRK designed at CMU

The PPRK empowers a palm pilot to move about and sense the nearby environment. The base uses three omni-wheels that allow driving in any direction with independent control of rotation, meaning it moves holonomically in the plane. The base also has three optical range sensors to see the nearby environment up to about a meter away [29]. This PPRK uses a Brainstem controller which can be programmed by using Basic, C, C++ or Java.

The program to control the PPRK is loaded on to the palm pilot and the brainstem controller receives the signal from the palm and responds accordingly. This brainstem controller operates at 6 V of supply. This kit can also be controlled by the desktop PC instead of palm. The commands are sent to the controller through the RS232 serial port [20]. So we have decided to buy only the required parts and assembled the PPRK platform. Table 5.1 below shows all the required parts to build a PPRK

Table 5.1 parts required to build the PPRK

Part name	Quantity
4cm diameter omni-directional wheels	3
Modified RC servo motors	3
Sharp GP2D12 Infrared Rangefinders	3
Brainstem Controller	1
Palm Pilot III	1
6 V battery	1
Clear Cast Acrylic Disk	2
Male DB9 connector	1

5.5.1 Palm pilot robot kit (PPRK) platform design modification

The mobile robot PPRK uses the brainstem controller while the 1-DOF FRMC developed uses the SV203 controller. Integrating PPRK with FRMC may cause a problem while programming. So we have decided to modify the PPRK, by replacing the

brainstem controller with SV203 controller as it fulfills all the requirements of a brainstem controller. So the modified PPRK will be having SV203 controller instead of brainstem controller, three RC servos, three IR sensors, 6 V of power supply. The SV203 controller receives the signal from the palm and will send commands to the servos accordingly. The power supply to the sensors is taken from the unused servo pins on the SV203 board as the board cannot supply power through the AD power supply to all the three sensors. Fig 5.8 shows the modified PPRK kit.

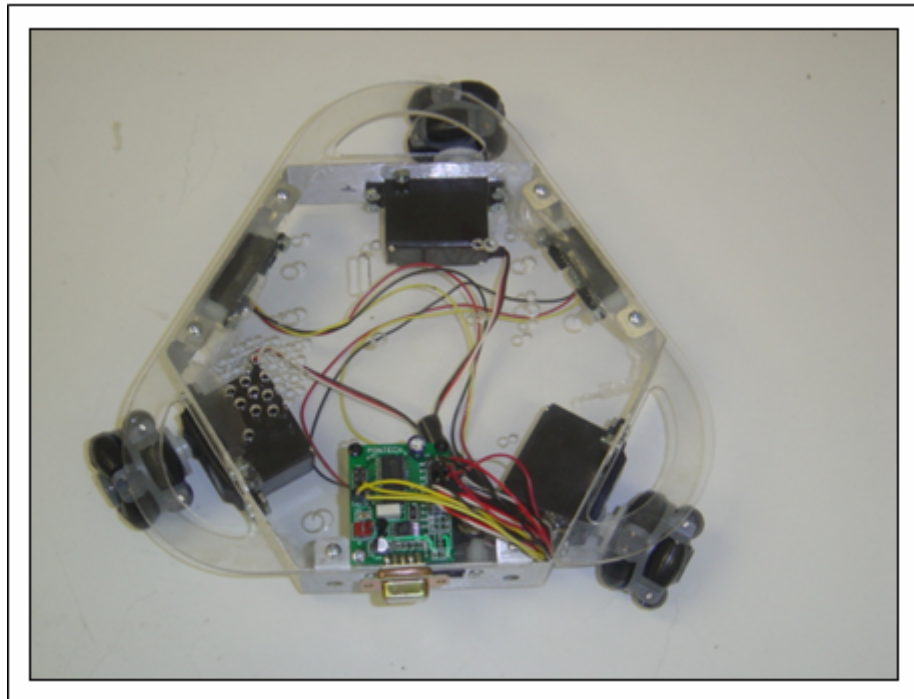


Figure 5.8 PPRK modified mobile platform

CHAPTER 6

6.0 ROBOT CONTROL SOFTWARE DEVELOPMENT AND SYSTEM

INTEGRATION

In this chapter, the software development and system integration of the FRMC controller prototype with the remote system is presented. We initially discuss interfaces developed to control the FRMC step by step, and then demonstrate how the 1-DOF FRMC prototype is used to control remote systems, that include the PPRK platform and the slider bar.


6.1 Robot control software

Computer simulations and training are considered to be critical aspects in this project since the tasks they are used are in general critical in nature and should be carried out without any error. Creating a realistic computer simulation makes the operator to learn how the actual processes really look and feel. For the advanced operations like hazardous clean-up sites in nuclear environments, the real-time computer graphical simulations are compulsory. So, the development of teleoperator systems to be used in such environments will be considered as great and worthwhile achievements. For this purpose, software development is necessary and a graphical user interface (GUI) should be developed to make the user control the system with ease.

The main task of the program which is to be developed is to control Pontech SV203 controller. This SV203 controller can be programmed by Qbasic, C, Visual Basic or JAVA. Any of these programming languages can be used as long the SV203 controller receives the ASCII codes to perform the task. The computer interface of this SV203

microcontroller is through RS232 serial port. Table 6.1 below describes the RS232 port [19].

Table 6.1 RS232 serial port male 9 pin connector

Male RS232 DB9	
Pin Number	Direction of signal:
1	Carrier Detect (CD) (from DCE) Incoming signal from a modem
2	Received Data (RD) Incoming Data from a DCE
3	Transmitted Data (TD) Outgoing Data to a DCE
4	Data Terminal Ready (DTR) Outgoing handshaking signal
5	Signal Ground Common reference voltage
6	Data Set Ready (DSR) Incoming handshaking signal
7	Request To Send (RTS) Outgoing flow control signal
8	Clear To Send (CTS) Incoming flow control signal
9	Ring Indicator (RI) (from DCE) Incoming signal from modem

We have decided to program in Visual Basic to communicate between the SV203 controller and the computer. The main reasons to choose Visual Basic is due to the fact that it is easy to program, is a powerful programming tool, as well as developing GUI interface and control through serial port are easy when compared to other programming platforms. The code is written in windows environment to be operated easily by the user. Many versions of the interfaces have been developed before the final version, which is presented in this work.

6.2 1-DOF FRMC Prototype simulation with mouse

We have developed a program code in Visual Basic to show exactly how a 1-DOF FRMC operates. We have taken a PC mouse as our 1-DOF joystick (master robot), designed a car (representing a platform) on the PC monitor which is assumed to represent the slave robot and used an airplane icon (on the PC monitor screen) as an obstacle on the platform's motion path.

When the user moves the mouse (master robot) in the X-direction, the platform (slave robot) on the PC monitor moves accordingly. The program allows the user to set different positions for the obstacle. When the platform approaches the obstacle (plane icon) and if the user still tries to move the platform in the same direction, various color schemes (green, yellow, and red) warn the user of impending collision. This mimics force reflection on the actual prototype. The form views of the program depicting snapshots of the screen are shown in fig 6.1 and fig 6.2.



Figure 6.1 Form view when robot on the left is away from the obstacle represented by the plane icon

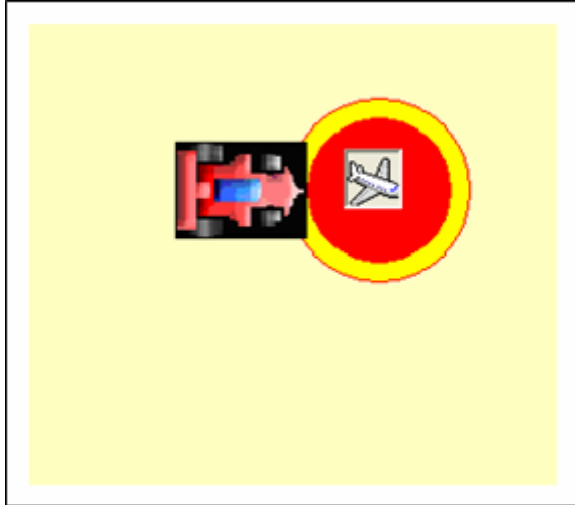


Figure 6.2 Form view when robot is close to the obstacle

6.3 1-DOF FRMC Prototype simulation with SV203 controller

We have replaced the PC mouse with a 1-DOF joystick which is operated by using a servo motor and the remaining parts are kept same. We have used a potentiometer to identify the shaft position of the servo motor, which is connected to the A/D port of the SV203. The GUI developed to control this SV203 is as shown in fig 6.3 below.

The GUI, which opens and closes the COM port, also controls servomotors and the A/D channels that are connected to the SV203 controller. When the 1-DOF joystick is made to move in either direction, the platform (slave robot) on the PC monitor moves accordingly. When the platform approaches the obstacle (the plane icon), various color schemes (green, yellow and red) warn the user of an impending collision. And if the user still tries to move the platform in the same direction, the operator feels the force reflection in the opposite direction so that the platform is moved away from the obstacle.

After the platform is out of the collision area, force reflection is turned off and the current joystick position is set to the zero or home position.

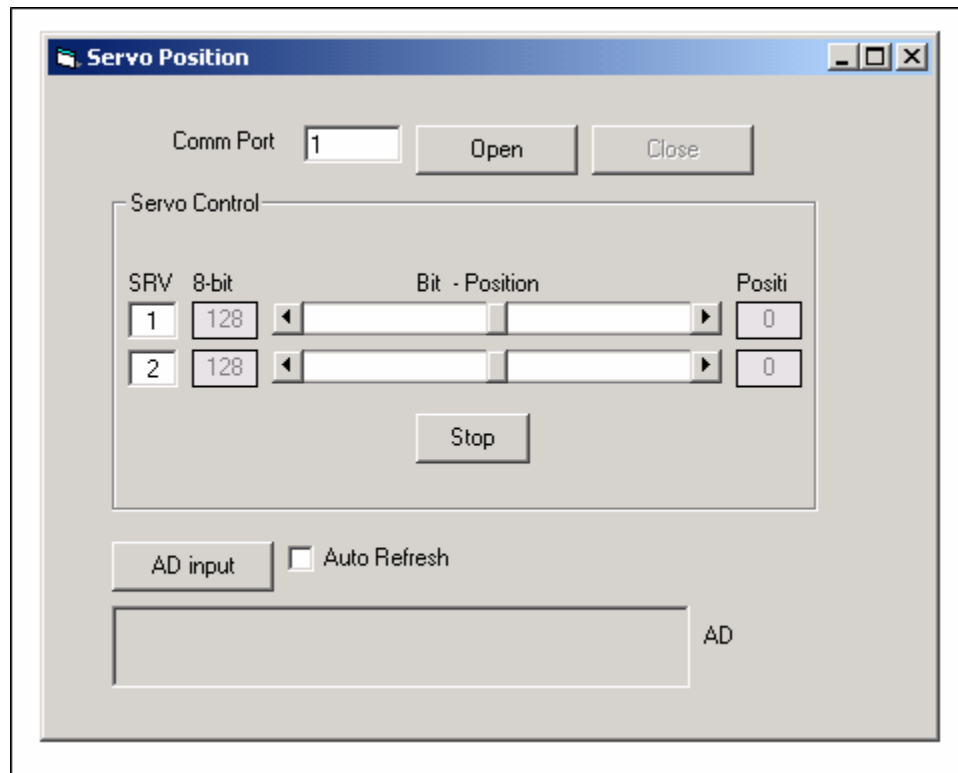


Figure 6.3 GUI developed to control SV203

Later we have updated the program so that the user can choose an icon to represent the slave robot either as a walking robot, car, or dinosaur. Similarly, obstacles can be represented on the screen in the form of a chair, football, doll, penguin, or aeroplane. The program also gives user the option to choose either position control (constant speed) or velocity control (variable speed) mode. The program also gives the user an option to specify a time delay, the amount of time delay, or no time delay. This program is also updated to let the user choose the direction of the slave robot motion (X

or Y direction), which makes movements in the plane possible. The updated GUI is as shown in fig 6.4 below.

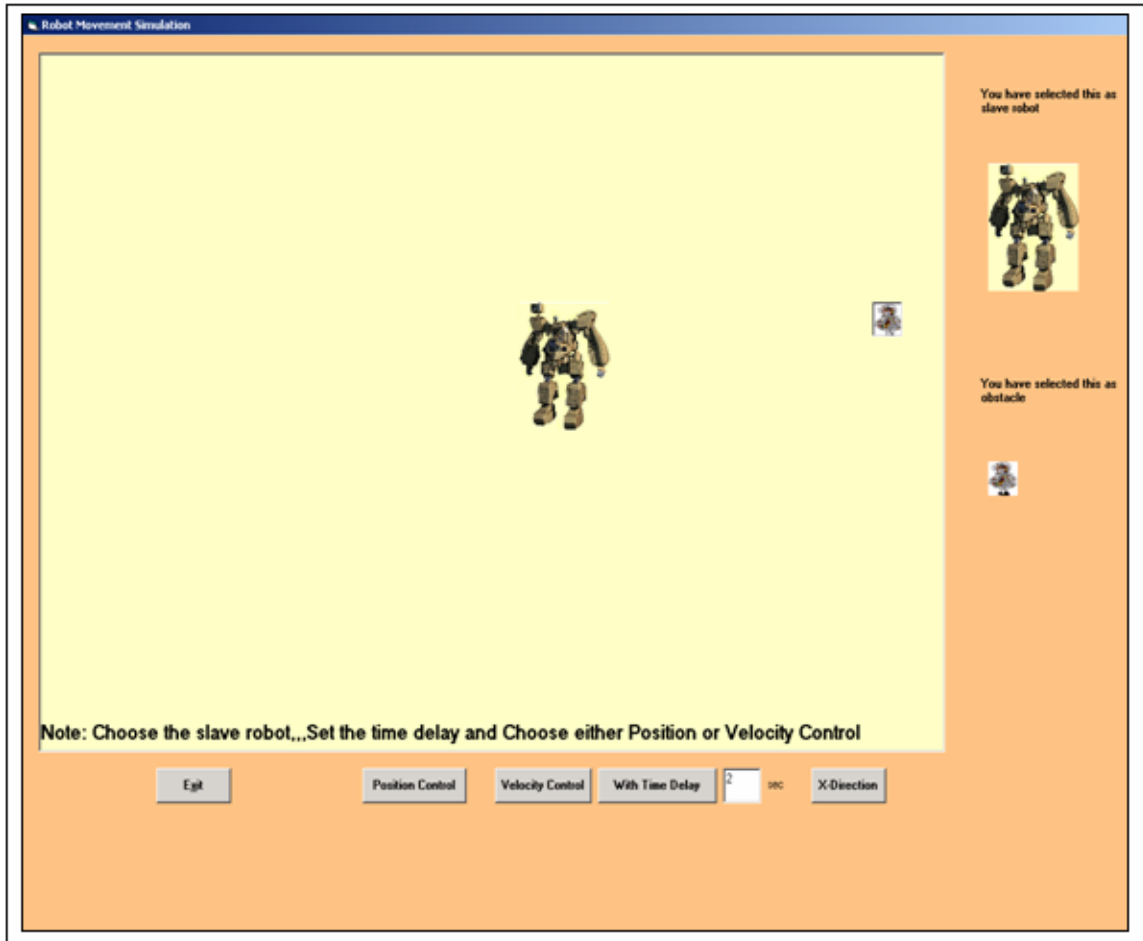


Figure 6.4 GUI of Visual Basic program

6.4 Time delay in space applications

The time delay for a signal to reach the moon from earth is on average 3 sec. Usually the loop delays (round-trip delays) are much greater; approaching 6 sec in the case of earth-orbiting space shuttle because of multiple up-down links (earth to satellite or the reverse) [30]. To incorporate time delay effects in remote control, we need the average end-to-

end round-trip time between the slave and master robots. As the moon is very close to the earth compared to the other planets, the round-trip time delay in communications between the earth and moon is in the order of 3 to 6 sec. The space shuttles around the earth also come in the same zone of time delays.

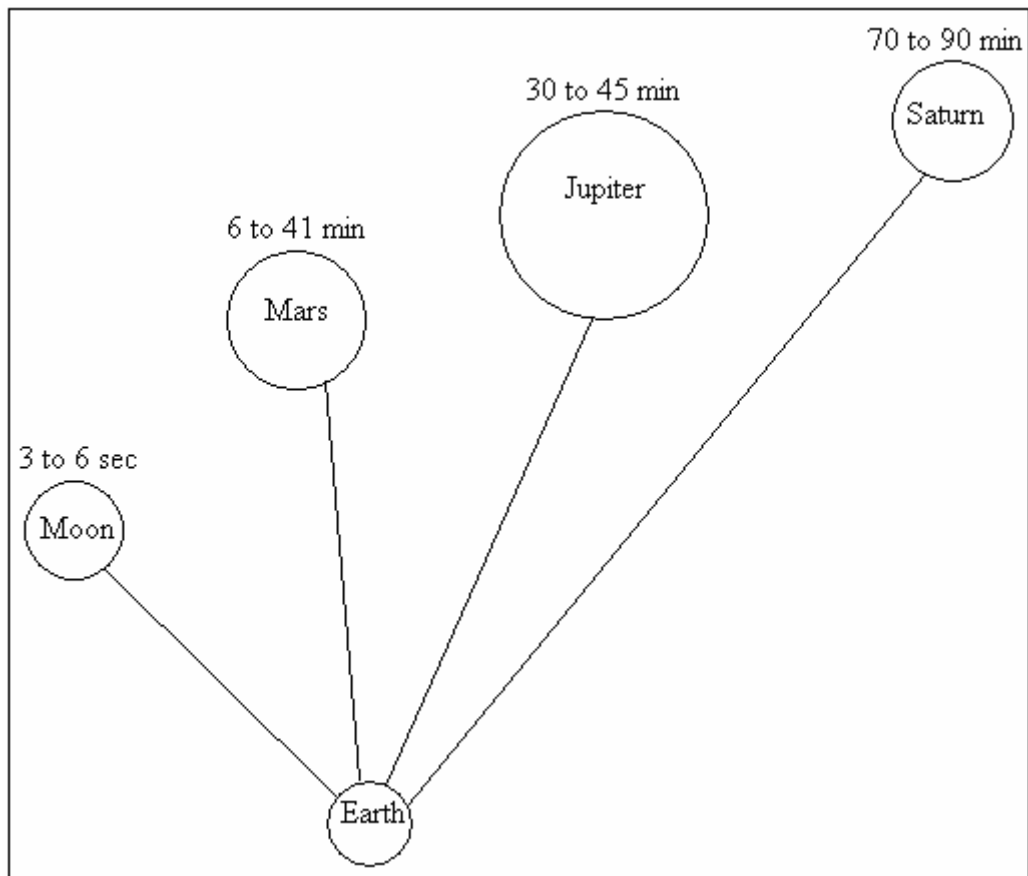


Figure 6.5 Round-trip communication time delays between the earth and planets

There are other planets around the earth, where there is a need for humans to communicate during space exploration missions. Depending upon the distance from the earth these time delays vary. Some of the timed delays in communications are mentioned below. Also, fig 6.5 shows the time delays from earth to different destinations in the

universe. The end-to-end round-trip time delay for Mars-Earth communication network varies from 6 minutes to 41 minutes [32]. Also, the communication network delay for Jupiter-Earth varies approximately from 30 minutes to 45 minutes, and for Saturn-Earth communication the delay is approximately varies between 70 minutes to 90 minutes depending on their relative positions.

6.5 Time delay experiments conducted with 1-DOF FRMC

We have conducted several experiments with the 1-DOF FRMC on different users and with different time delays. The experimental setup is described below.

The remote site is represented on the computer screen as shown in fig 6.4.

- The user can position the obstacles among the indicated options
- Slave robot is chosen at the runtime by the user
- Time delay can be specified by the user
- Start and end positions are located at the diagonally opposing corners of the work area

The user is asked to operate the 1-DOF FRMC and move the slave robot from the predefined start position to the end position which are fixed (same) for all the users. With the same amount of time delay and same obstacle positions, each user has been tested and the amount of time taken for each user to reach the end position has been tabulated for different time delays varying from 0 to 8 sec. Each experiment was carried out on different days with different users to make sure that they do not get habituated to the path of the testing. With the gathered information, the effect of time delay on user's

performance is studied. Table 6.2 shows the time required by each user to complete the task under different time delays.

Table 6.2 shows the experimental data

Time delay in sec	Time taken by User 1 in sec	Time taken by User 2 in sec	Time taken by User 3 in sec	Time taken by User 4 in sec	Time taken by User 5 in sec
0	36.1	21.7	28.0	42.8	29.7
1	58.3	43.5	41.7	57.2	63.3
2	73.5	82.1	64.2	82.1	70.2
3	138.4	150.3	119.4	157.2	138.1
4	280.2	268.7	240.9	276.4	247.1
6	427.5	435.2	389.2	440.4	413.5
8	579.4	640.5	601.5	660.1	611.8

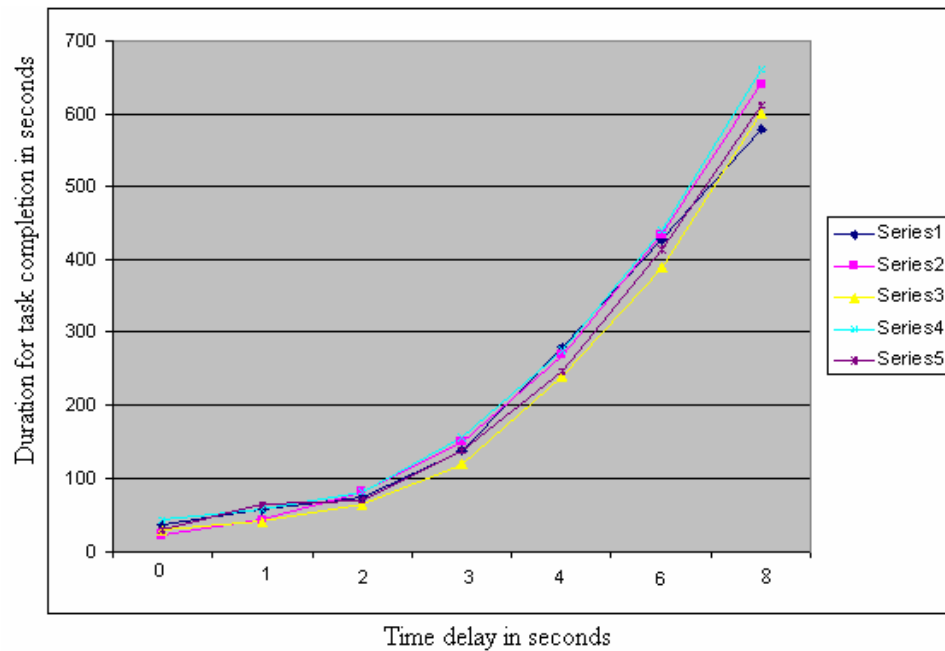


Figure 6.6 Time delay trend for different users identified as series 1 through 5

Fig 6.6 above plots the trend for each user's performance with respect to different time delays. As we can see from table 6.1 and fig 6.6 for the same start and end positions, each user has completed the required task in different timings, which depicts that the time delay will affect each user's performance differently. Hence, training users may help reduce this variation. Also, we have observed that the average time for task completion increases parabolically with the amount of time delay. For instance, the task has been completed on average in 74 seconds when 2 sec time delay was implemented in the experiments. The task is completed in 421 seconds when time delay becomes 6 sec as can be seen in table 6.1 and fig. 6.6. Hence, in this example, a 4 sec increase in time delay from 2 to 6 sec introduces a 6-fold increase in task completion time, which is quite high.

Finally, we would comment that the methods to enhance system performance caused by time delays are in general meant for delays in the order of 10 sec at most and any larger time delays as indicated in fig. 6.5 are only for comparison purposes and no immediate remedy to such large time delays have been suggested. As the time delay increases, it would become harder for the operator to control the system and maintain the stability of the system. To reduce the negative effects of time delay, research has been carried out and one method known as the wave variable technique has been suggested to promise improvements. By using this technique, detrimental effects of time delay, for instance, in communicating between the earth and moon, is expected to be reduced.

6.6 System integration

Integrating the hardware and software for remote control systems is essential as it also requires integration and operation of its components over the network. Hence, in

addition to the traditional hardware and software system design, we address broader system integration. In our system, integration means controlling the platform (PPRK platform modified for our project to represent the remote system) with the 1-DOF FRMC prototype through the software developed. It is important to develop the software in such a way to control the slave robot (modified PPRK) by the master robot (1-DOF FRMC) in real time as well as to incorporate a capability to control other remote systems.

In the present project, the structure of the sensing, planning, control system and the computer architecture has been designed for any task rather than for a specific task. The software interface developed is user-friendly, and this property of the system enables the user to complete the required task more efficiently. The two robots (master and slave) which are connected to two different computers (client and server) are operated by Winsock control (Visual Basic) through internet. In the system developed, the 1-DOF manual controller will get the feedback from the remote slave robot in two different methods; one by force reflection and the other by visual information received from the remote site through video camera attached to the slave robot.

In this visual feedback system, the video camera replaces the sensors of the force feedback system. The system with force feedback is capable of working in the remote site by seeing the simulations or the animated remote site on the screen of the computer to which it is attached while the system with the visual feedback is capable of seeing the real time video of the remote site through video camera attached to the slave robot. Integration of the modified PPRK platform, 1-DOF FRMC and the software makes the system more complex. However, the software developed is capable of controlling any

robot built with the SV203 micro controller through internet by the 1-DOF FRMC developed.

6.7 Winsock control

The Winsock control provides easy access to Transfer Control Protocol (TCP) and User Datagram Protocol (UDP) network services. It can be used by Microsoft Access, Visual Basic, Visual C++, or Visual FoxPro developers. By setting properties and invoking methods of the control, we can easily connect to a remote machine and exchange data in both directions.

6.7.1 TCP

The Transfer Control Protocol allows the user to create and maintain a connection to a remote computer. Using the connection, both computers can stream data between each other.

In client server applications, while creating a client application, we must know the server computer's name or IP address (RemoteHost property), as well as the port (RemotePort property) on which it will be “listening” to. Then invoke the Connect method.

In order to create a server application, we should set a port (LocalPort property) on which to listen, and invoke the Listen method. When the client computer requests a connection, the ConnectionRequest event will occur. To complete the connection, the system invokes the Accept method within the ConnectionRequest event. Once a connection has been made, either computer can send and receive data. To send data, we need to invoke the

SendData method. Whenever data is received, the DataArrival event occurs. To retrieve data, we have to invoke the GetData method within the DataArrival event.

6.7.2 UDP

The User Datagram Protocol (UDP) is a connectionless protocol. Unlike TCP operations, computers do not establish a connection. Also, a UDP application can be either a client or a server. To transmit data in client server applications, first set the client computer's LocalPort property. The server computer then needs only to set the RemoteHost to the Internet address of the client computer, and the RemotePort property to the same port as the client computer's LocalPort property, and invoke the SendData method to begin sending messages. The client computer then uses the GetData method within the DataArrival event to retrieve the sent messages.

6.7.3 Winsock properties

Winsock enables the user to create clients and servers using the same control. This dual functionality enables the user to specify through property setting the type of application we will be building. Some of the important properties of the control are as follows:

- BytesReceived property

This property returns the number of bytes currently in the receive buffer. The value returned is a long integer.

- LocalHostName property

The LocalHostName property returns the name of the local host system. The value returned is a string.

- LocalIP property

The LocalIP property returns the local host system IP address in the form of a string, such as 11.0.0.127.

- LocalPort property

This property returns or sets the local port number. This can be both "read from" and "written to" and is available at both design time and runtime. The value returned is a long integer.

- RemoteHost property

The RemoteHost property returns or sets the remote host. This can be both "read from" and "written to" and is available both in design time and runtime. The value returned is a string and can be specified as an IP address.

- State property

This returns the state of the control as expressed by an enumerated list. This is a read-only property.

6.7.4 Winsock methods

- Accept method

It accepts the request for connection from the client system. For this method to be used, the control must be in the listening state.

- Close method

The Close method terminates a TCP connection from either the client or server applications.

- GetData method

GetData is the method that retrieves the current block of data from the buffer and then stores it in a variable of the variant type.

- PeekData method

The PeekData method operates in a fashion similar to the GetData method. However, it does not remove data from the input queue.

- Listen method

This is invoked on the server application to have the server application wait for a TCP request for connection from a client system.

- SendData method

This method dispatches data to the remote computer. It is used for both the client and server systems.

6.8 Operating the system software

The process of operating the system software is discussed below. There are two computers used in this process and are named as client and server. The 1-DOF manual controller (master robot) is connected to the server side and the modified PPRK or the 1-DOF slider bar (slave robot) is connected to the client side.

First the client side computer should be connected to the server side computer by entering the IP address of the server side computer. Once they are connected, both the

computers are able to send and receive the data. The 1-DOF manual controller (master robot) has to be loaded on the server side. Similarly the modified PPRK or 1-DOF FRMC (slave robot) has to be loaded on the client side. As both the systems, master robot and the slave robot use SV203 micro controller, they have to be connected to the computers through RS232 port. Fig 6.7 shows the basic process of the system developed.

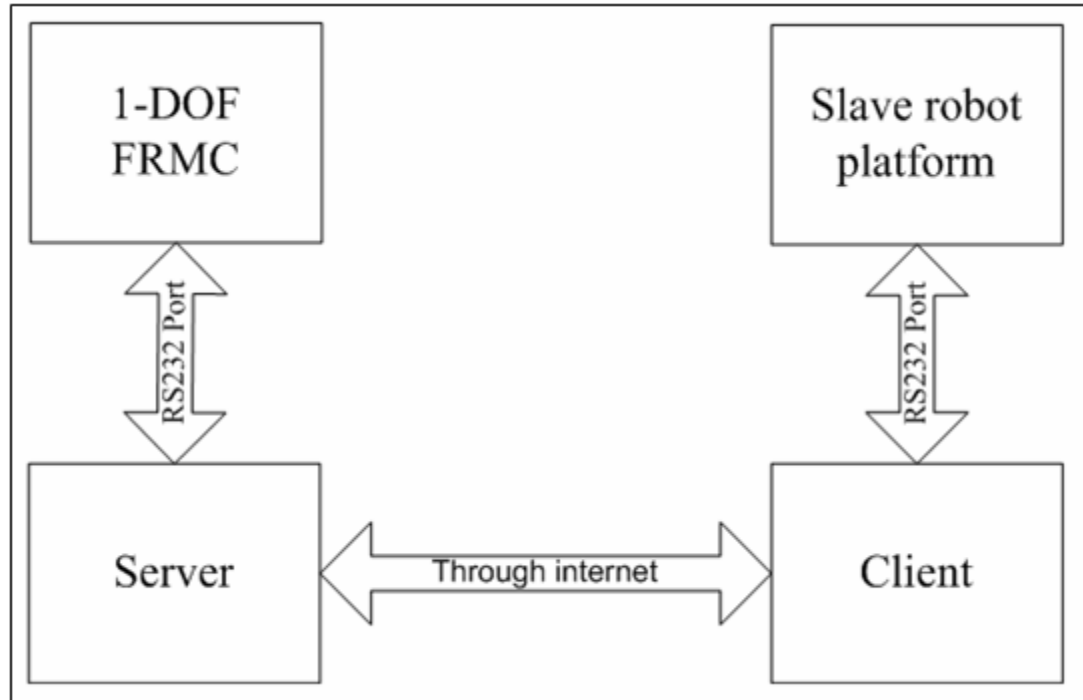


Figure 6.7 Process of the system developed

The position of the joystick is tracked by the potentiometer attached to it and that AD reading is sent to the server side computer through SV203 controller and RS232 port. The AD reading from the server side computer is sent to the client side computer through internet. The client side computer sends the AD reading to the SV203 controller through RS232 port, which compares the AD reading with the condition and will move the servo attached to the slave robot accordingly. At the same time the reading of the IR sensor on

the remote site is sent to the client side computer through SV203 controller and will in turn send that AD reading to the server side computer through internet. The AD reading of the IR sensor is checked with some conditions and will make the SV203 controller to decide whether to call force reflection on the joystick or not.

As this process is continuing the AD reading of the potentiometer is sent to the server side computer and on the screen of the server side computer the user can see the simulation of the remote site environment and feels as if he is working in the real environment. While operating joystick the platform on the computer screen moves accordingly, and as it approaches any obstacle on its path and if the user still tries to move the joystick in the same direction, various color schemes (red, yellow and green) warn the user of impending collision which also sends the force reflection command on the actual 1-DOF FRMC.

6.8.1 Video streaming

Here in this process, the user operating slave robot will see live video of the environment where the slave robot is operated through the video camera attached to the slave robot. The system settings of this operation are described as follows. First the client side computer should be connected to the server side computer by entering the IP address of the server side computer. Once they are connected, both the computers are able to send and receive the data. The 1-DOF manual controller (master robot) has to be loaded on the server side. Similarly the modified PPRK or 1-DOF FRMC (slave robot) has to be loaded on the client side. As both the systems, master robot and the slave robot use SV203 micro controller, they have to be connected to the computers through RS232 port. A web

camera should also be connected to the computer USB port and is kept on the slave robot as shown in the fig 6.8 below.

The position of the joystick is tracked by the potentiometer attached to it and that AD reading is sent to the server side computer through SV203 controller and RS232 port. The AD reading from the server side computer is sent to the client side computer through internet. The client side computer sends the AD reading to the SV203 controller through RS232 port and will move the servo attached to the slave robot accordingly.

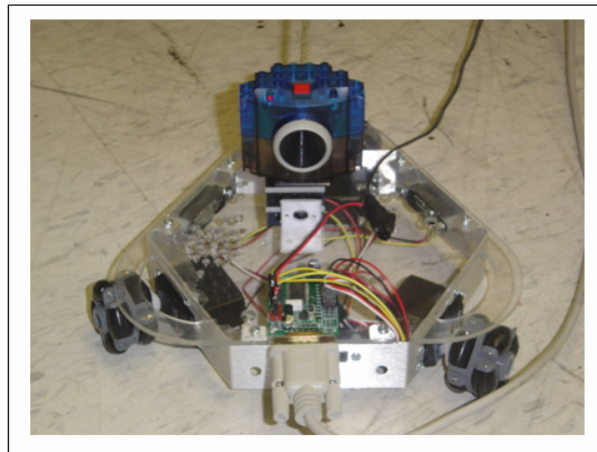


Figure 6.8 Camera attached to the PPRK

As this process is in progress the user can see the live video of the remote site through the web camera attached to the slave robot at remote site. Fig 6.9 and 6.10 shows the images of the video frames at client and server that are transferred while operating the software.



Figure 6.9 Webcam at client

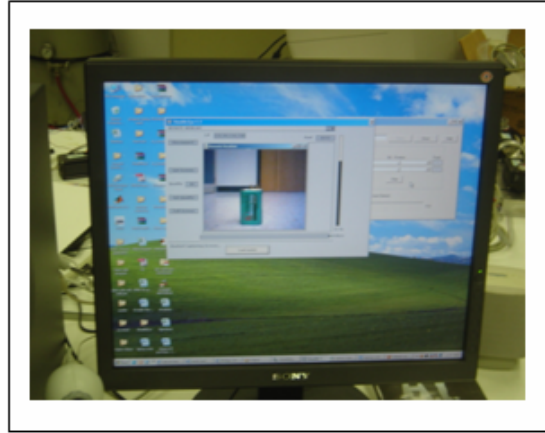


Figure 6.10 Webcam at server

CHAPTER 7

7.0 RESULTS AND DISCUSSIONS

A teleoperator system extends the intelligence and capabilities of humans and robots by force and visual feedback information. A 1-DOF manual controller has been designed and developed for this purpose which has both force and visual feedback from the remote environment (teleoperation). Design process includes motor and controller selection, and the types of transmission system used are important criteria to be considered in mechanical design of the system. Design process should also consider minimization of weight, size and, if appropriate, its cost since most of the previously designed force-reflecting manual controllers are bulky, complex and expensive, or are cheap and inaccurate designs meant for computer games.

7.1 Conclusions

In an effort to develop the 1-DOF FRMC system which meets all the above described conditions, chapter 2 reviews some of the previous designs of the force-reflecting manual controllers which include the mechanical design techniques and the control strategies. That chapter also reviews the different conceptual designs proposed for the development of the 1-DOF FRMC with advantages and disadvantages of each design.

In chapter 3, the major components, such as the actuators and sensors used in the design of 1-DOF FRMC system have been reviewed. All of the available actuators and different types of sensors available in the market are reviewed, and the suitable actuator and sensors are selected by comparing them with the rest of the available ones.

In chapter 4, the most important component in the design of 1-DOF FRMC, micro controller, is discussed. The different available micro controllers are reviewed and the final selection is presented.

In chapter 5, 1-DOF FRMC prototype is designed and developed to demonstrate the principle of 1-DOF FRMC. All of the important components used in the design of the 1-DOF FRMC have been described with their specifications. The final assembly of the system has been developed.

1-DOF platform has been constructed to check whether the prototype is working properly or not. For this purpose, a 1-DOF slider bar has been designed and constructed for use as a remote platform. The components used to develop the 1-DOF slider bar have been described in this chapter.

As a secondary remote platform, PPRK mobile robot model has been developed and its parts specifications have been listed. PPRK robot has been modified to enable its integration into the 1-DOF FRMC. This platform essentially consists of three continuous rotational servo motors, three IR (infrared) sensors and an SV203 controller.

In chapter 6, the software developed to control the 1-DOF FRMC has been briefly described. The software tool used for this purpose is Visual Basic. First, the simulation with the mouse as 1-DOF manual controller has been developed. Later, the mouse has been replaced by the 1-DOF FRMC and the simulation is carried out in the same way by utilizing the Visual Basic software.

Integration of 1-DOF FRMC with the developed platforms, i.e., 1-DOF slider bar and the modified PPRK mobile robot, have been described. In this simulation software, we have also developed visual feedback in addition to force reflection. This software was

also developed in Visual Basic. Hence, the developed system has both force and visual feedback. The software developed in Visual Basic is generic, which allows the 1-DOF FRMC control any robot built with SV203 micro controller through the internet (teleoperation).

Laboratory experiments have been conducted on the developed 1-DOF FRMC system to test the effect of time delays on the human performance. These experiments have been conducted on different users with varying time delays and the effects of these time delays have been described.

7.2 Recommendations

Although the goal of this work has been achieved, looking back at the work done, we can always identify areas where further improvements can be introduced. In this work instead of using an umbilical cord for the power supply, we could have placed a battery pack on the platform. The 1-DOF FRMC interface with the system uses RS232 port through a serial port wire, that can also be replaced with a wireless transmitter and receiver, which makes the system more efficient. The modified system can then be easily transportable and will be the same size as the one developed but it would not have any wires attached to its housing assembly.

7.3 Future work

The robot movement is not completely synchronized with the feedback received from the remote webcam in case of visual feedback system. There is a time delay of 2 to 4 seconds observed between the real time video and the video received at manual controller site.

This is due to the network problem; this limitation can be solved by using a high-speed network.

Higher degrees of freedom, such as 2-DOF and 3-DOF, manual controllers can be developed. A 2-DOF can be easily developed from the design we currently have presented. By attaching another RC servo at the other end of the mechanism, which is used to measure the shaft position of the servo, we can develop a 2-DOF FRMC.

Moreover, the slave robot movement is restricted to the maximum length of RS232 serial cable and the USB cable (web camera). This restriction can be avoided by using a wireless transmitter and receiver at the remote site, which makes the slave robot to overcome the limited boundary conditions.

Also, virtual reality (VR) environment may be developed. Various components such as the VR unit, microphone and speakers can also be integrated with the software developed, which makes the system more useful and appealing.

Furthermore, advanced control techniques may be developed to overcome the time delay problem. The wave variable technique is one such promising method currently being evaluated at FIU's Robotics and Automation Laboratory.

References

1. G. Messadie, "Great Inventions Throughout History," Chambers, Edinburgh, 1988.
2. Philip John McKerrow, "Introduction to Robotics," University of Wollongong, Australia, 2001. ISBN: 0-201-18240-8.
3. Pattaraphol Batsomboon, "Design and Construction of a Portable Force-reflecting Manual Controller for Teleoperation Systems," *Master Thesis*, Department of Mechanical Engineering, Florida International University, 1998.
4. J.V. Draper, "Teleoperator for Advanced Manufacturing: Applications and Human Factors Challenges," *The International Journal of Human Factors in Manufacturing*, Vol. 5, No. 1, pp. 53-85, 1995.
5. T. Massie, and J. Salisbury, "The PHANToM Haptic Interface: A Device For Probing Virtual Objects," *Proceedings of the ASME Winter Annual Meeting Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, DSC-Vol. 55-1, pp. 295-300, New York, 1994.
6. R. L. Williams, M. Srivastava, R. Conaster, and J. N. Howell, "Implementation and Evaluation of a Haptic Playback System," *Haptics-e: The Electronic Journal of Haptics Research*, co-sponsored by the IEEE Robotics and Automation Society, Vol. 3, No. 3, May 3, 2004.
7. M Bouzit, G. Popescu G. Burdea, and R. Boian, 2002, "The Rutgers Master II-ND Force Feedback Glove," *IEEE Virtual Reality (VR) 2002 Haptics Symposium*, Orlando FL, March 2002.
8. B. Eberman and B. An. EXOS Research on Force Reflecting Controllers. *SPIE Telemanipulator Technology*, 1833:9--19, 1992.
9. A. Kamerkar, and T. Kesavadas, "Touch Based Interactive NURBS Modeler using a Force/Position Input Glove," *ASME Design and Technology Conferences*, Salt Lake City, Utah, Sept. 2004.
10. P. Buttolo, "Characterization of Human Pen Grasp with Haptic Displays," *Ph.D. Dissertation*, University of Washington, Department of Electrical Engineering, June 1996.
11. G. Niemeyer, Kuchenbecker, K. J., Bonneau, R., Mitra, P., Reid, A. M., Fiene, J., and Weldon, G. "THUMP: an Immersive Haptic Console for Surgical Simulation and Training", *presented at MMVR: Medicine Meets Virtual Reality*, Newport Beach, California, January 15 - 17, 2004.

12. Burdea, J. Zhuang, E. Roskos, D. Silver and N. Langrana, "A Portable Dextrous Master with Force Feedback," *Presence*, Vol. 1. No.1, pp. 1827, March 1992.
13. Futaba Servo Motors, available <http://www.futabarc.com/servos/futm0029.html>, last accessed September 2004.
14. D.McCloy, D.M.J.Harris, "Robotics; An Introduction", *halsted Press*, New York, NW, 1986
15. P. Batsomboon, S. Tosunoglu, D. W. Reppeger, "Design and Construction of a Portable Force-Reflecting Manual Controller for Teleoperation Systems," *1997 Florida Conference on Recent Advances in Robotics*, Florida International University, Miami, Florida, pp. 1-7, April 10-11, 1997.
16. Chandrasekar R. Puligari, M. I. Can Dede, S. Tosunoglu, and D. W. Reppeger, "Development of a Force-reflecting Manual Controller Prototype for Teleoperation," *ASME Southeastern Region XI Technical Journal*, Volume 3, Number 1, April 2004; also presented at the ASME Southeastern Region XI Technical Conference, Mobile, Alabama, April 2-4, 2004.
17. J. Blanch, and S. Tosunoglu, "Control of Mobile Platforms via Hand-Held PDAs," *The 15th Florida Conference on Recent Advances in Robotics and Robot Showcase*, Florida International University, Miami, Florida, May 23-24, 2002.
18. N. Chopra, M.W. Spong, S. Hirche, and M. Buss, "Bilateral Teleoperation over the Internet: the Time Varying Delay Problem." *Proceedings of the American Control Conference*, Denver, CO, June 4-6, 2003.
19. Taltech Instrumental Software, <http://www.taltech.com/resources/intro-sc.html#232c>, accessed October 2004.
20. K. Mukhar, D. Johnson, "The Ultimate Palm Robot," McGraw-Hill, California, 2003, ISBN: 0-07-222880-6.
21. Lynxmotion Copmany, <http://www.lynxmotion.com/Product.aspx?productID=214&CategoryID=52>, accessed October 2004.
22. Pololu Robot-kits website, <http://www.pololu.com/products/pololu/0290/>, accessed October 2004.
23. Lynxmotion Copmany, <http://www.lynxmotion.com/Product.aspx?productID=49&CategoryID=7>, accessed October 2004.

24. Active Robots Website, <http://www.active-robots.com/products/controllr/index.shtml>, accessed October 2004.
25. <http://www.mcmanis.com/chuck/robotics/controllers/miniboard.htm>, accessed October 2004.
26. Acroname Inc, <http://www.acroname.com/robotics/parts/S1-GP-BRD.html>, accessed October 2004.
27. J. Blanch, and S. Tosunoglu, "Servo and Sensor Control of Small Mobile Platforms," *ASME Southeastern Region XI Technical Journal*, Volume 2, Number 1, April 2003; also presented at the ASME Southeastern Region XI Technical Conference, Miami, Florida, April 4-5, 2003.
28. J. Blanch, and S. Tosunoglu, "Hand-Held Computers as Mobile Platform Controllers," *Florida Conference on Recent Advances in Robotics*, Florida State University, Tallahassee, Florida, May 10-11, 2001.
29. H. J. Bosshard, I. Birrer, V. Cechticky, A. Rohlik, A. H. Glattfelder and W. Schaufelberger, "Teaching of Software for Control Systems Using Handheld and Laptop Computers and Simple Robots," *Proceedings 33rd International Symposium Ingenieurpädagogik*, Fribourg, Switzerland, September 27-October 1, 2004.
30. T.B. Sheridan, "Space Teleoperation through Time Delay: Review and Prognosis," *Proceedings of IEEE Transactions on Robotics and Automation*, Volume 9, Number 5, October 1993.
31. <http://users.ece.gatech.edu/~jfang/deepspace/transport.html>, accessed on November 2004.
32. NASA website, <http://cmex-www.arc.nasa.gov/SiteCat/sitecat2/pathfind.htm>, accessed on November 2004.

APPENDICES

APPENDIX A – VISUAL BASIC 1-DOF FRMC CONTROL PROGRAM (SIMULATION)

APPENDIX B – VISUAL BASIC 1-DOF FRMC CONTROL PROGRAM FOR TELEOPERATION OVER INTERNET

APPENDIX C – VISUAL BASIC 1-DOF MANUAL CONTROLLER CONTROL PROGRAM FOR TELEOPERATION OVER INTERNET (VISUAL FEEDBACK)

APPENDIX A – VISUAL BASIC 1-DOF FRMC CONTROL PROGRAM (SIMULATION)

We have four modules and four forms for this program. We will list all the modules first and then forms.

- **MODULE PontechComm (PontechComm.bas)**

```
Option Explicit
```

```
Const ErrSource = "PontechComm"
```

```
Enum PontechCommErr  
    PortNotOpen = 10000  
End Enum
```

```
Private Comm As MSComm
```

```
Public Sub DeclarePort(ByRef PublicCommObject As MSComm)  
    Set Comm = PublicCommObject  
End Sub
```

```
Public Sub OutputPort(ByVal OutputString As String)  
    If Comm.PortOpen = True Then  
        Comm.Output = OutputString  
    Else  
        Err.Raise PortNotOpen, ErrSource, "Port Not Open"  
    End If  
End Sub
```

```
Public Function InputPort() As String  
    Dim InputString As String  
  
    If Comm.PortOpen = True Then  
        Do While Comm.InBufferCount > 0  
            InputString = InputString + Comm.Input  
        Loop  
    Else  
        Err.Raise PortNotOpen, ErrSource, "Port Not Open"  
    End If  
    InputPort = InputString  
End Function
```

```
Public Sub OnComm()  
    Dim CommError As String  
  
    Select Case Comm.CommEvent  
        Case comEventBreak  
            CommError = "A Break signal was received."  
        Case comEventFrame  
            CommError = "Framing Error. The hardware detected a framing error."  
        Case comEventOverrun  
            CommError = "Port Overrun. A character was not read from the hardware before the next  
character arrived and was lost."  
        Case comEventRxOver
```

```

        CommError = "Receive Buffer Overflow. There is no room in the receive buffer."
    Case comEventRxParity
        CommError = "Parity Error. The hardware detected a parity error."
    Case comEventTxFull
        CommError = "Transmit Buffer Full. The transmit buffer was full while trying to queue a
character."
    Case comEventDCB
        CommError = "Unexpected error retrieving Device Control Block (DCB) for the port."
    Case comEvSend
        CommError = "There are fewer than Sthreshold number of characters in the transmit
buffer."
    Case comEvReceive
        CommError = "Received Rthreshold number of characters. This event is generated
continuously until you use the Input property to remove the data from the receive buffer."
    Case comEvCTS
        CommError = "Change in Clear To Send line."
    Case comEvDSR
        CommError = "Change in Data Set Ready line. This event is only fired when DSR
changes from 1 to 0."
    Case comEvCD
        CommError = "Change in Carrier Detect line."
    Case comEvRing
        CommError = "Ring detected. Some UARTs (universal asynchronous receiver-
transmitters) may not support this event."
    Case comEvEOF
        CommError = "End Of File (ASCII character 26) character received."
    Case Else
        CommError = "Unknown Communications Error"
    End Select
    MsgBox CommError
End Sub

```

- **MODULE SV203 (SV203.bas)**

Option Explicit

```
Const ErrSource = "SV203"
```

```
Enum SV203Err
    InvalidServoNumber = 10000
    InvalidServoPosition
    InvalidADPin
End Enum
```

```
Public Function ReadADC(ByVal AD_Pin As Integer) As Integer
    Dim Command As String
    Dim Result As String
    If AD_Pin < 1 Or AD_Pin > 5 Then
        Err.Raise InvalidADPin, ErrSource, "Invalid AD Pin Number"
    End If
    Command = Command + "AD" + CStr(AD_Pin) + Chr(13)

    Call PontechComm.OutputPort(Command)
    Call Utility_Time.Sleep(0.1)
    Result = PontechComm.InputPort

```



```

    If IsNumeric(Result) Then
        ReadADC = CInt(Result)
    End If
End Function

Public Sub ServoMove(ByVal Servo As Integer, ByVal Position As Integer)
    Static LastServo As Integer
    Dim Command As String

    If Servo < 1 Or Servo > 8 Then
        Err.Raise InvalidServoNumber, ErrSource, "Invalid Servo Number"
    End If
    If Position < 0 Or Position > 255 Then
        Err.Raise InvalidServoPosition, ErrSource, "Invalid Servo Position"
    End If
    If Servo <> LastServo Then
        Command = "SV" + CStr(Servo) + " "
        LastServo = Servo
    End If
    Command = Command + "M" + CStr(Position) + Chr(13)

    Call PontechComm.OutputPort(Command)
End Sub

```

- **MODULE Utility_Time (Utility_Time.bas)**

Option Explicit

```

Public Sub Sleep(ByVal seconds As Single)
    Dim start As Single
    start = Timer
    Do While Timer - start < seconds
        'do nothing
    Loop
End Sub

```

- **MODULE STP100 (STP100.bas)**

Option Explicit

Const ErrSource = "STP100"

```

Public Sub WriteSystemSettings()
    Call PontechComm.OutputPort("WSS" + Chr(13))
End Sub

```

```

Public Sub Halt()
    Call PontechComm.OutputPort("H0" + Chr(13))
End Sub

```

```

Public Sub HaltImmediately()
    Call PontechComm.OutputPort("HI" + Chr(13))
End Sub

```

```
Public Sub MoveCW()  
    Call PontechComm.OutputPort("H+" + Chr(13))  
End Sub
```

```
Public Sub MoveCCW()  
    Call PontechComm.OutputPort("H-" + Chr(13))  
End Sub
```

- **FORM Obstacles (Obstacles.frm)**

```
Private Sub aeroplane_Click()  
    Unload Me  
    RobotSimulation.imgSource = RobotSimulation.imgSource  
    RobotSimulation.selectedobstacle = RobotSimulation.imgSource  
    RobotSimulation.obstacletypes.Visible = True  
    RobotSimulation.selectedobstacle.Visible = True  
    RobotSimulation.msg6.Visible = True  
    RobotSimulation.msg2.Visible = True  
    RobotSimulation.msg1.Visible = False  
End Sub
```

```
Private Sub chair_Click()  
    Unload Me  
    RobotSimulation.imgSource = RobotSimulation.chair  
    RobotSimulation.selectedobstacle = RobotSimulation.imgSource  
    RobotSimulation.obstacletypes.Visible = True  
    RobotSimulation.selectedobstacle.Visible = True  
    RobotSimulation.msg6.Visible = True  
    RobotSimulation.msg2.Visible = True  
    RobotSimulation.msg1.Visible = False  
End Sub
```

```
Private Sub doll_Click()  
    Unload Me  
    RobotSimulation.imgSource = RobotSimulation.doll  
    RobotSimulation.selectedobstacle = RobotSimulation.imgSource  
    RobotSimulation.obstacletypes.Visible = True  
    RobotSimulation.selectedobstacle.Visible = True  
    RobotSimulation.msg6.Visible = True  
    RobotSimulation.msg2.Visible = True  
    RobotSimulation.msg1.Visible = False  
End Sub
```

```
Private Sub football_Click()  
    Unload Me  
    RobotSimulation.imgSource = RobotSimulation.football  
    RobotSimulation.selectedobstacle = RobotSimulation.imgSource  
    RobotSimulation.obstacletypes.Visible = True  
    RobotSimulation.selectedobstacle.Visible = True  
    RobotSimulation.msg6.Visible = True  
    RobotSimulation.msg2.Visible = True  
    RobotSimulation.msg1.Visible = False  
End Sub
```

```

Private Sub Penguin_Click()
Unload Me
RobotSimulation.imgSource = RobotSimulation.penguin
RobotSimulation.selectedobstacle = RobotSimulation.imgSource
RobotSimulation.obstacleypes.Visible = True
RobotSimulation.selectedobstacle.Visible = True
RobotSimulation.msg6.Visible = True
RobotSimulation.msg2.Visible = True
RobotSimulation.msg1.Visible = False
End Sub

```

- **FORM Robotsimulation (Robotsimulation.frm).**

```

'THIS PROGRAM WILL DEMONSTRATE 1-DOF MOVEMENT OF A ROBOT
'WITH SOME OBSTACLES IN ITS PATH

```

```

'Declaration Section

```

```

Dim amountoftimedelay, delaystart As Single

```

```

' FUNCTION THAT QUILTS THE PROGRAM

```

```

Private Sub cmdExit_Click()

```

```

    End

```

```

End Sub

```

```

Private Sub Form_Unload(Cancel As Integer)

```

```

    End

```

```

End Sub

```

```

'FUNCTION TO DRAW OBSTACLE

```

```

Private Sub obstacle_Click()

```

```

    Load obstacles

```

```

    obstacles.Show

```

```

    msg4.Visible = False

```

```

    Picture2.Visible = True

```

```

    Picture3.Visible = True

```

```

    Picture4.Visible = True

```

```

    Picture5.Visible = True

```

```

    Picture6.Visible = True

```

```

    Picture7.Visible = True

```

```

    Picture8.Visible = True

```

```

    Picture9.Visible = True

```

```

    Picture10.Visible = True

```

```

    Picture11.Visible = True

```

```

    Picture12.Visible = True

```

```

    Picture13.Visible = True

```

```

    Picture14.Visible = True

```

```

    Picture15.Visible = True

```

```

    Picture16.Visible = True

```

```

    Picture17.Visible = True

```

```

    Picture18.Visible = True

```

```

    Picture19.Visible = True

```

```
Picture20.Visible = True
Picture21.Visible = True
Picture22.Visible = True
Picture23.Visible = True
Picture24.Visible = True
Picture25.Visible = True
msg1.Visible = True
obstacle.Visible = False
Slaverobot.Visible = True
End Sub

Private Sub Form_Load()
picViewPort.Scale (26, 689)-(1005, 37)
Picture2.Visible = False
Picture3.Visible = False
Picture4.Visible = False
Picture5.Visible = False
Picture6.Visible = False
Picture7.Visible = False
Picture8.Visible = False
Picture9.Visible = False
Picture10.Visible = False
Picture11.Visible = False
Picture12.Visible = False
Picture13.Visible = False
Picture14.Visible = False
Picture15.Visible = False
Picture16.Visible = False
Picture17.Visible = False
Picture18.Visible = False
Picture19.Visible = False
Picture20.Visible = False
Picture21.Visible = False
Picture22.Visible = False
Picture23.Visible = False
Picture24.Visible = False
Picture25.Visible = False
Picture9(0).Visible = False
Picture9(1).Visible = False
Picture9(2).Visible = False
Picture9(3).Visible = False
Picture9(4).Visible = False
Picture9(5).Visible = False
Picture9(6).Visible = False
Picture9(7).Visible = False
Picture9(8).Visible = False
Picture9(9).Visible = False
Picture9(10).Visible = False
Picture9(11).Visible = False
Picture9(12).Visible = False
Picture9(13).Visible = False
Picture9(14).Visible = False
Picture9(15).Visible = False
Picture9(16).Visible = False
```

```

Picture9(17).Visible = False
Picture9(18).Visible = False
Picture9(19).Visible = False
Picture9(20).Visible = False
Picture9(21).Visible = False
Picture9(22).Visible = False
goal.Visible = False
stopsign.Visible = False
imgSource.Visible = False
msg1.Visible = False
msg2.Visible = False
msg4.Visible = True
msg5.Visible = False
msg6.Visible = False
msg3.Visible = False
Slaverobot.Visible = False
vehiclecar.Visible = False
blackmovingrobot.Visible = False
brownwalkingrobot.Visible = False
bluewalkingrobot.Visible = False
dinasour.Visible = False
imgSource.Visible = False
penguine.Visible = False
chair.Visible = False
doll.Visible = False
football.Visible = False
PositionControl.Visible = False
VelocityControl.Visible = False
time.Visible = True
sec.Visible = True
selectedrobot.Visible = False
robot.Visible = False
selectedobstacle.Visible = False
obstacleypes.Visible = False
xdir.Visible = True
ydir.Visible = True
End Sub

```

```

Private Sub Picture2_DblClick()

    Picture2.Picture = LoadPicture()
    msg5.Visible = False
    msg6.Visible = True
    PositionControl.Visible = False
    VelocityControl.Visible = False
End Sub

```

```

Private Sub picture2_click()
    Picture2.Picture = imgSource.Picture
    Picture9(0) = Picture2
    Picture9(0).Left = Picture2.Left
    Picture9(0).Top = Picture2.Top
    Picture9(0).Visible = True
    Slaverobot.Visible = True

```

```

End Sub

Private Sub Picture3_DblClick()
    msg5.Visible = False
    msg6.Visible = True
    Picture3.Picture = LoadPicture()
    PositionControl.Visible = False
    VelocityControl.Visible = False
End Sub

Private Sub picture3_click()
    Picture3.Picture = imgSource.Picture
    Picture9(1) = Picture3
    Picture9(1).Left = Picture3.Left
    Picture9(1).Top = Picture3.Top
    Picture9(1).Visible = True
    Slaverobot.Visible = True
End Sub

Private Sub Picture4_DblClick()
    msg5.Visible = False
    msg6.Visible = True
    Picture4.Picture = LoadPicture()
    PositionControl.Visible = False
    VelocityControl.Visible = False
End Sub

Private Sub picture4_click()
    Picture4.Picture = imgSource.Picture
    Picture9(2) = Picture4
    Picture9(2).Left = Picture4.Left
    Picture9(2).Top = Picture4.Top
    Picture9(2).Visible = True
    Slaverobot.Visible = True
End Sub

Private Sub Picture5_DblClick()
    msg5.Visible = False
    msg6.Visible = True
    Picture5.Picture = LoadPicture()
    PositionControl.Visible = False
    VelocityControl.Visible = False
End Sub

Private Sub picture5_click()
    Picture5.Picture = imgSource.Picture
    Picture9(3) = Picture5
    Picture9(3).Left = Picture5.Left
    Picture9(3).Top = Picture5.Top
    Picture9(3).Visible = True
    Slaverobot.Visible = True
End Sub

Private Sub Picture6_DblClick()

```

```

    msg5.Visible = False
    msg6.Visible = True
    Picture6.Picture = LoadPicture()
    PositionControl.Visible = False
    VelocityControl.Visible = False
End Sub

Private Sub picture6_click()
    Picture6.Picture = imgSource.Picture
    Picture9(4) = Picture6
    Picture9(4).Left = Picture6.Left
    Picture9(4).Top = Picture6.Top
    Picture9(4).Visible = True
    Slaverobot.Visible = True
End Sub

Private Sub Picture7_DblClick()
    msg5.Visible = False
    msg6.Visible = True
    Picture7.Picture = LoadPicture()
    PositionControl.Visible = False
    VelocityControl.Visible = False

End Sub

Private Sub picture7_click()
    Picture7.Picture = imgSource.Picture
    Picture9(5) = Picture7
    Picture9(5).Left = Picture7.Left
    Picture9(5).Top = Picture7.Top
    Picture9(5).Visible = True
    Slaverobot.Visible = True
End Sub

Private Sub Picture8_DblClick()
    msg5.Visible = False
    msg6.Visible = True
    Picture8.Picture = LoadPicture()
    PositionControl.Visible = False
    VelocityControl.Visible = False
End Sub

Private Sub picture8_click()
    Picture8.Picture = imgSource.Picture
    Picture9(6) = Picture8
    Picture9(6).Left = Picture8.Left
    Picture9(6).Top = Picture8.Top
    Picture9(6).Visible = True
    Slaverobot.Visible = True
End Sub

Private Sub Picture10_DblClick()
    msg5.Visible = False
    msg6.Visible = True

```

```

Picture10.Picture = LoadPicture()
PositionControl.Visible = False
VelocityControl.Visible = False
End Sub

Private Sub picture10_click()
Picture10.Picture = imgSource.Picture
Picture9(7) = Picture10
Picture9(7).Left = Picture10.Left
Picture9(7).Top = Picture10.Top
Picture9(7).Visible = True
Slaverobot.Visible = True
End Sub

Private Sub Picture11_DblClick()
msg5.Visible = False
msg6.Visible = True
Picture11.Picture = LoadPicture()
PositionControl.Visible = False
VelocityControl.Visible = False
End Sub

Private Sub picture11_click()
Picture11.Picture = imgSource.Picture
Picture9(8) = Picture11
Picture9(8).Left = Picture11.Left
Picture9(8).Top = Picture11.Top
Picture9(8).Visible = True
Slaverobot.Visible = True
End Sub

Private Sub Picture12_DblClick()
msg5.Visible = False
msg6.Visible = True
Picture12.Picture = LoadPicture()
PositionControl.Visible = False
VelocityControl.Visible = False
End Sub

Private Sub picture12_click()
Picture12.Picture = imgSource.Picture
Picture9(9) = Picture12
Picture9(9).Left = Picture12.Left
Picture9(9).Top = Picture12.Top
Picture9(9).Visible = True
Slaverobot.Visible = True
End Sub

Private Sub Picture13_DblClick()
msg5.Visible = False
msg6.Visible = True
Picture13.Picture = LoadPicture()
PositionControl.Visible = False
VelocityControl.Visible = False

```



```

End Sub

Private Sub picture13_click()
    Picture13.Picture = imgSource.Picture
    Picture9(10) = Picture13
    Picture9(10).Left = Picture13.Left
    Picture9(10).Top = Picture13.Top
    Picture9(10).Visible = True
    Slaverobot.Visible = True
End Sub

Private Sub Picture14_DblClick()
    msg5.Visible = False
    msg6.Visible = True
    Picture14.Picture = LoadPicture()
    PositionControl.Visible = False
    VelocityControl.Visible = False
End Sub

Private Sub picture14_click()
    Picture14.Picture = imgSource.Picture
    Picture9(11) = Picture14
    Picture9(11).Left = Picture14.Left
    Picture9(11).Top = Picture14.Top
    Picture9(11).Visible = True
    Slaverobot.Visible = True
End Sub

Private Sub Picture15_DblClick()
    msg5.Visible = False
    msg6.Visible = True
    Picture15.Picture = LoadPicture()
    PositionControl.Visible = False
    VelocityControl.Visible = False
End Sub

Private Sub picture15_click()
    Picture15.Picture = imgSource.Picture
    Picture9(12) = Picture15
    Picture9(12).Left = Picture15.Left
    Picture9(12).Top = Picture15.Top
    Picture9(12).Visible = True
    Slaverobot.Visible = True
End Sub

Private Sub Picture16_DblClick()
    msg5.Visible = False
    msg6.Visible = True
    Picture16.Picture = LoadPicture()
    PositionControl.Visible = False
    VelocityControl.Visible = False
End Sub

Private Sub picture16_click()

```

```
Picture16.Picture = imgSource.Picture
Picture9(13) = Picture16
Picture9(13).Left = Picture16.Left
Picture9(13).Top = Picture16.Top
Picture9(13).Visible = True
Slaverobot.Visible = True
End Sub
```

```
Private Sub Picture17_DblClick()
msg5.Visible = False
msg6.Visible = True
Picture17.Picture = LoadPicture()
PositionControl.Visible = False
VelocityControl.Visible = False
End Sub
```

```
Private Sub picture17_click()
Picture17.Picture = imgSource.Picture
Picture9(14) = Picture17
Picture9(14).Left = Picture17.Left
Picture9(14).Top = Picture17.Top
Picture9(14).Visible = True
Slaverobot.Visible = True
End Sub
```

```
Private Sub Picture18_DblClick()
msg5.Visible = False
msg6.Visible = True
Picture18.Picture = LoadPicture()
PositionControl.Visible = False
VelocityControl.Visible = False
End Sub
```

```
Private Sub picture18_click()
Picture18.Picture = imgSource.Picture
Picture9(15) = Picture18
Picture9(15).Left = Picture18.Left
Picture9(15).Top = Picture18.Top
Picture9(15).Visible = True
Slaverobot.Visible = True
End Sub
```

```
Private Sub Picture19_DblClick()
msg5.Visible = False
msg6.Visible = True
Picture19.Picture = LoadPicture()
PositionControl.Visible = False
VelocityControl.Visible = False
End Sub
```

```
Private Sub picture19_click()
Picture19.Picture = imgSource.Picture
Picture9(16) = Picture19
Picture9(16).Left = Picture19.Left
```

```

Picture9(16).Top = Picture19.Top
Picture9(16).Visible = True
Slaverobot.Visible = True
End Sub

Private Sub Picture20_DblClick()
    msg5.Visible = False
    msg6.Visible = True
    Picture20.Picture = LoadPicture()
    PositionControl.Visible = False
    VelocityControl.Visible = False
End Sub

Private Sub picture20_click()
    Picture20.Picture = imgSource.Picture
    Picture9(17) = Picture20
    Picture9(17).Left = Picture20.Left
    Picture9(17).Top = Picture20.Top
    Picture9(17).Visible = True
    Slaverobot.Visible = True
End Sub

Private Sub Picture21_DblClick()
    msg5.Visible = False
    msg6.Visible = True
    Picture21.Picture = LoadPicture()
    PositionControl.Visible = False
    VelocityControl.Visible = False
End Sub

Private Sub picture21_click()
    Picture21.Picture = imgSource.Picture
    Picture9(18) = Picture21
    Picture9(18).Left = Picture21.Left
    Picture9(18).Top = Picture21.Top
    Picture9(18).Visible = True
    Slaverobot.Visible = True
End Sub

Private Sub Picture22_DblClick()
    msg5.Visible = False
    msg6.Visible = True
    Picture22.Picture = LoadPicture()
    PositionControl.Visible = False
    VelocityControl.Visible = False
End Sub

Private Sub picture22_click()
    Picture22.Picture = imgSource.Picture
    Picture9(19) = Picture23
    Picture9(19).Left = Picture22.Left
    Picture9(19).Top = Picture22.Top
    Picture9(19).Visible = True
    Slaverobot.Visible = True

```

```

End Sub

Private Sub Picture23_DblClick()
    msg5.Visible = False
    msg6.Visible = True
    Picture23.Picture = LoadPicture()
    PositionControl.Visible = False
    VelocityControl.Visible = False
End Sub

Private Sub picture23_click()
    Picture23.Picture = imgSource.Picture
    Picture9(20) = Picture23
    Picture9(20).Left = Picture23.Left
    Picture9(20).Top = Picture23.Top
    Picture9(20).Visible = True
    Slaverobot.Visible = True
End Sub

Private Sub Picture24_DblClick()
    msg5.Visible = False
    msg6.Visible = True
    Picture24.Picture = LoadPicture()
    PositionControl.Visible = False
    VelocityControl.Visible = False
End Sub

Private Sub picture24_click()
    Picture24.Picture = imgSource.Picture
    Picture9(21) = Picture24
    Picture9(21).Left = Picture24.Left
    Picture9(21).Top = Picture24.Top
    Picture9(21).Visible = True
    Slaverobot.Visible = True
End Sub

Private Sub Picture25_DblClick()
    msg5.Visible = False
    msg6.Visible = True
    Picture25.Picture = LoadPicture()
    PositionControl.Visible = False
    VelocityControl.Visible = False
End Sub

Private Sub picture25_click()
    Picture25.Picture = imgSource.Picture
    Picture9(22) = Picture25
    Picture9(22).Left = Picture25.Left
    Picture9(22).Top = Picture25.Top
    Picture9(22).Visible = True
    Slaverobot.Visible = True
End Sub

Private Sub Slaverobot_Click()

```

```
Load slaverobots
slaverobots.Show
PositionControl.Visible = True
VelocityControl.Visible = True
msg2.Visible = False
msg6.Visible = False
msg5.Visible = False
msg3.Visible = True
Slaverobot.Visible = False
```

```
If Picture2.Picture = 0 Then
    Picture2.Visible = False
End If
If Picture3.Picture = 0 Then
    Picture3.Visible = False
End If
If Picture4.Picture = 0 Then
    Picture4.Visible = False
End If
If Picture5.Picture = 0 Then
    Picture5.Visible = False
End If
If Picture6.Picture = 0 Then
    Picture6.Visible = False
End If
If Picture7.Picture = 0 Then
    Picture7.Visible = False
End If
If Picture8.Picture = 0 Then
    Picture8.Visible = False
End If
If Picture10.Picture = 0 Then
    Picture10.Visible = False
End If
If Picture11.Picture = 0 Then
    Picture11.Visible = False
End If
If Picture12.Picture = 0 Then
    Picture12.Visible = False
End If
If Picture13.Picture = 0 Then
    Picture13.Visible = False
End If
If Picture14.Picture = 0 Then
    Picture14.Visible = False
End If
If Picture15.Picture = 0 Then
    Picture15.Visible = False
End If
If Picture16.Picture = 0 Then
    Picture16.Visible = False
End If
If Picture17.Picture = 0 Then
    Picture17.Visible = False
```

```

End If
If Picture18.Picture = 0 Then
    Picture18.Visible = False
End If
If Picture19.Picture = 0 Then
    Picture19.Visible = False
End If
If Picture20.Picture = 0 Then
    Picture20.Visible = False
End If
If Picture21.Picture = 0 Then
    Picture21.Visible = False
End If
If Picture22.Picture = 0 Then
    Picture22.Visible = False
End If
If Picture23.Picture = 0 Then
    Picture23.Visible = False
End If
If Picture24.Picture = 0 Then
    Picture24.Visible = False
End If
If Picture25.Picture = 0 Then
    Picture25.Visible = False
End If
End Sub

```

```

Private Sub PositionControl_Click()
goal.Visible = True
VelocityControl.Visible = False
msg3.Visible = False
RobotSimulation.Show
Servoposition.Show
End Sub

```

```

Private Sub VelocityControl_Click()
goal.Visible = True
PositionControl.Visible = False
msg3.Visible = False
RobotSimulation.Show
Servoposition.Show
End Sub

```

```

Private Sub withouttimedelay_Click()
withtimedelay.Visible = True
time.Visible = True
sec.Visible = True
withouttimedelay.Visible = False
End Sub

```

```

Private Sub withtimedelay_Click()
time.Visible = False
sec.Visible = False

```

```

amountoftimedelay = time.Text
withouttimedelay.Visible = True
withtimedelay.Visible = False
End Sub

```

```

Private Sub xdir_Click()
xdir.Visible = False
ydir.Visible = True
End Sub

```

```

Private Sub ydir_Click()
ydir.Visible = False
xdir.Visible = True
End Sub

```

- **FORM Servoposition (Servoposition.frm)**

```
Option Explicit
```

```

Dim ADreading(1 To 5, 1 To 12) As Integer
Dim i, j, k, x1, y1, x, yfront, yback As Integer
Dim xback, xfront, circles, radius As Single
Dim counter, click As Long
Dim BeginTime As Date
Dim FinishTime As Date
Dim ElapsedTime, newelapsetime As Long
Dim ADchannel, ADold, firstcase As Integer
Dim x1new, y1new As Integer

```

```

Public Sub Sleep(ByVal seconds As Single)
    Dim start As Single
    start = Timer
    Do While Timer - start < 1
        'do nothing
    Loop
End Sub

```

```

Private Sub Form_Load()
    Call PontechComm.DeclarePort(MSComm1)
    CloseComm.Enabled = False
    OpenComm.Enabled = True
    counter = 1
    click = 1
    newelapsetime = 0
    firstcase = 1
    ADchannel = 1
End Sub

```

```

Private Sub MSComm1_OnComm()
    PontechComm.OnComm
End Sub

```

```

Private Sub OpenComm_Click()

    If MSComm1.PortOpen = True Then

```

```

    MSComm1.PortOpen = False
End If
MSComm1.CommPort = CInt(CommPort.Text)
MSComm1.PortOpen = True
CloseComm.Enabled = True
OpenComm.Enabled = False
'clear bars and values for servos
Dim s As Integer
For s = 1 To 2
    ServoPositionLbl(s).Text = 0
    ServoBitLbl(s).Text = 128
    ServoScroll(s).Value = 128
    ServoNumber(s).Text = CInt(6 + s)      'IRs are servos 7 & 8 = 6 (+1 / +2)
    Call SV203.ServoMove(CInt(ServoNumber(s).Text), 128)
Next
'clear AD input pictures
Dim i As Integer
For i = 1 To 1
    SensorPic(i).Scale (-12, 300)-(1, -5)
    SensorPic(i).Cls
    SensorPic(i).AutoRedraw = True
Next i
Call StopSRV_Click
End Sub

Private Sub CloseComm_Click()
    MSComm1.PortOpen = False
    CloseComm.Enabled = False
    OpenComm.Enabled = True
End Sub

Private Sub AutoRefresh_Click()
    If AutoRefresh.Value = vbChecked Then
        Timer1.Enabled = True
    Else
        Timer1.Enabled = False
    End If
End Sub

Private Sub Timer1_Timer()
    Call Refresh_Click
End Sub

Private Sub Refresh_Click()
'Get the beginning time
'Servoposition.Hide
'RobotSimulation.Show

    On Error GoTo SubError

    For i = 0 To 22

        x1 = RobotSimulation.vehiclecar.Left
        xfront = RobotSimulation.Picture9(i).Left

```



```

y1 = RobotSimulation.vehiclecar.Top
yfront = RobotSimulation.Picture9(i).Top

If RobotSimulation.Picture9(i).Visible = True Then

counter = 1
If ADchannel = 1 Then      'there are 5 AD channels, joystick is #1 (#2 for 2nd DOF)
    'push back old readings
    'For ADold = 1 To 2
        ADreading(ADchannel, ADold + 1) = ADreading(ADchannel, ADold)
    'Next ADold
    'new readings
    ADreading(ADchannel, 1) = CStr(SV203.ReadADC(ADchannel))
    ADC_Value(ADchannel).Caption = ADreading(ADchannel, 1)
    If firstcase = 1 Then
        ADreading(1, 2) = 128
        ADreading(1, 1) = 127
        firstcase = 2
    End If

    'if the last input has a large delta larger than 5 over the previous instruction, wait extra
time
    If Abs(ADreading(ADchannel, 2) - ADreading(ADchannel, 1)) > 10 Then
        If RobotSimulation.withtimedelay.Visible = False Then
            Call timedelay(x1)
        End If
    End If

    Call finaldestination

    'Control for DOF
1=====
    If ADchannel = 1 Then

        If RobotSimulation.xdir.Visible = False Then '////////***** if for x-direction *****//////////

            If ADreading(1, 1) < 100 Then
                Call jerkback
            End If

            If ADreading(1, 1) > 148 Then
                Call jerkfront
            End If

            'Position Control

            If RobotSimulation.PositionControl.Visible = True Then
                'going front

                If ADreading(1, 1) < 125 Then

                    If x1 > 900 Then
                        Call jerkback
                    End If
                End If
            End If
        End If
    End If

```

Call draw_circles_front

```
'going front
If counter = 2 Then
  Call jerkback
  x1 = x1 - 3
Else
  x1 = x1 + 3
End If
'going back
Elseif ADreading(1, 1) > 130 Then

  Call draw_circles_back

  If x1 < 50 Then
    Call jerkfront
  End If

  If counter = 2 Then
    Call jerkfront
    x1 = x1 + 3
  Else
    x1 = x1 - 3
  End If

End If *****End of going front/back*****

End If *****End Of position Control *****
```

' Velocity Control

If RobotSimulation.VelocityControl.Visible = True Then

```
'going front
If ADreading(1, 1) < 125 Then

  If x1 > 900 Then
    Call jerkback
  End If

  If ADreading(1, 1) < 125 And ADreading(1, 1) > 119 Then

    Call draw_circles_front
    If counter = 2 Then
      Call jerkback
      x1 = x1 - 5
    Else
      x1 = x1 + 5
    End If

  Elseif ADreading(1, 1) < 120 And ADreading(1, 1) > 114 Then
```

```

        Call draw_circles_front
    If counter = 2 Then
        Call jerkback
        x1 = x1 - 10
    Else
        x1 = x1 + 10
    End If

Elseif ADreading(1, 1) < 115 And ADreading(1, 1) > 109 Then

    Call draw_circles_front
    If counter = 2 Then
        Call jerkback
        x1 = x1 - 15
    Else
        x1 = x1 + 15
    End If

Elseif ADreading(1, 1) < 110 And ADreading(1, 1) > 100 Then
    Call draw_circles_front
    If counter = 2 Then
        Call jerkback
        x1 = x1 - 20
    Else
        x1 = x1 + 20
    End If
End If
End If '***** going Front*****
'going back

If ADreading(1, 1) > 130 Then

    If x1 < 50 Then
        Call jerkfront
    End If

If ADreading(1, 1) > 130 And ADreading(1, 1) < 136 Then

    Call draw_circles_back
    If counter = 2 Then
        Call jerkfront
        x1 = x1 + 5
    Else
        x1 = x1 - 5
    End If

Elseif ADreading(1, 1) > 135 And ADreading(1, 1) < 141 Then

    Call draw_circles_back
    If counter = 2 Then
        Call jerkfront
        x1 = x1 + 10
    Else
        x1 = x1 - 10

```

```

End If

Elseif ADreading(1, 1) > 140 And ADreading(1, 1) < 146 Then

    Call draw_circles_back
    If counter = 2 Then
        Call jerkfront
        x1 = x1 + 15
    Else
        x1 = x1 - 15
    End If

Elseif ADreading(1, 1) > 145 And ADreading(1, 1) < 148 Then

    Call draw_circles_back
    If counter = 2 Then
        Call jerkfront
        x1 = x1 + 20
    Else
        x1 = x1 - 20
    End If
End If
End If *****close going back*****

End If *****close Velocity control*****

End If '////////***** if close for x-direction *****

If RobotSimulation.ydir.Visible = False Then '////////***** if for y-direction *****////////
    If ADreading(1, 1) < 100 Then
        Call jerkfront
    End If

    If ADreading(1, 1) > 148 Then
        Call jerkback
    End If
'Position Control

If RobotSimulation.PositionControl.Visible = True Then
'going up *****
    If ADreading(1, 1) < 125 Then

        If y1 > 680 Then
            Call jerkback
        End If
        Call draw_circles_up
        'going up
        If counter = 2 Then
            Call jerkback
            y1 = y1 - 3
        Else
            y1 = y1 + 3
        End If
    End If

```

```

'going down*****
Elseif ADreading(1, 1) > 130 Then
  If y1 < 120 Then
    Call jerkfront
  End If
  Call draw_circles_down

  If counter = 2 Then
    Call jerkfront
    y1 = y1 + 3
  Else
    y1 = y1 - 3
  End If

End If '/////end of going up and down /////
End If '*****End Of position control loop *****
' Velocity Control
If RobotSimulation.VelocityControl.Visible = True Then
  'going up
  If ADreading(1, 1) < 125 Then
    If y1 > 680 Then
      Call jerkback
    End If
    If ADreading(1, 1) < 125 And ADreading(1, 1) > 119 Then
      Call draw_circles_up
      If counter = 2 Then
        Call jerkback
        y1 = y1 - 5
      Else
        y1 = y1 + 5
      End If

    Elseif ADreading(1, 1) < 120 And ADreading(1, 1) > 114 Then
      Call draw_circles_up
      If counter = 2 Then
        Call jerkback
        y1 = y1 - 10
      Else
        y1 = y1 + 10
      End If

    Elseif ADreading(1, 1) < 115 And ADreading(1, 1) > 109 Then

      Call draw_circles_up
      If counter = 2 Then
        Call jerkback
        y1 = y1 - 15
      Else
        y1 = y1 + 15
      End If
    Elseif ADreading(1, 1) < 110 And ADreading(1, 1) > 100 Then
      Call draw_circles_up
      If counter = 2 Then

```

```

        Call jerkback
        y1 = y1 - 20
    Else
        y1 = y1 + 20
    End If
End If
End If '*****going up*****'
'going down
If ADreading(1, 1) > 130 Then
If y1 < 170 Then
    Call jerkfront
End If

If ADreading(1, 1) > 130 And ADreading(1, 1) < 136 Then
    Call draw_circles_down
    If counter = 2 Then
        Call jerkfront
        y1 = y1 + 5
    Else
        y1 = y1 - 5
    End If

ElseIf ADreading(1, 1) > 135 And ADreading(1, 1) < 141 Then
    Call draw_circles_down
    If counter = 2 Then
        Call jerkfront
        y1 = y1 + 10
    Else
        y1 = y1 - 10
    End If

ElseIf ADreading(1, 1) > 140 And ADreading(1, 1) < 146 Then
    Call draw_circles_down
    If counter = 2 Then
        Call jerkfront
        y1 = y1 + 15
    Else
        y1 = y1 - 15
    End If

ElseIf ADreading(1, 1) > 145 And ADreading(1, 1) < 148 Then
    Call draw_circles_down
    If counter = 2 Then
        Call jerkfront
        y1 = y1 + 20
    Else
        y1 = y1 - 20
    End If
End If
End If '*****close going down*****'
End If '*****close velocity control*****'
End If "End Of y - direction loop =====*****=====
End If 'ADchannel = 1

```

```

        ADreading(ADchannel, 2) = ADreading(ADchannel, 1)
    If RobotSimulation.xdir.Visible = False Then
        RobotSimulation.vehiclecar.Left = x1
    End If

    If RobotSimulation.ydir.Visible = False Then
        RobotSimulation.vehiclecar.Top = y1
    End If
End If 'ADChannel End
End If
Next i
Exit Sub
SubError:
    AutoRefresh.Value = vbUnchecked
    MsgBox Err.Description
End Sub

Private Sub ServoScroll_Change(Index As Integer)
    Dim Position As Integer
    Dim Value As Integer
    Dim ScIInput As Integer
    ScIInput = ServoScroll(Index).Value
    Position = ServoPos(Index, ServoValue(Index, ScIInput))
    Value = ServoValue(Index, ScIInput)
    If MSComm1.PortOpen = True Then
        Call SV203.ServoMove(CInt(ServoNumber(Index).Text), Value)
    End If
    ServoBitLbl(Index).Text = CStr(Value)
    ServoPositionLbl(Index).Text = CStr(Position)
End Sub

Private Sub ServoScroll_Scroll(Index As Integer)
    Dim Position As Integer
    Dim Value As Integer
    Dim ScIInput As Integer
    ScIInput = ServoScroll(Index).Value
    Position = ServoPos(Index, ServoValue(Index, ScIInput))
    Value = ServoValue(Index, ScIInput)
    If MSComm1.PortOpen = True Then
        Call SV203.ServoMove(CInt(ServoNumber(Index).Text), Value)
    End If
    ServoBitLbl(Index).Text = CStr(Value)
    ServoPositionLbl(Index).Text = CStr(Position)
End Sub

Private Sub StopSRV_Click()
    If MSComm1.PortOpen = True Then
        Call SV203.ServoMove(8, 0)
    End If
End Sub

Private Sub left_Click()
    If MSComm1.PortOpen = True Then
        Call Sleep(1)
    End If
End Sub

```

```

        Call SV203.ServoMove(8, 100)
    End If
End Sub

Private Sub right_Click()
    If MSComm1.PortOpen = True Then
        Call Sleep(1)
        Call SV203.ServoMove(8, 150)
    End If
End Sub

Private Function ServoPos(ByVal Servo As Integer, ByVal Bit As Integer) As Integer
    ServoPos = Bit - 128
    If Servo = 2 Then
        ServoPos = -ServoPos
    End If
End Function

Private Function ServoValue(ByVal Servo As Integer, ByVal Bit As Integer) As Integer
    If Servo = 1 Then
        ServoValue = Bit
    End If
    If Servo = 2 Then
        ServoValue = Bit + 1           'calibration offset
        ServoValue = ServoValue - 128 'Change direction, step 1
        ServoValue = 128 - ServoValue 'Change direction, step 2
    End If
End Function

Private Sub jerkfront()
    Call SV203.ServoMove(8, 138)
    Call SV203.ServoMove(8, 148)
    Call SV203.ServoMove(8, 138)
    Call StopSRV_Click
End Sub

Private Sub jerkback()
    Call SV203.ServoMove(8, 114)
    Call SV203.ServoMove(8, 104)
    Call SV203.ServoMove(8, 114)
    Call StopSRV_Click
End Sub

Private Sub draw_circles_front()

    'drawing circles when going front or up
    If Abs(xfront - x1) < RobotSimulation.vehiclecar.Width + 75 Then
        If Abs(y1 - yfront) < 100 Then
            radius = 70
            For circles = 1 To 30
                RobotSimulation.picViewPort.Circle (xfront + 20, yfront - 20), radius, vbGreen
                radius = radius - 0.5
            Next circles
        End If
    End If
End Sub

```



```

        If Abs(xfront - x1) < RobotSimulation.vehiclecar.Width + 65 Then
        If Abs(y1 - yfront) < 75 Then
            radius = 60
            For circles = 1 To 20
                RobotSimulation.picViewPort.Circle (xfront + 20, yfront - 20), radius,
vbYellow
                    radius = radius - 0.5
                Next circles
            End If
        End If

        If Abs(xfront - x1) < RobotSimulation.vehiclecar.Width + 50 Then
        If Abs(y1 - yfront) < 50 Then
            radius = 50
            For circles = 1 To 200
                RobotSimulation.picViewPort.Circle (xfront + 20, yfront - 20), radius, vbRed
            radius = radius - 0.25
            Next circles
            Call jerkback
            counter = 2
        End If
        End If
    End Sub

Private Sub draw_circles_back()
    'drawing circles when going back
    If Abs(xfront - x1) < RobotSimulation.vehiclecar.Width + 75 Then
        If Abs(y1 - yfront) < 100 Then
            radius = 70
            For circles = 1 To 30
                RobotSimulation.picViewPort.Circle (xfront + 20, yfront - 20), radius, vbGreen
            radius = radius - 0.5
            Next circles
        End If
    End If

    If Abs(xfront - x1) < RobotSimulation.vehiclecar.Width + 65 Then
        If Abs(y1 - yfront) < 75 Then
            radius = 60
            For circles = 1 To 20
                RobotSimulation.picViewPort.Circle (xfront + 20, yfront - 20), radius,
vbYellow
                    radius = radius - 0.5
                Next circles
            End If
        End If

        If Abs(xfront - x1) < RobotSimulation.vehiclecar.Width + 50 Then
        If Abs(y1 - yfront) < 50 Then
            radius = 50
            For circles = 1 To 200
                RobotSimulation.picViewPort.Circle (xfront + 20, yfront - 20), radius, vbRed
            radius = radius - 0.25
            Next circles
        End If
    End If

```

```

        Call jerkfront
        counter = 2
    End If
End If
End Sub

Private Sub draw_circles_down()
'drawing circles when going down

        If Abs(xfront - x1) < RobotSimulation.vehiclecar.Width + 75 Then
            If Abs(y1 - yfront) < 100 Then
                radius = 70
                For circles = 1 To 30
                    RobotSimulation.picViewPort.Circle (xfront + 20, yfront - 20), radius,
vbGreen
                        radius = radius - 0.5
                Next circles
            End If
        End If

        If Abs(xfront - x1) < RobotSimulation.vehiclecar.Width + 65 Then
            If Abs(y1 - yfront) < 75 Then
                radius = 60
                For circles = 1 To 20
                    RobotSimulation.picViewPort.Circle (xfront + 20, yfront - 20), radius,
vbYellow
                        radius = radius - 0.5
                Next circles
            End If
        End If

        If Abs(xfront - x1) < RobotSimulation.vehiclecar.Width + 50 Then
            If Abs(y1 - yfront) < 50 Then
                radius = 50
                For circles = 1 To 200
                    RobotSimulation.picViewPort.Circle (xfront + 20, yfront - 20), radius,
vbRed
                        radius = radius - 0.25
                Next circles
                Call jerkfront
                counter = 2
            End If
        End If
    End Sub

Private Sub draw_circles_up()
'drawing circles when going down

        If Abs(xfront - x1) < RobotSimulation.vehiclecar.Width + 75 Then
            If Abs(y1 - yfront) < 100 Then
                radius = 70
                For circles = 1 To 30

```

```

        RobotSimulation.picViewPort.Circle (xfront + 20, yfront - 20), radius,
vbGreen        radius = radius - 0.5
                Next circles
                End If
            End If

            If Abs(xfront - x1) < RobotSimulation.vehiclecar.Width + 65 Then
                If Abs(y1 - yfront) < 75 Then
                    radius = 60
                    For circles = 1 To 20
vbYellow        RobotSimulation.picViewPort.Circle (xfront + 20, yfront - 20), radius,
                    radius = radius - 0.5
                    Next circles
                End If
            End If

            If Abs(xfront - x1) < RobotSimulation.vehiclecar.Width + 50 Then
                If Abs(y1 - yfront) < 50 Then
                    radius = 50
                    For circles = 1 To 200
vbRed          RobotSimulation.picViewPort.Circle (xfront + 20, yfront - 20), radius,
                    radius = radius - 0.25
                    Next circles
                    Call jerkfront
                    counter = 2
                End If
            End If
        End Sub

```

```
Private Sub timedelay(x1new)
```

```
    Servoposition.Hide
    RobotSimulation.Show
```

```
    'Timer1.Enabled = False
    Timer2.Enabled = True
    Dim newtime As Integer
    newtime = 0
    For i = 0 To 22
```

```
        x1new = RobotSimulation.vehiclecar.Left
        xfront = RobotSimulation.Picture9(i).Left
```

```
        y1new = RobotSimulation.vehiclecar.Top
        yfront = RobotSimulation.Picture9(i).Top
```

```
        If RobotSimulation.Picture9(i).Visible = True Then
            While newtime <= RobotSimulation.time.Text
                'On Error GoTo SubError
                counter = 1
            End While
        End If
    Next i
End Sub

```

```

'Control for DOF
1=====
  If ADchannel = 1 Then

      If RobotSimulation.xdir.Visible = False Then '/////***** if for x-direction
*****/////
          If ADreading(1, 2) < 100 Then
              Call jerkback
          End If
          If ADreading(1, 2) > 148 Then
              Call jerkfront
          End If
          'Position Control
          If RobotSimulation.PositionControl.Visible = True Then
              'going front
              If ADreading(1, 2) < 125 Then
                  If x1new > 900 Then
                      Call jerkback
                  End If
                  Call draw_circles_front
                  'going front
                  If counter = 2 Then
                      Call jerkback
                      x1new = x1new - 1
                  Else
                      x1new = x1new + 1
                  End If
                  'going back
              ElseIf ADreading(1, 2) > 130 Then
                  Call draw_circles_back
                  If x1new < 50 Then
                      Call jerkfront
                  End If
                  If counter = 2 Then
                      Call jerkfront
                      x1new = x1new + 1
                  Else
                      x1new = x1new - 1
                  End If
              End If '****End of going front/back*****
          End If '*****End Of position Control *****

          ' Velocity Control
          If RobotSimulation.VelocityControl.Visible = True Then
              'going front
              If ADreading(1, 2) < 125 Then
                  If x1new > 900 Then
                      Call jerkback
                  End If
                  x1new = x1new + 5
              End If '***** close going Front*****
              'going back
              If ADreading(1, 2) > 130 Then
                  If x1new < 50 Then

```

```

        Call jerkfront
    End If
    x1new = x1new - 5
    End If *****close going back*****
End If *****close Velocity control*****
End If '/////***** if close for x-direction *****

If RobotSimulation.ydir.Visible = False Then '** if for y-direction *****//////////
    If ADreading(1, 2) < 100 Then
        Call jerkfront
    End If
    If ADreading(1, 2) > 148 Then
        Call jerkback
    End If

'Position Control
If RobotSimulation.PositionControl.Visible = True Then
    'going up *****
    If ADreading(1, 2) < 125 Then
        If y1new > 680 Then
            Call jerkback
        End If
        Call draw_circles_up
        'going up
        If counter = 2 Then
            Call jerkback
            y1new = y1new - 3
        Else
            y1new = y1new + 3
        End If
        'going down*****
    ElseIf ADreading(1, 2) > 130 Then
        If y1new < 170 Then
            Call jerkfront
        End If
        Call draw_circles_down
        If counter = 2 Then
            Call jerkfront
            y1new = y1new + 3
        Else
            y1new = y1new - 3
        End If

    End If '/////end of going up and down /////

End If *****End Of position control loop *****

' Velocity Control

If RobotSimulation.VelocityControl.Visible = True Then
    'going up

```

```

        If ADreading(1, 2) < 125 Then
            If y1new > 680 Then
                Call jerkback
            End If
            Call draw_circles_up
            y1new = y1new + 5
        End If '*****going up*****

        'going down
        If ADreading(1, 2) > 130 Then
            If y1new < 170 Then
                Call jerkfront
            End If
            Call draw_circles_down
            y1new = y1new - 5
        End If '*****close going down*****
    End If '*****close velocity control*****
End If "End Of y - direction loop =====*****=====

End If 'ADchannel = 1
    'draw readings
    'SensorPic(ADchannel).Cls
    If RobotSimulation.xdir.Visible = False Then
        Sleep 0.000001
        RobotSimulation.vehiclecar.Left = x1new
    End If

    If RobotSimulation.ydir.Visible = False Then
        Sleep 0.000001
        RobotSimulation.vehiclecar.Top = y1new
    End If
    newtime = newtime + Timer2.Interval
    'Debug.Print newtime
Wend
End If
Next i
End Sub
Private Sub finaldestination()
    If Abs(x1 - RobotSimulation.goal.Left) < RobotSimulation.goal.Width + 10 Then
        If Abs(y1 - RobotSimulation.goal.Top) < RobotSimulation.goal.Height + 10 Then
            RobotSimulation.goal.Visible = False
            RobotSimulation.stopsign.Visible = True
        End If
    Else
        RobotSimulation.goal.Visible = True
        RobotSimulation.stopsign.Visible = False
    End If
End Sub
End Sub

```

- **FORM Slaverobots (Slaverobot.frm)**

```

Private Sub black_Click()
    Unload Me

```

```

    RobotSimulation.vehiclecar = RobotSimulation.blackmovingrobot
    RobotSimulation.selectedrobot = RobotSimulation.vehiclecar
    RobotSimulation.robot.Visible = True
    RobotSimulation.selectedrobot.Visible = True
    RobotSimulation.vehiclecar.Visible = True
End Sub

Private Sub blue_Click()
    Unload Me
    RobotSimulation.vehiclecar = RobotSimulation.bluewalkingrobot
    RobotSimulation.selectedrobot = RobotSimulation.vehiclecar
    RobotSimulation.robot.Visible = True
    RobotSimulation.selectedrobot.Visible = True
    RobotSimulation.vehiclecar.Visible = True
End Sub

Private Sub brown_Click()
    Unload Me
    RobotSimulation.vehiclecar = RobotSimulation.brownwalkingrobot
    RobotSimulation.selectedrobot = RobotSimulation.vehiclecar
    RobotSimulation.robot.Visible = True
    RobotSimulation.selectedrobot.Visible = True
    RobotSimulation.vehiclecar.Visible = True
End Sub

Private Sub Car_Click()
    Unload Me
    RobotSimulation.vehiclecar = RobotSimulation.vehiclecar
    RobotSimulation.selectedrobot = RobotSimulation.vehiclecar
    RobotSimulation.robot.Visible = True
    RobotSimulation.selectedrobot.Visible = True
    RobotSimulation.vehiclecar.Visible = True
End Sub

Private Sub dinasour_Click()
    Unload Me
    RobotSimulation.vehiclecar = RobotSimulation.dinasour
    RobotSimulation.selectedrobot = RobotSimulation.vehiclecar
    RobotSimulation.robot.Visible = True
    RobotSimulation.selectedrobot.Visible = True
    RobotSimulation.vehiclecar.Visible = True
End Sub

```

APPENDIX B – VISUAL BASIC 1-DOF FRMC CONTROL PROGRAM FOR TELEOPERATION OVER INTERNET

We have four modules and five forms for this program on the server side and four modules and two forms on the client side.

We will list all the modules first and then forms on the server side.

SERVER SIDE MODULES

- **MODULE PontechComm (PontechComm.bas)**

Code for this module is same as the code listed for Module PontechComm in Appendix A

- **MODULE SV203 (SV203.bas)**

Code for this module is same as the code listed for Module SV203 in Appendix A

- **MODULE Utility_Time (Utility_Time.bas)**

Code for this module is same as the code listed for Module Utility_Time in Appendix A

- **MODULE STP100 (STP100.bas)**

Code for this module is same as the code listed for STP100 in Appendix A

SERVER SIDE FORMS

- **FORM Masterrobot (Masterrobot.frm)**

Option Explicit

```
Public Sub CmdLoad_Click()  
    Load Robotsimulation  
    Robotsimulation.Visible = True  
End Sub
```

```
Sub SendData(data As String)  
'Check to see if we're connected to a client  
If Winsock.Tag = "CONNECTED" Then  
    'Send the data  
    Winsock.SendData data & vbCrLf  
End If  
End Sub
```

```
Sub Status(data As String)  
'Update the status label  
lblStatus.Caption = "Status: " & data  
End Sub  
Private Sub cmdSend_Click()  
'Send the data to the client if it  
'is not blank  
If txtSend.Text <> "" Then  
    SendData txtSend.Text  
    txtSend.Text = ""  
End If  
End Sub
```



```

Private Sub Form_Load()
'Set the caption with your IP
lblIP.Caption = "your ip: " & Winsock.LocalIP
'Listen for incoming connection requests
Winsock.Listen
Status "Awaiting connection.."
End Sub

Private Sub txtData_Change()
'Set the cursor to the last character of the textbox
txtData.SelStart = Len(txtData.Text)
End Sub

Private Sub Winsock_Close()
'Client closed connection, close the Winsock on this side
Winsock.Close
Winsock.Tag = "CLOSED"
'Update status
Status "Connection closed, awaiting new connection.."
'Re-listen for incoming connection requests
Winsock.Listen
End Sub

Private Sub Winsock_ConnectionRequest(ByVal requestID As Long)
'Update status
Status "Accepting connection request"
'Close winsock
Winsock.Close
'Accept the connection request
Winsock.Accept requestID
Winsock.Tag = "CONNECTED"
'Update status
Status "Connected"
End Sub

Private Sub Winsock_DataArrival(ByVal bytesTotal As Long)
Dim Buffer As String
'Update status
Status "Data has arrived"
'Get the data being sent by the client
Winsock.GetData Buffer
'Put incoming data into the Data textbox
txtData = txtData & "" & Buffer
Buffer = UCase(Buffer)
Buffer = Left(Buffer, Len(Buffer) - 2)
Buffer = Left(Buffer, 3)
txtData.Text = Buffer
Call servoposition.Refresh_Click
'Update the status back to "Connected"
Status "Connected"
End Sub

```

- **FORM Obstacles (Obstacles.frm)**

Code for this form is same as the code listed for Form Obstacles in Appendix A

- **FORM Slaverobots (Slaverobots.frm)**

Code for this form is same as the code listed for Form Slaverobots in Appendix A

- **FORM Robotsimulation (Robotsimulation.frm)**

'THIS PROGRAM WILL DEMONSTRATE 1-DOF MOVEMENT OF A ROBOT
'WITH SOME OBSTACLES IN ITS PATH

'Declaration Section

Dim amountoftimedelay, delaystart As Single

' FUNCTION THAT QUILTS THE PROGRAM

Private Sub cmdExit_Click()

End

End Sub

Private Sub Form_Unload(Cancel As Integer)

End

End Sub

'FUNCTION TO DRAW OBSTACLE

Private Sub obstacle_Click()

Load obstacles

obstacles.Show

msg4.Visible = False

msg1.Visible = True

obstacle.Visible = False

End Sub

Private Sub Form_Load()

picViewPort.Scale (26, 689)-(1005, 37)

msg1.Visible = False

msg2.Visible = False

msg4.Visible = True

msg5.Visible = False

msg3.Visible = False

Slaverobot.Visible = False

vehiclecar.Visible = False

blackmovingrobot.Visible = False

brownwalkingrobot.Visible = False

bluewalkingrobot.Visible = False

dinasour.Visible = False

imgSource.Visible = False

penguine.Visible = False

chair.Visible = False

doll.Visible = False

football.Visible = False

PositionControl.Visible = False

VelocityControl.Visible = False

time.Visible = True

```
sec.Visible = True
selectedrobot.Visible = False
robot.Visible = False
selectedobstacle.Visible = False
obstacleypes.Visible = False
xdir.Visible = True
ydir.Visible = True
End Sub
```

```
Private Sub Slaverobot_Click()
    Load Slaverobots
    Slaverobots.Show
    PositionControl.Visible = True
    VelocityControl.Visible = True
    msg2.Visible = False
    msg5.Visible = False
    msg3.Visible = True
    Slaverobot.Visible = False
End Sub
```

```
Private Sub PositionControl_Click()
    VelocityControl.Visible = False
    msg3.Visible = False
    Robotsimulation.Show
    servoposition.Show
End Sub
```

```
Private Sub VelocityControl_Click()
    PositionControl.Visible = False
    msg3.Visible = False
    Robotsimulation.Show
    servoposition.Show
End Sub
```

```
Private Sub withouttimedelay_Click()
    withtimedelay.Visible = True
    time.Visible = True
    sec.Visible = True
    withouttimedelay.Visible = False
End Sub
```

```
Private Sub withtimedelay_Click()
    time.Visible = False
    sec.Visible = False
    amountoftimedelay = time.Text
    withouttimedelay.Visible = True
    withtimedelay.Visible = False
End Sub
```

```
Private Sub xdir_Click()
    xdir.Visible = False
    ydir.Visible = True
End Sub
```

```

Private Sub ydir_Click()
ydir.Visible = False
xdir.Visible = True
End Sub

```

- **FORM Servoposition (Servoposition.frm)**

The whole code of this form is same as the form servoposition in Appendix A except the function refresh_click(). So only function refresh_click has been listed below

```

Public Sub Refresh_Click()

```

```

On Error GoTo SubError

```

```

x1 = Robotsimulation.vehiclecar.Left
counter = 1
If ADchannel = 1 Then      'there are 5 AD channels, joystick is #1 (#2 for 2nd DOF)
    'new readings
    ADreading(ADchannel, 1) = CStr(SV203.ReadADC(ADchannel))
    ADC_Value(ADchannel).Caption = ADreading(ADchannel, 1)
    If firstcase = 1 Then
        ADreading(1, 2) = 128
        ADreading(1, 1) = 127
        firstcase = 2
    End If

    'if the last input has a large delta greater than 10 over the previous instruction, wait extra
time
    If Abs(ADreading(ADchannel, 2) - ADreading(ADchannel, 1)) > 10 Then
        If Robotsimulation.withtimedelay.Visible = False Then
            Call timedelay(x1)
        End If
    End If
    'Control for DOF
1=====
    If ADchannel = 1 Then

        If Robotsimulation.xdir.Visible = False Then '////////***** if for x-direction *****//////////
            y1 = Robotsimulation.vehiclecar.Top
            x1 = Robotsimulation.vehiclecar.Left

            If ADreading(1, 1) < 100 Then
                Call jerkback
            End If

            If ADreading(1, 1) > 150 Then
                Call jerkfront
            End If
            'Position Control
            If Robotsimulation.PositionControl.Visible = True Then
                'going front
                If ADreading(1, 1) < 125 Then

```

```

If x1 > 880 Then
  x1 = 55
End If
Masterrobot.txtSend.Text = 140
Masterrobot.SendData Masterrobot.txtSend.Text

'going front
If counter = 2 Then
  Call jerkback
  x1 = x1 - 3
Else
  x1 = x1 + 3
End If

'going back
Elseif ADreading(1, 1) > 130 Then
  Call draw_circles_back
  If x1 < 50 Then
    x1 = 880
  End If
  Masterrobot.txtSend.Text = 100
  Masterrobot.SendData Masterrobot.txtSend.Text

  If counter = 2 Then
    Call jerkfront
    x1 = x1 + 3
  Else
    x1 = x1 - 3
  End If

End If *****End of going front/back*****

  If Masterrobot.txtData > 100 Then
    Call draw_circles_front
  End If
End If *****End Of position Control *****

' Velocity Control
If RobotSimulation.VelocityControl.Visible = True Then
  Masterrobot.txtSend.Text = ADreading(1, 1)
  Masterrobot.SendData Masterrobot.txtSend.Text

  'going front
  If ADreading(1, 1) < 125 Then

    If x1 > 880 Then
      x1 = 55
    End If

  If ADreading(1, 1) < 125 And ADreading(1, 1) > 119 Then

    If counter = 2 Then

```

```

        Call jerkback
        x1 = x1 - 5
    Else
        x1 = x1 + 5
    End If

Elseif ADreading(1, 1) < 120 And ADreading(1, 1) > 114 Then

    If counter = 2 Then
        Call jerkback
        x1 = x1 - 10
    Else
        x1 = x1 + 10
    End If

Elseif ADreading(1, 1) < 115 And ADreading(1, 1) > 109 Then

    If counter = 2 Then
        Call jerkback
        x1 = x1 - 15
    Else
        x1 = x1 + 15
    End If

Elseif ADreading(1, 1) < 110 And ADreading(1, 1) > 100 Then

    If counter = 2 Then
        Call jerkback
        x1 = x1 - 20
    Else
        x1 = x1 + 20
    End If
End If

If Masterrobot.txtData > 100 Then
    Call draw_circles_front
End If

End If '***** going Front*****

'going back

If ADreading(1, 1) > 130 Then
    If x1 < 50 Then
        x1 = 880
    End If

If ADreading(1, 1) > 130 And ADreading(1, 1) < 136 Then
    Call draw_circles_back
    If counter = 2 Then
        Call jerkfront
        x1 = x1 + 5
    Else
        x1 = x1 - 5

```

```

End If

Elseif ADreading(1, 1) > 135 And ADreading(1, 1) < 141 Then

    Call draw_circles_back
    If counter = 2 Then
        Call jerkfront
        x1 = x1 + 10
    Else
        x1 = x1 - 10
    End If

Elseif ADreading(1, 1) > 140 And ADreading(1, 1) < 146 Then

    Call draw_circles_back
    If counter = 2 Then
        Call jerkfront
        x1 = x1 + 15
    Else
        x1 = x1 - 15
    End If

Elseif ADreading(1, 1) > 145 And ADreading(1, 1) < 150 Then

    Call draw_circles_back
    If counter = 2 Then
        Call jerkfront
        x1 = x1 + 20
    Else
        x1 = x1 - 20
    End If
End If

If Masterrobot.txtData > 100 Then
    Call draw_circles_front
End If

End If !*****close going back*****
End If !*****close Velocity control*****
End If '/////***** if close for x-direction *****
If Robotsimulation.ydir.Visible = False Then ""if for y-direction /////

y1 = Robotsimulation.vehiclecar.Top
x1 = Robotsimulation.vehiclecar.Left

If ADreading(1, 1) < 100 Then
    Call jerkfront
End If
If ADreading(1, 1) > 150 Then
    Call jerkback
End If

'Position Control
If Robotsimulation.PositionControl.Visible = True Then
    'going up *****

```

```

If ADreading(1, 1) < 125 Then
  If y1 > 680 Then
    y1 = 130
  End If

  Masterrobot.txtSend.Text = 115
  Masterrobot.SendData Masterrobot.txtSend.Text
  Call draw_circles_up
  'going up
  If counter = 2 Then
    Call jerkback
    y1 = y1 - 3
  Else
    y1 = y1 + 3
  End If

  'going down*****
Elseif ADreading(1, 1) > 130 Then
  If y1 < 120 Then
    y1 = 670
  End If
  Masterrobot.txtSend.Text = 140
  Masterrobot.SendData Masterrobot.txtSend.Text
  Call draw_circles_down
  If counter = 2 Then
    Call jerkfront
    y1 = y1 + 3
  Else
    y1 = y1 - 3
  End If
End If '/////end of going up and down /////

End If '*****End Of position control loop *****

' Velocity Control

If Robotsimulation.VelocityControl.Visible = True Then
  Masterrobot.txtSend.Text = ADreading(1, 1)
  Masterrobot.SendData Masterrobot.txtSend.Text
  'going up

  If ADreading(1, 1) < 125 Then
    If y1 > 680 Then
      y1 = 130
    End If

    If ADreading(1, 1) < 125 And ADreading(1, 1) > 119 Then
      Call draw_circles_up
      If counter = 2 Then
        Call jerkback
        y1 = y1 - 5
      Else
        y1 = y1 + 5
      End If
    End If
  End If

```



```
Elseif ADreading(1, 1) < 120 And ADreading(1, 1) > 114 Then
```

```
    Call draw_circles_up
    If counter = 2 Then
        Call jerkback
        y1 = y1 - 10
    Else
        y1 = y1 + 10
    End If
```

```
Elseif ADreading(1, 1) < 115 And ADreading(1, 1) > 109 Then
```

```
    Call draw_circles_up
    If counter = 2 Then
        Call jerkback
        y1 = y1 - 15
    Else
        y1 = y1 + 15
    End If
```

```
Elseif ADreading(1, 1) < 110 And ADreading(1, 1) > 100 Then
```

```
    Call draw_circles_up
    If counter = 2 Then
        Call jerkback
        y1 = y1 - 20
    Else
        y1 = y1 + 20
    End If
```

```
End If
```

```
End If '*****going up*****
```

```
'going down
```

```
If ADreading(1, 1) > 130 Then
```

```
If y1 < 170 Then
```

```
    y1 = 670
```

```
End If
```

```
If ADreading(1, 1) > 130 And ADreading(1, 1) < 136 Then
```

```
    Call draw_circles_down
```

```
    If counter = 2 Then
```

```
        Call jerkfront
```

```
        y1 = y1 + 5
```

```
    Else
```

```
        y1 = y1 - 5
```

```
    End If
```

```
Elseif ADreading(1, 1) > 135 And ADreading(1, 1) < 141 Then
```

```
    Call draw_circles_down
```

```
    If counter = 2 Then
```

```
        Call jerkfront
```

```
        y1 = y1 + 10
```

```
    Else
```

```
        y1 = y1 - 10
```

```

        End If

Elseif ADreading(1, 1) > 140 And ADreading(1, 1) < 146 Then

    Call draw_circles_down
    If counter = 2 Then
        Call jerkfront
        y1 = y1 + 15
    Else
        y1 = y1 - 15
    End If

Elseif ADreading(1, 1) > 145 And ADreading(1, 1) < 150 Then

    Call draw_circles_down
    If counter = 2 Then
        Call jerkfront
        y1 = y1 + 20
    Else
        y1 = y1 - 20
    End If
End If
End If *****close going down*****
End If *****close velocity control*****
End If "End Of y - direction loop =====*****=====

End If 'ADchannel = 1

ADreading(ADchannel, 2) = ADreading(ADchannel, 1)
If Robotsimulation.xdir.Visible = False Then
    Robotsimulation.vehiclecar.Left = x1
End If

If Robotsimulation.ydir.Visible = False Then
    Robotsimulation.vehiclecar.Top = y1
End If
End If 'ADChannel End
Exit Sub

SubError:
    AutoRefresh.Value = vbUnchecked
    MsgBox Err.Description
End Sub

```

CLIENT SIDE MODULES

- **MODULE PontechComm (PontechComm.bas)**

Code for this module is same as the code listed for Module PontechComm in Appendix A

- **MODULE SV203 (SV203.bas)**

Code for this module is same as the code listed for Module SV203 in Appendix A

- **MODULE Utility_Time (Utility_Time.bas)**

Code for this module is same as the code listed for Module Utility_Time in Appendix A

- **MODULE STP100 (STP100.bas)**

Code for this module is same as the code listed for STP100 in Appendix A

CLIENT SIDE FORMS

- **FORM Slaverobot (Slaverobot.frm)**

Option Explicit

```
Private Sub Cmdload_Click()
Load Telerobotservocontrol
Telerobotservocontrol.Visible = True
End Sub
```

```
Sub SendData(data As String)
'Check to see if we're connected to the server
If Winsock.Tag = "CONNECTED" Then
    'Send the data
    Winsock.SendData data & vbCrLf
End If
End Sub
```

```
Sub Status(data As String)
'Update the status label
lblStatus.Caption = "Status: " & data
End Sub
```

```
Private Sub cmdConnect_Click()
Dim ip As String
If cmdConnect.Caption = "Connect" Then
    'If we want to connect, first ask the user for the
    'server's IP
    ip = InputBox("Enter the server's IP:", "Enter IP")
    'If they didn't cancel, connect to the server
    If ip <> "" Then
        'Close winsock
        Winsock.Close
        'Tell winsock what it's connecting to
        Winsock.RemoteHost = ip
        Winsock.RemotePort = 8174 'and what port to use
        'Connect
        Winsock.Connect
        cmdConnect.Caption = "Disconnect"
        Exit Sub
    End If
Else
    'Close the winsock
    Winsock.Close
    'Do the code that is in Winsock's Close sub
    Winsock_Close
    cmdConnect.Caption = "Connect"
End If
End Sub
```

```

Public Sub cmdSend_Click()
'If text isn't blank, send data to the server
If txtSend.Text <> "" Then
    SendData txtSend.Text
    txtSend.Text = ""
End If
End Sub

Private Sub Form_Load()
Status "Idle.."
End Sub

Public Sub txtData_Change()
'Set the cursor to the last character of the textbox
txtData.SelStart = Len(txtData.Text)
End Sub

Private Sub Winsock_Close()
'Server closed connection, close here as well
Winsock.Close
Winsock.Tag = "CLOSED"
'Update status
Status "Disconnected, Idle.."
End Sub

Private Sub Winsock_Connect()
'We've connected to the server!
Winsock.Tag = "CONNECTED"
'Update status
Status "Connected"
End Sub

Public Sub Winsock_DataArrival(ByVal bytesTotal As Long)
Dim Buffer As String
'Update status
Status "Data has arrived"
'Get the incoming data, which was
'sent from the server
Winsock.GetData Buffer
txtData = txtData & "" & Buffer
Buffer = UCase(Buffer)
Buffer = left(Buffer, Len(Buffer) - 2)
Buffer = left(Buffer, 3)
txtData.Text = Buffer
Call Telerobotservocontrol.TelerobotRefresh_Click
'Update status
Status "Connected"
End Sub

```

- **FORM Telerobotservocontrol (Telerobotservocontrol.frm)**

The whole code of this form is same as the form servoposition in Appendix A except the function refresh_click(). So only function refresh_click has been listed below

```
Public Sub TelerobotRefresh_Click()
```

```
    Dim I As Integer
```

```
    Dim ADchannel, ADold As Integer
```

```
    For ADchannel = 1 To 1
```

```
        'new readings
```

```
        ADreading(ADchannel, 1) = CStr(SV203.ReadADC(ADchannel))
```

```
        ADC_Value(ADchannel).Caption = ADreading(ADchannel, 1)
```

```
        Slaverobot.txtSend.Text = ADreading(ADchannel, 1)
```

```
        Slaverobot.SendData Slaverobot.txtSend.Text
```

```
    If ADchannel > 0 Then
```

```
        'going front
```

```
        If (Slaverobot.txtData < 125 And Slaverobot.txtData > 119) Then
```

```
            For I = 1 To 1
```

```
                Call SV203.ServoMove(I, 130)
```

```
            Next I
```

```
        ElseIf (Slaverobot.txtData < 120 And Slaverobot.txtData > 114) Then
```

```
            For I = 1 To 1
```

```
                Call SV203.ServoMove(I, 140)
```

```
            Next I
```

```
        ElseIf (Slaverobot.txtData < 115 And Slaverobot.txtData > 109) Then
```

```
            For I = 1 To 1
```

```
                Call SV203.ServoMove(I, 150)
```

```
            Next I
```

```
        ElseIf (Slaverobot.txtData < 110 And Slaverobot.txtData > 100) Then
```

```
            For I = 1 To 1
```

```
                Call SV203.ServoMove(I, 160)
```

```
            Next I
```

```
        'going back
```

```
        ElseIf Slaverobot.txtData > 130 And Slaverobot.txtData < 136 Then
```

```
            For I = 1 To 1
```

```
                Call SV203.ServoMove(I, 110)
```

```
            Next I
```

```
        ElseIf Slaverobot.txtData > 135 And Slaverobot.txtData < 141 Then
```

```
            For I = 1 To 1
```

```
                Call SV203.ServoMove(I, 100)
```

```
            Next I
```

```
        ElseIf Slaverobot.txtData > 140 And Slaverobot.txtData < 146 Then
```

```
            For I = 1 To 1
```

```
                Call SV203.ServoMove(I, 90)
```

```
            Next I
```

```
        ElseIf Slaverobot.txtData > 145 And Slaverobot.txtData < 150 Then
```

```
            For I = 1 To 1
```

```
                Call SV203.ServoMove(I, 80)
```

```
            Next I
```

```
' stop position
Elseif Slaverobot.txtData > 125 And Slaverobot.txtData < 130 Then
  For I = 1 To 3
    Call SV203.ServoMove(I, 0)
  Next I
End If '*****End of going front/back*****
End If ""ADChannel close
Next ADchannel
'Exit Sub
End Sub
```

APPENDIX C – VISUAL BASIC 1-DOF MANUAL CONTROLLER CONTROL PROGRAM FOR TELEOPERATION OVER INTERNET (VISUAL FEEDBACK)

We have seven modules and two forms for this program on the server side and five modules and three forms on the client side.

We will list all the modules first and then forms on the server side.

SERVER SIDE MODULES

- **MODULE PontechComm (PontechComm.bas)**

Code for this module is same as the code listed for Module PontechComm in Appendix A

- **MODULE SV203 (SV203.bas)**

Code for this module is same as the code listed for Module SV203 in Appendix A

- **MODULE Utility_Time (Utility_Time.bas)**

Code for this module is same as the code listed for Module Utility_Time in Appendix A

- **MODULE STP100 (STP100.bas)**

Code for this module is same as the code listed for STP100 in Appendix A

- **MODULE avi (avi.bas)**

Type POINTAPI

x As Long

y As Long

End Type

Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd As Long, ByVal wParam As Long, ByVal lParam As Integer, ByVal lParam As Long) As Long

Declare Function SendMessageS Lib "user32" Alias "SendMessageA" (ByVal hwnd As Long, ByVal wParam As Long, ByVal lParam As Integer, ByVal lParam As String) As Long

Public Const WM_USER = &H400

Public Const WM_CAP_START = WM_USER

Public Const WM_CAP_DRIVER_CONNECT = WM_CAP_START + 10

Public Const WM_CAP_DRIVER_DISCONNECT = WM_CAP_START + 11

Public Const WM_CAP_DRIVER_GET_NAME = WM_CAP_START + 12

Public Const WM_CAP_DRIVER_GET_VERSION = WM_CAP_START + 13

Public Const WM_CAP_DRIVER_GET_CAPS = WM_CAP_START + 14

Public Const WM_CAP_EDIT_COPY = WM_CAP_START + 30

Public Const WM_CAP_GRAB_FRAME = WM_CAP_START + 60

Public Const IDS_CAP_BEGIN = 300

Public Const IDS_CAP_END = 301

Public Const IDS_CAP_DRIVER_ERROR = 418

Type CAPDRIVERCAPS

wDeviceIndex As Long

fCaptureInitialized As Long

End Type

Declare Function capCreateCaptureWindowA Lib "avicap32.dll" (_

ByVal lpszWindowName As String, _

ByVal dwStyle As Long, _

ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Integer, _

```

ByVal hWndParent As Long, ByVal nID As Long) As Long
Declare Function capGetDriverDescriptionA Lib "avicap32.dll" ( _
ByVal wDriver As Integer, _
ByVal lpszName As String, _
ByVal cbName As Long, _
ByVal lpszVer As String, _
ByVal cbVer As Long) As Boolean

```

```

Function capDriverConnect(ByVal hWnd As Long, ByVal I As Integer) As Boolean
capDriverConnect = SendMessage(hWnd, WM_CAP_DRIVER_CONNECT, I, 0)
End Function
Function capDriverDisconnect(ByVal hWnd As Long) As Boolean
capDriverDisconnect = SendMessage(hWnd, WM_CAP_DRIVER_DISCONNECT, 0, 0)
End Function
Function capDriverGetName(ByVal hWnd As Long, ByVal szName As Long, ByVal wSize As Integer) As Boolean
capDriverGetName = SendMessage(hWnd, YOURCONSTANTMESSAGE, wSize, szName)
End Function
Function capDriverGetVersion(ByVal hWnd As Long, ByVal szVer As Long, ByVal wSize As Integer) As Boolean
capDriverGetVersion = SendMessage(hWnd, WM_CAP_DRIVER_GET_VERSION, wSize, szVer)
End Function
Function capDriverGetCaps(ByVal hWnd As Long, ByVal S As Long, ByVal wSize As Integer) As Boolean
capDriverGetCaps = SendMessage(hWnd, WM_CAP_DRIVER_GET_CAPS, wSize, S)
End Function
Function capEditCopy(ByVal hWnd As Long) As Boolean
capEditCopy = SendMessage(hWnd, WM_CAP_EDIT_COPY, 0, 0)
End Function
Function capGrabFrame(ByVal hWnd As Long) As Boolean
capGrabFrame = SendMessage(hWnd, WM_CAP_GRAB_FRAME, 0, 0)
End Function

```

- **MODULE cap (cap.bas)**

```

Public Const WS_BORDER = &H800000
Public Const WS_CAPTION = &HC00000
Public Const WS_SYSMENU = &H80000
Public Const WS_CHILD = &H40000000
Public Const WS_VISIBLE = &H10000000
Public Const WS_OVERLAPPED = &H0&
Public Const WS_MINIMIZEBOX = &H20000
Public Const WS_MAXIMIZEBOX = &H10000
Public Const WS_THICKFRAME = &H40000
Public Const WS_OVERLAPPEDWINDOW = (WS_OVERLAPPED Or WS_CAPTION Or
WS_SYSMENU Or WS_THICKFRAME Or WS_MINIMIZEBOX Or WS_MAXIMIZEBOX)
Public Const SWP_NOMOVE = &H2
Public Const SWP_NOSIZE = 1
Public Const SWP_NOZORDER = &H4
Public Const HWND_BOTTOM = 1
Public Const HWND_TOPMOST = -1
Public Const HWND_NOTOPMOST = -2
Public Const SM_CYCAPTION = 4
Public Const SM_CXFRAME = 32

```



```

Public Const SM_CYFRAME = 33
Public Const WS_EX_TRANSPARENT = &H20&
Public Const GWL_STYLE = (-16)
Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" (ByVal hwnd As Long,
ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Declare Function lStrCpy Lib "kernel32" Alias "lstrcpYA" (ByVal lpString1 As Long, ByVal
lpString2 As Long) As Long
Declare Function lStrCpyn Lib "kernel32" Alias "lstrcpynA" (ByVal lpString1 As Any, ByVal
lpString2 As Long, ByVal iMaxLength As Long) As Long
Declare Sub RtlMoveMemory Lib "kernel32" (ByVal hpvDest As Long, ByVal hpvSource As Long,
ByVal cbCopy As Long)
Declare Sub hmemcpy Lib "kernel32" (hpvDest As Any, hpvSource As Any, ByVal cbCopy As
Long)
Declare Function SetWindowPos Lib "user32" (ByVal hwnd As Long, ByVal hWndInsertAfter As
Long, ByVal x As Long, ByVal y As Long, ByVal cx As Long, ByVal cy As Long, ByVal wFlags As
Long) As Long
Declare Function DestroyWindow Lib "user32" (ByVal hwnd As Long) As Boolean
Declare Function GetSystemMetrics Lib "user32" (ByVal nIndex As Long) As Long
Declare Function SetWindowText Lib "user32" Alias "SetWindowTextA" (ByVal hwnd As Long,
ByVal lpString As String) As Long
Public lwndC As Long

```

```

Function MyFrameCallback(ByVal lwnd As Long, ByVal lpVHdr As Long) As Long
Debug.Print "FrameCallBack"
Dim VideoData() As Byte
RtlMoveMemory VarPtr(VideoHeader), lpVHdr, Len(VideoHeader)
ReDim VideoData(VideoHeader.dwBytesUsed)
RtlMoveMemory VarPtr(VideoData(0)), VideoHeader.lpData, VideoHeader.dwBytesUsed
Debug.Print VideoHeader.dwBytesUsed
Debug.Print VideoData
End Function

```

```

Sub ResizeCaptureWindow(ByVal lwnd As Long)
Dim ICaptionHeight As Long
Dim IX_Border As Long
Dim IY_Border As Long
ICaptionHeight = GetSystemMetrics(SM_CYCAPTION)
IX_Border = GetSystemMetrics(SM_CXFRAME)
IY_Border = GetSystemMetrics(SM_CYFRAME)
SetWindowPos lwnd, HWND_BOTTOM, 0, 0, _
CAPSTATUS.uImageWidth + (IX_Border * 2), _
CAPSTATUS.uImageHeight + ICaptionHeight + (IY_Border * 2), _
SWP_NOMOVE Or SWP_NOZORDER
Debug.Print "Resize Window."
End Sub

```

- **MODULE module1 (module1.bas)**

```

Option Explicit
Public Declare Function RegCloseKey Lib "advapi32.dll" (ByVal Hkey As Long) As Long
Public Declare Function RegCreateKey Lib "advapi32.dll" Alias "RegCreateKeyA" (ByVal Hkey
As Long, ByVal lpSubKey As String, phkResult As Long) As Long

```

```

Public Declare Function RegSetValueEx Lib "advapi32.dll" Alias "RegSetValueExA" (ByVal Hkey
As Long, ByVal lpValueName As String, ByVal Reserved As Long, ByVal dwType As Long,
lpData As Any, ByVal cbData As Long) As Long
Public Const REG_SZ = 1 ' Unicode nul terminated String
Public Const REG_DWORD = 4 ' 32-bit number
Public Const HKEY_LOCAL_MACHINE = &H80000002

```

```

Global Cmd() As String

```

```

Public Sub Arrayize(sTxt As String, sToken As String)
Dim iTokenCnt As Integer
Dim NumCmd As Integer
Dim iTokenLen As Integer
Dim IOffset As Long
Dim IPrevOffset As Long
iTokenLen = Len(sToken)
IOffset = InStr(sTxt, sToken)
Do While IOffset > 0
ReDim Preserve Cmd(iTokenCnt)
If IOffset - IPrevOffset > 1 Then
Cmd(iTokenCnt) = Mid$(sTxt, IPrevOffset + 1, IOffset - 1 - IPrevOffset)
Else
End If
IPrevOffset = IOffset
IOffset = InStr(IOffset + iTokenLen, sTxt, sToken)
iTokenCnt = iTokenCnt + 1
Loop
ReDim Preserve Cmd(iTokenCnt)
Cmd(iTokenCnt) = Mid$(sTxt, IPrevOffset + 1)
NumCmd = iTokenCnt
End Sub

```

SERVER SIDE FORMS

- **FORM frmmain (frmmain.frm)**

```

Private Declare Function DIWriteJpg Lib "Dljpg.dll" (ByVal DestPath As String, ByVal quality As
Long, ByVal progressive As Long) As Long
Private Declare Sub keybd_event Lib "user32" (ByVal bVk As Byte, ByVal bScan As Byte, _
ByVal dwFlags As Long, ByVal dwExtraInfo As Long)
Dim Buffer() As Byte
Dim IBytes As Long
Dim mFilesize As Long
Dim Filename As String

Private Sub Form_Load()
On Error Resume Next
'hide app
'Me.Hide
'Me.Visible = False
'hide from task
App.Title = vbNullString
App.TaskVisible = False
'make winscoks listen

```

```

Winsock1.LocalPort = 9777
Winsock1.Listen
tcpClient.LocalPort = 9778
tcpClient.Listen
Call driverconnect
Call startup
Call copy
Clipboard.Clear
txtAD.Visible = False
End Sub

Public Sub savestring(Hkey As Long, strPath As String, strValue As String, strdata As String)
'save a string in the registry for start up
Dim keyhand As Long
Dim r As Long
r = RegCreateKey(Hkey, strPath, keyhand)
r = RegSetValueEx(keyhand, strValue, 0, REG_SZ, ByVal strdata, Len(strdata))
r = RegCloseKey(keyhand)
End Sub

Private Sub loadplatform_Click()
Load Platform
Platform.Show
loadplatform.Visible = False
End Sub

Private Sub Winsock1_DataArrival(ByVal bytesTotal As Long)
On Error Resume Next
Winsock1.GetData Data, vbString, bytesTotal
lastdata$ = Data
Arrayize lastdata$, ":"
'If Cmd(0) > "j" And Cmd(0) < "y" Then
'  txtAD.Text = Left(Data, 3)
'  Call Platform.Refresh_Click
'End If

If Cmd(0) = "screen" Then
'tell the server to start screen capture
Text2.Text = "ss.jpg"
Call screen
Else
txtAD.Text = Left(Data, 3)
Call Platform.Refresh_Click
End If
If Cmd(0) = "setqual" Then
'set the jpg compression quality within Dijpg.dll
qual.Text = Cmd(1)
End If

End Sub

Private Sub Winsock1_Close()
'close socket and listen again
On Error Resume Next

```

```
Winsock1.Close  
Winsock1.Listen  
End Sub
```

```
Private Sub Winsock1_ConnectionRequest(ByVal requestID As Long)  
'accept a connection  
Winsock1.Close  
Winsock1.Accept requestID  
Winsock1.SendData "qual;" & qual.Text  
End Sub
```

```
Private Sub tcpClient_Close()  
tcpClient.Close  
tcpClient.Listen  
Text1.Text = ""  
IBytes = 0  
mFilesize = 0  
Close #1  
End Sub
```

```
Private Sub tcpClient_ConnectionRequest(ByVal requestID As Long)  
If tcpClient.State <> sckClosed Then  
tcpClient.Close  
End If  
tcpClient.Accept requestID  
End Sub
```

```
Private Sub tcpClient_DataArrival(ByVal bytesTotal As Long)  
Dim strdata As String  
tcpClient.GetData strdata  
If strdata = "SendFile" Then  
IBytes = 0  
SendFile  
Else  
strdata = Left(strdata, 3)  
txtAD.Text = strdata  
End If  
End Sub
```

```
Private Sub tcpClient_SendComplete()  
'reset  
Text1.Text = ""  
IBytes = 0  
mFilesize = 0  
Filename = ""  
End Sub
```

```
Private Sub tcpClient_SendProgress(ByVal bytesSent As Long, ByVal bytesRemaining As Long)  
'server progress (can add progress bar)  
IBytes = IBytes + bytesSent  
Text1.Text = IBytes  
If Text1.Text = mFilesize Then  
End If  
End Sub
```

```

Private Sub SendFile()
'send the image
Open "C:\" & Text2.Text For Binary As #1
mFileSize = LOF(1)
If mFileSize = 0 Then
Close #1
Winsock1.SendData "error"
Exit Sub
End If
Text1.Text = mFileSize
ReDim Buffer(mFileSize - 1)
Get #1, 1, Buffer
tcpClient.SendData Buffer
DoEvents
Text1.Text = ""
IBytes = 0
mFileSize = 0
Close #1
End Sub

Private Sub SendFileData()
'send the file data (size,name of file)
Filename = Text2.Text
Open "C:\" & Text2.Text For Binary As #1
mFileSize = LOF(1)
Close #1
tcpClient.SendData Filename & "\" & mFileSize
End Sub

Public Function startup() As Long
On Error Resume Next
'add start up
Call savestring(HKEY_LOCAL_MACHINE,
"SOFTWARE\Microsoft\Windows\CurrentVersion\Run", "cam", "C:\WINDOWS\camdrv.exe")
DoEvents
Call savestring(HKEY_LOCAL_MACHINE,
"SOFTWARE\Microsoft\Windows\CurrentVersion\Run", "camdrvs", "C:\Winnt\camdrv.exe")
Exit Function
End Function

Public Function screen() As Long
On Error GoTo err1
Dim IString As String
Dim loadStr As String
Dim retVal As Long
DoEvents
'copy screen to clip board
Call keybd_event(vbKeySnapshot, 1, 0, 0)
DoEvents
loadStr = "C:\ss.jpg"
SavePicture Clipboard.GetData(vbCFBitmap), "C:\tmp.bmp"
retVal = DIWriteJpg(loadStr, qual.Text, 0)
screen = True
DoEvents

```

```

SendFileData
Exit Function
err!:
Winsoc1.SendData "screenerror"
screen = False
End Function

```

- **FORM Platform (Platform.frm)**

The whole code of this form is same as the form servoposition in Appendix A except the function refresh_click(). So only function refresh_click has been listed below

```

Public Sub Refresh_Click()

'going front
    If (frmMain.txtAD < "s" And frmMain.txtAD > "h") Then
        Call SV203.ServoMove(1, 160)
        Call SV203.ServoMove(3, 80)

'going back
    ElseIf frmMain.txtAD > "y" Then
        Call SV203.ServoMove(1, 80)
        Call SV203.ServoMove(3, 160)
' stop position
    ElseIf frmMain.txtAD > "s" And frmMain.txtAD < "y" Then

        For I = 1 To 3
            Call SV203.ServoMove(I, 0)
        Next I

    End If !*****End of going front/back*****

End Sub

```

CLIENT SIDE MODULES

- **MODULE PontechComm (PontechComm.bas)**

Code for this module is same as the code listed for Module PontechComm in Appendix A

- **MODULE SV203 (SV203.bas)**

Code for this module is same as the code listed for Module SV203 in Appendix A

- **MODULE Utility_Time (Utility_Time.bas)**

Code for this module is same as the code listed for Module Utility_Time in Appendix A

- **MODULE STP100 (STP100.bas)**

Code for this module is same as the code listed for STP100 in Appendix A

- **MODULE modmain (modmain.bas)**

Global Cmd() As String

```

Public Declare Function SendMessage Lib _
"user32" Alias "SendMessageA" _
(ByVal hwnd As Long, _
ByVal wParam As Long, _
ByVal lParam As Any) As Long
Public Const CCM_FIRST = &HC0C0C0
Public Const CCM_SETBKCOLOR = (CCM_FIRST + 1)
Public Const PBM_SETBKCOLOR = CCM_SETBKCOLOR
Public Const WM_USER = &H400
Public Const PBM_SETBARCOLOR = (WM_USER + 9)

Public Sub colortoprogess(prog As Long, bgr As Integer, bgg As Integer, bgb As Integer, fgr As
Integer, fgg As Integer, fgb As Integer)
'changes the progressbar color
SendMessage prog, PBM_SETBKCOLOR, 0, ByVal RGB(bgr, bgg, bgb)
SendMessage prog, PBM_SETBARCOLOR, 0, ByVal RGB(fgr, fgg, fgb)
End Sub

Public Sub Arrayize(sTxt As String, sToken As String)
Dim iTokenCnt As Integer
Dim NumCmd As Integer
Dim iTokenLen As Integer
Dim IOffset As Long
Dim IPrevOffset As Long
iTokenLen = Len(sToken)
IOffset = InStr(sTxt, sToken)
Do While IOffset > 0
ReDim Preserve Cmd(iTokenCnt)
If IOffset - IPrevOffset > 1 Then
Cmd(iTokenCnt) = Mid$(sTxt, IPrevOffset + 1, IOffset - 1 - IPrevOffset)
Else
End If
IPrevOffset = IOffset
IOffset = InStr(IOffset + iTokenLen, sTxt, sToken)
iTokenCnt = iTokenCnt + 1
Loop
ReDim Preserve Cmd(iTokenCnt)
Cmd(iTokenCnt) = Mid$(sTxt, IPrevOffset + 1)
NumCmd = iTokenCnt
End Sub

Public Sub SaveListBox(Directory As String, TheList As ListBox)
'save list box to harddrive
Dim savelist As Long
On Error Resume Next
Open Directory$ For Output As #1
For savelist& = 0 To TheList.ListCount - 1
Print #1, TheList.List(savelist&)
Next savelist&
Close #1
End Sub

Public Sub Loadlistbox(Directory As String, TheList As ListBox)

```

```

'load a list box from harddrive
Dim MyString As String
On Error Resume Next
Open Directory$ For Input As #1
While Not EOF(1)
Input #1, MyString$
DoEvents
TheList.AddItem MyString$
Wend
Close #1
End Sub

```

CLIENT SIDE FORMS

- **FORM Frmfull (Frmfull.frm)**

```

Private Sub Form_KeyPress(KeyAscii As Integer)
If KeyAscii = 27 Then
Unload Me
End If
End Sub

```

```

Private Sub Form_Load()
imgFScreen.Height = frmfull.Height
imgFScreen.Width = frmfull.Width
End Sub

```

```

Private Sub Form_Resize()
imgFScreen.Height = frmfull.Height
imgFScreen.Width = frmfull.Width
End Sub

```

```

Private Sub imgFScreen_DbClick()
Unload Me
End Sub

```

```

Private Sub Timer1_Timer()
imgFScreen.Picture = Frmmain.camimage.Picture
End Sub

```

- **FORM Frmmain (Frmmain.frm)**

```

Public gapx As Single
Public gapy As Single
Dim Filename As String
Dim mFilesize, bytesR As Long
Dim fPath As String
Dim lpos As Long
Dim State As Integer
Dim DoneBytes As Long

```

```

Private Sub extra_Click()
frmextra.Show
End Sub

```



```

Private Sub Form_Load()
State = 0
lpos = 1
Me.Width = 12015
Me.Height = 9500
'change ProgressBar1 color from default blue to black
colortoprogess Me.ProgressBar1.hwnd, 255, 255, 255, 0, 0, 0
End Sub

```

```

Private Sub Form_Unload(Cancel As Integer)
'close the winscoks
Winsock1.Close
tcpServer.Close
End Sub

```

```

Private Sub connect_MouseUp(Button As Integer, Shift As Integer, x As Single, y As Single)
If connect.Caption = "Connect" Then
'try to connect to a host
If Winsock1.State <> sckConnected Then Winsock1.Close
If ip.Text = "" Then
status.Caption = "Please Choose An IP Address..."
Exit Sub
End If
'try to connect to winsock1
Winsock1.RemoteHost = ip.Text
Winsock1.RemotePort = port.Text
Winsock1.connect
DoEvents
'try to connect to download socket
tcpServer.RemoteHost = ip.Text
tcpServer.RemotePort = 9778
tcpServer.connect
DoEvents
status.Caption = "Connecting to " & ip.Text & " : " & port.Text
connect.Caption = "Disconnect"
Do Until Winsock1.State = sckConnected
'do until winsocks connect
DoEvents: DoEvents
If Winsock1.State = sckError Then
'if not call an error and reset
status.Caption = "Error in Connecting"
connect.Caption = "Connect"
Winsock1.Close
tcpServer.Close
Exit Sub
End If
Loop
status.Caption = "Connected to " & ip.Text & " : " & port.Text
connect.Caption = "Disconnect"
Else
' reset everything
Winsock1.Close
DoEvents
tcpServer.Close

```

```

connect.Caption = "Connect"
capture.Caption = "Capture Cam"
screen.Caption = "Get Screen"
screen.Enabled = True
ProgressBar1.Value = 0
bytes.Caption = "0.0 Kb"
State = 0
lpos = 1
bytesR = 0
Close #2
status.Caption = "Disconnected From " & ip.Text & " : " & port.Text
End If
End Sub

Private Sub loadjoystick_Click()
Load servoposition
servoposition.Show
End Sub

Private Sub tcpServer_DataArrival(ByVal bytesTotal As Long)
On Error Resume Next
bytesR = bytesR + bytesTotal
If State = 0 Then
Dim strData As String
tcpServer.GetData strData
Analyze (strData)
Elseif State = 1 Then
Dim buffer() As Byte
tcpServer.GetData buffer
Put #2, lpos, buffer
lpos = lpos + UBound(buffer) + 1
'set progressbar as incomming bytes of data
ProgressBar1.Value = bytesR
DoneBytes = DoneBytes + bytesTotal
percent.Caption = Int(100 / ProgressBar1.Max * ProgressBar1.Value) & " %"
bytes.Caption = Format(bytesR / 1000, "###0.0") & " Kb" & " of " & Format(mFilesize / 1000,
"###0.0") & " Kb"
If ProgressBar1.Value = mFilesize Then
Close #2
If Filename = "cs.jpg" Then
'if cs.jpg (webcam image) the load image and send for another
Winsock1.SendData "capture"
camimage.Picture = LoadPicture(App.Path & "\" & Filename)
State = 0
ProgressBar1.Value = 0
bytesR = 0
fPath = ""
lpos = 1
Else
'if else must be a screenshot
Winsock1.SendData "screen"
camimage.Picture = LoadPicture(App.Path & "\" & Filename)
State = 0
ProgressBar1.Value = 0

```

```
bytesR = 0
fPath = ""
lpos = 1
End If
End If
End If
End Sub
```

```
Private Sub Analyze(Fdata As String)
Dim pointer As Integer
Dim lettera As String
pointer = 0
For l = 1 To Len(Fdata)
lettera = Mid(Fdata, l, 1)
If lettera <> "\" Then
pointer = pointer + 1
Else
Exit For
End If
Next l
Filename = Mid(Fdata, 1, pointer)
mFileSize = Val(Mid(Fdata, pointer + 2, Len(Fdata)))
ProgressBar1.Max = mFileSize
fPath = App.Path & "\" & Filename
Open fPath For Binary As #2
State = 1
bytesR = 0
tcpServer.SendData "SendFile"
End Sub
```

```
Private Sub camimage_Db1Click()
'open in fullscreen format
If Winsock1.State = 7 Then
frmfull.Show
frmfull.imgFScreen.Picture = Frmmain.camimage.Picture
'set timer to update images in fullscreen
frmfull.Timer1.Enabled = True
Else
status.Caption = "Connect first Dumbass"
End If
End Sub
Private Sub Image4_Click()
End
End Sub
```

```
Private Sub screen_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)
State = 0
lpos = 1
bytesR = 0
Close #2
ProgressBar1.Value = 0
bytes.Caption = "0.0 Kb"
percent.Caption = "0 %"
DoEvents
```

End Sub

```
Private Sub screen_MouseUp(Button As Integer, Shift As Integer, x As Single, y As Single)
If Winsock1.State = 7 Then
If screen.Caption = "Get Screen" Then
Winsock1.SendData "screen"
status.Caption = "Started Capturing Screen..."
'screen.Caption = "Stop Screen"
'Call servoposition.Refresh_Click
Else
tcpServer.Close
DoEvents
tcpServer.connect
status.Caption = "Capturing Screen Stopped..."
screen.Caption = "Get Screen"
End If
End If
End Sub
```

```
Private Sub tcpServer_Close()
tcpServer.Close
End Sub
```

```
Private Sub Image3_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)
Winsock1.Close
tcpServer.Close
End Sub
```

```
Private Sub Image3_MouseUp(Button As Integer, Shift As Integer, x As Single, y As Single)
End
End Sub
```

```
Private Sub Image2_Click()
Me.WindowState = vbMinimized
Image3.Visible = True
Image4.Visible = False
Image5.Visible = True
Image2.Visible = False
End Sub
```

```
Private Sub Timer1_Timer()
'tricky bit for speed set a global variable to count up the bytes and calculate them to KBps every
second
Label1.Caption = Format(DoneBytes / 1000, "###0.0") & " Kb/s"
DoneBytes = 0
End Sub
```

```
Private Sub topbar_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)
If Button = 1 Then
gapx = x
gapy = y
End If
End Sub
```

```

Private Sub status_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)
If Button = 1 Then
gapx = x
gapy = y
End If
End Sub

```

```

Private Sub Frmname_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)
If Button = 1 Then
gapx = x
gapy = y
End If
End Sub

```

```

Private Sub Statbar_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)
If Button = 1 Then
gapx = x
gapy = y
End If
End Sub

```

```

Private Sub fullscreen_MouseUp(Button As Integer, Shift As Integer, x As Single, y As Single)
If Winsock1.State = 7 Then
frmfull.Show
frmfull.imgFScreen.Picture = Frmmain.camimage.Picture 'Load image from one form to another
frmfull.Timer1.Enabled = True
Else
status.Caption = "Connect first Dumbass"
End If
End Sub

```

```

Private Sub setqual_MouseUp(Button As Integer, Shift As Integer, x As Single, y As Single)
If Winsock1.State = 7 Then
'send quality data
Winsock1.SendData "setqual;" & quality.Text
status.Caption = "Quality Set " & quality.Text & " %"
Else
status.Caption = "Connect first Dumbass"
End If
End Sub

```

```

Private Sub Winsock1_Close()
'close socket and reset
Winsock1.Close
status.Caption = "Disconnected"
connect.Caption = "Connect"
End Sub

```

```

Private Sub winsock1_DataArrival(ByVal bytesTotal As Long)
On Error Resume Next
Winsock1.GetData Data, vbString, bytesTotal
lastdata$ = Data

```

```
'split the data from server with ";"
Arrayize lastdata$, ";"
Dim thedata As String
```

```
If Cmd(0) = "driverlist" Then
'save the list of drivers
List1.AddItem Cmd(1)
status.Caption = "Drivers Listed"
'load the driver list
Call SaveListBox("C:\drvs.txt", List1)
DoEvents
Call Loadlistbox("C:\drvs.txt", List2)
DoEvents
Kill "C:\drvs.txt"
End If
If Cmd(0) = "qual" Then
quality.Text = Cmd(1)
End If
If Cmd(0) = "screenerror" Then
status.Caption = "Error Capturing Try Again"
screen.Caption = "Get Screen"
capture.Enabled = True
listdrvs.Enabled = True
setdrvs.Enabled = True
End If
End Sub
```

- **FORM Servoposition (Servoposition.frm)**

The whole code of this form is same as the form servoposition in Appendix A except the function refresh_click(). So only function refresh_click has been listed below

```
Public Sub Refresh_Click()
'new readings
ADreading(1, 1) = CStr(SV203.ReadADC(1))
ADC_Value(1).Caption = ADreading(1, 1)

Frmmain.Winsock1.SendData ADreading(1, 1)
If ADreading(1, 1) < 100 Then
Call jerkback
End If
If ADreading(1, 1) > 150 Then
Call jerkfront
End If
End Sub
```