LONDON IMPERIAL COLLEGE OF SCIENCE,
TECHNOLOGY AND MEDICINE

INFORMATION SYSTEMS ENGINEERING
4TH YEAR

***By:*** ***Hong-Siang Teo, ISE 4***

***Supervisor:*** ***Dr. Jeremy Pitt, EEE Department***

# INDIVIDUAL PROJECT REPORT

---

## INTERNET-CONTROLLED ROBOT

# TABLE OF CONTENTS

## PART I: INTRODUCTION



## PART II: DESIGN

## PART III: IMPLEMENTATION



## PART IV: EVALUATION



## PART V: CONCLUSION

# CHAPTER 1: OVERVIEW

The purpose of this project is to investigate the possibility of Tele-Robotics through the use of popular and established Internet protocols and state-of-the-art cellular mobile telephones. As an application, a remote-controlled robot system will be designed and built. The robot is connected to the Internet via a portable computer equipped with a mobile phone, while the remote controller accesses the robot through an application running on a popular World-Wide-Web (WWW) browser on his computer.

Several state-of-the-art technologies will be investigated in the course of realising this project: Linux is arguably the most exciting operating system currently available; networking techniques like Mobile-IP are very much in their infant stages; network programming using Java; hardware like GSM mobile phones and CCD video modules; as well as traditional robotics techniques. All these are to be integrated into a WWW access system.

This report has been organized into 5 parts, roughly indicative of the progress of this project.

- **Part I** is the introduction section, which provides some background information to the project, spells out the objectives to be reached, as well as a milestone chart describing the progress.

- **Part II** describes the overall design of the project, where a system architecture to facilitate implementation is proposed. Following this, several key problems will be identified, and possible solutions will be discussed. Finally, the system specification is laid down.

- **Part III** deals with the implementation specifics, covering every part of the system. Implementation is roughly divided into three areas: networking, software and hardware.

- **Part IV** documents the evaluation stage, where the performance and limitations of the system is gauged. Interesting results and

conclusions obtained from testing are also reported.

- Finally, **Part V** concludes this report, with a view on areas of potential future development and research.

# CHAPTER 2: MOTIVATIONS

## NEW TECHNOLOGIES

Recent advances in 3 key areas of technology have revolutionalized our concept of computing, namely the *Internet*, *portable computers* and *mobile cellular telephones*. The current decade has witnessed the Internet taking the whole world by storm. We can now communicate with people all around the globe, and have access to huge amounts of on-line multimedia resources as if they are on our own hard disks. More recently, the technology behind the Internet has also benefited the networking world in general. Through the idea of the Intranet, office computing has become more streamlined and efficient. From another perspective, these networking technologies can replace current proprietary networking techniques used in custom networked environment, while offering higher reliability and maintenance-free implementations.

With miniaturisation and high-scale integration, portable computers have become as powerful as any desktop workstation today, yet small enough to be used in a wider range of applications where mobility is important. Where the bulk of yesterday's computers fill an entire room, today's portable Hercules can be piggy-backed onto small robots, thus providing much more functionality than simple micro-controllers ever can.

With mobile telephones, we now have the potential to talk to people around the world anytime, anywhere. In particular, international cellular standards, such as GSM, has made a whole new range of digital services available to the mobile phone user, like emailing, paging, or even direct hook-ups to the Internet, all at the cheap cost of a local call.

By combining these technologies, a diverse variety of computing and control applications can be enhanced with added reliability, features and reach. New and revolutionary uses emerge everyday. Arguably, the field of Tele-Robotics stands to benefit the most from a marriage of these technologies. However, exploits based on this concept have not been widespread. Hence, this project aims to make use such a combination of technology, to improve and enhance the concept of remote controlling a robot.

## BACKGROUND

The concept of this project was motivated by the popularisation of the Internet, mobile phones and portable computers. To have a clearer perspective on this, consider a traditional remote-controlled robot, which is already a much-researched topic by both the academics and especially the hobbyists. However, 3 areas of implementation still give room for improvement:

1. Almost all the major processing functions of the robot are usually performed by a single central micro-controller like the popular Motorola M68HC11. Working on its own, its functionality is limited to only the most basic and low-level routines. For more complex functions, program development becomes difficult. If interfaced to a nearby host computer, mobility will be compromised.

   *With today's powerful yet lightweight portable computers, much higher local intelligence can be embedded onto the robot. Such intelligence can take the form of enhanced peripheral capability, or artificial intelligence (AI). For a mobile robot, AI can be especially important in assisting the controller to guide the robot about its tasks. Of course, low level hardware interfacing will probably still need to be delegated to micro-controllers, but they can then be of much simpler and cheaper variety, and program development on them will be straight forward.*

2. The transmission medium for the remote control is typically an analogue radio-wave channel, or a low-quality digital channel. The transmission protocols may well be proprietary implementations too. These techniques usually suffer from either high interference, low range or low reliability, the extents of which are largely dependent on the competence of the robot builder and/or budget. Moreover, proprietary techniques can have a high maintenance cost.

   *GSM mobile phones provide virtually worldwide reach at a cost almost independent of transmission range (when using local digital services). Yet interfacing them to computers is now a standard affair through commonly available Data Cards. What this gives us is a long range, highly reliable, and ready to use transmission medium, with almost no maintenance cost on our part.*

*Internet transmission protocols are well defined and proven reliable, and constantly maintained and improved by more capable bodies. Moreover, due to the layered networking structure[1], the programmer will only need to be concerned with application layer interfacing.*

3. More often than not, the term 'mobility' of the robot is taken with regards to physical movement facilities. If the robot is able to roam around the room, then it is considered 'mobile'.

*By making use of GSM mobile phones, the idea of 'mobility' takes on a whole new concept - that of worldwide location independence, in addition to its inherent wireless operation. Obviously this opens up new possibilities for Tele-Robotics.*

In summary, we see that not only does the portable computer provide the robot with much higher local intelligence, but connecting it to the Internet through a mobile phone will also increase its reach, mobility, as well as the reliability of the transmission. Moreover, if such a control system is then integrated into the WWW server and browser system, then the means of controlling the robot will be available independent of the user's location in the world, in a standardised fashion.

TO CONTROL THE ROBOT, ANYWHERE IN THE WORLD, FROM ANYWHERE IN THE WORLD.

At this point, it is also worthwhile to note that such exciting prospects can only be possible through the perhaps inevitable convergence of the Internet, mobile phone and portable computer technologies in recent years.

## OTHER APPLICATIONS

*Question: So... is it possible then to control the robot, say, on the moon ?*

*Answer: As long as you can make the long distance phone call to the moon.*

[1] The Open Systems Interconnect (OSI) reference model is the primary architectural model for inter-computer communications. It divides the problem of moving data between computers into 7 layers: Application, Presentation, Session, Transport, Network, Link and Physical.

An Internet-controlled robot with location independence can have wide implications for uses. One interesting area of development can be in *Mobile Intelligent Agents*. Currently, there has been much research on intelligent agents on the Internet, largely fuelled by the globalisation of the Internet. However, these research concentrate mainly on simple software agents distributed around the computers in the world, performing some co-operative tasks. The Internet-controlled robot can extend this idea in a physical manner: each robot can be an autonomous intelligent entity, able to interact, and indeed control, each other over the Internet.

One application of this idea is in active security monitoring, where each "Internet robot" can be an intelligent security camera, able to track personnel co-operatively. Being Internet aware, these robots can easily be part of a local network, appearing as simple information sources, much like our networked hard-disks or printers (which are indeed, technically, robots).

Another application is in intelligent networked conferencing, where the mobile phone can offer location independent connection. Instead of having a fixed visual scope, the robot can actively track the target. Of course, for this to happen, the current data rate of the mobile phone has to be increased.

# CHAPTER 3: AIMS AND OBJECTIVES

## PROJECT OBJECTIVES

1. To design a system architecture which facilitates the implementation of a Tele-Robotics system, through the use of Internet protocols and cellular mobile phones.

2. To design and implement a remote-controlled robot, that demonstrates the use of such a system architecture.

3. To evaluate the performance and efficiency of such a system.

4. To identify key areas where further research can be undertaken.

## DELIVERABLES

1. *Mobile robot platform* - the robot should have basic navigational capabilities like forward and backward movements, rotation etc. It will also house an on-board CCD camera, through which the remote-controller can have a visual appreciation of the robot's environment. A portable computer will be piggybacked onto the platform.

2. *Low-level control logic* - this is the piece of code embedded onto a set micro-controllers situated on the robot, that performs low-level closed-loop control functions. It also interfaces with the main control program on the portable computer.

3. *Main control program* - this software runs on the portable computer, providing the interface to the hardware on the robot. It will communicate with the remote applet, streaming information to it, and receiving control information from it to be passed to the robot.

4. *Remote applet* - This is the front-end application running on the WWW browser on the remote-controlling computer, which allows the user to interact with the robot. There will at least be a visual window and a control pad.

5. *WWW server* - this server runs on the portable computer to allow the remote WWW browser to access the robot.

6. *This written project report*

# CHAPTER 4: MILESTONE CHART

| Start date | End date | Task |
|---|---|---|
| **18 Oct** | | **Project Allocation** |
| 18 Oct | 29 Oct | Read-up on relevant background topics. |
| **30 Oct** | | **start of design phase** |
| 30 Oct | 22 Dec | design phase of project |
| **22 Dec** | | **basic design of system completed** |
| 23 Dec | 5 Jan | Christmas holidays |
| **6 Jan** | | **start of implementation phase** |
| 6 Jan | 19 Jan | low-level interface to QuickCam |
| 20 Jan | 26 Jan | start work on server control program |
| 27 Jan | 12 Feb | start work on control applet |
| 13 Feb | 14 Feb | final work on interim report |
| **14 Feb** | | **submit interim report** |
| 15 Feb | 23 Feb | further work on GUI layout of applet, control pad |
| 24 Feb | 17 Mar | start work on motorised platform, mechanical chassis |
| **18 Mar** | **21 Mar** | **Humanities exams** |
| 22 Mar | 23 Mar | rest and recovery |
| 24 Mar | 27 Apr | Easter holidays/ exam revisions |
| **28 Apr** | **9 May** | **Exams** |
| 10 May | 11 May | rest and recovery |
| 12 May | 25 May | continue work on motorised |
| 26 May | 6 Jun | integration, evaluation and trouble-shooting |
| 7 Jun | 11 Jun | final work on written report |
| **12 Jun** | | **submit written report** |
| 13 Jun | 5 Jun | preparations for project presentation |
| **16 Jan** | **17 Jun** | **project presentation** |
| **19 Jun** | | **project demonstration** |
| **20 Jun** | | **end of term/year** |

♦ *Table 1: Project time-table*

# CHAPTER 5: SYSTEM DESIGN

## DESIGN GOALS

In designing the architecture of this system, as well as choosing the components that make up its implementation, it is hoped to achieve the following aims:

- *The design should be built around existing tools that are easily available.*

The emphasis should be placed on using these tools to realise the system, instead of needing to come up with proprietary tools. But when such tools are deficient in some ways, suitable modifications to them should be possible.

- *Resource requirement should be kept to a minimum.*

Resources here can be hardware resources like computers and electronic hardware, or more intangible resources like IP-addresses, operating costs etc.

- *Client-side components are platform independent.*

The term "Internet" implies heterogeneity in hardware, as well as location independence. Therefore the software should be able to be run anywhere, on any platform. On the other hand, such a requirement is not imposed on the server-side components, since invariably specific hardware need to be used. Interfacing to these hardware is inevitably hardware specific.

- *Future upgrade should be as easy as possible.*

The technology used by this project is likely to be constantly evolving. Hence, maintenance of the components should not be an overwhelming task. This implies maximum modularity in both the software and hardware components, so that any part can be upgraded with minimum effect on the rest of the system.

## DESIGN OVERVIEW

The overall design of the system architecture can be broadly divided into 5 areas:

1. *Mobile networking model*

    - how to keep track of the whereabouts of the robot.
    - how does the robot hook-up with the agent.

2. *Remote access communications*

    - how the remote station can attempt to gain control of the robot
    - how the agent transfers request to the robot
    - how the agent monitors the transactions

3. *Software components*

    - the software involved in the control of the robot
    - the programming languages used includes both the high level software and the low level control drivers

4. *Hardware components*

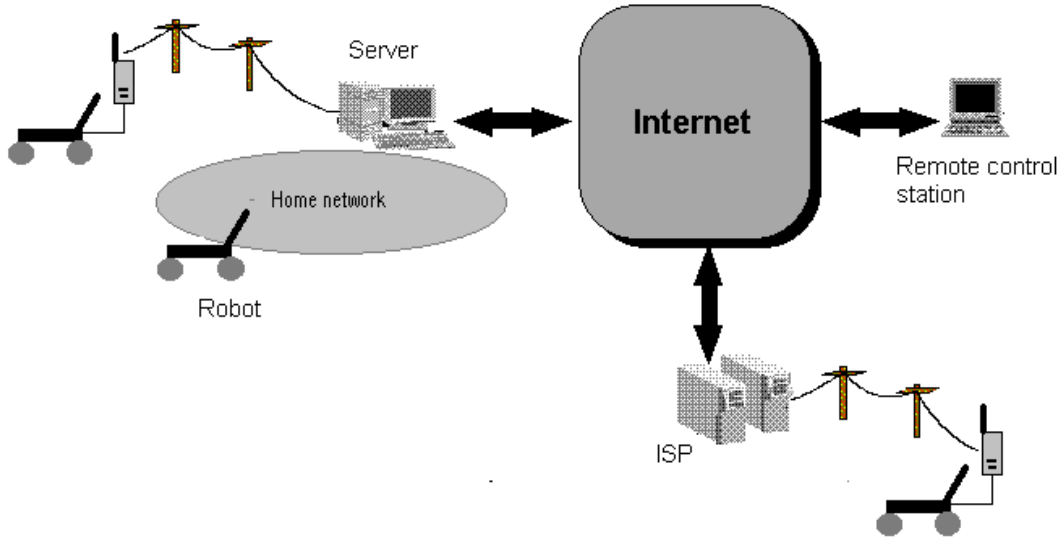
    - the proposed hardware used by each major components of the project.

5. *Robotics control and mechanics*

    - the physical interface between the laptop and the robot electronic and mechanical blueprints.

## SYSTEM OVERVIEW

FIGURE 1 shows an overview of the system design of this project :




♦ *Figure 1: System overview*

The robot has a portable computer embedded, and equipped with a GSM mobile phone. The portable computer runs a WWW server and control program, that act as the bridge between the remote control station and the robot. The remote control station is simply any computer connected to the Internet, running a popular WWW browser. To illustrate the location independence nature of the robot, it is shown in its 3 possible network configurations:

1. *Home network connection*. Here the robot is linked to its original native sub-net, through either a wired or wireless LAN connection. The important point to note here is that, since the robot is in the same domain as its Home Agent, or its router, no special networking set-up is needed to direct IP packets to the robot.

2. *Direct dial-up link*. In this configuration, the robot and the server are connected through a direct dial-up link, initiated by either of them. Since such a dial-up link is basically an extension of the home sub-net, the networking set-up is the same as the previous configuration.

3. *ISP dial-up link*. Here, the robot has physically migrated to a foreign network, and have dialled-up to a local ISP through it mobile phone. Hence it would have acquired a different IP address, and relying on the foreign gateway for routing its IP packets. However, the remote controller would not have known about it (ideally).

Nevertheless, he should still be able to access the robot without too many hassles. This calls for some pre-configured networking set-up.

In this project, the Connectix QuickCam is used as the 'eye' through which the user can obtain a visual appreciation of the robot's surroundings to aid him in his navigation of the robot. It plugs into the parallel port of the portable computer. The provision of such a visual aid is deemed necessary, because the robot may be easily beyond visual contact with the user. By offering a *first person perspective* of the headings of the robot, the user can control the robot much more confidently.

# CHAPTER 6: CORE DESIGN PROBLEMS

This chapter discusses some of the problems that arose while designing the system. Alternative solutions will be considered and discussed, which will affect the subsequent design decisions made.

## THE PROBLEM OF MOBILITY

Part of the aim of this project is to realise a location independent robot controlled through the Internet. However, wherever the robot may be, its location will still have to be described by a unique, but not necessary static, identifier. In the Internet terms, this of course refers to its Internet Protocol (IP) address. As mentioned before, this IP address need not be static. In fact, it is characteristic of the Internet domain that the robot is currently linked-up to.

This presents an interesting design problem, that concerning the mobility of the robot - mobility not only in the navigational sense, but more importantly with regards to its *physical migration*. The robot must be accessible as long as it currently has a valid IP address, regardless of the domain. In this sense, it can either hook up to a foreign network, or dial an Internet Service Provider via the mobile phone to be connected to the Internet. Obviously, there exists a problem of keeping track of the robot's current IP address, especially from the point of view of the remote-controller. *Ideally, he should be able to contact the robot with just one well-known name or IP address.*

The main difficulty lies in the configuration of the Domain Name Server (DNS) and the network Gateway. The DNS is responsible for resolving symbolic Internet names, like gray.ee.ic.ac.uk, into the native 32-bit 'dotted' IP address, while the Gateway admits IP packets destined for the hosts in its sub-net, while rejecting others, hence streamlining network traffic. Both are essentially for the proper working of the Internet. Unfortunately, neither of the two is meant to be dynamically configurable. More often than not, they rely on tabled files that are only occasionally updated. Clearly, some modifications to current schemes have to be considered. There can be a few solutions:

- *Dynamic IP allocation*. The addition of a new host in a sub-net usually requires the updating of routing tables and host-name tables in some servers, for the reasons given above. Although this is conceptually trivial, it is administratively troublesome. Recently there have been schemes which dynamic name servers, that allocate host names and addresses on a on-demand basis. A popular example of this is the Windows Internet Name Service (WINS)[1]. This scheme was specifically designed to tackle the dynamic IP allocation problem, both in LAN and in dial-up connections. However, the downside is that all clients must be WINS-aware, making it incompatible with 90% of the Internet community. Moreover, it does not really work when the client is connected in a foreign network.

- *Central database*. In this scheme, a central well-known dedicated server is assigned to keep track of new IP addresses of its hosts. To access these hosts, the server will then have to be queried first. This scheme is useful for clients connected to foreign networks, but is probably only useful for IP addressing using the 'dotted' numeric form, hence incompatible with DNS. This clearly creates difficulty for the remote user. Moreover, it is not quite transparent.

- *Broadcast*. This is a simple scheme where the robot, when connected to some network, sends a broadcast message throughout the Internet, indicating its new location information. The remote application, once started up, will listen for such a message before establishing the connection. This scheme may only be possible through the use of IP-multicasting, although this has not been tested. There is an apparent danger in saturating the Internet bandwidth.

- *IP tunnelling*. This is a fairly new idea, which extends the conventional IP packet destined for some network, by enclosing it in another IP packet, destined for another network. As such, it is also known as 'IP-in-IP'. The

[1] Windows Internet Name Service (WINS) is a suite of network services for the Windows platform that features, among other things, dynamically allocated Internet addresses and hostnames.

power of this scheme lies in its capability of *tunnelling* through a gateway into another network (hence the name), thus solving the gateway problem. It is also fairly transparent, but only if suitable *Home Agents* and *Foreign Agents* are set-up beforehand. Another advantage is that DNS will still work without modifications.

## INITIATING THE NETWORK CONNECTION

The astute reader may notice from the previous section that, one cannot assume that the robot will always have a permanent network connection. In migrating from one location to another, there will be a period of disconnection of the robot. If the robot is truly mobile, it may not even have human assistance in its vicinity. Hence, we have a real problem of how to re-establishing the network connection to the robot, which can affect the operation of the robot from the remote controller's point of view. He/she inevitable assumes that the robot is currently connected somehow, somewhere. The presence of the mobile phone here offers another dimension of bi-directional communication. There are 3 perspectives that can be taken with this issue:

1. *Human assisted connection*. The simplest case is when the robot is in the physical control of a human in the vicinity, who has to either manually connect the robot to the LAN, or issue the dial-up command on the portable computer. The remote controller has no idea whether the robot is connected or not.

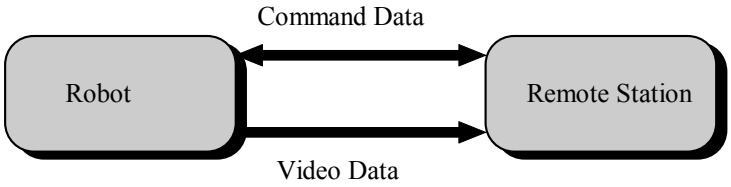
2. *Robot initiated connection*. In this scheme, the control program on the robot has to have some form of access to the modem interface, and perform the dial-up link as and when required by its operation. The dial-up may be to the home dial-up server, or a local ISP. Again, the remote controller has no knowledge of when this is done.

3. *Server initiated connection*. Here, a server is responsible for ensuring that the robot is connected and accessible. When a remote controller tries to contact the robot, and if the robot is not currently connected, then the server will have to intercept this message and try to contact the robot through its mobile phone. Once the robot is contacted, the server may connect directly to a dial-up server running on the portable computer on the robot, or it may initiate a callback sequence that

subsequently results in a robot initiated connection. In the remote controller's point of view, this method is more transparent.

## NETWORK BANDWIDTH CONSTRAINTS

The real-time control nature of this project requires that a considerable amount of data to be passed between the robot and the remote control station. Two main types of information are of particular importance: video information and control information.




♦ *Figure 2: Communication between the robot and the remote controller*

Obviously, video information forms the bulk of the data transfers. However, the control information cannot be overlooked, since it affects the responsiveness of the real-time control. A danger may exist where the sheer volume of the video information begins to starve the control information of network capacity. Another area of concern is the 'smoothness' of the overall information flow, which is largely dependent on any bottlenecks that may exist in the underlying networking system. These bottlenecks can determine the effective network bandwidth to work with. Basically, the bottlenecks can occur in 2 main areas:

1. *Server*. An overloaded server is obviously less capable of feeding information requested. It should be equipped adequately for such a task: huge memory, fast hard disk, fast network connection etc. Alternately, *handler threads* can often be useful in managing the load.

2. *Client*. The application software used and the hardware platform of the client can determine the effective information rate the user receives.

3. *Virtual network link*. Of course, the network connection between a client and its server actually consists of several *hops* of sub-networks. In this scenario, it is right to say that *the chain is only as strong as its weakest link*. Hence any low bandwidth link in

this chain will limit the overall bandwidth as well.

As any Internet user will undoubtedly know, the transfer rate of the Internet can range from a hair-ripping crawl of a few bytes per second, to a blinding burst of several hundred kilobytes per second, or simply a heart-stopping stall. This may be dependent on the time zone and geography of the WWW sites concerned. The dial-up link imposes a further bottleneck to this channel. However, its bandwidth is often more constant, as determined by the specified capability of the modem used.

This effective network bandwidth will affect various aspects of the project, like the nature of the socket connection, video resolution, video compression techniques etc. An in-depth statistical analysis of this topic is clearly beyond the scope of this project. Hence a more high-level view is adopted. Current GSM standard offers a digital bandwidth of 9.6Kbaud in its supported modems. This translates to roughly a 1Kbytes per second transfer rate. Since the transfer rate of the Internet is randomly variable and non-deterministic, we will assume that the GSM bottleneck is the dominant factor. This will give us a more tangible limit to work with.

## PROGRAMMING TOOLS AND ENVIRONMENT

One of the design goals of this project is to make the system as platform independent as possible. This is in part due to the WWW, which inherently implies *heterogeneity* in the operating environment. Whoever is remote controlling the robot should not be bounded by any particular hardware device. All he need to have is a WWW browser running on his computer, which in today's term, is more often than not. Because of this, a programming language suitable for the WWW is needed. In this area, two current technologies stand out:

- *Sunsoft's Java*

  Java started out as a way to make WWW pages 'come to life': dancing icons, interactive tools etc. But in a very short span of time, it has also evolved into a computing platform. The philosophy behind Java is 'write once, run anywhere', and says a lot about Java's platform independence nature. Java applications used to come in the form of applets, downloaded from a WWW server, and run in a WWW browser's window. However, the maturing of Java as a major programming language has also

made it suitable for writing stand-alone programs, especially networked applications.

Java's strong point is its platform independence. It is also very secure in the networking department, since it was designed from the ground up as a network programming language. On the other hand, as it is still an evolving language, it is lagging behind its rival, ActiveX, in terms of Application Programming Interfaces (APIs). There is also a serious lack in the development tools currently available for writing Java programs.

- *Microsoft's ActiveX*

ActiveX is an extension of Microsoft's Component Object Model (COM) architecture, that make it possible to embedded COM objects into WWW pages, turning them into so-called 'Active documents'. Because it builds on Microsoft's existing COM technology, which has served them so well in the Visual C++ platform, the learning curve of ActiveX is considerably less steep than Java. Also, the availability of APIs, like Multimedia API and Video Conferencing API make ActiveX an attractive option for Internet development. The development tools and environment for ActiveX is also very established.

The glaring weakness in ActiveX is that it can only work under the Microsoft Windows platform. Moreover, as a network programming language, there are serious doubts over its security features. However, in some ways, its development tools and APIs more than make up for that.

## OPERATING SYSTEM

Needless to say, a suitable operating system (OS) has to be found and used in this project. To many other projects, this perhaps is a foregone conclusion in the shape of Microsoft Windows. However, other OS's have become popular, stable and powerful enough to demand a careful choice. In making such a choice, we should consider some special requirement of the project:

- *Networking capability*

This project is heavily dependent on the underlying networking support. There are several design problems specifically concerned with various aspects of networking, and the OS will have to be compatible with decisions made to address these problems. The OS should also provide a fast and stable networking platform to work with.

- *Development stability*

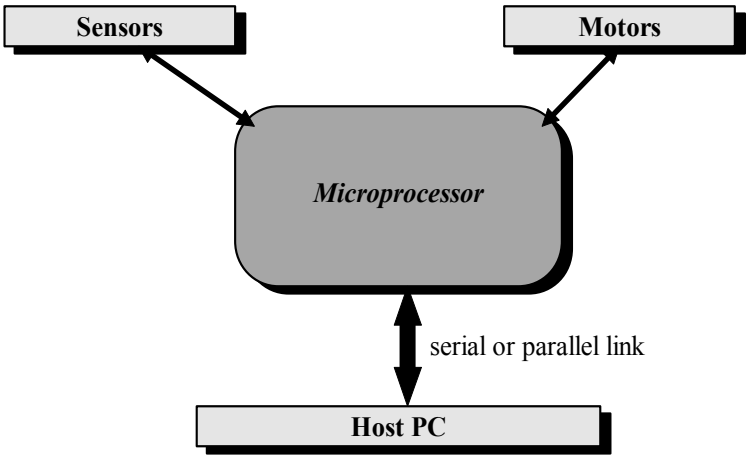
The OS should provide a stable and productive platform to develop the programs used by this project. The OS must support the choice of the primary programming language used in this project. Since I/O interfacing with external devices has to be contended with, the OS should not crash irrecoverably on I/O failures, as this may result in lost data, danger to the I/O devices, and last but not least, frustration to the developer.

- *Compatibility with other software/drivers*

Several hardware devices will be used in this project: the QuickCam, a GSM mobile phone etc. Drivers must exist to easily interface these devices to the portable computer. Several software applications and drivers will also be needed in the networking and client/server department.

## ROBOT CONTROL STRATEGY

Robotics control has been around for a long time. Traditionally, it is based around a single micro-controller, like the popular Motorola M68HC11, which performs most of the low-level closed-loop control functions, monitors sensory inputs and possibly also interfaces with a host computer through a serial link. Such a design has been tried and tested in many robotics projects:



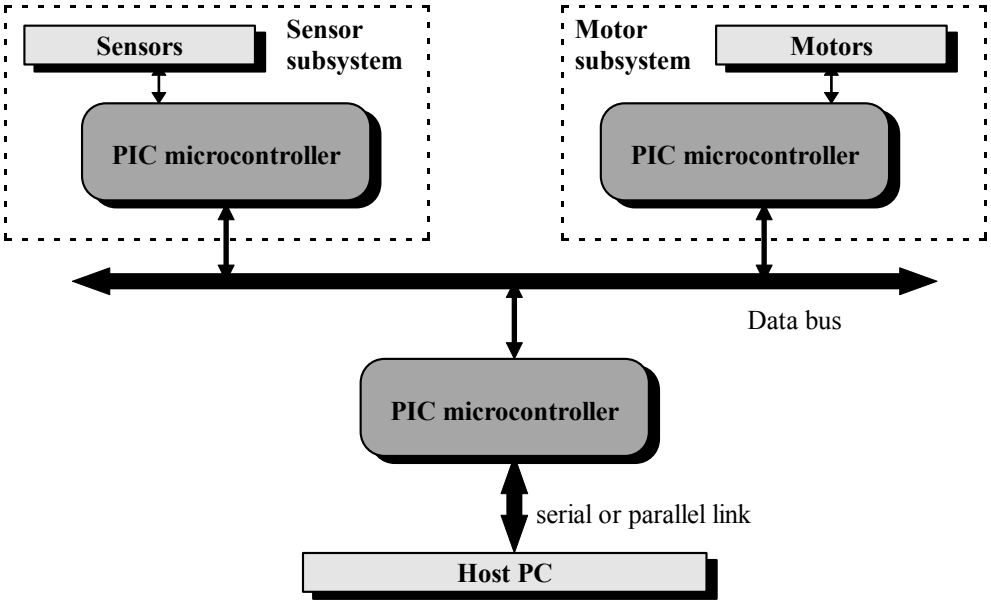

♦ *Figure 3: Traditional control model*

However, due to the size and cost of these micro-controllers, a hierarchical design is usually not feasible. Therefore as more and more peripherals are used, the load on the micro-controller increases, and development can get complicated. On the other hand, the advantage with using, say the M68HC11, is that since they are so common and popular, example modules for every conceivable application abound, thus making the design task simpler.

Recently, Microchip Technology[1] has introduced a series of RISC[2] micro-controllers, the PIC 16cXX, which have changed the robotics control concept. These micro-controllers are small, cheap, but by no means lacking in functional features. Because they come in DIL packages, they are extremely convenient in prototyping situations. With these micro-controllers, a modular design of robotics control is feasible:

[1] Microchip Technology Inc. Http://www.microchip.com

[2] Reduced Instruction Set Computers. RISC has become a general term given to that class of microprocessors with a small set of simple core instructions. The PIC series has only about 35 instructions.




♦ *Figure 4: Modular robotics design*

## VIDEO IMAGE COMPRESSION

The QuickCam used in this project is a black and white model, with an image depth of either 4-bit or 6-bit greys, and a resolution that can be as high as 320x240 pixels. It is interesting to note that even at a resolution of 160x120 (MPEG[1] standard) in 4 bit grey, every image frame is 160*120*(1/2) = 9.375 kilobytes per second. Remembering that the current network data rate of the GSM mobile phones as only about 1 kilobytes per second, one clearly has to consider the option of image compression to achieve a refresh rate of even 1 frame per second.

As video image compression goes, the obvious scheme that springs to mind is the MPEG scheme. Although it can typically offer a 40:1 compression ratio, the nature of the compression procedure is extremely computationally intensive. Hence it may not be suitable for use with a real-time application. Clearly, a compromise between compression performance and processor load has to be reached.

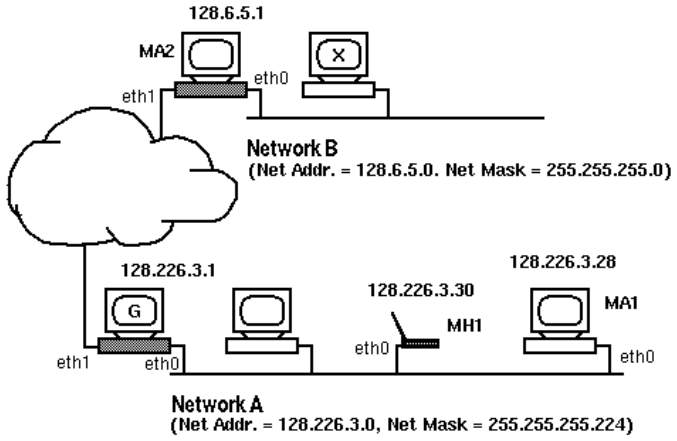
[1] Motion Picture Engineering Group. MPEG is a compression standard for audio, video and data established by the International Telecommunications Union and International Standards Organization (ISO).

# CHAPTER 7: DESIGN DECISIONS

After discussing the problems that arose from the design phase in the previous chapter, and discussing their possible solutions, we are ready to make some design decisions.

## MOBILE-IP

Mobile-IP is a technique that has been selected to tackle the problem of the migration capability of the robot. It is also sometimes known as "IP-tunnelling" in its other guise. To understand its operation, the robot is shown schematically in the figure below as the *mobile host MH1*. It has been assigned a permanent home address in its native domain, Network A. Also in this domain is an assigned *home agent MA1*.



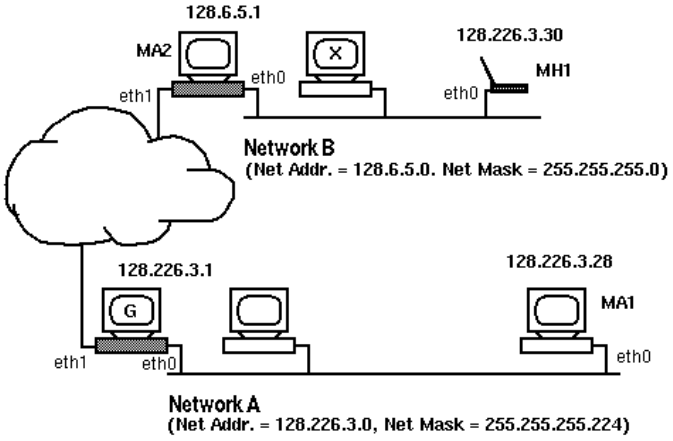

♦ *Figure 5: Mobile host in home network*

As the *mobile host* migrates to a new domain Network B, thus acquiring a new IP address, it informs the *home agent* of its new location and its new gateway in a form of handshaking. Henceforth, the job of the *home agent* will be to intercept IP packets destined for the original address of the *mobile host*, and forwards them to its new location.

It is worthy to note that the remote controller need not keep track of the IP address that the robot currently holds. In fact, he will always use the original IP address when accessing the robot, thus achieving transparency at the network layer.

There are currently several implementation of Mobile-IP in the Internet community.

The version used by the project was developed by Binghamton University, New York. It complies fully with the IETF[1] proposed standard (draft 15), and has a unique advantage of being able to work in the absence of a Foreign Agent. The Mobile Agent, in this case, performs its own de-capsulation of the IP packets. This obviously enhances its mobile property.




♦ *Figure 6: Mobile host in foreign network*

## IMAGE SIZE AND VIDEO COMPRESSION SCHEME

The default configuration that has been selected for the QuickCam is 160x120 pixels in 4-bit (16-level) greys. Such a decision was taken after examining the resultant image obtained from the supplied applications, and deeming it to be of acceptable visual quality. 160x120 is also the resolution specified by the MPEG-1 standard.

As for video image compression, it was decided not to use the MPEG scheme. Although MPEG decoding can be very speedy, MPEG encoding is a tedious affair. To compress a 'normal' sized image frame of 160x120, it typically takes at least a few seconds on even a Pentium processor. Moreover, MPEG is not optimized for our chosen image depth of 4-bit greys.

Instead, a more straightforward and simple compression scheme is used. The very first image is to be encoded in Run-Length-Encoded (RLE) format. After that, if a particular

[1] The Internet Engineering Task Force (IETF) is a loosely self-organized group of network designers, operators, vendors, researchers, who make technical and other contributions to the evolution of the Internet and its technologies.

pixel has not changed its colour, a 0 bit is sent. Otherwise a 1 bit followed by the 4-bit grey level is sent. Although this scheme sounds trivial, it can be performed relatively fast, which is important in our case. It is hoped that in this way, a minimum of at least 1 to 2 frames per second refresh rate can be achievable under the 9.6Kbaud bandwidth limit.

**JAVA**

There is little that separates ActiveX, Java and JavaScript as the choice of the primary programming language. Java was finally chosen mainly because almost all the popular WWW browsers support Java. This is important in ensuring platform independence. To further this goal of platform independence, a design decision was made to code all the software components in this project in Java. (However, in actual fact, some hardware specific components will still have to be coded in a native language). It is envisioned that, with the coming of the JavaChip and JavaOS, true mobile networking agents will be realisable. To this end, using Java exclusively is hoped to simplify such an upgrade, if at all possible.

**LINUX**

Linux is a freely distributed implementation of UNIX for the Intel x86 family of microprocessors. It aims to make the power of a true 32-bit multitasking UNIX operating system available to the masses. The fact that Linux is free is already an obvious plus point. Besides that, several other factors are influential.

Currently, Linux is arguably the most advanced networking platform available, offering significantly superior network performance over the other OS's, as well as a host of other features that they can only struggle to match. It also supports IP-tunnelling and IP-aliasing, which will greatly facilitate the use of Mobile-IP. Moreover, there is currently a Linux implementation of the Mobile-IP standard, which is also the only implementation that does not require the presence of an additional Foreign Agent on the foreign network, thus making the robot even more location independent.

Since a WWW server will be running on the laptop, the availability of a good WWW server for the platform is important. At the time of designing this system, WWW servers for most other platforms are expensive software. Only Linux has a freely available server in

Apache, which incidentally also happens to be the most popular WWW server on the Internet today.

Another factor is with regards to low-level hardware access. On the laptop, the control program will have to access both the serial port and the parallel port. Writing such code invariably threatens the stability of the operating system concerned. Linux, as a true multitasking operating system, provides the needed stability to experiment with these low-level programming.

On the other hand, it is worthwhile to point out some minus points in using Linux. Firstly, the learning curve for Linux is relatively steep, since it is essentially a UNIX implementation. Setting up the system for any purpose beyond the conventional is perhaps a bit daunting for the uninitiated. Secondly, SunSoft does not directly support the Java development kit for Linux. Although there is an ongoing project to port the JDK to the Linux platform, there will obviously be slight delays before the Linux ports catch up.

## PIC MICROCONTROLLERS

The PIC micro-controllers have been chosen as the basis for the robot control. Their cheap cost and convenient DIL packaging make them invaluable in prototyping. Also, a modular design based on them can make the robot extensible in features, without affecting the other modules. The primary PIC micro-controller used will be the PIC 16c84. This is an EEPROM based device, so that program development is extremely straightforward. The PICs will communicate via a software implemented $I^2C$ bus[1] interface, which is ideal in this situation, since it only requires 2 I/O lines.

## HP OMNIBOOK

Part of the design is to mount a portable computer onto the robot to provide for 2 main functionality: additional local intelligence for complex tasks like image processing, and as a reliable Internet platform through which the robot can link-up to the Internet. However, the size of the portable computer and its consequent incumbency on the robot

[1] Inter-IC Bus is a simple bi-directional 2-wire bus developed by Philips Semiconductor for efficient inter-IC control.

chassis undoubtedly becomes a necessary consideration. Hence several criteria was laid-down in choosing such a machine:

- lightweight
- a small footprint
- a long battery life
- a full complement of expansion ports and slots, for interfacing with hardware

Processor power, hard disk capacity, memory sizes and screen quality was placed on secondary importance. If it is affordable, it will be an added bonus. In this sense, **Hewlet-Packett's Omnibook** series of portable computers stand out. They typically weigh in the region of 3-4 pounds, smaller than a sheet of A4, yet has all the features one would expect from a larger portable computer.

## NOKIA 2110 GSM MOBILE PHONE AND DATA CARD

There is little that separates the various models of GSM mobile phones offered by manufacturers. This is partly due to the fact that GSM is an international standard that these manufacturers have to adhere to, hence their features and performance will likely to be similar.

Finally, it was decided that the **Nokia 2110** model be the GSM mobile phone of choice. This decision was made more on the basis that Nokia is an established brand name and the 2110 model has been known to be reliable.

Having said that, it should be apparent that any GSM mobile phone would suffice.

# CHAPTER 8: SYSTEM SPECIFICATION

At this point, we are ready to specify the exact hardware and software components that constitute the Internet-Controlled Robot.

## ROBOT

- *Motorised platform*
- *Connectix QuickCam*
- *HP Omnibook portable computer*
- *Nokia 2110 GSM mobile phone and Data Card*
- *Linux operating system*
- *Java 1.0.2*
- *Apache HTTP server*
- *Mobile-IP, mobile host module*
- *Home IP address*
- *Robot control program*

The robot is mounted with a portable computer, which links into the Internet through the GSM mobile phone. The portable computer will run the Apache HTTP sever, and the robot control program. These programs will run under the Linux operating system. The robot has to be allocated an IP address in its home network for Mobile-IP to work.

## HOME AGENT

- *Home Agent computer*
- *Mobile-IP, home agent module*
- *Home Agent IP address*

- *Modem*

A computer in the home network has to be used as a router to act as the Mobile-IP Home Agent. This can either be an available networked computer to be dedicated to that role, or an existing UNIX platform already acting as a router for some sub-nets. In the latter case, care must be taken to ensure that the required system software is compatible with it. The Home Agent's IP address has to be known to the robot beforehand. If dial-up networking to the robot is to be realised, then the Home Agent must be equipped with a modem.

## REMOTE CONTROL STATION

- *Computer connected to the Internet*
- *WWW browser*
- *Remote control applet*

No configuration or set-up on the remote control station is needed. The remote control applet is downloaded from the robot web server, and executes in the browser's window.

# CHAPTER 9: STATUS QUO

Since the design of this project was completed and implementation work began, several events have transpired which have altered the implementation of this project.

Firstly, the use of a GSM mobile phone has encountered administrative problems, mainly due to cost in the phone charges. The alternative being investigated is to use a DEC Ericsson mobile phone, which was previously used by the college in internal mobile phone operations. This mobile phone only has a local radius where its antennas are fitted. However, this alternative is still not confirmed yet, as it is still unsure whether it can be linked up to a computer. The purchase of the HP Omnibook portable computer has also proved to be equally elusive, again mainly due to the cost.

Originally, the robot platform is going to be developed from a kit ordered from the US, and then subsequently modified appropriately in this project. However, although the order was made some months ago, the delivery of this kit is not in sight.

Although several key components in this project are not available, the basic principles behind this project still need to be demonstrated.

To demonstrate and evaluate the real-time control issues, a simple rotating platform for the QuickCam has been built. Hopefully when the motorised robot kit does arrive, this rotating platform can then be simply mounted onto the chassis of the robot.

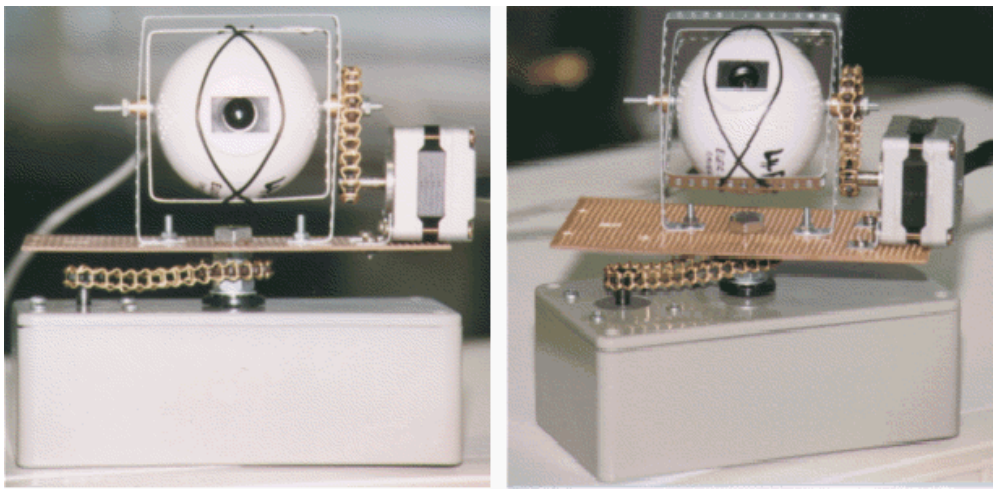
The principle of location mobility can actually be investigated even without the portable computer and the mobile phone. In network topology terms, a node is considered to be on another sub-net if it is served by a different gateway, because the gateway will screen off IP packets not destined for the hosts it is currently serving. Hence we can simulate location mobility by using a desktop computer, and switching it between different gateways.

The software side of the project has been completed.

# CHAPTER 10: HARDWARE

## ROBOT MECHANICS

2 separate stepper motors in the horizontal and vertical directions operate the motorised QuickCam platform. Theoretically the design is capable of rotating in a full 360° spherical fashion. However, due to the physical constraints of the wiring, only a limited turning angle in both directions have been implemented.



♦ *Figure 7: Snapshots of the QuickCam rotating platform*

The stepper motors used in this project has been salvaged from defunct 5 ¼" floppy disk drives. Their specifications are 12V, 3.6° per step and uni-polar. These motors cost next to nothing, but are completely functional, as opposed to stepper motors from industrial or catalogue sources that cost at least 20 pounds each.

Most of the control circuitry is kept in the enclosure base.

## CIRCUIT IMPLEMENTATION

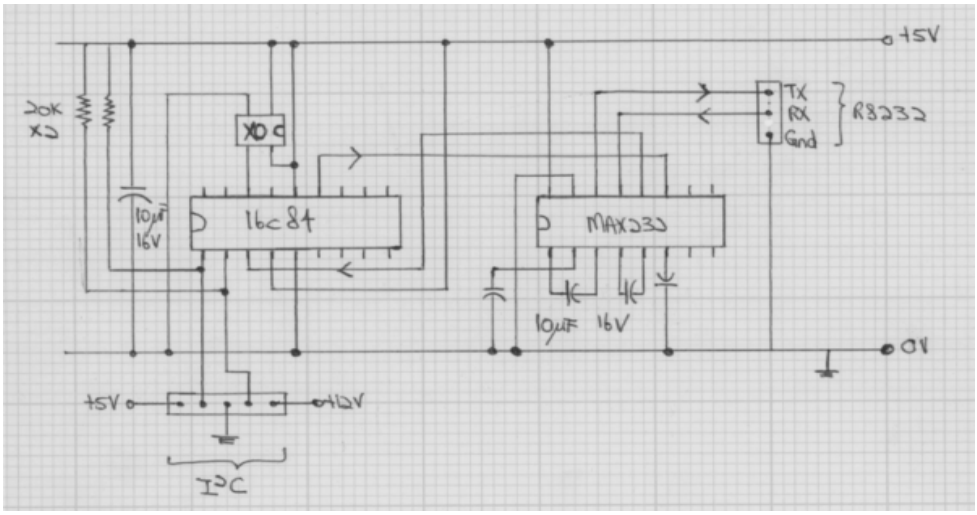
The whole circuit is made up of 2 independent circuit modules:

### SERIAL-TO-I²C MODULE

This module acts as the bridge between the RS232 interface of the portable

computer, and the $I^2C$ interface of the slave hardware devices. Commands sent by the host computer through the RS232 link is received by the 16c84. Then the 16c84 assumes the role of the bus master and conveys these commands onto the $I^2C$ bus. The baud rate of the RS232 interface is set at 1200, 1 start bit, no stop bit and no parity, which seems to be the standard configuration for a serial port. The maximum baud rate achievable in duplex mode is 9600.

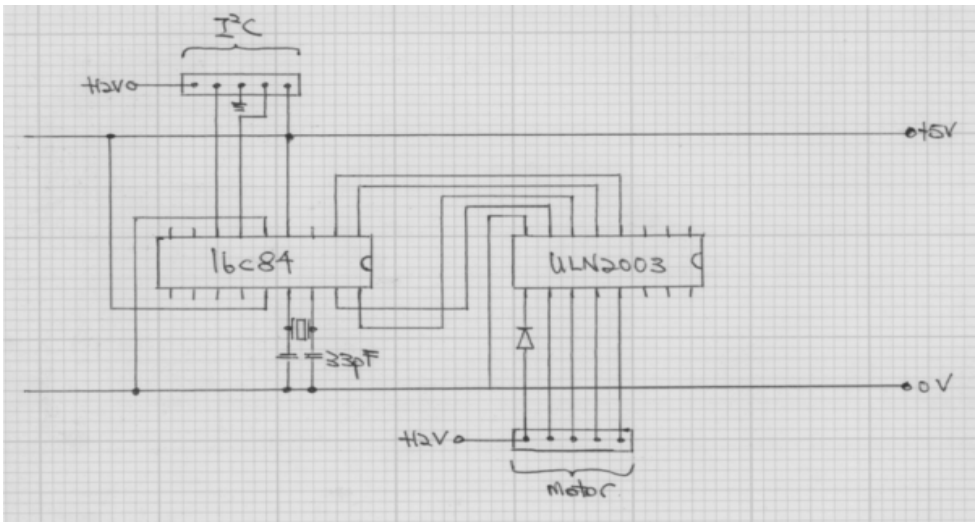
Note that pull-up resistors are needed on **SDA** and **SCL** for the proper operation of the $I^2C$ bus.




♦ *Figure 8: Schematics of Serial-to-$I^2C$ module*

STEPPER MOTOR DRIVER MODULE

The stepper motor drivers are implemented as slave $I^2C$ devices. 3 registers of the 16c84 are used for this purpose: 1 of them to store the number of steps to turn and the direction, and the remaining 2 tracks the stepper motor's current position. Note that the driver is implemented in such a way that the $I^2C$ bus is not held while the motor is being turn, since that can take some time. This allows maximum multi-access on the bus. Start bit detection is external interrupt driven on port B, pin 0 (**SDA**).




♦ *Figure 9: Schematics of the stepper motor driver module*

## IMPLEMENTATION ISSUES

### SETTING UP THE SERIAL PORT

Several problems were encountered when implementing the Serial-to-I$^2$C module. The main problem is related to the proper configuration of the serial port on the host computer. Because the host computer runs Linux, which is a variant of UNIX, setting up of the serial port is tricky to say the least. The secret lies in the obscure and less known command of '**stty'**. This command can configure every conceivable aspect of the serial port. The exact command needed by this project is:

**stty 1200 pass8 raw -echo < /dev/ttyS1**

This command sets serial port 1 to the baud rate of 1200, 8-bit mode, pass raw characters and no echoes.

### CONNECTING THE GROUND LINE

Although the RS232 interface can be used with only 2 lines (TX and RX), it was found that by connecting the ground line as well, more reliable transmission can be achieved.

### INTERRUPT DRIVEN OR SOFTWARE POLLING?

The reception module of the RS232 interface on the 16c84 can either by interrupt driven or software polled. Both methods was implemented and tested, with different

degrees of success. Software polling is easier to program, but is less reliable. The odd wrong characters are occasionally received especially when there is a long string of 1's or 0's. The interrupt driven technique is more complex, but has almost 100% success rate. For this project, the interrupt driven technique is finally kept. The transmission module, on the other hand, is software timed.

### MAX232 OR MAX220?

In order to interface the 16c84 to RS232 signals, a line driver has to be used to convert the TTL voltages from 0 and 5V to ±12V, and vice versa. The standard line driver in the industry is the MAX232. Indeed, the earlier version of the *Serial-to-I²C* module used this chip. But it also ran off a 9V battery. This combination resulted in the discovery that the 9V alkaline battery was occasionally too weak to drive the charge pumps in the MAX232. An alternative to use is MAX220, which is a low power version of MAX232, but can only guarantee a maximum speed of 19200 baud. This is not, however, crippling for this project. Consequently, the MAX220 is currently being used by this project.

### BI-DIRECTION I/O WITH 16C84

The stepper motor slave device currently uses pin 0 and pin 1 of port B of the 16c84 as the **SDA** and **SCL** lines of the I²C bus respectively. Using pin 0 of port B as **SDA** is necessary, since signal detection is external interrupt-driven. However, this causes a problem when using the rest of the pins of port B as output to step the motor.

> *...Caution must be used when these instructions are applied to a port with both inputs and outputs defined. For example, a BSF operation on bit5 of PORTB will cause all eight bits of PORTB to be read into the CPU. Then the BSF operation takes place on bit5 and PORTB is written to the output latches. If another bit of PORTB is used as a bi-directional I/O pin (i.e. bit0) and it is defined as an input at this time, the input signal present on the pin itself would be read into the CPU and rewritten to the data latch of this particular pin, overwriting the previous content. As long as the pin stays in the input mode, no problems occur. However,* ***if bit0 is switched into output mode later on, the content of the data latch is unknown.***

> - PIC16c84 data sheet, section 5.3.1

This unknown state of the data latch causes spurious logic levels on the I²C bus, which is very dependent on the accuracy of the logic levels. As a result, originally when pins 4-7 of port B was used to step the motor, the I²C bus failed. The remedy to this situation is to use pins 0-3 of port A instead to step the motor.

A possible improvement to this situation is to use port A to interface to the I²C bus. **SDA** can be connected to pin 4 (T0CLK), and an external interrupt can then be generated. Then the whole of port B can be used to step 2 motors simultaneously.

### I²C BUS

The I²C bus is used to allow the various modules to communicate with each other. More specifically, it carries the commands to turn the stepper motors from the Serial-to-I²C module to the Motor-driver modules. The I²C bus implementation in this project is heavily based on the example programs supplied by Microchip Technology. Nevertheless, the I²C bus implementation in this project is slightly different from standard:

1. *Software only implementation*. The I²C bus implemented in this project is entirely software based. Because of this, only a single bus master is allowed in the connection. Multiple bus master mode requires *Bus Arbitration*, which is very difficult to achieve in software due to the accurate timings necessary.

2. *Bus speed*. The standard I²C bus speed is 100 kHz. A faster version runs at 400 kHz, although this has not been widely used yet. But even for the standard 100 kHz bus speed, the hold times for the high and low logic states must be as low as 4 μsec. Remembering that the time taken for a single instruction in a 4Mhz 16c84 is 1 μsec, one can realise that this is a very difficult target to reach in software. To make things simpler, the I²C bus speed in this implementation has been moderated down to 25 kHz. As long as there is only a single bus master in the system, it does not affect the effectiveness of our project, as most of our slave devices will be external hardware devices anyway, which do not take advantage of a higher clock speed. The type of

$I^2C$ devices that are disadvantaged by this slower speed will be the memory devices.

3. *Supported $I^2C$ commands*. There are many high-level $I^2C$ commands that can be defined in terms of the primitive signals. A complete implementation of the full set is not essential for $I^2C$ to work. Therefore this project only implements the more useful functions :

    - `Write-to-Sub-Address` writes a single byte to a sub-address location of a slave.
    - `Read-from-Sub-Address` reads a block of bytes from a sub-address location of a slave.

## LOW-LEVEL CONTROL LOGIC

This project uses PIC 16c84 exclusively. This is mainly because it is an EEPROM device, hence program development on them is easy. There are roughly 4 main sets of low-level control logic that has been embedded into them. Most of these codes are canned routines modified from examples offered by Microchip Technology. As such, their logic is pretty straightforward. The only difficulty is in integrating them, where various problems arose.

### RS232 LOGIC

Although the RS232 interface implemented here is theoretically full duplex, the software synchronization used (*see next section*) makes it essentially half-duplex. The RX line of the RS232 is connected to pin 4 of port A, so that the receive mode is externally interrupt driven, while the TX line is connected to pin 7 of port B.

The timings used by the RS232 logic to detect or send the pulses are also interrupt driven for reliability, this time by the interrupt generated by the internal timer module of the 16c84. Once the start bit is detected by the external interrupt, it is masked out to prevent further interruption, and the internal timer interrupt is enabled with the proper timing parameter set-up. The transmit code uses the timer interrupt in the same way. This actually caused a minor conflict with the $I^2C$ master logic (*see below*), but has been accommodated.

### I²C BUS MASTER LOGIC

The I²C bus required 2 I/O lines, for **SDA** (data) and **SCL** (clock). Any I/O line may be used for either. Currently, **SDA** is connected to pin 2 of port A, while **SCL** is connected to pin 3 of port A. It is important to use a different port for the I2C bus then the RS232 interface, as bi-directional I/O is involved (*see previous section*).

The I²C bus master code uses mainly the code obtained from the example applications. They were meant to be canned routines, so that any application can simply plug them in and use them. But, of course, life is not a bed of roses.

Firstly, the delay code used by these routines was designed for a 16 MHz 16c71. They don't actually work on a 4Mhz clock (the values for the delay counters becomes negative). Therefore the delay constants have to be moderated down to accommodate a 4Mhz clock used by the 16c84 in this project. The consequence of this is that the I²C bus in this project only has a speed of 25 kHz, instead of the 100 kHz standard.

The canned routines consist mainly of macros that accept constants as parameters. Therefore modifications have to be made to make them accept register files as parameters.

There are several bugs in the implementation of the high-level I²C routines, particularly with the write and read routines. Although these bugs were identified after a long search, they have not been entirely cleaned out yet, due to time constraints. Instead they were merely circumvented. For details, refer to chapter 13.

Finally, the I²C bus specifications actually contains a built-in feature that allows a slow slave device to hold-up the bus while it performs its task[1]. However, to implement this on the bus master in software requires the use of the internal timer interrupt, which is also required by the RS232 logic. Although both of them do not use the interrupt at the same time, sharing them will involve some complex juggling. Hence such this feature is disabled.

[1] This feature is called 'Clock Stretching'. Although such a feature exists, many I²C devices don't really use it, since it is bad etiquette to hold up the bus.

### I²C SLAVE LOGIC

**SDA** has to be connected to pin 0 of port B, so that the reception can be interrupt driven. **SCL** has been arbitrarily assigned to pin 1 of port B.

When the 16c84 chip is interrupted on detection of a start bit, subsequent logic will determine if the operation required is read or write. For read operation, blocks of up to 8 bytes may be read at one time. For write, only single byte transfer is allowed.

8 register files, R0-R7, of the 16c84 are defined as sub-address of the slave device, and can be directly accessed by the read and write operations.

There are only 2 slave devices in this project, one for each stepper motor. The slave addresses assigned to them are 0xD6 and 0xD8, purely arbitrary[1].

### STEPPER MOTOR CONTROLLER LOGIC

The logic to control the stepper motor is actually driven by internal time. To maintain a reasonable speed of turning, the internal timer is used to generate an interrupt to step the motor, so that the current speed of turning is roughly 10 steps per second.

2 sub-addresses of the slave devices are used for controlling the stepping codes. R0 contains the steps required to turn the motor, while R1 tracks the current position of the stepper motor. The most significant bit of R0 indicates the direction to turn, while the remaining 7 bits indicate the number of steps to turn. User requests from the I²C bus are directly stored in R0. When R0 is checked, and found to have a non-zero value, the motor will be stepped accordingly.

Each time the motor is stepped, its position will be updated in R1. Extreme positions of the stepper motor have been defined, so that once R1 reaches these values, it ceases to be turned, and can then only be reversed.

Most standard uni-polar stepper motors have criss-crossed wiring. But the floppy drive stepper motors have more straightforward wiring, probably in order to simply logic and

[1] Actually, Philips Semiconductors, who invented the I²C bus, did specify a set of addresses to be used for user devices. Several addresses were reserved for special purposes. However, in this project, this is deemed unnecessary.

keep production cost down. There are 5 wires leading out from the motor, with the 1st (depending on your perspective, of course) to be connected to the power supply. The remaining wires then only need to be energized in left-to-right sequence for the motor to be stepped. If these wire are number 1-4 from the power supply wire, then the half step codes are:

| Sequence | Wire 1 | Wire 2 | Wire 3 | Wire 4 |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 |
| 4 | 0 | 1 | 1 | 0 |
| 5 | 0 | 0 | 1 | 0 |
| 6 | 0 | 0 | 1 | 1 |
| 7 | 0 | 0 | 0 | 1 |
| 8 | 1 | 0 | 0 | 1 |

♦ *Table 2: Half-stepping codes*

## COMMUNICATION PROTOCOLS

The main communication protocols that need to be discussed is concerned with the RS232 interface. The $I^2C$ interface basically takes care of itself.

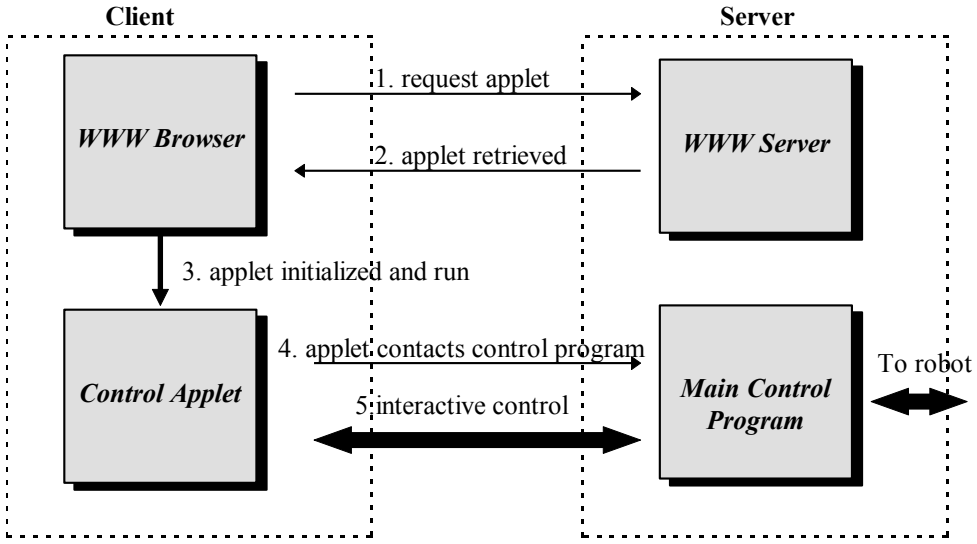
There are no special pre-fix or post-fix codes designed for the RS232 interface in this project. However, every byte transmitted in either direction has to be echoes back to the sender. This is not so much to ensure data integrity, but to act as a synchronizing method, so that there will not be any over-writing of data. This is particularly true for the *Serial-to-$I^2C$* module, where the 16c84 basically only offer a 1 byte buffer. Only when that previously byte has been absorbed by 16c84 logic will it send a echo back to the host computer to signal that it is ready to receive another byte. In this way, the host computer will not overflow the 16c84 with data.

One can argue that such an objective can be achieved with hardware synchronization using the RTS and CTS lines of the RS232. However, to implement this method, two more I/O lines and more control logic will be needed instead, which inevitably means more margins for error. It will only pay off if the data flow rate is very high. Otherwise, software synchronization is more than adequate, as is the case with this project.

# CHAPTER 11: SOFTWARE

## SOFTWARE COMPONENTS OVERVIEW

The following diagram shows the main software components in this project, and the sequence of communication between them:




♦ *Figure 10: Software components overview*

Almost all of the codes are written in Java, except for the low-level parallel port access routines, which have to be implemented in native format[1] for performance reasons. They consist of sets of object classes.

Java program development is based on the version 1.0.2 development kit.

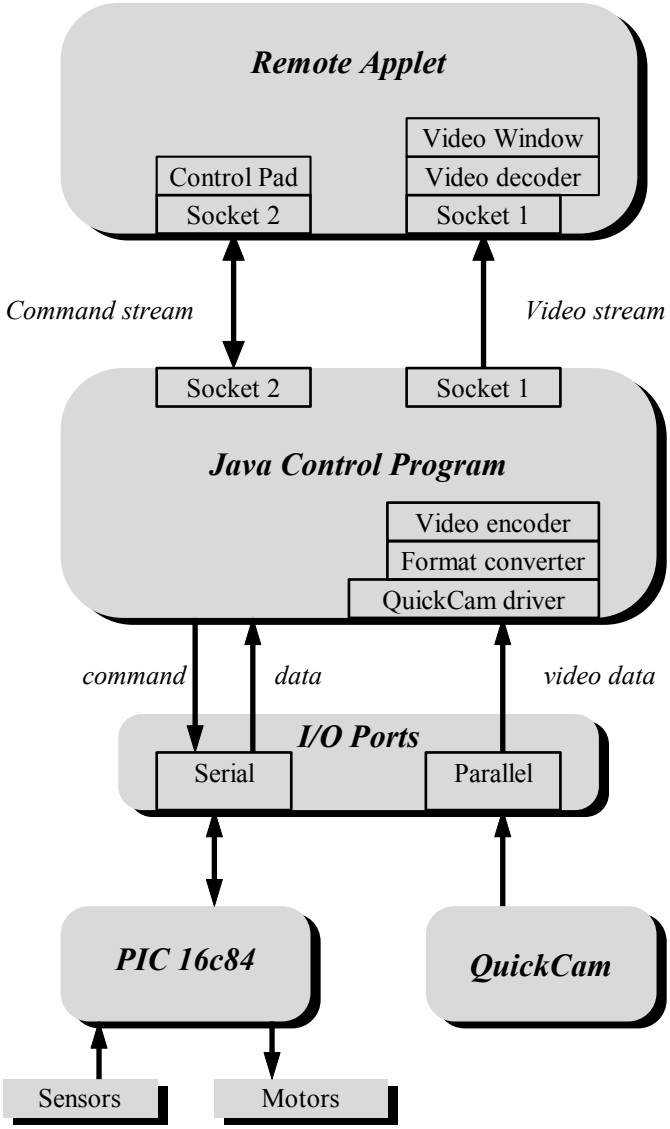
## INFORMATION FLOW

Figure 8 shows the information flow of the various components from the software point of view. Several points are noteworthy:

[1] Native format here means a platform dependent library of functions, which provides the low-level access routine to the operating system. In Unix, it takes the form of a shared library, while in the Windows platform, it is a DLL library.

- 2 separate I/O streams are used for the video data and command and control data. This is to prevent the command and control data from being starved by the huge volume of video data. This implies opening 2 separate sockets for them.

- Although not apparent from the diagram, several client applets can access the video stream simultaneously, while only 1 client have access to the command stream. That is, many users can *see*, but only one can *control*.




♦ *Figure 11: Information flow diagram*

## IMPLEMENTATION ISSUES

### RE-ALLOCATION SCHEME FOR CONTROL PRIVILEGE

As mentioned earlier, although multiple clients can *see* through the QuickCam simultaneously, only 1 client can *control* the robot. Now, when this controlling client disconnects, the privilege to control has to be re-assigned. The natural re-allocation scheme that springs to mind is First-Come-First-Serve. However, in this project, a more controversial scheme is used.

When the controlling clients disconnects, the waiting clients will be notified. This involves sending a 'wake-up' message from the CommandHandler thread to the remote client. Then the CommandHandler will wait for the remote applet to reply. The first client to reply will be granted the privilege to control.

At first sight, this *broadcast* scheme seems unfair. However, it does provide a simple discriminant for an important factor in this project: network speed. Generally, it can be assumed that whichever client who replied first will be the one whose network link is least congested. *Hence this scheme actually selects the client which arguably is the most capable of controlling the robot*. There is simply no point in granting the control to a client whose network link is saturated, no matter how early it is in the queue, because it will not be able to control the robot effectively anyway.

### JAVA, LINUX AND I/O

Besides file and console I/O, Java is not equipped with native class libraries to handle I/O devices, because that compromises its platform independence. Hence the use of Java to perform real-time I/O is uncommon, to say the least. The addition of Linux into this fray seems to further promote the situation to an almost unheard of status[1]. Fortunately, unbeknownst to most, Linux can actually help Java access I/O devices!

[1] Java has become such a widespread computing platform that several companies have introduced commercial I/O packages to facilitate accessing the serial and parallel ports. However, most of these implementation are for the Windows platform.

Two I/O ports are needed by this project: a parallel port and a serial port. The parallel port is used to access the QuickCam, while the serial port is used to control the robot. Under Linux, there are 2 ways this can be achieved:

1. Native implementation can be written to provide a low-level interface to a Java I/O class. In Linux this takes the form of a shard library.

2. Being POSIX[1] compliant, Linux maps all I/O devices into devices file in the **/dev** directory. Therefore, for general purpose I/O access, opening **/dev/port** as a RandomAccessFile class allows the user to control any I/O port location. Alternatively, more specific device files like **/dev/lp?** and **/dev/ttyS?** reads and writes from the parallel and serial ports respectively.

Both methods have been tried and implemented in this project. The main difference between them is speed. Native libraries are inevitably faster than device files. Hence for parallel port I/O, where a high data rate is needed, a native library is used. As for the serial port, a modest 1200 baud rate is set, hence the device file method suffice.

### JAVA AND REAL-TIME VIDEO

At the present, there are already several pieces of work on the Internet to enable live video on the desktop, most notably an application called CU-SeeMe[2]. Although these applications are excellent, they require an extra piece of software (i.e. the application itself) to be set-up and running on the client machine. This compromises the platform independence objective laid down in Part I of this report. Besides the WWW browser, no other software or set-up should be needed. Moreover, these applications are designed for permanent IP connections, not mobile phone connections. Hence work has to be done on embedding a live video applet into the browser window.

The Java Development Kit used by this project is 1.0.2. Although this version has excellent image file handling capability, smooth animation of video image sequences in an

[1] Portable Operating Systems Interface Extensions.

[2] CU-SeeMe (http://cu-seeme.cornell.edu/Welcome.html) is developed by Cornell University as a low cost video-conferencing facility.

applet is clumsy to say the least[1]. To address this problem, a freely distributed Java utility class, called **MemoryAnimationSource**, is used to provide a more transparent and efficient updating of new video frames.

### JAVA QUICKCAM DRIVER

An under-development low-level installable module for the QuickCam is available for the Linux platform. However, it originated before Connectix decides to release QuickCam's specifications. Hence it was more of a reverse-engineered driver, which tends to be buggy and unreliable.

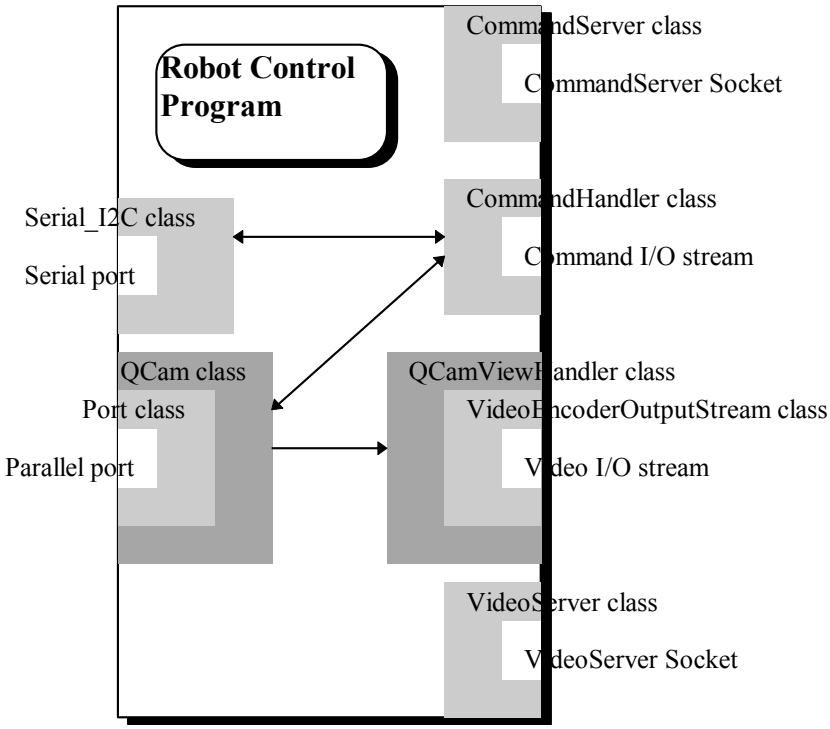
Since the QuickCam's specification has been made available, it was decided to implement a high-level interface to QuickCam in the Java language, complemented by a low-level parallel port I/O functions library. It is expected that such an interface will facilitate the later development of a video stream encoding scheme that is suited for our application.

## ROBOT CONTROL PROGRAM

This program runs on the robot's host computer, for as long as the robot is powered up. Once the remote applet is loaded from the HTTP server and initialized, it will communicate directly with the control program. The control program thus acts as the bridge between the remote controller and the robot, conveying commands and information to and from these two sources.

The implementation of the robot control program is heavily multi-threaded, so that a higher responsiveness can be achieved.

[1] In Java 1.1, the Abstract Windowing Toolkit has been greatly revamped with improved video animation handling capabilities. However, this project has not upgraded to it yet.




♦ *Figure 12: Class overview of Robot Control Program*

SERVER.CLASS

This class contains the main program, whose primary job is to fire up the various server threads (*see below*). These server threads may acquire some system resources (like sockets, QuickCam etc). They will throw exceptions if such acquisitions fail for some reasons. The Server class then has to issue a fatal exit.

Another job of the Server class is to initialize the HTML[1] file that allows the remote WWW browser to load the remote control applet. The fixed location of this HTML file is "/var/lib/httpd/htdocs/robot.html".

COMMANDSERVER.CLASS

This is the 'listener' thread for the command I/O stream. It simply opens a socket and endlessly waits for incoming connections. Once an incoming client is detected, it grabs a free socket, and spawns a CommandHandler class. Then it resumes to wait for another connection.

[1] Hyper Text Markup Language is the standard form followed by all WWW pages on the Internet.

Currently, the CommandServer class is tied to socket 8000.

#### COMMANDHANDLER.CLASS

This class starts the actual thread that handles the command stream. As mentioned earlier, only 1 client can control the robot at any 1 time. Therefore at any one time, only 1 CommandHandler thread is actually active. The others simply waits for this active thread to exit (when user disconnects). Once this happens, these other threads will be notified, and they will compete for control of the robot (*see below for control re-allocation scheme*).

The CommandHandler class uses the Serial_I2C class to convey the commands to the $I^2C$ slave devices through the RS232 interface. It also communicates with the QCam class for any adjustments needed for the video attributes, i.e. brightness, exposure and contrast.

#### VIDEOSERVER.CLASS

This is the 'listener' thread for the video I/O stream. Its operation is similar to the CommandServer class, but spawns the QCamViewHandler class instead.

Currently, the VideoServer class is tied to socket 8001.

#### QCAMVIEWHANDLER.CLASS

This class starts a thread to handle the video I/O stream. Because more than 1 client can *see* through the QuickCam, there is 1 QCamViewHandler class for each client.

There is only 1 QCam class that is supplying the video information (because there is only 1 QuickCam). Hence each thread communicates synchronously with the QCam class to obtain the current video frame in a mutually exclusively manner. To minimize the disruption to the QCam class in acquiring video frames, each handler threads merely secures a lock on the current video frame array, copies it to its own buffer, then releases the lock for the other threads. Once the current video frame has been copied into the buffer, it is passed into a VideoEncoderOutputStream class, to be sent out into the video I/O stream.

#### VIDEOENCODEROUTPUTSTREAM.CLASS

This class sits between the QCamViewHandler class and the video output stream to

encode the video information. The current encoding scheme is a simple RLE cum differential encoding scheme. The first frame is encoded in RLE format. After that, if a particular pixel remains the same between frames, a 0 bit is sent. Otherwise, a 1 bit follow by the 4 bit intensity level is sent.

#### QCAM.CLASS

This is the class that contains the Java interface to the QuickCam. It also contains a thread that acquires video frame from QuickCam in the background.

The Java QuickCam driver follows closely to the specifications laid down by Connectix. However, some peculiarities in its operation exist, mainly concerning the strange echoes that are periodically received from the QuickCam. These problems are discussed in details in chapter 13.

#### SERIAL_I2C.CLASS

This class communicates with the PIC 16c84 through the serial port to convey commands onto the $I^2C$ bus. It receives the parameters from the CommandHandler class, and then transmits them byte by byte to the 16c84.

#### PORT.CLASS

This class is the Java interface to the native libPortIO library. It contains parallel port access functions like open, close, read and write. Theoretically, this class is platform independent. It only depends on the implementation of the libPortIO library, which *is* platform dependent.

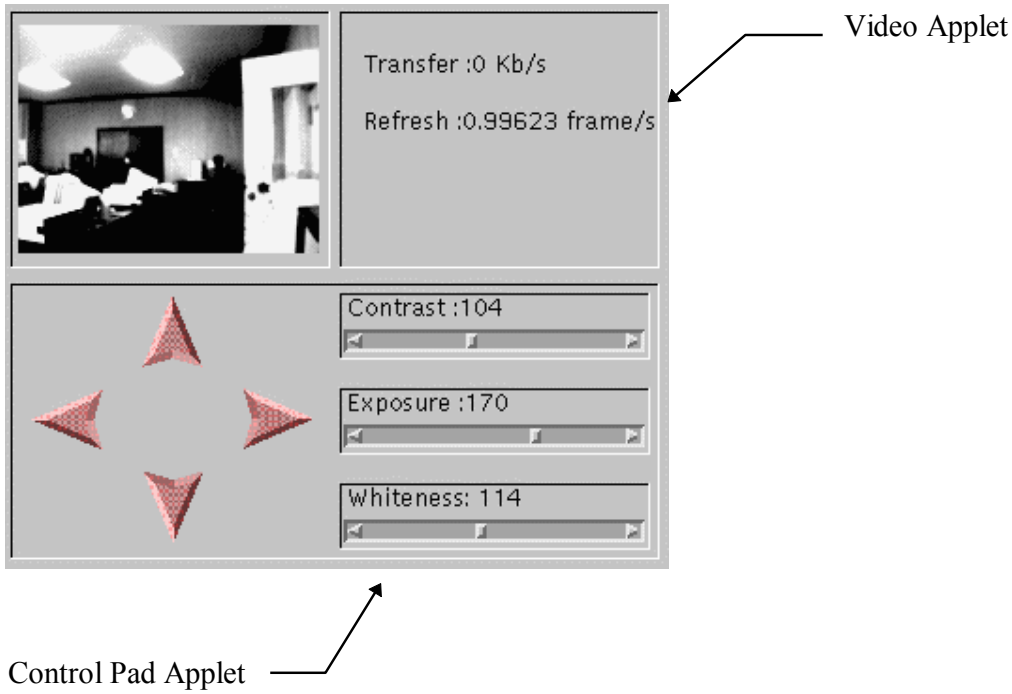
#### LIBPORTIO LIBRARY

This is the platform dependent parallel port I/O access native library. In Linux, this is compiled as a shared library. The provision of such a library is mainly for I/O speed, so that a higher frame rate can be achieved. The Port class accesses the I/O functions through this library.

## REMOTE CONTROL CLIENT

The remote client actually consists of two separate autonomous clients: The Video

applet and the ControlPad applet. Such a design not only greatly simplifies program development, it also re-enforces the object-oriented nature of the software.




♦ *Figure 13: Remote control applet*

CONTROLPAD.CLASS

This is the applet class that gives the user the tools to control the robot. When it initially link up with the CommandServer class on the server side, it will be informed whether some other clients currently has control over the robot. If so, it disables the control pad and notifies the user, before going to 'sleep'. Once the control is freed, it will be woken up, and will compete for the privilege to control in the manner that is previously outlined. If it fails, it 'sleeps' again. Otherwise, the control pad will be set-up, and the user can proceed to exercise control of the robot.

The user interface of the control pad is intuitive. Clicking on the any of the arrows move the robot is the respective directions, while the slider bars allow the user to alter the contrast, exposure and whiteness settings of the QuickCam.

IMAGEBUTTON.CLASS

This is a utility class that provides bitmapped buttons for the control pad.

HIGHLIGHTFILTER.CLASS

This is a utility class that provides the highlighting capability of the ImageButton class.

VIDEO.CLASS

This is the applet class that communicates with the QCamViewHandler class on the server side to obtain the video information. This information received directly from the video I/O stream is in encoded format. Therefore to obtain the raw video frame data, it must first be filtered through the VideoDecoderInputStream. The Video class then updates the QCamView class to refresh the video window, and also updates the Status class to display new readings.

VIDEODECODERINPUTSTREAM.CLASS

This class sits between the video input stream and the Video class, to decode the incoming video information into raw video frame data.

QCAMVIEW.CLASS

This class hosts the MemoryAnimationSource class to display an animated video window.

MEMORYANIMATIONSOURCE.CLASS

This is a utility class obtained from the Internet, which provides almost transparent animation capability to the standard Java image classes.

STATUS.CLASS

This class provides 2 status readings on the applet: data transfer rate, and frame refresh rate. These readings are primary used for performance evaluation purposes.

# CHAPTER 12: NETWORKING

## COMPUTERS SET-UP

A total of 3 machines are in the network set-up of this project.

- **Rupee** (155.198.147.9), in room EE404, is configured as the Home Agent.
- **Vatu** (155.198.147.23), also in room EE404, acts the 'residential' robot, i.e. in the scenario when the mobile host is in the home network. The QuickCam rotating platform is hooked to it.
- **Marder** (155.198.130.6), in laboratory EE501, is used to simulate the mobile host in a foreign network.

Both **vatu** and **marder** are Pentium machines, while both **rupee** is a 'lowly' DX2-66 machines.

## LINUX KERNEL

The computer hosting the robot runs the Linux operating system. Although the system files required are straightforward distributions, a proper kernel has to be configured with the necessary options included:

- The appropriate network card driver. For **vatu**, an updated network driver compatible with the 3c900 series (included in the newer kernel source) is needed. Bus mastering must also be disabled in the driver. For **rupee**, the Western Digital 803 driver is need, while for **marder**, the SMC Elite Ultra driver is needed.
- Dummy network driver. This is needed by the Mobile-IP programs.
- IP-tunnelling. This enables IP encapsulation and decapsulation for Mobile-IP.

## MOBILE-IP

The implementation of Mobile-IP used in this project has the capability to work in the absence of a foreign mobility agent. This is considered a plus point, as it increases the migration capability of the robot to just about any network. Since no foreign agents are involved, the set-up is only slightly different as that in Figure 6 of chapter 7. In particular, only the routing table of the robot and the home agent need to be altered. Note that the tables below are slightly different from the examples given in [1], to reflect the changes in recent Linux kernels:

HOME AGENT

```
# Tunnel device configuration
   ifconfig tunl0 155.198.147.9 up
   route add -host 155.198.147.0 gw 155.198.130.6 tunl0
   # advertize rupee's hardware address to intercept vatu's packets
   arp -s 155.198.147.24 00:00:C0:F1:CB:1B pub
```

ROBOT (MOBILE HOST)

```
   # Tunnel device configuration
   ifconfig tunl0 155.198.147.24 up
   route add -net 155.198.147.0 netmask 255.255.255.0 gw 155.198.147.9
tunl0
```

Once the routing table has been changed, the daemon programs, *mh* and *agent*, can be fired up on the robot and the home agent respectively to fire up Mobile-IP. Subsequently, the robot should still be contactable with **vatu**'s IP address, even though it has 'migrated' to **marder**'s location.

# CHAPTER 13: SYSTEM TESTING

The system evaluation is carried out with the computer set-up as detailed in chapter 12. The QuickCam rotating platform is connected to **vatu**, and the Home Agent is set-up on **rupee**.

## BASIC OPERATION

### SERVER SETUP

First, the robot control program has to be started up.

1. With **vatu** booted up into Linux, log-in as 'root'. No password is required.
2. Change Directory (cd) into 'project/server'.
3. Issue the command 'java Server', to fire up the robot control program. One should see some debug messages, and then the control program will be up and running.

### CLIENT

To access the robot, use any Java-supporting WWW browser to access the following Internet document:

http://vatu.ee.ic.ac.uk/robot.html

The remote control applet should start-up. If no one else is currently controlling the robot, the ControlPad applet will display all the control tools. Otherwise, a message will be shown to tell the user that control is currently held by some other user.

## CLIENT APPLET COMPATIBILITY WITH WWW BROWSERS

The location independence nature of this project relies largely on the running of the remote control applet on a popular WWW browser. The two dominant browsers on the Internet today are Netscape and Internet Explorer (IE). The following browser/platform combinations were tested, which should cover more than 95 % of the situations in the

world.

| BROWSER | PLATFORM | OBSERVATIONS |
|---|---|---|
| Netscape V3 | Unix | No problem. |
| Netscape V3 | Windows 95/NT | No problem. |
| Netscape V2 | Unix | No problem. |
| Netscape V2 | Windows 3.1 | The applet fail to appear. |
| Internet Explorer V3 | Windows 95/NT | No problem. |
| Internet Explorer V2 | Windows 95/NT | This version of IE does not support Java. |

♦ *Table 3: WWW browser test platforms*

In summary, the two incompatible platforms are Netscape V2 on Windows 3.1, and Internet Explorer V2. IE V3 is rapidly superseding the latter anyway, so it is not worrying. The former, however, still commands a sizeable user base in the world, despite the obvious inferiority of the platform. The source of the error is believed to be Windows 3.1's inability to cope with the applet's multi-threaded implementation. No rectification can be done except for a rewrite of the applet's code into a single threaded architecture for this platform.

## PERFORMANCE OF CURRENT VIDEO ENCODING SCHEME

### RAW TRANSMISSION

For a 160x120-image, with 4-bit intensity levels, each frame is 19200 bytes big if the bits are unpacked. If the bits are packed together, each frame will be 9600 bytes big. Using this form of raw video data, the average frame refresh rate is about 2 frames per second.

### COMPRESSION RATIO OF ENCODING SCHEMES USED

Initially, the Run-Length-Encoding (RLE) of the video frame data was carried out using a rather crude but computationally simple method. Each unit of data consists of a byte value, where the first 4 bits represent a run-length of 1 to 16, and the last 4 bits represent the intensity level. This scheme allows byte-sized data alignment, which simplifies the encoding and decoding processes. Obviously, the theoretical maximum compression ratio is 16:1. The average frame size achieved with the scheme is about 8500 bytes, which was expectedly low. The data transfer rate is slightly affected, which resulted in a refresh rate of around 1 fps.

Subsequently, a supposedly more efficient RLE encoding method using Shannon/Fono codes was tested. The results were interesting. Not only was the average frame size only a little better, being slightly less than 8000 bytes, the complicated bit-wise operation required slowed down the date transfer rate by more than 3 fold. Consequently, the frame refresh rate is disappointing, at about 0.2-0.3 fps.

From this exercise, 2 conclusions were reached:

1. For any encoding scheme, there is a trade off between performance and processing time. Real-time applications, like this project, need to be more sensitive to the time factor.

2. Image data does not respond well to standard statistical techniques.

These conclusions are the main reasons behind the adoption of the encoding scheme currently used, which was described in chapter 7.

#### CURRENT TRANSFER RATE

The current encoding scheme is delivering an average transfer rate of about 3-4 Kbytes per second.

#### CURRENT FRAME REFRESH RATE

On the average, the refresh rate of the remote applet is at present about 1-2 frames per second. This is acceptable, as smooth video has never been a priority anyway. The important target is to achieve an appreciable video sequence, at about 1-2 fps, within the data transfer rate of 9.6Kbps. Looking at the previous section, it is right to conclude that this objective has not yet been achieved with the current video-encoding scheme.

#### DELAYED VIDEO

One interesting observation was made about real-time video display. Due to the fact the video window is only refreshed when a complete frame is received, and also because of the real propagation delay in the transmission of the video data, video sequences always tend to be slightly delayed. This results in a phenomenon which the author calls *control latency*. It is discussed in more detail in chapter 16.

## ROBOT RESPONSIVENESS

It has been found that although the rotating platform actually responds very quickly to user commands, the video feedback received from this movement can take around a second to be updated onto the remote applet's window. As a result, the remote controller may get the false impression that the robot is sluggish, when it is actually quite responsive.

## TESTING OF MOBILE-IP

So far, the only aspect of Mobile-IP that was tested is the normal set-up of the home agent and the mobile host. Under the current set-up, the simulated robot at **marder** was successfully contacted through the help of home agent **rupee**, using the IP address for **vatu**. However, this is to be expected, since it is very much a standard set-up for Mobile-IP.

The testing that should be more relevant to this project is with regards to the robustness of the robot control program as the robot migrates to another network. However, this aspect cannot be simulated effectively under the present desktop set-up.

# CHAPTER 14: BUGS AND LIMITATIONS

## JAVA QUICKCAM DRIVER

Although the Java QuickCam driver has followed closely to the specifications laid down by Connectix, a particular peculiarity is still present, and that is with respect to its echoes. While the QuickCam is exposing a frame, it is unable to respond to any commands. Hence, to indicate that it is still 'alive', it will periodically complements all bits of the Nybble[0-3] bus. However, this Nybble[0-3] bus is also where the QuickCam driver expects to receive its echoes from. Apparently, there are occasions when the echoes themselves actually get complemented, and there is no way of predicting this behaviour. Hence, during the initial development of the driver, errors in the echoes were occasionally detected, which was initially quite baffling, to say the least.

Although this bug has since been uncovered, it has only been accommodated, by checking for both the correct echo *and* the complemented version.

## SERVER SOCKETS

Currently, the CommandServer and VideoServer classes are tied to sockets 8000 and 8001 respectively. Ideally, they should try to acquire the first available sockets instead. Although this can be easily done (by specifying 0 as the socket number), it was found that even on exit of the control program, these sockets are not totally freed. As a result, a re-run of the control program will result in the allocation of 2 more sockets, different from the previous 2.

Although this bug does not really affect the operation of the control program, it is potentially a 'time-bomb'. Therefore it is decided to acquire fix sockets 8000 and 8001 instead, while this bug is being rectified.

## $I^2C$ ROUTINES

Although quite a large set of high-level $I^2C$ commands are available as 'canned' routines from Microchip Technology's example applications, only the 2 most useful functions in this

context were adapted: Write-to-Sub-Address and Read-from-Sub-Address. However, the implementation of Read-from-Sub-Address is actually quite a 'hacking' job.

Ideally, Read-from-Sub-Address should be based on the 'canned' routine $I^2C$_READ_SUB. Unfortunately, the $I^2C$ bus could not be made to work using it. Consequently, the function $I^2C$_READ was used as a substitute. Now, since $I^2C$_READ itself does not specify the sub-address, the whole 8 sub-address bytes from the slave device are read *en-block* instead. The required sub-address is then selected from this block on the master side. Admittedly, this is shamelessly inefficient. However, it has been the only way to make the $I^2C$ bus work, and will remain so until the bug is uncovered.

# CHAPTER 15: COMPARISON WITH OTHER INTERNET ROBOTS

Internet-controlled robots are not new. At present, there are several WWW sites on the Internet that offers one form of implementation or another. They can mainly be categorized into two groups: live cameras, and tele-robots.

In WWW sites with live cameras, the visitor is allow to take a picture of what the camera is facing right at that moment, and then have that picture sent to his/her browser. The user may further specify the facings of the camera, the image size, the zoom ratio etc. Most of them deliver colour pictures as well. Examples of live cameras are the Swiss Sunsite LiveCam[1], and SchoolNet's Robotics Centre WebCam[2]. However, they all lack interactivity. The method of input from the user is typically form-based. Hence there is little sense of real-time control of a robot.

Tele-robots on the Internet, on the other hand, mostly involves the visitor posting a request for an action. The robot performs the action once it is available, then reports the results back to the visitor. A typical implementation is Australia's TeleRobot on the Web[3]. The user requests control of a robotic arm again with a form, and the robot performs the action. The user may observe the final status of the robot from 4 images of the robot arm captured from different angles. Again, there is a lack of interactivity.

The Internet-controlled robot in this project is meant to be a real-time control, live video feedback robot. Besides being a lot more interactive, the user interface is also more intuitive. The user presses an arrow, and the camera moves. More importantly, the user can *see* that the camera is moving. There are also potentially a lot more real-time issues to tackle in this project.

[1] http://sunsite.switch.ch

[2] http://webcam.engsoc.carleton.ca

[3] http://telerobot.mech.uwa.edu.au

Perhaps the nearest equivalent of this implementation on the Internet now is the Mechanical Gaze[1] from Berkeley University. The description from its WWW page sounds promising:

> *…Mechanical Gaze, a World Wide Web (WWW) Telerobotic Remote Environment Browser. Mechanical Gaze allows multiple remote WWW users to actively control up to six degrees of freedom (DOF) of a robot arm with an attached camera to explore a real remote environment… Live images captured from the camera of a remote robotic arm will be delivered to your web browser… we have developed an intuitive user interface for the WWW that allows actual navigation of a robot through a real-life physical space.*

Unfortunately, the server is temporarily down at the time of this writing. It would have been great to make a comparison with the robot in this project.

[1] http://vive.cs.berkeley.edu/capek

# CHAPTER 16: FUTURE DEVELOPMENT

This chapter discusses some important areas of this project that has not been investigated in detail by the author in the present implementation of the project, but nonetheless demands further attention. These are areas where future development projects can be undertaken.

## IMPROVED VIDEO ENCODING

The current bandwidth of 9.6Kbits per second offered by today's GSM mobile phones imposes a considerable constraint on the performance of this project. As noted in chapter 6, a constant stream of images at even a lowly 1 frame per second can easily result in a data rate more than 10 times of that offered by the mobile phone. On the other hand, command data tends to be short and occasional. Hence its bandwidth requirement is negligible.

Technically, there is no reason why this bandwidth is limited to 9.6Kbits per second. This selection was made more on the basis of economic reasons on the part of the mobile phone operators. Indeed, current work on the GSM standard is now in the third major phase - generally referred to as Phase 2 Plus. A key element of the work concerns data transmission, including bearer services and packet switched data at 64kbit/s and above, called High Speed Circuit Switched Data (HSCSD). This phase is expected to complete by 1998. Having said that, so long as such a constraint exists, one has to find ways to combat it. Even if the bandwidth is increased in the future, techniques adopted now will help to ensure an efficient use of the bandwidth in the future.

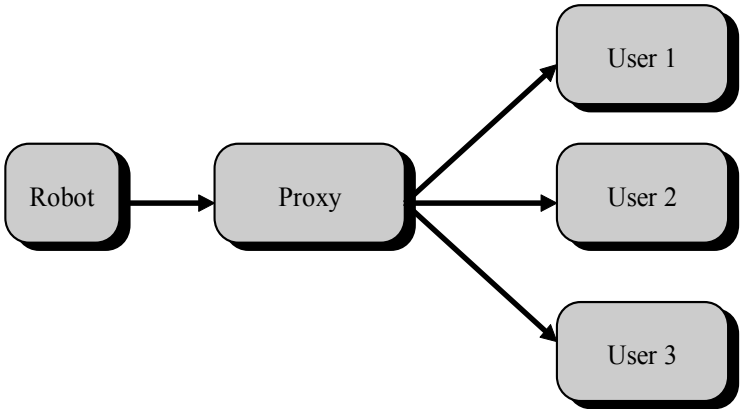
Basically, we need to look into *data compression* as the primary means to overcome this limitation. Immediately, MPEG encoding/decoding comes to mind. But MPEG encoding is also notoriously time-consuming. Hence it may not be suitable for real-time applications without supporting hardware. Alternatively, standard techniques like RLE and LZW offers fast encoding and decoding times, and acceptable compression ratios.

However, it is important to note that none of these schemes are specifically designed for the context of our application: 4-bit greys, 160x120 resolution, relatively slow moving

images. In information encoding, optimum performance is often only achieved if the characteristics of the data is taken into consideration while designing the encoder and decoder. For instance, spatial similarities tend to occur in both x and y directions, and temporal differences tend to be small. Also *lossy* techniques may be viable. By considering these factors, it may be possible to come up with an optimum encoding and decoding scheme, specifically for our case, and able to accommodate the 9.6Kbps bandwidth limitation currently imposed.

Another area where this bandwidth limit affects this project is in the broadcast of data. Currently, although only 1 connected user may control the robot at any one time, several users can actually obtain the visual applet simultaneously. There is already a problem trying to transmit large volume of information to just 1 user, let alone many.

One possible way of solving this is to define a *proxy* for the broadcast of information. As far as the robot is concerned, it only needs to transmit to the proxy. It is then the responsibility of the proxy to relay the information to the various recipients. To conserve computing resources, the proxy can be configured into the Home Agent. It will most definitely have a permanent Ethernet link, hence is more capable of the broadcasting role.




♦ *Figure 14: Proxy for broadcast of information*

The obvious difficulty with this scheme lies in the fact that the proxy may not know (indeed it should not know) who are currently connected to the robot. Hence it need to be told where it should broadcast. Therefore, the information sent from the robot to the proxy must be *tagged* with the connection details of the intended recipients.

The recent development of *IP-Multicasting* presents a more elegant way of broadcasting information to multiple clients. The advantage of this method is that no overhead is incurred in the IP packets when transmitting to more than 1 client. Perhaps a slight processing overhead is incurred in the extra decoding needed in the network drivers. Another advantage is that most of the work is done in the transport layer, not the application layer. However, *IP-Multicasting* is still very much under development.

## INCONSISTENCY IN DATA RATE

Although the GSM mobile phone, when used with a data card, guarantees a data bandwidth of 9.6Kbps, what is not apparent is that this transfer rate is less than constant. This inconsistency arises because of the nature of mobile communications. It suffers from some distortion effects quite typical of wireless communications. The principle culprits are Raleigh Fading, and shadowing loss. The former results from the reflections of the signal into different paths, while the latter results from obstacles in the signal transmission direction. Besides these 'natural causes', the *handover* procedure that takes place when the mobile user enters another cell will also cause a considerable length of disruption.

Well-known techniques are widely used to effectively compensate for these distortions. The end result is that, although digital transmission is guaranteed to be error-free, they often occur in bursts. For voice communications, this seldom matter, since our ears are relatively immune to phase changes, and we can comprehend interrupted speech quite well. However, when using the mobile phone in real-time applications, an inconsistent stream of data will often result in unsatisfactory performance.

For the Internet-controlled robot, such an inconsistent data stream will be most heavily felt in the video images, which will consequently become 'jerky'. Obviously, this will reduce the visual appreciation that the remote controller receives. If the command stream is also inconsistent, then the real-time control of the robot will become less effective.

## ISSUES IN CONTROL LATENCY

One interesting aspect of this project that surfaced through the course of the implementation phase is that of control latency. This problem relates to the responsiveness of the robot to the remote controller's commands, and arises due to the real difference in

time and space between the remote controller and the robot. Remember that in our case, the robot and its remote controller can potentially be half the world apart. There are 2 areas where such latency can occur:

1. Commands sent from the remote-controller to the robot may take a finite time period to reach the robot. This may result in the robot not being able to react soon enough to commands sent by the controller. For example, the controller may have issued a command to stop the robot because he/she has observed an obstacle directly ahead of the robot. However, due to the delay in receiving the command, the robot could have already hit the obstacle.

2. Feedback information from the robot may also take a finite time period to reach its remote controller. This is especially true for visual information, which involved a huge volume of data. This may result in the remote controller having a delayed appreciation of the current situation of the robot, thus not being able to issue appropriate actions quick enough, or overdoing some actions.

To address the first problem, what is clearly required is some form of artificial intelligence on the robot, giving it the ability to perform anticipatory actions, in addition to actions in response of the remote controller's commands. The robot need not be intelligent enough to navigate autonomously, but it should have the capability of ensuring it's own survival (e.g. prevent bumping into obstacle), cruising through some commands without user intervention, and possibly forward-anticipating the remote controller's commands. Note that this consideration further strengthens the need for a portable computer on the robot, as the computer can give enough processing power to generate such intelligence.

One interesting parallel to draw from this scenario is the neural cognitive concepts of human consciousness and human instincts. Although our actions are almost always governed by our consciousness, but in times of crisis, our instinctive behaviour can become overpowering to ensure our well being. There are also times when our consciousness is not apparently involved in some of our actions (e.g. while we are driving). Instincts almost always react to primitive sensory inputs, unless trained to be otherwise. We may thus think of the intelligence on the robot as its instinctive capability, while the remote controller takes

the guiding role in the form of its consciousness.

To address the second problem, a more immediate video rendering method is required. Currently, the video information is updated as a sequence of complete still images. Although the first pixel of the image may arrive immediately, the last pixel may well take some time. Therefore if one is to wait for all pixels to arrive before the image is updated, the delay in the video will roughly be equal to the difference in arrival time of the first and last pixel. Two methods can be investigated as possible improvements:

1. *Progressive updating*. Each pixel is updated immediately after it arrives, without waiting for the other pixels.

2. *Interlaced video transmission*. Similar to television transmission, instead of transmitting a whole image every time, interlaced sequences can be sent. This can reduce the data being sent, yet still provides a good appreciation of the moving image.

Note that such image handling capability is not primitive to Java in its current form. Hence realising it may involve some imaginative Java coding.

One may also install more primitive sensors on the robot, whose feedback information to the remote controller take a considerably less amount of time than that of video information. Such sensors can include infrared sensors, magnetic directional sensors etc.

## DYNAMIC CONNECTIONS

In order for the robot to be truly mobile, it must be capable of being contacted, or to contact someone else. That is what the GSM mobile phone is used for. It offers a wireless method to establish dial-up networking between the robot and the Internet. Ideally, the home agent should be able to dynamically establish a dial-up link with the robot as and when some remote controllers request access to it. This will simply require a dial-up networking server to be set-up on the robot's portable computer.

However, if the robot is currently not in the local vicinity, then establishing the network link may involve a long-distance telephone call, which can be financially costly. In this case, it is perhaps wiser to make the robot connect to the Internet through a local Internet Service Provider in its vicinity, as this only involves a local call on the part of the robot's

mobile phone. For this to happen, a series of communication protocols need to be initiated between the home agent and the robot:

1. The home agent calls the robot's mobile phone, and pass on a code to tell the robot to connect itself to the Internet.

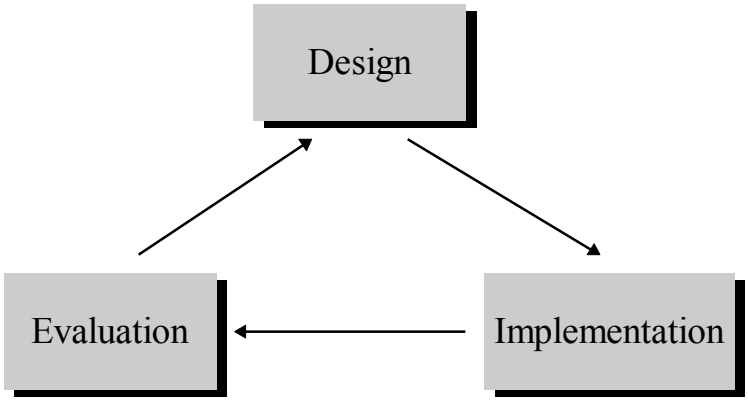
2. The robot, on receiving this code, disconnects itself from the call, then makes a call to the local ISP.

3. Once the robot is connected to the Internet, it contacts the home agent, and informs it about its new IP address (this can be done automatically through Mobile-IP).

This, of course, will call for some telephony functions to be programmed into the robot and the home agent, as well as a design of the communication protocol between them.

## DETAILED SYSTEM EVALUATION

This project has a heavy design component in it. The system design laid out in Part II of this report can only be considered preliminary, until some tests and evaluations can be performed on its performance and efficiency, with regards to some pre-defined criteria. Indeed, in the engineer's point of view, the evaluation stage is an important part in the circle of development.




♦ *Figure 15: Cycle of development*

In the design phase of this project, several areas were singled out as factors to be used

in the evaluation plan:

- *Network performance* can determine if the currently network configurations and methods are effective.
- *Video performance* can determine if the system has successfully circumvented the 9.6Kbps barrier of the GSM mobile phone system, since video data forms the bulk of data transferred.
- *Control latency* can determine the responsiveness of the system, which is important in real-time control applications.
- *Robustness* can determine the reliability of the system with regards to breakage in connections.

Sadly, no detailed investigation has been carried out due to time constraints. If a qualitative and quantitative analysis on these factors can be performed, then any shortcomings could be rectified, and a better designed system can be realised.

# CHAPTER 17: CONCLUSION

Looking back at the objectives laid down in chapter 3, it is right to conclude that they have mostly been achieved. The design of a Tele-Robotics system architecture that uses Internet protocols and mobile phones has been completed, although it has not been thoroughly evaluated yet. Instead of a remote-controlled mobile robot, a motorised rotating platform for the QuickCam has been built, which is sufficient to demonstrate the real-time control issues in this project. Key areas where further research can be undertaken have also been discussed in chapter 15.

Most of the promised deliverables have been completed, with the obvious exception of the mobile robot platform.

In retrospect, this project has been an exhausting but worthwhile venture on the part of the student. It is technically challenging, covering diverse areas of current technologies. It also has a good equal weighting of design, hardware and software aspects. Such a type of project is ideally suited for an Information Engineering System student.

Tele-Robotics is an exciting and stimulating field, which will undoubtedly reach new heights in the years to come with the widespread acceptable of the mobile phones and the Internet. The Internet-controlled Robot can be an important contributor to this development.

However, it must be said that the scope of this project is perhaps a bit too wide for what is essentially a 9 months part-time working schedule. Now that the specifications of the project have been spelt out, it is hoped that enough foundation has been laid for subsequent participants to have more clearly defined targets in their pursuit.

# APPENDIX A: ACKNOWLEDGEMENT

The author would like to express his deepest appreciation for the help rendered by the following individuals:

1. Dr. Jeremy Pitt, for his patient supervision of this project and being a constant source of enlightenment.

2. Mr Peter Cheung, for his invaluable opinions.

3. Mr Ian Harries, for his assistance in my understanding of the workings of a stepper motor.

4. Mr Jason Jordan, for his fine management of the laboratory resources.

# APPENDIX B: REFERENCES


1. 'The Design and Deployment of a Mobility Supporting Network', V. Gupta and A. Dixit, ISPAN'96, Wireless and Mobile Networks track.

2. Lecture notes on 4th year option, "Mobile Radio Communications", Dr M. Gurcan, 1997.

3. "The $I^2C$ bus and how to use it (including specifications), 1995 update", Philips Semiconductors, Arpil 1995.

4. "Connectix PC QuickCam Interface Specification, version 1.3" Connectix Corporation, September 24 1996.

5. "PIC16c84 data sheet", Microchip Technology, 1995

# APPENDIX C: PROGRAM FILES

## ROBOT CONTROL PROGRAM

| *File name* | *Lines of code* | *File size* |
|---|---|---|
| Server.java | 142 | 3727 |
| CommandHandler.java | 133 | 2816 |
| QCamViewHandler.java | 49 | 1139 |
| QCam.java | 506 | 13240 |
| Serial_I2C.java | 147 | 3940 |
| VideoEncoderOutputStream.java | 48 | 1100 |
| Lock.java | 25 | 455 |

## REMOTE CLIENT APPLET

| *File name* | *Lines of code* | *File size* |
|---|---|---|
| Video.java | 75 | 1605 |
| ControlPad.java | 202 | 5633 |
| Status.java | 53 | 1159 |
| QCamView.java | 24 | 515 |
| FramePanel.java | 81 | 2905 |
| HighligFilter.java | 63 | 1412 |
| ImageButton.java | 51 | 1292 |
| VideoDecoderInputStream.java | 47 | 938 |

## UTILITY JAVA CLASSES

| *File name* | *Lines of code* | *File size* |
|---|---|---|
| BitInputStream.java | 64 | 1311 |
| BitOuputStream.java | 43 | 928 |
| Port.java | 23 | 467 |
| PortImp.c | 51 | 1197 |

## I²C MASTER

| *File name* | *Lines of code* | *File size* |
|---|---|---|
| 16cXX.h | 187 | 4221 |
| i2c.h | 100 | 3575 |
| i2c_high.asm | 690 | 24844 |
| i2c_low.asm | 426 | 13149 |
| i2c_mast.asm | 348 | 10871 |
| rs232.h | 323 | 11249 |
| rcvr.asm | 169 | 4391 |
| txmtr.asm | 176 | 4454 |

## I²C SLAVE

| *File name* | *Lines of code* | *File size* |
|---|---|---|
| 16cXX.h | 187 | 4221 |
| i2c.h | 100 | 3575 |
| i2c_slav.asm | 454 | 12018 |