A PROPOSED ARCHITECTURE FOR REMOTE MECHATRONICS LABORATORY

by

C. ONUR SOZBILIR

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science
Sabanci University
Spring 2002

A PROPOSED ARCHITECTURE FOR REMOTE MECHATRONICS LABORATORY

APPROVED BY :

Prof. Dr. Asif Sabanovic …….
(Dissertation Supervisor)

Assoc. Prof. Dr. Seta Bogosayan …….

Assist. Prof. Dr. Berrin Yanikoglu …….

Assist. Prof. Dr. Kemalettin Erbatur …….

Assist. Prof. Dr. Ahmet Onat …….

DATE OF APPROVAL: ……

ABSTRACT

Experimentation is a very important part of education in engineering. This is also true for mechatronics engineering which is a relatively new field, combining three engineering disciplines : mechanical engineering, electrical engineering and software engineering. The equipments needed for experiments in mechatronics are generally expensive. Examples are robot manipulators, mobile robots, electrical motors, fast DSP cards, CNC machines, etc. One solution for expensive equipments is sharing the available equipments with other universities around the world. This relatively new concept, called "*remote laboratory*", is based on the computer communication technology and the internet. With the internet, sharing the available resources with the world costs almost nothing. In this thesis a new architecture for a remote mechatronics laboratory is proposed. The proposed architecture is original in the sense that multiple users can use multiple experiments at the same time. The system is flexible so that new experiments can be added quite easily. In order to reduce the overall cost and increase efficiency, multithreaded programming is proposed to reduce the number of computers necessary. For flexibility, communication is done using objects. Verification of the communication part is done implementing the DC motor control experiment, remotely.

ÖZET

Deney yapmak mühendislik eğitiminin önemli bir parçasıdır. Bu gerçek nispeten yeni bir alan olan, makina mühendisliği, elektrik mühendisliği ve yazılım mühendisliğinin birleşimi mekatronik mühendisliği için de geçerlidir. Genellikle mekatronik mühendisliği deneyleri yapmak için gerekli olan araçlar pahalıdır. Örnek olarak robot kollar, hareketli robotlar, elektrik motorları, hızlı DSP kartları ve CNC makinaları verilebilir. Pahalı araçlar için bir çözüm varolan araçları diğer üniversitelerle ve dünyayla paylaşmaktır. Nispeten yeni bir kavram olan uzak laboratuar şu günlerde hızla gelişen internet teknolojisine dayanmaktadır. İnternet teknolojisi sayesinde halihazırda varolan araçları dünyayla paylaşmanın hemen hemen hiç bir ek maliyeti yoktur. Bu çalışmada uzak mekatronik laboratuarı için yeni bir mimari önerilmiştir. Maliyeti azaltmak ve verimi artırmak için kullanılan bilgisayar sayısını azaltan multithreaded (çok iplikçikli) programlam önerilmiştir. Esneklik için haberleşme nesneler kullanılarak yapılmıştır. Haberleşme kısmı uzak doğru akım motor kontrol deneyi uygulanarak test edilmiştir.

Dedicated to my family, Baris, Naciye and Halim
and to the spirit of Asimov.

ACKNOWLEDGEMENTS

I would like to thank to my professors, Professor Asif Sabanovic and Professor Gokhan Goktug for their support and encouragement during my two years master's study.

Special thanks goes to my family, my brother Baris, my mother Naciye and my father Halim for their support.

I would like to thank to also my dear friends in Mechatronics graduate laboratory, and professors Kemalettin Erbatur and Ahmet Onat for their kindness.

TABLE OF CONTENTS

Page

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Remote Laboratory

Education in engineering has two main parts: theory and practice. Theory involves mathematical and scientific rules, axioms, theorems and some derivations and conclusions which does not require much cost. Practice, on the other hand, involves experimentation and experimentation requires tools, both hardware and software. This is also true for mechatronics engineering education. Mechatronics is a relatively new term, first coined in 1970s, combining the traditional engineering disciplines of mechanical engineering and electrical engineering with a relatively new engineering discipline, information or software engineering. Therefore mechatronics equipments are among the most complex engineering equipments having parts related with all three disciplines. A strong mechatronics education can not be thought without practice gained in experimentation. Mechatronics equipments are among the expensive equipments, like electrical motors, DSPs, robots, PLCs, etc. So, a university considering to build a well-established strong mechtronics laboratory has to spend considerable amount of money. A relatively new concept called remote laboratory or virtual laboratory offers a good way of sharing the available resources with other universities and institutions in the world with almost zero cost. This is due to the already established computer communication network, called internet. With the use of internet, the laboratory of a university can be at least partially open to other students in other universities without any addititonal cost. There is only one requirement for remote users of the laboratory which is they must have a computer connected to the internet and this is quite affordable nowadays. This idea is roughly skectched in Fig1.1.

Remote laboratory concept does not only allow to share the available resources with other universities but also allows Sabanci University students to use the laboratory more effectively and efficiently. A Sabanci University student may access the laboratory from his/her dorm or even from his/her house in a remote city.

Remote Laboratory is relatively new concept but there is some good work done on the subject. Some good examples are the virtual engineering laboratory developed by Carnegie Mellon University[1], remote control engineering laboratory at Oregon State University[2], remote control engineering laboratory at the National Polytechnique Institute of Grenoble, France[3], a remote measurement laboratory at University of Naples and University of Salemo in Italy[4], internet-based real-time control engineering laboratory at Polytechnique University[5], and remote process control laboratory by Case Western Reserve and Cooper Union Universities[6].

In [1], electronic equipments such as signal generator, voltmeter and oscilloscope are connected to the internet to provide a remote electronics laboratory. Two experiments, "the black box" and "martian rescue" are conducted by undergraduate students remotely. In "the black box" experiment, students are asked to characterize an unknown circuit from terminal measurements. In the "martian rescue" experiment, students are asked to control the operation of a reversible motor drive for a camera.

In [2], they described a remote control engineering laboratory based on client/server scheme and demonstrated the remote experimentation with a 3 degree-of-freedom (3-DOF) robot arm. They enabled remote users to receive visual and audio information from the laboratory and a communication tool, which they called the whiteboard, to enable users to collaborate with each other. This is a piece of CRT screen shared among users.

In [3], remote access to the "ball and beam" experiment, which is a widely used control engineering experiment, is established. Remote users can select among pretuned PD, PID or RST controllers or define their own RST controllers and test it on the system to see the success or the controller.

In [4], a remote access to automatic measuring setups and instruments is described. They connected a DSP analyzer, a digital oscilloscope and a sine generator to internet. They used Visual C++, HTML, CGI and PERL as software.

In [5], they describe the "Client-Enhanced Internet-Based Remote Laboratory" which gives the remote user more authority and responsibility over the experiment. In

this scheme, the remote user can decide whether the controller runs on the server computer located in the plant site or on the client computer, and will send the control signals through network. They implemented this scheme on a remote digital PID control where there is a delay between the plant and controller. Their remote controller works well because the plant's response time is about 3-5 seconds, which is much larger than the delay on Local Area Networks (LAN).

In [6], a remote access to a process control setup is established. The process is a water flow system where flow, level and temperature may be measured and controlled. There can be two experiments done on the setup: Remote sensor calibration experiment and remote PI control of flow and level experiment.

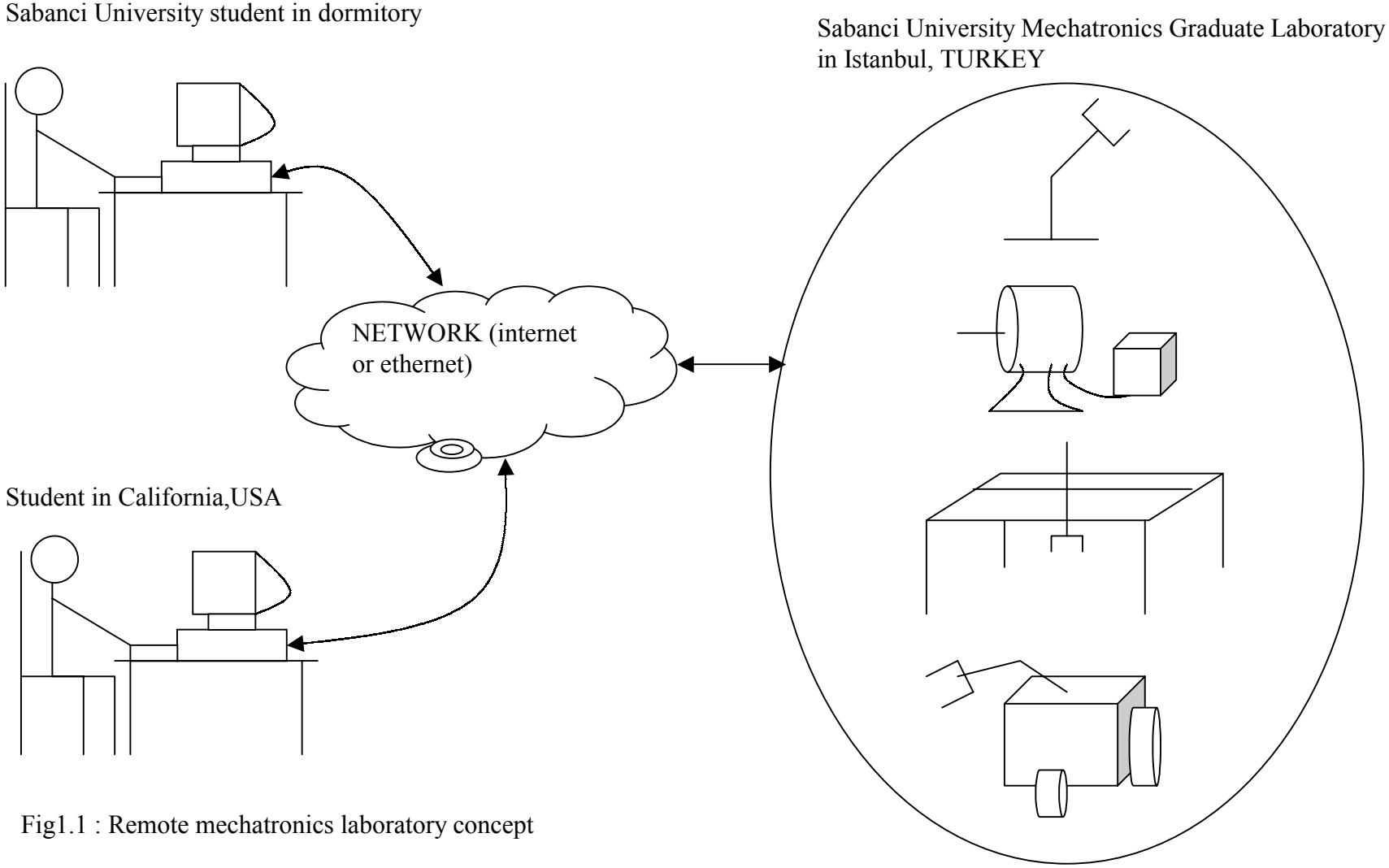


Fig1.1 : Remote mechatronics laboratory concept

1.2 Communication with TCP/IP Sockets

1.2.1 International Organisation for Standards Open System Interconnection Reference Model (ISO OSI-RM)

The internet can be defined as a huge network of computers distributed throughout the world connected in such a way that any computer can send and receive data to any other computer. If someone is to connect two computers with the same properties, then he/she can write a program to send and receive data between these computers. But if it is desired to write a program that can send/receive data to computer with unknown properties then some set of rules must be established. These rules are called protocols. In order to set standard international rules there is an organisation called International Organisation for Standards (ISO). For computer network communications, ISO defined a set of standards known as Open System Interconnection (OSI). The "Open Standard" means that the software is open to anyone who wants to design a product using the standard. So, the OSI is not hidden but available to anyone.

OSI consists of seven layers each responsible for managing some function related to data transfer between machines like transport of data, packaging of messages, end-user applications, etc[X]. These layers are application, presentation, session, transport, network, data link and physical layer, shown in Fig1.2.

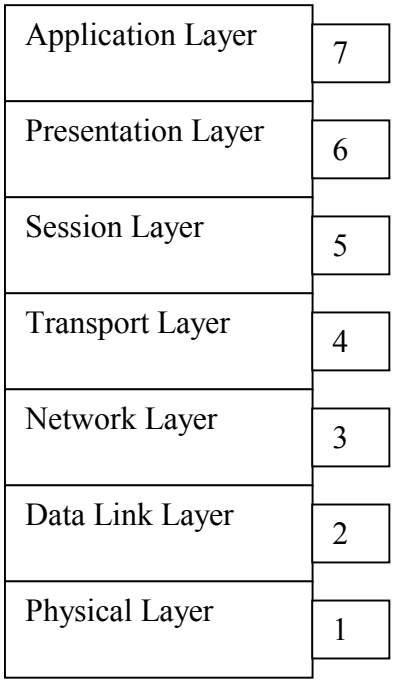


Fig1.2: Seven layers of OSI

Application Layer :

The application layer is the layer between the user and the other lower layers. It displays the received data to the user and sends the user data to the lower layers.

Presentation Layer :

The presentation layer converts the data from machine dependent format to the machine independent format called canonical representation.

Session Layer :

The session layer is responsible for synchronisation of the exchange of data between processes which can be in the same or different machines.

Transport Layer :

The transport layer is responsible for establishing, maintaining and terminating the communication between two machines. The order and priority of data is managed by this layer also.

Network Layer :

The network layer is responsible for routing the data and determining routing paths between machines.

Data Link Layer :

The data link layer detects and sometimes corrects the error occurred in the physical layer.

Physical Layer :

The physical layer deals with the electrical transmission of data as its name implies. So, the wiring is in this layer.

In order to have more efficient communication data is sent in uniform quantities, called packets. Each layer adds it own protocol header or protocol control information to the front of the user data to be sent and when the data is received each layer removes its own header to obtain the raw data. This is shown in Fig1.3.

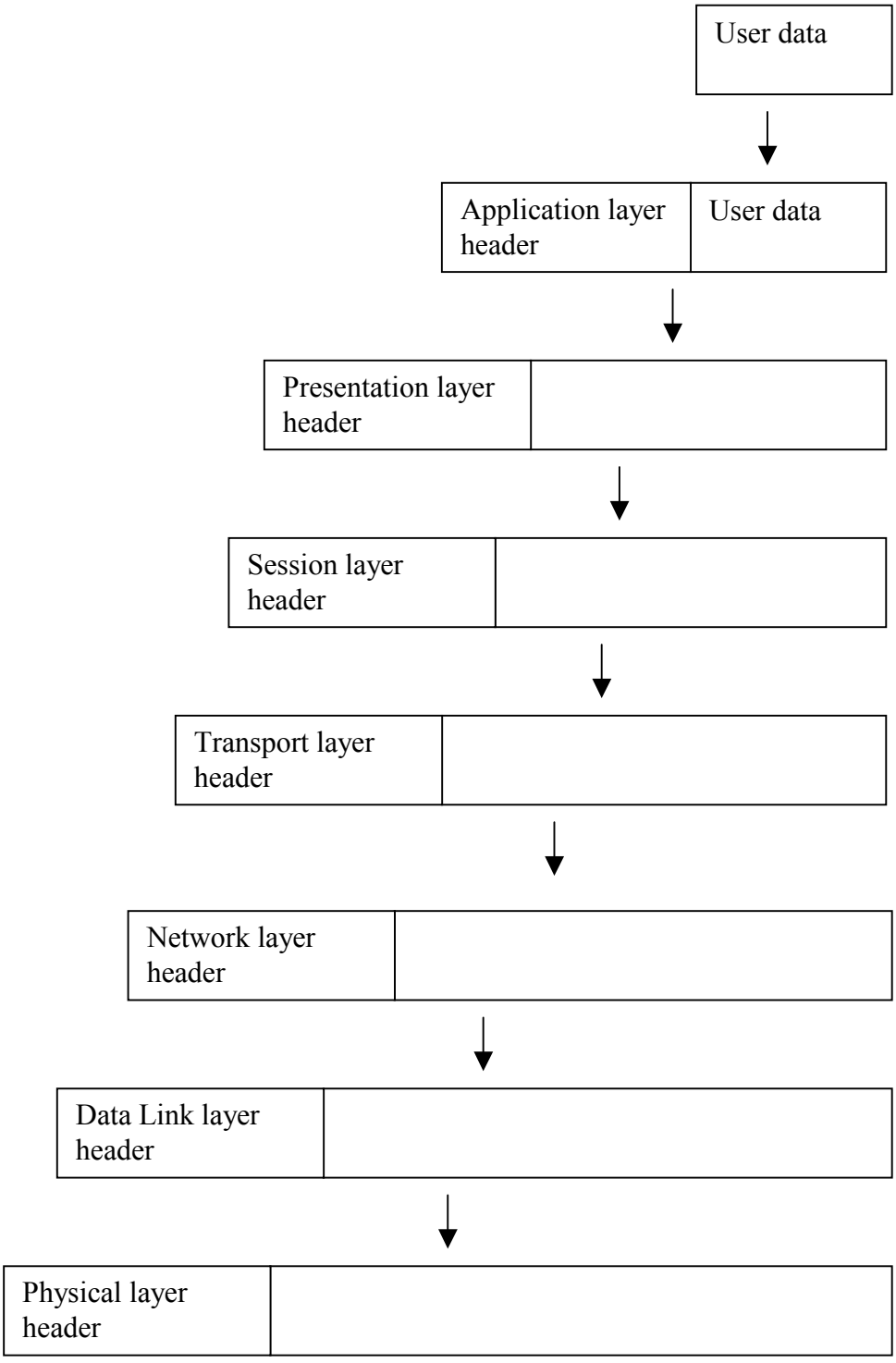


Fig1.3: Each layer adding headers to the user data

1.2.2 TCP/IP

TCP/IP is a software-based communications protocol. It can handle errors in transmission, manage the routing and delivery of data, and control the actual transmission by the use of predetermined status signals[X]. TCP/IP resides in the network layer and transport layer of the OSI reference model. So, it is not dependent on the bottom two layers, data link layer and physical layer. Internet Protocol (IP) is in the network layer and Transmission Control Protocol (TCP) is in the transport layer. TCP/IP is based on the concept of client/server. The device initiating the communication is the client and the device responding to the request is the server. TCP/IP is the most widely used protocol over the internet. The reasons are:

- TCP/IP is proved to be running effectively and efficiently for quite a time.
- TCP/IP has a well-established management system.
- There are thousands of programs using TCP/IP.
- It has well-documented Application Programming Interface (API).
- It is vendor-independent protocol.

TCP/IP also works well with Ethernet on Local Area Networks (LAN). Ethernet provides the lowest two layers, data link layer and physical layer and TCP/IP provides the network layer and transport layer.

Internet Protocol (IP)

Internet Protocol (IP) sits in the network layer of the OSI reference model. All machines on the internet can use IP and IP can be used on networks that are not connected to internet also. The responsibilities of IP are datagram routing, determining where the datagram will be sent and finding alternate routes when a problem occurs. But the delivery of datagrams is not guaranteed by IP. It has no capability to verify that a sent message is received correctly by the destination. IP is connectionless. This means that all datagrams don't need to follow the same route. The IP header is five or six bytes including sending address, destination address and other information about the datagram.

Transmission Control Protocol (TCP)

Transmission Control Protocol (TCP) sits in the transport layer of the OSI reference model. It is a connection-oriented protocol meaning that all datagrams are sent through the same route once a communication is started between two machines. A connection-oriented communication has three stages : establishing the communication,

data transfer and terminating the communication. TCP assures that the datagram is sent correctly. If the datagram is lost or corrupted, TCP handles the retransmission. So, TCP provides end-to-end communications ensuring the transfer of a datagram from the source to the destination. The sending machine waits for an acknowledgement (ACK) signal for a certain amount of time and if the time expires resends the datagram. This way the correct delivery of data is guaranteed with a sacrifice of time delay. If a slight increase in speed of transfer is much more important than the correct delivery of data then an alternative protocol, User Datagram Protocol (UDP) can be used. UDP is a connectionless protocol with smaller protocol header.

1.2.3 Programming with Sockets

When using connection-oriented communication, TCP/IP uses sockets which can be thought as end points of a virtual circuit. This is shown in Fig.1.4.

Fig1.4 : Computers communicating with sockets

If a programmer wants to write a program that is using TCP/IP, he/she must use the socket programming interface functions. These are well-defined, operating system independent and programming language independent functions, called Socket Application Programming Interface (Socket API). The Socket API was developped at the University of California at Berkeley. Basically there are six communications functions of the Socket API: open, send, receive, status, close and abort.

open: open function is used to initiate the communication. There are basically two types of open commands: passive open which is used to prepare a server to communication and active open for client. Passive open involves the following : creating the socket returning a number for the new socket, binding the new socket to a local port address, listening to the socket for any requests from clients and accepting client requests. Active open involves the following: creating the socket returning a number for the new socket and connecting to the server socket.

send: send function is used to send data to the remote computer. It has five different versions: send, sendto, sendmsg, write and writev each are similar with slight differences.

receive: receive function is used to receive data from the remote computer. It also has five different versions: read, readv, recv, recvfrom and recvmsg.

status: status functions are used to obtain information about a connection. They are usually used in case of errors.

close: close function is used to terminate the connection.

abort: abort function is similar to close function but it is used in more urgent cases when an emergency shutdown is required.

Using the MFC Winsock Classes

Before Windows95, networking was not built into the Windows operating system. So, a programmer who wanted to build a network communication program had to buy a network software from numerous different companies. The software of each of these companies had their own ways of using sockets for applications. So, a program that performed network communications had a list of different networking software in order to work properly. In order to prevent this, all the networking companies and Microsoft together developed the Windows Socket API (Winsock API). The advanced Integrated Development Environment (IDE) of Microsoft, Visual Studio 6, has its strong built-in libraries called Microsoft Foundation Class (MFC). MFC includes a lot of useful functions and classes that facilitates programming a lot. Using MFC Winsock classes building software for network communication is quite helpful. Among some of the base classes the class CAsyncSocket is one of the most powerful base class that provides event-driven communications. Event-driven means that you can define some function that will be called upon some interrupts. So, for example when the socket receives a data, it informs you by setting an event and your function that handles this data runs when this event is set. This removes the need of polling, which consumes CPU time and makes the program more complicated.

1.3 Multithreaded Programming

1.3.1 What is multithreading?

Traditional computer programs are typically composed of sequential program statements executed one after the other. This sequence of program statements executed by the computer is a single thread of execution. If the program needs to accomplish several different tasks simultaneously then the programmer must consider and design the program himself in such a way that it achieves the desired concurrency. The required mechanism is shown in Fig1.5 below.

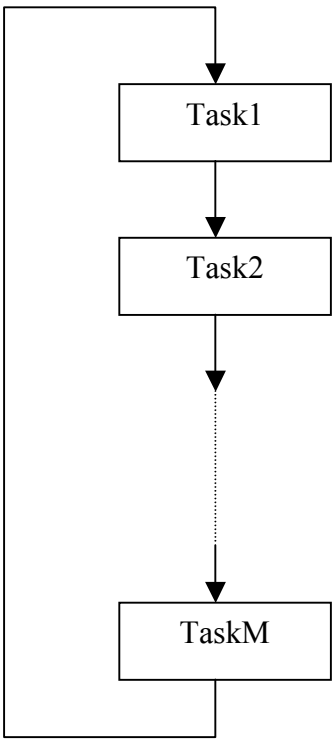


Fig1.5 : The mechanism to achieve concurrency in single-threaded programming

Here, in order to achieve a certain level of concurrency the single-thread does not complete a task fully before executing next task. Instead the single-thread partially completes the task for a certain time and then saves the information about the current state of the task in order to resume correctly in the next iteration of the loop. Although this seems to be a solution for dealing with multiple tasks at the same time, it loads a lot of burden on the programmer and has other disadvantages. For example, if a task is blocked for some reason, the whole program will stop.

Some operating systems use the multi-tasking technique described above by running multiple programs, or processes, at the same time. In these systems the operating system is responsible for switching from one process to the other.

A good alternative to multi-process programming is multi-threaded programming where multiple threads, which can be thought as light-weight processes, are running in a process at the same time.

With the use of pre-emptive multi-tasking in modern operating systems like Windows NT, the switching between threads is under the responsibility of the operating system. So, the programmer does not need to deal with the scheduling of tasks but may simple regard them individual threads as running concurrently.

The structure of multi-threaded program from the programmers' point of view is shown in Fig1.6 below.

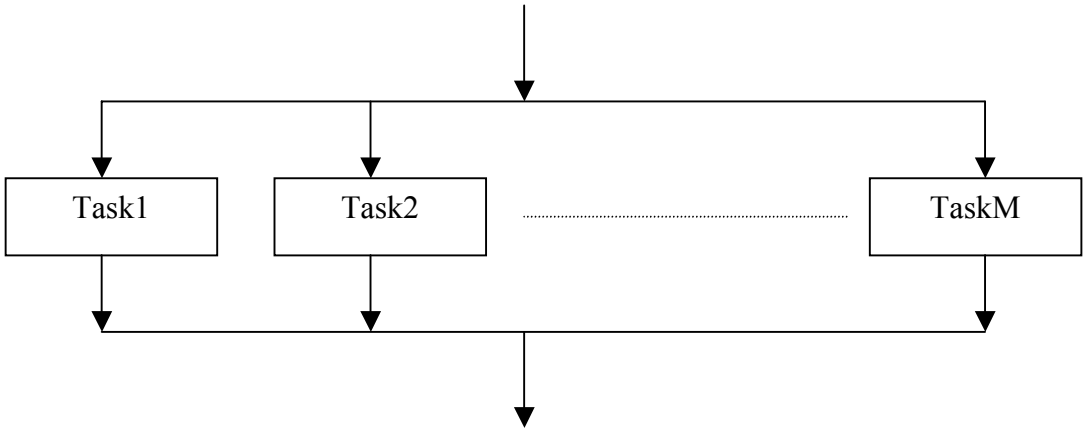


Fig1.6 : The structure of program from programmer's point of view in multithreaded porogramming

Thread Scheduling and States of a Thread

In most of the systems there are more threads than processors. In order to achieve concurrence some kind of scheduling must be done by the processor. For systems with single-processor usually CPU utilises the time-slicing scheme as scheduling. In time-slicing scheme each thread is allowed to use CPU for a certain amount of time. This is shown in Fig1.7. Context switch includes saving the context of currently running thread and loading the context of newly scheduled thread.

The lifetime of a thread consists of four states, namely ready, running, sleeping and terminated. This is shown in Fig1.8. Running state is the state when the thread is actually using the CPU. Ready state is the state when the thread waits for the scheduler to give the CPU to the thread. Most of the time the thread is in ready or running state. Some times the thread must wait for some event of some time to continue running correctly. In this situation the thread enters the sleeping state. Scheduler doesn't allocate the CPU time to the thread until the thread enters the ready state. After the thread

accomplishes its task fully, it enters the terminated state, waiting for the CPU to delete it. This may be important in some cases because other threads may need to know whether this thread is still alive or not.

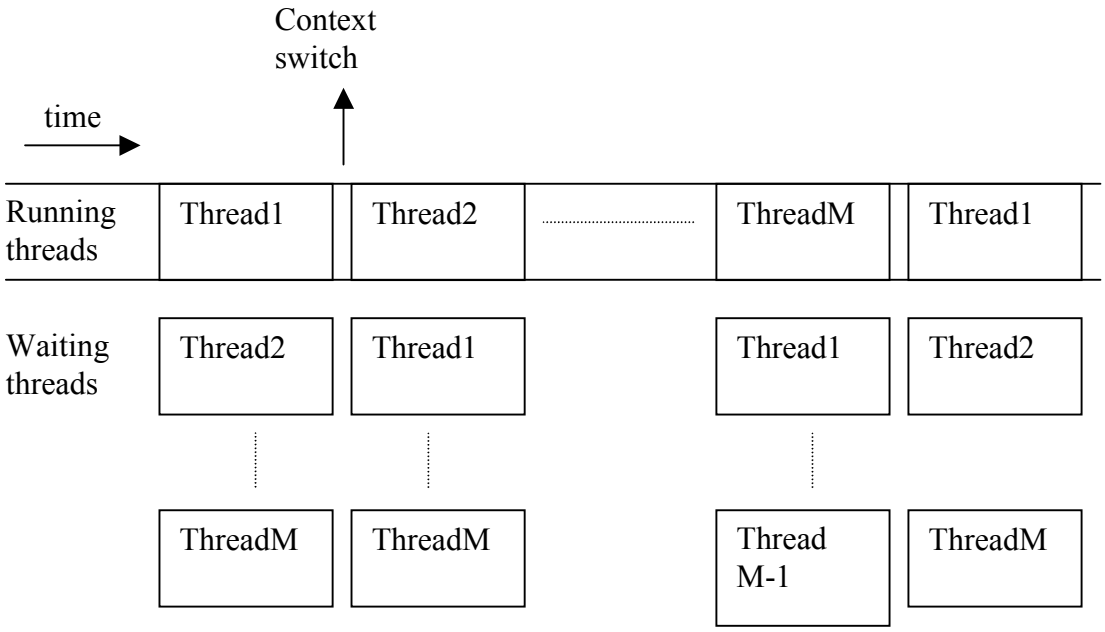


Fig1.7 : The timing chart of the running threads organized by the scheduler

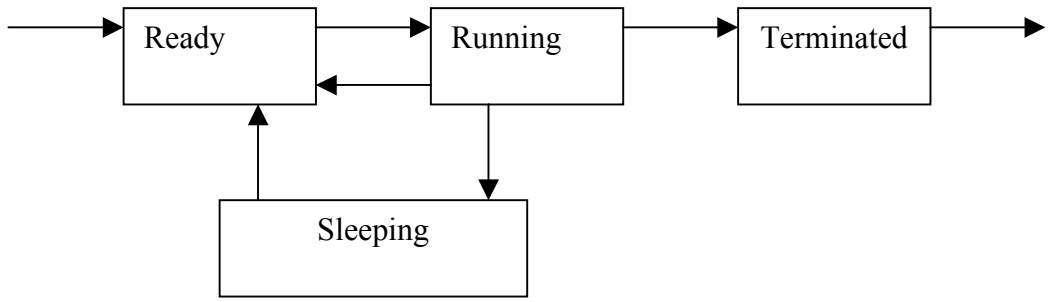


Fig1.8 : The states of a thread

Thread environment

Threads are in the environment of a process. Some part of this environment is commonly shared by all threads belonging to the process while some part is specific to each thread and not shared by other threads. This is shown in Fig1.9. Shared environment consists of the executable code and data storage. Thread specific environment or namely thread-specific context, consists of stack, registers of CPU and keys defined by user. Data in storage exist whole time the process is running while thread-specific context must be switched when the scheduler assignes CPU to a new thread.

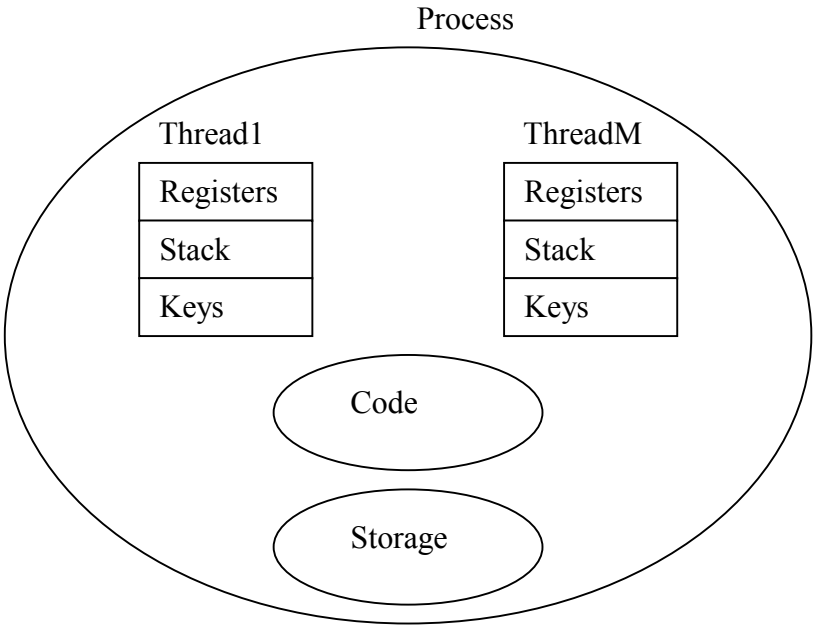


Fig1.9 : The environmet of a thread is within a process

1.3.2 Why is multi-threading used?

A programmer must have a reason to learn, exercise and use multi-threaded programming because it takes time to learn this style and it is not so easy to write well-behaved, efficient, bug-free multi-threaded programs. Some of the advantages of multi-threaded programming may be listed as follows:

Faster processing and more efficient use of CPU:

Typically if the application is suited to multi-threaded programming and it is well designed, then it will increase the speed of program and a better utilisation of CPU will be achieved. For a single-threaded program the time it takes to complete the multiple tasks is the sum of the times needed to complete each task. This is shown in Fig1.10.

However a multi-threaded program completes all of the tasks in almost the same time needed to complete the longest task. This is shown in Fig1.11.

Additional small amount of time is due to switching between threads. This may be ignored in most cases if the multi-threaded program is written is written carefully for the application it is suitable for.

More robust program:

By isolating tasks into different threads the robustness of the program may increase. If one of the tasks fail for some reason, other tasks will be accomplished by threads responsible for them in a carefully designed multi-threaded program.

Better response to the users:

By using separate threads to tasks taking long time, the user can still use the system with the threads responsible for user interface. Background printing is a good example for this. A user can still use the computer while printing a long document.

Some of the advantages above can be more or less gained by using multi-process programming. So, what is the advantage of multiple threads to multiple processes? The biggest advantage of multiple threads is absolute: threads are cheap. They can be started-up and shut down very fast, much faster than processes and this results in minimal use of system resources. So, a few hundred threads may run without any problem on the same processor while a few hundred processes will have big load on processor resulting in degradation in performance. Second, the switch from one process to other takes much more time than that of threads. This will also effect system performance a lot. Another advantage of threads over processes is threads share the same address space in memory, so they can communicate between each other much faster and easier than processors can do.

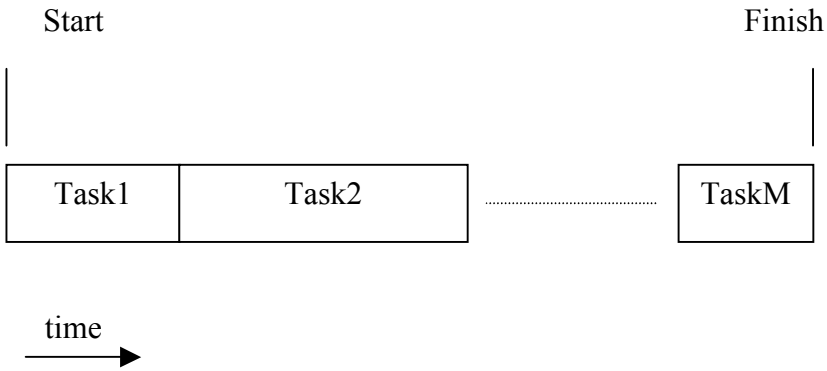


Fig1.10 : In single-threaded programming the time needed to complete multiple tasks is sum of the times needed to complete each task

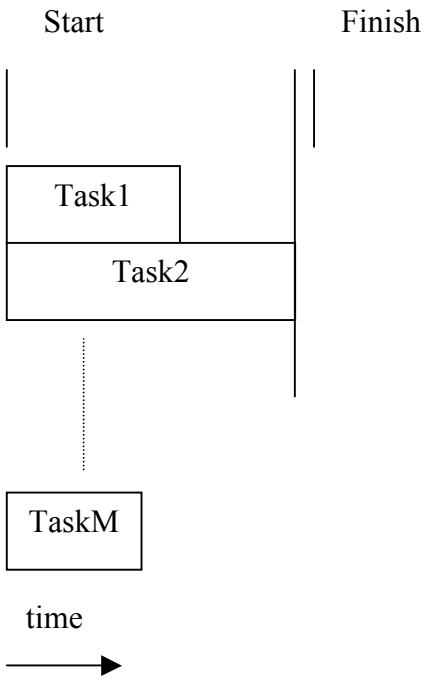


Fig1.11 : In multi-threaded programming the time needed to complete multiple tasks is almost the same as the time needed to complete longest task

Why is multi-threading preferred in this study?

One of the main concerns in this study is to allow multiple users control multiple plants virtually at the same time using minimum number of PCs. So, this immediately requires that one PC must control multiple plants at the same time. There are two choices to accomplish multiple tasks with a single processor virtually at the same time: multi-tasking using multi-process programming and multi-threading. In order to have relatively fast system and for the reasons mentioned in the previous section, multi-threading is preferred in this study. Consider two users trying to conduct experiments controlled by the same PC at the same time. If single-threaded programming is used and if one of the users sends a long task to be performed on the experiment side then the other user has to wait until that task is completed. If the number of users and number of plants controlled by a single PC increases considerably then it will be much worse with single-threaded programming. With multi-threaded programming approach no matter how long the task of some users, other users will still use the system efficiently.

1.3.3 Thread Synchronisation

Why?

Communication between threads of a process is usually made by modification of the shared process environment. However, the interrupt times of the scheduler may be unpredictable. This brings the probability that the process environment is not left by one thread in a consistent state ready for the other thread to use it. In order to prevent this there are some "thread synchronisation" techniques used. The main idea lying behind these techniques is to ensure that each modification of the process data is completed before control is given to another thread. These techniques are needed not only to prevent data corruption but also to allow multiple threads work in a desired way. For example, one thread may need to wait for something to happen depending on the application and other thread may inform this thread to continue operation. Some of the synchronisation mechanisms used are:

- Critical section
- Mutex
- Semaphore
- Event

These are briefly explained in this section.

Critical Section

Critical Section is a part of the program that has access to a shared resource. It is used to prevent data corruption due to multiple threads accessing a shared data at the same time. This is done by allowing only one thread to be inside the critical section. So, a thread may access to a critical section only if there are no other threads using that code at that moment. Critical section can only be used for threads in the same process. For threads in different processes mutex, which is explained next, must be used.

Mutex

Mutex is very similar to critical section in functionality but it has some differences. Some of the differences between mutex and critical section are:

- A mutex needs much more time to be locked (entered) and unlocked (left) than a critical section does.
- A mutex can be used with threads between processes but this is not true for a critical section.
- A timeout for waiting on a mutex can be defined but not for critical section.

Semaphore

A semaphore can be thought as a more general form of mutex. Only one thread can access a mutex at a certain time while the number of threads that can access a semaphore can be adjusted. This number is set as initial count of semaphore. When a thread accesses the semaphore the count is decreased by one and when a thread leaves the semaphore this count is increased by one. When the number reaches to zero, no more thread can access the semaphore. So, this way number of threads accessing a semaphore is limited in a desired way.

Event

Event is a kernel object, which can be in either of two states: signalled or non-signalled. This state is controlled by the programmer in a desired way. A thread waiting for an “event” to be “signalled” will go into the “blocked” mode saving more CPU time for the whole application. When the event object becomes signalled the thread will go into the ready mode and execute whenever the scheduler gives the CPU to it.

CHAPTER 2
SYSTEM DESCRIPTION

2.1 System structure : Hardware

We can divide the system into two main parts (Figure 2.1) :

- Client side
- Server side

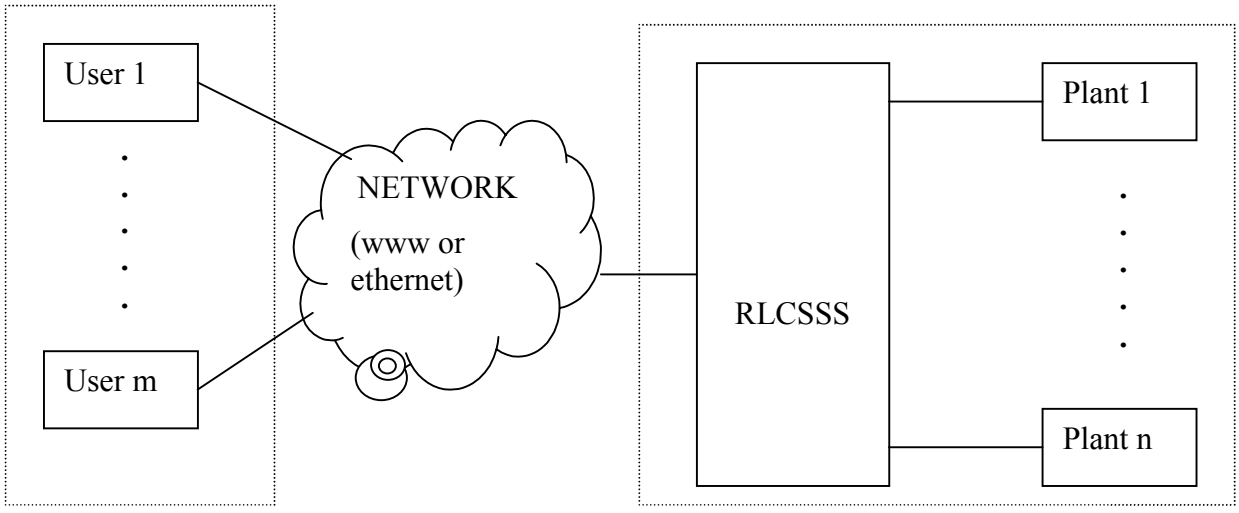


Figure 2.1 Client/server view of the system

Client side is composed of user pcs connected to the internet or ethernet. This is Somehow user interface of remote users to the remote laboratory system. Users generate their desired commands to conduct the experiment and see the experimental results using their pcs.

Server side is composed of pcs responsible for accepting the remote user commands, controlling the plants and sending back the experiment results as feedback

to the remote users. The communication between client side and server side is done through internet or ethernet. The system server side is designed in such a way that additional plants can be integrated to the system without any problem easily.

One of the main concerns in this study is to allow multiple users control multiple plants at the same time with minimum number of PCs on the serverside in a reliable, efficient way. For this reason the structure in Figure 2.2 is chosen.

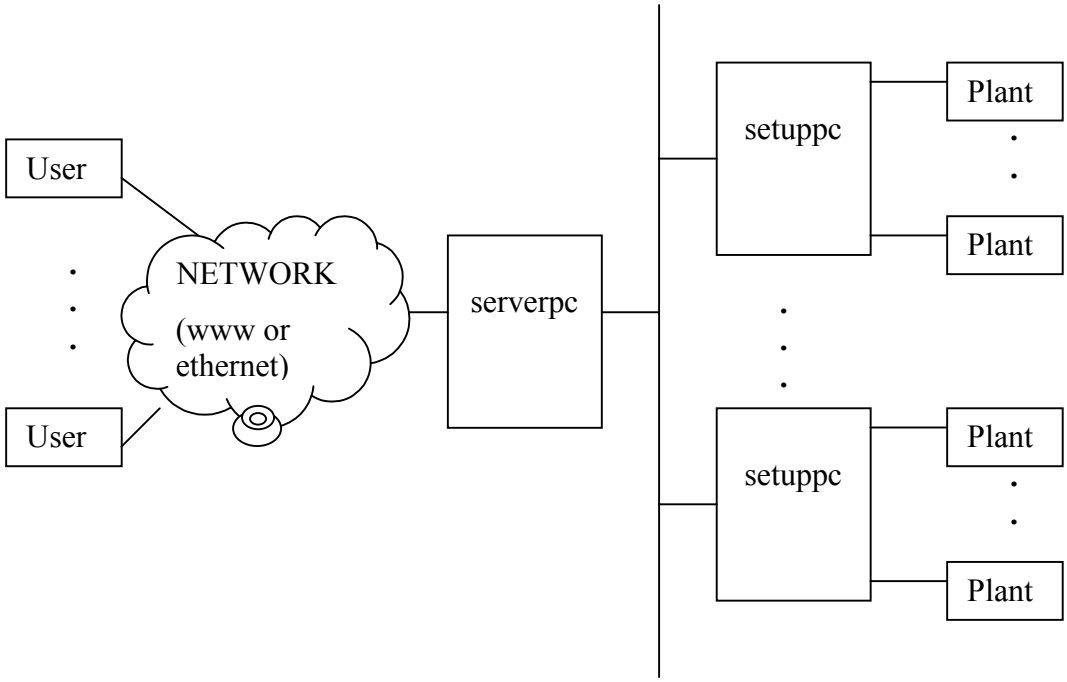


Figure 2.2 Hardware topology of the server side for efficient use of pcs

Here, there are two types of PCs in server side. One is **setuppc** responsible for controlling the plant and recording the state of the plant and second is **serverpc** which is the gate of outside world to the laboratory, responsible for accepting remote user commands and sending them to the setuppc controlling the corresponding plant, receiving feedback from plants and sending them to the corresponding remote user. Here, multiple plants are connected to each setuppc. Number of plants on each PC depends on the computation power for each plant. Furthermore, multiple setuppcs are connected to one serverpc. Here, there can be more serverpcs depending on the number of plants and users which effect the traffic. If the demand on the laboratory is high then additional serverpcs can be added to take the communication load.

2.2 Communication with objects

There are two types of interaction for any experiment performed by a human : One is the set of actions by the human to control the experiment in a desired way, such as setting the parameters or starting and stopping the experiment. Second is recording, observing and commenting on the experiment results. Control of a robot arm can be given as a good example (Fig2.3).

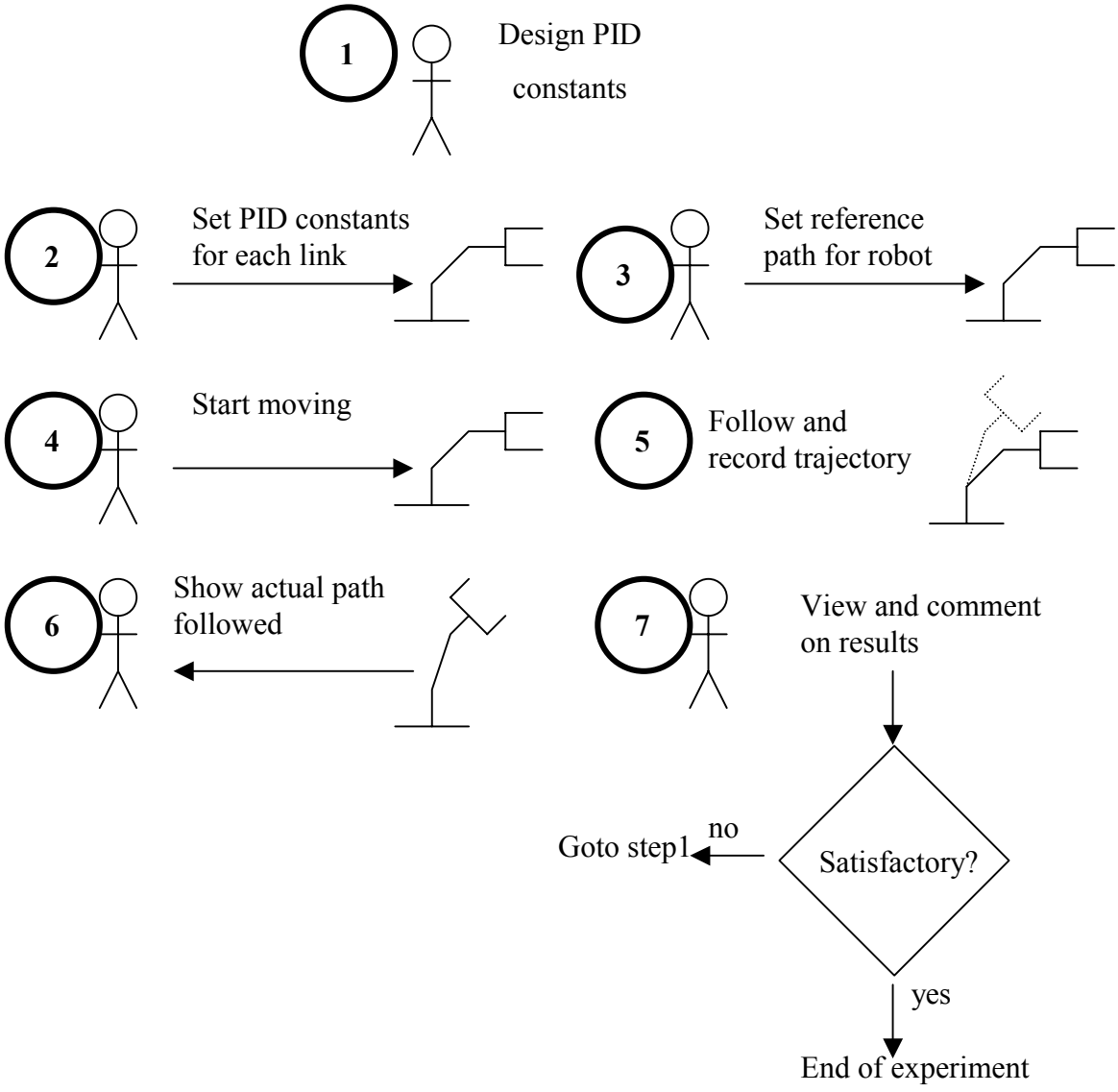


Figure 2.3 A typical robot-arm control experiment

If a control engineering student wants to test the independent joint PID control algorithm he/she has designed practically on a robot arm, he must be able to set the PID constants for each link and give the reference path to be followed by the robot. Also, he/she must be able to start and stop the experiment. These belong to the first type of interaction between the human and the experiment, action from the human to the experiment, namely **command**. In order to complete the experiment the actual angles or path followed by the robot must be recorded and shown to the human. These belong to the second type of interaction, action from robot to human, namely **feedback**. These two interactions, controlling and observing an experiment, are necessary and sufficient to perform any kind of experiment. So, if one can perform these two remotely, he/she can conduct an experiment from a remote place. For this reason, in this study two main objects are defined : **cmd** (for command) **object** for controlling the experiment and **fb** (for feedback) **object** for observing the experiment results. Different sets of cmd and fb objects must be defined for different experiments. For similar experiments similar objects can be used. Two examples of set of objects for two experiments are defined below.

cmdname	**arg**	**fbname**	**arg**
set controller type	0, 1, 2	controller type	0, 1, 2
set reference type	0, 1	reference type	0, 1
set kp	kp	current kp	kp
set ki	ki	current ki	ki
set kd	kd	current kd	kd
set kfr	kfr	current kfr	kfr
set amplitude	amplitude	current amplitude	amplitude
set frequency	frequency	current frequency	frequency
start experiment	0	angular velocity	w

cmdname	**arg**	**fbname**	**arg**
set reference type	0, 1	reference type	0, 1
set kp	kp1, kp2, …, kp7	current kp	kp1, …, kp7
set ki	ki1, …, ki7	current ki	ki1, …, ki7
set kd	kd1, …, kd7	current kd	kd1, …, kd7
add reference 1	x, y, z, yw, p, r	current pos & or	x,y,z,yw,p,r
add reference 2	theta vector	current thetas	theta vector

In the example experiments above, the user decides on the action he/she wants to perform and sends this to the laboratory through the internet or ethernet. The computers in the laboratory receive this desired action, or command, and transmit it to the corresponding plant through serverpc and setuppc, respectively. The desired command is executed, result is recorded as fb and sent back to the remote user for feedback to complete the experiment. Since the communication is done through internet or ethernet there is unpredicted amount of delay. In order to perform a more efficient experiment reducing the effect of delay, two other objects are defined, namely **task** and **feedback**. Task object consists of consecutive set of cmd objects defined by the user. This way the user can first design the whole set of actions he/she wants to be executed by the experiment and then send them to the laboratory. And the setuppc records the fb objects for each cmd object while executing the experiment, collects them in a feedback object and sends the feedback object back to the user. So, the two new objects defined are task object consisting of successive cmd objects and feedback object consisting of successive fb objects.

Since there will be multiple plants and multiple users, there will be multiple task and feedback objects at the same time in the system. In order to send the task and feedback objects to the correct plant and user, respectively, each object must carry source and destination addresses. So, each plant and user are assigned an address. Plant addresses are static while user addresses are dynamic. Each time a user reenters the system, he/she is assigned a new address. This address may depend on, but is not the same as, the IP address of the user.

In order to have a more generic and flexible system, each task and feedback object is assigned a priority and an accessrank. For example the owner of the laboratory, Sabanci University (SU), may want to give more priviliges to its own students or to the students of some other universities. The priority of a task or feedback object is defined as follows:

If there are two or more task or feedback objects on a PC at the same time, the one having higher priority is executed first. So, the users having higher priorities will reach the system faster than others.

Accessrank is defined for only tasks. The accessrank of a task object defines the level of authority of the user to use the plant. For example, graduate students registered to the system studying robot control may have full authority to use the robot arm so that they can define their own controllers while unregistered users having low accessrank

may only define reference path of the robot. As mentioned above, these two properties are defined to make the whole system more generic and flexible, but they may or may not be used. In order to collect statistics and record the events all task and feedback objects are labeled by an id, namely taskid and feedbackid, respectively.

So, the whole structure of task and feedback objects of the system are as shown below:

Task	**Feedback**
Task.id	Feedback.id
Task.source	Feedback.source
Task.destination	Feedback.destination
Task.priority	Feedback.priority
Task.accessrank	Feedback.taskaccepted
Task.cmdlist	Feedback.fblist

2.3 System Structure : Software

One of the main concerns in this study is to allow multiple users conduct multiple experiments virtually at the same time with minimum number of PCs. In order to reduce number of PCs used, multiple plants need to be controlled by each SetupPC. So, SetupPC must accomplish multiple tasks at the same time. Two approaches can be used for this case. One is multitasking by writing multiple programs and ask the operating system to run all at the same time. The other is multithreading by writing multiple special functions, called threads, and running them at the same time in the same program. Multithreading is more preferable to multitasking since communication between threads is easier, faster and safer than communication between processes. Some advanced Computer Aided Software Engineering (CASE) tools, like Visual C++, have functions to manage reliable thread communication and syncronization. So, multithreaded programming is chosen in this study.

There are basically three types of threads:

- **PAT** : Plant Application Threads
- **IT** : Interface Threads
- **GUIT** : Graphical User Interface Threads

Plant Application Threads (PAT) are responsible for controlling the experiments. For each experiment there is a separate PAT and TaskList.

PAT's operation can be divided into four steps :

- Read the next Task to be executed from TaskList
- Check whether the Task is safe
- Apply commands of the task one by one to the experiment and record fbs
- Build Feedback object and write it to the FeedbackList

PAT blocks itself if the TaskList is empty in order to save processor time to other threads.

Interface Threads (IT) are responsible for communication of Task and Feedback objects between PCs. There are basically two types of ITs: Receiver IT and Sender IT.

Receiver IT's operation can be divided into two steps:

- Receive Task or Feedback object from other PC through TCP/IP socket
- Write received object to TaskList or FeedbackList.

Sender IT's operation can be divided into two steps:

- Read next object from TaskList or FeedbackList
- Send the read object to the corresponding PC.

Each IT is given a specific name according to its position. The ITs between SetupPC and ServerPC are called SetupPC-ServerPC Interface Thread (**SSIT**), the ITs between ServerPC and UserPC are called ServerPC-UserPC Interface Thread (**SUIT**) in ServerPC and UserPC-ServerPC Interface Thread (**USIT**) in UserPC.

Graphical User Interface Threads (**GUIT**) are responsible for allowing user to define his/her command and show feedback to user. There are two types of GUITs:

- Task GUIT
- Feedback GUIT

Task GUIT's operation is can be divided into two basic steps:

- Take user actions as input and build Task object
- Write Task object to the TaskList

Feedback GUIT's operation can be divided into two basic steps:

- Read Feedback object from FeedbackList
- Show Feedback data to the user

Software structure of programs running on each PC are explained in this section. First, general block diagram is given. In block diagrams, circles represent threads, large rectangles represent object arrays of Task and Feedback objects, namely TaskList and FeedbackList, respectively, and small rectangles represent TCP/IP sockets. Then flow chart of each thread is explained in detail.

2.3.1 SetupPC

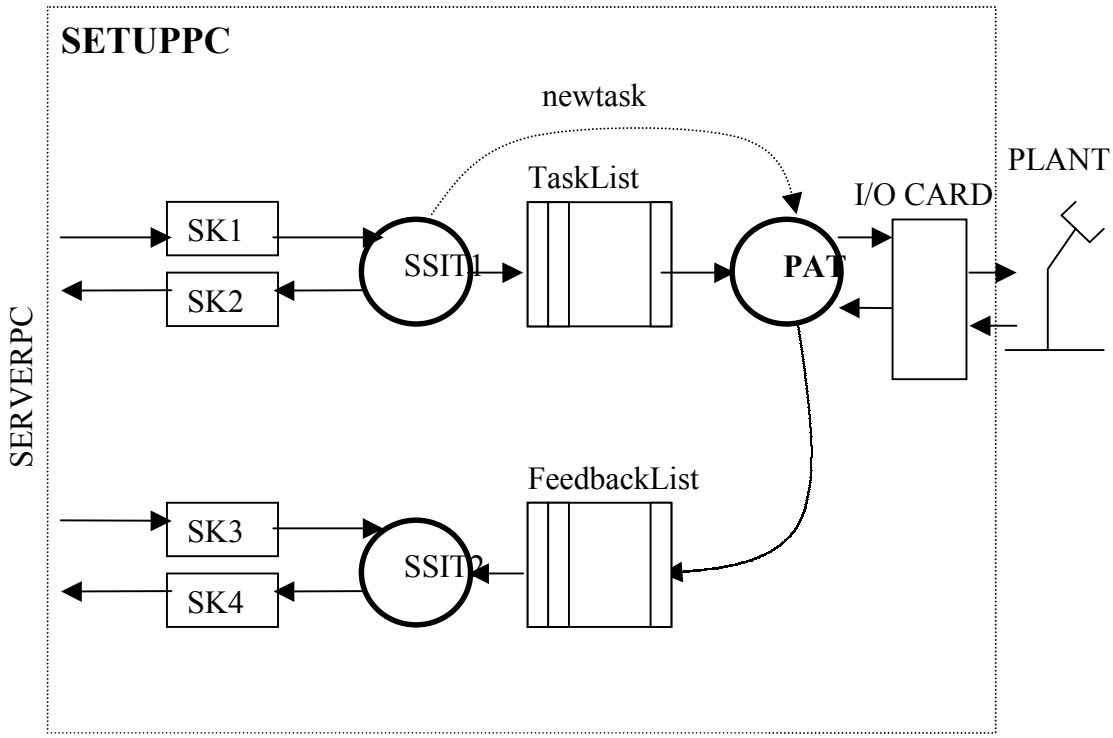
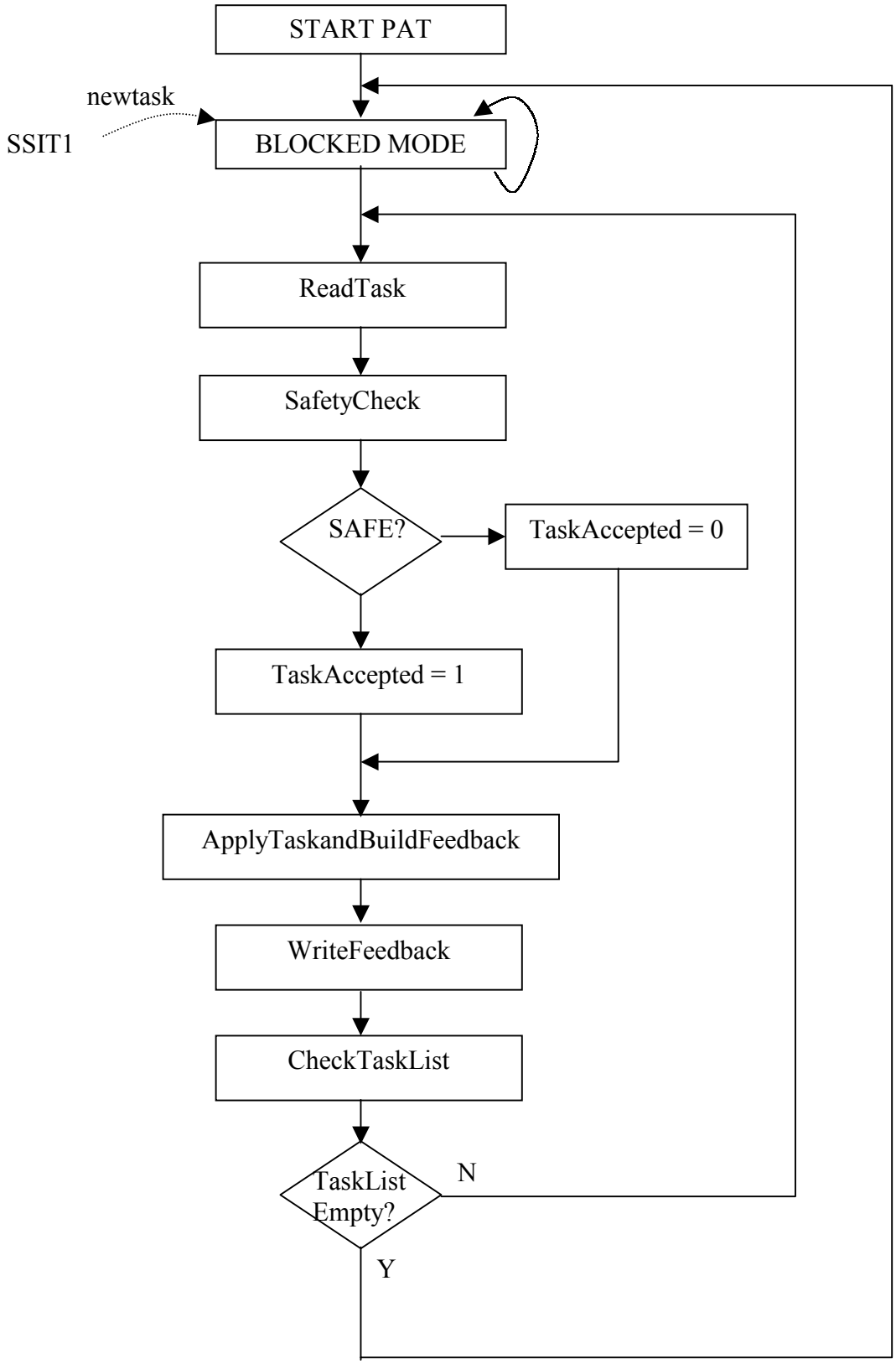


Fig. 2.3.1 Software Structure of SetupPC

Once a Task object is received by a SetupPC, first it must be saved in the TaskList object array. This is done by SSIT1. SSIT1 receives Task object from ServerPC through SK1 and writes it to TaskList. The main thread in SetupPC is PAT, which controls the plant. PAT reads this Task object into its own local variable, processes it, constructs Feedback object and writes it to FeedbackList. This Feedback object must be sent back to ServerPC and this is done by SSIT2. SSIT2 reads Feedback object from FeedbackList into its own local variable and sends it to ServerPC through SK4. SK2 is used to inform ServerPC whether SSIT1 is ready to receive next task object or busy with writing the last received Task object to TaskList. Similarly, SK3 is used to learn whether the ServerPC is ready to receive next Feedback object. The flow chart of each thread is explained next.

PAT:


START PAT
newtask
SSIT1
BLOCKED MODE
ReadTask
SafetyCheck
SAFE?
TaskAccepted = 0
TaskAccepted = 1
ApplyTaskandBuildFeedback
WriteFeedback
CheckTaskList
TaskList Empty?
N
Y

ReadTask :

PAT first locks the TaskList object in order to prevent it to be modified by other threads. Then, PAT copies the next Task object in the TaskList into its own local Task variable.

SafetyCheck :

Each cmd object of the Task object is checked whether it is safe or dangerous for the plant. For example, for the PA-10 robot move_to_XYZ cmds that have Z parameters less than certain value are immediately rejected since it will crash the robot to the ground. Once an unsafe cmd is found, this function returns with negative return value to inform the calling function that the requested Task object is not accepted.

ApplyTaskandBuildFeedback :

Each cmd object is executed and corresponding fb object is recorded. For example, for PA-10 robot if the next cmd object is move_to_XYZ(X_r , Y_r, Z_r), the robot is moved to < X_r , Y_r , Z_r > coordinates and the resulting actual < X , Y , Z > coordinates are recorded in the XYZ(X , Y, Z) fb object. After all of the cmd objects are executed and corresponding fb objects are recorded, a Feedback object is built.

WriteFeedback :

The FeedbackList is locked in order to prevent it to be used by other threads. The built Feedback object is copied into the FeedbackList. At the end the FeedbackList is unlocked to allow other waiting threads to access it.

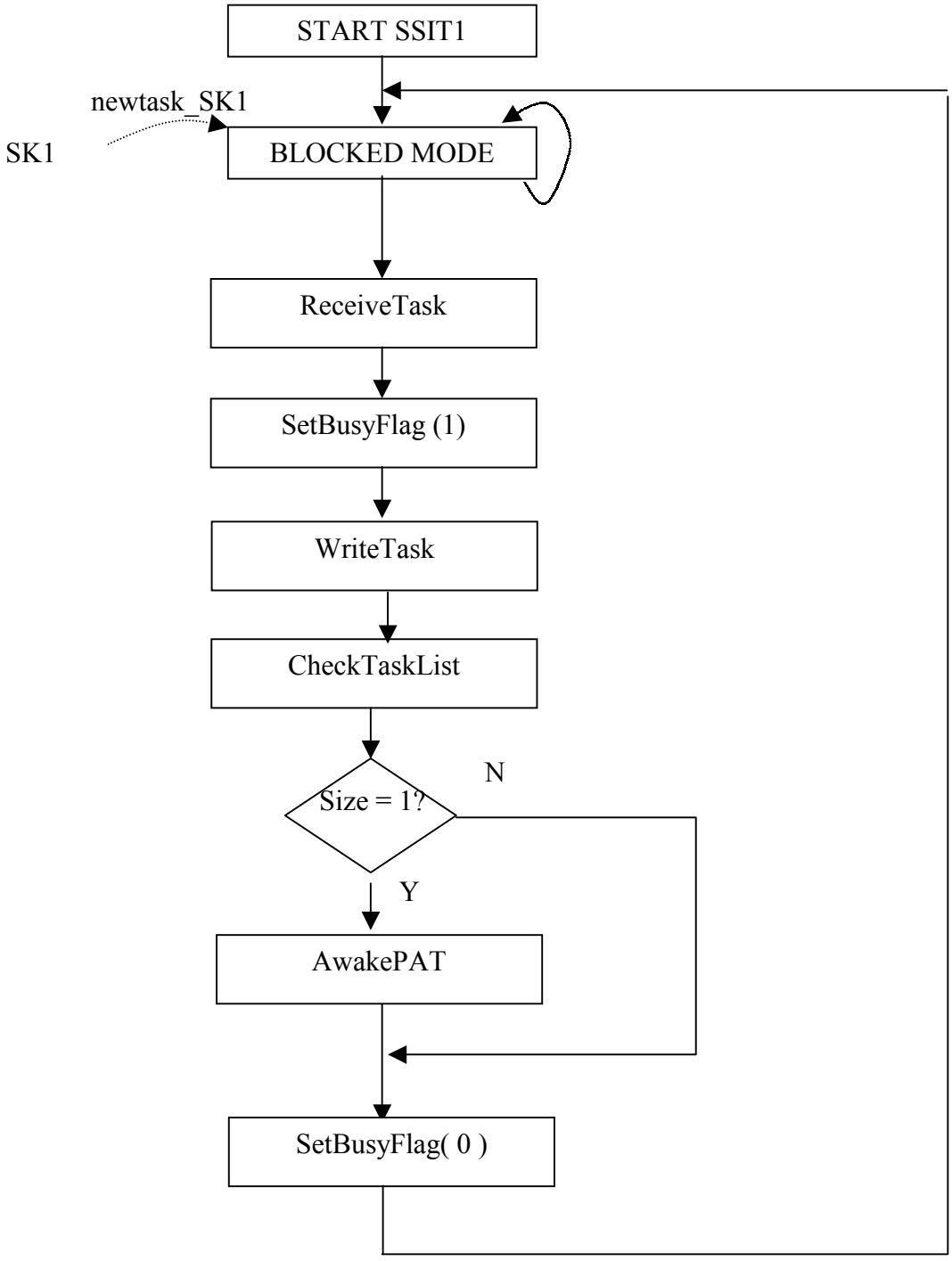
CheckTaskList :

The TaskList is checked whether it is empty or not. If the TaskList is empty and there are no other Task objects to be executed for the time, PAT enters into the blocked state in order not to waste CPU time. If the TaskList is not empty and there are Tasks to be executed, PAT goes to the ReadTask step.

Blocked Mode :

Threads have three types of modes : Stopped, running and blocked.
Blocked mode is the mode entered when the thread has nothing to do in order to save processor time. PAT quits from blocked mode when it receives newtask event message from SSIT1.

SSIT1 :


START SSIT1
newtask_SK1
SK1
BLOCKED MODE
ReceiveTask
SetBusyFlag (1)
WriteTask
CheckTaskList
Size = 1?
N
Y
AwakePAT
SetBusyFlag(0)

ReceiveTask :

SetupPC receives Task object from ServerPC through TCP/IP socket SK1. SK1 sets event newtask to inform SSIT1 that new task has arrived. SSIT1 reads this Task object into its own local variable.

SetBusyFlag :

In order to prevent ServerPC to send another task object while SSIT1 is writing last received Task into TaskList, SSIT1 sends ServerPC a short message. If the message is 1 then it means that SSIT1 is busy with writing the last received Task object into the TaskList. If the message is 0 then it means that SSIT1 is ready to receive the next Task object.

WriteTask :

SSIT1 writes the received Task object into the TaskList.

CheckTaskList :

After writing the received Task object into the TaskList SSIT1 checks whether the TaskList was empty or not.

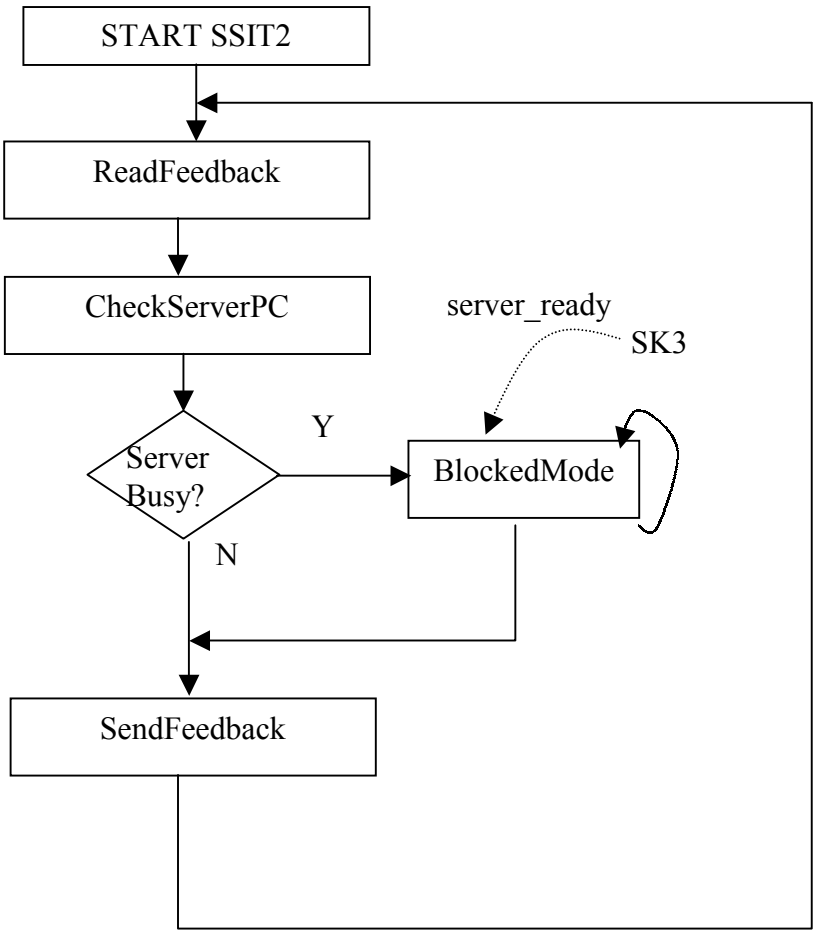
AwakePAT :

If TaskList was empty this means that PAT is in blocked mode so SSIT1 sends a message by setting the newtask event in order to release PAT from blocked mode.

BlockedMode :

In order to save CPU time SSIT1 enters into the blocked mode until next Task is sent by the ServerPC. When TCP/IP socket SK1 receives the next Task it send a message by setting the newtask_SK1 event in order to release SSIT1 from blocked mode.

SSIT2 :



ReadFeedback :

SSIT2 copies next Feedback object, constructed by PAT, from the FeedbackList into its own local variable.

CheckServerPC :

SSIT2 checks whether the ServerPC is ready to receive the Feedback object or busy, by checking a local variable which stores the status of the ServerPC.

SendFeedback :

If the ServerPC is ready to receive the Feedback object, SSIT2 sends the Feedback through TCP/IP socket SK4 and goes to ReadFeedback step.

BlockedMode :

If ServerPC is not ready then SSIT2 enters into the blocked mode until a message comes from the ServerPC through SK3. This short message releases SSIT2 from blocked mode and allows it to send the Feedback object to ServerPC.

2.3.2 ServerPC

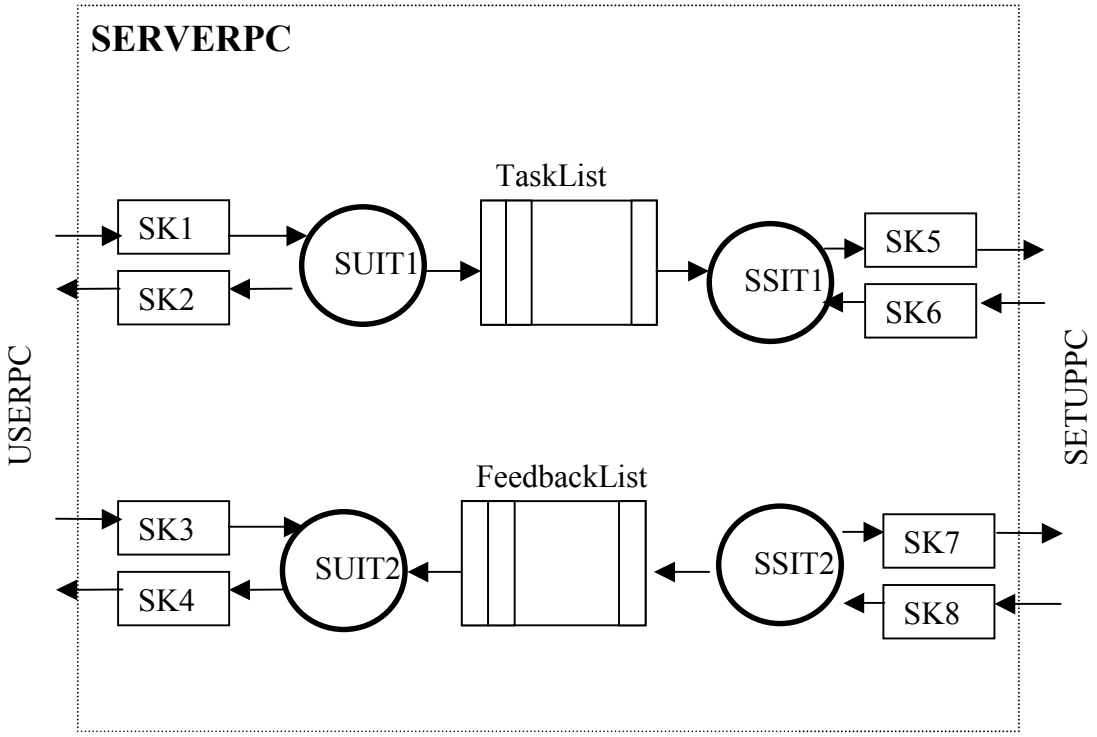
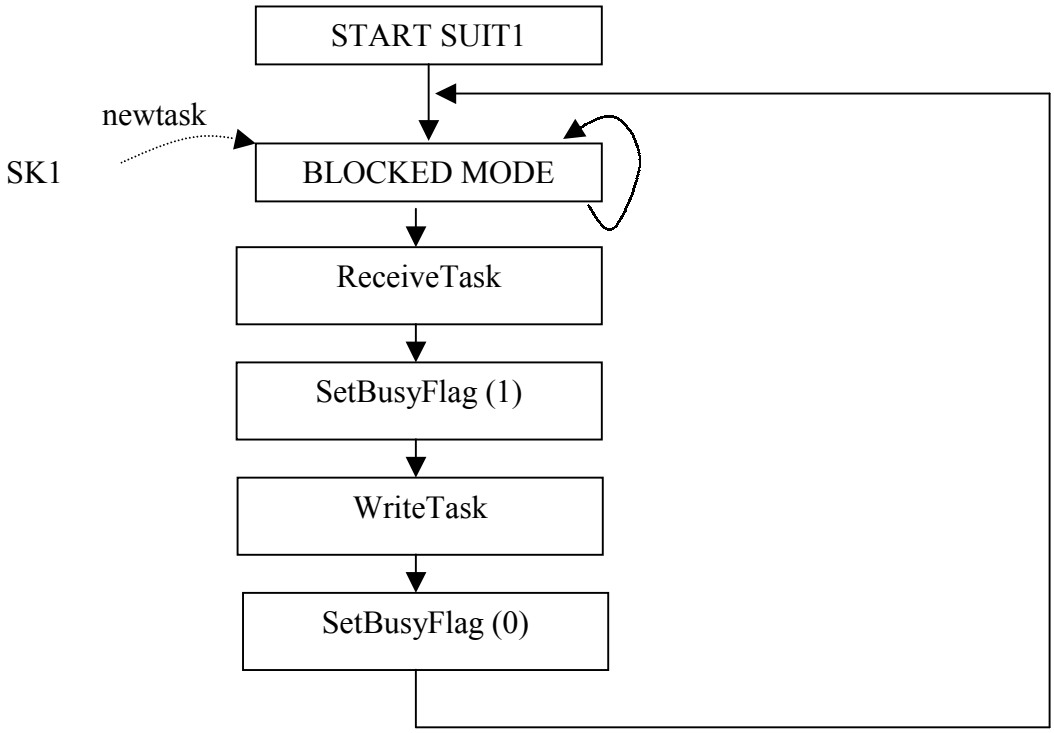


Fig. 2.3.2 Software Structure of ServerPC

ServerPC is the bridge between UserPC and SetupPC. Every Task object sent from a UserPC to a SetupPC and every Feedback object sent from a SetupPC to a UserPC must pass through ServerPC. Once a Task object is received by the ServerPC, first it must be saved in the TaskList object array. This is done by SUIT1. SUIT1 receives Task object from UserPC through TCP/IP socket SK1 and writes it to TaskList. Then, SSIT1 reads this Task object into its own local variable and sends it to the corresponding SetupPC through SK5. Processing of Feedback object is similar. SSIT2 receives a Feedback object from a SetupPC and writes it into FeedbackList. SUIT2 reads the Feedback object from FeedbackList and sends it to corresponding UserPC. SK2, SK3, SK6 and SK7 are used by threads in separate PCs whether they are busy or ready to receive objects. The flow chart of each thread is explained next.

SUIT1 :



ReceiveTask :

ServerPC receives Task object from UserPC through TCP/IP socket SK1. SK1 sets event newtask to inform SUIT1 that new task has arrived. SUIT1 reads this Task object into its own local variable.

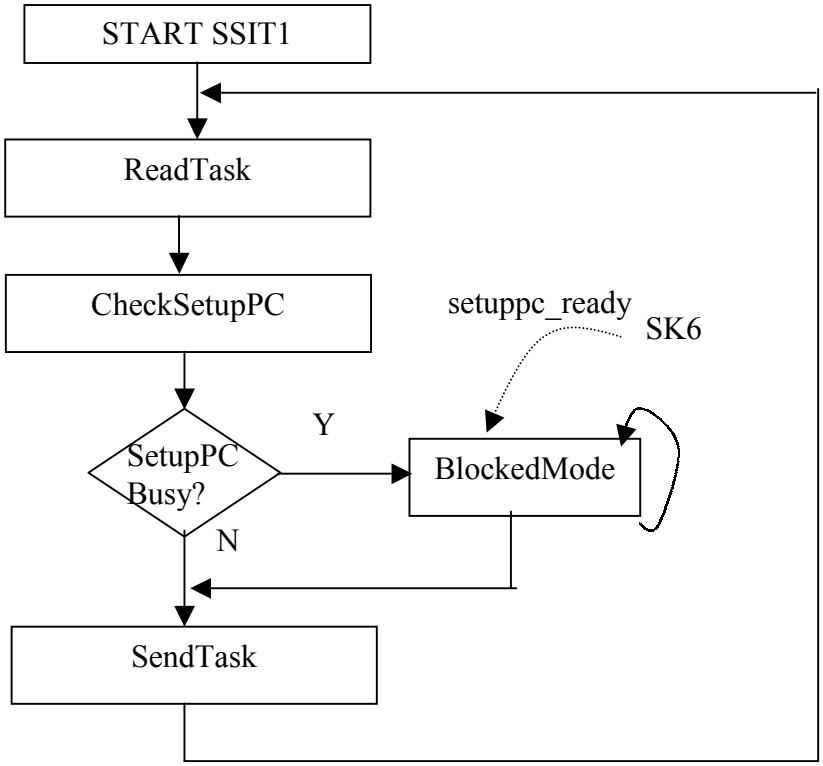
SetBusyFlag :

In order to prevent UserPC to send another task object while SUIT1 is writing last received Task into TaskList, SUIT1 sends UserPC a short message. If the message is 1 then it means that SUIT1 is busy with writing the last received Task object into the TaskList. If the message is 0 then it means that SUIT1 is ready to receive the next Task object.

WriteTask :

SUIT1 writes the received Task object into the TaskList.

SSIT1 :



ReadTask :

SSIT1 copies next Task object from the TaskList into its own local variable.

CheckSetupPC :

SSIT1 checks whether the SetupPC is ready to receive the Task object or busy, by checking a local variable which stores the status of the SetupPC.

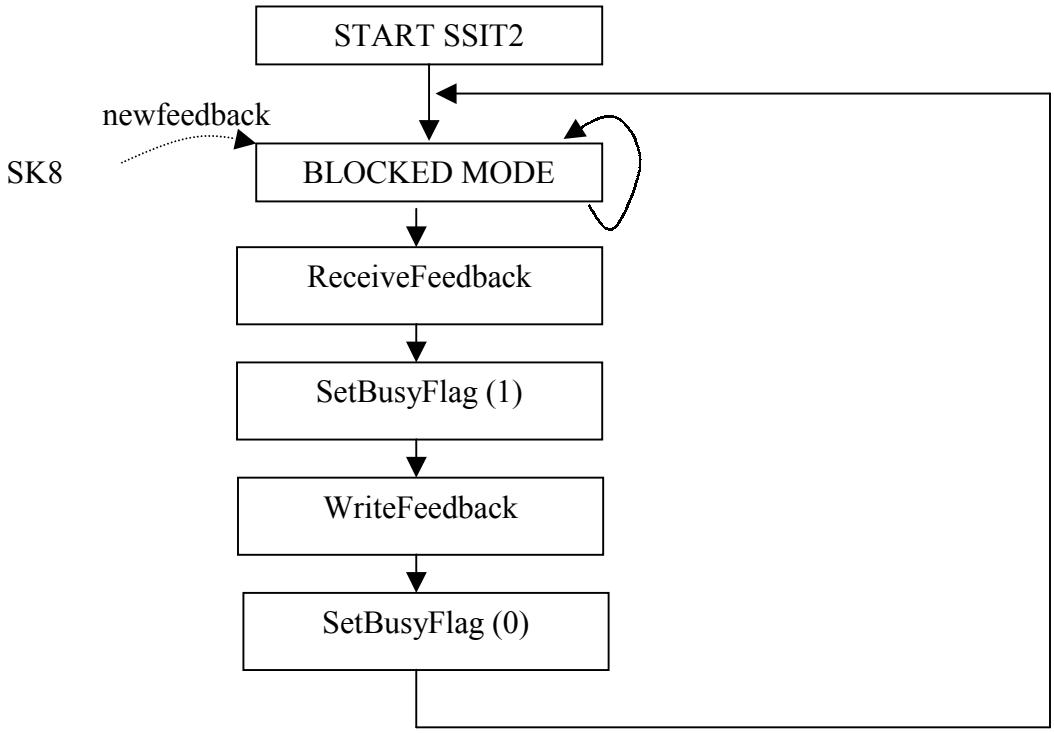
SendTask :

If the SetupPC is ready to receive the Task object, SSIT1 sends the Task through TCP/IP socket SK5 and goes to ReadTask step.

BlockedMode :

If SetupPC is not ready then SSIT1 enters into the blocked mode until a message comes from the SetupPC through SK6. This short message releases SSIT1 from blocked mode and allows it to send the Task object to SetupPC.

SSIT2 :



ReceiveFeedback :

ServerPC receives Feedback object from SetupPC through TCP/IP socket SK8. SK8 sets event newfeedback to inform SSIT2 that new feedback has arrived. SSIT2 reads this Feedback object into its own local variable.

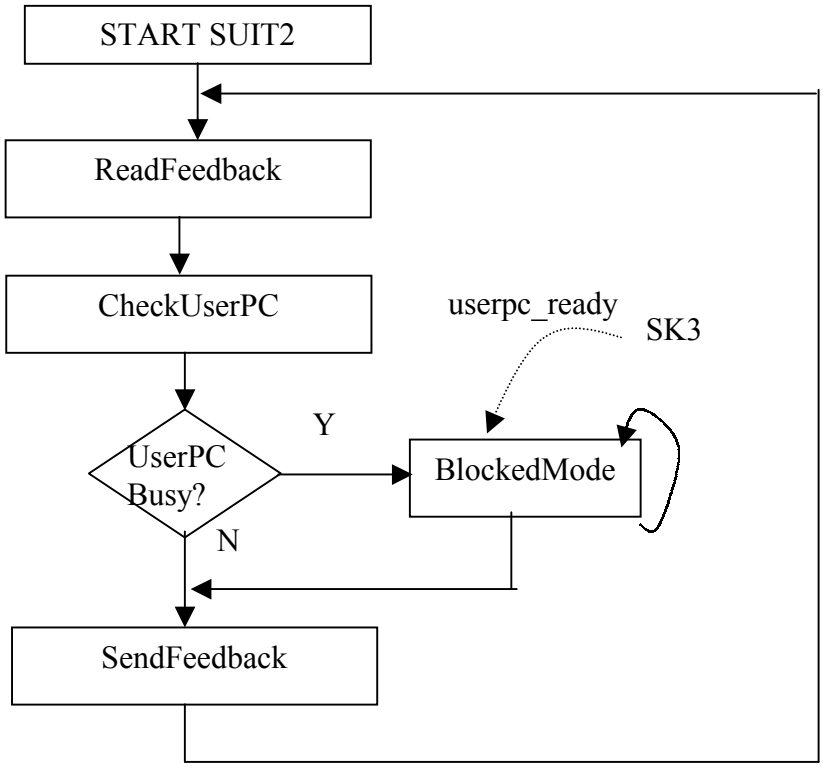
SetBusyFlag :

In order to prevent SetupPC to send another feedback object while SSIT2 is writing last received Feedback into FeedbackList, SSIT2 sends SetupPC a short message. If the message is 1 then it means that SSIT2 is busy with writing the last received Feedback object into the FeedbackList. If the message is 0 then it means that SSIT2 is ready to receive the next Feedback object.

WriteTask :

SSIT2 writes the received Feedback object into the FeedbackList.

SUIT2 :



ReadFeedback :

SUIT2 copies next Feedback object from the FeedbackList into its own local variable.

CheckUserPC :

SUIT2 checks whether the UserPC is ready to receive the Feedback object or busy, by checking a local variable which stores the status of the UserPC.

SendFeedback :

If the UserPC is ready to receive the Feedback object, SUIT2 sends the Feedback through TCP/IP socket SK4 and goes to ReadFeedback step.

BlockedMode :

If UserPC is not ready then SUIT2 enters into the blocked mode until a message comes from the UserPC through SK3. This short message releases SUIT2 from blocked mode and allows it to send the Feedback object to UserPC.

2.3.3 UserPC

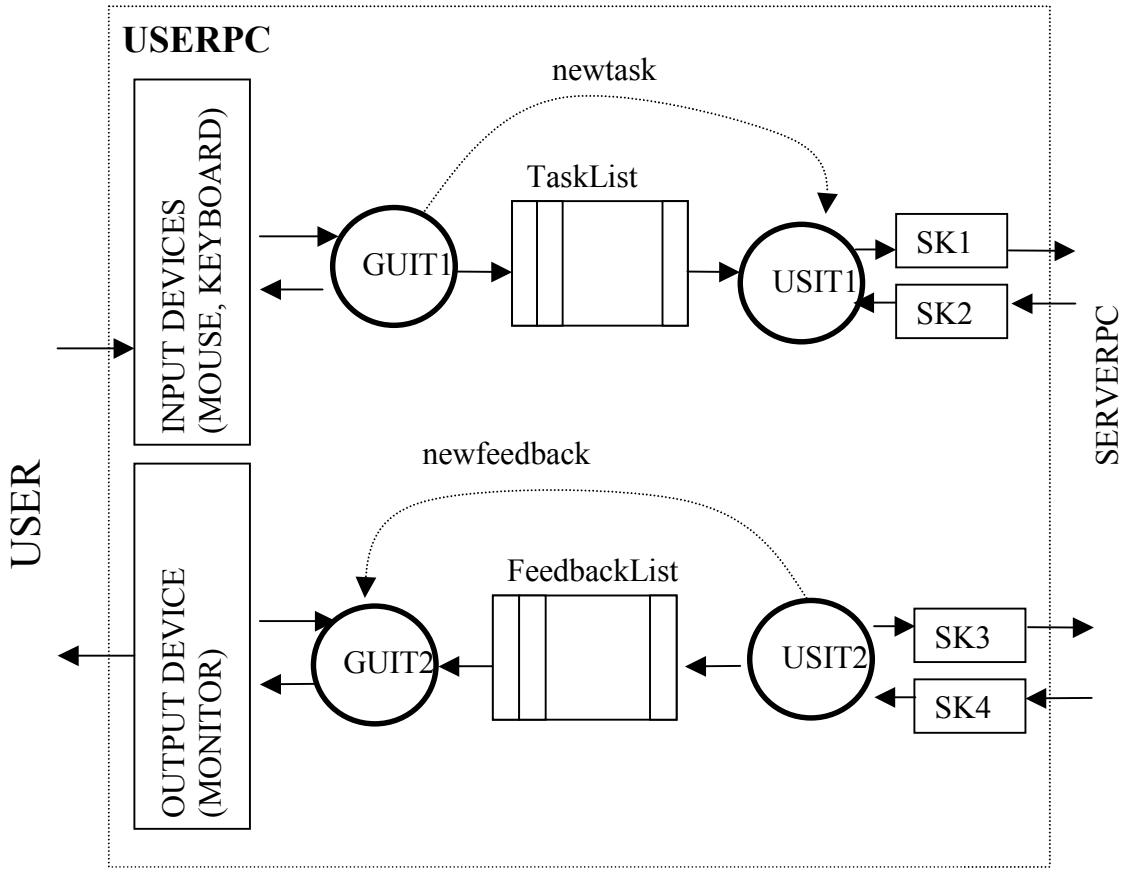
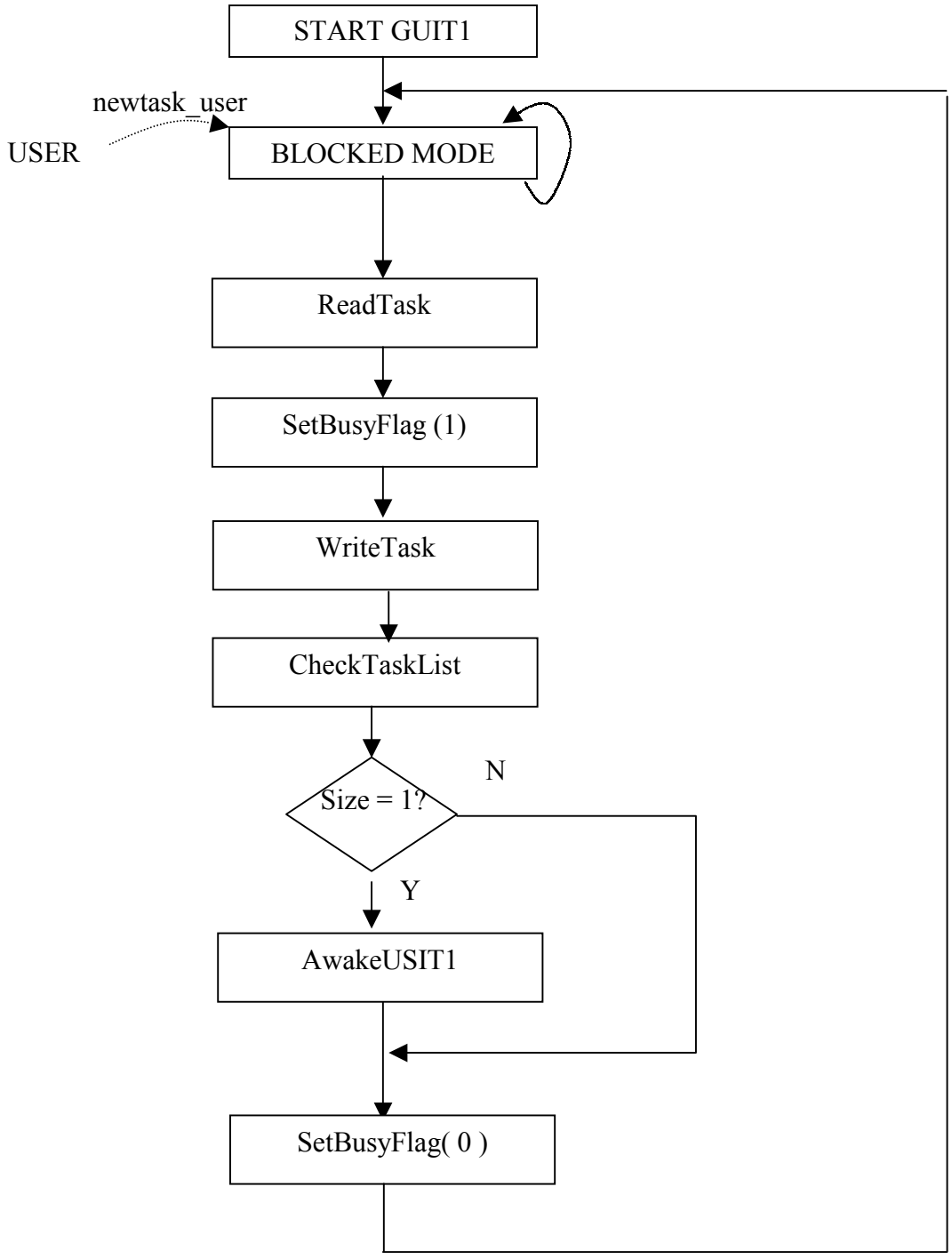


Fig. 2.3.3 Software Structure of UserPC

UserPC is the device that allows remote users conduct their experiments. User commands are generated using the input devices, keyboard and mouse and feedback to the user is shown by the output device, monitor. Every Task object is built and every Feedback object is terminated in UserPC. Once user decides that the Task object he constructed is the desired one he asks the client software to send it by clicking the SendTask button on the GUI window. GUIT1 writes the Task object to the TaskList and informs USIT1 that there is a new Task object by sending an event message, newtask. USIT1 reads the Task object from TaskList into its own local variable and sends it to ServerPC through TCP/IP socket SK1. USIT2 is the thread to deal with a received Feedback object. Once USIT2 receives a Feedback object from ServerPC through SK4 it writes it to the FeedbackList and informs GUIT2 that there is a new Feedback object arrived. GUIT2 reads this Feedback object from FeedbackList and shows it to the user to complete the whole remote control loop. The flow chart of each thread is explained next.

GUIT1 :


START GUIT1
newtask_user
USER
BLOCKED MODE
ReadTask
SetBusyFlag (1)
WriteTask
CheckTaskList
Size = 1?
N
Y
AwakeUSIT1
SetBusyFlag(0)

ReadTask :

With the newtask_user event message sent by user clicking the “SendTask” button on the GUI, GUIT1 copies the Task object saved in a temporary variable into its own local variable.

SetBusyFlag :

In order to prevent user to send another task object while GUIT1 is writing last Task into TaskList, GUIT1 sends user a message. If the message is 1 then it means that GUIT1 is busy with writing the last Task object into the TaskList. If the message is 0 then it means that GUIT1 is ready to read the next Task object. This message is shown on the GUI window informing user whether he can send the next Task he has built.

WriteTask :

GUIT1 writes the Task object into the TaskList.

CheckTaskList :

After writing the Task object into the TaskList GUIT1 checks whether the TaskList was empty or not.

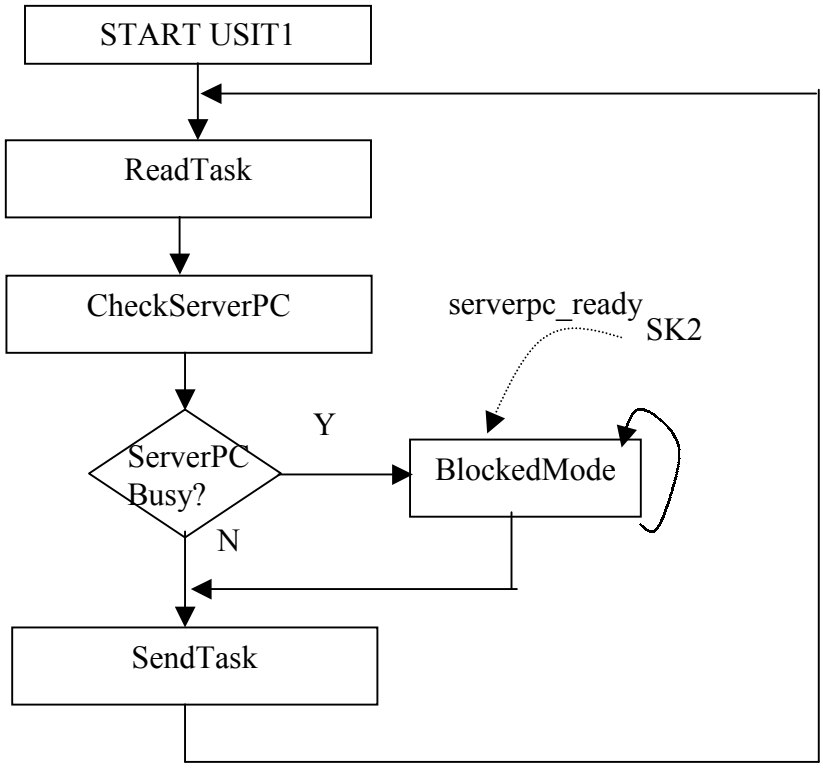
AwakeUSIT1 :

If TaskList was empty this means that USIT1 is in blocked mode so GUIT1 sends a message by setting the newtask event in order to release USIT1 from blocked mode.

BlockedMode :

In order to save CPU time GUIT1 enters into the blocked mode until next Task is built by the user. When user builts next Task he sends a message by setting the newtask_user event in order to release GUIT1 from blocked mode.

USIT1 :



ReadTask :

USIT1 copies next Task object from the TaskList into its own local variable.

CheckServerPC :

USIT1 checks whether the ServerPC is ready to receive the Task object or busy, by checking a local variable which stores the status of the ServerPC.

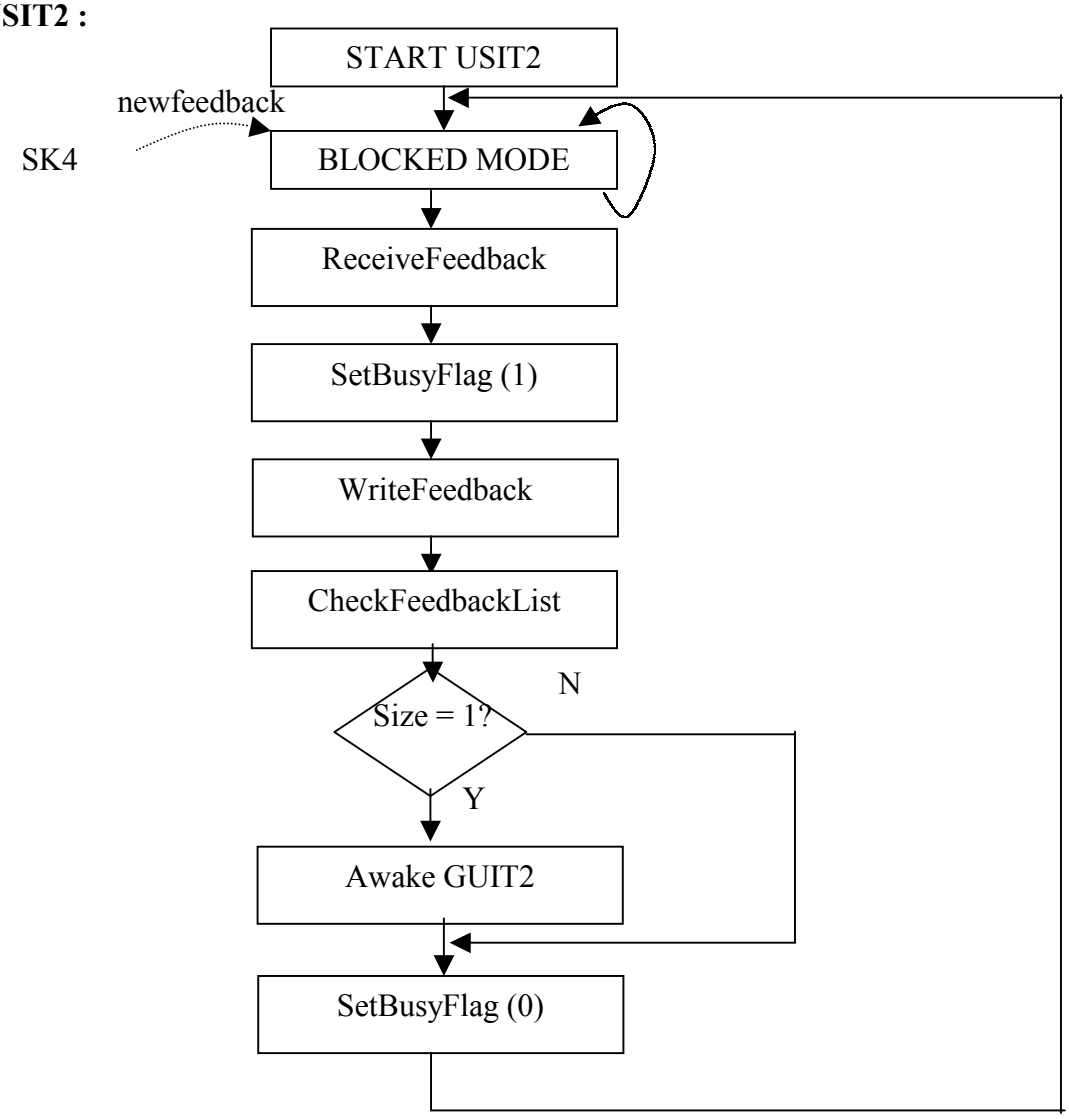
SendTask :

If the ServerPC is ready to receive the Task object, USIT1 sends the Task through TCP/IP socket SK1 and goes to ReadTask step.

BlockedMode :

If ServerPC is not ready then USIT1 enters into the blocked mode until a message comes from the ServerPC through SK2. This short message releases USIT1 from blocked mode and allows it to send the Task object to ServerPC.

USIT2 :



ReceiveFeedback :

USIT2 receives Feedback object from ServerPC through TCP/IP socket SK4.

SetBusyFlag :

USIT2 sends ServerPC a short message through SK3 to inform that it is busy.

WriteFeedback :

USIT2 writes the received Feedback object into the FeedbackList.

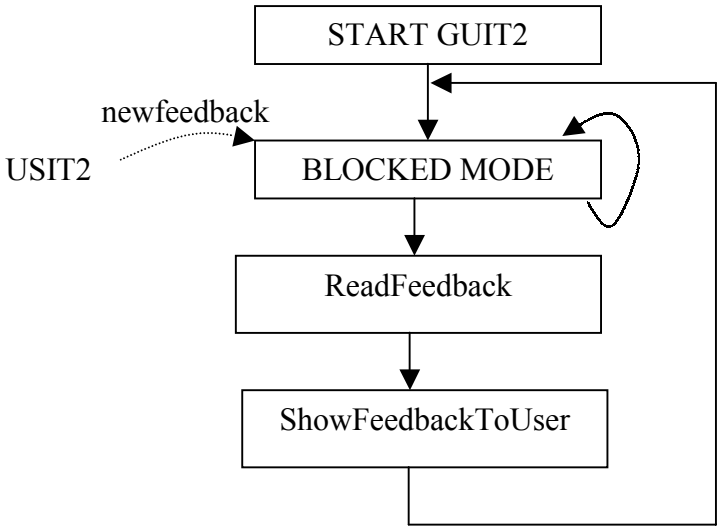
CheckFeedbackList :

USIT2 checks whether the size of the FeedbackList is one.

AwakeGUIT2 :

If the size of the FeedbackList is one then it means that GUIT2 is in blocked mode. Then USIT2 sends an event message to GUIT2 in order to release it from the blocked mode.

GUIT2 :



ReadFeedback :

GUIT2 reads a Feedback object from FeedbackList into its own local variable.

ShowFeedbackToUser :

GUIT2 shows the Feedback object fb-by-fb to user on the GUI window. This completes the remote control and experimentation loop.

BlockedMode :

If there no more Feedback object in the FeedbackList, GUIT2 enters the blocked mode in order to save CPU time. When new feedback object arrives, USIT2 releases GUIT2 from blocked mode by sending an event message, newfeedback.

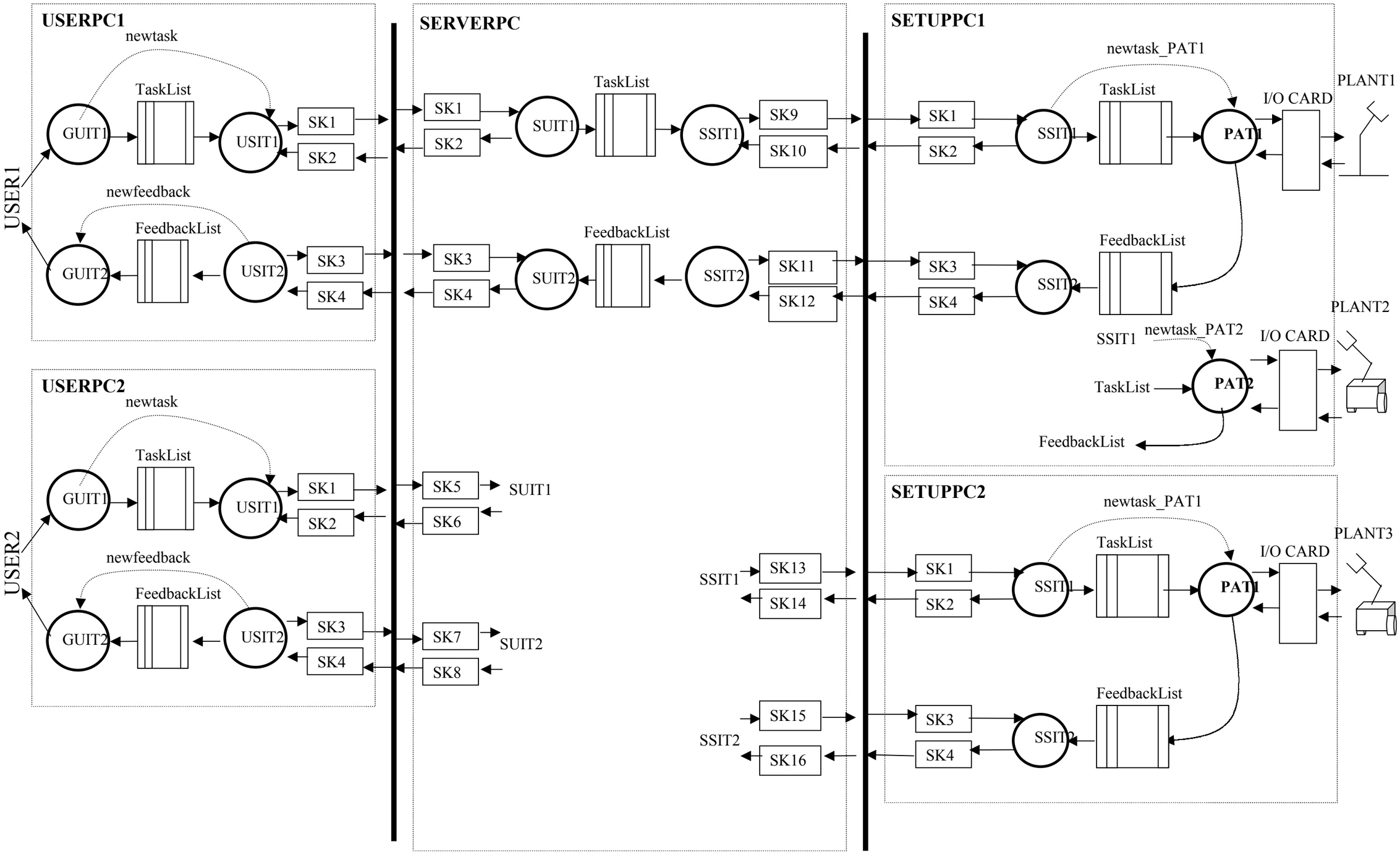


Figure 2.3.4 Software structure of whole system for two-user – three-plant scheme

Software structure of whole system for two-user – three-plant scheme is shown in the figure above. Note that two plants are connected to the first SetupPC. Addition of an extra plant to a SetupPC involves initialisation of one more PAT. Number of plants one SetupPC can control depend on the processing power of the SetupPC and the complexity of the cmds for the plants. The advantage of the proposed structure may not be seen for low number of expensive plants, like robots, since the price of PC may be ignored compared with the price of robot. But if the price of plant is low comparable with the price of a PC, like a DC motor control experiment setup or a relatively cheap mobile robot, then the proposed structure is very effective and efficient.

CHAPTER 3
PLANTS

Definition of the communication objects as cmd and fb gives flexibility enabling connection of various plants to the system. In this study two different kinds of plants are connected to the system. One is “DC Motor Control Setup” which is relatively simple and constitutes a good experiment for especially undergraduate mechatronics and control engineering students. Second is a robot arm which constitutes a very important part of mechatronics field. In this chapter these plants are described in detail.

3.1 DC Motor Control Experiment

As an introductory experiment for mechatronics engineering education, DC Motor Control Experiment is chosen. There are several reasons for this. First, DC motor control setup involves all components of a mechatronics product, mechanical, electrical, control and software. Second, DC motor is linear in certain region which means classical control techniques can be applied. Third, it is widely used in industry and robotics which means it is also a good investment to learn.

The experiment consists of four main parts. First part is learning about the basics of components of the experiment, mechanical, electrical and software components. Second part is mathematical modelling of the DC motor. Some parameters are given to the student so that the mathematical model of the model can be derived numerically. Third part is open-loop control of the DC motor. Student designs an open-loop proportional controller first without friction compensator, then with friction compensator so that the actual speed of the motor is equal to the desired speed. Then the student sets the desired speed and obtains the step response of the motor. He/she must see that the controller with friction compensator works well for step function. But when the desired velocity is

sinusoidal function the motor can not track the reference above some frequency. Fourth and last part is closed-loop P, PI, PD, PID control of DC motor. Student designs a PID controller to meet certain performance criteria. Then he/she sets the desired speed and obtains the step response of the motor. Closed-loop controller also works well for step function. When the reference is sinusoidal the closed-loop controller achieves better response than the open-loop controller. The block diagram representation of the setup is shown in Fig3.1.

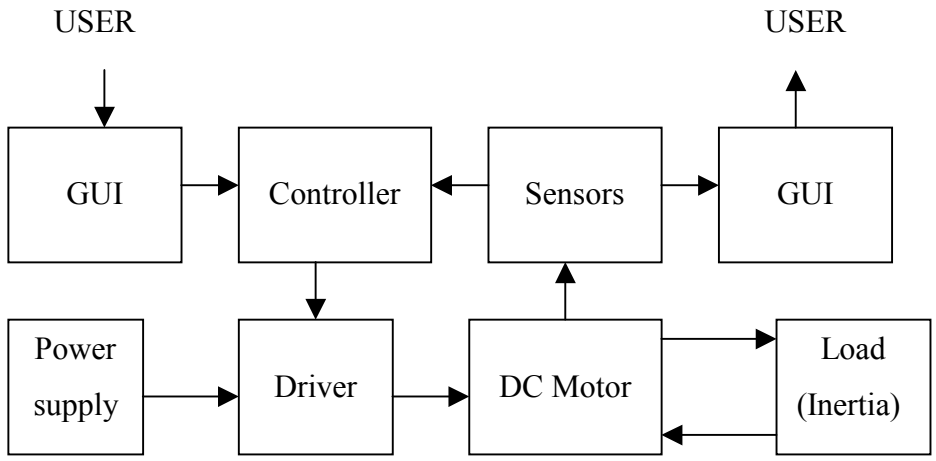


Fig3.1 : Functional block diagram representation of DC motor control setup

Power supply is adjustable DC power supply which can give voltage between 0-32 volts. Driver is H-bridge in order to have bi-directional drive of DC motor. DC motor is permanent magnet brush DC motor with maximum voltage of 24V. There are two sensors : a simple resistor as current sensor and a dual channel optical encoder with 3600ppr as velocity sensor. Controller is one of open-loop proportional controller with/without friction compensator, closed-loop P,PI,PD,PID controller. GUI enables the user to adjust the controller parameters and shows the response of the system as feedback.

In order to conduct the experiment student has to have two kind of interaction with the setup : he/she must be able to set the controller parameters, start and stop the experiment at any time he/she wants and he/she must be able to see the experimental result, that is the plot of the response of the motor. So, the cmd and fb objects can be defined as follows:

cmdname	arg	fbname	arg
set controller type	0, 1, 2	controller type	0, 1, 2
set reference type	0, 1	reference type	0, 1
set kp	kp	current kp	kp
set ki	ki	current ki	ki
set kd	kd	current kd	kd
set kfr	kfr	current kfr	kfr
set amplitude	amplitude	current amplitude	amplitude
set frequency	frequency	current frequency	frequency
start experiment	0	angular velocity	w

3.2 Mitsubishi PA-10 7-DOF Robot Manipulator

Among all products, robots are one of the best example to an advanced mechatronics system. First, they involve all parts of a mechatronics system to the full extend : complex mechanics, complex electronics, complex control and complex software. Second, they are used in various real world applications from welding, grinding to pick&place, painting. So, they constitute an important part of mechatronics field. Adding a good robot to remote mechatronics laboratory also makes sense since a good robot is quite expensive and so not affordable by every university. Mitsubishi PA-10 7-DOF Robot Arm is a well-designed, smart, general purpose robot with open architecture which is a good candidate for educational purposes. As an example robot control experiment independent joint PID control of PA-10 can be given. The cmd and fb objects are as follows.

cmdname	arg	fbname	arg
set reference type	0, 1	reference type	0, 1
set kp	kp1, kp2, …, kp7	current kps	kp1, …, kp7
set ki	ki1, …, ki7	current kis	ki1, …, ki7
set kd	kd1, …, kd7	current kds	kd1, …, kd7
add reference 1	x, y, z, yw, p, r	current pos & or	x,y,z,yw,p,r
add reference 2	theta vector	current thetas	theta vector
start experiment			

CHAPTER 4
EXPERIMENTAL RESULTS

As an experimental study of the remote laboratory concept, DC motor control experiment is chosen. The DC motor control experiment consists of the following parts and shown in Fig4.1:

- 24V Permanent Magnet DC motor
- 3600 ppr, 3 channel optical encoder
- L298n dual H-bridge motor driver
- 20V DC power supply
- Humusoft MF604 multi-function I/O card
- SetupPC :
    - CPU : Pentium 733MHz
    - RAM : 128 MB
    - Operating system : Windows 98
- Ethernet Card

UserPC is a laptop computer consisting of Celeron 500MHz CPU, 64 MB of RAM and an Ethernet Card.

In SetupPC a server software is running, listening to port 4000 for client requests. In UserPC, a client software with a simple GUI is running as shown in Fig4.2. User connects to the experiment by clicking the Experiment->Connect on the menu, Fig4.3. Server accepts the client request and establishes the connection. User sets the parameters, Kp, Ki, Kd of the PID controller by selecting the Task->Set Experiment Parameters, Fig4.4, Fig4.5. Then user sends the Task objects by selecting Task->Send

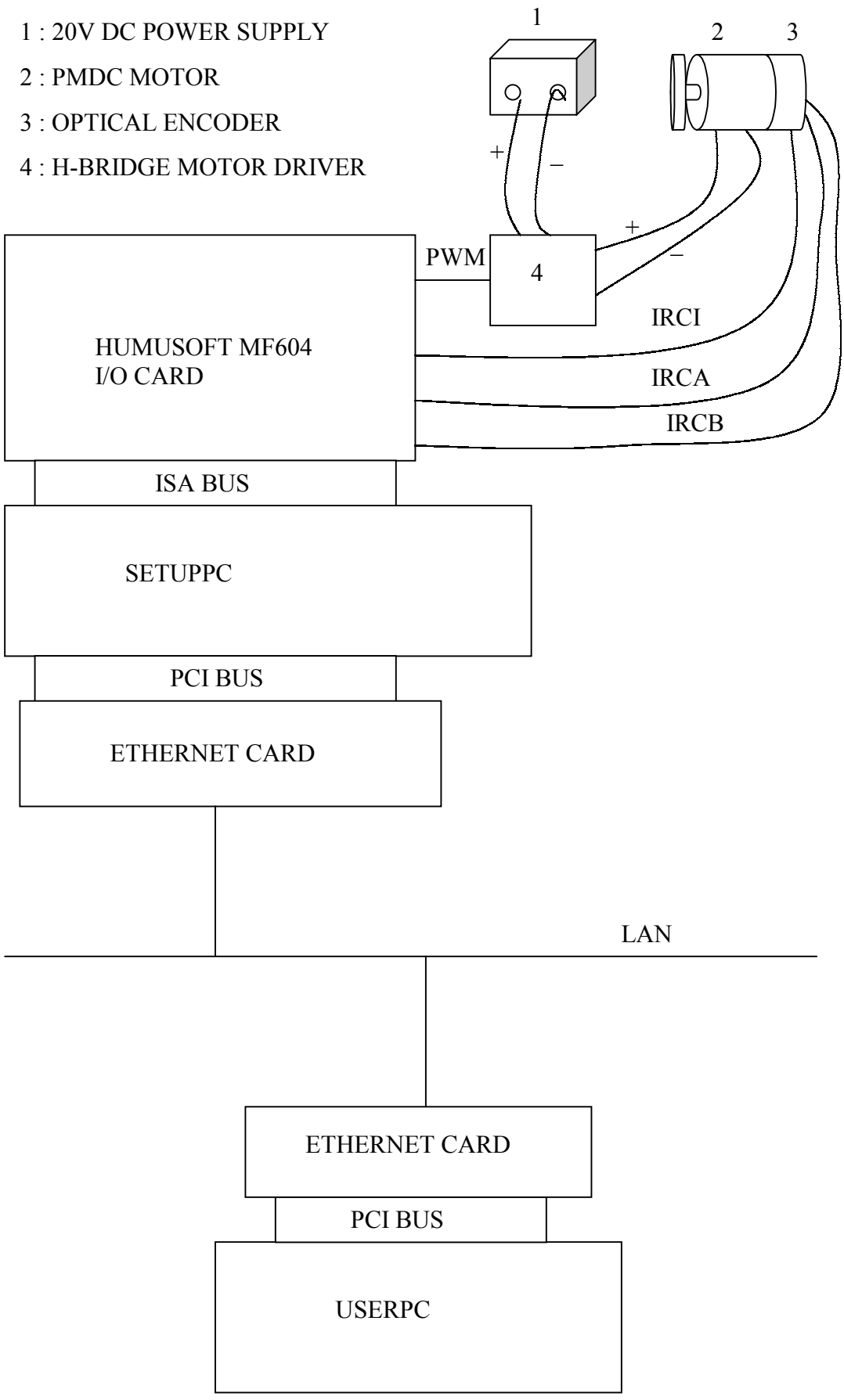


Fig4.1 : Permanent Magnet DC Motor Experimental Setup

Task, Fig4.6. SetupPC receives the Task object and runs the experiment. The velocity of the DC motor is recorded as feedback. When the experiment is finished the Feedback object is sent to the UserPC. When UserPC receives this Feedback object “FEEDBACK RECEIVED!!!!” message is shown to the user on the status read-only edit box. User clicks Feedback->Show Feedback on the menu to see the response curve of the DC motor. If he/she wants user can readjust the parameters of the controller. Different response curves for different PID parameters are shown in Fig4.7-Fig4.10.

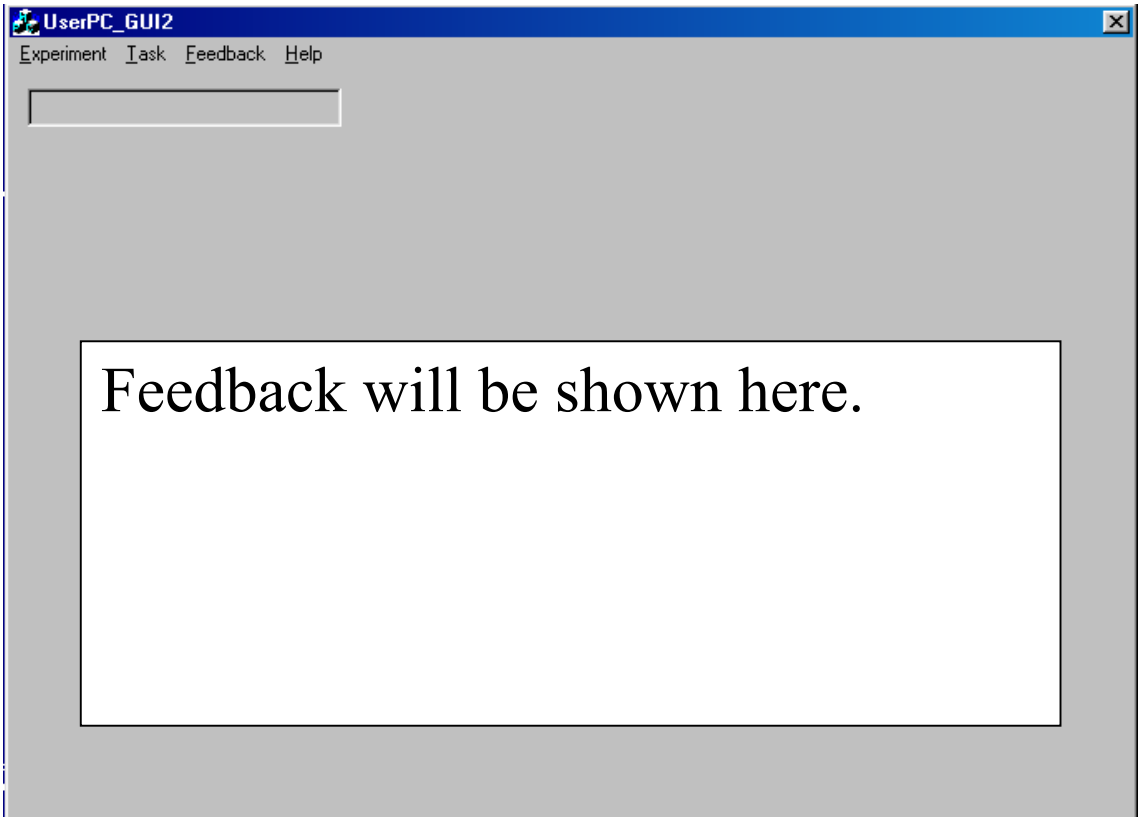


Fig4.2 : GUI of the Client software running on the UserPC

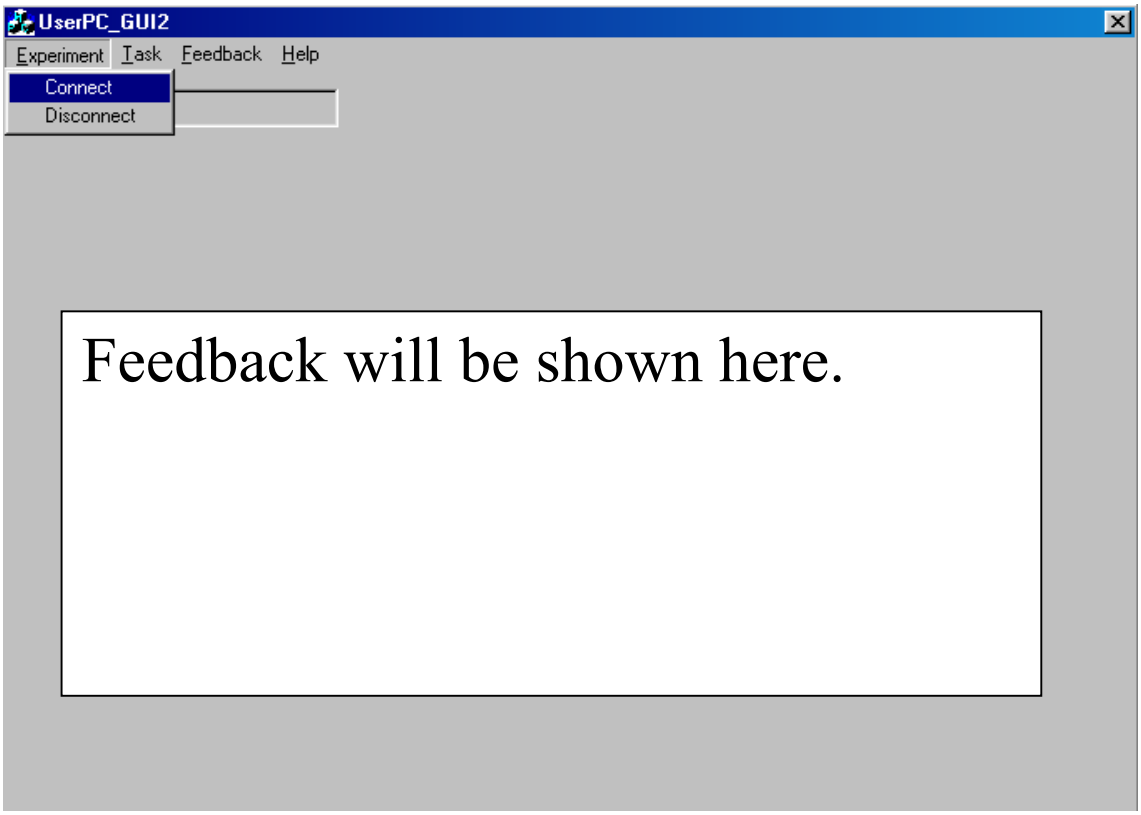


Fig4.3 : User connects to the remote experiment

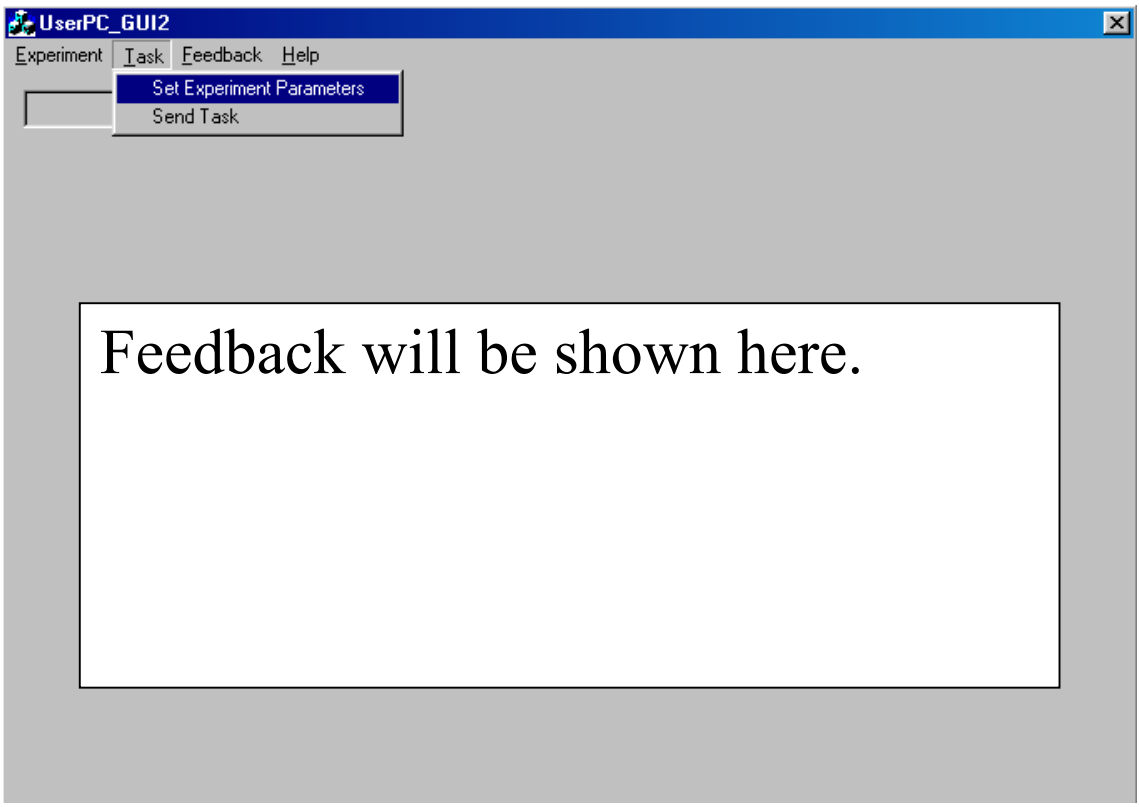


Fig4.4 : User clicks Task -> Set Experiment Parameters to adjust PID parameters

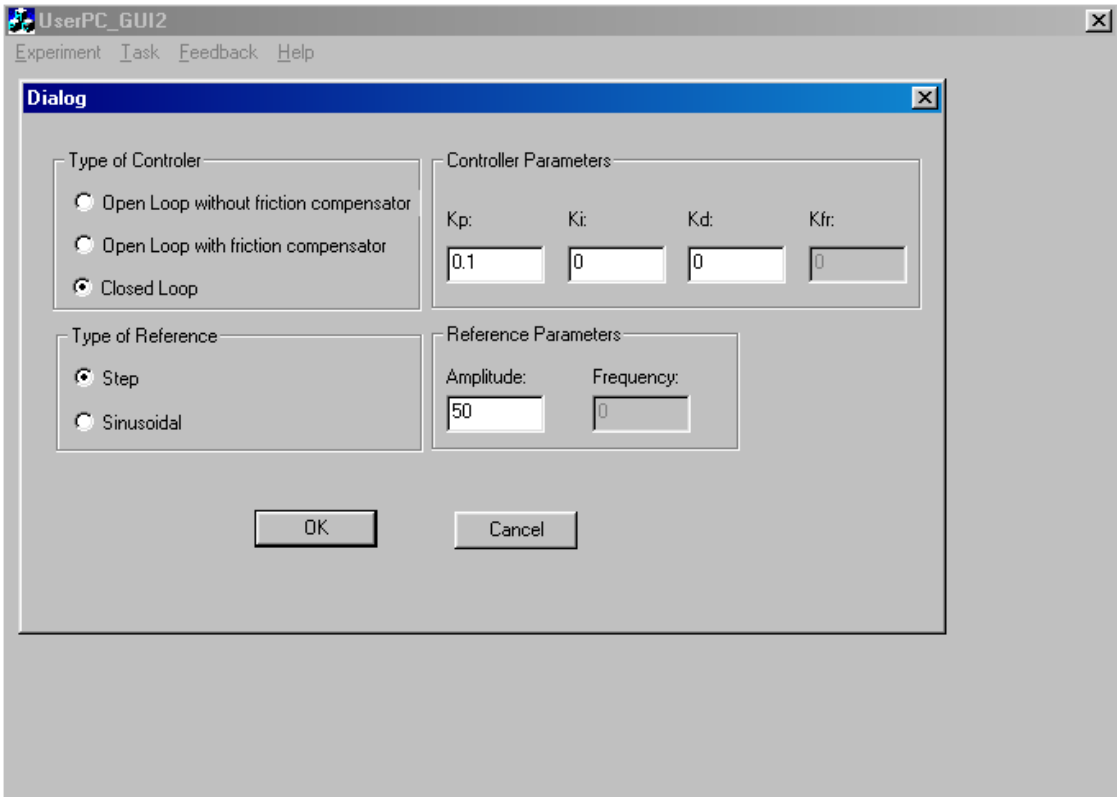


Fig4.5 : Experiment parameters dialog box

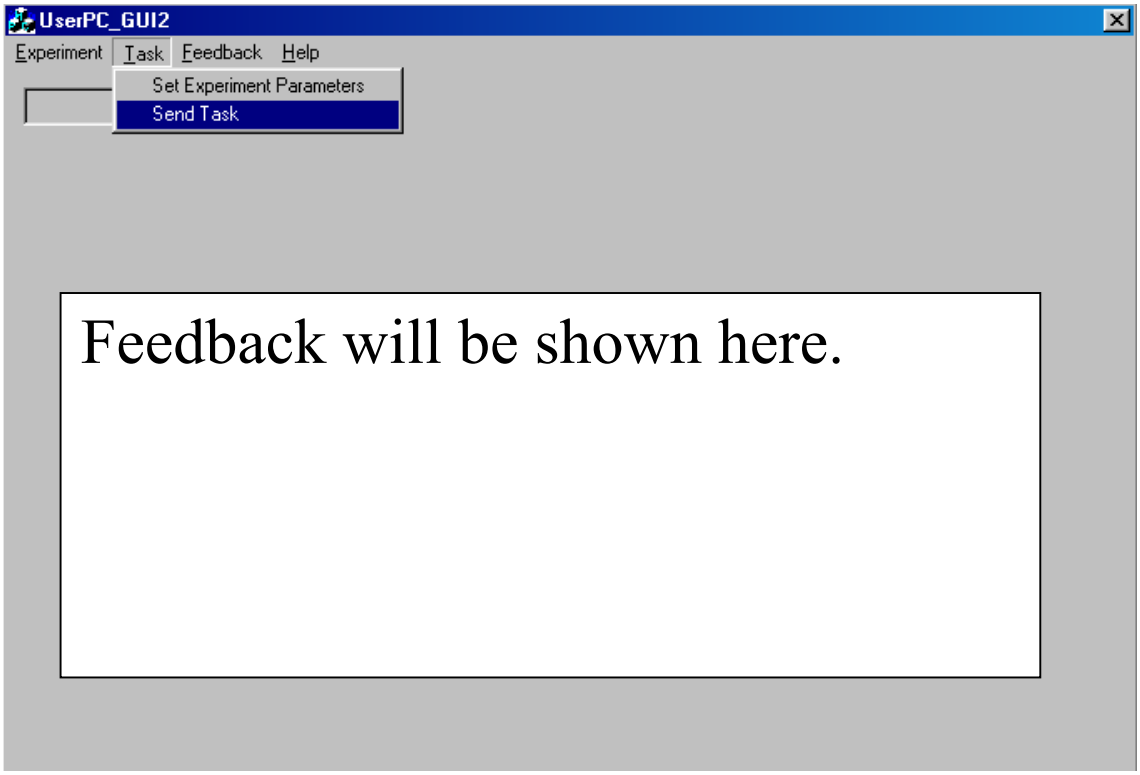


Fig4.6 : User clicks Task->Send Task in order to send the Task object

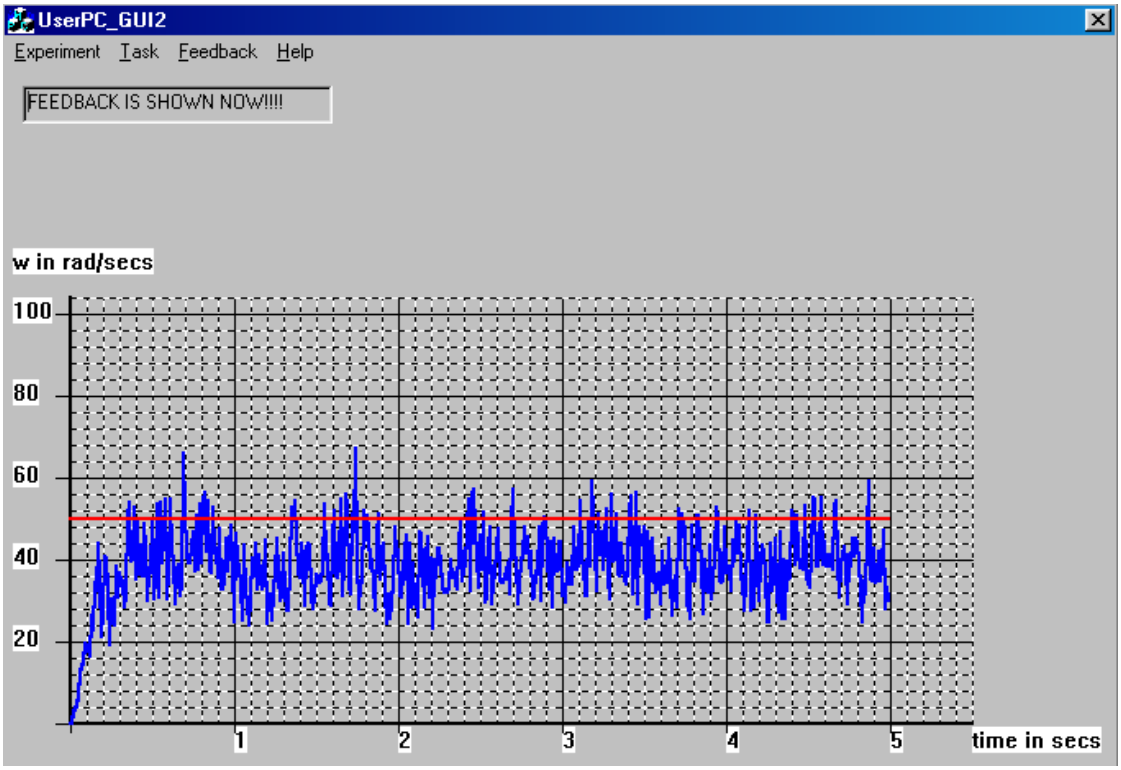


Fig4.7 : Response curve for Kp=0.1, Ki=0, Kd=0, wref = 50 rad/sec

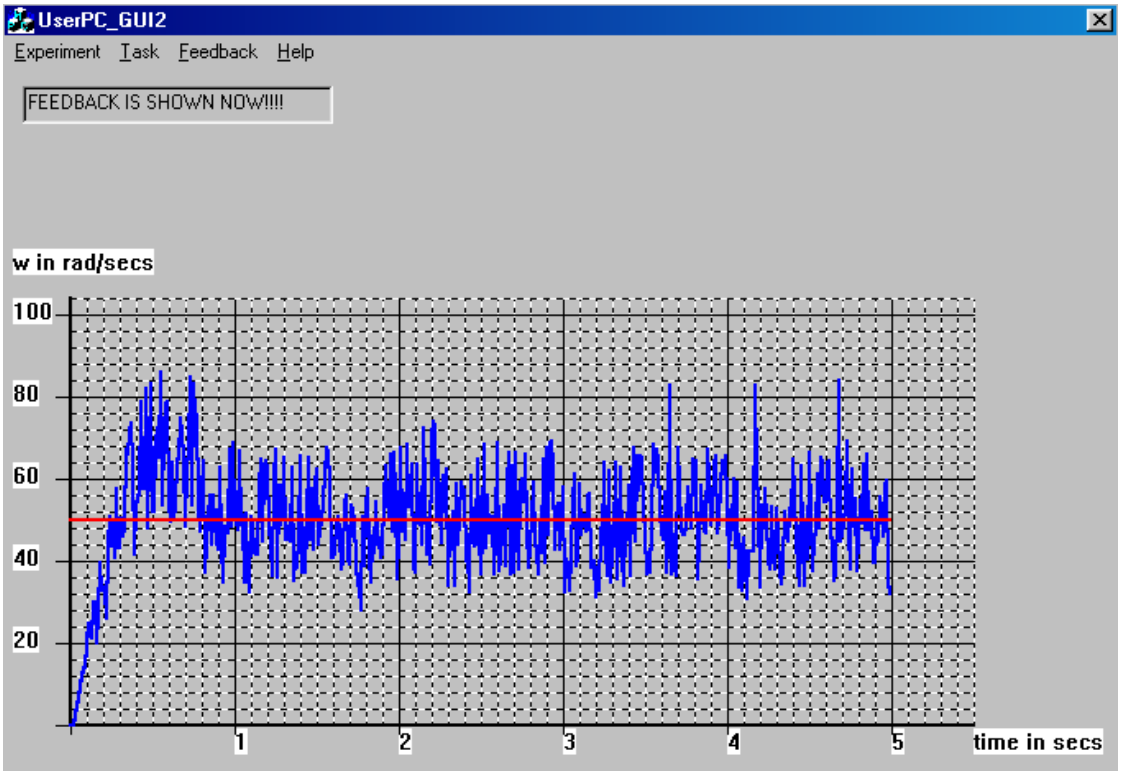


Fig4.8 : Response curve for Kp=0.1, Ki=1, Kd=0, wref = 50 rad/sec

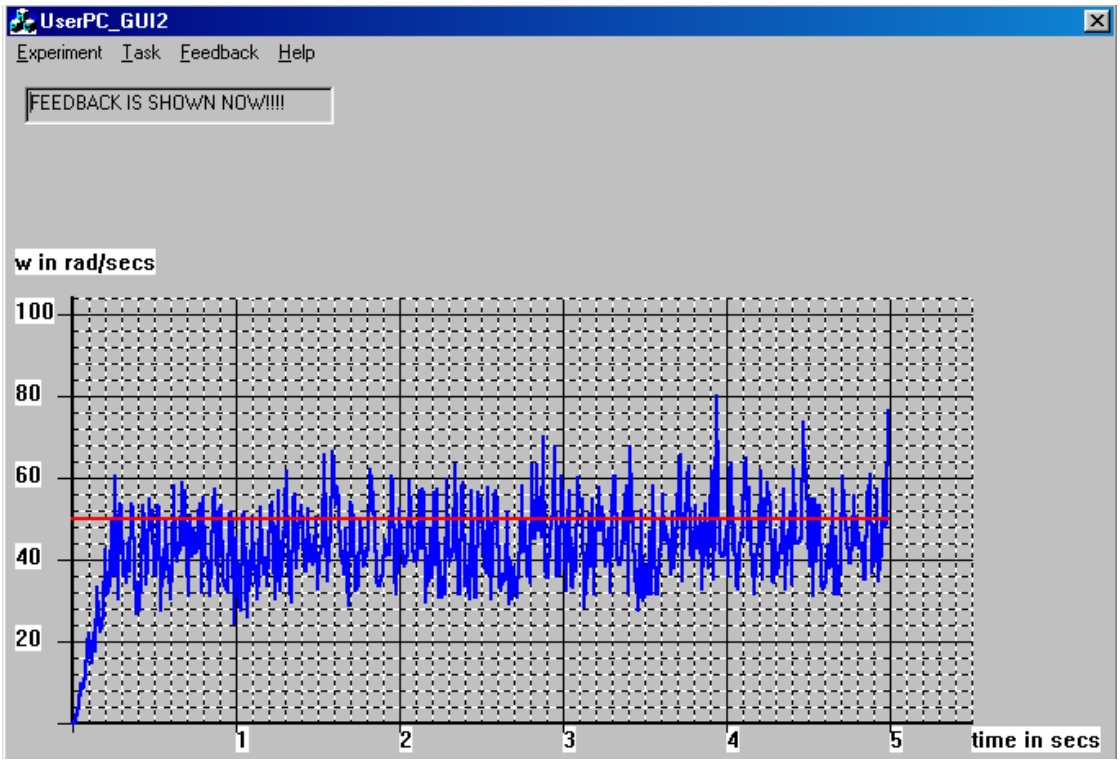


Fig4.9 : Response curve for Kp=21.8, Ki=1, Kd=0.001, wref = 50 rad/sec

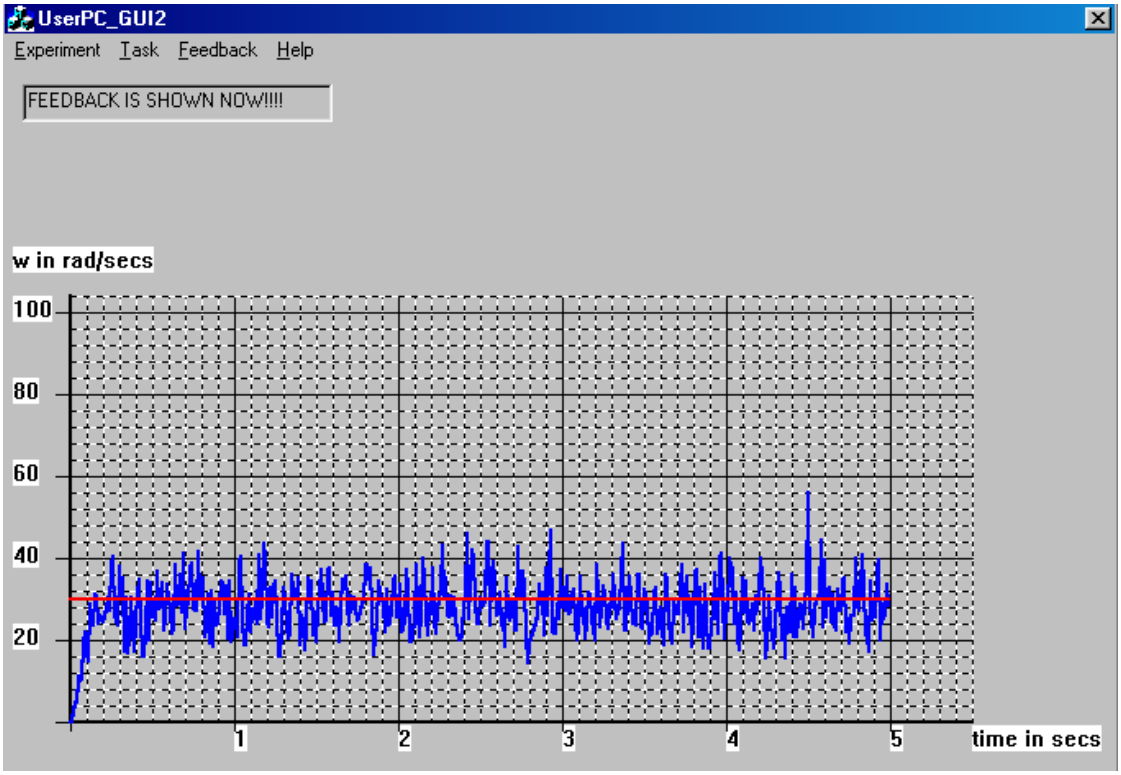


Fig4.10 : Response curve for Kp=21.8, Ki=1, Kd=0.001, wref = 30 rad/sec

CHAPTER 5
CONCLUSION AND FUTURE WORK

In this study a new architecture for remote mechatronics laboratory is proposed. In order to increase efficiency using fewer number of computers, multi-threaded programming is proposed. For flexibility of the system, communication via objects is used. This way, a new plant can be added to the system by simply defining new set of cmd and fb objects and designing the necessary GUI specific to that plant. Two set of cmd and fb objects are given as example, one for DC motor control experiment and other for a 7-DOF robot manipulator. Due to time restriction, only DC motor control experiment is implemented but implementation of the other plants is straightforward. This way, communication via objects is proved to be successful.

For future work first multi-threaded programming wil be implemented for multi-user multi-plant scheme. As experiment, fifty-users eight-plants scheme will be established in order to make efficient use of equipments like computers, input/output cards, and software license for MATLAB and real-time toolbox. Second, other types of plants like robot manipulator, XYZ table, mobile robots, microrobot, signal generator, oscilloscope etc will be connected to the system. Third visual feedback will be added in order to give a more realistic experiment to the user. Finally, JAVA will be used to write the client software to enable full platform independence and enable Linux users to use the system. Additionally, the system can be used as an infrastructure for a different application : Hierarchical hybrid (central + distributed) intelligent control. An example system for this type of application is shown in Fig5.1. In this application a group of mobile robots, robot manipulators, XYZ tables, CNC machines are used in order to accomplish a certain task, like production and transportation of certain products. Each

robot has its own embedded controller and certain intelligence to accomplish low level tasks, like going to some certain position. Artificial Intelligence is running in the UserPC as higher layer. Human supervisors can program, observe and interrupt the system via a GUI. The proposed architecture in this study constitutes the organisation part of the hierarchical control. So, the AI unit in the higher layer decides what each robot must do in order to achieve a high level objective, sends the low level objectives to each robot as cmd object, receives the status of the whole plant as fb objects and decides on the next action. User can program the high level AI and knowledge base. This experiment is also thought as a future work.

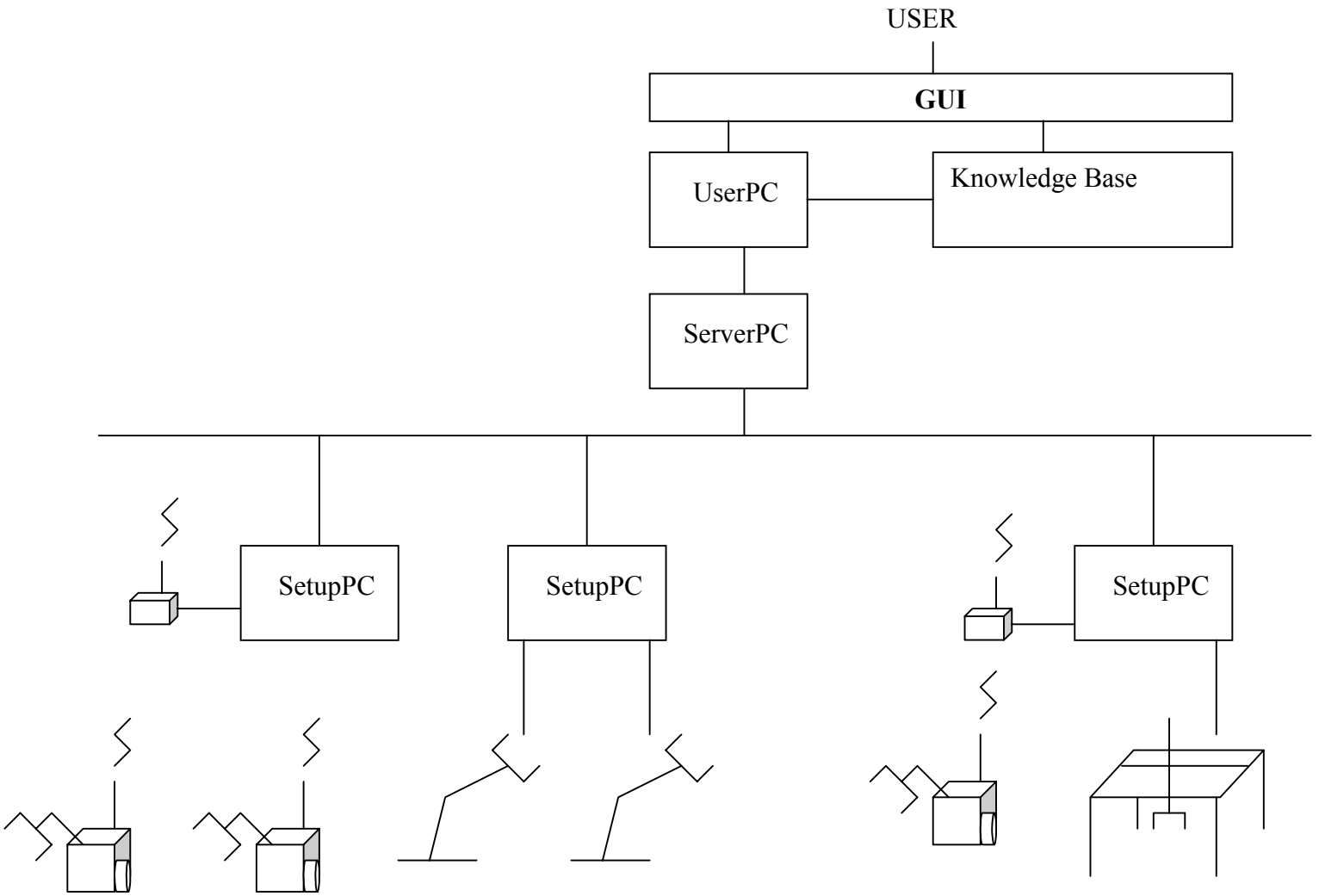


Fig5.1 : A different kind of application using the proposed architecture as organisation layer

APPENDIX A

SOURCE CODE FOR USERPC

```
// UserPC_GUI2.h : main header file for the USERPC_GUI2 application
//

#if
!defined(AFX_USERPC_GUI2_H__F7C2A932_842E_11D6_BBA3_0010A4BF96E1_
_INCLUDED_)
#define
AFX_USERPC_GUI2_H__F7C2A932_842E_11D6_BBA3_0010A4BF96E1__INCLU
DED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CUserPC_GUI2App:
// See UserPC_GUI2.cpp for the implementation of this class
//

class CUserPC_GUI2App : public CWinApp
{
public:
	CUserPC_GUI2App();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CUserPC_GUI2App)
	public:
```

```
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL

// Implementation

        //{{AFX_MSG(CUserPC_GUI2App)
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.
```

#endif //
!defined(AFX_USERPC_GUI2_H__F7C2A932_842E_11D6_BBA3_0010A4BF96E1__INCLUDED_)

```
// UserPC_GUI2.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "UserPC_GUI2.h"
#include "UserPC_GUI2Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CUserPC_GUI2App

BEGIN_MESSAGE_MAP(CUserPC_GUI2App, CWinApp)
	//{{AFX_MSG_MAP(CUserPC_GUI2App)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CUserPC_GUI2App construction

CUserPC_GUI2App::CUserPC_GUI2App()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
```

```
// The one and only CUserPC_GUI2App object

CUserPC_GUI2App theApp;

/////////////////////////////////////////////////////////////////////////////
// CUserPC_GUI2App initialization

BOOL CUserPC_GUI2App::InitInstance()
{
	if (!AfxSocketInit())
	{
		AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
		return FALSE;
	}

	AfxEnableControlContainer();

	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a shared
DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC statically
#endif

	CUserPC_GUI2Dlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
```

```
            // TODO: Place code here to handle when the dialog is
            //  dismissed with OK
        }
        else if (nResponse == IDCANCEL)
        {
            // TODO: Place code here to handle when the dialog is
            //  dismissed with Cancel
        }

        // Since the dialog has been closed, return FALSE so that we exit the
        //  application, rather than start the application's message pump.
        return FALSE;
}
```

```
// UserPC_GUI2Dlg.h : header file
//

#if
!defined(AFX_USERPC_GUI2DLG_H__F7C2A934_842E_11D6_BBA3_0010A4BF9
6E1__INCLUDED_)
#define
AFX_USERPC_GUI2DLG_H__F7C2A934_842E_11D6_BBA3_0010A4BF96E1__IN
CLUDED_

#include "MyReceiveSocket.h"         // Added by ClassView
#include "Task.h"       // Added by ClassView
#include "SetExpParamDlg.h"          // Added by ClassView
#include "Cmd.h"       // Added by ClassView
#include "MySocket.h"          // Added by ClassView
#include "Feedback.h"          // Added by ClassView
#include "Fb.h"          // Added by ClassView
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <afxtempl.h>

/////////////////////////////////////////////////////////////////////////////
// CUserPC_GUI2Dlg dialog

class CUserPC_GUI2Dlg : public CDialog
{
// Construction
public:
        void OnSendRecSock();
        void OnReceiveRecSock();
        void OnConnectRecSock();
        void OnCloseRecSock();
```

```
        void OnAcceptRecSock();
        void OnReceive();
        void OnSend();
        void OnClose();
        void OnConnect();
        void OnAccept();
        CUserPC_GUI2Dlg(CWnd* pParent = NULL);       // standard constructor

// Dialog Data
        //{{AFX_DATA(CUserPC_GUI2Dlg)
        enum { IDD = IDD_USERPC_GUI2_DIALOG };
        CString         m_status;
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CUserPC_GUI2Dlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);       // DDX/DDV
support
        //}}AFX_VIRTUAL

// Implementation
protected:
        HICON m_hIcon;

        // Generated message map functions
        //{{AFX_MSG(CUserPC_GUI2Dlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        afx_msg void OnPaint();
        afx_msg HCURSOR OnQueryDragIcon();
        afx_msg void OnSetexperimentparameters();
        afx_msg void OnSendtask();
        afx_msg void OnDisconnect();
```

```
	afx_msg void OnShowfeedback();
	afx_msg void OnConnectMenu();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
private:
	float m_reference_amplitude;
	void ProcessFeedback();
	CFb m_cfbdefault;
	CMyReceiveSocket m_sReceiveSocket;
	CFb m_GUIfb;
	CFeedback m_GUIfeedback;
	CCmd m_ccmddefault;
	void ReceiveCFeedback();
	CMySocket m_sConnectSocket1;
	void SendCTask();
	CTask m_GUItask;
	CCmd m_GUIcmd;
	CSetExpParamDlg m_dSetExpParamDlg;
	int toggle;
	CArray<double, double> m_GUIresponse_curve;
	CArray<double, double> m_GUIreference_curve;
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_USERPC_GUI2DLG_H__F7C2A934_842E_11D6_BBA3_0010A4BF9
6E1__INCLUDED_)
```

```
// UserPC_GUI2Dlg.cpp : implementation file
//

#include "stdafx.h"
#include "UserPC_GUI2.h"
#include "SetExpParamDlg.h"
#include "UserPC_GUI2Dlg.h"
#include "Task.h"
#include "MySocket.h"
#include "math.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
```

```
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CUserPC_GUI2Dlg dialog

CUserPC_GUI2Dlg::CUserPC_GUI2Dlg(CWnd* pParent /*=NULL*/)
	: CDialog(CUserPC_GUI2Dlg::IDD, pParent)
{
```

```
	//{{AFX_DATA_INIT(CUserPC_GUI2Dlg)
	m_status = _T("");
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CUserPC_GUI2Dlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CUserPC_GUI2Dlg)
	DDX_Text(pDX, IDC_STATUS, m_status);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CUserPC_GUI2Dlg, CDialog)
	//{{AFX_MSG_MAP(CUserPC_GUI2Dlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_COMMAND(IDM_SETEXPERIMENTPARAMETERS,
OnSetexperimentparameters)
	ON_COMMAND(IDM_SENDTASK, OnSendtask)
	ON_COMMAND(IDM_DISCONNECT, OnDisconnect)
	ON_COMMAND(IDM_SHOWFEEDBACK, OnShowfeedback)
	ON_COMMAND(IDM_CONNECT, OnConnectMenu)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CUserPC_GUI2Dlg message handlers

BOOL CUserPC_GUI2Dlg::OnInitDialog()
{
```

```
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here
	m_dSetExpParamDlg.m_controllertype = 0;
	m_dSetExpParamDlg.m_referencetype = 0;

	m_sConnectSocket1.SetParent(this);
	m_sReceiveSocket.SetParent(this);

	m_ccmddefault.cmdname = 0;
```

```
	m_ccmddefault.arg1 = 0;

	m_cfbdefault.fbname = 0;
	m_cfbdefault.arg1 = 0;

	toggle = 1;

	float n=0.0;
	for (int i=0; i<500; i++)
	{
		n=i;
		m_GUIresponse_curve.Add(100*sin(n/10));
		m_GUIreference_curve.Add(100);
	}

	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CUserPC_GUI2Dlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.
```

```
void CUserPC_GUI2Dlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM)
dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CUserPC_GUI2Dlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}
```

```
void CUserPC_GUI2Dlg::OnSetexperimentparameters()
{
	// TODO: Add your command handler code here
	if (m_dSetExpParamDlg.DoModal() == IDOK)
	{
		// BUILD CMD OBJECT AND ADD IT TO TASK OBJECT HERE!!!!
		int size=0;

		m_GUIcmd.cmdname = 1;
		m_GUIcmd.arg1 = m_dSetExpParamDlg.m_controllertype;
		size = m_GUItask.cmdlist.GetSize();
		m_GUItask.cmdlist.SetAtGrow(size, m_GUIcmd);

		m_GUIcmd.cmdname = 2;
		m_GUIcmd.arg1 = m_dSetExpParamDlg.m_referencetype;
		size = m_GUItask.cmdlist.GetSize();
		m_GUItask.cmdlist.SetAtGrow(size, m_GUIcmd);

		m_GUIcmd.cmdname = 3;
		m_GUIcmd.arg1 = m_dSetExpParamDlg.m_kp;
		size = m_GUItask.cmdlist.GetSize();
		m_GUItask.cmdlist.SetAtGrow(size, m_GUIcmd);

		m_GUIcmd.cmdname = 4;
		m_GUIcmd.arg1 = m_dSetExpParamDlg.m_ki;
		size = m_GUItask.cmdlist.GetSize();
		m_GUItask.cmdlist.SetAtGrow(size, m_GUIcmd);

		m_GUIcmd.cmdname = 5;
		m_GUIcmd.arg1 = m_dSetExpParamDlg.m_kd;
		size = m_GUItask.cmdlist.GetSize();
		m_GUItask.cmdlist.SetAtGrow(size, m_GUIcmd);

		m_GUIcmd.cmdname = 6;
```

```
            m_GUIcmd.arg1 = m_dSetExpParamDlg.m_kfr;
            size = m_GUItask.cmdlist.GetSize();
            m_GUItask.cmdlist.SetAtGrow(size, m_GUIcmd);

            m_GUIcmd.cmdname = 7;
            m_GUIcmd.arg1 = m_dSetExpParamDlg.m_amplitude;
            size = m_GUItask.cmdlist.GetSize();
            m_GUItask.cmdlist.SetAtGrow(size, m_GUIcmd);

            m_GUIcmd.cmdname = 8;
            m_GUIcmd.arg1 = m_dSetExpParamDlg.m_frequency;
            size = m_GUItask.cmdlist.GetSize();
            m_GUItask.cmdlist.SetAtGrow(size, m_GUIcmd);

            m_GUIcmd.cmdname = 9;
            m_GUIcmd.arg1 = 0;
            size = m_GUItask.cmdlist.GetSize();
            m_GUItask.cmdlist.SetAtGrow(size, m_GUIcmd);

            m_reference_amplitude = m_dSetExpParamDlg.m_amplitude;
        }
}

void CUserPC_GUI2Dlg::OnSendtask()
{
        // TODO: Add your command handler code here
        m_GUItask.TaskID = 1;
        m_GUItask.source = 1;                   // FOR USERPC#1
        m_GUItask.destination = 1;   // FOR DC MOTOR SETUP#1
        m_GUItask.accessrank = 1;
        m_GUItask.priority = 1;
        // !!!! Here the task must be sent !!!!
        SendCTask();
        m_GUItask.cmdlist.RemoveAll();
```

```
	m_GUItask.cmdlist.FreeExtra();
}

void CUserPC_GUI2Dlg::OnConnectMenu()
{
	// TODO: Add your command handler code here
	m_sConnectSocket1.Create();
	m_sConnectSocket1.Connect("10.92.54.218",4000);

	m_sReceiveSocket.Create();
	m_sReceiveSocket.Connect("10.92.54.218",4001);
}

void CUserPC_GUI2Dlg::OnDisconnect()
{
	// TODO: Add your command handler code here
	m_sConnectSocket1.Close();
	m_sReceiveSocket.Close();
	m_status = "DISCONNECTED!!!!";
	UpdateData(FALSE);
}

void CUserPC_GUI2Dlg::OnShowfeedback()
{
	// TODO: Add your command handler code here

	/////// INITIALIZATION OF THE PLOT ///////////////

	this->RedrawWindow();

	int i=0, i1=0;
	float n=0.0;
	CClientDC dc(this);
	int number_of_sample_points = 500;
```

```
CPoint initial_point;
initial_point.x = 40;
initial_point.y = 400;
dc.MoveTo(initial_point.x, initial_point.y);

CPen lPenBlackWide(PS_SOLID, 2, RGB(0,0,0));
dc.SelectObject(&lPenBlackWide);
dc.MoveTo(initial_point.x ,initial_point.y);
dc.LineTo(550+initial_point.x,initial_point.y);
dc.MoveTo(initial_point.x ,initial_point.y);
dc.LineTo(initial_point.x ,initial_point.y - 260);

dc.TextOut(initial_point.x +100 ,initial_point.y+2, "1");
dc.TextOut(initial_point.x +200 ,initial_point.y+2, "2");
dc.TextOut(initial_point.x +300 ,initial_point.y+2, "3");
dc.TextOut(initial_point.x +400 ,initial_point.y+2, "4");
dc.TextOut(initial_point.x +500 ,initial_point.y+2, "5");
dc.TextOut(initial_point.x +550 ,initial_point.y+2, "time in secs");

dc.TextOut(initial_point.x -35 ,initial_point.y-50-10, "20");
dc.TextOut(initial_point.x -35 ,initial_point.y-100-10, "40");
dc.TextOut(initial_point.x -35 ,initial_point.y-150-10, "60");
dc.TextOut(initial_point.x -35 ,initial_point.y-200-10, "80");
dc.TextOut(initial_point.x -35 ,initial_point.y-250-10, "100");
dc.TextOut(initial_point.x -35 ,initial_point.y-280-10, "w in rad/secs");

CPen lPenGrid(PS_DOT, 1, RGB(0,0,0));
dc.SelectObject(&lPenGrid);
for (i1=0; i1<56; i1++)
{
        dc.MoveTo(initial_point.x +10*i1 ,initial_point.y);
        dc.LineTo(initial_point.x +10*i1 ,initial_point.y-260);
}
```

```
for (i1=0; i1<27; i1++)
{
	dc.MoveTo(initial_point.x ,initial_point.y -10*i1);
	dc.LineTo(initial_point.x +550 ,initial_point.y -10*i1);
}

CPen lPenBlack(PS_SOLID, 1, RGB(0,0,0));
dc.SelectObject(&lPenBlack);
for (i1=0; i1<6; i1++)
{
	dc.MoveTo(initial_point.x +550 ,initial_point.y -50*i1);
	dc.LineTo(initial_point.x -10 ,initial_point.y -50*i1);
}
for (i1=0; i1<6; i1++)
{
	dc.MoveTo(initial_point.x + 100*i1,initial_point.y +5);
	dc.LineTo(initial_point.x + 100*i1 ,initial_point.y -260);
}
//////// end of initialization of the plot ///////////////

// DRAWING THE REFERENCE AND RESPONSE CURVES //////
CPen lPenBlue(PS_SOLID, 2, RGB(0,0,255));
dc.SelectObject(&lPenBlue);

for (i1=0; i1<499; i1++)
{
	n=i1;
	dc.MoveTo(initial_point.x +i1 ,initial_point.y -
2.5*m_GUIresponse_curve.ElementAt(i1));
	dc.LineTo(initial_point.x +i1+1 ,initial_point.y -
2.5*m_GUIresponse_curve.ElementAt(i1+1));
}

CPen lPenRed(PS_SOLID, 2, RGB(255,0,0));
```

```
	dc.SelectObject(&lPenRed);

	for (i1=0; i1<499; i1++)
	{
		n=i1;
		dc.MoveTo(initial_point.x +i1 ,initial_point.y -
2.5*m_GUIreference_curve.ElementAt(i1));
		dc.LineTo(initial_point.x +i1+1 ,initial_point.y -
2.5*m_GUIreference_curve.ElementAt(i1+1));
	}
	// end of drawing the reference and response curves //


	m_status = "FEEDBACK IS SHOWN NOW!!!!";
	UpdateData(FALSE);
}

void CUserPC_GUI2Dlg::SendCTask()
{
	int iSent;
	CCmd tmpcmd = m_ccmddefault;
	CCmd *tempcmd = &tmpcmd;
	int tmpint = 1;
	int *temptaskvariables = &tmpint;

	// sending the taskid
	*temptaskvariables = m_GUItask.TaskID;
	do
	{
		iSent = m_sConnectSocket1.Send(temptaskvariables,
sizeof(*temptaskvariables));
	} while (iSent == SOCKET_ERROR);

	// sending the tasksource
```

```
	*temptaskvariables = m_GUItask.source;
	do
	{
		iSent = m_sConnectSocket1.Send(temptaskvariables,
sizeof(*temptaskvariables));
	} while (iSent == SOCKET_ERROR);

	// sending the taskdestination
	*temptaskvariables = m_GUItask.destination;
	do
	{
		iSent = m_sConnectSocket1.Send(temptaskvariables,
sizeof(*temptaskvariables));
	} while (iSent == SOCKET_ERROR);

	// sending the taskpriority
	*temptaskvariables = m_GUItask.priority;
	do
	{
		iSent = m_sConnectSocket1.Send(temptaskvariables,
sizeof(*temptaskvariables));
	} while (iSent == SOCKET_ERROR);

	// sending the taskaccessrank
	*temptaskvariables = m_GUItask.accessrank;
	do
	{
		iSent = m_sConnectSocket1.Send(temptaskvariables,
sizeof(*temptaskvariables));
	} while (iSent == SOCKET_ERROR);

	// sending the number of commands
	*temptaskvariables = m_GUItask.cmdlist.GetSize();
	do
```

```
	{
		iSent = m_sConnectSocket1.Send(temptaskvariables,
sizeof(*temptaskvariables));
	} while (iSent == SOCKET_ERROR);

	// sending the commands
	for (int k=1; k<m_GUItask.cmdlist.GetSize()+1 ; k++)
	{
		*tempcmd = m_GUItask.cmdlist.ElementAt(k-1);
		do
		{
			iSent = m_sConnectSocket1.Send(tempcmd, sizeof(*tempcmd));
		} while (iSent == SOCKET_ERROR);
	}
}

void CUserPC_GUI2Dlg::ReceiveCFeedback()
{
	int iRcvd;
	int number_of_fbs;
	CFb tmpfb = m_cfbdefault;
	CFb *tempfb = &tmpfb;
	int tmpint = 1;
	int *tempfeedbackvariables = &tmpint;

	m_GUIfeedback.fblist.RemoveAll();
	m_GUIfeedback.fblist.FreeExtra();

	// receiving the feedbackid
	do
	{
		iRcvd = m_sReceiveSocket.Receive(tempfeedbackvariables,
sizeof(*tempfeedbackvariables));
	} while (iRcvd == SOCKET_ERROR);
```

```
	m_GUIfeedback.FeedbackID = *tempfeedbackvariables;

	// receiving the feedbacksource
	do
	{
		iRcvd = m_sReceiveSocket.Receive(tempfeedbackvariables,
sizeof(*tempfeedbackvariables));
	} while (iRcvd == SOCKET_ERROR);
	m_GUIfeedback.source = *tempfeedbackvariables;

	// receiving the feedbackdestination
	do
	{
		iRcvd = m_sReceiveSocket.Receive(tempfeedbackvariables,
sizeof(*tempfeedbackvariables));
	} while (iRcvd == SOCKET_ERROR);
	m_GUIfeedback.destination = *tempfeedbackvariables;

	// receiving the feedbackpriority
	do
	{
		iRcvd = m_sReceiveSocket.Receive(tempfeedbackvariables,
sizeof(*tempfeedbackvariables));
	} while (iRcvd == SOCKET_ERROR);
	m_GUIfeedback.priority = *tempfeedbackvariables;

	// receiving the taskaccepted
	do
	{
		iRcvd = m_sReceiveSocket.Receive(tempfeedbackvariables,
sizeof(*tempfeedbackvariables));
	} while (iRcvd == SOCKET_ERROR);
	m_GUIfeedback.task_accepted = *tempfeedbackvariables;
```

```
	// receiving the number of fbs
	do
	{
		iRcvd = m_sReceiveSocket.Receive(tempfeedbackvariables,
sizeof(*tempfeedbackvariables));
	} while (iRcvd == SOCKET_ERROR);
	number_of_fbs = *tempfeedbackvariables;

	// receiving the fbs
	for (int k=1; k<number_of_fbs+1; k++)
	{
		do
		{
			iRcvd = m_sReceiveSocket.Receive(tempfb, sizeof(*tempfb));
		} while (iRcvd == SOCKET_ERROR);
		m_GUIfeedback.fblist.SetAtGrow(k-1, *tempfb);
	}
	ProcessFeedback();
}

void CUserPC_GUI2Dlg::OnAccept()
{

}

void CUserPC_GUI2Dlg::OnConnect()
{
	m_status = "CONNECTED!!!!";
	UpdateData(FALSE);
}

void CUserPC_GUI2Dlg::OnClose()
{
	m_sConnectSocket1.Close();
```

```
	m_status = "CONNECTION CLOSED!!!!";
	UpdateData(FALSE);
}

void CUserPC_GUI2Dlg::OnSend()
{
	m_status = "TASK SENT!!!!";
	UpdateData(FALSE);
}

void CUserPC_GUI2Dlg::OnReceive()
{
}

void CUserPC_GUI2Dlg::OnAcceptRecSock()
{

}

void CUserPC_GUI2Dlg::OnCloseRecSock()
{
	m_sReceiveSocket.Close();
	m_status = "CONNECTION CLOSED!!!!";
	UpdateData(FALSE);
}

void CUserPC_GUI2Dlg::OnConnectRecSock()
{

}

void CUserPC_GUI2Dlg::OnReceiveRecSock()
{
	ReceiveCFeedback();
```

```
	m_status = "FEEDBACK RECEIVED!!!!";
	UpdateData(FALSE);
}

void CUserPC_GUI2Dlg::OnSendRecSock()
{

}

void CUserPC_GUI2Dlg::ProcessFeedback()
{
	int i=0;
	int size=0;
	int fbname=0;
	float arg1=0;
	int number_of_samples=500;
	int reference_type = 0;
	float reference_amplitude=0, reference_frequency=0;
	float experiment_sampling_time=0.01;

	reference_amplitude = m_reference_amplitude;

	m_GUIresponse_curve.RemoveAll();
	m_GUIresponse_curve.FreeExtra();
	m_GUIreference_curve.RemoveAll();
	m_GUIreference_curve.FreeExtra();

	size = m_GUIfeedback.fblist.GetSize();
	for (i=0; i<size; i++)
	{
		fbname = m_GUIfeedback.fblist.ElementAt(i).fbname;
		arg1 = m_GUIfeedback.fblist.ElementAt(i).arg1;
		if (fbname == 9)
			m_GUIresponse_curve.Add(arg1);
```

```
		if (fbname == 2)
			reference_type = arg1;
		if (fbname == 7)
			reference_amplitude = arg1;
		if (fbname == 8)
			reference_frequency = arg1;
	}
	if (reference_type == 0)
	{
		for (i=0; i<number_of_samples; i++)
		{
			m_GUIreference_curve.Add(reference_amplitude);
		}
	}
	if (reference_type == 1)
	{
		for (i=0; i<number_of_samples; i++)
		{

	m_GUIreference_curve.Add(reference_amplitude*sin(i*reference_frequency*e
xperiment_sampling_time));
		}
	}
}
```

```
#if
!defined(AFX_SETEXPPARAMDLG_H__F7C2A93C_842E_11D6_BBA3_0010A4B
F96E1__INCLUDED_)
#define
AFX_SETEXPPARAMDLG_H__F7C2A93C_842E_11D6_BBA3_0010A4BF96E1__I
NCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// SetExpParamDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CSetExpParamDlg dialog

class CSetExpParamDlg : public CDialog
{
// Construction
public:
	CSetExpParamDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CSetExpParamDlg)
	enum { IDD = IDD_SETEXPERIMENTPARAMETERS };
	float	m_amplitude;
	float	m_frequency;
	float	m_kd;
	float	m_kfr;
	float	m_ki;
	float	m_kp;
	int		m_controllertype;
	int		m_referencetype;
	//}}AFX_DATA
```

```
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CSetExpParamDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        virtual BOOL OnInitDialog();

        // Generated message map functions
        //{{AFX_MSG(CSetExpParamDlg)
        afx_msg void OnOlwithfrcomp();
        afx_msg void OnOlwithoutfrcomp();
        afx_msg void OnCl();
        afx_msg void OnStep();
        afx_msg void OnSinusoidal();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_SETEXPPARAMDLG_H__F7C2A93C_842E_11D6_BBA3_0010A4B
F96E1__INCLUDED_)
```

```
// SetExpParamDlg.cpp : implementation file
//

#include "stdafx.h"
#include "UserPC_GUI2.h"
#include "SetExpParamDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSetExpParamDlg dialog


CSetExpParamDlg::CSetExpParamDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CSetExpParamDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CSetExpParamDlg)
	m_amplitude = 0.0f;
	m_frequency = 0.0f;
	m_kd = 0.0f;
	m_kfr = 0.0f;
	m_ki = 0.0f;
	m_kp = 0.0f;
	m_controllertype = -1;
	m_referencetype = -1;
	//}}AFX_DATA_INIT
}


void CSetExpParamDlg::DoDataExchange(CDataExchange* pDX)
```

```
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CSetExpParamDlg)
	DDX_Text(pDX, IDC_AMPLITUDE, m_amplitude);
	DDX_Text(pDX, IDC_FREQUENCY, m_frequency);
	DDX_Text(pDX, IDC_KD, m_kd);
	DDX_Text(pDX, IDC_KFR, m_kfr);
	DDX_Text(pDX, IDC_KI, m_ki);
	DDX_Text(pDX, IDC_KP, m_kp);
	DDX_Radio(pDX, IDC_OLWITHOUTFRCOMP, m_controllertype);
	DDX_Radio(pDX, IDC_STEP, m_referencetype);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CSetExpParamDlg, CDialog)
	//{{AFX_MSG_MAP(CSetExpParamDlg)
	ON_BN_CLICKED(IDC_OLWITHFRCOMP2, OnOlwithfrcomp)
	ON_BN_CLICKED(IDC_OLWITHOUTFRCOMP, OnOlwithoutfrcomp)
	ON_BN_CLICKED(IDC_CL, OnCl)
	ON_BN_CLICKED(IDC_STEP, OnStep)
	ON_BN_CLICKED(IDC_SINUSOIDAL, OnSinusoidal)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSetExpParamDlg message handlers


void CSetExpParamDlg::OnOlwithfrcomp()
{
	// TODO: Add your control notification handler code here
	GetDlgItem(IDC_KP)->EnableWindow(TRUE);
	GetDlgItem(IDC_KI)->EnableWindow(FALSE);
```

```
	GetDlgItem(IDC_KD)->EnableWindow(FALSE);
	GetDlgItem(IDC_KFR)->EnableWindow(TRUE);
}

void CSetExpParamDlg::OnOlwithoutfrcomp()
{
	// TODO: Add your control notification handler code here
	GetDlgItem(IDC_KP)->EnableWindow(TRUE);
	GetDlgItem(IDC_KI)->EnableWindow(FALSE);
	GetDlgItem(IDC_KD)->EnableWindow(FALSE);
	GetDlgItem(IDC_KFR)->EnableWindow(FALSE);
}

void CSetExpParamDlg::OnCl()
{
	// TODO: Add your control notification handler code here
	GetDlgItem(IDC_KP)->EnableWindow(TRUE);
	GetDlgItem(IDC_KI)->EnableWindow(TRUE);
	GetDlgItem(IDC_KD)->EnableWindow(TRUE);
	GetDlgItem(IDC_KFR)->EnableWindow(FALSE);
}

BOOL CSetExpParamDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	GetDlgItem(IDC_KP)->EnableWindow(TRUE);
	GetDlgItem(IDC_KI)->EnableWindow(FALSE);
	GetDlgItem(IDC_KD)->EnableWindow(FALSE);
	GetDlgItem(IDC_KFR)->EnableWindow(FALSE);
	GetDlgItem(IDC_FREQUENCY)->EnableWindow(FALSE);

	return TRUE;
}
```

```
void CSetExpParamDlg::OnStep()
{
	// TODO: Add your control notification handler code here
	GetDlgItem(IDC_AMPLITUDE)->EnableWindow(TRUE);
	GetDlgItem(IDC_FREQUENCY)->EnableWindow(FALSE);
}

void CSetExpParamDlg::OnSinusoidal()
{
	// TODO: Add your control notification handler code here
	GetDlgItem(IDC_AMPLITUDE)->EnableWindow(TRUE);
	GetDlgItem(IDC_FREQUENCY)->EnableWindow(TRUE);
}
```

```
#if
!defined(AFX_MYSOCKET_H__0EB06560_86CC_11D6_BBA3_0010A4BF96E1__I
NCLUDED_)
#define
AFX_MYSOCKET_H__0EB06560_86CC_11D6_BBA3_0010A4BF96E1__INCLUD
ED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// MySocket.h : header file
//



/////////////////////////////////////////////////////////////////////////////
// CMySocket command target

class CMySocket : public CAsyncSocket
{
// Attributes
public:

// Operations
public:
	CMySocket();
	virtual ~CMySocket();

// Overrides
public:
	void SetParent(CDialog* pWnd);
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CMySocket)
	//}}AFX_VIRTUAL
```

```
	// Generated message map functions
	//{{AFX_MSG(CMySocket)
		// NOTE - the ClassWizard will add and remove member functions here.
	//}}AFX_MSG

// Implementation
protected:
	virtual void OnSend(int nErrorCode);
	virtual void OnReceive(int nErrorCode);
	virtual void OnClose(int nErrorCode);
	virtual void OnConnect(int nErrorCode);
	virtual void OnAccept(int nErrorCode);
private:
	CDialog* m_pWnd;
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_MYSOCKET_H__0EB06560_86CC_11D6_BBA3_0010A4BF96E1__I
NCLUDED_)
```

```
// MySocket.cpp : implementation file
//

#include "stdafx.h"
#include "UserPC_GUI2.h"
#include "MySocket.h"
#include "UserPC_GUI2Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMySocket

CMySocket::CMySocket()
{
}

CMySocket::~CMySocket()
{
}


// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CMySocket, CAsyncSocket)
	//{{AFX_MSG_MAP(CMySocket)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif  // 0
```

```
//////////////////////////////////////////////////////////////////////////
// CMySocket member functions

void CMySocket::SetParent(CDialog *pWnd)
{
	m_pWnd = pWnd;
}

void CMySocket::OnAccept(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CUserPC_GUI2Dlg*)m_pWnd)->OnAccept();
	}
}

void CMySocket::OnConnect(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CUserPC_GUI2Dlg*)m_pWnd)->OnConnect();
	}
}

void CMySocket::OnClose(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CUserPC_GUI2Dlg*)m_pWnd)->OnClose();
	}
}

void CMySocket::OnReceive(int nErrorCode)
{
```

```
	if (nErrorCode == 0)
	{
		((CUserPC_GUI2Dlg*)m_pWnd)->OnReceive();
	}

}

void CMySocket::OnSend(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CUserPC_GUI2Dlg*)m_pWnd)->OnSend();
	}

}
```

```
#if
!defined(AFX_MYRECEIVESOCKET_H__94C02542_882B_11D6_BBA3_0010A4B
F96E1__INCLUDED_)
#define
AFX_MYRECEIVESOCKET_H__94C02542_882B_11D6_BBA3_0010A4BF96E1__
INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// MyReceiveSocket.h : header file
//



/////////////////////////////////////////////////////////////////////////////
// CMyReceiveSocket command target

class CMyReceiveSocket : public CAsyncSocket
{
// Attributes
public:

// Operations
public:
	CMyReceiveSocket();
	virtual ~CMyReceiveSocket();

// Overrides
public:
	void SetParent(CDialog* pWnd);
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CMyReceiveSocket)
	//}}AFX_VIRTUAL
```

```
	// Generated message map functions
	//{{AFX_MSG(CMyReceiveSocket)
		// NOTE - the ClassWizard will add and remove member functions here.
	//}}AFX_MSG

// Implementation
protected:
	virtual void OnSend(int nErrorCode);
	virtual void OnReceive(int nErrorCode);
	virtual void OnConnect(int nErrorCode);
	virtual void OnClose(int nErrorCode);
	virtual void OnAccept(int nErrorCode);
private:
	CDialog* m_pWnd;
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_MYRECEIVESOCKET_H__94C02542_882B_11D6_BBA3_0010A4B
F96E1__INCLUDED_)
```

```
// MyReceiveSocket.cpp : implementation file
//

#include "stdafx.h"
#include "UserPC_GUI2.h"
#include "MyReceiveSocket.h"
#include "UserPC_GUI2Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMyReceiveSocket

CMyReceiveSocket::CMyReceiveSocket()
{
}

CMyReceiveSocket::~CMyReceiveSocket()
{
}


// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CMyReceiveSocket, CAsyncSocket)
	//{{AFX_MSG_MAP(CMyReceiveSocket)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif  // 0
```

```
//////////////////////////////////////////////////////////////////////////
// CMyReceiveSocket member functions

void CMyReceiveSocket::OnAccept(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CUserPC_GUI2Dlg*)m_pWnd)->OnAcceptRecSock();
	}
}

void CMyReceiveSocket::OnClose(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CUserPC_GUI2Dlg*)m_pWnd)->OnCloseRecSock();
	}
}

void CMyReceiveSocket::OnConnect(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CUserPC_GUI2Dlg*)m_pWnd)->OnConnectRecSock();
	}
}

void CMyReceiveSocket::OnReceive(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CUserPC_GUI2Dlg*)m_pWnd)->OnReceiveRecSock();
	}
}
```

```
void CMyReceiveSocket::OnSend(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CUserPC_GUI2Dlg*)m_pWnd)->OnSendRecSock();
	}
}

void CMyReceiveSocket::SetParent(CDialog *pWnd)
{
	m_pWnd = pWnd;
}
```

```
// Cmd.h: interface for the CCmd class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_CMD_H__F7C2A93D_842E_11D6_BBA3_0010A4BF96E1__INCLU
DED_)
#define
AFX_CMD_H__F7C2A93D_842E_11D6_BBA3_0010A4BF96E1__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CCmd
{
public:
	int cmdname;		// 1 for SetExpParamsandStart
	float arg1;			// Type of Controller : 1->OL without fr.c.
	CCmd();
	virtual ~CCmd();

};

#endif //
!defined(AFX_CMD_H__F7C2A93D_842E_11D6_BBA3_0010A4BF96E1__INCLU
DED_)
```

```
// Cmd.cpp: implementation of the CCmd class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "UserPC_GUI2.h"
#include "Cmd.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CCmd::CCmd()
{

}

CCmd::~CCmd()
{

}
```

```
// Fb.h: interface for the CFb class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_FB_H__94C02540_882B_11D6_BBA3_0010A4BF96E1__INCLUDED
_)
#define
AFX_FB_H__94C02540_882B_11D6_BBA3_0010A4BF96E1__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CFb
{
public:
	int fbname;			// 1-> status_of_exp, 2-> response curve
	float arg1;			// type_of_controller
	CFb();
	virtual ~CFb();

};

#endif //
!defined(AFX_FB_H__94C02540_882B_11D6_BBA3_0010A4BF96E1__INCLUDED
_)
```

```
// Fb.cpp: implementation of the CFb class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "UserPC_GUI2.h"
#include "Fb.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CFb::CFb()
{

}

CFb::~CFb()
{

}
```

```
// Task.h: interface for the CTask class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_TASK_H__F7C2A93E_842E_11D6_BBA3_0010A4BF96E1__INCLU
DED_)
#define
AFX_TASK_H__F7C2A93E_842E_11D6_BBA3_0010A4BF96E1__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "Cmd.h"
#include <afxtempl.h>

class CTask
{
public:
	int TaskID;
	int source;
	int destination;
	int priority;
	int accessrank;
	CArray<CCmd,CCmd> cmdlist;
	CTask();
	virtual ~CTask();

};

#endif //
!defined(AFX_TASK_H__F7C2A93E_842E_11D6_BBA3_0010A4BF96E1__INCLU
DED_)
```

```
// Task.cpp: implementation of the CTask class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "UserPC_GUI2.h"
#include "Task.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CTask::CTask()
{

}

CTask::~CTask()
{

}
```

```
// Feedback.h: interface for the CFeedback class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_FEEDBACK_H__94C02541_882B_11D6_BBA3_0010A4BF96E1__IN
CLUDED_)
#define
AFX_FEEDBACK_H__94C02541_882B_11D6_BBA3_0010A4BF96E1__INCLUDE
D_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "Fb.h"
#include <afxtempl.h>

class CFeedback
{
public:
	int task_accepted;
	int priority;
	int destination;
	int source;
	int FeedbackID;
	CArray<CFb, CFb> fblist;
	CFeedback();
	virtual ~CFeedback();

};
```

```
#endif //
!defined(AFX_FEEDBACK_H__94C02541_882B_11D6_BBA3_0010A4BF96E1__IN
CLUDED_)
```

```
// Feedback.cpp: implementation of the CFeedback class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "UserPC_GUI2.h"
#include "Feedback.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CFeedback::CFeedback()
{

}

CFeedback::~CFeedback()
{

}
```

APPENDIX B
SOURCE CODE FOR SETUPPC

```
// iodnm1.h : main header file for the IODNM1 application
//

#if
!defined(AFX_IODNM1_H__B1A80DE4_87CB_11D6_8435_00010318EDFF__INCL
UDED_)
#define
AFX_IODNM1_H__B1A80DE4_87CB_11D6_8435_00010318EDFF__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CIodnm1App:
// See iodnm1.cpp for the implementation of this class
//

class CIodnm1App : public CWinApp
{
public:
	CIodnm1App();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CIodnm1App)
	public:
	virtual BOOL InitInstance();
```

```
        //}}AFX_VIRTUAL

// Implementation

        //{{AFX_MSG(CIodnm1App)
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.
```

#endif //
!defined(AFX_IODNM1_H__B1A80DE4_87CB_11D6_8435_00010318EDFF__IN
CLUDED_)

```
// iodnm1.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "iodnm1.h"
#include "iodnm1Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CIodnm1App

BEGIN_MESSAGE_MAP(CIodnm1App, CWinApp)
	//{{AFX_MSG_MAP(CIodnm1App)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CIodnm1App construction

CIodnm1App::CIodnm1App()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
```

```
// The one and only CIodnm1App object

CIodnm1App theApp;

/////////////////////////////////////////////////////////////////////////////
// CIodnm1App initialization

BOOL CIodnm1App::InitInstance()
{
	if (!AfxSocketInit())
	{
		AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
		return FALSE;
	}

	AfxEnableControlContainer();

	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a shared
DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC statically
#endif

	CIodnm1Dlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
```

```
            // TODO: Place code here to handle when the dialog is
            //  dismissed with OK
        }
        else if (nResponse == IDCANCEL)
        {
            // TODO: Place code here to handle when the dialog is
            //  dismissed with Cancel
        }

        // Since the dialog has been closed, return FALSE so that we exit the
        //  application, rather than start the application's message pump.
        return FALSE;
}
```

```
// iodnm1Dlg.h : header file
//

#if
!defined(AFX_IODNM1DLG_H__B1A80DE6_87CB_11D6_8435_00010318EDFF__I
NCLUDED_)
#define
AFX_IODNM1DLG_H__B1A80DE6_87CB_11D6_8435_00010318EDFF__INCLUD
ED_

#include "MySocket1.h"        // Added by ClassView
#include "MySocket2.h"        // Added by ClassView
#include "Task.h"       // Added by ClassView
#include "Cmd.h"       // Added by ClassView
#include "Feedback.h"          // Added by ClassView
#include "Fb.h"          // Added by ClassView
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////////////////
// CIodnm1Dlg dialog

class CIodnm1Dlg : public CDialog
{
// Construction
public:
        void OnReceiveSK2();
        void OnSendSK2();
        void OnCloseSK2();
        void OnConnectSK2();
        void OnAcceptSK2();
        void OnReceiveSK1();
        void OnSendSK1();
```

```
        void OnCloseSK1();
        void OnConnectSK1();
        void OnAcceptSK1();
        CIodnm1Dlg(CWnd* pParent = NULL);       // standard constructor

// Dialog Data
        //{{AFX_DATA(CIodnm1Dlg)
        enum { IDD = IDD_IODNM1_DIALOG };
        float    m_frequency;
        float    m_duty;
        int              m_irc;
        float    m_kp;
        float    m_ki;
        float    m_kd;
        float    m_freq_of_reference;
        float    m_amplitude_of_ref;
        float    m_totalexptime;
        float    m_ew;
        CString          m_status;
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CIodnm1Dlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);       // DDX/DDV
support
        //}}AFX_VIRTUAL

// Implementation
protected:
        HICON m_hIcon;

        // Generated message map functions
        //{{AFX_MSG(CIodnm1Dlg)
```

```
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnBupdate();
	afx_msg void OnInitializepwm();
	afx_msg void OnBreadirc();
	afx_msg void OnBstartpid();
	afx_msg void OnBlisten();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
private:
	void SendCFeedback();
	float m_kfr;
	void ApplyControl(int, int, float, float, float, float, float, float);
	int m_reference_type;
	int m_controller_type;
	CFb Applycmdandbuildfb(CCmd cmd1);
	int CheckSafe();
	CFeedback m_setuppc_Feedback;
	void ApplyTaskandBuildFeedback();
	CCmd m_ccmddefault;
	CFb m_cfbdefault;
	CTask m_setuppc_Task;
	void ReceiveCTask();
	CMySocket2 m_sConnectSocketSK2;
	CMySocket2 m_sListenSocketSK2;
	CMySocket1 m_sConnectSocketSK1;
	CMySocket1 m_sListenSocketSK1;
	void PlotResponseCurve();
};

//{{AFX_INSERT_LOCATION}}
```

// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif //
!defined(AFX_IODNM1DLG_H__B1A80DE6_87CB_11D6_8435_00010318EDFF__INCLUDED_)

```
// iodnm1Dlg.cpp : implementation file
//

#include "stdafx.h"
#include "iodnm1.h"
#include "iodnm1Dlg.h"
#include "conio.h"
#include <afxtempl.h>
#include "math.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define BASE 0x300

CArray<float, float> temp_fblist;



void PWMbas(float frequency, float duty);
void InitializePWM();
void InitializeIRC();
int readIRC();
void resetIRC();

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();
```

```
// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
```

```
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CIodnm1Dlg dialog

CIodnm1Dlg::CIodnm1Dlg(CWnd* pParent /*=NULL*/)
	: CDialog(CIodnm1Dlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CIodnm1Dlg)
	m_frequency = 0.0f;
	m_duty = 0.0f;
	m_irc = 0;
	m_kp = 0.0f;
	m_ki = 0.0f;
	m_kd = 0.0f;
	m_freq_of_reference = 0.0f;
	m_amplitude_of_ref = 0.0f;
	m_totalexptime = 0.0f;
	m_ew = 0.0f;
	m_status = _T("");
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CIodnm1Dlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CIodnm1Dlg)
	DDX_Text(pDX, IDC_EFREQ, m_frequency);
	DDX_Text(pDX, IDC_EDUTY, m_duty);
	DDX_Text(pDX, IDC_IRC, m_irc);
```

```
	DDX_Text(pDX, IDC_EKP, m_kp);
	DDX_Text(pDX, IDC_EKI, m_ki);
	DDX_Text(pDX, IDC_EKD, m_kd);
	DDX_Text(pDX, IDC_EFREQUENCY, m_freq_of_reference);
	DDX_Text(pDX, IDC_EAMPLITUDE, m_amplitude_of_ref);
	DDX_Text(pDX, IDC_ETOTALEXPTIME, m_totalexptime);
	DDX_Text(pDX, IDC_EW, m_ew);
	DDX_Text(pDX, IDC_ESTATUS, m_status);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CIodnm1Dlg, CDialog)
	//{{AFX_MSG_MAP(CIodnm1Dlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BUPDATE, OnBupdate)
	ON_BN_CLICKED(IDC_INITIALIZEPWM, OnInitializepwm)
	ON_BN_CLICKED(IDC_BREADIRC, OnBreadirc)
	ON_BN_CLICKED(IDC_BSTARTPID, OnBstartpid)
	ON_BN_CLICKED(IDC_BLISTEN, OnBlisten)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CIodnm1Dlg message handlers

BOOL CIodnm1Dlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
```

```
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here
	m_frequency = 20000;
	m_duty = 0.5;
	m_kp = 21.8;
	m_ki = 1.0;
	m_kd = 0.001;
	m_amplitude_of_ref = 50.0;
	UpdateData(FALSE);

	m_sListenSocketSK1.SetParent(this);
	m_sConnectSocketSK1.SetParent(this);
	m_sListenSocketSK2.SetParent(this);
	m_sConnectSocketSK2.SetParent(this);
```

```
	m_ccmddefault.cmdname = 1;
	m_ccmddefault.arg1 = 0;

	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CIodnm1Dlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CIodnm1Dlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM)
dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
```

```
			int cxIcon = GetSystemMetrics(SM_CXICON);
			int cyIcon = GetSystemMetrics(SM_CYICON);
			CRect rect;
			GetClientRect(&rect);
			int x = (rect.Width() - cxIcon + 1) / 2;
			int y = (rect.Height() - cyIcon + 1) / 2;

			// Draw the icon
			dc.DrawIcon(x, y, m_hIcon);
		}
		else
		{
			CDialog::OnPaint();
		}
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CIodnm1Dlg::OnQueryDragIcon()
{
		return (HCURSOR) m_hIcon;
}

void CIodnm1Dlg::OnBupdate()
{
		// TODO: Add your control notification handler code here
		UpdateData(TRUE);
		PWMbas(m_frequency, m_duty);
}

void CIodnm1Dlg::OnInitializepwm()
{
		// TODO: Add your control notification handler code here
		InitializePWM();
```

```
	PWMbas(20000, 0.5);
	InitializeIRC();
}

void CIodnm1Dlg::OnBreadirc()
{
	// TODO: Add your control notification handler code here
	m_irc = readIRC();
	UpdateData(FALSE);
}

void InitializePWM()
{
	_outp(BASE+1, 0xFF);		// master reset
	_outp(BASE+1, 0x5F);		// resets all counters loading 0x0000 from load
registers



	_outp(BASE+1, 0x01);		// select counter1 mode register
	_outp(BASE, 0x62);		// select mode J duty cycle generator
	_outp(BASE, 0x0B);

	PWMbas(20000, 0.5);

	_outp(BASE+1, 0x21);		// arms counter1

}

void PWMbas(float frequency, float duty)
{
	float LOADREGLO, LOADREGHI, HOLDREGLO, HOLDREGHI;
	double load, hold;
	float F1 = 2e7;
```

```
	load = (duty*F1)/frequency;
	hold = (F1/frequency)*(1-duty);

	LOADREGLO = (int)load % 256;
	LOADREGHI = ((int)load / 256);

	HOLDREGLO = (int)hold % 256;
	HOLDREGHI = ((int)hold / 256);



	_outp(BASE+1, 0x09);		// select counter1 load register
	_outp(BASE, LOADREGLO);
	_outp(BASE, LOADREGHI);

	_outp(BASE+1, 0x11);		// select counter1 hold register
	_outp(BASE, HOLDREGLO);
	_outp(BASE, HOLDREGHI);

	_outp(BASE+1, 0x41);		// loads and arms counter1
}

void InitializeIRC()
{
	_outp(BASE+0x11, 0x01);	// reset BP

	_outp(BASE+0x10, 0);		// PR0 = 0

	_outp(BASE+0x11, 0x18);	// PR0 -> PSC
	_outp(BASE+0x11, 0x01);	// reset BP
	_outp(BASE+0x10, 0x00);	// reset PR
	_outp(BASE+0x10, 0x00);
	_outp(BASE+0x10, 0x00);
	_outp(BASE+0x11, 0x08);	// PR -> CNTR
	_outp(BASE+0x11, 0x38);	// CMR
```

```
	_outp(BASE+0x11, 0x41);	// IOR
	_outp(BASE+0x11, 0x65);	// IDR
}

int readIRC()
{
	int irc=0;
	_outp(BASE+0x11, 0x11);	// CNTR -> OL, reset BP
	irc = _inp(BASE+0x10);
	irc+= _inp(BASE+0x10)<<8;
	irc+= _inp(BASE+0x10)<<16;
	return(irc);
}

void resetIRC()
{
	_outp(BASE+0x11, 0x01);	// reset BP
	_outp(BASE+0x10, 0x00);	// reset PR
	_outp(BASE+0x10, 0x00);
	_outp(BASE+0x10, 0x00);
	_outp(BASE+0x11, 0x08);	// PR -> CNTR
}

void CIodnm1Dlg::OnBstartpid()
{
	// TODO: Add your control notification handler code here
	UpdateData(TRUE);

	clock_t prevtime,newtime, prevtime_long, newtime_long;
	float elapsedtime=0, elapsedtime_long=0;
	float w=0, wp=0, wac=0, wacp=0, wref = 0;
	float e=0, ep=0, et=0, ed=0;

	int number_of_samples=0;
```

```
float experiment_total_time = 5.0;              // in seconds
float experiment_sampling_time = 0.01;      // in seconds
number_of_samples = experiment_total_time / experiment_sampling_time;

float ppr = 3600;
float pulse_to_w_coef=0;

float Kp=0, Ki=0, Kd=0;
float u=0.0;

int i=0;
long i1=0;

temp_fblist.RemoveAll();
temp_fblist.FreeExtra();

UpdateData(TRUE);

wref = m_amplitude_of_ref;

Kp = m_kp;
Ki=m_ki;
Kd=m_kd;

pulse_to_w_coef = ((1/ppr)/experiment_sampling_time)*6.28;

InitializePWM();
InitializeIRC();

prevtime_long = clock();
for (i=0; i<number_of_samples; i++)
{
        // SAMPLING CLOCK //
        for (i1=0; i1<1600000; i1++)
```

```
                ;
        // end of sampling clock //*/


        // READING W //
        w = readIRC();
        wac = w - wp;
        wac = pulse_to_w_coef*wac;
        if (wac>90)
                wac = wacp;
        if (wac<-90)
                wac = wacp;
        wp = w;
        wacp = wac;

        temp_fblist.Add(wac);
        // end of reading w //

        // CALCULATING ERROR //
e = wref - wac;
        et = et + e*experiment_sampling_time;
ed = (e - ep)/experiment_sampling_time;
        ep = e;
        // end of calculating error //

        // PID CONTROLLER //
        u = Kp*e + Ki*et + Kd*ed;
        //u = 0.9;
        // end of PID controller //
        u = -u;

        if (u>1)
                u = 1;
        if (u<-1)
```

```
                u = -1;

            u = (u+1)/2;

            if (u>1)
                u = 1;
            if (u<0)
                u = 0;

            PWMbas(20000.0, u);

        }

        newtime_long = clock();
        elapsedtime_long = (double)(newtime_long-
prevtime_long)/CLOCKS_PER_SEC;
        m_totalexptime = elapsedtime_long;
        UpdateData(FALSE);

        PWMbas(20000, 0.5);

        PlotResponseCurve();
}

void CIodnm1Dlg::PlotResponseCurve()
{
        this->RedrawWindow();

        int i=0, i1=0;
        float n=0.0;
        CClientDC dc(this);
        int number_of_sample_points = 500;

        CPoint initial_point;
```

```
initial_point.x = 40;
initial_point.y = 550;
dc.MoveTo(initial_point.x, initial_point.y);

CPen lPenBlackWide(PS_SOLID, 2, RGB(0,0,0));
dc.SelectObject(&lPenBlackWide);
dc.MoveTo(initial_point.x ,initial_point.y);
dc.LineTo(550+initial_point.x,initial_point.y);
dc.MoveTo(initial_point.x ,initial_point.y);
dc.LineTo(initial_point.x ,initial_point.y - 260);

dc.TextOut(initial_point.x +100 ,initial_point.y+2, "1");
dc.TextOut(initial_point.x +200 ,initial_point.y+2, "2");
dc.TextOut(initial_point.x +300 ,initial_point.y+2, "3");
dc.TextOut(initial_point.x +400 ,initial_point.y+2, "4");
dc.TextOut(initial_point.x +500 ,initial_point.y+2, "5");
dc.TextOut(initial_point.x +550 ,initial_point.y+2, "time in secs");

dc.TextOut(initial_point.x -35 ,initial_point.y-50-10, "20");
dc.TextOut(initial_point.x -35 ,initial_point.y-100-10, "40");
dc.TextOut(initial_point.x -35 ,initial_point.y-150-10, "60");
dc.TextOut(initial_point.x -35 ,initial_point.y-200-10, "80");
dc.TextOut(initial_point.x -35 ,initial_point.y-250-10, "100");
dc.TextOut(initial_point.x -35 ,initial_point.y-280-10, "w in rad/secs");

CPen lPenGrid(PS_DOT, 1, RGB(0,0,0));
dc.SelectObject(&lPenGrid);
for (i1=0; i1<56; i1++)
{
        dc.MoveTo(initial_point.x +10*i1 ,initial_point.y);
        dc.LineTo(initial_point.x +10*i1 ,initial_point.y-260);
}
for (i1=0; i1<27; i1++)
{
```

```
            dc.MoveTo(initial_point.x ,initial_point.y -10*i1);
            dc.LineTo(initial_point.x +550 ,initial_point.y -10*i1);
        }

        CPen lPenBlack(PS_SOLID, 1, RGB(0,0,0));
        dc.SelectObject(&lPenBlack);
        for (i1=0; i1<6; i1++)
        {
            dc.MoveTo(initial_point.x +550 ,initial_point.y -50*i1);
            dc.LineTo(initial_point.x -10 ,initial_point.y -50*i1);
        }
        for (i1=0; i1<6; i1++)
        {
            dc.MoveTo(initial_point.x + 100*i1,initial_point.y +5);
            dc.LineTo(initial_point.x + 100*i1 ,initial_point.y -260);
        }
        ///////// end of initialization of the plot ///////////////

        // DRAWING THE REFERENCE AND RESPONSE CURVES //////
        CPen lPenBlue(PS_SOLID, 2, RGB(0,0,255));
        dc.SelectObject(&lPenBlue);

        for (i1=0; i1<499; i1++)
        {
            n=i1;
            dc.MoveTo(initial_point.x +i1 ,initial_point.y -
2.5*temp_fblist.ElementAt(i1));
            dc.LineTo(initial_point.x +i1+1 ,initial_point.y -
2.5*temp_fblist.ElementAt(i1+1));
        }

        CPen lPenRed(PS_SOLID, 2, RGB(255,0,0));
        dc.SelectObject(&lPenRed);
```

```
	for (i1=0; i1<499; i1++)
	{
		n=i1;
		dc.MoveTo(initial_point.x +i1 ,initial_point.y -
2.5*m_amplitude_of_ref);
		dc.LineTo(initial_point.x +i1+1 ,initial_point.y -
2.5*m_amplitude_of_ref);
	}
	// end of drawing the reference and response curves //

}

void CIodnm1Dlg::OnAcceptSK1()
{
	m_sListenSocketSK1.Accept(m_sConnectSocketSK1);
	m_status = "OnAcceptSK1";
	UpdateData(FALSE);
}

void CIodnm1Dlg::OnConnectSK1()
{

}

void CIodnm1Dlg::OnCloseSK1()
{
	m_sConnectSocketSK1.Close();
	m_status = "OnCloseSK1";
	UpdateData(FALSE);
}

void CIodnm1Dlg::OnSendSK1()
{
```

```
}

void CIodnm1Dlg::OnReceiveSK1()
{
	ReceiveCTask();
	ApplyTaskandBuildFeedback();
	SendCFeedback();
}

void CIodnm1Dlg::OnAcceptSK2()
{
	m_sListenSocketSK2.Accept(m_sConnectSocketSK2);
	m_status = "OnAcceptSK2";
	UpdateData(FALSE);
}

void CIodnm1Dlg::OnConnectSK2()
{

}

void CIodnm1Dlg::OnCloseSK2()
{
	m_sConnectSocketSK2.Close();
	m_status = "OnCloseSK2";
	UpdateData(FALSE);
}

void CIodnm1Dlg::OnSendSK2()
{

}

void CIodnm1Dlg::OnReceiveSK2()
```

```
{

}

void CIodnm1Dlg::OnBlisten()
{
	// TODO: Add your control notification handler code here
	m_sListenSocketSK1.Create(4000);
	m_sListenSocketSK1.Listen();

	m_sListenSocketSK2.Create(4001);
	m_sListenSocketSK2.Listen();
}

void CIodnm1Dlg::ReceiveCTask()
{
	int iRcvd;
	int number_of_commands;
	CCmd tmpcmd = m_ccmddefault;
	CCmd *tempcmd = &tmpcmd;
	int tmpint = 1;
	int *temptaskvariables = &tmpint;

	// receiving the taskid
	do
	{
		iRcvd = m_sConnectSocketSK1.Receive(temptaskvariables,
sizeof(*temptaskvariables));
	} while (iRcvd == SOCKET_ERROR);
	m_setuppc_Task.TaskID = (*temptaskvariables);

	// receiving the tasksource
	do
	{
```

```
		iRcvd = m_sConnectSocketSK1.Receive(temptaskvariables,
sizeof(*temptaskvariables));
	} while (iRcvd == SOCKET_ERROR);
	m_setuppc_Task.source = (*temptaskvariables);

	// receiving the taskdestination
	do
	{
		iRcvd = m_sConnectSocketSK1.Receive(temptaskvariables,
sizeof(*temptaskvariables));
	} while (iRcvd == SOCKET_ERROR);
	m_setuppc_Task.destination = (*temptaskvariables);

	// receiving the taskpriority
	do
	{
		iRcvd = m_sConnectSocketSK1.Receive(temptaskvariables,
sizeof(*temptaskvariables));
	} while (iRcvd == SOCKET_ERROR);
	m_setuppc_Task.priority = (*temptaskvariables);

	// receiving the taskaccessrank
	do
	{
		iRcvd = m_sConnectSocketSK1.Receive(temptaskvariables,
sizeof(*temptaskvariables));
	} while (iRcvd == SOCKET_ERROR);
	m_setuppc_Task.accessrank = (*temptaskvariables);

	// receiving the number of commands
	do
	{
		iRcvd = m_sConnectSocketSK1.Receive(temptaskvariables,
sizeof(*temptaskvariables));
```

```
	} while (iRcvd == SOCKET_ERROR);
	number_of_commands = *temptaskvariables;

	// receiving the commands
	for (int k=1; k<number_of_commands+1; k++)
	{
		do
		{
			iRcvd = m_sConnectSocketSK1.Receive(tempcmd,
sizeof(*tempcmd));
		} while (iRcvd == SOCKET_ERROR);
	m_setuppc_Task.cmdlist.SetAtGrow(k-1, *tempcmd);
	}
	m_status = "RECEIVED!!!!";
	UpdateData(FALSE);
}

void CIodnm1Dlg::ApplyTaskandBuildFeedback()
{
	int i=0;
	int size=0;
	CCmd cmd;

	m_setuppc_Feedback.FeedbackID = m_setuppc_Task.TaskID;		// this must
be changed
	m_setuppc_Feedback.source = m_setuppc_Task.destination;		// this must
be changed
	m_setuppc_Feedback.destination = m_setuppc_Task.source;
	m_setuppc_Feedback.priority = m_setuppc_Task.priority;
	m_setuppc_Feedback.task_accepted = CheckSafe();
	if ( CheckSafe() == 1 )
	{
		size = m_setuppc_Task.cmdlist.GetSize();
		for (i=0; i<size; i++)
```

```
                {
                        // Applycmdandbuildfb
                        cmd = m_setuppc_Task.cmdlist.ElementAt(i);
                        Applycmdandbuildfb(cmd);
                }
        }
}

int CIodnm1Dlg::CheckSafe()
{
        return 1; // if task is safe else return 0
}

CFb CIodnm1Dlg::Applycmdandbuildfb(CCmd cmd1)
{
        CFb fb;
        fb.fbname=0;
        fb.arg1=0;
        int controller_type=0, reference_type=0;
        float Kp=0,Ki=0,Kd=0,Kfr=0,amplitude=0,frequency=0;
        switch (cmd1.cmdname)
        {
        case 1:
                m_controller_type = cmd1.arg1;
                break;
        case 2:
                m_reference_type = cmd1.arg1;
                break;
        case 3:
                m_kp = cmd1.arg1;
                break;
        case 4:
                m_ki = cmd1.arg1;
                break;
```

```
	case 5:
		m_kd = cmd1.arg1;
		break;
	case 6:
		m_kfr = cmd1.arg1;
		break;
	case 7:
		m_amplitude_of_ref = cmd1.arg1;
		break;
	case 8:
		m_freq_of_reference = cmd1.arg1;
		break;
	case 9:
		ApplyControl(m_controller_type, m_reference_type, m_kp, m_ki, m_kd,
m_kfr, m_amplitude_of_ref, m_freq_of_reference);
		break;
	default:
		break;
	}
	return fb;
}

void CIodnm1Dlg::ApplyControl(int c_type, int r_type, float kp, float ki, float kd, float
kfr, float a, float f)
{
	float w=0, wp=0, wac=0, wacp=0;
	float e=0, ep=0, et=0, ed=0;

	int number_of_samples=0;
	float experiment_total_time = 5.0;		// in seconds
	float experiment_sampling_time = 0.01;	// in seconds
	number_of_samples = experiment_total_time / experiment_sampling_time;

	float ppr = 3600;
```

```
float pulse_to_w_coef=0;

float Kp=0, Ki=0, Kd=0, Kfr=0;
float u=0.0;

int i=0;
long i1=0;
int i2=0;

CFb fb;

CArray<float, float> wref;
temp_fblist.RemoveAll();
temp_fblist.FreeExtra();
m_setuppc_Feedback.fblist.RemoveAll();
m_setuppc_Feedback.fblist.FreeExtra();

if (r_type == 0)
{
	for (i2=0; i2<number_of_samples; i2++)
	{
		wref.Add(a);
	}
}
if (r_type == 1)
{
	for (i2=0; i2<number_of_samples; i2++)
	{
		wref.Add(a*sin(f*6.28*experiment_sampling_time));
	}
}

Kp = kp;
Ki = ki;
```

```
Kd = kd;
Kfr = kfr;

pulse_to_w_coef = ((1/ppr)/experiment_sampling_time)*6.28;

InitializePWM();
InitializeIRC();

for (i=0; i<number_of_samples; i++)
{
        // SAMPLING CLOCK //
        for (i1=0; i1<2000000; i1++)
                ;
        // end of sampling clock //*/

        // READING W //
        w = readIRC();
        wac = w - wp;
        wac = pulse_to_w_coef*wac;
        if (wac>90)
                wac = wacp;
        if (wac<-90)
                wac = wacp;
        wp = w;
        wacp = wac;

        fb.fbname = 9;
        fb.arg1 = wac;
        m_setuppc_Feedback.fblist.Add(fb);
        // end of reading w //

        // CALCULATING ERROR //
    e = wref.ElementAt(i) - wac;
        et = et + e*experiment_sampling_time;
```

```
ed = (e - ep)/experiment_sampling_time;
        ep = e;
        // end of calculating error //

        // PID CONTROLLER //
        if (c_type == 0)
        {
                u = Kp*wref.ElementAt(i);
        }
        if (c_type == 1)
        {
                u = Kp*wref.ElementAt(i)+Kfr;
        }
        if (c_type == 2)
        {
                u = Kp*e + Ki*et + Kd*ed;
        }
        // end of PID controller //
        u = -u;

        if (u>1)
                u = 1;
        if (u<-1)
                u = -1;

        u = (u+1)/2;

        if (u>1)
                u = 1;
        if (u<0)
                u = 0;

        PWMbas(20000.0, u);
```

```
	}

	PWMbas(20000, 0.5);

}

void CIodnm1Dlg::SendCFeedback()
{
	int iSent;
	CFb tmpfb = m_cfbdefault;
	CFb *tempfb = &tmpfb;
	int tmpint = 1;
	int *tempfeedbackvariables = &tmpint;

	// sending the feedbackid
	*tempfeedbackvariables = m_setuppc_Feedback.FeedbackID;
	do
	{
		iSent = m_sConnectSocketSK2.Send(tempfeedbackvariables,
sizeof(*tempfeedbackvariables));
	} while (iSent == SOCKET_ERROR);

	// sending the feedbacksource
	*tempfeedbackvariables = m_setuppc_Feedback.source;
	do
	{
		iSent = m_sConnectSocketSK2.Send(tempfeedbackvariables,
sizeof(*tempfeedbackvariables));
	} while (iSent == SOCKET_ERROR);

	// sending the taskdestination
	*tempfeedbackvariables = m_setuppc_Feedback.destination;
	do
	{
```

```
                iSent = m_sConnectSocketSK2.Send(tempfeedbackvariables,
sizeof(*tempfeedbackvariables));
        } while (iSent == SOCKET_ERROR);

        // sending the taskpriority
        *tempfeedbackvariables = m_setuppc_Feedback.priority;
        do
        {
                iSent = m_sConnectSocketSK2.Send(tempfeedbackvariables,
sizeof(*tempfeedbackvariables));
        } while (iSent == SOCKET_ERROR);

        // sending the taskaccessrank
        *tempfeedbackvariables = m_setuppc_Feedback.task_accepted;
        do
        {
                iSent = m_sConnectSocketSK2.Send(tempfeedbackvariables,
sizeof(*tempfeedbackvariables));
        } while (iSent == SOCKET_ERROR);

        // sending the number of commands
        *tempfeedbackvariables = m_setuppc_Feedback.fblist.GetSize();
        do
        {
                iSent = m_sConnectSocketSK2.Send(tempfeedbackvariables,
sizeof(*tempfeedbackvariables));
        } while (iSent == SOCKET_ERROR);

        // sending the commands
        for (int k=1; k<m_setuppc_Feedback.fblist.GetSize()+1 ; k++)
        {
                *tempfb = m_setuppc_Feedback.fblist.GetAt(k-1);
                do
                {
```

```
                iSent = m_sConnectSocketSK2.Send(tempfb, sizeof(*tempfb));
            } while (iSent == SOCKET_ERROR);
        }
    }
```

```
#if
!defined(AFX_MYSOCKET1_H__51411EA6_8966_11D6_8435_00010318EDFF__I
NCLUDED_)
#define
AFX_MYSOCKET1_H__51411EA6_8966_11D6_8435_00010318EDFF__INCLUDE
D_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// MySocket1.h : header file
//



/////////////////////////////////////////////////////////////////////////////
// CMySocket1 command target

class CMySocket1 : public CAsyncSocket
{
// Attributes
public:

// Operations
public:
        CMySocket1();
        virtual ~CMySocket1();

// Overrides
public:
        void SetParent(CDialog* pWnd);
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMySocket1)
        //}}AFX_VIRTUAL
```

```
	// Generated message map functions
	//{{AFX_MSG(CMySocket1)
		// NOTE - the ClassWizard will add and remove member functions here.
	//}}AFX_MSG

// Implementation
protected:
	virtual void OnReceive(int nErrorCode);
	virtual void OnSend(int nErrorCode);
	virtual void OnClose(int nErrorCode);
	virtual void OnConnect(int nErrorCode);
	virtual void OnAccept(int nErrorCode);
private:
	CDialog* m_pWnd;
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_MYSOCKET1_H__51411EA6_8966_11D6_8435_00010318EDFF__I
NCLUDED_)
```

```
// MySocket1.cpp : implementation file
//

#include "stdafx.h"
#include "iodnm1.h"
#include "MySocket1.h"
#include "iodnm1Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMySocket1

CMySocket1::CMySocket1()
{
}

CMySocket1::~CMySocket1()
{
}


// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CMySocket1, CAsyncSocket)
	//{{AFX_MSG_MAP(CMySocket1)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif  // 0
```

```
//////////////////////////////////////////////////////////////////////////
// CMySocket1 member functions

void CMySocket1::SetParent(CDialog *pWnd)
{
	m_pWnd = pWnd;
}

void CMySocket1::OnAccept(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CIodnm1Dlg*)m_pWnd)->OnAcceptSK1();
	}
}

void CMySocket1::OnConnect(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CIodnm1Dlg*)m_pWnd)->OnConnectSK1();
	}
}

void CMySocket1::OnClose(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CIodnm1Dlg*)m_pWnd)->OnCloseSK1();
	}
}

void CMySocket1::OnSend(int nErrorCode)
{
```

```
	if (nErrorCode == 0)
	{
		((CIodnm1Dlg*)m_pWnd)->OnSendSK1();
	}
}

void CMySocket1::OnReceive(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CIodnm1Dlg*)m_pWnd)->OnReceiveSK1();
	}
}
```

```
#if
!defined(AFX_MYSOCKET2_H__51411EA8_8966_11D6_8435_00010318EDFF__I
NCLUDED_)
#define
AFX_MYSOCKET2_H__51411EA8_8966_11D6_8435_00010318EDFF__INCLUDE
D_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// MySocket2.h : header file
//



/////////////////////////////////////////////////////////////////////////////
// CMySocket2 command target

class CMySocket2 : public CAsyncSocket
{
// Attributes
public:

// Operations
public:
        CMySocket2();
        virtual ~CMySocket2();

// Overrides
public:
        void SetParent(CDialog* pWnd);
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMySocket2)
        //}}AFX_VIRTUAL
```

```
	// Generated message map functions
	//{{AFX_MSG(CMySocket2)
		// NOTE - the ClassWizard will add and remove member functions here.
	//}}AFX_MSG

// Implementation
protected:
	virtual void OnReceive(int nErrorCode);
	virtual void OnSend(int nErrorCode);
	virtual void OnClose(int nErrorCode);
	virtual void OnConnect(int nErrorCode);
	virtual void OnAccept(int nErrorCode);
private:
	CDialog* m_pWnd;
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_MYSOCKET2_H__51411EA8_8966_11D6_8435_00010318EDFF__I
NCLUDED_)
```

```
// MySocket2.cpp : implementation file
//

#include "stdafx.h"
#include "iodnm1.h"
#include "MySocket2.h"
#include "iodnm1Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMySocket2

CMySocket2::CMySocket2()
{
}

CMySocket2::~CMySocket2()
{
}


// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CMySocket2, CAsyncSocket)
	//{{AFX_MSG_MAP(CMySocket2)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif  // 0
```

```
//////////////////////////////////////////////////////////////////////////
// CMySocket2 member functions

void CMySocket2::SetParent(CDialog *pWnd)
{
	m_pWnd = pWnd;
}

void CMySocket2::OnAccept(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CIodnm1Dlg*)m_pWnd)->OnAcceptSK2();
	}
}

void CMySocket2::OnConnect(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CIodnm1Dlg*)m_pWnd)->OnConnectSK2();
	}
}

void CMySocket2::OnClose(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CIodnm1Dlg*)m_pWnd)->OnCloseSK2();
	}
}

void CMySocket2::OnSend(int nErrorCode)
{
```

```
	if (nErrorCode == 0)
	{
		((CIodnm1Dlg*)m_pWnd)->OnSendSK2();
	}
}

void CMySocket2::OnReceive(int nErrorCode)
{
	if (nErrorCode == 0)
	{
		((CIodnm1Dlg*)m_pWnd)->OnReceiveSK2();
	}
}
```

```
// Cmd.h: interface for the CCmd class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_CMD_H__51411EA9_8966_11D6_8435_00010318EDFF__INCLUDE
D_)
#define
AFX_CMD_H__51411EA9_8966_11D6_8435_00010318EDFF__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CCmd
{
public:
	int cmdname;
	float arg1;
	CCmd();
	virtual ~CCmd();

};

#endif //
!defined(AFX_CMD_H__51411EA9_8966_11D6_8435_00010318EDFF__INCLUDE
D_)
```

```
// Cmd.cpp: implementation of the CCmd class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "iodnm1.h"
#include "Cmd.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CCmd::CCmd()
{

}

CCmd::~CCmd()
{

}
```

```
// Fb.h: interface for the CFb class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_FB_H__51411EAA_8966_11D6_8435_00010318EDFF__INCLUDED
_)
#define AFX_FB_H__51411EAA_8966_11D6_8435_00010318EDFF__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CFb
{
public:
	int fbname;
	float arg1;
	CFb();
	virtual ~CFb();

};

#endif //
!defined(AFX_FB_H__51411EAA_8966_11D6_8435_00010318EDFF__INCLUDED
_)
```

```
// Fb.cpp: implementation of the CFb class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "iodnm1.h"
#include "Fb.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CFb::CFb()
{

}

CFb::~CFb()
{

}
```

```
// Task.h: interface for the CTask class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_TASK_H__51411EAB_8966_11D6_8435_00010318EDFF__INCLUD
ED_)
#define
AFX_TASK_H__51411EAB_8966_11D6_8435_00010318EDFF__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <afxtempl.h>
#include "Cmd.h"

class CTask
{
public:
	int accessrank;
	int priority;
	int destination;
	int source;
	int TaskID;
	CTask();
	virtual ~CTask();
	CArray<CCmd, CCmd> cmdlist;

};

#endif //
!defined(AFX_TASK_H__51411EAB_8966_11D6_8435_00010318EDFF__INCLUD
ED_)
```

```
// Task.cpp: implementation of the CTask class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "iodnm1.h"
#include "Task.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CTask::CTask()
{

}

CTask::~CTask()
{

}
```

```
// Feedback.h: interface for the CFeedback class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_FEEDBACK_H__51411EAC_8966_11D6_8435_00010318EDFF__IN
CLUDED_)
#define
AFX_FEEDBACK_H__51411EAC_8966_11D6_8435_00010318EDFF__INCLUDED
_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <afxtempl.h>
#include "Fb.h"

class CFeedback
{
public:
	int task_accepted;
	int priority;
	int destination;
	int source;
	int FeedbackID;
	CArray<CFb, CFb> fblist;
	CFeedback();
	virtual ~CFeedback();

};
```

```
#endif //
!defined(AFX_FEEDBACK_H__51411EAC_8966_11D6_8435_00010318EDFF__IN
CLUDED_)
```

```
// Feedback.cpp: implementation of the CFeedback class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "iodnm1.h"
#include "Feedback.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CFeedback::CFeedback()
{

}

CFeedback::~CFeedback()
{

}
```

REFERENCES

[1] http://www.ece.cmu.edu/~stancil/virtual-lab/concept.html

[2] Distance Learning Applied to Control Engineering Laboratories, Burçin Aktan, Carisa A. Bohus, Lawrence A. Crowl, and Molly H. Shor, IEEE Transactions on Education, Vol 39, No. 3, August 1996, 320-326

[3] Simulation workshop and remote laboratory: two web-based training approaches for Control, M. Exel, S. Gentil, F. Michau and D. Rey, Proceedings of the American Control Conference, Chicago, Illinois, June 2000

[4] A Measurement Laboratory on Geographic Network for Remote Test Experiments, P. Arpaia, A. Baccigalupi, F. Cennamo, P. Daponte, IEEE Instrumentation and Measurement Technology Conference, St. Paul, Minnesota (USA), May 18-21, 1998

[5] An Internet-Based Real-Time Control Engineering Laboratory, Jamahl W. Overstreet, Antony Tzes, IEEE Control Systems, October 1999, 19-34

[6] Remote Laboratory Experimentation, Mohammed Shaheen, Kenneth A. Loparo and Marcus R. Buchner, Proceedings of the American Control Conference, Philadelphia, Pennsylvania, June 1998