University of Southampton

Communication and control of a remotely operated underwater vehicle, using a distributed architecture approach

Stéphanie Michelle Rolland

Doctor of Philosophy

Institute of Sound and Vibration Research

October 2000

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING

INSTITUTE OF SOUND AND VIBRATION RESEARCH

Doctor of Philosophy

COMMUNICATION AND CONTROL OF A REMOTELY OPERATED UNDERWATER VEHICLE, USING A DISTRIBUTED ARCHITECTURE APPROACH

by Stephanie Michelle Rolland

Remotely Operated underwater Vehicles (ROVs) have been used in the oil industry since the 1970's. With the increase in the availability and complexity of instruments that can be fitted to the vehicles, the ability to modify the configuration becomes critical. By using a distributed communication architecture, where various functions of the vehicle are separated into several independent units, it becomes possible to interchange units more easily.

After a review of the available networking techniques, a particular solution has been selected, and used in a prototype vehicle. The vehicle has been tested successfully in water trials. A particular problem occurring with networks over which dynamic control systems operate was highlighted : if a control system was to be established over the network, the variation of the transportation delay could cause the controller to fail. For example, with a network node used for gathering heading data and another network node used for driving the thrusters, the time it takes for the heading data and thruster command data to be received depends highly on the behaviour of the other nodes present on the network.

In order to establish how this delay varies, a simulation of the network has been created, allowing for various configurations to be investigated.

To achieve total flexibility, it should be possible to keep the same controller for a control system running over the network, whatever the state of the network configuration. Such a controller is implemented by using a recursive least square estimator, the results of which are used to estimate the delay. The value of the delay is then used to tune the parameters of a PID controller. This self-tuning controller has been successfully tested both in simulation and experiments.

CONTENTS

ABSTRACT II

CONTENTS III

LIST OF TABLES VI

LIST OF FIGURES VII

LIST OF ABBREVIATIONS IX

ACKNOWLEDGEMENT X

1. REMOTELY OPERATED VEHICLE BACKGROUND 1-1

1.1 EVOLUTION OF OCEAN TECHNOLOGY 1-1
1.2 ROVS 1-1
1.3 INSTRUMENTATION 1-2
1.3.1 Thrusters *1-4*
1.3.2 Cameras *1-4*
1.3.3 Navigation systems *1-5*
1.3.4 Other instrumentation *1-5*
1.4 . CONTROL METHODS 1-6
1.5 LIMITATIONS OF MODERN VEHICLES 1-6
1.6 RESEARCH OBJECTIVES AND CONTRIBUTION 1-10
1.7 LAYOUT OF THE THESIS 1-10

2. NETWORKS AND COMMUNICATION 2-1

2.1 GENERAL COMMUNICATION CONCEPTS 2-1
2.1.1 Error Control *2-3*
2.1.2 Encoding *2-3*
2.1.3 Flow control *2-5*
2.1.4 Data link protocol *2-5*
2.2 LOCAL AREA NETWORKS 2-5
2.2.1 Topologies *2-5*
2.2.2 OSI model *2-6*
2.2.3 CSMA/CD (Carrier Sense Multiple Access with Collision Detection) *2-7*
2.2.4 Token Passing *2-8*

2.2.5 *Other methods of medium access control* 2-9
2.2.6 *Response to traffic load* 2-10
2.3 MAP AND FIELDBUS 2-12
2.4 DESIGN FEATURES 2-13
2.4.1 *Losses and Reflection* 2-13
2.4.2 *Signal Distortion due to Rise Time and Duty Cycle* 2-13
2.5 FUNCTIONAL ASPECTS 2-14
2.5.1 *Delays* 2-14
2.5.2 *Noise* 2-16
2.6 SELECTED METHODS 2-16

3. IMPLEMENTATION OF THE NETWORK 3-1

3.1 SDLC 3-2
3.1.1 *Error correction in HDLC* 3-3
3.1.2 *Limitations of HDLC due to framing and bit insertion* 3-3
3.2 PROTOCOL DESIGN 3-4
3.2.1 *Control field* 3-4
3.2.2 *Error management on protocol level* 3-6
3.2.3 *Addresses* 3-8
3.2.4 *Token passing* 3-8
3.3 PRELIMINARY TESTS AND DESIGN STEPS 3-11
3.4 COMPARISON WITH COMMERCIAL NETWORKS 3-12

4. EXPERIMENTAL SETUP 4-1

4.1 OVERALL CONCEPT 4-1
4.2 DETAILED INFORMATION 4-3
4.2.1 *PC Surface Unit* 4-3
4.2.2 *Thruster Card (Node 1A and Node 1B)* 4-8
4.2.3 *Navigation Card (Node 2)* 4-10
4.2.4 *Video Card (Node 3)* 4-12
4.2.5 *Hand Control Unit (HCU)* 4-13
4.3 NOISE SENSITIVITY 4-17

5. SIMULATION OF A NETWORK 5-1

5.1 LAN SIMULATION 5-1
5.2 STATISTICAL ASPECTS 5-2
5.3 CHOICE OF PROGRAMMING LANGUAGE OBJECT-ORIENTED APPROACH 5-4
5.3.1 *Existing languages for network simulation* 5-4
5.3.2 *Example of a C-code program* 5-4
5.3.3 *Object-oriented approach* 5-7
5.4 SIMULATION OF FIELDBUS NETWORK 5-7

5.4.1 *Results and tests* 5-12

6. **EFFECT OF VARIABLE DELAY ON CLOSED LOOP CONTROL** 6-1

6.1 DEFINITION OF SYSTEM STUDIED 6-1
6.2 EXPERIMENTAL SETUP 6-3
6.3 EXPERIMENTS WITHOUT INTRODUCED DELAY 6-5
6.3.1 *Open loop measurements* 6-5
6.3.2 *Closed loop control simulation and experiment* 6-8
6.4 DELAYED CASE 6-10
6.4.1 *Simulation* 6-10
6.4.2 *Experimental results* 6-13
6.4.3 *Comparison and conclusion* 6-15
6.5 SELF-TUNING SYSTEM FOR DELAYED PROCESSES 6-16
6.5.1 *Theory* 6-16
Example of application to a first order system 6-19
6.5.3 *Simulation* 6-23
6.5.4 *Experiments* 6-27
6.5.5 *Advantages and limitation of the method* 6-31

7. **CONCLUSION** 7-1

7.1 ACHIEVEMENTS 7-1
7.2 CONTRIBUTION TO RESEARCH 7-2
7.3 LIMITATIONS 7-3
7.4 SUGGESTIONS FOR FURTHER WORK 7-4

LIST OF TABLES

TABLE 3.1 STANDARD HDLC CONTROL FIELD USED 3-4
TABLE 3.2 CUSTOM CONTROL FIELD BIT ENCODING 3-5
TABLE 4.1 SEAEYE SURVEYOR SPECIFICATIONS 4-2
TABLE 4.2 HCU FUNCTIONS 4-14

LIST OF FIGURES

FIGURE 1.1 A REMOTELY OPERATED UNDERWATER VEHICLE (SEAEYE MARINE SCRUTINEER)....................1-3
FIGURE 1.2 SEAEYE VEHICLE CENTRALISED ARCHITECTURE....................1-8
FIGURE 1.3 AN EXAMPLE OF A NETWORKED VEHICLE ARCHITECTURE....................1-9
FIGURE 2.1 ASYNCHRONOUS TRANSMISSION OF A CHARACTER....................2-2
FIGURE 2.2 SYNCHRONOUS TRANSMISSION OF A CHARACTER....................2-2
FIGURE 2.3 DIFFERENT TYPES OF DATA ENCODING....................2-4
FIGURE 2.4 NETWORK TOPOLOGY....................2-6
FIGURE 2.5 THE 7 LAYER OSI MODEL....................2-7
FIGURE 2.6 CSMA/CD MODE OF OPERATION....................2-8
FIGURE 2.7 EXAMPLE OF A TOKEN PASSING LOOP....................2-9
FIGURE 2.8 DELAY VERSUS TRAFFIC LOAD WITH CSMA/CD....................2-11
FIGURE 2.9 DELAY VERSUS LOAD WITH TOKEN-PASSING PROTOCOL....................2-11
FIGURE 2.10 SIGNAL DISTORTION DUE TO DUTY CYCLE....................2-13
FIGURE 2.11 SIGNAL DISTORTION DUE TO THRESHOLD LEVEL....................2-14
FIGURE 2.12 PROPAGATION DELAY CHART....................2-14
FIGURE 2.13 TRANSMISSION DELAY CHART....................2-15
FIGURE 2.14 TIMING DIAGRAM....................2-15
FIGURE 3.1 SDLC FRAME FORMAT....................3-2
FIGURE 3.2 OCCURRENCE OF A SPURIOUS FLAG....................3-3
FIGURE 3.3 RESIDUAL ERROR DUE TO SPURIOUS FLAGS....................3-4
FIGURE 3.4 EXAMPLE OF SDLC TRANSFER....................3-6
FIGURE 3.5 CORRUPTED TRANSFER WITH RECOVERY....................3-7
FIGURE 3.6 CORRUPTED DATA TRANSFER WITHOUT RETRANSMISSION....................3-8
FIGURE 3.7 TOKEN PASSING FLOWCHART....................3-10
FIGURE 3.8 TOKEN INITIALISATION PROCEDURE....................3-11
FIGURE 4.1 PROTOTYPE ROV COMMUNICATION SYSTEM....................4-2
FIGURE 4.2 SIDE AND FRONT VIEWS OF THE VEHICLE BASED UPON THE PROTOTYPE FIELDBUS SYSTEM.........4-3
FIGURE 4.3 'PC DEVELOPMENT SOFTWARE' FLOWCHART....................4-5
FIGURE 4.4 PC MONITORING SOFTWARE FLOWCHART....................4-7
FIGURE 4.5 PULSE WIDTH MODULATION SPEED SIGNAL....................4-8
FIGURE 4.6 THRUSTER NODE SOFTWARE FLOWCHART....................4-9
FIGURE 4.7 NAVIGATION NODE SOFTWARE FLOWCHART....................4-11
FIGURE 4.8 VIDEO NODE SOFTWARE FLOWCHART....................4-13
FIGURE 4.9 HCU ARCHITECTURE....................4-15
FIGURE 4.10 HCU MAIN SOFTWARE STRUCTURE....................4-16

Figure 4.11 HCU menu structure 4-17
Figure 4.12 Noise tests setup 4-18
Figure 4.13 Noise tests results at 4.8 Kbds 4-19
Figure 4.14 Noise tests results at 10.5 Kbds 4-19
Figure 4.15 Noise tests results at 31.25 Kbds 4-20
Figure 5.1 Flowchart of Sadiku and Ilyas' simulation software 5-6
Figure 5.2 Class hierarchy diagram 5-9
Figure 5.3 Latency measurement 5-10
Figure 5.4 Main menu flowchart 5-11
Figure 5.5 Delay estimation and measurements results 5-13
Figure 6.1 Speed step response of Seaeye thruster 6-2
Figure 6.2 Schematic of armature control motor 6-3
Figure 6.3 The control tests experimental setup 6-5
Figure 6.4 Tacho calibration curve 6-6
Figure 6.5 Experimental open loop step response 6-7
Figure 6.6 Open loop simulation in Simulink 6-7
Figure 6.7 Simulated open loop step response (Step demand generated at t=1 sec) 6-8
Figure 6.8 Closed-loop PID control in Simulink 6-9
Figure 6.9 Simulated locally run (case a) PID step response (setpoint = 3000 RPM) 6-9
Figure 6.10 Experimental locally run (case a) PID step response 6-10
Figure 6.11 Closed loop PID simulation with variable delay 6-11
Figure 6.12 PID step response with no delay added 6-12
Figure 6.13 PID step response with 50 msec added delay 6-12
Figure 6.14 PID step response with 100 msec added delay 6-12
Figure 6.15 PID step response with 500 msec delay added 6-12
Figure 6.16 Remote PID step response without added delay 6-13
Figure 6.17 Remote PID step response with 50 msec added delay 6-14
Figure 6.18 Remote PID step response with 200 msec delay added 6-14
Figure 6.19 Remote PID step response with 500 msec delay added 6-15
Figure 6.20 Diagram of the Self-tuning controller 6-23
Figure 6.21 Structure of the simulation software for the self-tuning controller 6-25
Figure 6.22 Simulation result- PID response, using the self-tuning response 6-26
Figure 6.23 Simulation result - Delay estimate 6-27
Figure 6.24 Structure of the experimental software 6-28
Figure 6.25 Experimental setup for self-tuning controller 6-29
Figure 6.26 PID Self tuning control with no added delay 6-30
Figure 6.27 PID self tuning control with 50 ms added delay 6-30
Figure 6.28 PID Self tuning control with 100 ms added delay 6-31
Figure 6.29 PID Self tuning control with 200 ms added delay 6-31

LIST OF ABBREVIATIONS

ADC	Analogue to Digital Converter
AUV	Autonomous Underwater Vehicle
BER	Bit Error Rate
CCD	Charge Coupled Device
CP	Cathodic Probe
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DAC	Digital to Analogue Converter
DPLL	Digital Phase Lock Loop
FCS	Frame Check Sequence
FDDI	Fibre Distributed Data Interface
HCU	Hand Control Unit
HDLC	High-level Data Link Control
IEC	International Electrotechnical Commission
ISO	International Standard Organisation
LAN	Local Area Network
MAC	Media Access Control
MAP	Manufacturing Automation Protocol
NMEA	National Marine Electronics Association
NRZ	Non Return to Zero
NRZI	Non Return to Zero Inverted
OSI	Open System Interconnect
PID	Proportional Integral and Derivative
PWM	Pulse Width Modulation
RLS	Recursive Least Squares
ROV	Remotely Operated underwater Vehicle
RZ	Return to Zero
SDLC	Synchronous Data Link Control
SIT	Silicon Intensified Target
TRT	Token Rotation Time
TVP	Television Photographic camera
UART	Universal Asynchronous Receiver Transmitter

Acknowledgement

I would like to thank my supervisors : Dr Stephanie Merry and Dr Robert Allen, and my sponsors Seaeye Marine Ltd. for their support. Also I would like to thank James Perrett and Miles Pebody from the Ocean Technology Division at the Oceanography Centre for their help on technical matters.
Technical support was also provided by the ISVR laboratory technicians at the start of the project, and by the Mechanical Engineering department at the end of the project.

1. REMOTELY OPERATED VEHICLE BACKGROUND

1.1 Evolution of Ocean Technology

Thirty years ago the vision for the future was that technology would allow man to live on the moon as well as at the bottom of the sea. Such a progress in technology has not been as easy as expected.

It is the discovery of oil beneath the ocean that triggered the main progress in ocean technology. Oil is being exploited at greater depth, with little or no accessibility from divers. ROVs (Remotely Operated Underwater Vehicles) are an important tool for such undersea operations, such as survey, inspection and repair. They are still very much in use nowadays, and their future seems only threatened by the development of autonomous underwater vehicles (AUVs), which do not require a pilot. However, the extreme conditions where ROVs are used mean that the AUV technology would have to be well proven and advanced before a real competition appears.

1.2 ROVs

ROVs are an important tool for underwater exploration and exploitation, such as scientific and military surveys, inspections and repairs of subsea structures. Indeed, their use is constantly increasing, as human divers limitations offer less and less competition to a tele-operated robotic vehicle. The basis of the vehicle consists of a subsea unit, incorporating thrusters and usually a camera, and a surface unit which provides the pilot with means of tele-operation.

ROVs are used primarily in three different areas. The first is scientific observation, where the ROV's presence often interferes less with the actual scientific experiment than a human diver would. The second application area is military use, where ROVs can carry out high risk underwater operations. The third area is the oil exploitation industry, where ROVs can be used to inspect the state of corrosion and the integrity of subsea structures, cables and pipelines. These routine inspections have to be carried out regularly to comply with safety and insurance regulations. They can also be used during the construction period, as they can be fitted with tools such as manipulators, or cable cutters.

Since this research project is funded by a company whose main customers come from the oil-industry, the study emphasises the industrial aspects of ROV operations. A case study has also been carried out with one of the company's vehicles.

1.3 Instrumentation

The type of instrumentation used on ROVs can be divided in two categories: essential instruments and the optional ones. In the case of the Seaeye 'Scrutineer', the basic instrumentation consists of four thrusters, one black and white camera, one colour camera, a compass and a depthmeter (Figure 1.1). Those provide direct control and feedback to and from the pilot.

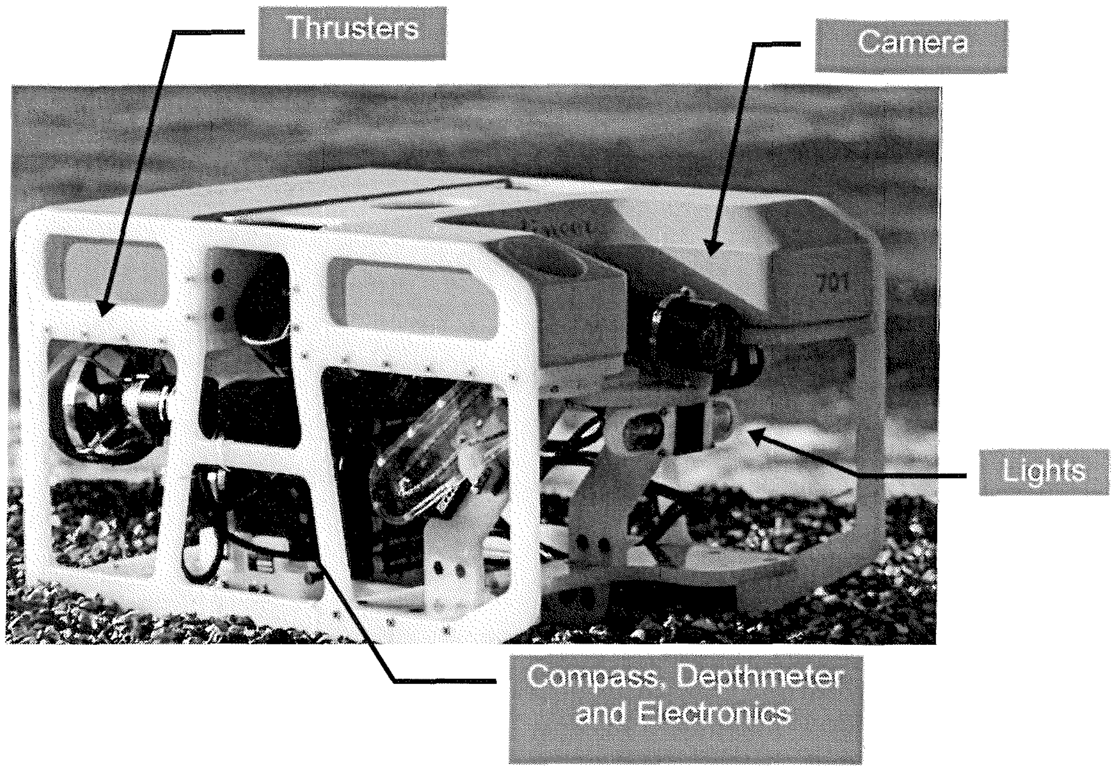


Figure 1.1 A Remotely Operated Underwater Vehicle (Seaeye Marine Scrutineer)

ROVs will also be fitted with specialised instrumentation, which allow it to complete certain tasks. For example, for a scientific survey, the vehicle could be fitted with temperature and current sensors; for an industrial pipeline survey, a cathodic potential probe and a pipe tracker would be fitted [1].

Interchanging these various instruments is often a difficult and time consuming task, sometimes demanding important modifications to the vehicle. The major requirement for a ROV is to make those unavoidable modifications as easy as possible. Such a need for flexibility places great demands on the system architecture, and interfacing and data communication is not straightforward. The range of instruments is wide and therefore it was necessary to review those most commonly used, in order to have a perspective on what the system architecture can be.

1.3.1 Thrusters

Propulsion is the most essential feature of the vehicle. Although novel propulsion mechanisms have been suggested, commercial ROVs are fitted with thrusters. Electric thrusters are often favoured, as they offer many advantages over hydraulic ones [2]:

- **reliability**: a mean time between failures of 1000 hours an average compared to less than 170 hours for hydraulic thrusters
- **simplicity**: it is much easier to interface electric devices with a controller than a hydraulic mechanism, as they generally have a linear behaviour, and so control is more accurate
- **payload:** hydraulic systems can be very heavy and bulky

1.3.2 Cameras

In the most common configuration, the ROV is fitted with both a black and white and a colour camera. Usually during the navigation the pilot would use the black and white output, and then switch to colour for close inspection of an object. Lighting conditions can be very poor underwater, and cameras have to be very sensitive. In some cases , a still photograph of the inspection is also required for documentation purposes. The camera is often fitted with a pan and tilt facility, allowing the pilot to inspect a wider area without having to move the vehicle. This pan and tilt facility can be provided by fitting the camera on a moving platform, or with a single module camera, where the pan and tilt functions are integrated.

Two principal technologies are available : CCD (Charge Coupled Device) based on solid state electronics or SIT (Silicon Intensified Target) based on a tube[3]. CCD cameras are more compact and lightweight than tube cameras for a given image size. They also use less power and require little or no set-up.

A typical example of those new generation cameras is the Osprey OE1386 series. This is a CCD camera, which incorporates Pan, Tilt and Rotation functions, allowing a 160 ° horizontal coverage and 180 ° vertical coverage (including lens angle of view). The Iris control is automatic, and the focus is controlled from the surface by the pilot.

Some cameras, known as TVP (Television Photographic Camera), incorporate both video and stills photography functions.

New types of viewing system are being developed, such as 3D acoustic cameras or laser viewing systems, however the cost of these remains very high and they are not yet very common, although they can be a great advantage in turbid waters [4].

Most commercial ROVs provide the ability to overlay the video image with comments and other data available, such as depth and heading, during the ROV diving operation. The resulting image is recorded onto a video tape. This procedure allows the ROV operators to prove to their customer that the work has actually been carried out. This task is a major contribution to the pilot's workload. Recent innovations now allow the interfacing of video signals with laser and computer disks for efficient storage, retrieval and analysis.

1.3.3 Navigation systems

There is a wide range of gyros and depth meters available, and prices vary depending on accuracy, speed, depth rating and size. All newer models incorporate RS232 communications. Alternative positioning systems use acoustic signals, that can be related to the mother ship's positioning system [5].

1.3.4 Other instrumentation

Apart from the above instrumentation, ROVs are often fitted , depending on the tasks to be carried out, with sophisticated sensors and tools, such as sonars, profilers, pipe and cable trackers, CP probes, manipulators, current meters etc... These instruments are generally fitted with a digital interface. The detail of how they operate is irrelevant to this research, but the way they can be linked in the vehicle is the key issue. It is indeed those more expensive instruments that will not be fitted permanently, and therefore rely on a flexible and modular vehicle architecture. Scientific instruments pose a particular problem, as the amount of data gathered makes real-time transmission unrealistic. Local storage is often used, allowing the data to be analysed only when the vehicle is taken out of the water. The real time communication only controls the instrument and the data storage facility [6].

1.4 . Control methods

All of Seaeye vehicles provide auto-heading and auto-depth features, which allow the ROV trajectory to be locked towards a certain heading or depth. This is implemented using a PID controller. The coefficients have been chosen after a series of trials in a sea water lake, and remain the same for all the ROVs produced. As a result the system is not efficient in all circumstances. The problem is that the model used by the PID has been set in a specific environment, and does not take into account variable parameters such as sea state, depth and equipment fitted on ROV. There is a need for a system which can adapt to those different environments .

This leads us to intelligent control theories such as sliding law control, self tuning control, fuzzy logic or even neural networks. Some work has already been done in those areas. [34][35][36][37][38]

The ROV developed by Woods Hole Oceanographic Institution, named JASON is using the sliding control technique with success. Heriot Watt University, Edinburgh is developing their ROV ANGUS H_∞ control law. Development in the University of Hawaii has been considering Parameter Adaptation Algorithm (PAA) and lately Neural Net .

However, apart from the JASON case, most of this work remains in simulation only and nothing has actually been implemented on the hardware.

Fuzzy logic seems the best solution for this case, as it would cope with the high non-linearities of the ROV, and is also very robust. Neural network could also be as a solution, but it requires much more computing and that it is difficult to prove its reliability.

Another important point is that all those control techniques require a fair amount of processing, and this should be taken into account when designing the hardware.

1.5 Limitations of modern vehicles

The current communication system used for Seaeye ROVs is based on RS485 communications. The main loop links the subsea unit, the surface unit and a video overlay unit (Figure 1.2). Although the system is fully functional, the communication architecture is not optimised, and improvements and modification to the vehicle are not straightforward. In this type of data loop, data from the sender has to go to all possible receiving units. This means, for example, that when the subsea unit sends data to the video unit, the surface unit also reads the message. Although only the two first destination identifying characters are actually read, this can be time consuming, especially when traffic is heavy. One can see that if the amount of information transmitted, or the number of units connected, was to increase, the processing time left at each unit for non-communication tasks would be badly affected. As instruments tend to be 'smarter', that is to say that they are increasingly using digital technology, the more information we can expect from them. Unfortunately, this also means larger message sizes.

The main problems encountered with the system at the moment are due to the lack of modularity and the lack of maintenance facilities, which are extremely important in a market where time is restricted by the parent vessel's availability, or sea conditions. The crew using the ROV is not generally highly trained technically, and diagnostics and repair should be as straightforward as possible. This also applies to the upgrading and modifying of the ROV configuration.

A good way to introduce modularity in the vehicle's system is to use networks instead of a centralised architecture. Such a modular architecture provides a more flexible platform for adaptation, with a number of intelligent nodes networked together (Figure 1.3).

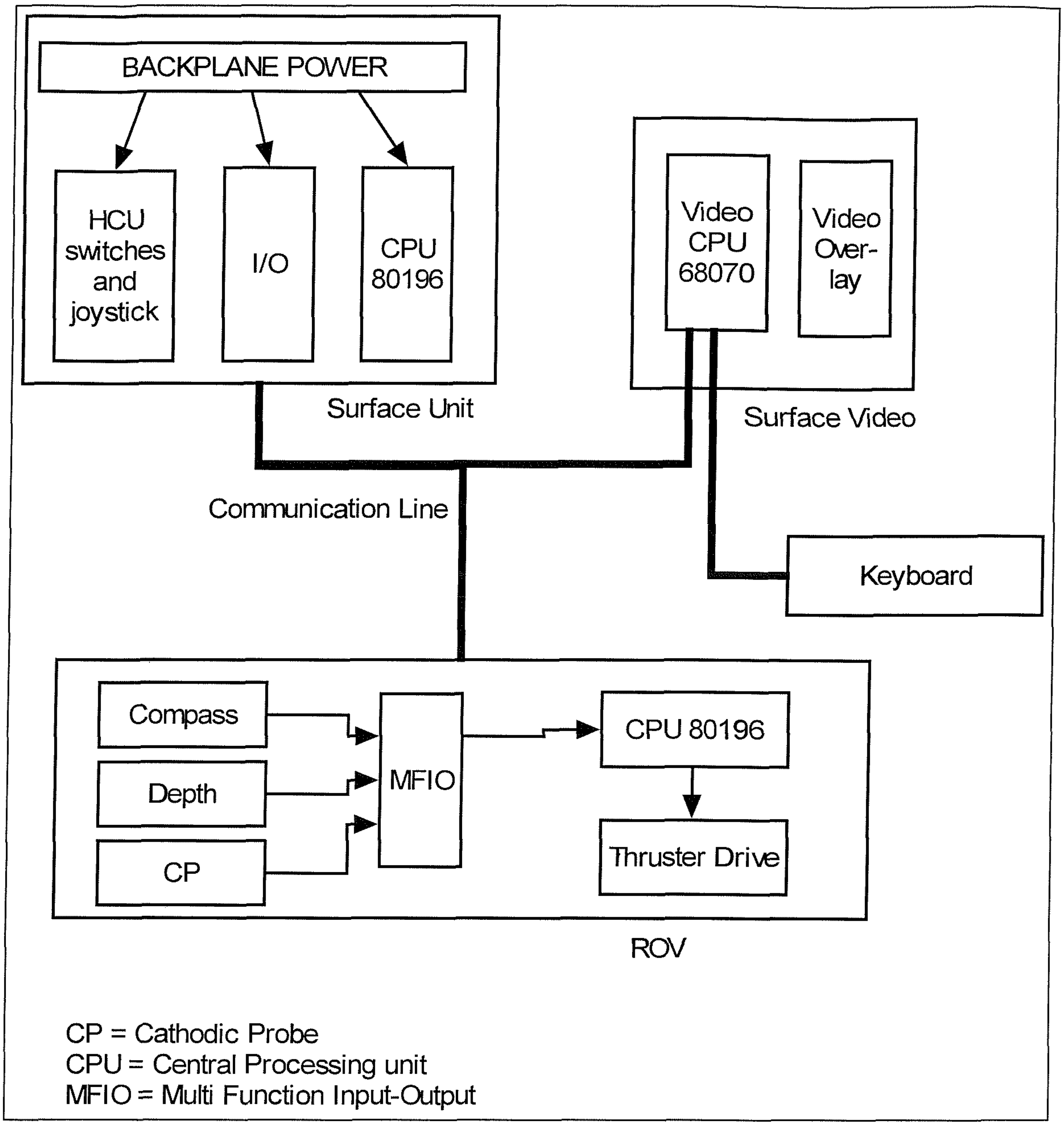


Figure 1.2 Seaeye Vehicle Centralised Architecture

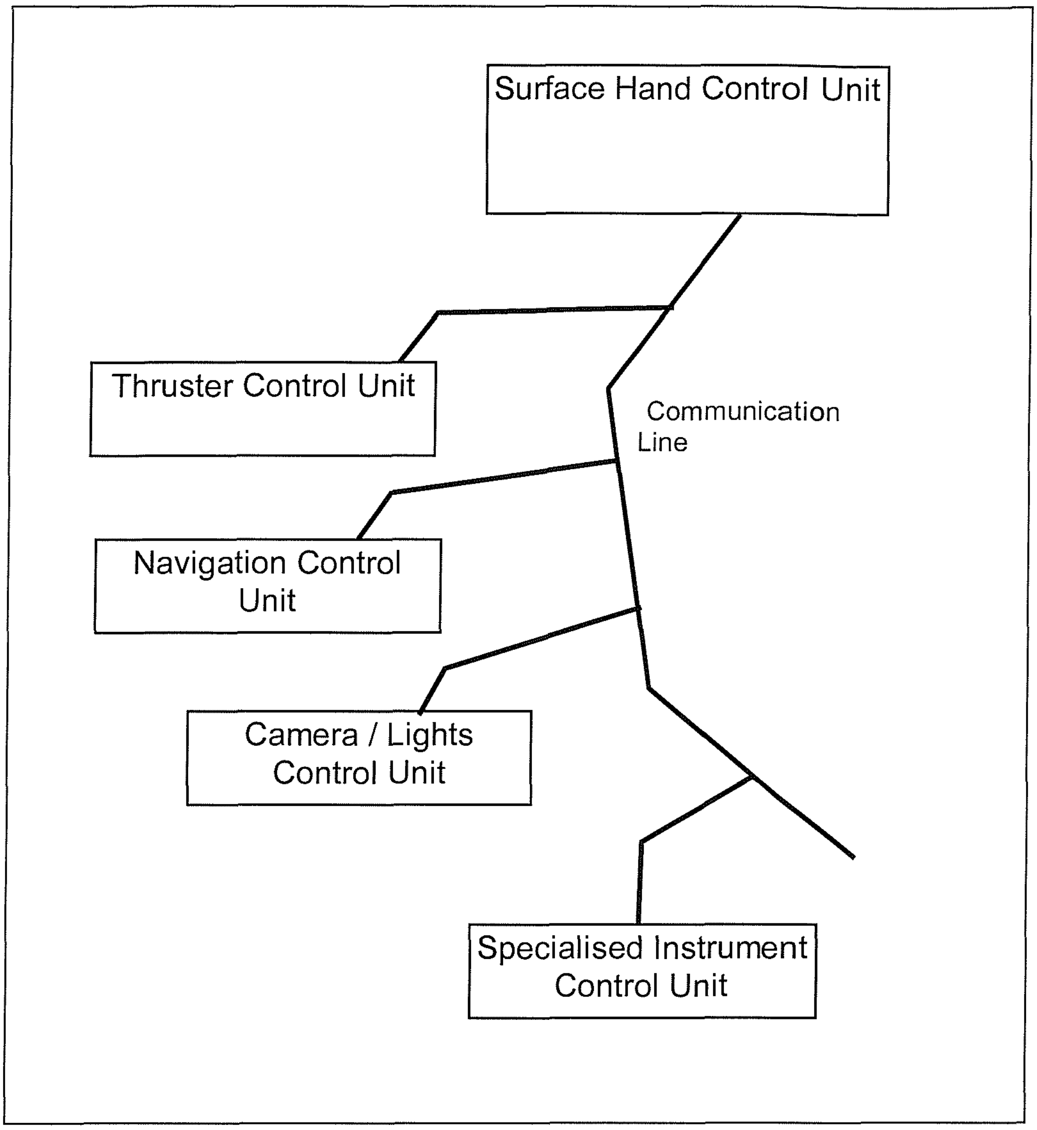


Figure 1.3 An Example of a Networked Vehicle Architecture

The centralised approach is used widely within the industry, most probably for historical reasons. When the first generation of vehicles emerged in the 70's, digital electronics were still very basic, and networks were not a realistic solution. The advantages of distributed architecture have now been recognised in another area of ocean technology. New Autonomous Underwater Vehicles (AUVs) such as Autosub and Martin are based on networks[6][7]. These are research vehicles, and therefore the design exercise is quite different, but the concept of distributed control is similar.

1.6 Research Objectives and Contribution

The aim of the project has been to design a suitable communication system for a ROV. No specification as to the form of communication was given, the choice had to be made with considerations for the application, the problems encountered with existing ROVs, and particularly the ability to run a dynamic controller over the communication loop.

The initial stage consisted of reviewing all communications systems that could be applied; the outcome of the review led to the selection of a Fieldbus-based system. The second stage was to evaluate the suitability of such a system for dynamic control. A simulation of the network was required for that purpose, and the simulation developed has been validated against a real hardware system. Simulation results showed that in some cases the delays originated by the network could cause the instability of a closed-loop control system. The particular problem was that part of the network should be able to be removed or added, without necessarily having to re-tune all controllers. A suitable answer has been found by using adaptive control. An estimate of the process parameters is obtained using a Recursive Least squares estimator, the value of which is then used to calculate the delay of the process. The calculated delay is then used in a self tuning law, in order to alter the controller's parameters.

On the practical side, a prototype communication system has been built, and then fitted within an existing ROV chassis for a demonstration. Most of the elements of this prototype were also used to verify results obtained with theory, such as the network simulation and the self-tuning controller. The prototype system was demonstrated in operation at the Ocean Basin, DERA Haslar, where the main aim was to show that the prototype electronics working in laboratory conditions were operating as expected.

1.7 Layout of the thesis

The thesis is organised in several chapters. The first one offers some background information about ROVs, and general information about the PhD project.

The second chapter presents a review of networks and communication methods, it is followed by a detailed description of the chosen network in chapter 3. A prototype vehicle control system has been designed and built during the project, and has been used as a base for experiments. Details of this prototype can be found in chapter 4. Chapter 5 explains why and how a simulation of the the network has been developed and results from the simulation are also compared with experimental values obtained from the prototype. Chapter 6 is concerned with the effects of a variable delay on a closed loop control system, and describes a self tuning controller able to cope with such systems. The final chapter concludes the thesis.

2. NETWORKS AND COMMUNICATION

2.1 General communication concepts

Computers are now used in every walk of life. In the home, the office, but also in the process and manufacturing industries. Although in most instances they are used to perform their intended role in a stand alone mode, they increasingly need to interchange data with other computers. The type of data exchanged can vary from databases, Email, pictures, to instrumentation and control commands.

The basic requirement in all those applications is the provision of a suitable data communication facility. A wide range of facilities exist, and they have to be suited to the particular application. Inside a computer, information is usually transferred in a parallel mode, i.e. in a 16 bit system, 16 signal lines are dedicated, one for each bit [8].

In order to be transmitted on a data communication line, this information has to be converted to a serial form, where the 16 bits would be transmitted one after another on the same line. Some means of detecting corruption (error control), and of regulating the data rate (flow control) are often provided.

Three modes of operations can be used when information is exchanged between two computers:

i. **Simplex** : This is used when data is flowing one way only. For example a data logging system where the measuring device returns data at regular intervals to a data gathering computer.

ii. **Half duplex** : This is when data is flowing in both ways alternately. For example a data logging system where the data gathering computer sends a request to the measuring device, which then returns some data.

iii. **Duplex** : This is when data is flowing both ways simultaneously.

Data is normally transmitted between computers in multiples of a fixed length unit, usually 8 bits (or a 'byte'). Each byte is transmitted serially, the receiving computer receives one of the two levels which vary accordingly to the bit pattern, making up the message. In order to interpret this bit pattern correctly, the receiving computer must be able to find :

i. **Clock synchronisation**: the start of each bit (in order to sample in the middle of the bit)

ii. **Byte synchronisation**: the start and end of each byte

iii. **Frame synchronisation**: the start and end of each complete message block (or frame)

The above tasks can be executed in one of two ways, depending on whether the receiver and transmitter clocks are independent (asynchronous) or synchronised (synchronous). With asynchronous transmission each character is treated independently, and the receiver's clock is resynchronised at the start of each character received (Figure 2.1).

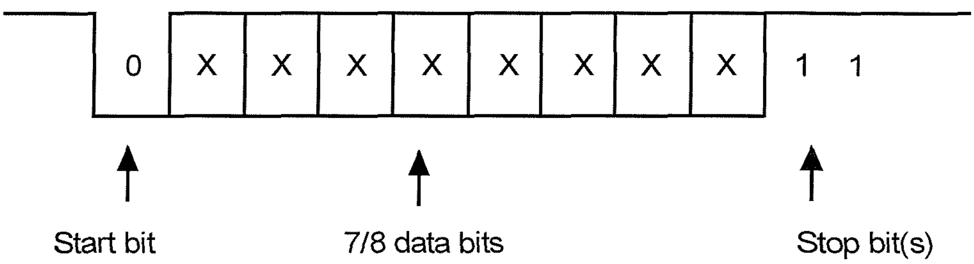


Figure 2.1 Asynchronous transmission of a character

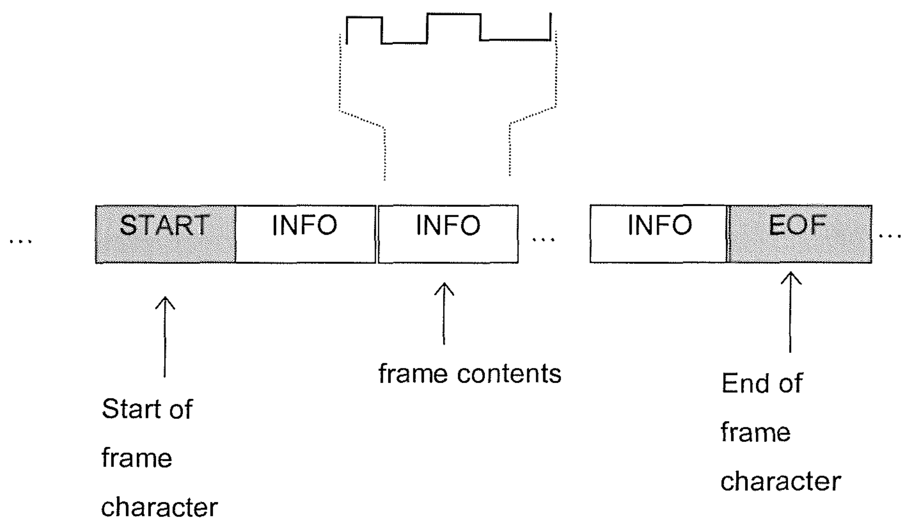


Figure 2.2 Synchronous transmission of a character

With synchronous communication, a complete frame of characters is transmitted in a continuous string of bits. The receiver has to keep in synchronisation for the duration of the complete frame (Figure 2.2).

2.1.1 Error Control

In asynchronous transmission, error control can be implemented by adding a parity bit before each stop bit. The value of that bit is computed by adding together the number of '1' bits in the byte (Modulo 2); the parity bit is chosen so that the total number of '1' bits (including the parity bit) is either even (even parity) or odd (odd parity). When the receiver gets the character, the same calculation is completed, and if the result matches the parity bit, the character is assumed to be correct. This method will allow the detection of all single bit errors.

For synchronous transmission, since block of characters are transmitted, there is an increased probability that a frame would be corrupted. It is possible to extend the parity bit method described above, by assigning a parity bit for each character transmitted (row parity), as well a bit for each bit position in the complete frame(column parity). A more robust method is to use polynomial codes, where a single set of check digits is computed for each frame. The receiver then performs a similar calculation on the frame and check digits. A fixed result is expected when no errors have been induced. This method is known as Frame Check Sequence (FCS) or Cyclic Redundancy Check (CRC).

2.1.2 Encoding

Encoding is the way in which the binary data ('0' or '1') is represented as electrical signals. Many ways of encoding data are available (Figure 2.3). For asynchronous communication, where no clocking information needs to be transmitted, the most common encoding method is Non Return to Zero (NRZ) where '1's and '0's are encoded as positive or negative voltage levels on the transmission line.

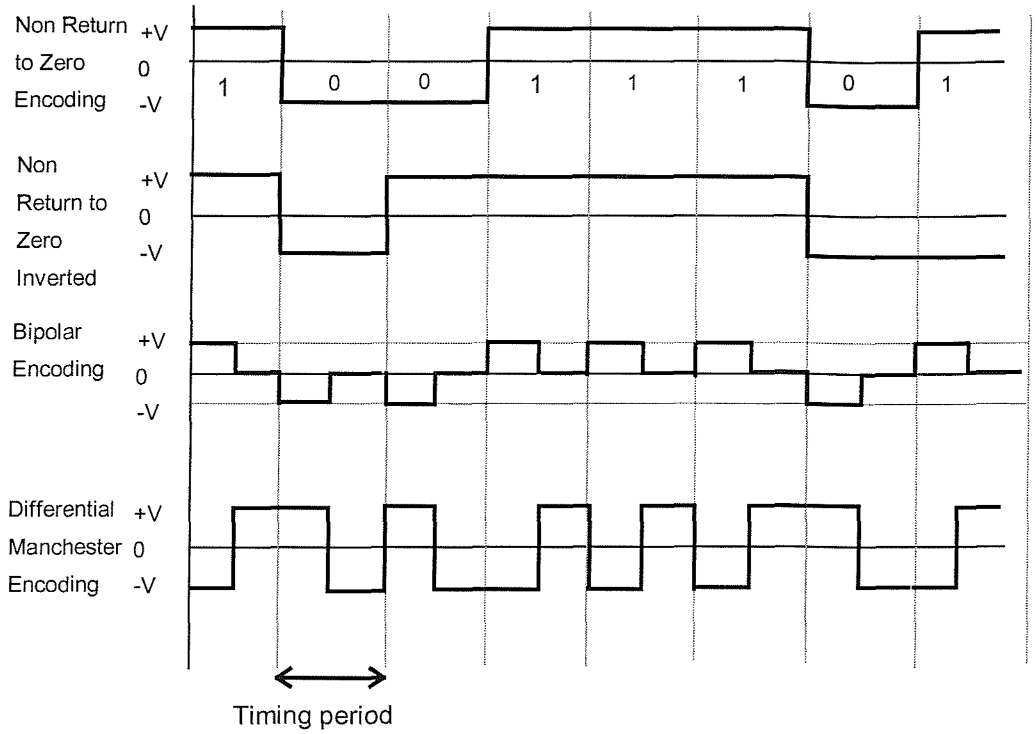


Figure 2.3 Different types of data encoding

For synchronous transmissions, the clocking information is often embedded in the bit stream. For example in bipolar encoding, a binary '1' is represented by a positive pulse, and a binary '0' by a negative pulse. As shown on Figure 2.3, the sequence '1001' is represented by a positive pulse '1', followed by two negative pulses '00' and then a positive pulse '1'. Since there is always a change in the signal at each clock period, the clocking information can be retrieved by the receiver. Differential Manchester encoding follows the same principle, except that pulses are now replaced with falling or rising signal transitions.

In Non Return to Zero Inverted (NRZI) encoding, a transition represents a binary '0', and no changes represent a binary '1'. If the data being send consisted of only '1' no transitions would be present on the line, and the receiver would loose track of the timing information. In order to ensure that enough transitions are received in order to recover the clock information, a '0' is inserted after five consecutive 1's (known as 'bit stuffing'). The clock information is recovered using a technique known as Digital Phase Lock Loop (DPLL).

Manchester encoding is mainly used in Local Area Networks, in relatively short cable runs. NRZI is favoured for longer distances, as each bit occupies a full width pulse, making it less error prone.

2.1.3 Flow control

Flow control ensures that when two devices are communicating, the receiving device has sufficient storage space to hold the data that is transmitted. This can be implemented in software, by using dedicated messages to confirm that the station is ready to receive the next message. Flow control can also be implemented in hardware, by having dedicated signal lines indicating whether a device is ready to accept incoming data. The hardware method has the disadvantage of requiring more physical data lines between the devices, the software requires that some special messages are reserved for controlling the flow, therefore modifying the original data. This lack of transparency can sometimes be a problem, when trouble shooting communications failures for example.

2.1.4 Data link protocol

The data link protocol deals with error correction and flow control. It also defines the format of communication, i.e. the number of bits per digit and the type of encoding used. The protocol also specifies the type and order of messages that are exchanged. For example, the messages that are exchanged when first establishing a communication, what the procedure is when an error is detected.

2.2 Local area networks

2.2.1 Topologies

In the context of networking, the first basic characteristic to consider is the way in which the end points, or stations are interconnected (Figure 2.4).

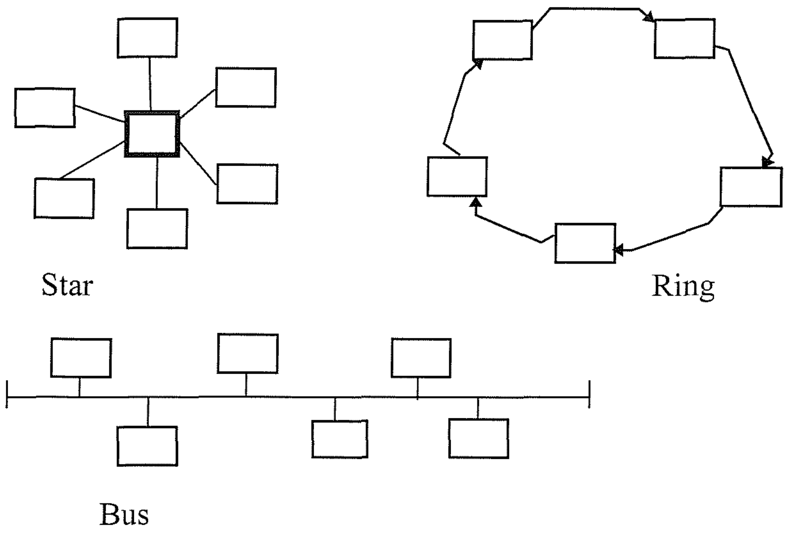


Figure 2.4 Network topology

In a star topology, each station is connected to a centre node, and in order for one station to exchange data with another, all messages have to go through the centre node. With a bus topology, a single line is used (for example a cable) and is connected to each station. With a ring topology, each node is interconnected to its neighbour via a unidirectional connection, so that the group of nodes forms a complete ring.

The physical signal paths, or transmission media, that have commonly been used for local area networks are twisted pair cable, coaxial cable and optical fibre.

The introduction of such topologies requires some form of management to regulate the access to the medium and to resolve issues such as addressing (i.e. to ensure that a message can go from one node to the other, we need to know who has access to the medium, and a way of identifying each node).

2.2.2 OSI model

In an effort to facilitate the process of designing internationally compatible communication systems the International Standards Organisation (ISO) have defined a multi-level communications protocol model. This model is designed to be used as a guideline for the development of actual protocols, employing a strategy known as Open System Interconnect (OSI). The ISO/OSI model describes the flow of data across a network as a downwards progression through different layers, from the application layer to the physical layer, across the physical medium and back up the stack of the receiving station (Figure 2.5).

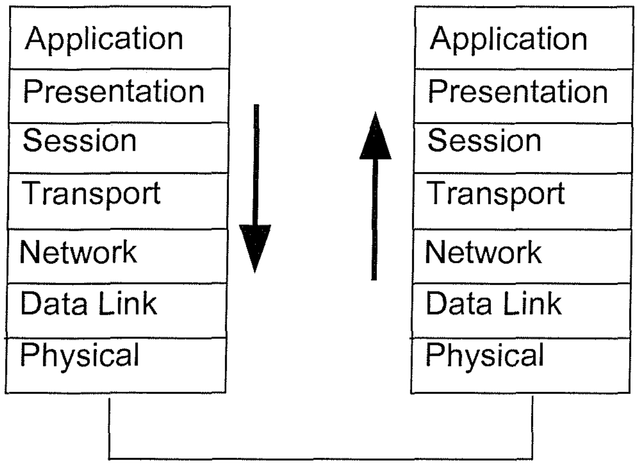


Figure 2.5 The 7 Layer OSI Model

The physical layer deals with the mechanical and electrical interface to the medium, the data link layer deals with the way data is formatted, usually some sort of low level error recovery is also implemented. The network layer deals with addressing issues, the transport layer deals with flow control and error control. The upper three layers deal with aspects related to the application itself such as data representation, transfer syntax etc...

2.2.3 CSMA/CD (Carrier Sense Multiple Access with Collision Detection)

Local area networks are widely used in the office environment as a link between computers. The most common networking standard is the IEEE 802.3 standard, more often known under its trademark name as Ethernet. It is based on a 7 layer Open System Interconnect (OSI) model. The MAC (Media Access Control) is based on is the CSMA/CD (Carrier Sense Multiple Access with Collision Detection).

CSMA/CD is a method of controlling bus access. Each node is free to transmit at any time. When a node tries to access a busy bus, a collision is detected, and the transmission is corrupted (Figure 2.6). To make sure that all the nodes involved in the collisions are aware that the collision has occurred, a random bit pattern is send for a short time Tj. This is the jam sequence. The two nodes then wait for a short random time before trying to retransmit. This type of bus access is probabilistic and depends on the network loading. Under worst case conditions the amount of time to detect the collision is twice the propagation delay tp.

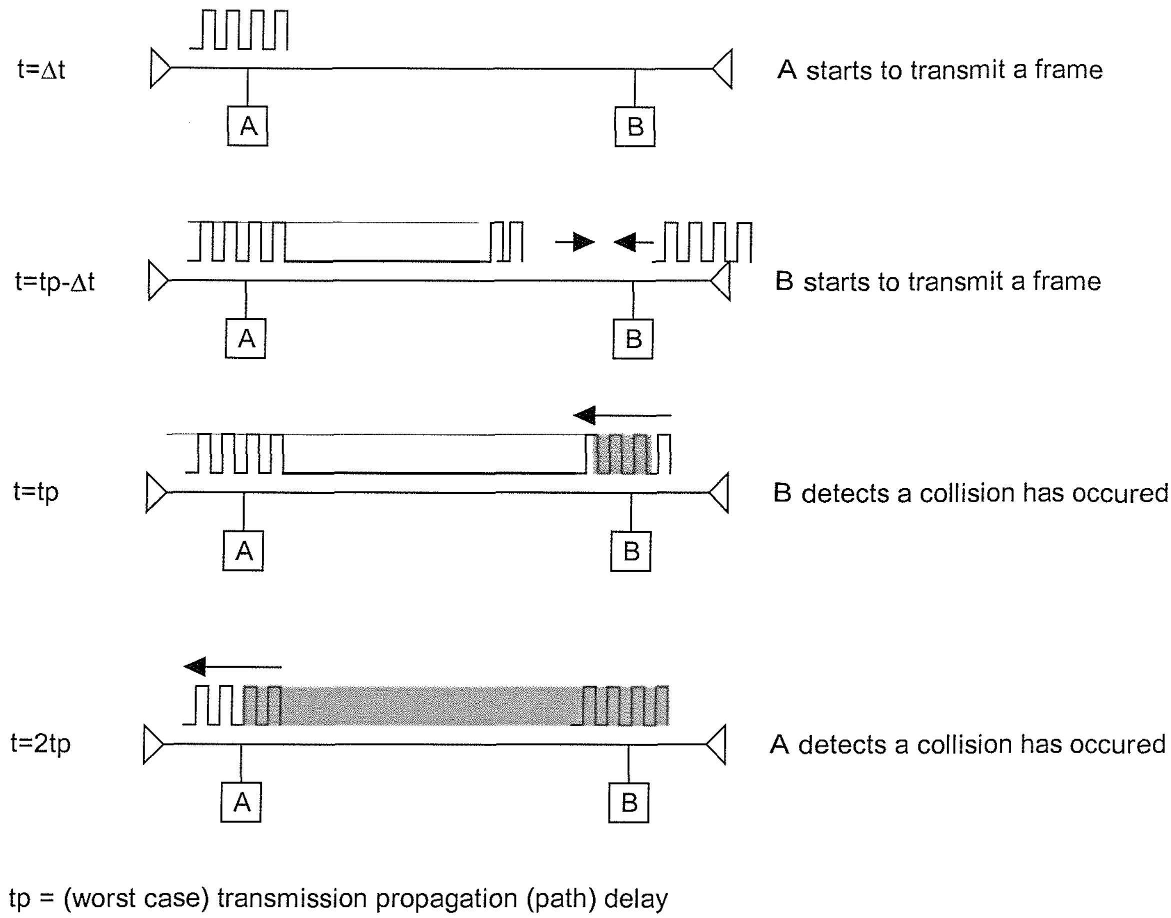


Figure 2.6 CSMA/CD mode of operation

We can guess that this system is very efficient for long messages. In the case of short and very frequent messages from a few nodes, as in our case, the number of collisions occurring will rise and some nodes might not be able to get access to the line at all. As this system is not deterministic , i.e. it is not guaranteed that a node will gain access to the network, it is not a good solution for our real time system.

2.2.4 Token Passing

Another widely known network standard is the IEEE 802.4, also known as token bus. Like CSMA/CD it is based on the 7 layer OSI model, although the topology is bus. The nodes are considered as a logical ring. That is, the stations assume an ordered sequence, and each station knows the identity of the stations preceding and following it. A control frame, known as Token regulates the right of access. The station receiving the token has access to the medium for a limited time, and must pass the token to the next station when it has either nothing to send on the medium, or it has finished using the medium or the station's time has expired

(Figure 2.7). The advantages of this standard is that each node is guaranteed to have access on the medium, which makes it ideal for a real time application. It also means that the implementation is much more complex, having to deal with the logical ring management and fault detection .

Token ring, or IEEE 802.5, is very similar to the token bus standard, and is based on a ring topology. When all the stations are idle, the token circulates on the medium, if a station wishes to use the medium it must seize the token by changing one bit on the token pattern. The transmitting station will return the token to the ring once it has finished. There also is a priority mechanism, allowing certain stations to seize the token before the others.

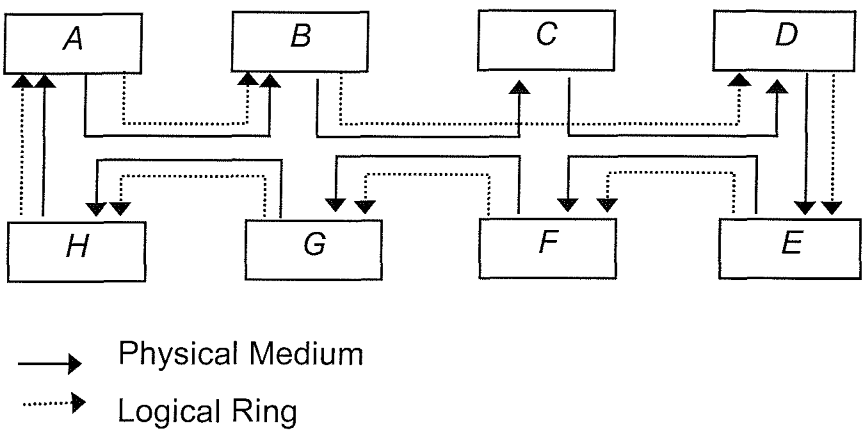


Figure 2.7 Example of a token passing loop

Figure 2.7 shows a token passing loop where node C is inactive and has been removed from the logical loop. Each node has a logical predecessor and successor.

2.2.5 Other methods of medium access control

To keep up with the progress brought by the optical fibre technology, the FDDI (Fibre Distributed Data Interface) has been derived from the Token Ring standard and can support higher data rates, 100 Mbps for FDDI, compared with 1 or 4 Mbps for Token Ring [9],

2.2.6 Response to traffic load

Figures 2.8 and 2.9 show the effect that traffic load has on the delay[1], for both CSMA/CD and token passing protocols. Those are for a 50 node network, with a data rate of 10 Mbps for the token passing and 20 Mbps for the CSMA/CD; a packet length of 1000 bits; a medium length of 2000 meters and, where applicable, a token length of 10 bits.

These are the results from a simulation program by Sadiku [10], and represent a statistical average, the error bar shows the 95% confidence interval. The scales for each graph is different to highlight the different behaviour of each method as traffic increases. It shows clearly that a token passing method copes with traffic increases in a better way than CSMA/CD. As traffic increases, the delay not only increases sharply, but it becomes less and less predictable (shown by the confidence interval).This is the major issue that lead to the choice of a token passing system for our network.

[1] The delay being the time difference between when the message is available at the sender station and when this message is received correctly by receiver station.

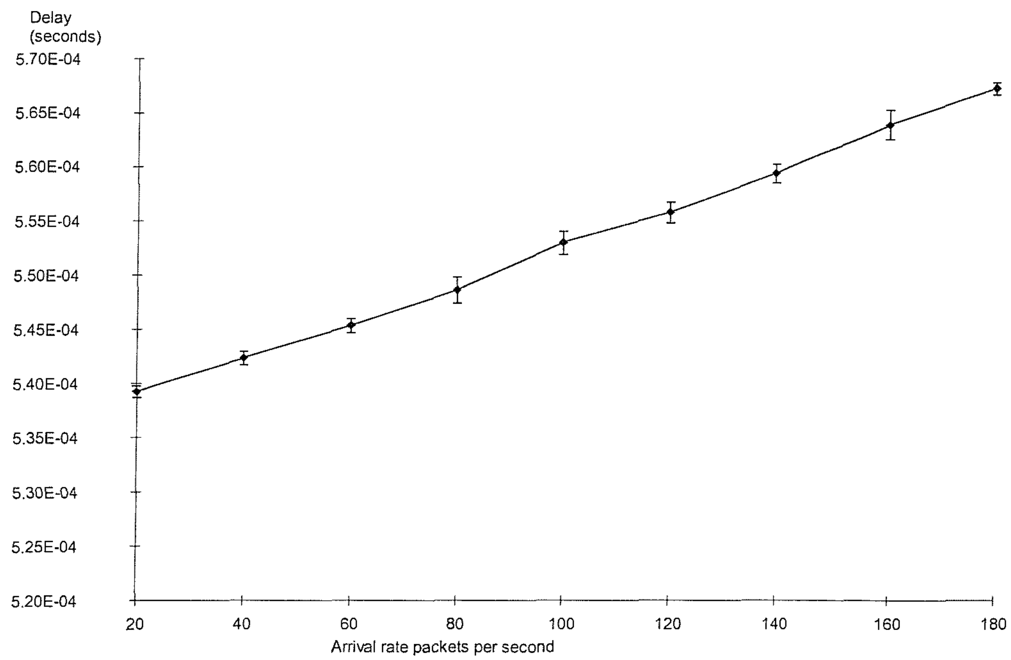


Figure 2.8 Delay versus traffic load with CSMA/CD

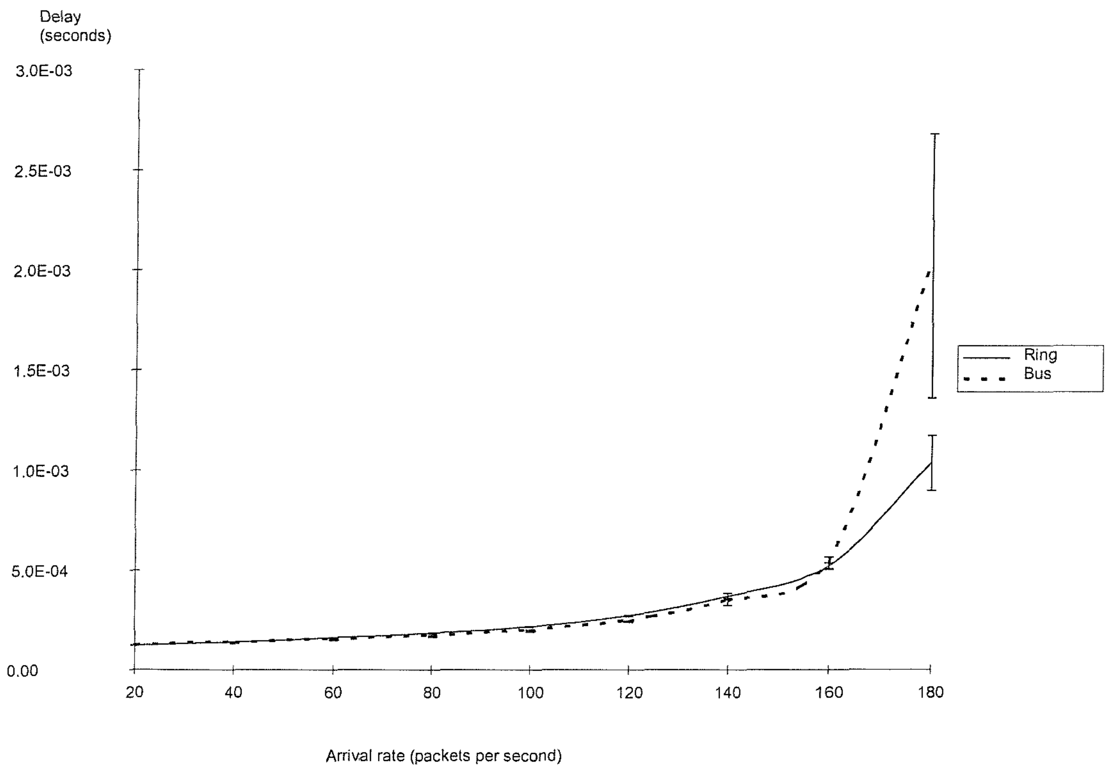


Figure 2.9 Delay versus load with token-passing protocol

2.3 *MAP and Fieldbus*

Another standard has been derived from the token bus standard : MAP (Manufacturing Automation Protocol). It has the advantage over Ethernet and token bus of having been designed for real-time networking. It is used in the USA by major companies like General Motors, who designed it, and also by Boeing, Kodak etc. This technology did not take on in Europe, mainly because of its high cost.

The FieldBus concept has evolved from the MAP protocol. As MAP was covering the full 7 layer OSI model, the time response was limited for real time applications, and thus was not adapted to low level instrumentation. The concept of FieldBus is to use a "collapsed" version of the OSI model, reducing it to a three layer model, containing only the application layer, the data link layer and the physical layer.

Some national standards already exist, such as FIP in France and Profibus in Germany, each being influenced by their target application. FIP emphasised an accurate time response, while Profibus emphasised sharing the bus resources. However the requirements of the multi-national user companies led to demands for an international standard. The IEC Fieldbus standard (International Electro-technical Commission) has been developed through international agreement using the best features of the leading industry and national standards, this process is very slow, and the complete standard has not been approved yet. National and commercial self-interests make the voting process for the standard's agreement very difficult, and the prospect of having an agreed international standard is very small [11].

Meanwhile, other types of protocols are being developed for very specialised targets. Lonworks, for example, for House Automation, HART, CAN and VAN for the Automotive Industry [12][13]. All these proprietary solutions have two major inconveniences. Firstly, the application layer was designed for very specific application, and therefore it is very likely that it would need modifying, secondly the development kit can be very expensive.

Since the prototype communication system could potentially be used as commercial solution by the sponsor company, it was felt that committing to a particular supplier would have been a burden. A customised prototype system based on Fieldbus has been designed and built as part of this project for experimental purposes, and is used to implement a distributed architecture on the ROV.

2.4 *Design features*

2.4.1 Losses and Reflection

Digital systems require the transmission of signals to different elements on the system. The high frequency components of a step input are attenuated and delayed more than the low frequency components, mainly due to skin effect. As a result a pulse is distorted, as shown in appendix B.

2.4.2 Signal Distortion due to Rise Time and Duty Cycle

The duty cycle of the transmitted signal also causes distortion. The effect is related to the rise time. If the signal has a 1/2 (50%) duty cycle and the threshold of the receiver (Vth) is halfway between the logic levels, the distortion is small . When the duty cycle decreases, the signal is considerably distorted and might not reach the threshold level at all (Figure 2.10).

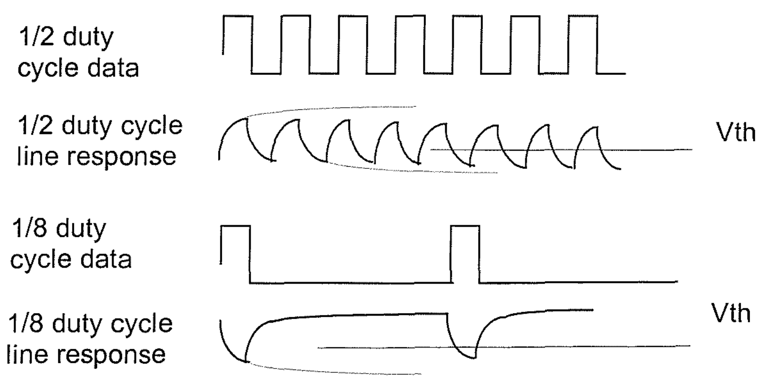


Figure 2.10 Signal distortion due to duty cycle

If the threshold level of the receiver is not halfway between logic level one and zero, the receiver will contribute to the distortion effect. As shown on Figure 2.11, a pulse would be either lengthened or shortened.

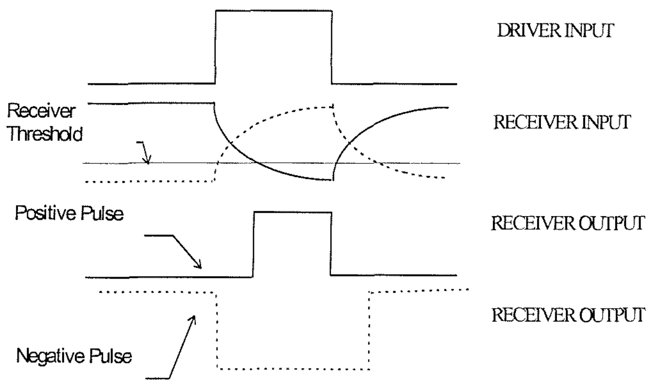


Figure 2.11 Signal distortion due to threshold level

2.5 Functional aspects

2.5.1 Delays

The propagation speed of a signal on a twisted pair cable is[2] typically 2.10^8 m/s. So for a 1 km long line, the propagation delay $T_p = 1000 / 2.10^8 = 5\ \mu s$. Values for cables up to 1km long were computed, and can be obtained from Figure 2.12.

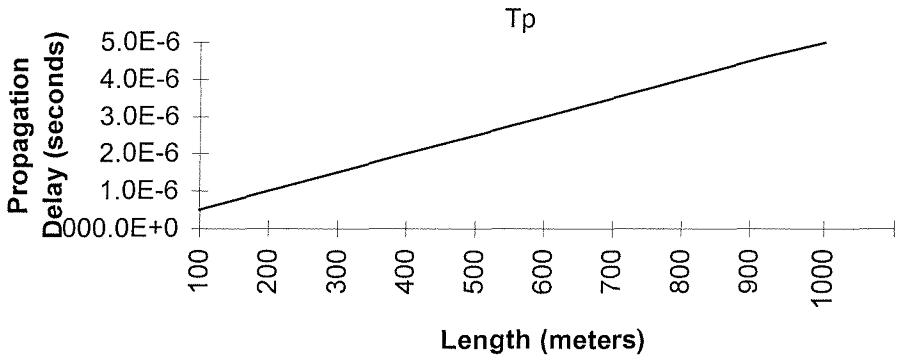


Figure 2.12 Propagation delay chart

For a frame of 10 bytes (80 bits), and with a transmission speed of 10.5 Kbd, the transmission delay is :

---

[2] This is the typical speed of the signal for twisted-pair or coaxial cable [1]

Tx = 80 / 10500 = 7.6 ms. Other values can be obtained from (Figure 2.13).

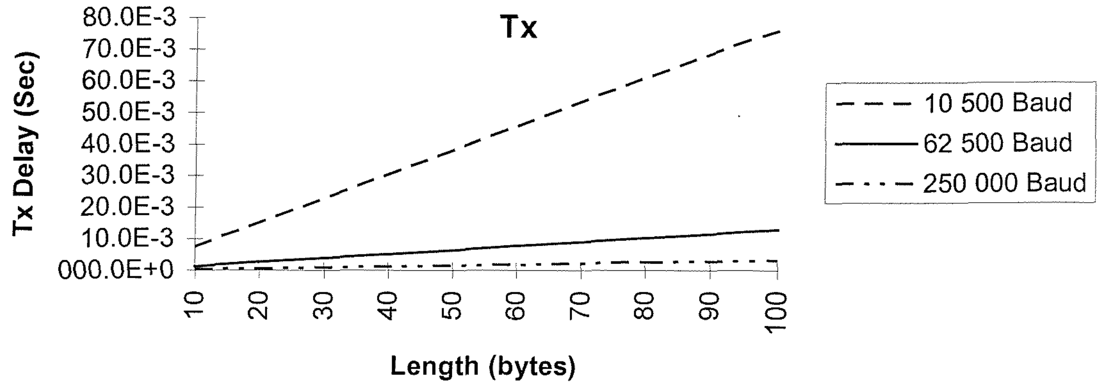
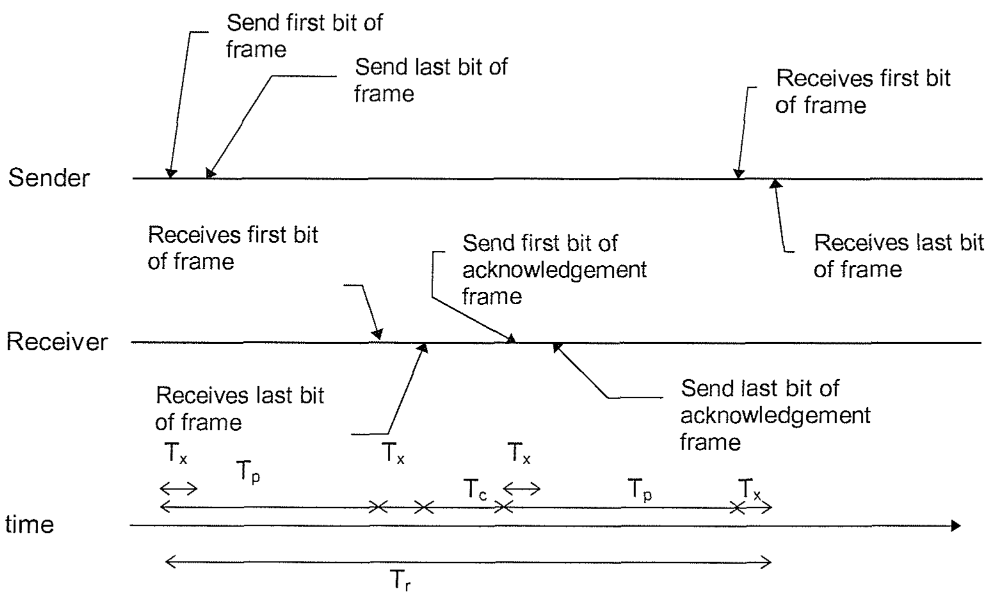


Figure 2.13 Transmission delay chart

The ratio a= T_p/T_x is much smaller than 1. The transmission delay dominates the 'round trip delay', that is the time delay between the first bit of a block being transmitted by the sender and the last bit of its associated acknowledgement being received (Figure 2.14).



T_r = round trip time
T_c = computing time
T_x = Transmission delay
T_p=Propagation Delay

Figure 2.14 Timing Diagram

2.5.2 Noise

Noise in received signals constitutes the most prevalent factor limiting the performance of a communication system, since noise introduces errors in the receiver. One source of noise is crosstalk. It is due to capacitive coupling between two lines, and is significant in high speed circuits.

Another form of noise caused by external activity is impulse noise. An example would be a lightning discharge. Its main characteristic is that it occurs in bursts. A burst of half a second might corrupt 4800 bits of data at a transmission rate of 9600 bps. Error decoding techniques allow that type of error to be detected.

A third type of noise, thermal noise, is present in all types of electronic device. It is due to the thermal agitation of the electrons, associated with each atom making up the device or transmission line material. It is made up of random frequency components, across the whole spectrum, of continuously varying amplitude (white noise). A minimum signal level must be used to achieve a minimum Bit Error Rate (BER). For example a BER of 10^{-4} means that on average, 1 bit every 10^4 received will be misinterpreted.

It is possible to calculate the BER caused by a defined amount of noise (appendix A). For example, with a differential value of 5V and a noise variance of 525 mV, we get a BER of about 10^{-6}. However, this is for the case only of a single ended transmission line. For a differential transmission line, two signals of equal and opposite polarity are produced for every bit to be transmitted, the receiver is sensitive only to the difference between the two signals. Any noise picked up in both wires will have its effect cancelled at reception.

A fourth type of noise is intersymbol noise, when the transmission rate increases, some frequency components associated with each bit are delayed and interfere with a later bit.

All those sources contribute to the total error rate. The error rate can be reduced by increasing the signal level or by implementing error-control coding techniques in the higher level of the protocol.

2.6 Selected methods

In light of the review carried out in the above paragraphs, a particular approach was selected for a prototype system. The deciding factor was the selection of a token passing network, rather than a collision based one. The reason was the behaviour of such networks as traffic increases (Figure 2.8 and Figure 2.9). Other implementation choices mentioned above were made with respect to the actual possibilities available in hardware. Error control algorithms and encoding rules are often embedded within communication hardware.

The choice of the topology was of a bus type, although each node is within a logical ring on the network.

The choice of the baud rate, which would closely affect the delays, was left open. The value of the baud rate that can be used is limited by the length and quality of the cable, which could vary. By using a low value by default, the worst cases are dealt with.

The software implementation of the prototype network is described in the following chapter.

3. IMPLEMENTATION OF THE NETWORK

This chapter describes how the communication concepts described in the previous chapter were selected and implemented. The selection of a particular network was driven not only by functionality, as described in the previous chapter, but also by commercial aspects, such as the availability and costs of hardware and software components. Technical aspects also came into consideration as the final system has to fit in a relatively small enclosure, with a limited power supply.

One of the initial decision made was to use a token passing network, as having a deterministic response was identified as a key issue for such a control system. A Fieldbus type network was seem as an adequate implementation. At the time, the fieldbus standard was only partially specified with only the lower protocol layers defined. There was also many uncertainties in the industry about the future of the standard, which was evolving very slowly compared to similar proprietory networks such as CAN and Profibus.

The cost of developing proprietory fieldbus solutions such as CAN or Lonworks was above the projects's budget, and the sponsor company was concerned about committing a design to a third party supplier.

The choice was made to implement a basic fieldbus version by using industry standard communication controllers and microprocessor. The initial decision was to use and Intel 8344 microcontroller, which includes a serial communication controller and allowed all the necessary hardware to fit in a small space. The familiarity and popularity of the Intel microcontrollers was also a great advantage, as a choice of software development kits was widely available. The microcontroller implements several communication features in hardware : NRZI encoding, SDLC (Synchronous Data Link Control) framing, which includes FCS error correction . The topology was chosen to be bus, as it suits the layout of the ROV hardware better.

During development, the speed at which the network was to be run was initially set at 9600 baud, the hardware could support speeds up to 62Kbds and this value is easily changeable in software if needed. There was no need to increase this value in development.
The higher level of the communication protocol was implemented in software, written in C language. The includes feature such as token management and recovery, and data format. Part of the protocol was taken from the SDLC specification, some new features were added to adapt the network to the ROV appication.
This technology was implemented in a prototype vehicle, described in the next chapter.

3.1 SDLC

A typical SDLC frame consists of five fields (Figure 3.1): flag, address, control, information and Frame Check Sequence (FCS). The FCS is used to check for transmission errors between the two data link stations, this is implemented by a Cyclic Redundancy Check (CRC). The transmitting station performs Modulo 2 division, based on an established polynomial, on the address, control and information fields and appends the remainder as the FCS field. In turn the receiving station performs a division with the same polynomial. If the remainder equals a predetermined value, the chances are very high that the transmission occurred without any errors. Otherwise, it indicates a probable transmission error, in which case the receiving station sends a negative acknowledgement.

Opening Flag	Address Field	Control Field	Information Field	Frame Check Sequence	Closing Flag
01111110	8 bits	8 bits	Variable length(only in Information frames)	16 bits	01111110

Figure 3.1 SDLC frame format

3.1.1 Error correction in HDLC

In HDLC[3] , the generator polynomial used for error correction is [8] :

$g(x) = x^{16} + x^{12} + x^{5} + 1$. The FCS is calculated using the following method :

Let M a k-bit number representing the frame contents, R an n-bit number, such that k>n, representing the FCS, and G an (n+1)-bit number representing the generator polynomial.

$$\text{if } R = \frac{M \times 2^{n}}{G} \text{ (Modulo 2) then } \frac{M \times 2^{n} + R}{G} = 0$$

$$\text{can be checked since : } \frac{M \times 2^{n} + M \times 2^{n}}{G} = 0 \text{ (Modulo 2)}$$

The FCS (R) is calculated using a Modulo 2 multiplication and division. The FCS is decoded by checking that the second expression is zero.

3.1.2 Limitations of HDLC due to framing and bit insertion

There is still a possibility that an error remains undetected, for example if a single bit error generates a spurious flag. As in Figure 3.2, if the sequence '01110110' is to be transmitted, a single bit error could generate a spurious flag, '01111110'. The leading and trailing '0' have to be transmitted error free, while the '0' in position 2 to 7 has to be affected by bit errors in order to generate a flag.

```
position 1 2 3 4 5 6 7 8
octet    0 1 1 1 0 1 1 0
bit error

FLAG     0 1 1 1 1 1 1 0
```

Figure 3.2 Occurrence of a spurious flag

The probability of this type of error happening is described in Appendix I. Figure 3.3 shows how the probability of such an error happening R(FLAG) increases, as the probability p of a bit error varies, for various message lengths. The chance of this error happening can also be reduced by implementing other error detection protocol at higher protocol level.

[3] HDLC: High Level Data Link Control, of which SDLC is a subset

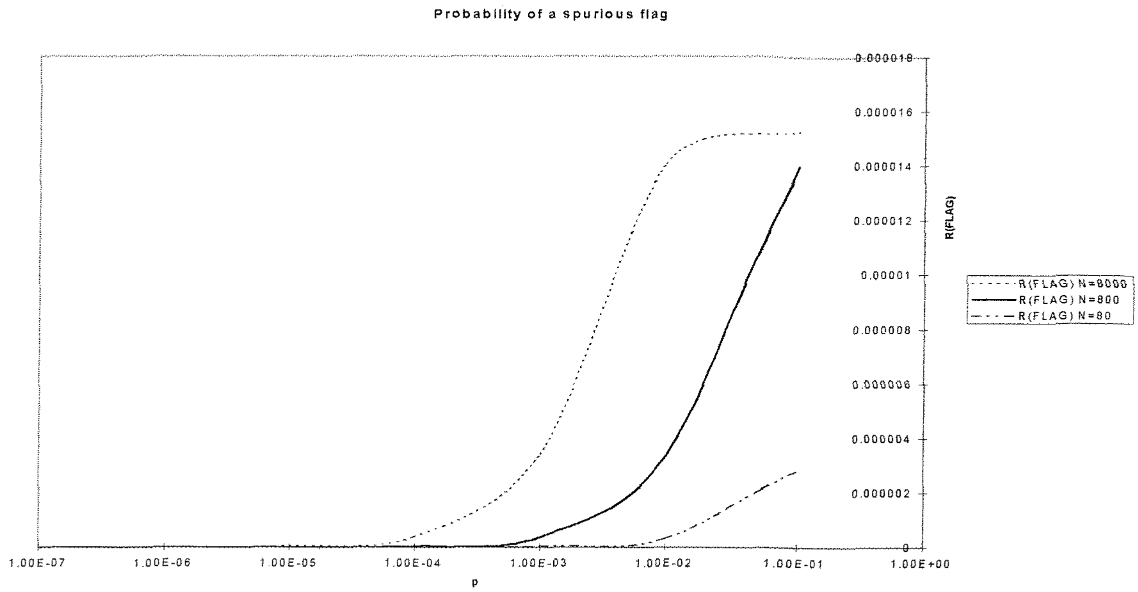


Figure 3.3 Residual error due to spurious flags

3.2 Protocol design

3.2.1 Control field

A fixed number of control fields are used, some were taken from the HDLC specification (Table 3.1). As well as the standard control fields, custom control fields were defined for this particular project (Table 3.2). Those are used for implementing the token passing protocol, as shown in section 3.2.4.

Control Field Bit Encoding										
Format	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Commands**	**Responses**
Unnumbered	1	1	0	0	□	0	0	0	UI	UI
	1	1	0	0	□	0	0	1	SNRM	
	1	1	0	0	□	0	1	0	DISC	RD
	1	1	0	0	□	1	0	0	UP	
	1	1	0	0	□	1	1	0		UA
	1	1	1	0	□	0	0	1		FRMR
	1	1	1	1	□	0	0	0		DM
	1	1	1	1	□	1	0	1	XID	XID
	1	1	0	0	□	1	1	1	TEST	TEST

Legend

I	Information	XID	Exchange Identification
UI	Unnumbered Information	DM	Disconnect Mode
SNRM	Set Normal Response Mode	□	The P/F bit
DISC	Disconnect	FRMR	Frame Reject
RD	Request Disconnect	TEST	Test
UP	Unnumbered Poll	UA	Unnumbered Acknowledge

Table 3.1 Standard HDLC control field used

Custom Control Field Bit Encoding										
Format	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Commands**	**Responses**
Unnumbered	1	1	1	1	□	1	1	0	SS	UA
	1	1	1	1	□	1	0	0	SP	UA
	1	1	1	1	□	0	1	0	WFM	
	1	1	1	1	□	0	0	0	STTRT	
	1	1	0	1	□	0	0	0	CTF	
	1	1	0	1	□	0	0	1	TOK-A	
	1	1	0	1	□	0	1	0	TOKEN	

Legend

SS	Set Successor	UA	Unnumbered Acknowledge
SP	Set Predecessor	CTF	Claim Token Frame
WFM	Who Follows Me	STTRT	Set Target Token Rotation Time
TOK-A	Token Acknowledge	TOKEN	Token

Table 3.2 Custom Control Field Bit Encoding

An example of how those messages are used is given in Figure 3.4. The primary station starts by establishing communication with the secondary station by sending a SNRM (Set Normal Response Mode) message. When the secondary station has acknowleged with a UA (Unnumbered Acknowledge), the actual data transfer can start. The stations can send information frames (UI unnumbered Information) or test messages (TEST).

In order to disconnect its connection to the secondary station, the primary station sends a DISC (Disconnect) message. Following this test messages send to the secondary station are answered by a DM (Disconnect Mode) message, rather than TEST.

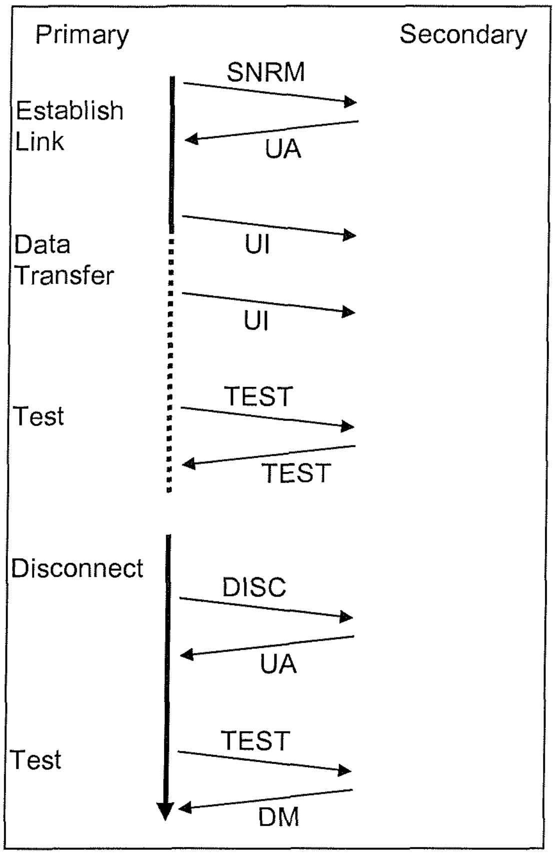


Figure 3.4 Example of SDLC transfer

3.2.2 Error management on protocol level

When data is corrupted during transmission, there are two ways the protocol can deal with the problem, either a retransmission is requested as in Figure 3.5, or the data is ignored as in Figure 3.6.

The overhead incurred in the retransmission case by the possible reception time-out and retransmission request can be a problem when dealing with fast changing data. By the time the same message is retransmitted, the value could be obsolete.

The non-retransmission case is better suited to fast changing data, for example the heading value of a ROV. In the implemented protocol, each node will transmit recent high priority data when it owns the token, thus we know that data is retransmitted within a certain time.

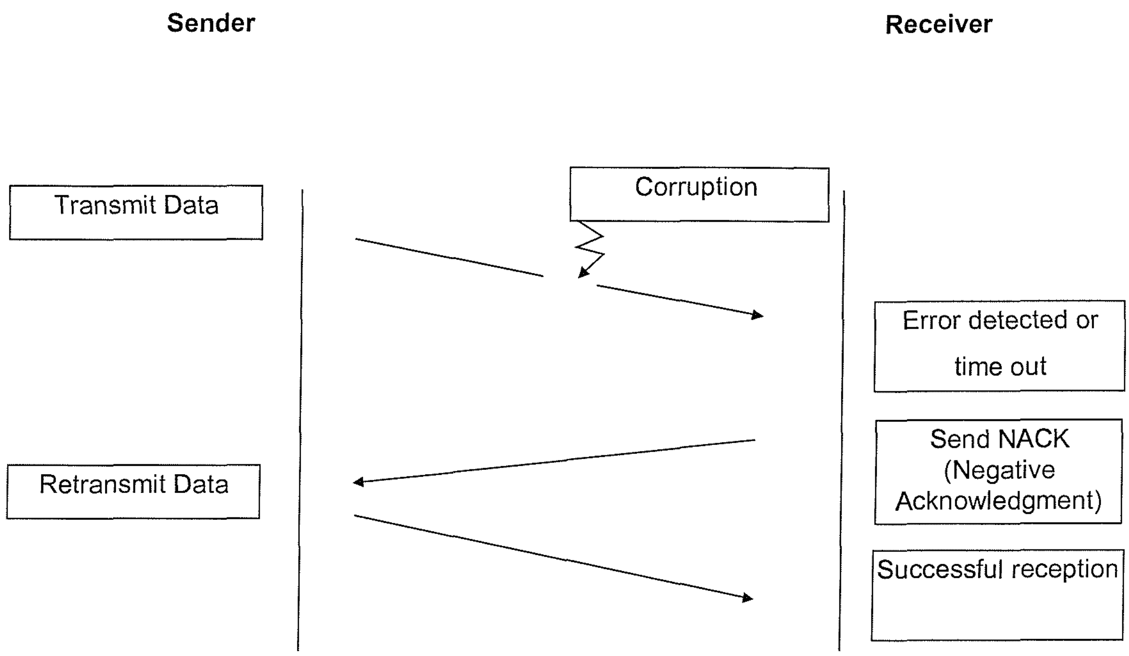


Figure 3.5 Corrupted transfer with recovery

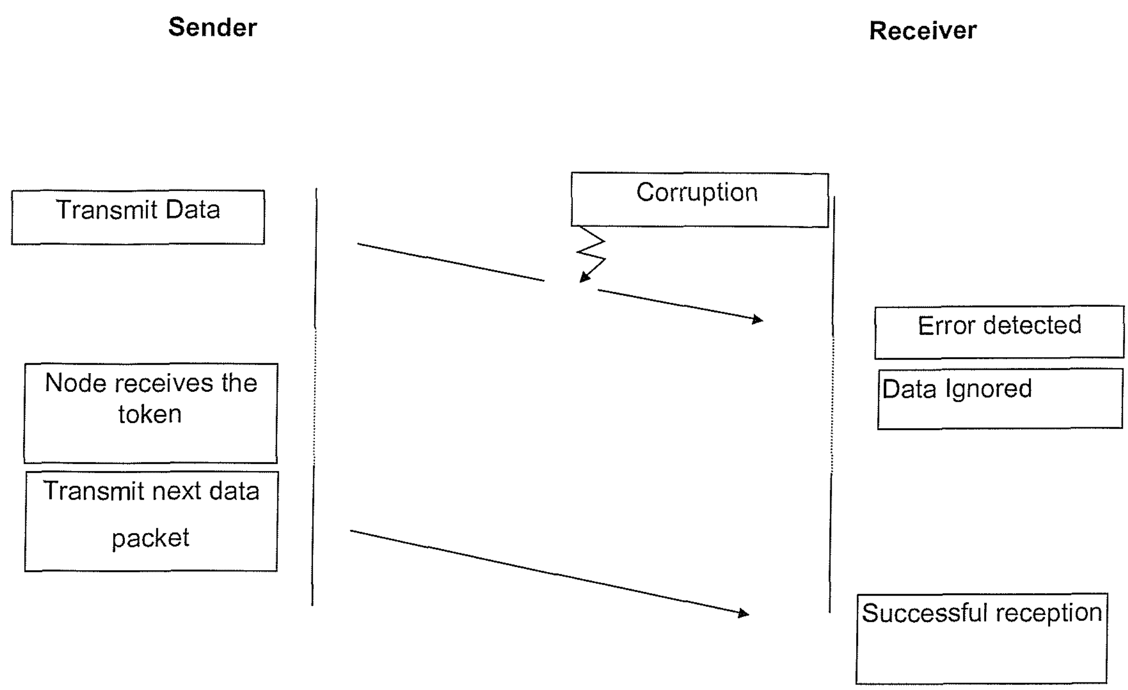


Figure 3.6 Corrupted data transfer without retransmission

In our protocol, the data transmissions use a non-retransmission transfer, however for messages used in the token passing protocol, a transfer system is implemented. Should the transmission of the token be corrupted, this would be detected, and recovery procedure can be triggered.

3.2.3 Addresses

Ranges of addresses were reserved for certain type of nodes. This facilitates future changes. These addresses are defined as:

- 0x81 to 0x90 Thruster card compatible nodes. (e.g. Node 1)
- 0x91 to 0xA0 Navigation card compatible nodes (e.g. Node 2)
- 0xA1 to 0xB0 Camera control node (e.g. Node 3)
- 0x10 for Surface Unit (e.g. PC)

3.2.4 Token passing

This high level function is implemented in software. The flowcharts in Figure 3.7 and Figure 3.8 show how it is implemented. Each node knows the address of its successor and predecessor in the logical ring. The token is passed around the ring and nodes only have control over the media while they own the token.

This version implements a two-level priority mechanism. Each node needs to keep two timers: the Inactivity Timer, which can detect, for example if a token is lost; and the Token Rotation Time TRT timer, which monitors the time since the node last had the token. Low priority frames are only transmitted if the TRT timer does not exceed the Target Token Rotation Time (TTRT), which is fixed.
Each node will support the protocol described above. In addition, the master (Surface Unit) is also able to build a database of the node present on the ring. This is used as a monitoring device, and provides useful features for maintenance and fault detection, such as logging events on files. The master also calculates a TTRT and distributes it to all the nodes .

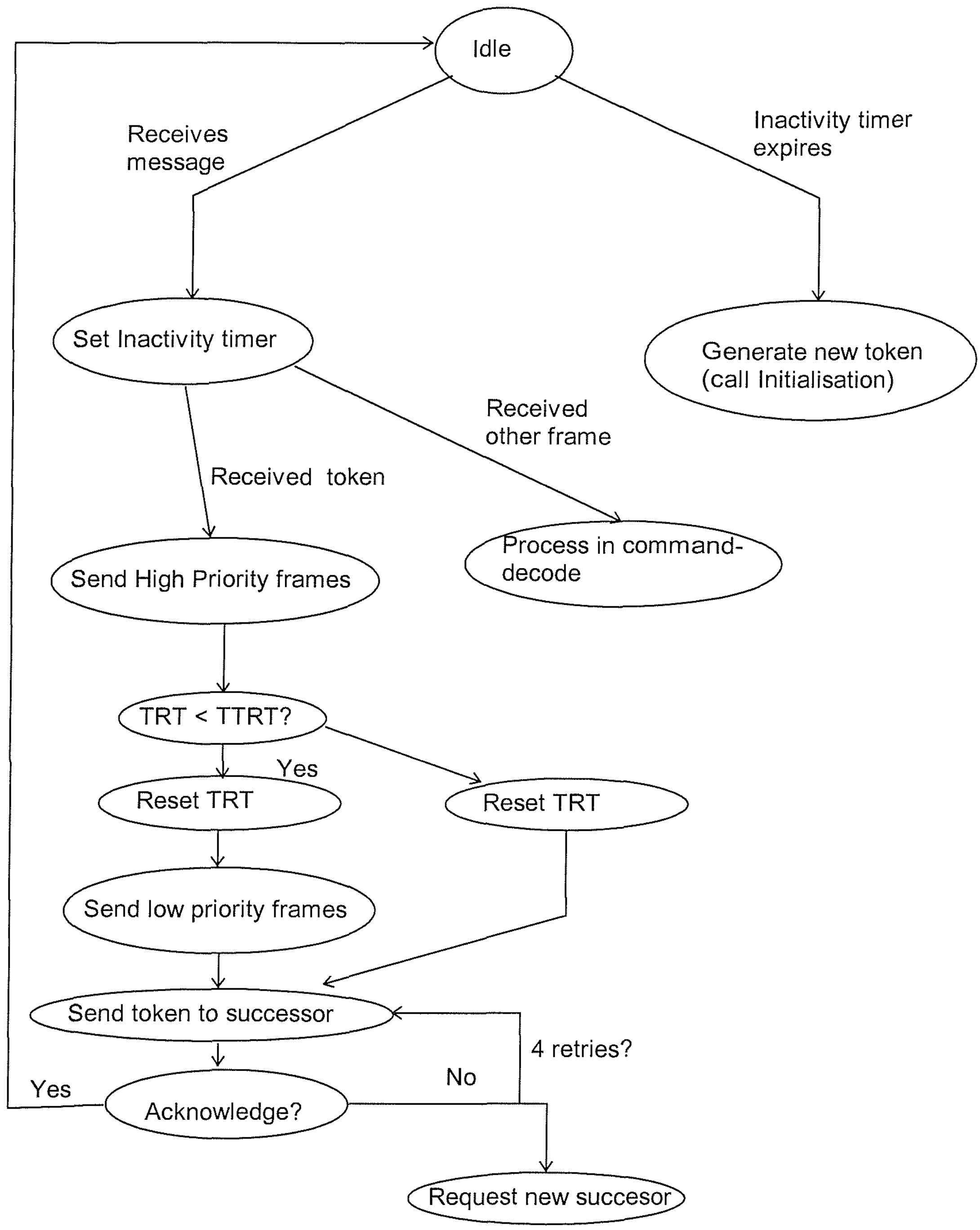


Figure 3.7 Token passing flowchart

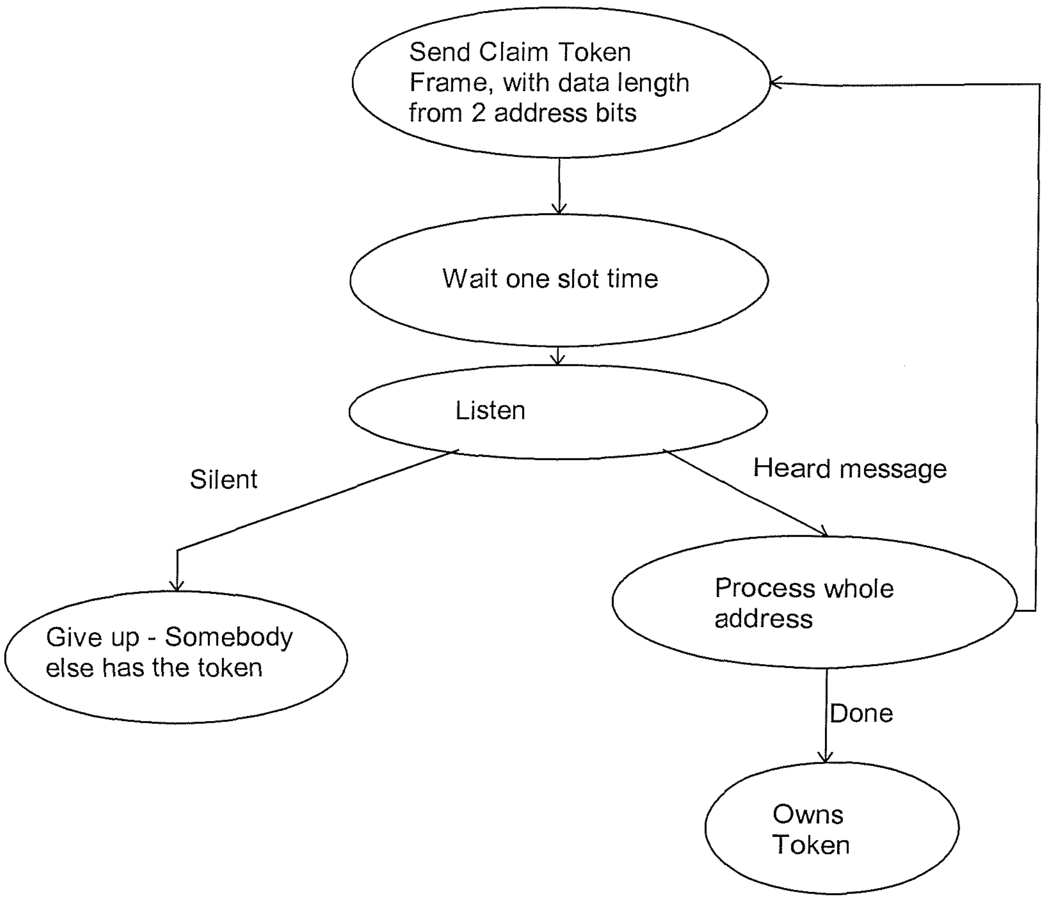


Figure 3.8 Token initialisation procedure

3.3 Preliminary tests and design steps

A step-by-step approach to the building of the prototype vehicle and network has been taken. The hardware was built at the same time as the software evolved. The first step was to have the local function of the first node operational. This provided a test bed to ensure that the programming tools, such as compiler, EPROM programmer and emulator, were operational; and to assess the validity of the hardware.

Then a basic communication was established with the PC used for development. At that stage a demonstration was arranged showing the PC controlling a Seaeye thruster remotely via the thruster node.

The network protocol was then implemented and tested with the prototype hardware. A library was written, allowing all the network communication functions to be standardised for all nodes. [17][19]

The software design for each of the additional node followed the same process: first implementing the local routines in a stand-alone mode. This allowed to check that the hardware was operating as expected, and to establish and test the local software procedures. The network library was then included and the node was tested in networked mode. [20]

3.4 Comparison with commercial networks

The major factor for choosing to implement our own network, as opposed to using an 'off-the-shelf' package, was the economic aspect.

Off-the-shelf solutions would offer have offered better performance: since they are designed commercially, larger manufacturing quantities mean that it is worth designing dedicated transceiver and hardware.

The whole Fieldbus standard is not yet published at the time of the research, so the design was based on the currently published parts, with some additional design features described below. Currently, we can list several points where our design differs from the Fieldbus IEC standard.

- Encoding[4]: our design uses NRZI[5] whereas Fieldbus uses Manchester encoding. This choice was mainly driven by the availability of the encoding hardware.
- Priority level : the draft standard makes provision for 3 levels, we only implement 2. This made the design simpler, and there was no requirement for more priority levels.

As far as the prototype vehicle is concerned, the main feature that could have been useful was to have a high performance transceiver device, however the standard RS485 device used proved sufficient for the speed response needed by such vehicles.

The advantages of having designed a customised network are:

---

[4] Encoding is the way a logical value is transmitted electrically over the transmission line

[5] NRZI: Non Return to Zero Encoding : an encoding method where the signal level does not change for a binary '1', and where a voltage transition represents a logical '0'.

- All the details are known, which made the network very easy to model
- All the features are there because they were needed, rather than because the network manufacturer provides them by default, this saves on memory requirements.
- The network is very basic, this limits the number of possible failures.

The disadvantages were that all the levels of design and implementation had to carried out for the research, this added a considerate amount of work, and was a riskier approach.
However the choice of the network has little importance when the study of time delays within control loops is concerned. Indeed all types of networks will show a variation in the transportation delay when the network configuration is modified.

4. EXPERIMENTAL SETUP

4.1 Overall concept

Much emphasis was given to the practical aspects of this communication and control system. In collaboration with a teaching company associate working with Seaeye Marine, a prototype ROV was built and subsequently tested in the manoeuvring tank at DERA. The hardware used for the prototype ROV was also used for bench experiments, in conjunction with some dedicated test software.
The chassis used to house the prototype communication system was a Seaeye Surveyor, for ease of reading this is referred to as 'the prototype vehicle'.
The prototype electronics replaced the Seaeye communication system and interfaced to the existing instruments. The original chassis and electronics pressure pods were used [33]. A picture of the prototype vehicle is shown in Figure 4.2.

The Surveyor is a survey/inspection vehicle. The original Seaeye specifications are described in Table 4.1:

Vehicle	Total Length: 1450 mm Width: 820 mm. Height: 815 mm. Weight: 175 Kgs. Forward thrust: 80 Kgs. Payload: 45 Kgs. Lateral thrust: 35 Kgs. Depth rating: 300 Metres. Vertical thrust: 35 Kgs
Camera	Colour CCD television camera with wide angle lens, fixed focus and auto-iris.
Camera tilt	± 90° of tilt, providing optimum coverage.
Lighting	2 x 150 Watts Quartz Halogen lamps, variable intensity and mounted on camera tilt.
Navigation	Flux-gate compass with solid state rate sensor for additional azimuth stability. Depth sensor
Auto-pilot	Automatic pilot is provided for heading and depth.

Umbilical	Lifting umbilical cable complete with electrical and mechanical terminations. Used to launch and recover the vehicle Specifications:- Sheathing : Polyurethane. OD : 24.5 mm Weight in air : 618 Kg/km Weight in seawater : 134 Kg/km Minimum bend radius: 240 mm (Dynamic.) Break strength : 3000 Kgf.
Surface Unit	Free standing (19" rack) console housing surface control electronics and keypad. Height: 370 mm. Width: 495 mm. Depth: 495 mm. Weight: 25 Kgs.
Surface Power Supply Unit (PSU)	mounted in a steel cabinet, 2 Power Supply Units, supplied with 440v three phase AC power. Height: 1450 mm. Width: 600 mm. Depth: 500 mm. Weight: 207 Kgs.
Controller	Small self-contained hand control unit containing all vehicle controls. Supplied with a flying lead. Height: 112 mm Nominal. (190 mm max.) Width: 145 mm. Depth: 150 mm. Weight: 2 Kgs. Power: 380 Vac /415 Vac/480 Vac 3-Phase 50/60 Hz. 15 kva.

Table 4.1 Seaeye Surveyor Specifications

These specifications were kept on the prototype vehicle: the original thrusters, Camera tilt unit, Lighting, Navigation, Umbilical and power supplies were interfaced to the prototype electronics.

The following structure was used for interfacing to the various instruments:(Figure 4.1)

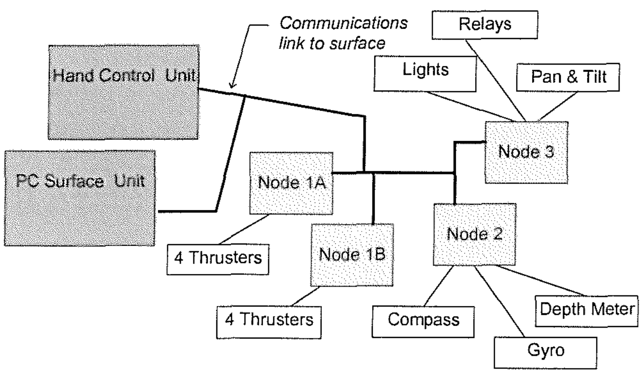


Figure 4.1 Prototype ROV Communication system

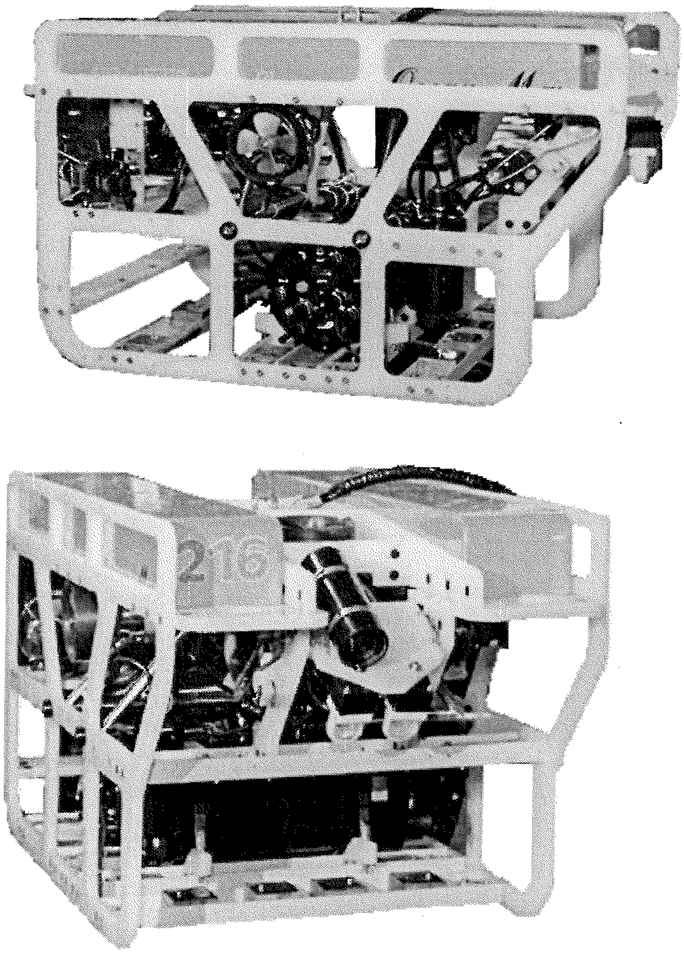


Figure 4.2 Side and front views of the vehicle based upon the prototype Fieldbus system

4.2 Detailed information

4.2.1 PC Surface Unit

The PC is a standard PC fitted with an RS485/Zilog 8530 serial communication card [16]. During the development of the prototype vehicle, the PC was used as the communications master node. Several versions of software were created as the development of the subsea nodes evolved into a full vehicle (Figure 4.3). After ensuring that the communication hardware is present on the PC and initialised correctly, the software enters a loop that can be exited by the user pressing ‘Q’ on the keyboard. Within the loop, the master starts by not owning the token, and attempts to find it by listening to incoming messages. Within a certain time limit, the master can assume that the token has been lost and starts a recovery routine. Once the station owns the token, it can now send data to any node. When the master has finished sending data, it can pass the token to its successor on the logical ring. The data that the master sends is taken from the user input, for example pressing the up-arrow would cause the master to send a message to the thruster nodes, requesting an upward thrust.

The HCU (Hand Control Unit) was built at the end of the project, to allow the PC surface unit to be replaced by a cheaper alternative. It also has the advantages of being smaller and portable.

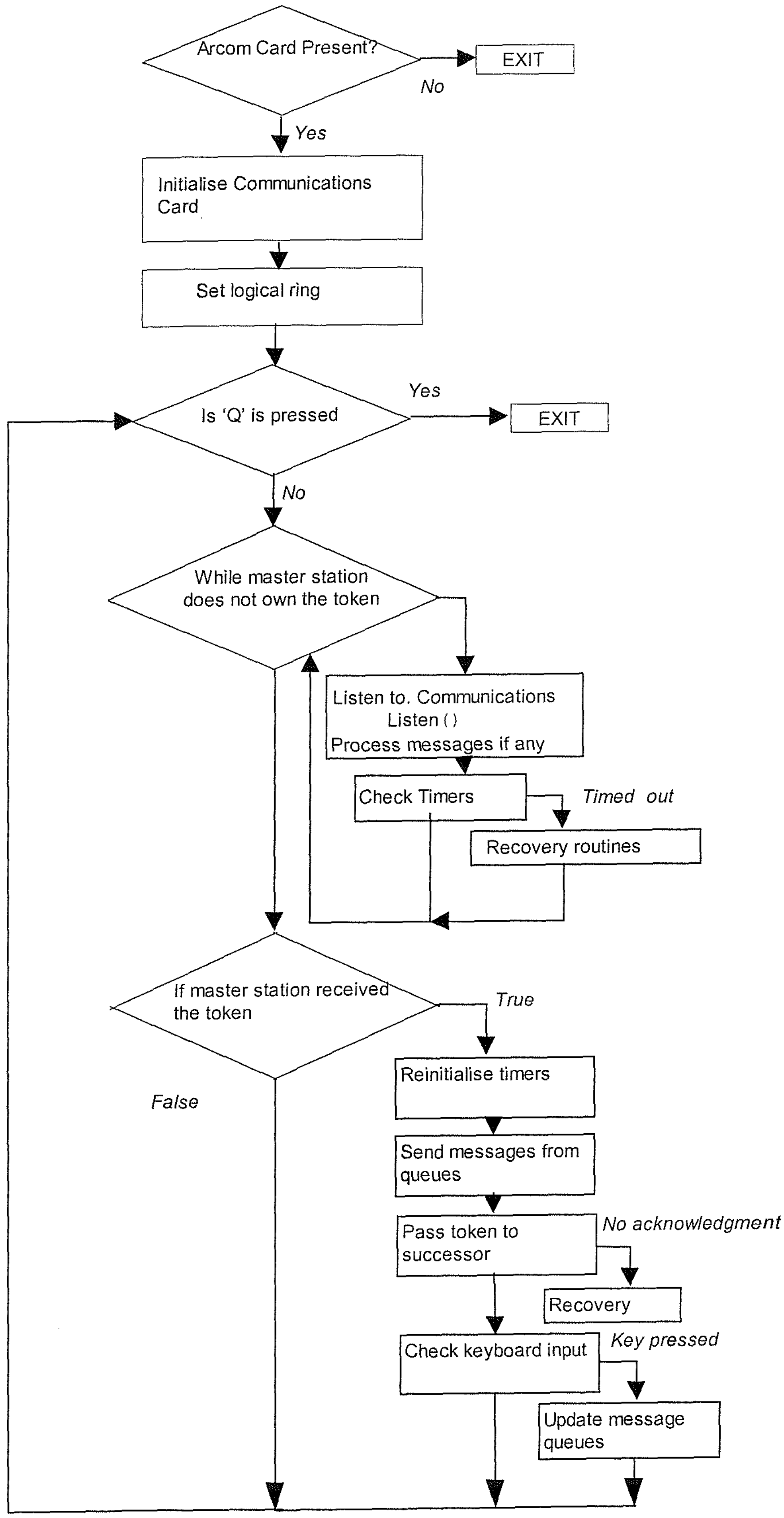
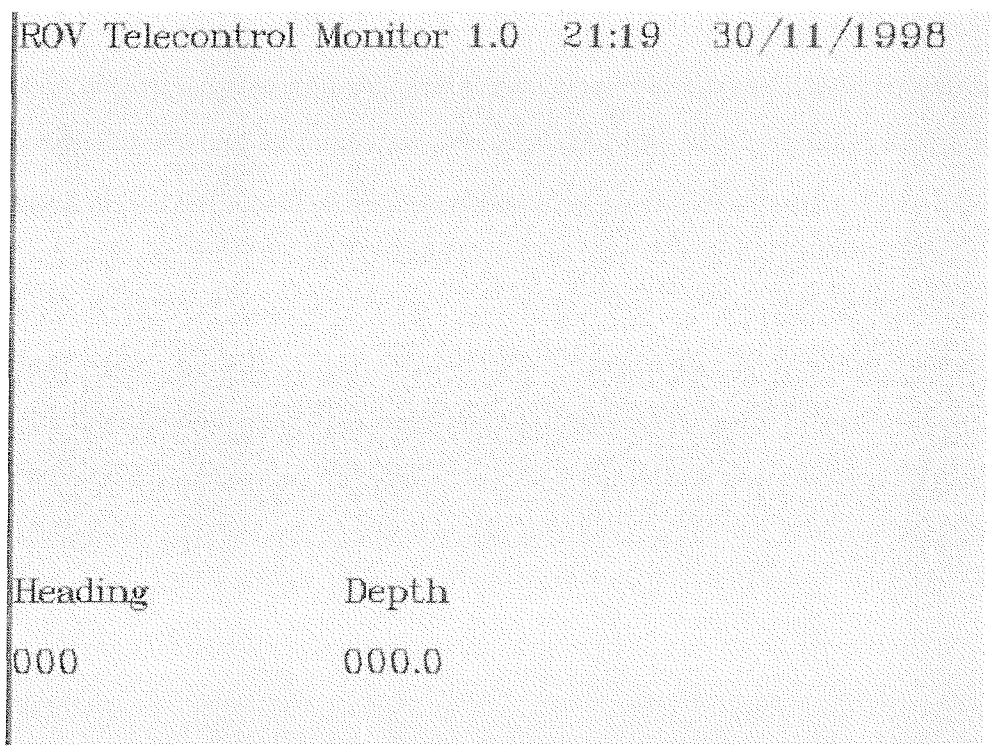


Figure 4.3 PC 'Development software' flowchart

A software library has been created [17], allowing the communication routines to be imported easily. This has proved very useful as the various test software routines were developed for experimental purposes (e.g. noise tests, control system).

Once the HCU was completed, the PC was not needed any longer to run the ROV. However the graphical interface could still be used by overlaying the PC output with the live video image coming from the vehicle's camera. A basic monitoring software was created, allowing the PC to have a listening only role [18]. The main feature was to display on a monitor information such as depth and heading (Figure 4.4). After ensuring that the communication hardware is present on the PC and initialised correctly, the software enters a loop that can be exited by the user pressing a key on the keyboard. Within the loop, the PC listens to all messages on the communication line, and reacts to internal events such as time change and when a new value is detected the display is updated. A major advantage is that, as the PC is not part of the network as such (it was not assigned an address), the software can be started and stopped independently of the vehicle. A screenshot of the monitor software is shown below, this display was overlaid on top of the live video picture coming from the ROV's camera. The output from the video picture is not shown on the figure.



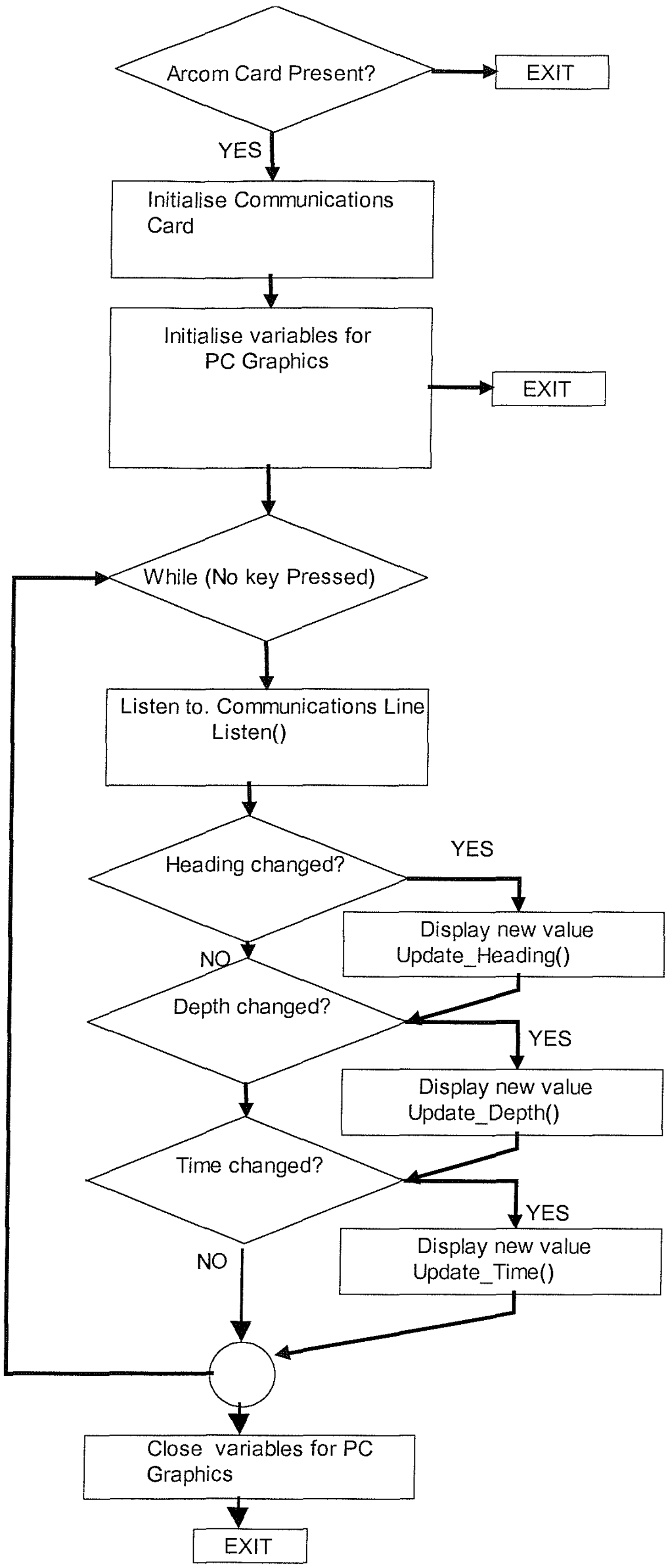


Figure 4.4 PC Monitoring software flowchart

4.2.2 Thruster Card (Node 1A and Node 1B)

The thruster node's functionality is to provide interfacing to the Seaeye SM4 thrusters. This includes power amplification. Each node can drive four thrusters.
The Seaeye Marine SM4 thruster motor is a brushless DC unit containing integral electronics. It requires a 250 V. DC, 5A power supply, and can provide 20 kg of dynamic thrust. It is designed to operate at depths down to 1000 m.
The thruster is controlled by three lines : two direction lines and a 50 Hz Pulse Width Modulated (PWM) speed signal. The direction signals have an amplitude of 24 V (peak-to-peak), the PWM speed signal has an amplitude of 12V (peak-to-peak) (Figure 4.5).

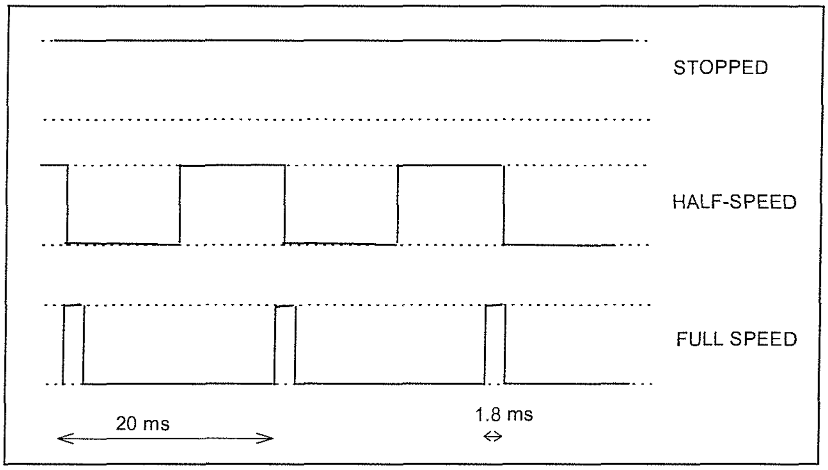


Figure 4.5 Pulse Width Modulation Speed Signal

The thrusters have to receive an 'Init. pulse' when started, this is implemented in software. The structure of the software is described in flowcharts in Figure 4.6. After an initialisation sequence, necessary for the microprocessor card hardware, a loop is entered, where the timer values are checked. If the node has not received any command for a long time (set to 10 sec), something has gone wrong, and the thrusters are stopped. The thruster command value is also monitored to convert the high level command received in the message (e.g. upwards, full speed) to local commands (e.g. thruster number 1, full speed forward). The communication routine is called by interrupt, the timer interrupt is used to create the PWM signal, based on the low level commands, and also to keep track of timers.

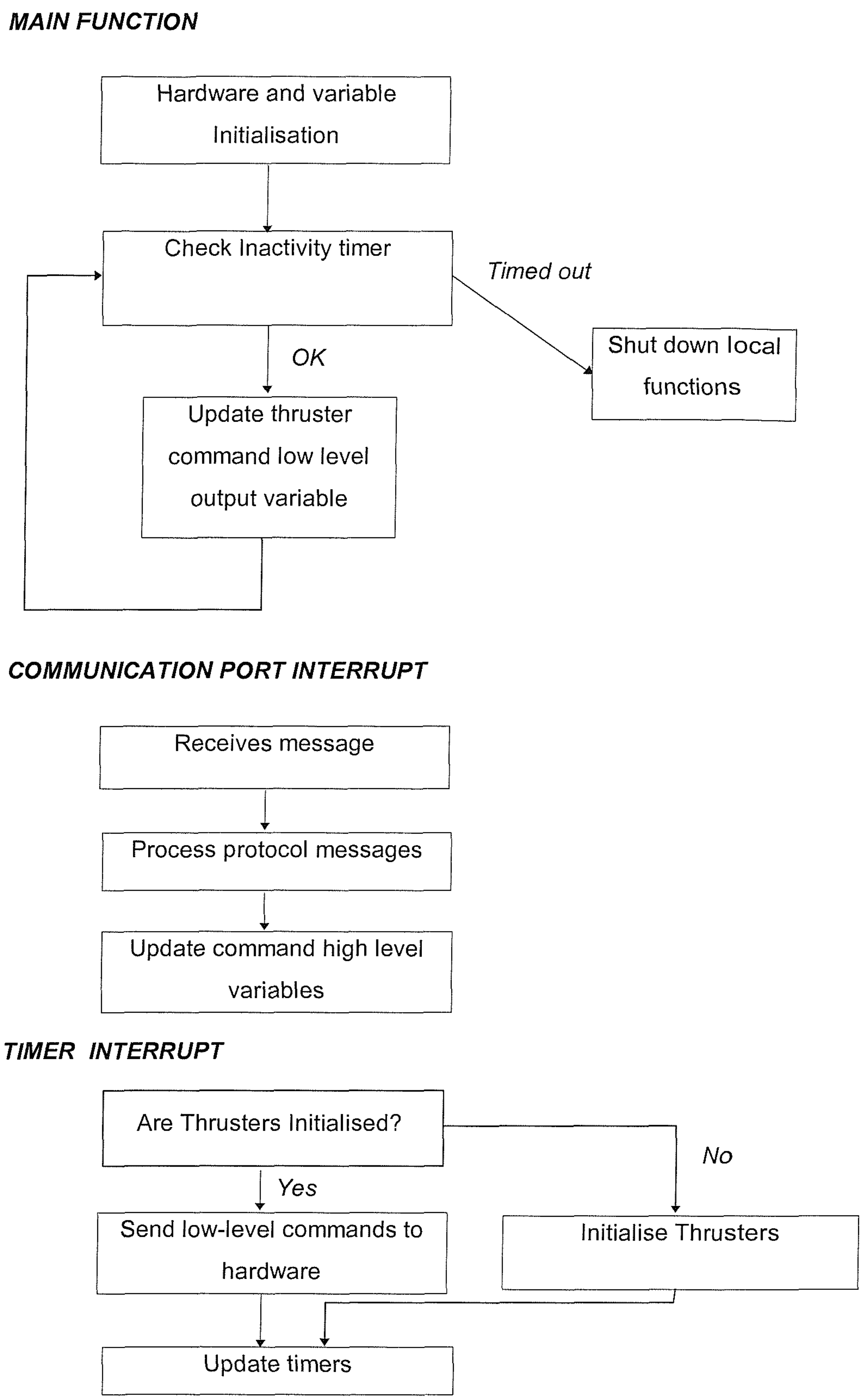


Figure 4.6 Thruster node software flowchart

The vehicle was used for the DERA tank tests. Two thruster cards were required, each controlling four thrusters. The choice of thruster assignment was such that in the event of one thruster node being non-functional, the vehicle could still move in all its axes of freedom.
The software used to implement those functions comprises of a set of standard slave communication routines [19], and of local applications routines [20]; those were created as a common module to be shared between all slave nodes.

4.2.3 Navigation Card (Node 2)

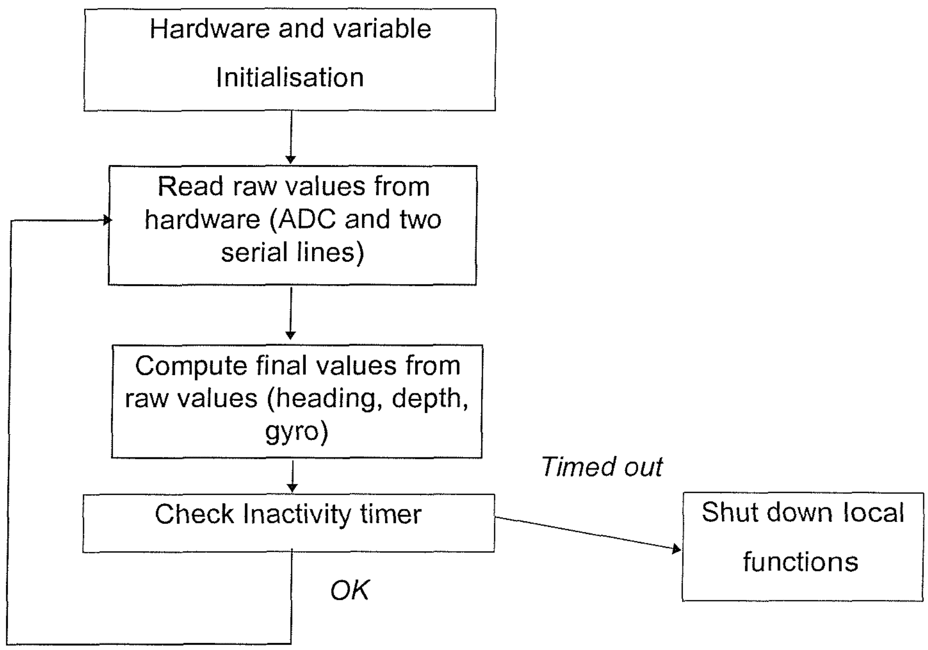
The navigation node is used to interface to various navigation sensors. The card is fitted with two serial ports and an analogue port, allowing it to interface to most instruments. In the case of the prototype ROV, those were: a Cetrek Compass, a Gyro and a Depth meter.
The Cetrek compass is a flux gate compass, with a serial data output mode. The output follows the NMEA (National Marine Electronics Association) standard. The format is defined as 4800 bauds, 8 data bits, no parity, one or more stop bits.
The gyro is Gyrostar ENV-05A, manufactured by Murata. It is connected to the card's analogue input, linking it to the Analogue-to-Digital Converter (ADC).
The Depth-meter is based on a pressure transducer, the output of which is a frequency modulated square wave, ranging from 1 kHz to 6 kHz, corresponding respectively to depths of 0 to 500 m. This is converted to a voltage, via a frequency to voltage converter, and then digitised via an serial ADC, allowing the value to be read on the card's UART[6].
The software constantly reads the values of the navigation devices; because the conversion time of the ADC is very fast compared to the network (25 μsec for a conversion cycle), when the navigation nodes ends the values on the network, the most recently read values are sent.
The software used to implement those functions comprises of a set of standard slave communication routines [19], and of local applications routines [20]. A flowchart describes how those functions are used (Figure 4.7). After an initialisation sequence, necessary for the microprocessor card hardware, a loop is entered. The first action is to read the available data from the connected sensors, and to convert those value into

---

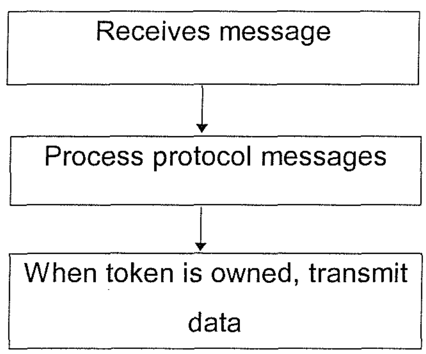
[6] UART: Universal Asynchronous Receiver Transmitter; the electronic device used to convert between serial data and parallel data used by a microprocessor.

standard units. The timer values are checked (e.g. if the node has not received any command for a long time (10 sec), something has gone wrong). The communication routine is called by interrupt, and processes incoming and outgoing messages. The timer interrupt is used to update the timer values.

MAIN FUNCTION



COMMUNICATION PORT INTERRUPT



TIMER INTERRUPT

Figure 4.7 Navigation node software flowchart

4.2.4 Video Card (Node 3)

This card has various input and output facilities:

- three relays, one for Sonar switching, one for switching the video signal going to the umbilical between two cameras, and one for operating a 'stills' camera.
- tilt platform closed loop control for the camera; provision has also been made for a pan facility. This also includes a trip-detection system, which allows for the pan and tilt facilities to be stopped in case a mechanical fault occurs and the motor is drawing more current than expected.
- light level control signal

The software used to implement those functions comprises a set of standard slave communication routines [19], and local applications routines [20].

A flowchart describes how those functions are used (Figure 4.8). The structure is very similar to the navigation node, Digital to Analogue Converters (DAC) and Analogue to Digital Converters (ADC) are used to interface with the local instruments. A simple PID (Proportional, Integral and Derivative) controller is implemented to control the Pan and Tilt position.

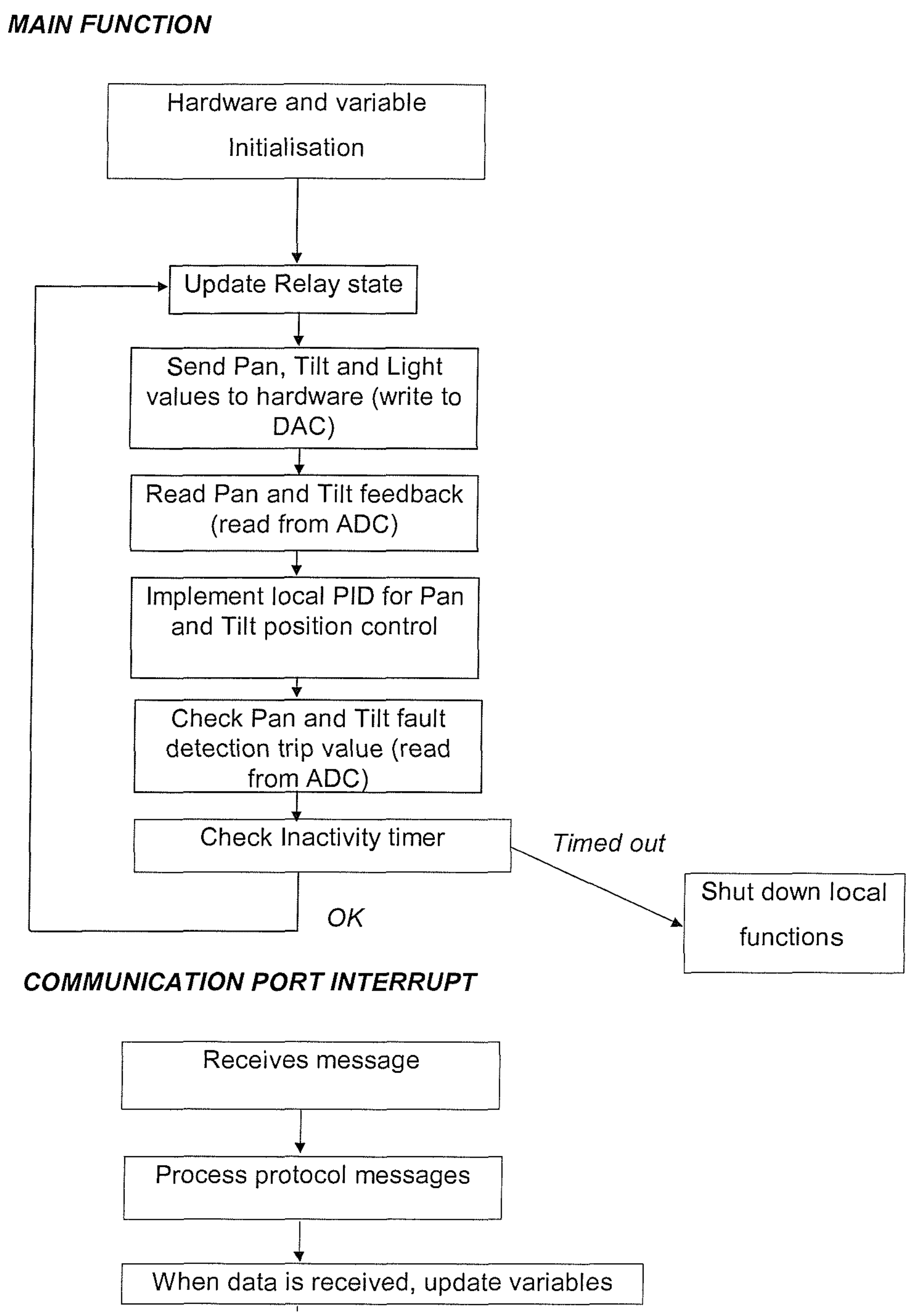


Figure 4.8 Video node software flowchart

4.2.5 Hand Control Unit (HCU)

The HCU is a hand-held control box used by the pilot to control the ROV. A list of commands available to the pilot is given in Table 4.2, with a reference to how the function is implemented in the hardware.

The HCU was used as a communications master node in the final ROV prototype. The software used is described in Figure 4.10 and Figure 4.11 [21] and, because the microprocessor used was different to the other nodes, the low-level communications routines had to be rewritten to accommodate the hardware changes.

Function	**Hardware used**
Display	2 line LCD screen
Audible alarm	Buzzer
XY + Twist ROV movement *moving ROV within horizontal plan*	Joystick (Analogue Inputs)
Z ROV Movement *moving ROV up or down*	Potentiometer (Analogue Input)
Auto-Depth ON/OFF switch	Digital Inputs
Auto_Heading On/OFF switch	
Thruster Enable ON/OFF switch	
Sonar ON/OFF switch	
Camera 1/2 switch	
Stills camera ON/OFF switch	
Full up switch *moving ROV up as fast as possible*	
Full down switch *moving ROV down as fast as possible*	
Lights potentiometer	Analogue Inputs
Tilt position potentiometer	Analogue Inputs
PID tuning potentiometers	Analogue Inputs
Backup memory for joystick calibration	e2PROM

Table 4.2 HCU functions

The HCU's node architecture is described in Figure 4.9. The software used is described in Figure 4.10. After having initialised the hardware components as required, an attempt is made to retrieve previous calibration results from e2PROM. If this is unsuccessful, the calibration routine has to be called, allowing the user to calibrate the potentiometers and joysticks. The communication circuitry is then started, and user inputs are read for the first time. The software then enters a loop, where input from the user is read, and accordingly either a menu option is offered, or a normal run mode ('Go') is entered (Figure 4.11). In run mode, the values read are transmitted as command messages to the relevant nodes. Incoming messages are also processed, and heading and depth values are displayed.

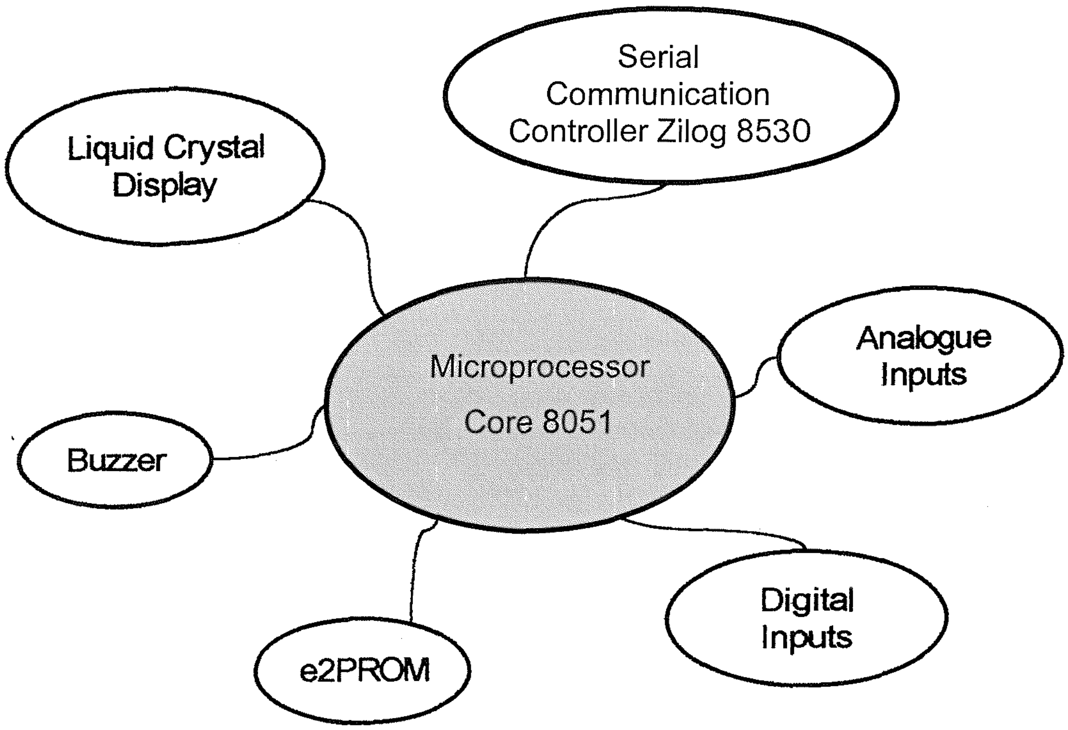


Figure 4.9 HCU architecture

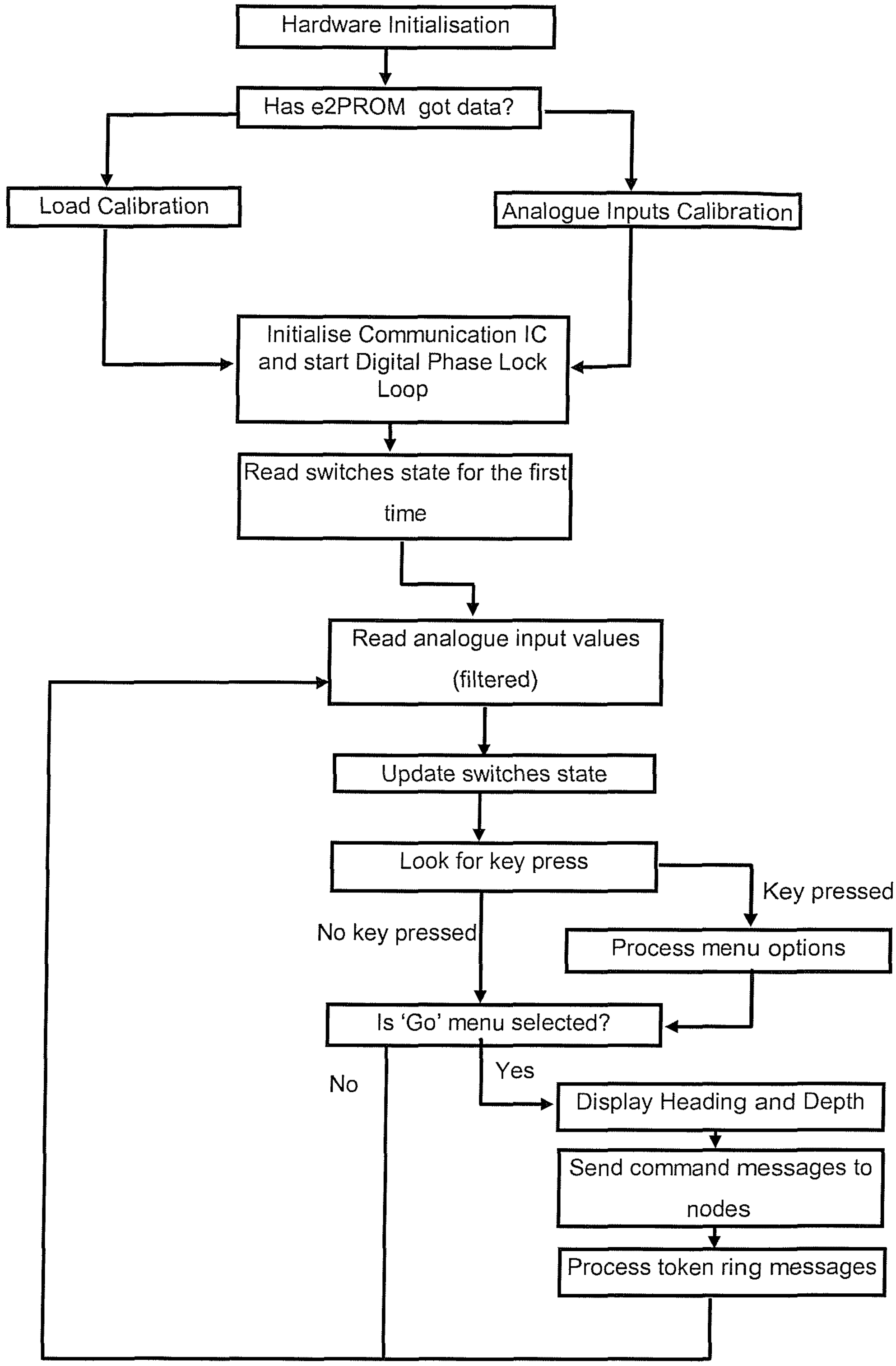


Figure 4.10 HCU Main software structure

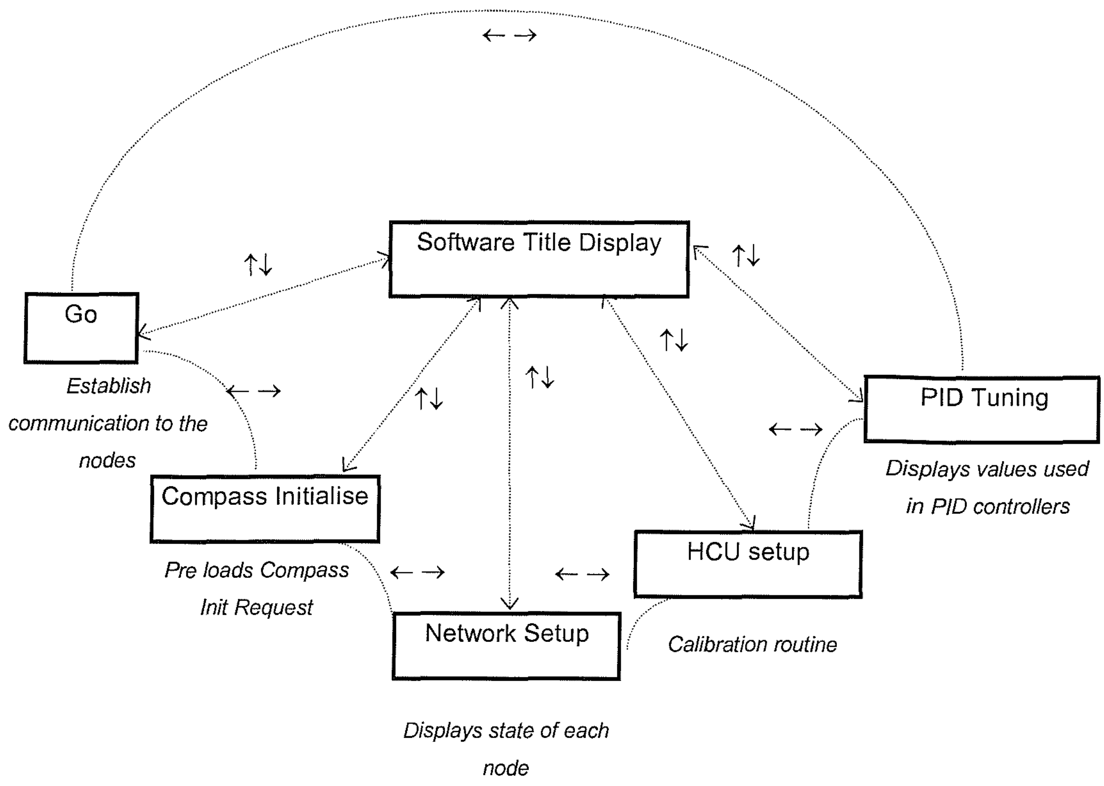


Figure 4.11 HCU menu structure

4.3 Noise sensitivity

The effect of noise on the Bit Error Rate (BER) in our system has been evaluated experimentally, by introducing noise artificially on the line.

The system was set up as : a PC sending repeatedly a set message to Node 1, node 1 was listening for the set message. Counters were set at both the sending and receiving end (Figure 4.12). No Frame Check Sequence (FCS) was used, so that we could count : the number of valid frames received, the number of valid frames with corrupted data received and the number of corrupted frames that were lost. Valid frames with corrupted data would be detected when using a FCS, corrupted frames would be ignored.

Noise was produced by a white noise generator [22] and introduced on the line via a torroidal transformer. Different measurements were made for different baud rates and noise levels.

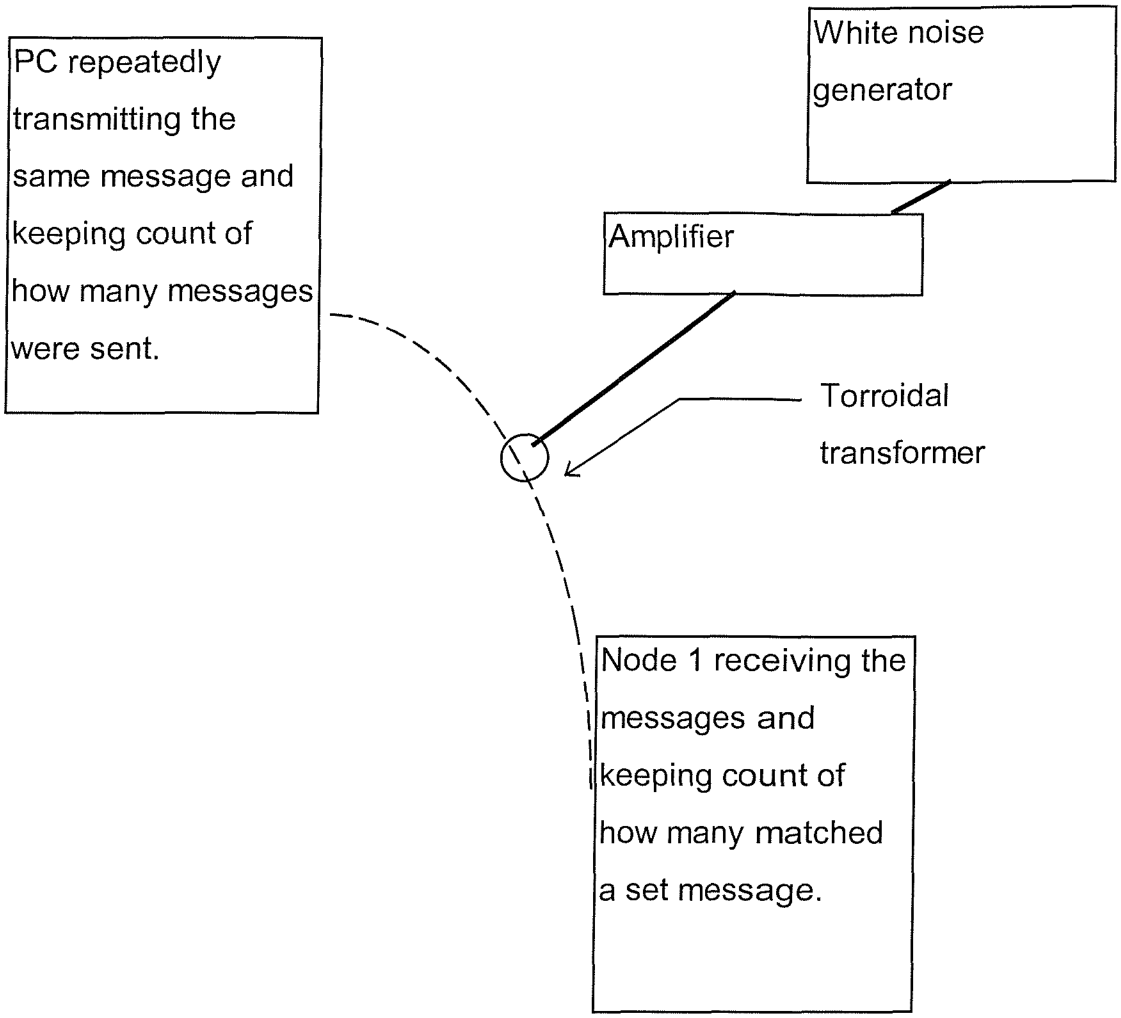


Figure 4.12 Noise tests setup

The results (Figure 4.13, Figure 4.14, Figure 4.15) show that small levels of noise have little effect on the frame error rate. At a higher level, noise causes the number of faulty frames to increase sharply. The number of valid frames with corrupted data increases more slowly, and decreases when the level of noise prevents any valid frame to go through at all.

The responses have the same shapes for the different baud rates, the main difference being that the rise starts at lower noise level for higher baud rates.

Those results have been used later to evaluate the stability of the network under high noise level.

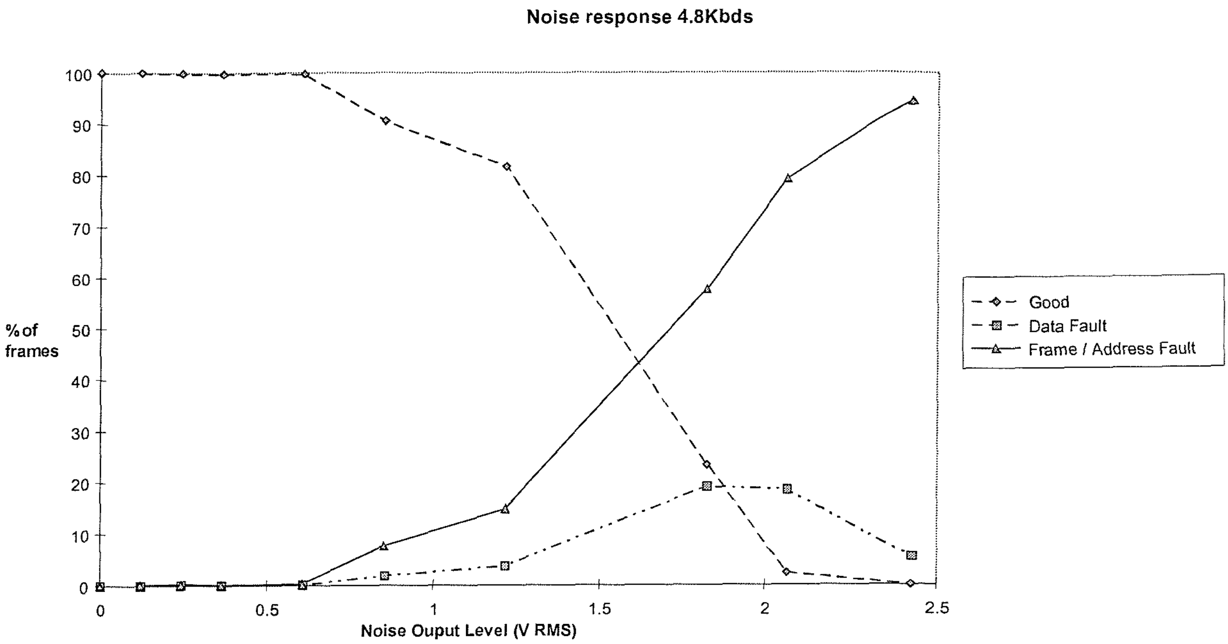


Figure 4.13 Noise tests results at 4.8 Kbds

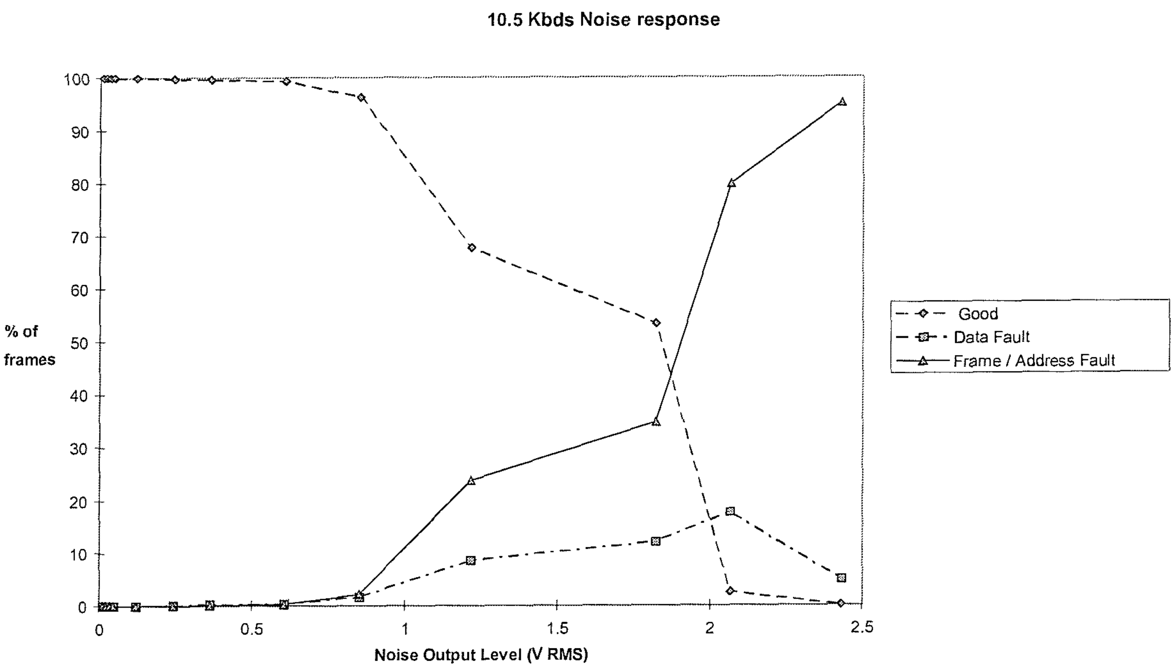


Figure 4.14 Noise tests results at 10.5 Kbds

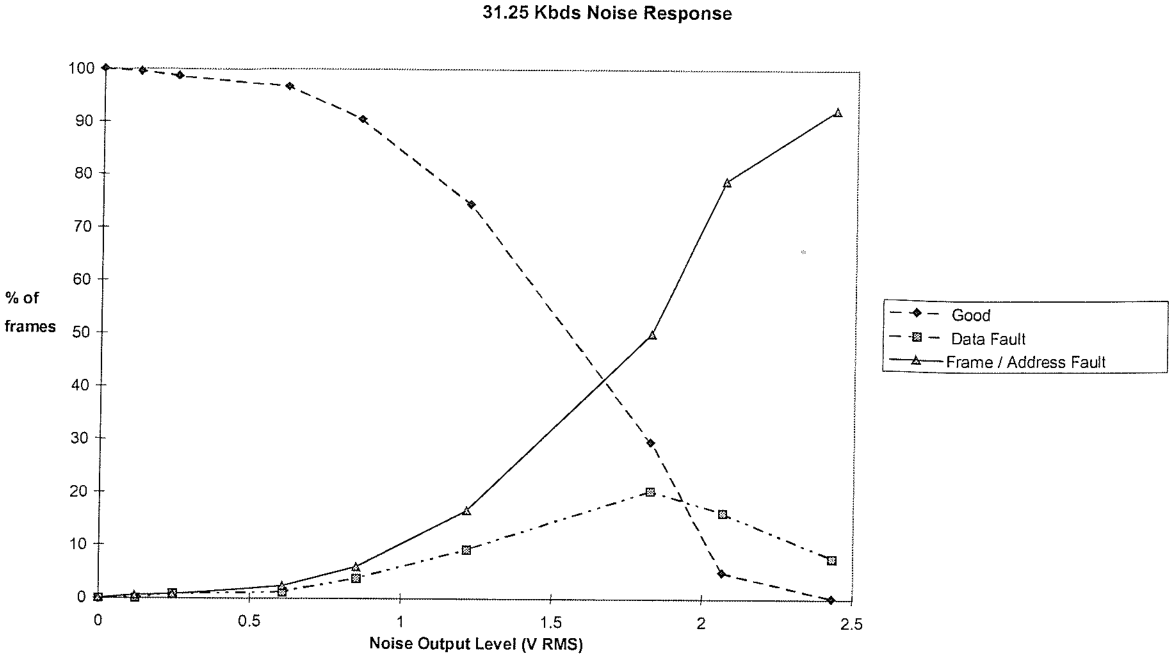


Figure 4.15 Noise tests results at 31.25 Kbds

5. SIMULATION OF A NETWORK

The performance of a network can be measured by one major criteria: the delay between when the message is available at the sender station and when this message is received correctly by receiver station. This delay will vary depending on many factors: transmission speed, number of stations, length of messages, error rate. A network simulation is a good way to predict this delay, which can then be analysed for different configurations. It is a good alternative to building and testing the network at the outset. Not only this would be expensive, but should the results be unsatisfactory, the cost and complexity of changing the network would be a drawback. Simulation allows for much more flexibility, and makes trying different configurations much more practical than having to implement them for real.

5.1 LAN simulation

Simulation of Local Area Networks currently exist; they are mainly used during the planning phase of a network design, allowing the designer to evaluate its performance before committing to hardware. Different type of simulations exist:

1. Analytic models : Analytic simulations are based on a mathematical representation of the system. Assumptions about the system have to made in order to find such a mathematical representation and this makes the simulation difficult to develop. In addition, since the simulation is unlike the real-life situation only gross answers can be obtained.
2. Modelled simulations : In this case the network is modelled up to the level of detail required. Since less assumptions have to be made than in 1, the result is more accurate [23]. However, programming can be complex and costly, and the resulting simulation is slower to run.
3. Hybrid : An hybrid simulation is a combination of the two methods above; it gives a compromise between accuracy, complexity and run time.

5.2 Statistical aspects

This section deals with the probability aspects of a computer simulation. Firstly, the simulation must be able to generate truly random numbers in order to model arrival rate for messages that are triggered randomly. Secondly, the batch of results gathered is limited in number, and must be interpreted in statistical terms. There are many ways in which to generate a random number X, from its probability distribution F(x). Two techniques commonly used are described in Appendix D; the inverse method and the rejection method. The inverse method was used in the simulation software.

The simulation gives out estimated values which is only an average of a number of tests. These sample statistics will vary from one experiment to another. Hence the values obtained will fluctuate about a mean value.

Supposing that X is a random variable, its mean μ is defined as : $\mu = \int_{-\infty}^{+\infty} x f(x) dx$,

with $f(x)$ defining the probability density function of X.

If we draw random and independent samples $x_1, x_2, x_3, \ldots, x_N$ from $f(x)$ our estimate of x would take the form of the mean of N samples : $\hat{\mu} = \frac{1}{N}\sum_{n=1}^{N} x_n$

μ is the true mean value of X and $\hat{\mu}$ is the unbiased estimator of μ. $\hat{\mu}$ is close to μ, but $\hat{\mu} \neq \mu$. The spread of the difference between the two values is given by the standard deviation : $\sigma(x) = \left[E(X^2) - \mu^2\right]^{1/2}$

The confidence we place in the estimate of the mean is given by the variance of $\hat{\mu}$: $\sigma(\hat{\mu}) = \frac{\sigma(x)}{\sqrt{N}}$

This shows that the spread of the results falls as the number of samples increase.

The spread in $\hat{\mu}$ is defined as the sample variance : $S^2 = \frac{1}{N-1}\sum_{n=1}^{N}(x_n - \hat{\mu})^2$

Using the central limit theorem, which states that the sum of a large number of random variables tends to be normally distributed, this gives [23]:

$$f(\hat{\mu}) = \sqrt{\frac{N}{2\pi}} \frac{1}{\sigma(x)} exp\left[-\frac{N(\hat{\mu} - \mu)^2}{2\sigma^2(x)}\right]$$

Since the number of samples N is finite, we can estimate some confidence interval around μ, so that we can predict that $\hat{\mu}$ falls within the interval between $\mu-\varepsilon$ and $\mu+\varepsilon$.

$$P[\mu-\varepsilon<\hat{\mu}<\mu+\varepsilon]=\int_{\mu-\varepsilon}^{\mu+\varepsilon} f(\hat{\mu})d\hat{\mu}$$

$$by\ letting\ \lambda=\frac{(\hat{\mu}-\mu)}{\sqrt{2/N}\sigma(x)}\ we\ get$$

$$P[\mu-\varepsilon<\hat{\mu}<\mu+\varepsilon]=\frac{2}{\sqrt{\pi}}\int_{0}^{(\sqrt{N/2})(\varepsilon/\sigma)} e^{-\lambda^2}d\lambda$$

$$=erf\left((\sqrt{N/2})\frac{\varepsilon}{\sigma(x)}\right)$$

$$or\ P\left[\mu-z_{\alpha/2}\frac{\sigma}{\sqrt{N}}<\hat{\mu}<\mu+z_{\alpha/2}\frac{\sigma}{\sqrt{N}}\right]=1-\alpha$$

$Z_{\alpha/2}$ is the upper $\alpha/2$ percent of the standard deviation. The confidence interval is $\hat{\mu}\pm\varepsilon$, the confidence level is $erf\left((\sqrt{N/2})\frac{\varepsilon}{\sigma(x)}\right)$

Generally ε is chosen as $\frac{\sigma(x)}{\sqrt{N}}$, this implies that the probability of the sample mean $\hat{\mu}$ lying within the interval $\hat{\mu}\pm\sigma(x)/\sqrt{N}$ is 68.26%. Other values are shown below, with M the number of standard deviation:

$$P\left[\mu-M\frac{\sigma(x)}{\sqrt{N}}<\hat{\mu}<\mu+M\frac{\sigma(x)}{\sqrt{N}}\right]=\begin{cases}0.6826 & M=1\\ 0.954 & M=2\\ 0.997 & M=3\end{cases}$$

Usually σ is not known, we can obtain it from a t-distribution table, knowing values for S and N. This gives:

where $t_{\alpha/2}$ is the upper 100 x ($\alpha/2$) percentage point of the t-distribution.

$$P\left[\mu-\frac{S\,t_{\alpha/2;N-1}}{\sqrt{N}}<\hat{\mu}<\mu+\frac{S\,t_{\alpha/2;N-1}}{\sqrt{N}}\right]=1-\alpha$$

In practice, this means that when analysing a set of simulation results, we must first decide the level of confidence we require, for example 95%.

Then we need to calculate the mean, in order to get the sample variance S.

This allows us to find the confidence interval by applying : $\varepsilon=\frac{St_{\alpha/2;N-1}}{\sqrt{N}}$

This theory is used in the simulation program to represent the set of results obtained.

5.3 Choice of programming language object-oriented approach

5.3.1 Existing languages for network simulation

Some dedicated languages are used for network simulation, such as SIMSCRIPT [24] which has been used for the simulation of circuit switched networks and for token passing bus.

General purpose simulation languages are also used, especially process-oriented languages such as SIMULA [25] and SIMAN (SIMulation ANalysis) [26].

The main feature of those programs is to be able to obtain the average message delay. Additional features such as : calculating bus throughput, utilisation, being able to simulate faulty or normal operation, dealing with priorities, generating random messages at each queue, and selecting randomly the frame length can also be implemented. Some programs also offer a graphic interface to present results to the user.

5.3.2 Example of a C-code program

A basic simulation program for a token-passing ring and bus developed by Sadiku and Ilyas [23] has been studied, and the source code was supplied by the authors in [23]. In order to keep the program simple, a large number of assumptions have been made by the authors :

- The arrival rate at all stations follows a Poisson process
- All stations generate the same amount of traffic (same rate and packet lengths)
- The transmission medium is error free
- Physical spacing between stations is the same
- Source and destination is on average 1/2 ring size apart
- Propagation delay of 5μs/km (from a signal propagation speed in copper of 2.10^8 m/s)

This led to a program with the following structure (Figure 5.1), which was derived from the source code made available by the authors.

Figure 5.1 shows that the software has an event-based structure. The event can be one of three types:

- *Arrival of packet :* this is when the message is 'created' on the node, the rate of creation follows a Poisson process. The time of creation is referred to as start-time of the packet.
- *Token arrival :* this is when a node receives a token, allowing it to transmit messages.
- *Departure of packet :* this is when the node transmits a packet once it received the token. The delay is calculated at this point relative to the start-time. Once the packet has departed, the token can be passed to the following station.

The results are computed following the rules from section 5.2, and information such as average delay within 95% interval confidence can be displayed in textual form.

The software was developed to simulate Local Area Networks, and although the structure is of interest, major modifications have to be made in order to simulate the network used on our prototype vehicle.

In order to reduce the number of assumptions made, the structure of the program has been altered and some application-oriented information as been added: the above program is designed for a loop of computers generating a random amount of data. In our case, the application's structure, i.e. the vehicle, is very different, and we need to take this into account during the simulation. Those alterations can prove difficult to implement within the existing software structure. An alternative approach was to use an object-oriented language.

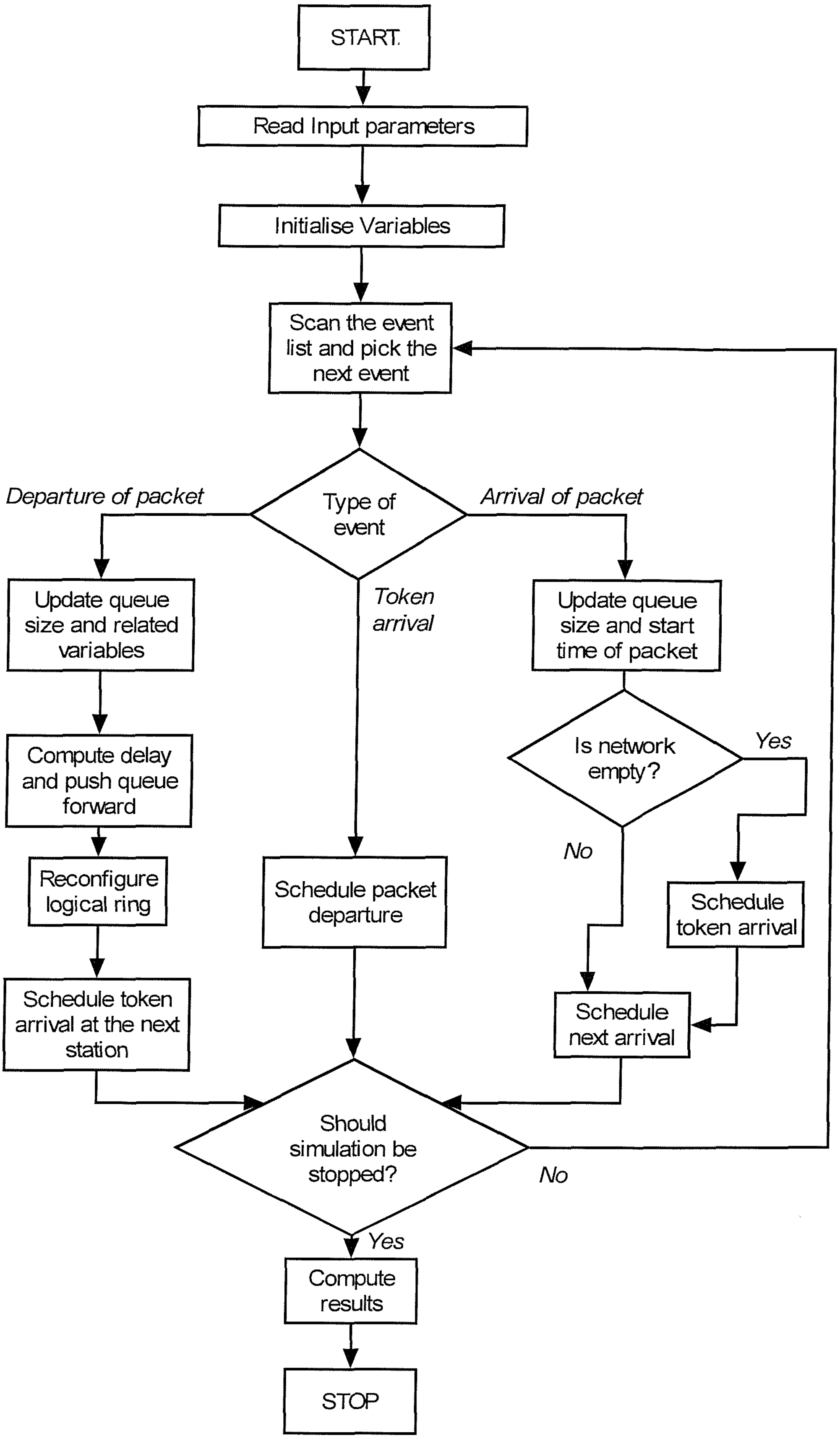


Figure 5.1 Flowchart of Sadiku and Ilyas' simulation software

5.3.3 Object-oriented approach

An object-oriented language allows for great reusability and modularity.
Some of its advantages over a standard programming technique are :

- information hiding, where the name of variables, constants, functions and types can be made local to a module.
- data abstraction, allowing basic facilities for defining a set of operations for an object type, and restricting the access to objects of the type to that set of operations.
- inheritance, allowing to create subclasses (or 'derived' classes) from a superclass (or 'base' class)
- polymorphism, allowing one routine, for example, to be applied to objects of many different types.
- dynamic binding, virtual functions can be used to define a set of operations for the most general version of a base[7] class. When necessary the interpretation of these operations can be refined for particular derived classes.

All those advantages mean that a basic library can be reused very easily.
Because an object-oriented language is based on data rather than functionality, it is particularly well suited for creating simulations. C++ being a common object-oriented language has therefore been selected as the base for the simulation work.

5.4 Simulation of Fieldbus network

The aim of this simulation is to model a Fieldbus-type network, as proposed for the ROV. This means having the basic network simulation structure, but also being able to highlight the issues that are particular to a ROV network, i.e. the nodes can behave very differently depending on what function they have. For example, a navigation node with a compass will have very different communication requirement to a node fitted with a manipulator.
When the program starts, the network is defined as empty by default, stations have to be added. This is not a major problem since the networks we are

[7] It is possible to define subsets of a class (the base class), called derived classes, which can reuse functions and variables defined in the base class.

simulating are of small size. In case this would prove to be a problem in the future, a storage and retrieval facility could be added. Each node can be one of two types : master or slave; a slave can be actuator, sensor or tool (Figure 5.2). The choice of the type of node implies its rate and packet length, however those values can be modified. The basic user interface allows the user to firstly create the network, and secondly to run the simulation. It avoids having to recompile some code for each different configuration.

A class hierarchy diagram shows the organisation of objects in the program (Figure 5.2).

The class *Simu* contains instances[8] of the classes *Medium*, *Net. Simu* deals with all the statistical calculations and results management. It also starts the simulation process. *Medium* contains the particulars of the network such as topology, station latency, propagation delay and medium length. *Net* represents the network, and contains instances of the nodes that are in the network simulation. *Net* also manages queues and schedules start and arriving times. *Node* is the base class for each node, and is derived into *Master, Slave, Actuator, Sensor,* and *Tool* classes to take into account the various arrival rates and packet lengths of each node.

Obviously this model has its limits, and the following points are important:

- error rate: the error rate is taken into account in the simulation, however the model is only valid if errors occur in different frames. The model cannot cope with successive and repeated errors and would give erroneous results.
- latency : this is the time the receiving device takes to decode the incoming message, and to act upon it. The latency values used in the simulation have been taken from measurements made on the node. This was measured by running the networking software, and outputting a signal on one of the output lines, when network messages were received. A time measurement was made with an oscilloscope, by looking at both the communication line and the output line. The time measurement was taken as shown in Figure 5.3. The latency value was 8.5 msec.

[8] An instance of a class is a particular specimen of such a class.

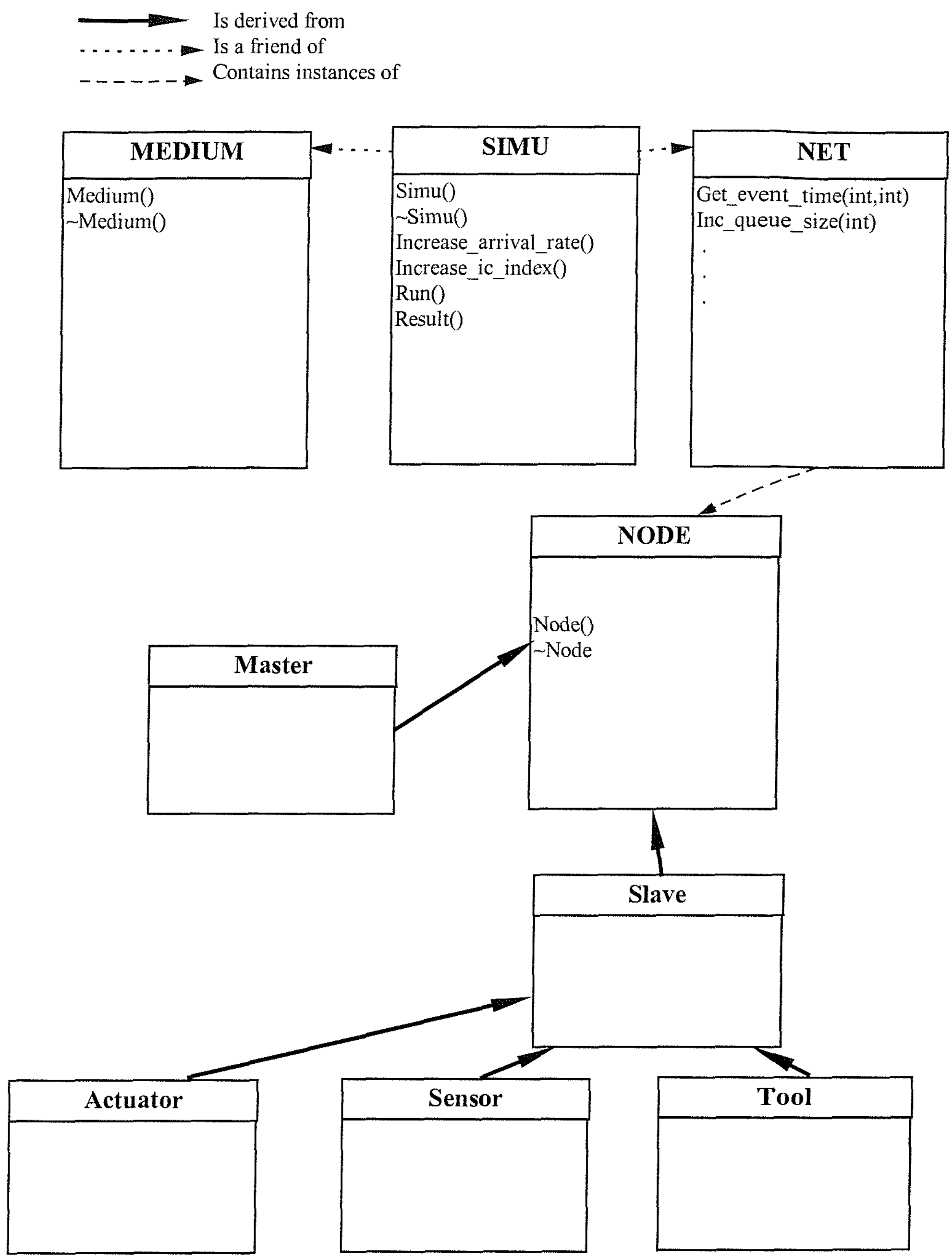


Figure 5.2 Class hierarchy diagram

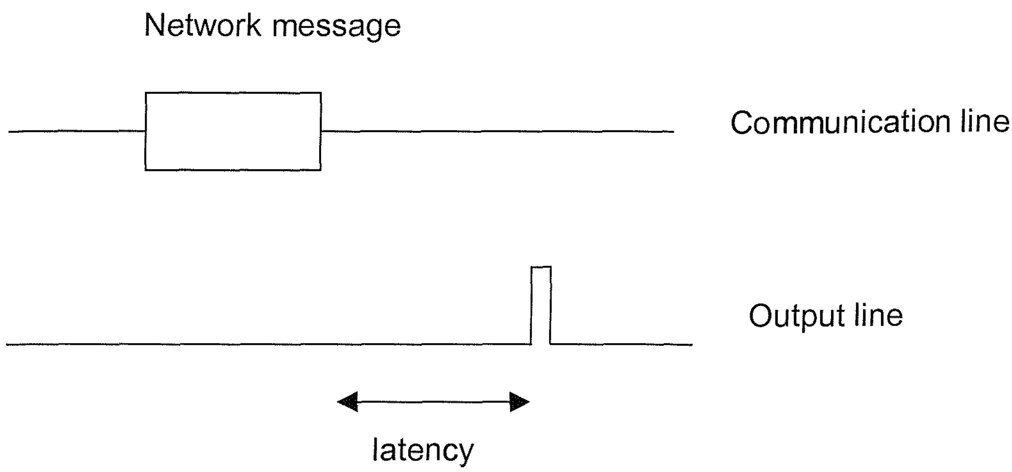


Figure 5.3 Latency measurement

These classes are used by the main program as follows (Figure 5.4):

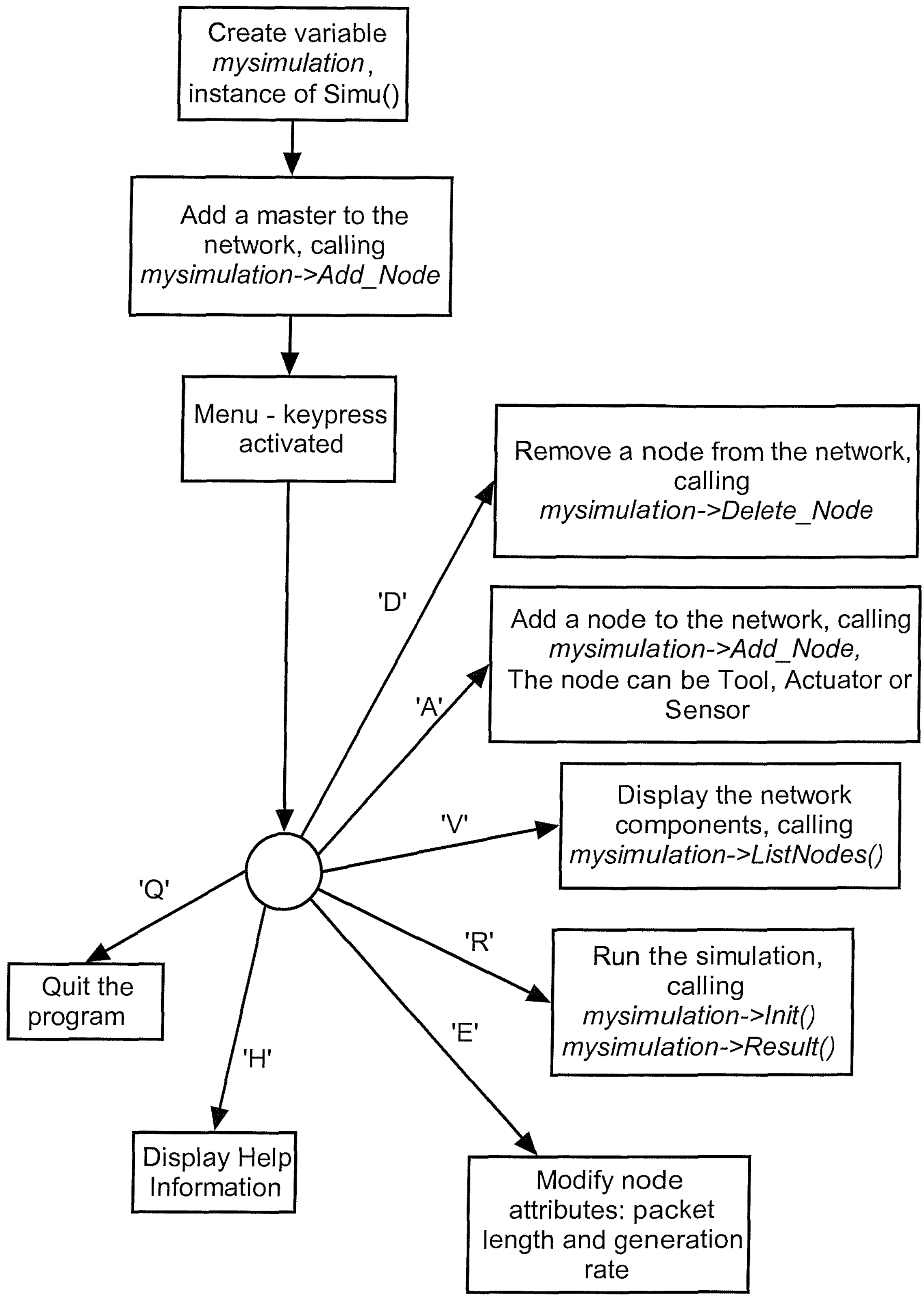


Figure 5.4 Main menu flowchart

Source code of the simulation software is included in Appendix C.

5.4.1 Results and tests

At the time of those tests, the prototype network consisted of a PC master node and three slave nodes: one thruster node (actuator), one navigation node (sensor) and one video node (tool).

Measurements have been made on the prototype of the following values:

1. Token Rotation Time : this is the time it takes for the token to loop around all the nodes. This measurement was made by the PC by keeping a log file of the time when it received the token.

2. Delay at node : this is the time it takes between when the message is created and when it is transmitted. For example, the time between when a node reads the result of a DAC conversion, and when the result of that conversion is transmitted over the network. A time stamp was created at the time of the DAC reading, and another at the time of transmission. Both time stamps were transmitted within the message, and could then be processed by the PC.

3. Delay at PC : this is the time it takes between when the message is created and when it is transmitted. Time stamps for those events were logged onto a file and processed later.

All the timing information was saved onto files which were readable by a spreadsheet program. This allowed the user to get information such as average trt and delay.

Results from the simulation have been compared with measured values.
The delay is variable, and can be defined as moving randomly around an average value. Only the average result is represented on Figure 5.5, although a confidence interval has also been obtained.
Results were also obtained from the actual test rig; the token rotation time (trt) was measured. For each node using the network for transmission, the delay between when the message is generated, and when it is actually transmitted was measured.

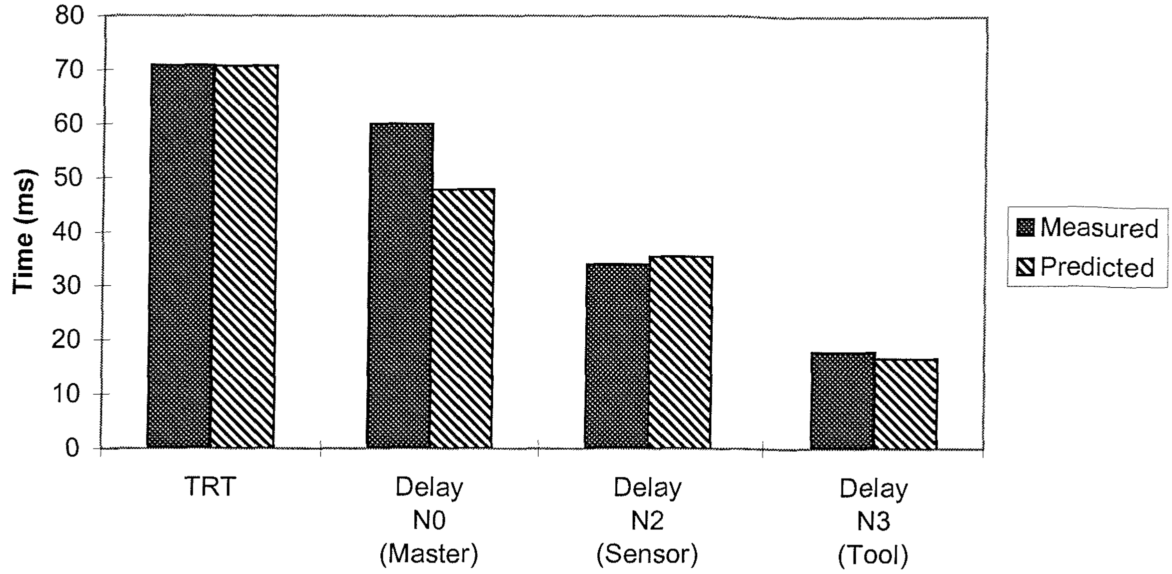


Figure 5.5 Delay estimation and measurements results

The results showed a good match for the trt and the slave nodes (Figure 5.5). The results are poor for the PC prediction (Node 0), this is probably due to the fact that the model used in the simulation for station latency is simple. This is fine for real time embedded systems as the slave nodes, but insufficient for a more complex behaviour such as a PC running software under an operating system.

The values obtained can also be used as representative values of the delay that the experimental system will be expected to cope with. The following set-up was chosen as a typical ROV configuration, as it is representative of the prototype vehicle :

- Number of nodes: 4 nodes in total, one master node and three slave nodes (thruster, navigation and tools nodes).
- Thruster node : receives packet sizes of 48 bits, but does not transmit any data.
- Navigation node : transmits packet of 80 bits.
- Video node : transmits packets of 96 bits.
- Frame error rate 1 in 1000 frame is corrupted.
- Baud rate : 9.6Kbd.
- Umbilical length : 500 meters.
- Latency for each node 85 ms.

For the above values the resulting transportation delays occuring for each node varied between 20 to 70 ms on average (See Figure 5.5).

The simulation is also a very valuable tool that can be used to design, develop, validate and modify networks. Many parameters can be varied: number of nodes, packet length, transmission rate, and latency. As a design tools it allows the user to experiment with different configurations and find out the effect of modifying several parameters before investing in hardware. As a development tool, it allows the user to compare the expected results against real measurements, and therefore to detect any malfunction which would not be obvious otherwise. If the results match the network can be shown to function as expected and can be validated. In a similar way the simulation can also be used to investigate the effect of possible modifications. For example it is possible to find out by how much will the delay increase when more nodes are added, and by how much the transmission speed has to be increased if the choice is to have the same delays as before the nodes were added.

6. EFFECT OF VARIABLE DELAY ON CLOSED LOOP CONTROL

This chapter investigates the effect of delays within control loops, and describes a way of controlling processes with variable delays. This is particularly relevant to the prototype ROV. Indeed the transportation delay within a control loop run over a network is affected by parameters such as the number and type of nodes present on the network.

ROVs are often tailored to suit a particular task, and having to re-tune all closed loop controllers within the vehicle at each modification would be impractical.

The idea of having an 'universal' controller, which is not affected by changes in transportation delay would seem to be the answer. A potential solution, using self-tuning control, has been found and has been tested both in simulation and in experiments.

6.1 Definition of system studied

Originally it was expected that control experiments would be carried on one of the ROV instruments. First the camera tilt mechanism unit was targeted, where a position control system could be designed. Initial tests however showed that this unit was very slow compared to the range of delays introduced by the prototype communication system. The unit could be modelled as a delayed integrator, with a delay of 0.9 sec.

One other way of experimenting was to control the speed of the thruster. However it proved difficult to find a suitable speed feedback signal without any major design changes. The internal Seaeye thruster electronics give a step response as follows: a delay of 90 msec and a sharp rise (taking about 10 msec) to a settling point. The speed value was taken from the Hall-effect sensor existing in the thruster, giving a frequency proportional to the thruster speed: (f(Hz)= speed(RPM) x 0.1). A plot of the measured response is shown in Figure 6.1, where the A trace is the demand, and the B trace is the output from the Hall sensor.

Not enough transient points (4 points can be read in the rising step) could be obtained in order to build a model which could represent the motor behaviour accurately.

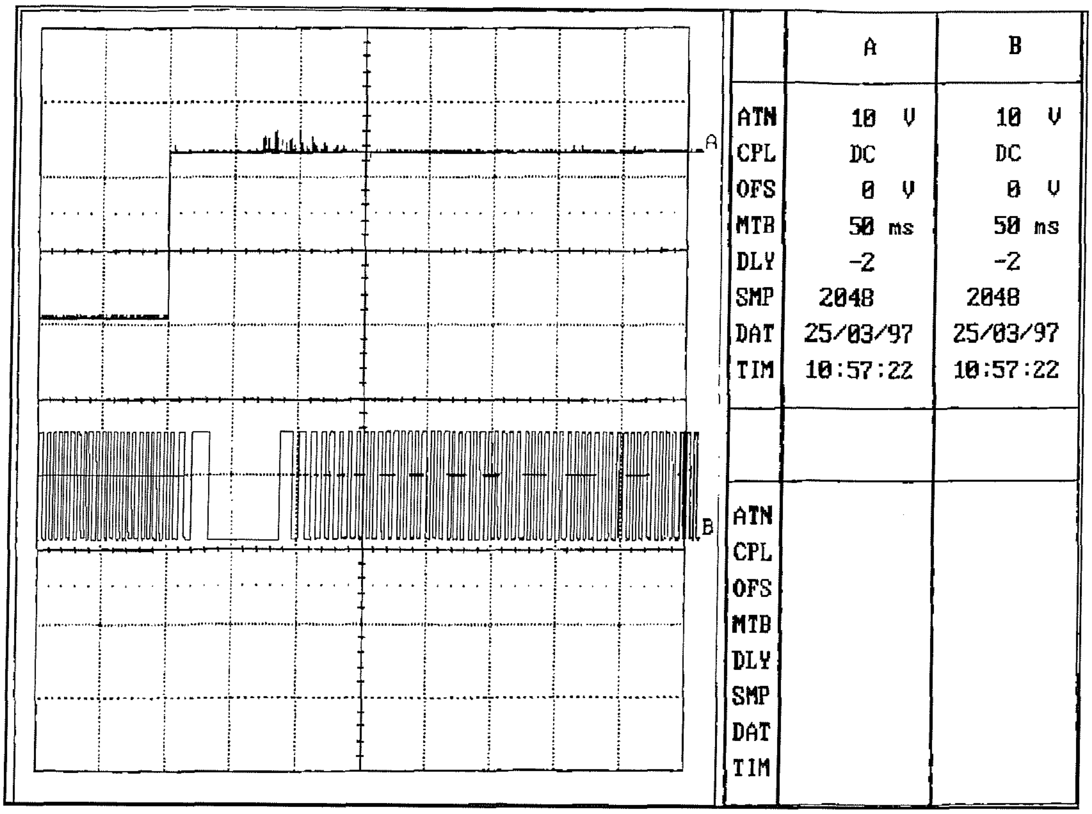


Figure 6.1 Speed step response of Seaeye thruster

The only other possibility of testing closed loop control on the vehicle was the vehicle itself, by controlling heading or depth. Although a simple PID controller was implemented for the water tests, it was decided that this would not be a very good starting point for the following reasons:

- difficulty of setting up the test: need access to tank, a vehicle chassis, help to launch vehicle
- difficulty of modelling: it would be problematic to assess exactly what contribution the delay has with respect to the other unknown parameters.

The auto-heading PID controller used during the water tests was tuned in-situ using a trial and error method, which gave acceptable results. Because the vehicle's response will vary with respect to ballast and payload, trial and error tuning remains a favourite method in industry to cope with such a complex behaviour.

The solution chosen was to use a laboratory servo system. Minor changes had to be made to one of the existing nodes to provide adequate amplification for the rig command and feedback signals. The existing ADC and DAC were used, allowing most of the software for both the network and the local application to be re-used. Since all tests could now be implemented on the bench, this allowed for more flexibility and testing time.

The servo system used was a 'Feedback Modular servo system MS150 MkII', which is primarily intended for experimental use by students investigating closed-loop systems [27]. The motor used has split field winding, with current flow in each part of the coil being controlled by a transistor. (Figure 6.2)

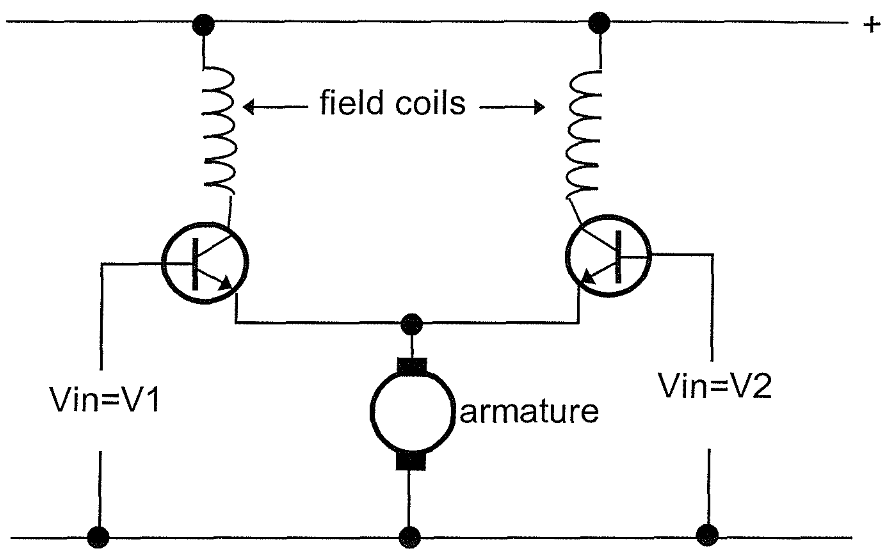


Figure 6.2 Schematic of armature control motor

The result of this arrangement is that the speed of the motor is proportional the input voltage Vin. Due to friction, a minimum voltage is needed to start the motor. An integral tacho-generator is fitted with the unit.

6.2 Experimental Setup

The experimental setup for the control experiments were as follows:

The PC was acting as a master and connected to one subsea node (test node). The hardware of this test node originated from a thruster node but the I/O side was modified to integrate the ADC and DAC.

The test node has one output which is the command line going to the feedback rig, and two inputs, one is the speed feedback and the other is the position feedback (Figure 6.3).

The software was designed so that the control loop could either be closed or open. The closed loop algorithm could be run either:

a) **locally** : *the node implements the PID controller independently*

The test node software executes the following tasks:

1. read the ADC conversion results on both channels
2. compute PID controller output
3. set the DAC to convert the PID output and go back to 1.
4. When the access to the network is available (interrupt driven), transmit a message to the PC containing feedback , command and time values.

The PC executes the following tasks:

1. run network software
2. read values received and log into file

b) **remotely** : *the PC computes the PID depending on results obtained on remote node*

The test node software executes the following tasks:

1. read the ADC conversion results on both channels
2. set the DAC to convert the command and go back to 1.
3. When the access to the network is available (interrupt driven), transmit a message to the PC containing feedback values.
4. On interrupt, receive the command value from the PC

The PC executes the following tasks:

1. run network software
2. read values received and log into file
3. compute PID controller output

The PC could also be made to add a time delay in the control loop.

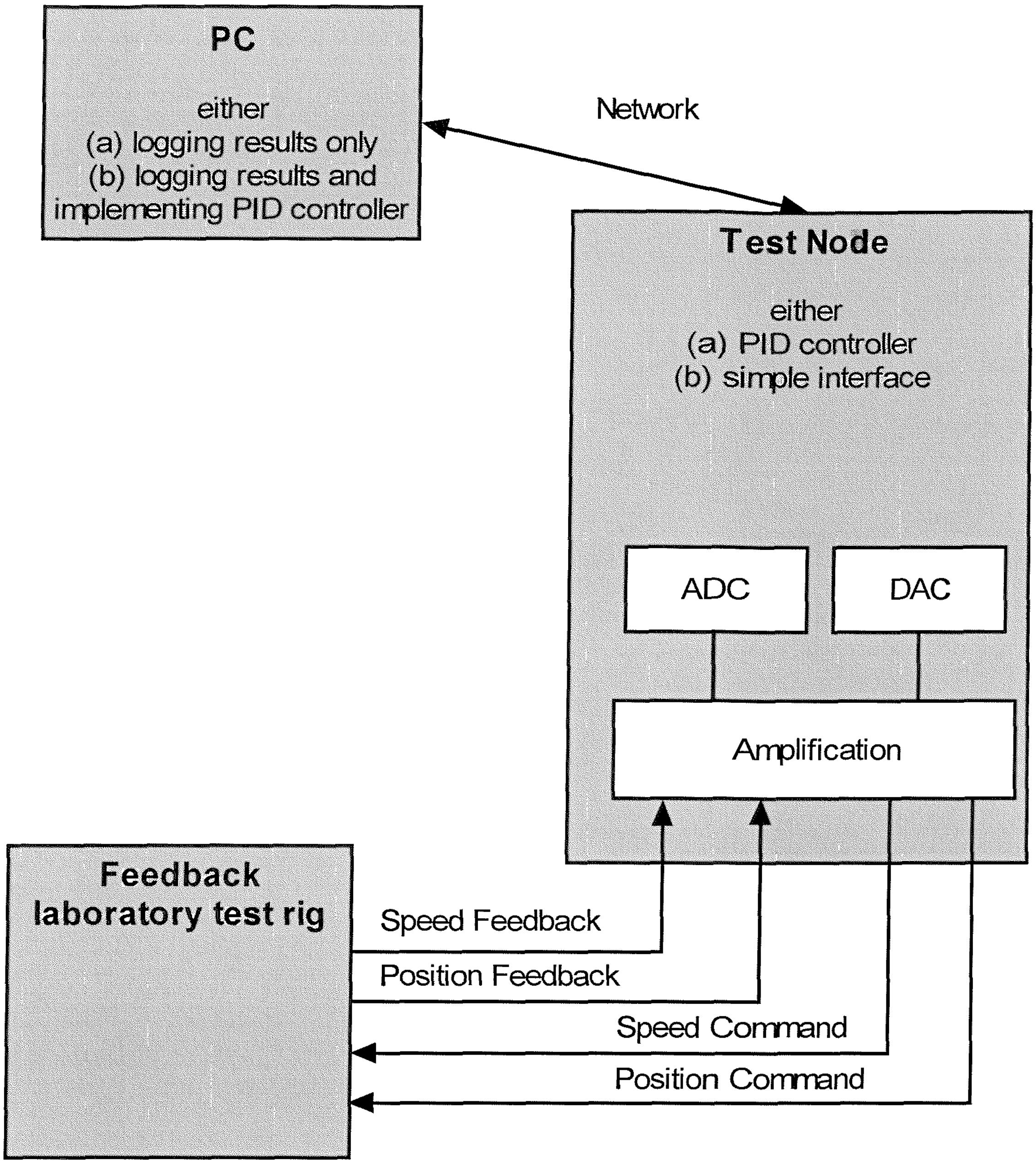


Figure 6.3 The control tests experimental setup

The network used here was minimal, with only one master and one slave. Additional delays could be added artificially by the PC (master), rather than by adding other slave nodes.

6.3 Experiments without introduced delay

6.3.1 Open loop measurements

The purpose of the first experiment was to determine a dynamic model of the test rig, The information obtained can then be used to model and analyse the rig's behaviour in more complex situations.

The first step was to establish the tachometer calibration. The results are shown in Figure 6.4. The tachometer readings could be interpreted as a linear function $y = 25.7x - 953.3$.

This linear function is only valid for digital readings below 255, after this point the ADC is saturated. This implies that we are not able to read speeds above 5610 RPM. Therefore the experiments were operating the motor around a much lower speed (typically 3000 rpm). The offset of the digital reading is due to an offset in the amplification stage, and has been included in the above equation.

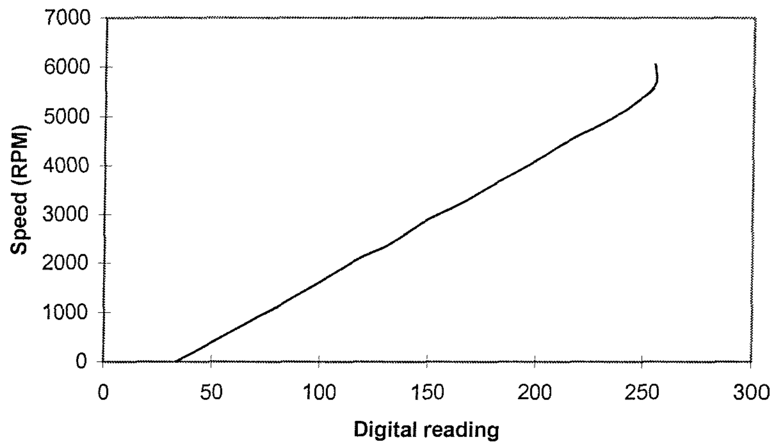


Figure 6.4 Tacho calibration curve

An open loop configuration was setup, the test node generated the step function (height 255 digital value, or 14V) which was feeding the motor and transmitted the readings of the speed output to the PC. Those readings were then converted to RPM using the calibration values. Results are shown in Figure 6.5.

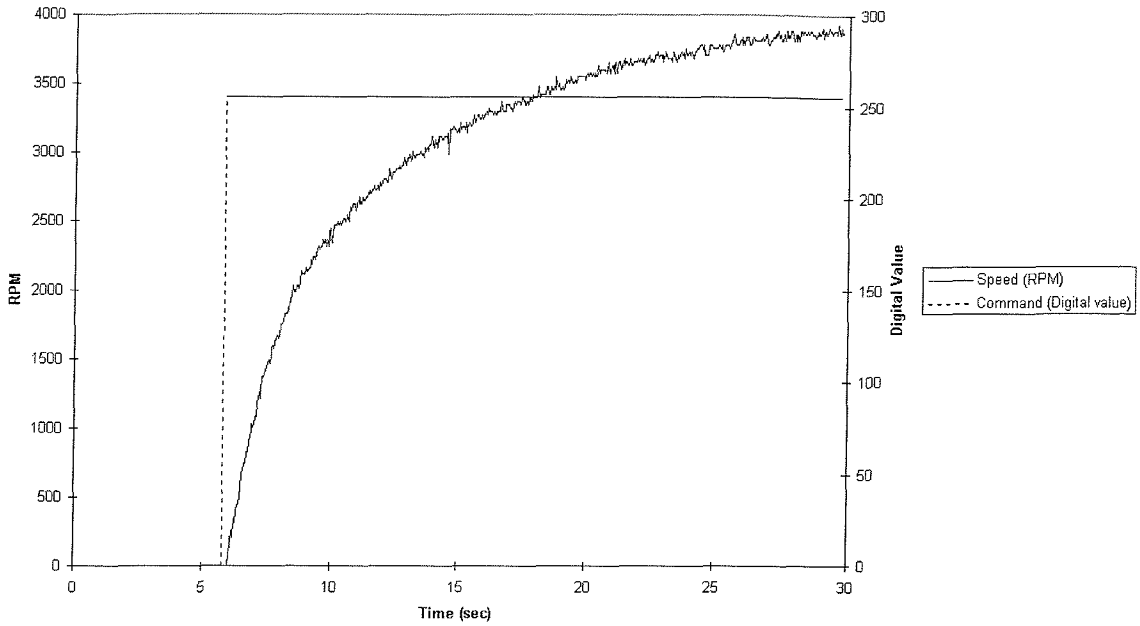


Figure 6.5 Experimental open loop step response

The open loop response was obtained by giving a step speed command to the motor, shown as 'Command' in Figure 6.5. The response, shown as 'Speed' in Figure 6.5, allowed us to approximate the model of the rig as a first order model, with a time constant of 4.4 sec and a steady state gain of 4000/14=285 rpm/V, giving the following function:

$$G(s) = \frac{285/\tau}{s + 1/\tau} = \frac{64.93}{s + 0.23}$$

This transfer function was entered in a simple Simulink model and a similar open loop test was setup (Figure 6.6). The results are shown in Figure 6.7 and match the experimental results.

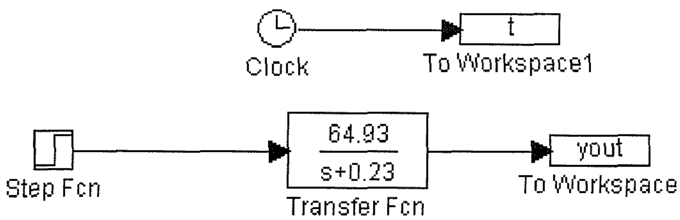


Figure 6.6 Open loop simulation in Simulink

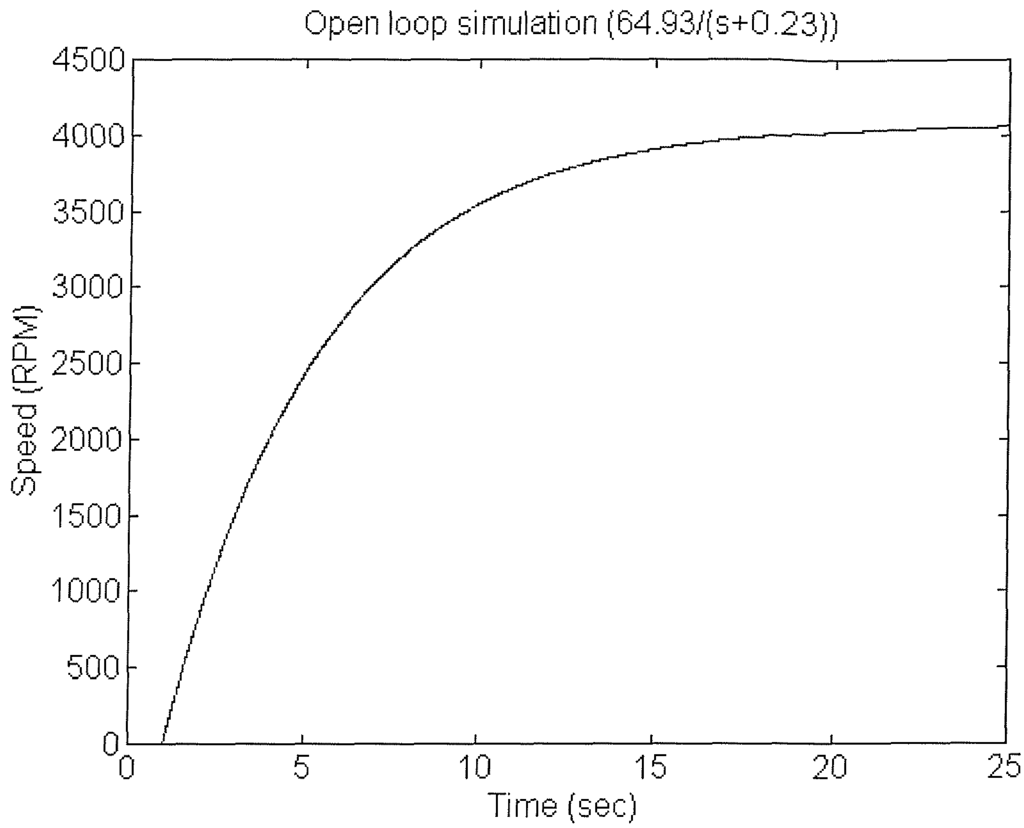


Figure 6.7 Simulated open loop step response (Step demand generated at t=1 sec)

6.3.2 Closed loop control simulation and experiment

Simulink was also used to test PID values on the model (Figure 6.8); adequate values were found to be P=12.5 I=0.5 D=10 (Figure 6.9). The saturation block in Figure 6.8 represents the characteristic of the amplifier.

The controller values were obtained using the Ziegler-Nichols [29] tuning rule and then by iterative trials in the simulation. The demand is shown as a solid line and the response as a dotted line.

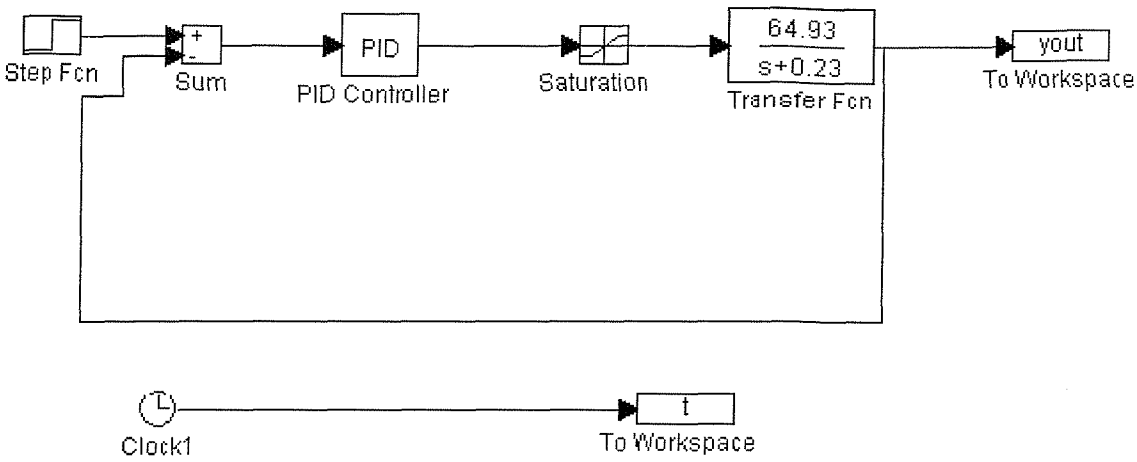


Figure 6.8 Closed-loop PID control in Simulink

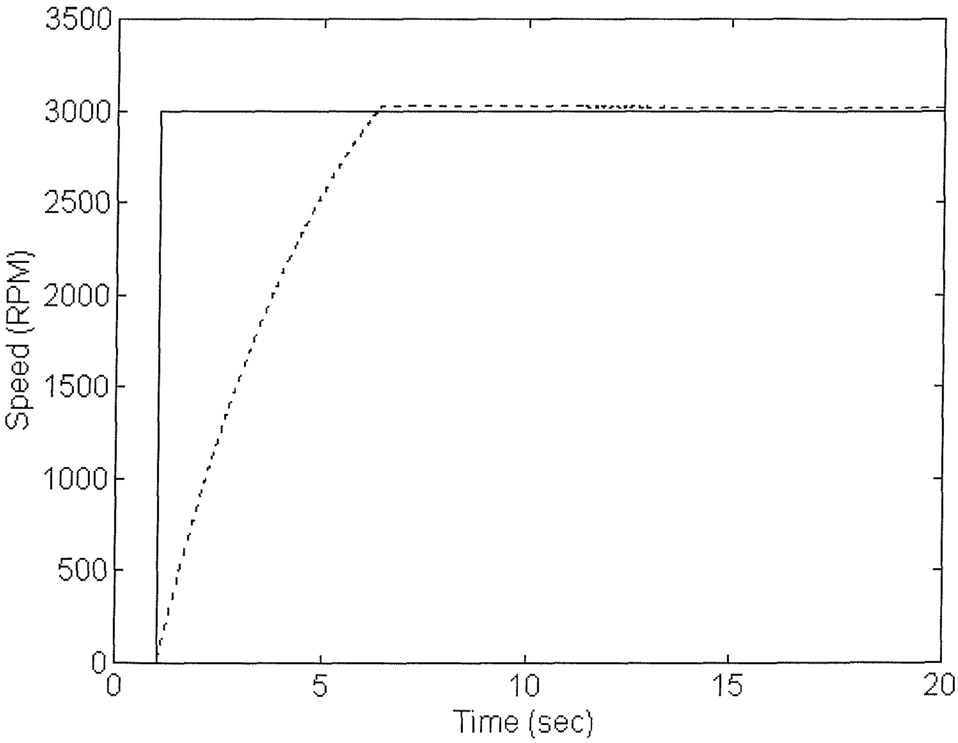


Figure 6.9 Simulated locally run (case a) PID step response (setpoint = 3000 RPM)

The PID values were also tested on the rig, with the PID implemented locally (case a in Figure 6.3) on the test node, and the PC only logging results. Experimental result are shown in Figure 6.10.

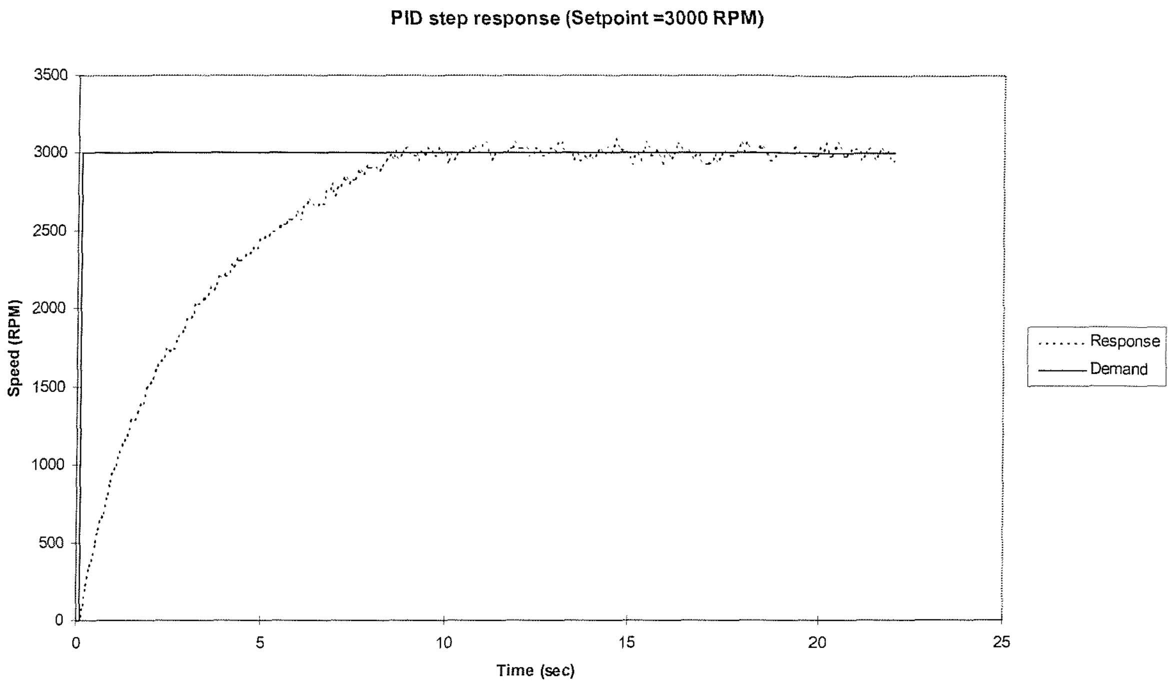


Figure 6.10 Experimental locally run (case a) PID step response

This confirms the results obtained from the simulation with no overshoot or steady state error. The rise time is much longer in the experimental result.

6.4 Delayed case

6.4.1 Simulation

A delay was added in the closed loop PID simulation (Figure 6.11). Results showed that increasing the delay without modifying the PID controller coefficients causes the response to deteriorate. The response cannot reach a steady state value, but oscillates around a final value.

The spread of the oscillation becomes more noticeable as the delay increases (Figure 6.12 to Figure 6.15).

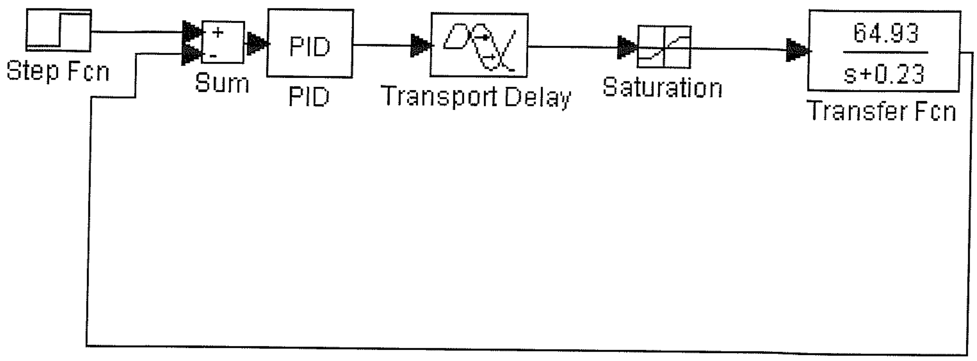


Figure 6.11 Closed loop PID simulation with variable delay

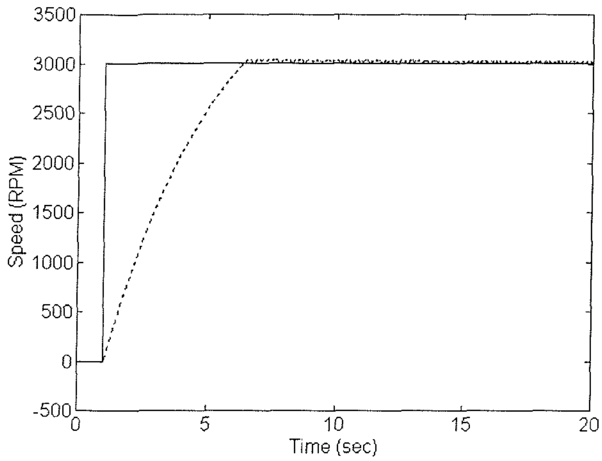


Figure 6.12 PID step response with no delay added

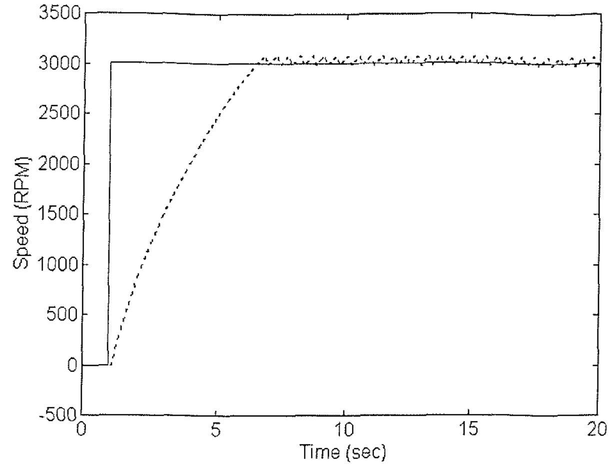


Figure 6.14 PID step response with 100 msec added delay

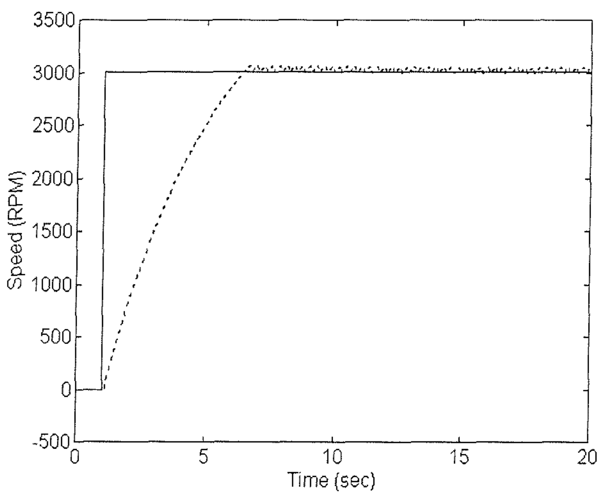


Figure 6.13 PID step response with 50 msec added delay

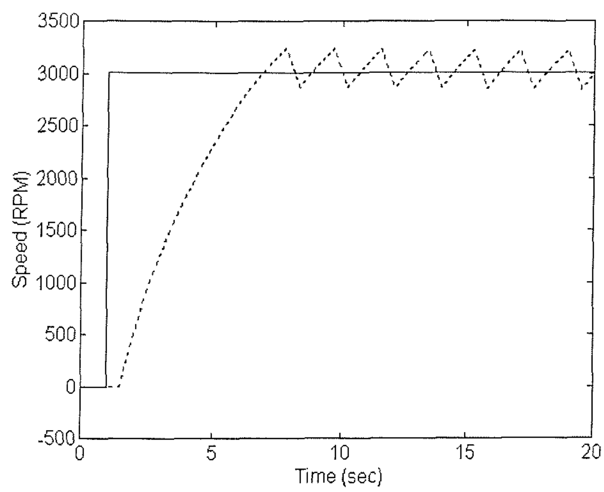


Figure 6.15 PID step response with 500 msec delay added

6.4.2 Experimental results

The software implemented on the test rig was such that the PID controller was implemented on the PC (Case b in Figure 6.3). As well as the communication delay between the two nodes, a delay could be artificially added by the PC. Different values of delay were added, and results are shown in figures 6.16 to 6.19. An oscillation appears in the steady state, with an amplitude that increases with the delay.

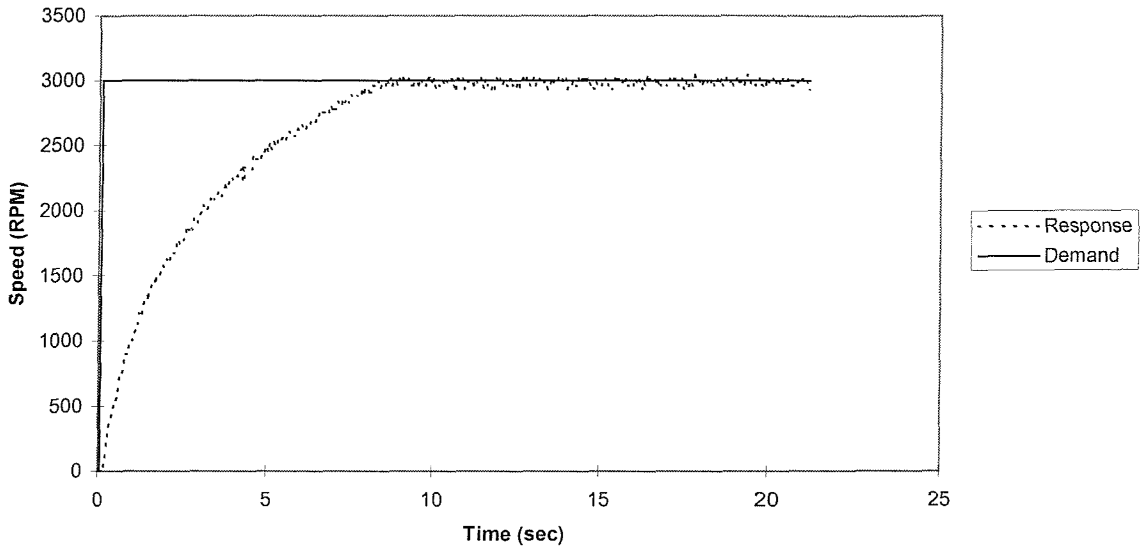


Figure 6.16 Remote PID step response without added delay

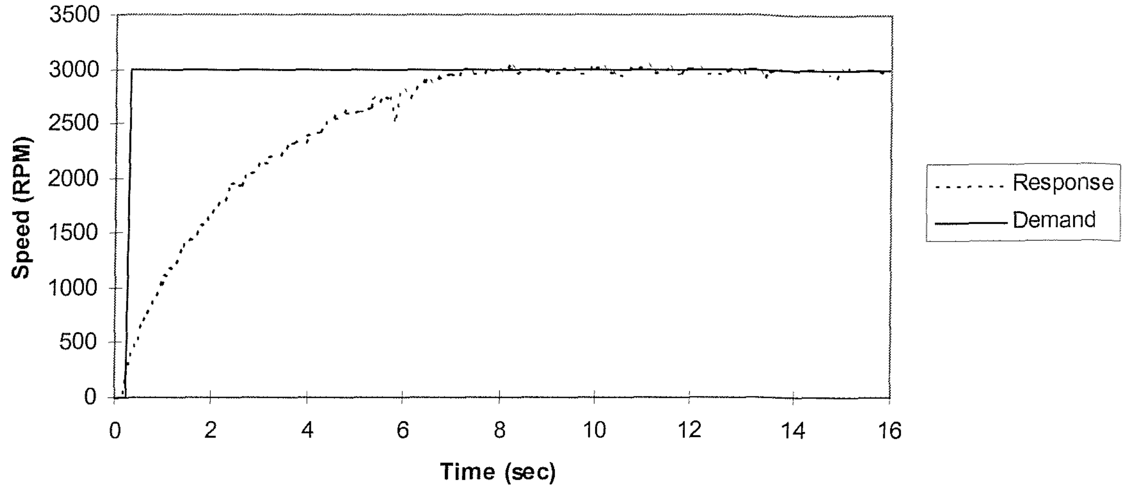


Figure 6.17 Remote PID step response with 50 msec added delay

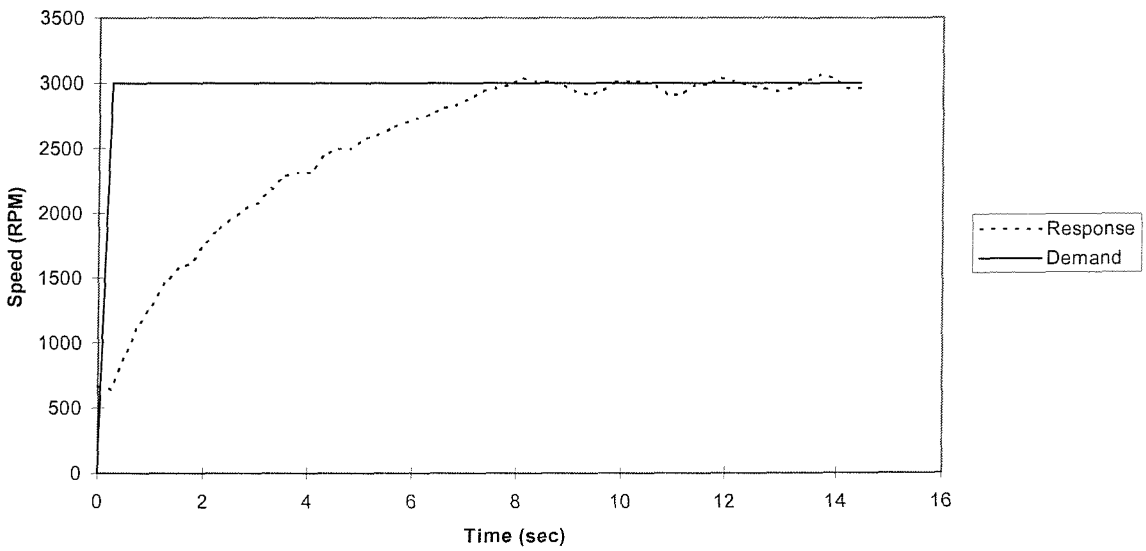


Figure 6.18 Remote PID step response with 200 msec delay added

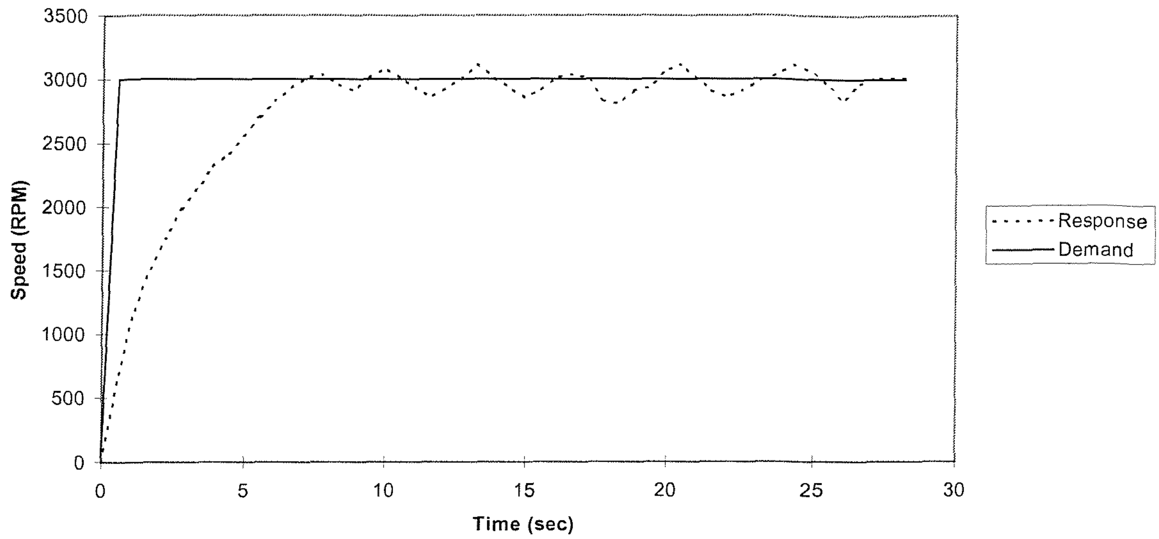


Figure 6.19 Remote PID step response with 500 msec delay added

6.4.3 Comparison and conclusion

The experimental results obtained are very similar to what the simulation predicted. This is most obvious with the case where the delay was 500 msec (Figure 6.19); the oscillation is very clear and corresponds to the simulated prediction (Figure 6.15).

This type of oscillation would be a problem in practice, for example if this control system was the auto-heading of a ROV. Although the overall direction would be accurate, the oscillations would make the video image taken by the vehicle 'shaky' and unusable.

Solutions to this particular problem have been researched in the past, but mainly for system with unknown time delays, rather than a variable time delay, as in this case. Because of the spread in the use of distributed control systems, this particular problem will become more generalised. Most of those solutions are based on an adaptive PID controller; where the PID controller's parameters are calculated, based either on the results of a parameter estimation or pattern recognition of the system. They have been shown to give good results for fixed and non-significant delays [30]. One other way of coping with this problem is to calculate, based on knowledge of the network, what the worst case delay is, and use this as a factor to design controller [32].

Typically, the type of adaptive controller described above have been tested in two phase tests, where the first phase typically consists of applying disturbance to the system in order to estimate parameters for a model, and the second phase implements a suitable controller. A better solution would be to continuously estimate the delay, and modify the PID parameters accordingly. This idea has been developed, both in theory and practice, and the self-tuning system is described next.

6.5 Self-tuning system for delayed processes

6.5.1 Theory

This section shows how a self-tuning controller can be implemented, so that the response of a process does not deteriorate when a transport lag is introduced or modified in the process, as shown in the previous section.

6.5.1.1 Time delays in state space

It is well known how to model systems with time delays in state space [28]:

The non delayed case can be written as

$\dot{x} = \boldsymbol{A}x + \boldsymbol{B}u$

with $\mathbf{x}$ *the state variable vector and* $\mathbf{u}$ *the input signal vector, an* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *matrices.*

The outputs of a linear system can be related to the state variables and input signals by the following :

$y = \boldsymbol{C}x + \boldsymbol{D}u$, *where* $\mathbf{y}$ *is a vector of output signals*

With the introduction of a delay τ in the system, this becomes:

$\dot{x} = \boldsymbol{A}\,x(t) + \boldsymbol{B}\,u(t-\tau)$

6.5.1.1.1 Delay smaller than sampling period τ<h

The above leads to discrete time state equations [28]:

$$\mathbf{x}(kh+h) = e^{Ah}\mathbf{x}(kh) + \int_{kh}^{kh+h} e^{A(kh+h-s')}\mathbf{B}\,\mathbf{u}(s'-\tau)ds'$$

$$= e^{Ah}\mathbf{x}(kh) + \int_{kh}^{kh+\tau} e^{A(kh+h-s')}\mathbf{B}\,ds'\,\mathbf{u}(kh-h) + \int_{kh+\tau}^{kh+h} e^{A(kh+h-s')}\mathbf{B}\,ds'\,\mathbf{u}(kh)$$

$$= \Phi\mathbf{x}(kh) + \Gamma_1\mathbf{u}(kh-h) + \Gamma_0\mathbf{u}(kh)$$

with

$$\Phi = e^{Ah}$$

$$\Gamma_1 = e^{A(h-\tau)} \int_0^{\tau} e^{As'} ds' \boldsymbol{B}$$

$$\Gamma_0 = \int_0^{h-\tau} e^{As'} ds' \boldsymbol{B}$$

In a standard representation matrix form this gives

$$\begin{bmatrix} \boldsymbol{x}(kh+h) \\ \boldsymbol{u}(kh) \end{bmatrix} = \begin{bmatrix} \Phi & \Gamma_1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}(kh) \\ \boldsymbol{u}(kh-h) \end{bmatrix} + \begin{bmatrix} \Gamma_0 \\ I \end{bmatrix} \boldsymbol{u}(kh) \qquad \textbf{(Eq. 1)}$$

6.5.1.1.2 Delay larger than sampling period $\tau > h$

For the case of the time delay being larger than the sampling period, $\tau = (d-1)h + \tau' \;\; 0 < \tau' \leq h$, d representing the entire part of the delay and τ' the fractional part of the delay, the discrete state equation can be generalised as [28]:

$$\boldsymbol{x}(kh+h) = \;\; \boldsymbol{x}(kh) + \Gamma_1 \boldsymbol{u}(kh-dh) + \Gamma_0 \boldsymbol{u}(kh-dh+h)$$

In matrix form this gives:

$$\begin{bmatrix} \boldsymbol{x}(kh+h) \\ \boldsymbol{u}(kh-dh+h) \\ \vdots \\ \boldsymbol{u}(kh-h) \\ \boldsymbol{u}(kh) \end{bmatrix} = \begin{bmatrix} & \Gamma_1 & \Gamma_0 & \cdots & 0 \\ 0 & 0 & I & \cdots & 0 \\ \vdots & & & & \\ 0 & & & \cdots & I \\ 0 & & & \cdots & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}(kh) \\ \boldsymbol{u}(kh-dh) \\ \vdots \\ \boldsymbol{u}(kh-2h) \\ \boldsymbol{u}(kh-h) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ I \end{bmatrix} \boldsymbol{u}(kh) \qquad \textbf{(Eq. 2)}$$

6.5.1.2 Z transforms of delayed processes

6.5.1.2.1 Z transforms with no delays

It is possible to apply a Z transform to discrete state space equations [28]. The classic algorithm for a non-delayed case is given below.

Given the discrete state equation

$$x(k+1)=\Phi x(k)+\Gamma u(k)$$

and taking its Z transform

$$\sum_0^\infty Z^{-k}\mathbf{x}(k+1)=\sum_0^\infty Z^{-k}(\Phi\mathbf{x}(k)+\Gamma\mathbf{u}(k))$$

$$Z\left(\sum_0^\infty Z^{-k}\mathbf{x}(k)-\mathbf{x}(0)\right)=\sum_0^\infty \Phi Z^{-k}\mathbf{x}(k)+\sum_0^\infty \Gamma Z^{-k}\mathbf{u}(k)$$

$$Z[X(Z)-x(0)]=\Phi X(Z)+\Gamma U(Z)$$

$$X(Z)=(ZI-\Phi)^{-1}[Zx(o)+\Gamma U(Z)]$$

$$\text{and } Y(Z)=C(ZI-\Phi)^{-1}Zx(o)+C(ZI-\Phi)^{-1}\Gamma U(Z)$$

The Z transform of the system is $H(Z)=C(ZI-\mathbf{\Phi})^{-1}\ \Gamma$

6.5.1.2.2 Z transforms for delayed cases

A similar approach can be taken for a delayed case (τ<h), this highlights the contribution of the delay.

Given the discrete state equations (from 6.5.1.1.1)

$\mathbf{x}(k+1)=\Phi\mathbf{x}(k)+\Gamma_1\mathbf{u}(k-1)+\Gamma_0\mathbf{u}(k)$ *this can also be expressed as :*

$$\mathbf{x}(k+2)=\Phi\mathbf{x}(k+1)+\Gamma_1\mathbf{u}(k)+\Gamma_0\mathbf{u}(k+1)$$

Taking the Z transform of the above equation :

$$\sum_0^\infty Z^{-k}\mathbf{x}(k+2)=\sum_0^\infty Z^{-k}(\Phi\mathbf{x}(k+1)+\Gamma_1\mathbf{u}(k)+\Gamma_0\mathbf{u}(k+1))$$

$$Z^2\left(\sum_0^\infty Z^{-k}\mathbf{x}(k)-\mathbf{x}(0)-Z^{-1}\mathbf{x}(1)\right)=\sum_0^\infty \Phi Z^{-k}\mathbf{x}(k+1)+\sum_0^\infty \Gamma_1 Z^{-k}\mathbf{u}(k)$$

$$+\sum_0^\infty \Gamma_0 Z^{-k}\mathbf{u}(k+1)$$

$$Z^2[X(Z)-\mathbf{x}(0)-Z^{-1}\mathbf{x}(1)]=Z\Phi X(Z)+\Gamma_1 U(Z)+Z\Gamma_0 U(Z)$$

$$X(Z)=(Z^2I-Z\Phi)^{-1}[Zx(o)+x(1)+U(Z)(\Gamma_1+Z\Gamma_0)]$$

and

$$Y(Z)=\mathbf{C}X(Z)$$

$$Y(Z)=\mathbf{C}(Z^2I-Z\Phi)^{-1}[Zx(o)+x(1)]$$

$$+\mathbf{C}(ZI-\Phi)^{-1}(\Gamma_1+Z\Gamma_0)U(Z)$$

the first term is negligeable, this simplifies to

$$Y(Z)=\mathbf{C}(ZI-\Phi)^{-1}(\Gamma_1+Z\Gamma_0)U(Z)$$

The Z transform of the system is $H(Z)=\dfrac{Y(Z)}{U(Z)}$

$$H(Z)=\mathbf{C}(Z^2I-Z\Phi)^{-1}(\Gamma_1+Z\Gamma_0)$$

6.5.2 Example of application to a first order system

As an example, the above results can be applied to a first order system, with the transfer function:

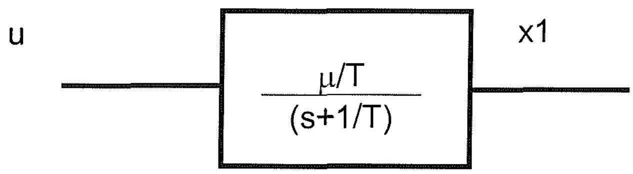
$$G(s)=\frac{\mu}{1+sT}=\frac{\frac{\mu}{T}}{s+\frac{1}{T}}$$

Converting into state space form $\dot{x}=\boldsymbol{A}\,\boldsymbol{x}+\boldsymbol{B}\,\boldsymbol{u}$

$$x1=\frac{\frac{\mu}{T}}{s+\frac{1}{T}}U \Rightarrow \dot{x}1=\frac{\mu}{T}U-\frac{x1}{T}$$

$$y=x1$$

$$\boldsymbol{A}=-\frac{1}{T}\quad \boldsymbol{B}=\frac{\mu}{T}\qquad C=1$$



6.5.2.1 Non-delayed case

In order to illustrate how the theory developed in 6.5.1 can be applied to the above system, we can initially obtain the Z-transform for a non-delayed case. This simple example is a good starting point before moving on to the more complex delayed case. This gives the following digital state equation values[9]:

$$\Phi = e^{Ah} = e^{\frac{-h}{T}}$$

$$\Gamma = \int_0^h e^{\frac{-s}{T}}ds\frac{\mu}{T}=\frac{\mu}{T}(1-e^{\frac{-h}{T}})$$

The Z transform of the system is :

$$H(Z)=C(ZI-\Phi)^{-1}\Gamma=(Z-e^{\frac{-h}{T}})^{-1}\frac{\mu}{T}(1-e^{\frac{-h}{T}})=\mu\frac{(1-p)}{Z-p}$$

with $p=e^{\frac{-h}{T}}$

[9] The purpose of those calculations is to highlight the way in which the theory developed can be applied. In this particular case, looking up in a Z-transform table would be much easier and give the same result, this is only used as an example.

We have moved from continuous to digital state-space variables, and then obtained the Z-transform of the system, the same method is used for a delayed system in the following section.

6.5.2.2 Delayed case

In a delayed case, a similar reasoning can be applied. The discrete state equations values are:

$$\Phi = e^{Ah} = e^{\frac{-h}{T}}$$

$$\Gamma_1 = e^{A(h-\tau)}\int_0^{\tau} e^{As}ds\boldsymbol{B} = e^{\frac{\tau-h}{T}}\int_0^{\tau} e^{\frac{-s}{T}} ds\frac{\mu}{T}$$

$$= \mu e^{\frac{\tau-h}{T}}\frac{T-Te^{\frac{-\tau}{T}}}{T} = \mu e^{\frac{\tau-h}{T}}(1-e^{\frac{-\tau}{T}})$$

$$\Gamma_0 = \int_0^{h-\tau} e^{As}ds\boldsymbol{B} = \int_0^{h-\tau} e^{\frac{-s}{T}}ds\frac{\mu}{T} = \mu(1-e^{\frac{\tau-h}{T}})$$

The Z transform is given as:

$$H(Z) = C(Z^2I - Z\Phi)^{-1}(\Gamma_1 + Z\Gamma_0) \text{ from 6.5.1.2.2, using the values of } \Phi, \Gamma_0 \text{ and } \Gamma_1$$

$$H(Z) = \mu\frac{e^{\frac{\tau-h}{T}}(1-e^{\frac{-\tau}{T}}) + Z(1-e^{\frac{\tau-h}{T}})}{Z(Z-e^{\frac{-h}{T}})} = \frac{\mu}{Z(Z-e^{\frac{-h}{T}})}((e^{\frac{\tau-h}{T}} - e^{\frac{-h}{T}}) + Z(1-e^{\frac{\tau-h}{T}}))$$

using $\tau = \varepsilon h$

$$H(Z) = \frac{\mu}{Z(Z-e^{\frac{-h}{T}})}(e^{\frac{-h}{T}}(e^{\frac{\varepsilon h}{T}} - 1) + Z(1-e^{\frac{(\varepsilon-1)h}{T}}))$$

$$H(Z) = \frac{\mu}{Z(Z-e^{\frac{-h}{T}})}(e^{\frac{-h}{T}}(\left(e^{\frac{-h}{T}}\right)^{-\varepsilon} - 1) + Z(1-e^{\frac{-h}{T}(1-\varepsilon)}))$$

using $p = e^{\frac{-h}{T}}$

$$H(Z) = \frac{\mu}{Z(Z-p)}(p(p^{-\varepsilon} - 1) + Z(1-p^{(-\varepsilon+1)}))$$

By using $\alpha = \dfrac{p(p^{-\varepsilon} - 1)}{(1 - p)}$, *this simplifies to*

$$H(Z)=\frac{\mu(1-p)}{Z(Z-p)}(\alpha+Z\frac{(1-p^{(-\varepsilon+1)})}{1-p})$$

$$H(Z)=\frac{\mu(1-p)}{Z(Z-p)}(\alpha+Z\frac{1-p-p^{(-\varepsilon+1)}+p)}{1-p})$$

$$H(Z)=\frac{\mu(1-p)}{Z(Z-p)}(\alpha+Z\left(1-\frac{p(p^{-\varepsilon}-1)}{1-p}\right))$$

$$H(Z)=\frac{\mu(1-p)}{Z(Z-p)}(\alpha+Z(1-\alpha))$$

As a reminder, the Z-transform of a non-delayed case is : $H(Z)==\mu\dfrac{(1-p)}{Z-p}$

This can be generalised to a first order model with a delay $\tau = dh + \varepsilon h$, larger than the sampling period.

$$G(s)=\frac{\mu}{1+sT}e^{-s\tau} \text{ with } \tau = dh+\varepsilon h, T>0,\ 0<\varepsilon<1$$

The effect of the delay can be simplified to:

- *for* $0<\varepsilon<1$ (delay)

$$\frac{y(Z)}{u(Z)}=Z^{-d}\mu\left(\frac{1-p}{Z-p}\right)\left(\frac{(1-\alpha)Z+\alpha}{Z}\right)=Z^{-(d+1)}\mu\left(\frac{1-p}{Z-p}\right)((1-\alpha)Z+\alpha)$$

$$\text{with } P=e^{-h/T} \text{ and } \alpha=\frac{p\left(p^{-\varepsilon}-1\right)}{1-p}$$

  The delay εh gives rise to a pole at the origin and a real negative zero q=-α/(1-α)

- *for* $-1<\varepsilon<0$ (anticipation)

$$\frac{y(Z)}{u(Z)}=Z^{-d}\mu\left(\frac{1-p}{Z-p}\right)((1-\beta)Z+\beta)=Z^{-(d+1)}\mu\left(\frac{1-p}{Z-p}\right)((1-\beta)Z^2+\beta Z)$$

$$\text{with } P=e^{-h/T} \text{ and } \beta=\frac{\left(p^{-\varepsilon}-p\right)}{1-p}$$

  The anticipation εh gives rise to a real negative zero q=-β/(1-β)

- For an uncertain delay τ=dh±εh -1<ε<1, this generalises to

$$H(Z) = Z^{-(d+1)} \frac{b0 + b1Z + b2Z^2}{Z - p}$$

$$\text{with for } 0 < \varepsilon < 1, b2 = 0, \frac{b1}{b0} = \frac{1-\alpha}{\alpha}$$

$$\text{and for } -1 < \varepsilon < 0, b0 = 0, \frac{b2}{b1} = \frac{1-\beta}{\beta}$$

The zeros are:

- $for \ \ 0 < \varepsilon < 1 \, \varepsilon = 1 - \sigma \ \ Zeros\ are - b0/b1 \ \ b2 = 0$

$$\varepsilon = 1 - \frac{1}{1 + \frac{b0}{b1}} = 1 - \frac{b1}{b0 + b1} = \frac{b0}{b0 + b1}$$

- $for \ \ -1 < \varepsilon < 0 \, \varepsilon = 1 - \sigma \ \ Zeros\ are - b1/b2 \ \ b0 = 0$

$$\varepsilon = 1 - \left(\frac{1}{1 + \frac{b1}{b2}} + 1 \right) = 1 - \frac{b2}{b1 + b2} - 1 = \frac{-b2}{b1 + b2}$$

- *So we can generalise for* $-1 < \varepsilon < 1$ *that*

$$\boxed{\varepsilon = \frac{b0 - b2}{b0 + b1 + b2}}$$

This feature allows us to predict the unknown delay of a first order system by analysing delay contributions in the Z model terms b0 and b2. [30] All of the following work is based on this result. By using an estimator to find b0, b1 and b2, we can calculate the value of the delay introduced.

6.5.3 Simulation

Simulation software has been developed in order to prove that the delay estimation theory can be applied within a closed loop control system.
This control method can be represented as in the diagram in Figure 6.20.

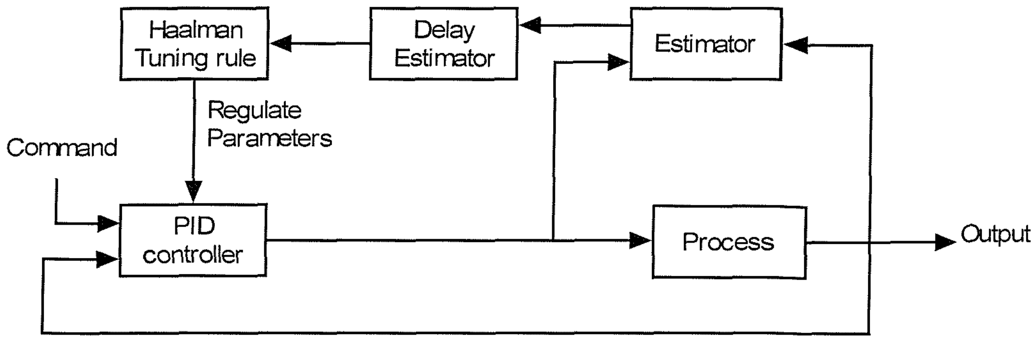


Figure 6.20 Diagram of the Self-tuning controller

Source code for this piece of software can be found in Appendices E and F.
The simulation software consists of several elements, which are represented in flowchart form in Figure 6.21:

1) **Data generation** : a default Z-transform model of a first order system is used to generate the data in order to simulate the process. Also at time step 250, a delay is added within that model and at time 650, it is removed. The Command value is also updated to show several step responses (Figure 6.22) The virtual sampling rate used in the Z model was of 40 ms, as it matches the experimental sampling rate.
2) **RLS Estimator**: a classic recursive least-squares estimation (RLS) method is used [29], which is described in Appendix H. It estimates the parameters of the Z model of the process. The model used is of a form:

   $H(Z) = Z^{-(d+1)} \dfrac{b0 + b1Z + b2Z^2}{Z - p}$. The RLS algorithm is stopped when the estimation error remains within a 2% band around the actual value.
3) **Delay estimation** : using the theory detailed in 6.5.1, it is then possible to estimate ε, the fractional part of the delay. If the absolute value of ε rises above 0.8, the model is 'shifted' by one sampling time, the number of shifts is contained in the variable d, representing the integer part of the delay. Figure 6.23 shows that the software could keep track of the generated delay. The time it takes for the software to shift the model depends greatly on what excitation is

applied to the system at the time. Indeed if the system has reached a steady state at the time of addition of the delay, no changes will be made until a new excitation is applied. By adding small random signal to the command signal, it is possible to overcome this particular problem, without affecting the overall response.

4) **Self tuning PID values** The values of parameters found for the model are then used to calculate PID controller coefficients. The method used is that of Haalman [31][10] Those values are then used at the next sampling time to calculate an appropriate output. This gives a good response when the estimator has locked to correct values. However the response is erratic when the estimator is still trying to find the right model, as for example at time 650.(See Figure 6.22)

[10] **Haalman tuning rule:** the paper gives a tuning rule for delayed processes for both the PI and PID controller. The rule is obtained by trial and error on a computer simulation. The performance of the controller is measured by calculating the least mean square value of the error, in the response to a step disturbance. This gives a relatively simple rules for tuning. In our particular case of a first order delayed system:

$$G(s)=\frac{e^{-sT}}{1+s\tau}$$

The following PI controller is recommended :

$$H = K\left(1+\frac{1}{sTi}\right)$$

$$with\ K = \frac{2\tau_2}{3T}\ and\ Ti = \tau_2$$

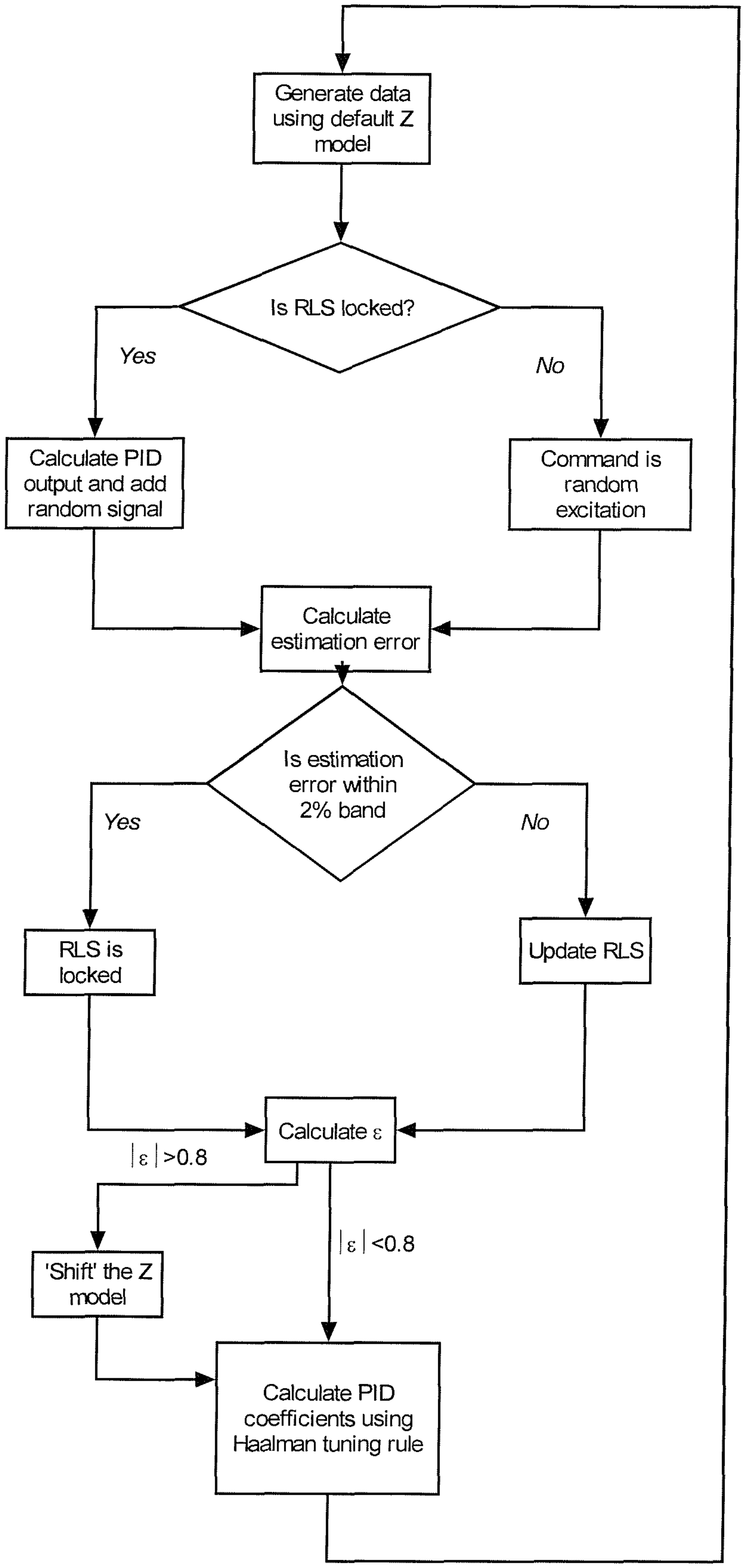


Figure 6.21 Structure of the simulation software for the self-tuning controller

The result of the simulation include representations of :

1. The PID response (Figure 6.22) : this shows the simulated response of the closed loop system (dotted line)against the demand (solid line).
2. The delay estimate (Figure 6.23) : this shows that the delay estimation (d) has kept a good track of the system delay, which was artificially added at time step 250 and removed at time step 650. The artificial delay was created by shifting the reference model by one sample time; in the code the value of the reference matrix is shifted from the non-delayed value: `[-α1 b2 b1 b0 0]` to the delayed value: `[-α1 0 b2 b1 0]`. The value of 'd' is the entire part of the estimated delay, and when the estimation of the partial part of the delay ε becomes close to unity, the entire part of the delay is shifted. Therefore the estimated change from 1 to 2 is correct Figure 6.23. The time unit used (time steps) related to the number of time the simulation software has looped, that is a new time step would represent a new sample in practice.

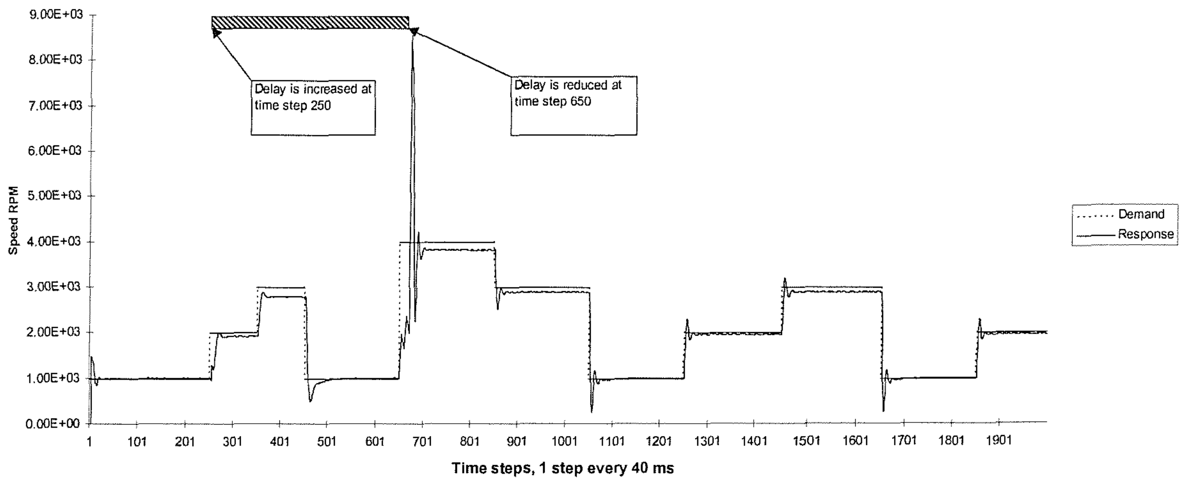


Figure 6.22 Simulation result- PID response, using the self-tuning response

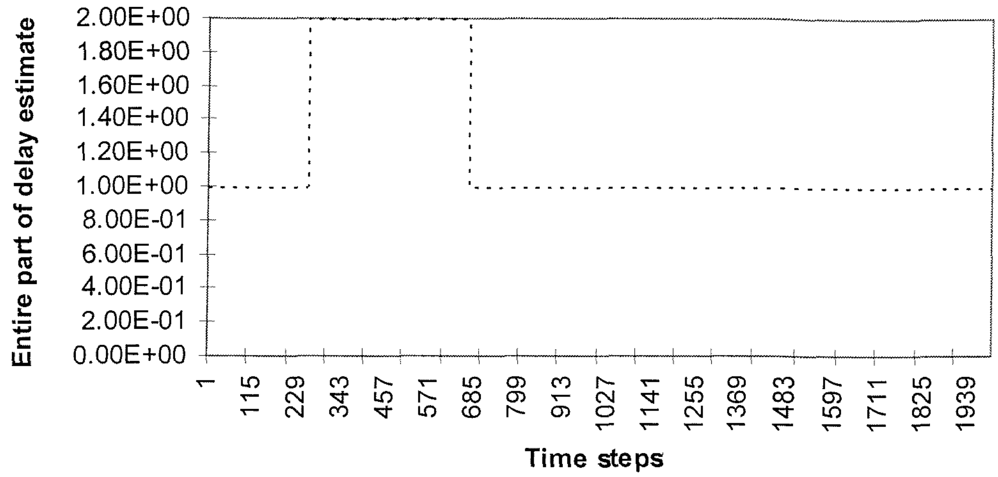


Figure 6.23 Simulation result - Delay estimate

6.5.4 Experiments

Once the simulation showed that it was possible to use a self-tuning controller, an experimental test was implemented.

This was similar in structure to the simulation software (Figure 6.24). Major differences are:

- Data is not generated by the software itself but acquired on the network from node 1.
- The estimation error band has been increased to 5%, this is to cope with the fact that the readings are digitised by the ADC and in the presence of noise.
- an additional delay can be added within the loop to simulate the addition and removal of nodes on the network
- the sampling time for the control process is the time period in between each network access loop, obviously this will vary with the added delay. The local sampling time used by the test node is very short, as the node is constantly activating the converters, therefore when the a new value is transmitted on the network, it is the result from a very recent reading.

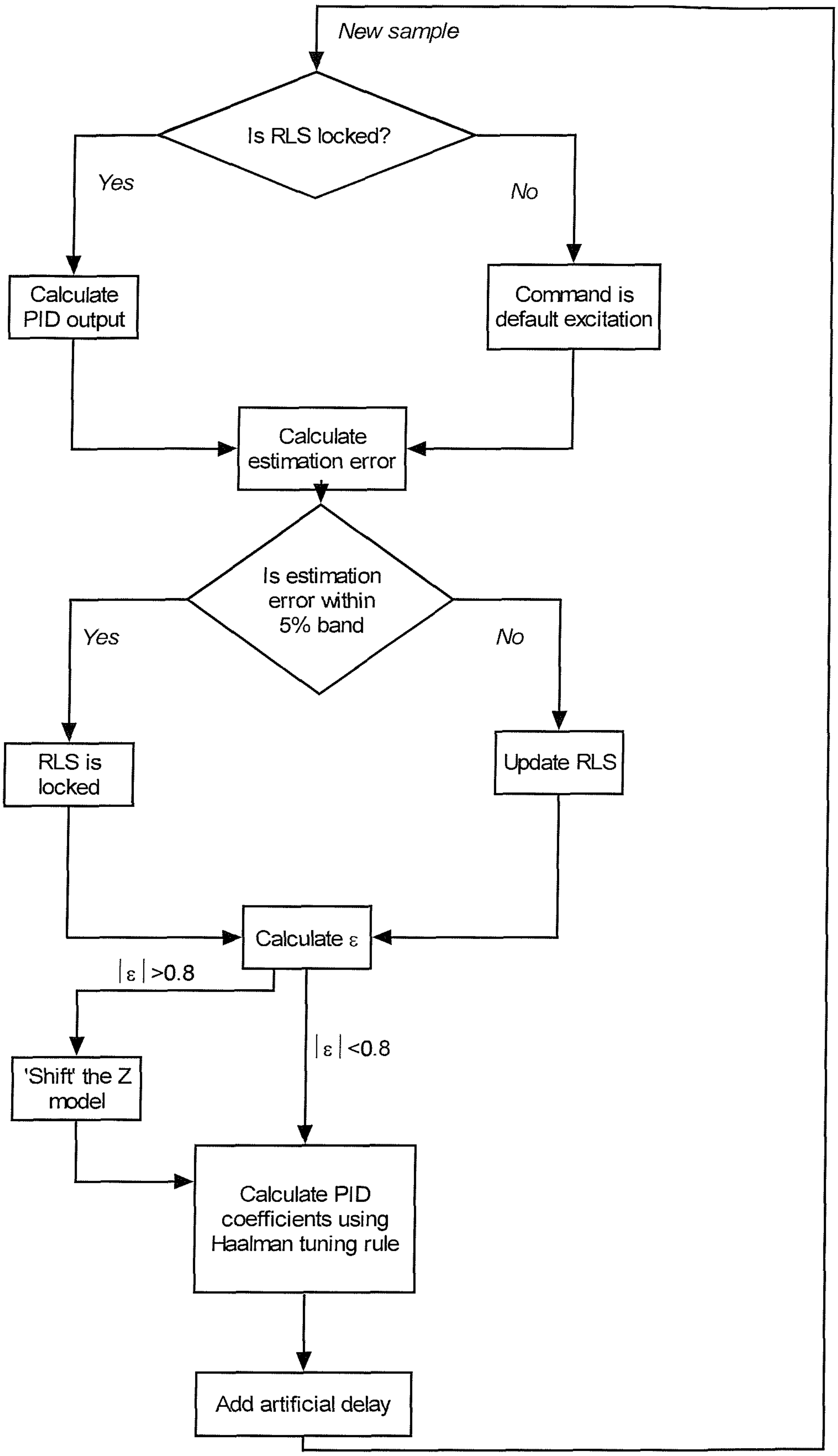


Figure 6.24 Structure of the experimental software

The setup used was the same as with the first experimental work, with the PC implementing the self-tuning PID controller, and the test node acting as an interface to the laboratory test rig (Figure 6.25).

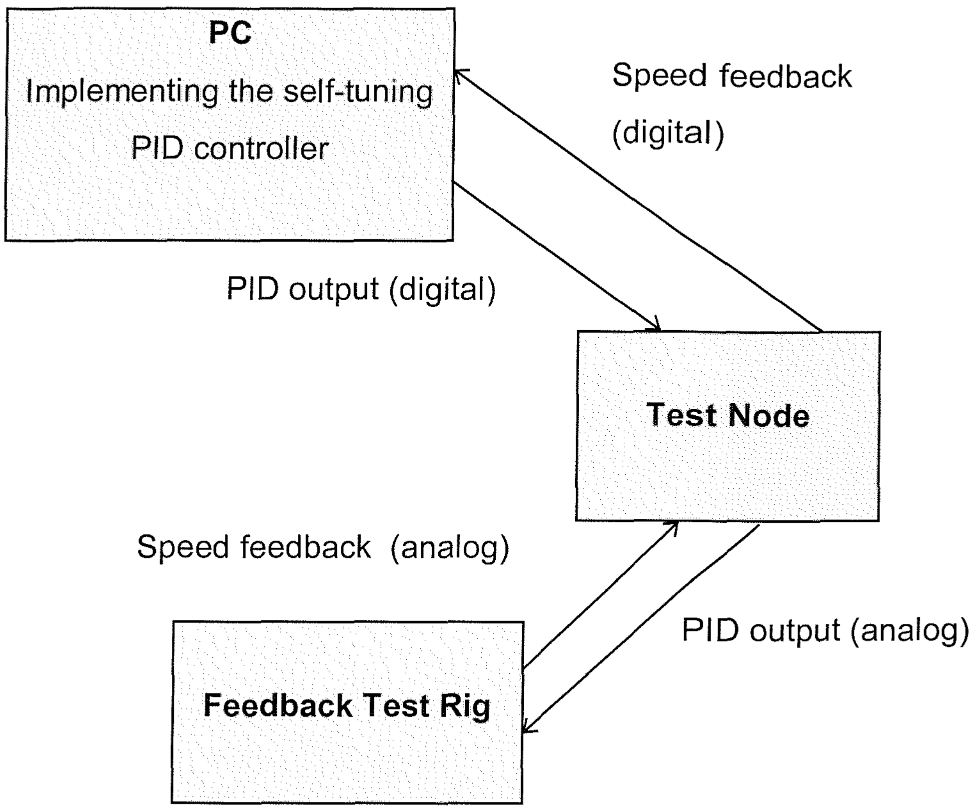


Figure 6.25 Experimental setup for self-tuning controller

Results were taken with various delays, the speed setpoint was 1500 RPM. The following graphs show both the estimated and measured speed, as logged by the PC. The estimation error shows the accuracy of the RLS estimator. Since the PID controller is only active once the RLS has converged, responses will not only reflect the choice of PID coefficients, but also the convergence of the RLS algorithm. The sharp changes at the start of the estimation, which can be seen in Figure 6.26 to Figure 6.29, reflects the fact that the RLS has not converged at that point yet, and the output estimate is using the initial model parameter values. On Figure 6.26, the changes at time step 60 happen when the RLS estimation goes outside the estimation error band,(5%), and the RLS estimation process has to be restarted. One possible estimation for the bad convergence of the RLS could be an outside event that caused the motor to alter its response. The source code of the software can be found in Appendices F and G.

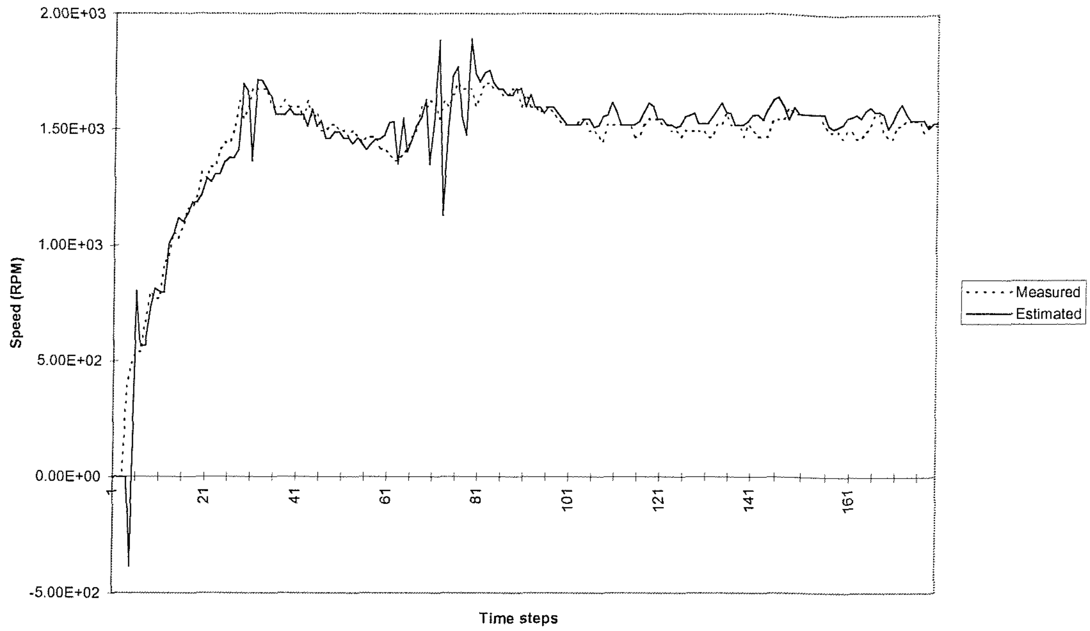


Figure 6.26 PID Self tuning control with no added delay

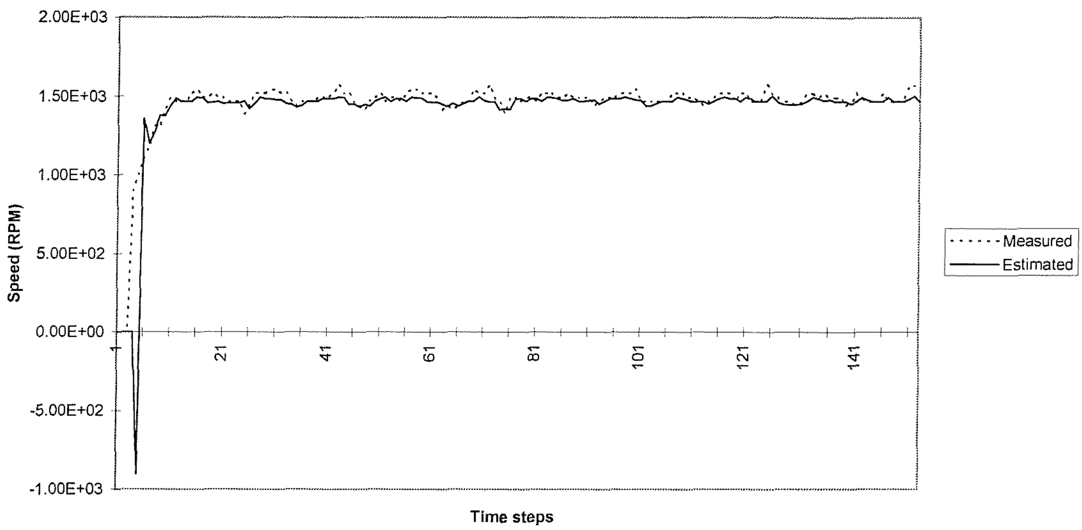


Figure 6.27 PID self tuning control with 50 ms added delay

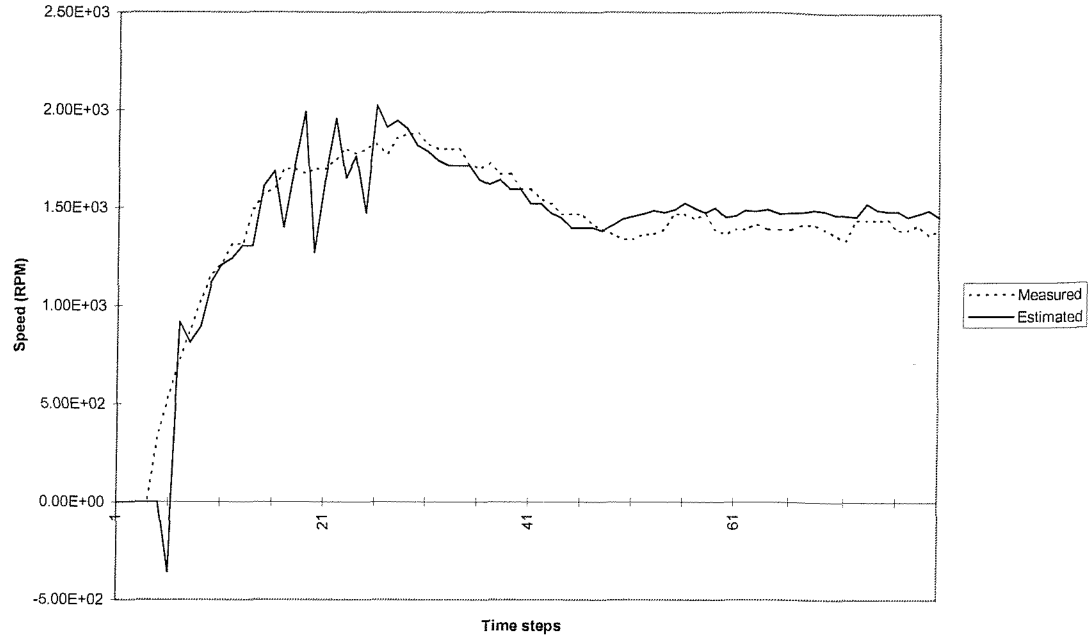


Figure 6.28 PID Self tuning control with 100 ms added delay

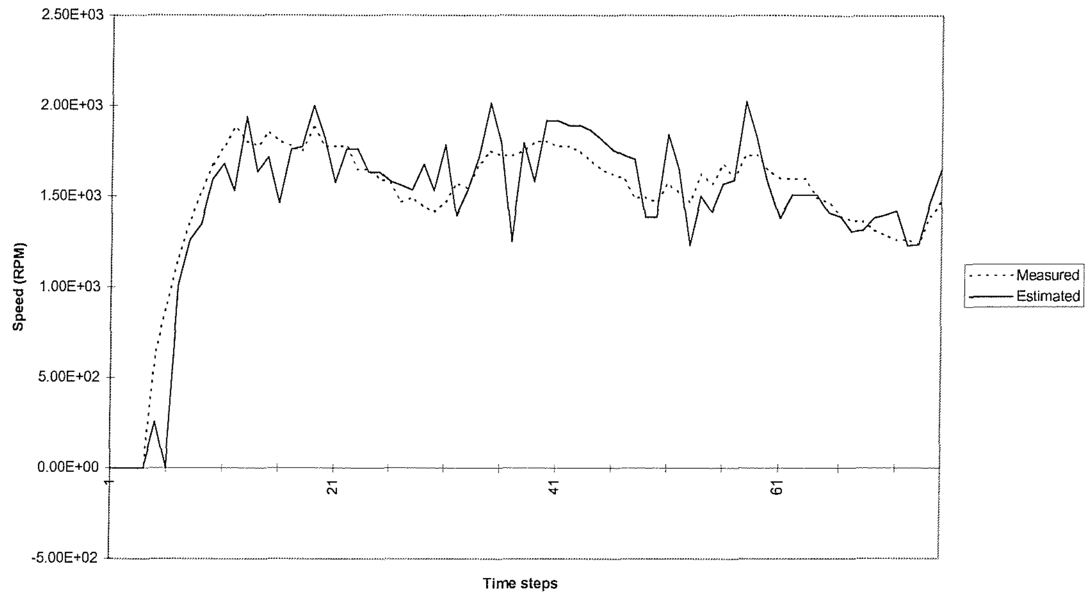


Figure 6.29 PID Self tuning control with 200 ms added delay

6.5.5 Advantages and limitation of the method

The method of self-tuning control was shown to work on a computer simulation, where a known model was estimated and controlled, after a certain time, a delay was introduced, and then removed in the model and both changes were successfully detected by the RLS estimation. The PID controller parameters were also adjusted in a suitable manner (Figure 6.22 and Figure 6.23).

In the experiments, several values of delay were added artificially, and the same controller was run, with a setpoint of 1500 rpm. In most cases a satisfactory steady state response was obtained. However the transient behaviour is often quite poor, and in the worst case the overall response is unsatisfactory, as in Figure 6.29.

The poor quality of the reponse has later been analysed as a consequence of the following fact; the PID controller is only activated once the RLS estimation has converged to a set of estimated model parameters, and due to the presence of noise, this can take a long time, in which the process remains uncontrolled . In the case of figure Figure 6.26, the values that were first estimated by the RLS were wrong, and the RLS estimation process had to be restarted.

A better response could be obtained by starting the process with default PID parameters value, the PID controller could then be started straight away, while the RLS estimation is being run.

7. CONCLUSION

7.1 Achievements

The approach taken at the start was to identify the problems existing with current industry ROVs, and to set new requirements for the design of a future vehicle. A case study of a commercial vehicle was undertaken. The main problem identified was the lack of flexibility of the communication system, which links the several components of the vehicle. This prevents the vehicle setup being modified at short notice, as it often is required in the industry.

The first step in the research was to select a distributed architecture to link the vehicle sub-systems, as opposed to a centralised architecture, which was used on the current vehicle. This allowed for more flexibility, and also has the potential of making maintenance and error detection quicker and easier. The next step was to review available networking techniques, and to select a suitable method, which was to be used for linking the ROV sub-systems, or nodes. A fieldbus-based network was selected, as it showed the most suitable for the application. A prototype vehicle was build, using the selected networking technique, and was later demonstrated in underwater operation. This prototype was build in stages: first only a basic 'propulsion node' was build on a bench system, and communicated to a master PC. A 'navigation' node, supporting compass and depth meter was then added, followed by a third node, which supported other components such as camera and lights control. A hand control unit was also build to ease the operation, and replaced the master PC. The PC was kept on the system, and used as a monitoring function only. When this system was shown to be operational on the bench, it was then integrated in an actual vehicle, with the help of the sponsor company, and demonstrated underwater.

This prototype has the advantage over the case study vehicle of being completely modular, as each of the sub-systems can be added or removed with minimum of disruption. It also has the potential of supporting many more instruments, indeed the prototype vehicle supported all the functionality existing on the version of the case study vehicle.

The design of the vehicle is fully documented in technical reports (references: [17][18][19][20][21]).

Alongside the prototype building tasks, the theory of the network communication was studied, and a simulation of the ROV network was created. This simulation is a useful design tool that allows to experiment with changes in various parameters. For example there might be a need to add a node to the vehicle in order to carry out a special task, in this case the simulation would allow to find out by how much the transportation delay would increase. Should the resulting increase in the delay overload the network in an unacceptable way, the simulation could then be used to find ways to improve the performance, for example what would be the beneficial effect of increasing the transmission speed or of shortening the packet length.

Another side to the problem is that the delays not only vary according to the vehicle configuration, but also whilst the network is running, The simulation gives out an average result and a confidence interval of the estimated delay.

Ideally, the vehicle should be able to be modified, for example by adding a camera control node, without having to alter any controller settings. This is a very important aspect of the practical problems faced by the oil industry. In order to achieve this, a self-tuning controller was implemented, as described in chapter 6. A simulation of such a control system was shown to cope well with delay variations (Figure 6.22). However the experimental results were less encouraging (Figure 6.28 and Figure 6.29), this could be for several reasons: a mismatch of the simulated model and the real hardware, or an inefficient RLS estimation method. One particular suggestion for improvement is to implement a default controller to be used whilst the RLS estimator is giving unstable results.

7.2 Contribution to research

The main contribution is to have designed a self-tuning PID controller, for systems where the transportation delay may vary. These delays occur in distributed control systems, such as the modular prototype ROV, and are not supported by standard control techniques.

Distributed control systems are becoming increasingly popular, with applications ranging from automated manufacturing lines, to cars and building automation. Not being dependant on a fixed transportation delay is a major issue, as in most of those applications a dynamic closed loop control is established over the network. The network makes the transportation delay subject to many variable factors: effect of noise, bandwidth, size of networks and properties of each node such as latency and transmission behaviour. A network simulation was used to estimate transportation delays for the studied ROV.

By using a RLS estimator the contribution of the delay can be estimated, then suitable PID parameters can be calculated. This type of controller is ideal for the ROV system, as the setup of the network is likely to be modified often. Only one controller can be used for any configuration of the network. Details of the control system development are described in Chapter 6.

The review of existing networking methods is also an important point, as once the distributed approach is selected, the choice of a particular networking method is a difficult one to make. Many networks exist, all with their advantages and disadvantages, the review showed that in a commercial environment this choice would be driven by the financial aspects. The network simulation was also useful to investigate the impact of some design decisions such as the choice of transmission speed.

7.3 Limitations

The limitations of such a self-tuning controller were found to be that there is a possibility that a change in the model could be wrongly identified as a change in the transportation delay. Since the vehicle is to be used in a widely changing environment, this is a possible cause for problems. Decreasing the sensitivity of the parameter updating algorithm, it might be possible to override this problem.

One other factor that caused problems in the experiments was that the RLS estimator needs to be excited in order to converge. This was solved by adding a small randomly varying signal to the command signal, and this random signal was small enough not to affect the target system, while allowing the RLS estimator to converge more easily.

7.4 Suggestions for further work

As far as the design of the prototype vehicle is concerned, some improvements could be made by adding more nodes and therefore allowing for a wider functionality. The vehicle is still a prototype, and in order to be produced commercially, would need to be more reliable. During the development the majority of the faults that occurred were due to weak connections and poor quality printed circuit boards. This is the area needing the most improvements, and where much time was spend diagnosing and repairing trivial problems..

A far as the theory is concerned, the control system could be extended to be applied to the ROV heading and depth control. This involves firstly obtaining a model of the ROV, and secondly applying the self-tuning controller. This is a much more complex system to control than the first order system studied in the laboratory, especially when the ballast, position of thrusters, instruments and environmental conditions can vary considerably between each vehicle launch.

Another important factor to model would be the amount of disturbances, as the vehicle is to be used in extreme conditions. The robustness of the controller would be a key factor.

REFERENCES

[1]*"TSS 340 - Pipe and cable tracking system"*, Product data sheet

[2]S.M.Abu Sharkh, M.R.Harris, R.M.Crowder, 1994, *"Comparative studies of electric and hydraulic drive systems for thrusters of remotely Operated Vehicles"*, Proceedings of Oceanology International 94 Exhibition & Conference, 8-11 March 1994, Brighton, UK

[3]*"Underwater Television Sensors - Selection criteria"*, Simrad Ltd., December 1993

[4]R.K.Hansen, 1993, *"An acoustic camera for 3D underwater imaging"*, Proceedings Acoustics sensing and imaging, 29-30 March 1993 Conference publication 369 IEE, pages 99-101

[5]*"Simrad HiPAP High Precision Acoustic Positioning P2448E"*, Simrad Ltd., January 1992

[6]S.D.McPhail, 1995, *"Autosub: an autonomous unmanned submersible"*, Electronics and Communication Engineering Journal, June 1995

[7]H.Masden, A.Bjerrum, B.Krogh, 1996, *"MARTIN- an AUV for Offshore Surveys"*, Oceanology International, Brighton 5-8 March 1996

[8]F.Halsall, *"Data Communications, Computer Networks and Open Systems"*, 1996, Addison-Wesley

[9]G.Bucci, *"Performance Analysis of two different algorithms for Ethernet FDDI connection"*, IEEE Transactions on Parallel and Distributed Systems, vol. 5 No. 6 June 1994, Pages 614-29

[10]Sadiku, M.Ilyas, *"Simulation of local area networks"*, 1995, CRC Press

[11]G.Wood, 1997, *"No pain, no gain"*, IEE Review, 18th September 1997, Vol. 43, No 5.

[12]J.Hollingum, 1994, *"Lonworks Solutions in Silicon"*, Assembly Automation, Vol.14, No 1, pages 25-27

[13]M.Howarth, 1994, *"Hart opens up smart solutions"*, Assembly Automation, Vol.14, No 1, pages 22-24

[14]G.Funk, 1982, *"Error detecting properties of HDLC protocols",* IEEE Transactions on communications, Vol.COM-30, No. 1, January 1982

[15]R.Matick, 1969, *"Transmission lines for digital and communication networks",* McGraw-Hill

[16]Arcom PCSER4 card manual

[17]S.M.Rolland, February 1997, *"Communication Library - ROV Master Node Telemetry Software Library"*, University of Southampton, Mechanical Engineering Report ME 97/08

[18]S.M.Rolland, July 1997, *"Prototype ROV Software Documentation - Surface PC Monitoring Software"*, University of Southampton, Mechanical Engineering Report ME 97/12

[19]S.M.Rolland, February 1997, *"Communication Library - ROV Slave Nodes Telemetry Software Library"*, University of Southampton, Mechanical Engineering Report ME 97/09

[20]S.M.Rolland, July 1997, *"Prototype ROV Software Documentation - Subsea Nodes Application Software"*, University of Southampton, Mechanical Engineering Report ME 97/11

[21]S.M.Rolland, July 1997, *"Prototype ROV Software Documentation - Hand Control Unit Software"*, University of Southampton, Mechanical Engineering Report ME 97/10

[22]Brüel & Kjær, Instruction Manual for Noise Generator type 1405

[23]M.N.O.Sadiku, M.Ilyas, 1995, *"Simulation of local area networks",* CRC press

[24]M.W.Atkinson, 1984, *"Network simulation using SIMSCRIPT",* IEEE Global Telecommunications Conference, GLOBECOM '84: Communications in the Information Age, 26-29 Nov 1984

[25]D.J.Baker, J.P.Hauser, 1985, *"Event-process facility built on Simula: a tool for simplifying the simulation of distributed control systems",* Proceedings of the Summer Computer Simulation Conference, Chicago, 22-24 July 1985

[26]J.A.Pence, 1986, *"Using SIMAN to model an Ethernet local area network"*, Modelling and Simulation on Microcomputers: 1986, Proceedings if the conference, SanDiego, CA, 13-25 Jan 1986, pages 110-114

[27]Feedback, Modular Servo System MS150 MkII, Book1 (Basic Experiments) and Book3 (Advanced Experiments), 1984

[28]K.J.Åström and B.Wittenmark, 1984, *"Computer controlled systems - Theory and design",* Prentice-Hall

[29]K.J.Åström and B.Wittenmark, 1989, *"Adaptive control"*, Addison-Wesley

[30]A.Leva, C.Maffezzoni, R.Scattolini, 1994, "Self-tuning PI-PID regulators for stable systems with varying delay", Automatica, Vol.30, No.7, pages 1171-1183

[31]A.Haalman, 1965, *"Adjusting controllers for deadtime processes"*, Control Engineering, Vol.12, pages 71-75

[32]J.D.Decotignie, D.Auslander, 1996, *"Integrated Communication and Control Systems with Fiedlbusses"*, Japan/USA Symposium on Flexible Automation, Volume1, pages 517-520, ASME 1996

[33]S.Hutchings,S.L.Merry,S.Rolland,R.Allen, 1996, *"Communication and control systems for remotely operated underwater vehicles"*, Oceanology International 96 Brighton UK, Volume1, pages 235-244, ISBN 0900254114

[34]J. Yuh, *Modeling and control of underwater robotic vehicles,* IEEE transactions on systems, man and cybernetics, November/December 1990 volume 20 no.6, pages 1475-1483

[35]D.R.Yoerger, J.B. Newman, J.J.E. Slotine, *Supervisory Control System for the Jason ROV,* IEEE Journal of Oceanic Engineering, 1986, volume 11 no.3 July 1986, pages 392-399

[36]K.Goheen, *The hardware and software development of a fully adaptative ROV autopilot,* Proceedings ROV 1986 Aberdeen UK, Aberdeen UK 1986, pages 235-258

[37]K.Goheen, E.R.Jeffreys, *Multivariable Self-Tuning Autopilots for Autonomous Underwater Vehicles,* IEEE Journal of Oceanic Engineering, volume 15 no.3 July 1990

[38]H.N.Farnbrother,B.A.Stacey, R.Sutton, *Fuzzy self-organizing control of a remotely operated submersible,* IEE International Conference on Control 1991, pages 499+

Bibliography

Abu Sharkh S.M.;Harris M.R; Crowder R.M., *Comparative studies of electric and hydraulic drive systems for thrusters of Remotely Operated Vehicles,* Proceedings of Oceanology International 94 Exhibition & Conference, 8-11 March 1994 Brighton UK

Allen B., *A low cost local area network for robotic cell communications,* CADCAM '86 Conference, 1986, 8-10 April 1986

Allen C., *The Interoperable Systems Project (ISP),* Measurement and Control, 1994, vol. 27 March 1994, Pages : 38-41

Applied Acoustics Engineering, *4320 Series,* Mutibeacon 4320 Series

Atkinson J.K., *An explanation of the IEC Fieldbus Standard*

Atkinson J.K., *Digital Signal Transmission Standards*

Ayyagari A.; Ray A., *A fiber optic Network protocol for Computer Integrated Manufacturing,* Journal of Engineering for Industry, 1992, vol. 114 August 1992, Pages : 345-351

Bucci G., *Performance Analysis of two different algorithms for Ethernet FDDI connection,* IEEE Transactions on Parallel and Distributed Systems, vol. 5 nb 6 June 1994, Pages : 614-29

Bylanski P.;Ingram D.G.W., *Digital Transmission Systems,* ISBN 0-966048-42-7

Cesbron N.; Alais P. ;Ollivier F.; Challande P., *A 3-D Underwater Acoustics Camera,* Underwater Systems Design, March/April 1994, Pages : 21-23

Chandler K.E., *Underwater video imaging and ROVs,* Unmanned systems, Spring 1991, Pages : 35-38

Comer D., *Internetworking with TCP/IP Principles Protocols and Architecture,* 1988, ISBN 0-13-470154-2-025

Desjardins M., *WorldFIP,* Measurement and Control, vol. 27 March 1994, Pages : 42-46

Det Nordske Veritas, *Monitoring of cathodic protection systems,* Technical Note Fixed Offshore Installations, 1984,

Dobosiewicz W. Gburzynski P., *An alternative o FDDI: DPMA and the Pretzel Ring,* IEEE Transactions on Communications, vol. 42 nb 2-4 pt. 2 Feb.-April 1994, Pages : 1076-83

Edgar G.K.;Pope J.C.D.,Craig, I.R. *Visual accommodation problems with head up and helmet mounted displays,* Displays, 1994, vol 15 nb 2, Pages : 68-75

Etchemendy S.; Davis D., *Designing ROV for Oceanographic Research,* Sea Technology, February 1991, Pages : 21-24

Flatman A., *Wireless LANs development in technology and standards,* Computing and Control Engineering Journal, October 1994, Pages : 219-224

Gawthrop R.J., *Continuous Time Self-tuning Control -Volume1,* ISBN 0-86380-049-1

GEE K.C.E, *Local Area Networks,* ISBN 0-85012-365-8

Gelb A., *Applied Optimal estimation,* 1974, ISBN 0-262-20027-9

Given Deam, ROV design concepts for the 1990's, Sea Technology, 1991.00, August 1991, Pages : 57-59

Grimble M.J., *Use of Kalman filtering techniques in dynamic ship positioning systems,* IEE proceedings, vol 127 part D nb 3 May 1980

Hansen R.K., *An acoustic camera for 3D underwater imaging,* Proceedings Acoustics Sensing and Imaging, 29-30 March 1993 Conference publication 369 IEE, Pages : 99-101

Harris C.J.; Moore C.G.; Brown M., *Intelligent Control Aspects of Fuzzy logic and neural nets,* World Scientific, 1993, ISBN 981-02-1042-6

Hodgkinson G., *A single world standard forget it,* Assembly Automation, 1994, vol 14 nb 1 1994, Pages : 3

Hollingum J, *Lonworks Solutions in Silicon,* Assembly Automation, vol 14 nb 1 1994, Pages : 25-27

Hopper, *Local Area Networks Design,* ISBN 0-201-13797-6

Howarth M., *HART opens up smart options,* Assembly Automation, vol 14 nb 1 1994, Pages : 22-24

Howarth M., *HART Standard for 4-10 mA digital communications,* Measurement and Control, vol 27 February 1994, Pages : 5-7

James S., *New devices for VCR using teletext services,* IEEE Transactions on Consumer Electronics, vol 38 nb 3 August 1992, Pages : 288-295

Khoh S.B.; McLaughlin R.T., *Autos carry the CAN,* Assembly Automation, vol 14 nb 1 1994, Pages : 17-19

Liebnitz-Lann, *207 LR,* Liebnitz Lann Video Overlay 207 LR

Lockyer G., *Profibus A user's view,* Assembly Automation, vol 14 nb 1 1994, Pages : 30-39

Loose G., *Fieldbus the user's perspective,* Measurement and Control, vol 27 March 1994, Pages : 47-51

Macwilliams P.D., *Serial Bus simplifies distributed control,* Control Engineering, June 1984, Pages : 101-104

Malcolm N. Zhao W., *The timed token protocol for Real Time Communications,* Computer, 1, vol. 27 nb 1 Jan 1994, Pages : 35-41

Marine Electronics, *ROV Navigation,* Marine Electronics Brochure,

MDL Engineering, *Trim Cube Data sheet,* TRIM Cube, 1993, Mellingh E.C., *Instrument Bus: an electronic system..,* Woods Hole Oceanographic Institute, 1986, WHOI-86-30

Mindell D.A., *Images from the deep,* Byte, 1990, June 1990, Pages : 256-260

Moore C.G., *Ph.D. Thesis Indirect adaptative fuzzy controllers,* 1992,

Mosley J.D., *The best LAN may be found off the MAP,* EDN, 1991.00, November 7 1991, Pages : 63-73

Navitronics, *SVP-1 / SVM-1,* Data Sheet,

Newman J.B., *Fiber Optic Data Network for the Argo/Jason vehicle system,* IEEE Journal of Oceanic Engineering, April 1990, Pages : 66-71

Newman J.B.; Robinson B.H., *Development of a dedicated ROV for Ocean Science,* MTS Journal, vol 26 nb 4, Pages : 46-53

Nomoto N.; Hattori M., *A deep ROV 'Dolphin 3K' Design and Performance Analysis,* IEEE Journal of Oceanic Engineering, vol 11 nb 3 July 1986, Pages : 373-391

Osprey, *Brochure OE2300A,* OE2300A SIT,

Panasonic, *AG IA232 TC and AG 5700 RS 232,* Product Information Panasonic AG IA 232 TC,

Photosea, *Camera Catalogue,* Photosea Catalogue,

Photosea, *Cobra,* Cobra Series Data sheet,

Rigaud V; Marce L., *Absolute location of underwater robotic vehicles by acoustic data fusion,* Proceedings IEEE Int. Control 1990, 1990, 13-18 May Cincinnati OH USA v12 ISBN 081-8690615, Pages : 1310-1315

ROV Review 5th edition, Wave Publication, ISBN 0-9623145-3-6

Rumelhart D.E.; Widrow B.; Lehr M.A., *The Basic Ideas in Neural Networks,* Communications of the ACM, 1994, March 1994 vol 37 nb 3, Pages : 87-92

Ryther J.H.; Harris D.B.; .Perry Fish J., *Putting ROVs to work investigating shipwrecks,* Sea Technology, May 1990, Pages : 43-53

Seatex, *Preliminary Technical Specifications,* Motion Reference Unit MRU 6

Seatex, Seatex Spotrange

Senior J., *Optical Fiber Communications,* ISBN 0-13-638248-7

Simrad, *Simrad HPR 300P,* Simrad Product Information

Simrad, *Technical Specification,* OE1386/OE1397

Sohn Y., *Networking with the 8044,* Digital Design, vol 17 March 1984, Pages : 136-138

Spalding M.; Dawson B., *Finding the Titanic,* Byte, March 1986, Pages : 97-110

Squirrell B., *Profibus : a working standard fieldbus,* Measurement and Control, 1994, vol 27 February 1994, Pages : 9-12

Stallings W., *Handbook of computer communications standards Local area Network Standards volume 2,* 1988

Steward D., *Underwater TV Cameras Approximate Viewing Ranges,* Underwater system Design, 1994, March/April 1994, Pages : 17-19

Suspek, *ROVProbe System Overview,* Subspek fax copy

T.E.Smith, *ROVs and Science: Riding the learning curve,* Sea Technology, August 1992, Pages : 47-51

Tarrant D.R., *A new teletext decoder with advanced OSD features for wide screen TV,* IEEE Transactions on Consumer Electronics, 1993, vol 39 nb 3 August 1993, Pages : 166-174

Tetley L.; Calcutt D., *Electronics Aids to Navigation,* ISBN 0-7131-3548-4

Tetlow S., *Development of a Hybrid Laser Viewing System,* Underwater system Design, January/February 1994, Pages : 17-19

Theunissen E., *Factors influencing the design of perspective flight path displays for guidance and navigation,* Displays, 1994, vol 15 nb 4, Pages : 241-253

TSS, *Operating Manual,* TSS 335B Operating Manual

TSS, *TSS 330 Data sheet,* TSS 330 Motion Sensor

TSS, *TSS330 Series,* TSS Brochure

TSS, *TSS330,* Seasense Brochure

TSS, *TSS340,* TSS 340 Brochure

Valeport, *VEM003,* Valeport VEM003 Data Sheet

Williams S.J.; Marshfield W.B., *A switched Hsup strategy for Control of a submarine,* IEE Conference on Control '91, 1991, Pages : 487-492

Yuh J. Lakshmi R., *An intelligent Control System for ROV,* IEEE Journal of Oceanic Engineering, 1993.00, vol. 18 nb 1 Jan 1993, Pages : 55-61

Zadeh L.A., *Fuzzy Logic, Neural Networks and Soft Computing,* Communications of the ACM, March 1994 vol 37 nb 3, Pages : 77-84

Appendix Index

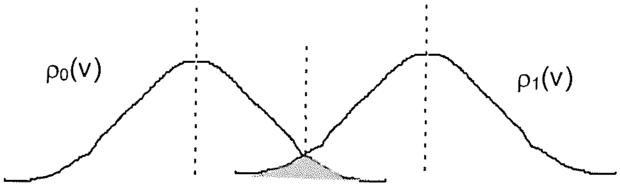
A. Noise variance and Bit Error Rate
B. Transmission line theory
C. Network simulation software (source code listing)
D. Random number generation
E. Simulation Self-tuning software (source code listing)
F. Matrix library (source code listing)
G. Experimental Estimation software (source code listing)
H. RLS estimation
I. Probability of spurious flag

Appendix A Noise variance and Bit Error Rate

Supposing that the noise has a Gaussian probability distribution with zero mean :

$$\rho(V) = \frac{e^{-\frac{v^2}{2\sigma^2}}}{\sqrt{2\pi\sigma}}$$

Consider a logic '0' being represented by A_0 and a logic '1' by A_1 .



When no noise is present , a logical '0' is represented by a voltage V set to A_0, and a logical '1' is represented by a voltage V set to A_1. As noise contributes to the signal, it is possible that the voltage value changes and crosses the threshold level d, this causes a transmission error.

As the noise is added, there will be a probability distribution about A_0 and A_1. An error occurs when V<d when a '1' was transmitted (P_{e1})and when V>d when a '0' was transmitted(P_{e0}).

$$P_{e0} = \rho_0(V > d) = \int_d^{\infty} \rho_0(V)dV \text{ error on a '0' transmission}$$

$$P_{e1} = \rho_1(V < d) = \int_{-\infty}^{-d} \rho_1(V)dV \text{ error on a '1' transmission}$$

Due to the symmetry in the distributions, we have $P_{e0} = P_{e1}$. The average probability of error is :

$P_e = P_1 P_{e1} + P_0 P_{e0}$. , where P_1 is the probability of a '1' being transmitted, and where P_0 is the probability of a '0' being transmitted. Considering a probability of occurrence of '1' and '0' as in HDLC of P_0 = 32/63 and P_1=31/63, (there is a higher probability of a '0' being transmitted due to the bit stuffing process) we have $P_e = P_{e1}[P_0 + P_1] = P_{e0}[P_0 + P_1] = P_{e0} = P_{e1}$

Using Gaussian statistics :

$$\mathrm{P}_e = \mathrm{P}_{e0} = \int_d^\infty \mathrm{P}_0(V)dV$$

$$\mathrm{P}_e = \int_d^\infty \frac{e^{-\frac{(V-A_0)^2}{2\sigma^2}}}{\sqrt{2\Pi}\sigma}dV$$

$$\text{Let } x = \frac{V - A_0}{\sqrt{2}\sigma} \text{ then } dx = \frac{dV}{\sqrt{2}\sigma}$$

$$\mathrm{P}_e = \frac{1}{\sqrt{\Pi}}\int_{\frac{d-A_0}{\sqrt{2}\sigma}}^\infty e^{-x^2}dx = \frac{1}{2}cerf\left(\frac{d-A_0}{\sqrt{2}\sigma}\right)$$

$$\text{as } d = \frac{A_1 + A_0}{2}$$

$$\text{then } \mathrm{P}_e = \frac{1}{2}cerf\left(\frac{A_1 - A_0}{\sqrt{2}\sigma}\right)$$

cerf is the complimentary error function defined as:

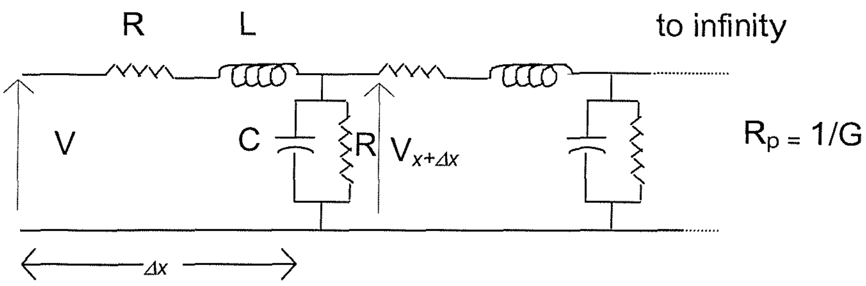
$$\mathrm{cerf(u)} = \frac{2}{\sqrt{\Pi}}\int_u^\infty e^{-x^2}dx \text{ approximated to:}$$

$$\mathrm{cerf(u)} \approx \frac{\mathrm{e}^{-\mathrm{u}^2}}{u\sqrt{\Pi}} \text{ for u} > 3$$

This definition of P_e is very important, since it links the BER (Bit Error Rate) to the thermal noise variance.

Appendix B Transmission line theory

From Matick [8] a transmission line can be modelled as a succession of the following circuit:



The basic effects occurring along a line are phase shift and attenuation.

The analysis of a small length of the circuit (Δx) gives :

$$V_x = i_x R\Delta x + i_x j\omega L\Delta x + V_{x+\Delta x}$$

$$V_{x+\Delta x} - V_x = -i_x(R + j\omega L)\Delta x$$

$$i_{x+\Delta x} - i_x = \frac{V_{x+\Delta x}}{R_p\Delta x} + \frac{V_{x+\Delta x}}{1/jC\Delta x\omega}$$

$$i_{x+\Delta x} - i_x = -V_{x+\Delta x}(G + j\omega C)\Delta x$$

$$\frac{\Delta V_x}{\Delta x} = -i_x(R + j\omega L)$$

$$\frac{\Delta i_x}{\Delta x} = -V_{x+\Delta x}(G + j\omega C)$$

when $\Delta x \to 0$ this becomes

$$\frac{dV_x}{dx} = -(R + j\omega L)i_x$$

$$\frac{di_x}{dx} = -(G + j\omega C)V_x$$

$$\frac{d^2V_x}{dx^2} = (R + j\omega L)(G + j\omega C)V_x = \gamma^2 V_x$$

a solution to that differential equation is :

$$V_x = V_A e^{-\gamma x} + V_B e^{\gamma x}$$

and for i_x

$$\frac{d^2 i_x}{dx^2} = (R + j\omega L)(G + j\omega C)i_x = \gamma^2 i_x$$

a solution to that differential equation is :

$$i_x = i_A e^{-\gamma x} + i_B e^{\gamma x}$$

$$\frac{dV_x}{dx} = -(R + j\omega L)i_x = -\gamma V_A e^{-\gamma x} + \gamma V_B e^{\gamma x}$$

$$\Rightarrow i_x = \frac{\gamma}{(R+j\omega L)}(V_A e^{-\gamma x} - V_B e^{\gamma x}) = \frac{1}{\sqrt{Z/Y}}(V_A e^{-\gamma x} - V_B e^{\gamma x})$$

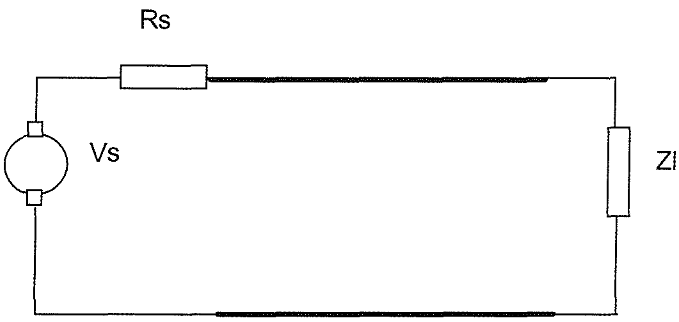
$$with\ \sqrt{\frac{Z}{Y}} = \frac{(R+j\omega L)}{\gamma} = \sqrt{\frac{(R+j\omega L)}{(G+j\omega C)}}\ so\ that:$$

$$i_A = \frac{V_A}{\sqrt{Z/Y}}\ and\ i_B = \frac{-V_B}{\sqrt{Z/Y}}$$

using sinusoial excitation :

$$V_x = e^{j\omega t}(V_A e^{-\gamma x} + V_B e^{\gamma x})$$

$$i_x = \frac{e^{j\omega t}}{\sqrt{Z/Y}}(V_A e^{-\gamma x} - V_B e^{\gamma x})$$



at x = 0 $V_x = V_s - V_{IR}$

$$V_A + V_B = V_s - V_{IR} = V_s - iR_s$$

at x = λ

$$\frac{V_X}{i_X} = Z_\lambda = Z_0 \frac{(V_A e^{-\lambda\gamma} + V_B e^{\lambda\gamma})}{(V_A e^{-\lambda\gamma} - V_B e^{\lambda\gamma})}\ \text{with}\ Z_0 = \sqrt{\frac{Z}{Y}}$$

if $R_s = Z_0$ then : $V_A + V_B = V_s - (i_A - i_B)Z_0 \Rightarrow V_A = \frac{Vs}{2}$

$$Z_\lambda = Z_0 \frac{\left(\frac{V_S}{2}e^{-\lambda\gamma} + V_B e^{\lambda\gamma}\right)}{\left(\frac{V_S}{2}e^{-\lambda\gamma} - V_B e^{\lambda\gamma}\right)}$$

$$Z_\lambda e^{-\gamma\lambda}\frac{V_s}{2} - Z_\lambda e^{\gamma\lambda}V_B = Z_0\frac{V_s}{2}e^{-\gamma\lambda} + Z_0 V_{|B}e^{\gamma\lambda}$$

$$\frac{V_s}{2}\left(Z_\lambda e^{-\gamma\lambda} - Z_0 e^{-\gamma\lambda}\right) = V_B\left(Z_0 e^{\gamma\lambda} + Z_\lambda e^{\gamma\lambda}\right)$$

$$V_B = \frac{V_s}{2}e^{-2\gamma\lambda}\frac{(Z_\lambda - Z_0)}{(Z_0 + Z_\lambda)}$$

e_γ^{x} represents waves travelling in the negative x directions end $e^{-}_{\gamma}{}^{x}$ waves travelling in the positive direction. This refection has a distortion effect on pulses. The propagation constant γ is a complex, its real part α is the attenuation constant, and its imaginary part β is the phase constant.

$$\gamma^2 = (R + j\omega L)(G + j\omega C)$$

$$\gamma = \left(RG - \omega^2 LC + j\omega(GL + RC)\right)^{\frac{1}{2}}$$

$$\gamma = \left(\omega^2 LC\left(\frac{RG}{\omega^2 LC} - 1 + \frac{j\omega(GL + RC)}{\omega^2 LC}\right)\right)^{\frac{1}{2}}$$

$$\gamma = \left(-\omega^2 LC\right)^{\frac{1}{2}}\left(1 - \frac{RG}{\omega^2 LC} + j\omega\left(\frac{G}{\omega^2 C} + \frac{R}{\omega^2 L}\right)\right)^{\frac{1}{2}}$$

$$\gamma \approx \frac{R}{2\sqrt{L/C}} + \frac{G}{2}\sqrt{\frac{L}{C}} + j\omega\sqrt{LC}\left(1 + \frac{R}{8\omega^2 L^2} + \frac{G}{8\omega^2 C^2} - \frac{RG}{4\omega^2 LC}\right)$$

$$\alpha = \frac{R}{2\sqrt{L/C}} + \frac{G}{2}\sqrt{\frac{L}{C}} \text{ attenuation in rad / meter}$$

$$\beta = \omega\sqrt{LC}\left(1 + \frac{R}{8\omega^2 L^2} + \frac{G}{8\omega^2 C^2} - \frac{RG}{4\omega^2 LC}\right) \text{ phase constant rad / meter}$$

The above calculation does not take into account the skin effect. Matick shows that when including the skin effect the attenuation becomes :

$$\alpha = \frac{R_{sk}}{2\sqrt{L/C}}\sqrt{\frac{\omega}{2}} \text{ with no shunt loss}$$

and the phase constant is : $\beta = \omega\sqrt{LC} + \frac{R_{sk}}{2\sqrt{LC}}\sqrt{\frac{\omega}{2}}$

This means that high frequency signals are attenuated more than low frequency ones and this is the main cause of distortion of pulses.

APPENDIX C Network Simulation Software


```
/*******************************************************************
* File :random.h                                                   *
* Description : random number genaration library header file       *
* Adapted from "Numerical Recipes in C - The art of          *
* scientific computing", Press, Flannery, Teukolsky Vetterling,    *
* Cambridge University Press, ISBN 0-521-35465-X                   *
* HISTORY: Date          Author        Comments                    *
*          ----------------------------------------------------- *
*          20/09/96     S.M.Rolland  Creation                      *
*******************************************************************/
#include <stdlib.h>
#include <math.h>

/*_________________________________________________________________
* Procedur: Ran0                                                   *
* Input : idum - negative for initialisation                       *
* Output : random number                                           *
* Comments : uses the standard rand() but reshuffled               *
* HISTORY: Date          Author        Comments                    *
*          ----------------------------------------------------- *
*          20.09.96     S.M.Rolland  Creation                      *
__________________________________________________________________*/
float ran0(int *idum);
```

APPENDIX C Network Simulation Software


```
/*******************************************************************
* File :random.c                                                   *
* Description : random number genaration library                   *
*   Mostly taken from "Numerical Recipes in C - The art of         *
* scientific computing", Press, Flannery, Teukolsky Vetterling,    *
* Cambridge University Press, ISBN 0-521-35465-X                   *
* HISTORY: Date        Author       Comments                       *
*          --------------------------------------------------- *
*          20/09/96    S.M.Rolland  Creation                       *
*******************************************************************/
#include "random.hpp"
/*_________________________________________________________________
* Procedur: Ran0                                                   *
* Input : idum - negative for initialisation                       *
* Output : random number                                           *
* Comments : uses the standard rand() but reshuffled               *
* HISTORY: Date        Author       Comments                       *
*          --------------------------------------------------- *
*          20.09.96    S.M.Rolland  Creation                       *
__________________________________________________________________*/
float ran0(int* idum)
{
static float y,maxran,v[98];
float dum;
static int iff=0;
int j;
void nerror();

if (*idum <0 || iff==0)
    {
    iff=1;
    maxran=RAND_MAX +1.0;
    srand(* idum);
    *idum=1;
    for (j=1;j<=97;j++) dum=rand();
    for (j=1;j<=97;j++) v[j]=rand();
    y=rand();
    }
j=1+97.0 *y/maxran;
/*if (j >97 || j<1) nerror("RAN0: This cannot happen");*/
y=v[j];
v[j]=rand();
return y/maxran;
}
```

APPENDIX C Network Simulation Software


```
/****************************************************************
* File :simu.hpp                                                *
* Description : OO network simulation  project                  *
* HISTORY: Date         Author       Comments                   *
*          ---------------------------------------------------- *
*          19/06/96     S.M.Rolland  Creation                   *
****************************************************************/
#include <math.h>
#include <stdlib.h>
#include <iostream.h>
#include <conio.h>
#include "random.hpp"
#define  MAX_Q_SIZE 1
#define  MAX_PACKETS  10000
#define  RATE  10500.0
#define  FACTOR  1000.0
#define MEDIUM_LENGTH 500 /* 500 meters*/
#define MAX_NODES 32
// Mode of output used in list nodes

#define VERBOSE 1
#define TABULAR 2
#define TRUE 0
#define FALSE 1
extern float T_dist_par[10];

/*______________________________________________________________
* Class definition : Node                                       *
* HISTORY: Date         Author       Comments                   *
*          ---------------------------------------------------- *
*          19.06.96     S.M.Rolland  Creation                   *
_______________________________________________________________*/
class Node{
friend class Net;
     int queue_size; // queue size at station
     int hp_queue_size; // high priority queue size
     int next_stn; // identifies next station
     int previous_stn; // identifies previous station
     int in; // status of station
     float start_time[MAX_Q_SIZE]; // starting time of packets
     float hp_start_time[MAX_Q_SIZE]; // starting time of high priority packets
     float event_time[4]; // time of occurence of an event
     unsigned char corrupt_frame_flag;
     unsigned char skipped_flag;
   protected:
     float infinite;
   int * inum;
     public:
        Node();
        ~Node();
        virtual   float Get_arrival_rate();
        virtual   float Get_hp_arrival_rate();
        virtual   float Schedule_next_arrival();
        virtual   float Schedule_next_hp_arrival();
        virtual float Get_packet_length();
        virtual void Set_arrival_rate(float rate);
        virtual void Set_hp_arrival_rate(float rate);
        virtual void Set_packet_length(float length);
        virtual void Describe(ostream & strm,unsigned char mode)
                 { strm <<"Generic Node";}
        void Set_corrupt_frame_flag(unsigned char flag)
           {corrupt_frame_flag=flag;}
        unsigned char Get_corrupt_frame_flag()
           {return corrupt_frame_flag;}
        void Set_skipped_flag(unsigned char flag)
           {skipped_flag=flag;}
        unsigned char Get_skipped_flag()
           {return skipped_flag;}

        virtual float Get_Token_Rotation_Time();
        virtual void Set_Token_Rotation_Time(float value);



     }; // Node class
/*______________________________________________________________
* Class definition : Slave                                      *
* Inheritance from : Node                                       *
* HISTORY: Date         Author       Comments                   *
*          ---------------------------------------------------- *
*          19.06.96     S.M.Rolland  Creation                   *
_______________________________________________________________*/
class Slave : public Node{
     protected:
     float local_arrival_rate;
     float local_hp_arrival_rate;
     float local_packet_length;
     float Token_Rotation_Time;
     public:
     Slave(float rate,float hp_rate, float plength);
     ~Slave();
```

```
      virtual float Get_arrival_rate();
      virtual float Get_hp_arrival_rate();
      virtual   float Schedule_next_arrival();
      virtual   float Schedule_next_hp_arrival();
    virtual float Get_packet_length();
      virtual void Set_arrival_rate(float rate);
      virtual void Set_hp_arrival_rate(float rate);
      virtual void Set_packet_length(float length);
      virtual void Describe(ostream& strm,unsigned char mode)
                { strm <<"Generic Slave";}
      virtual float Get_Token_Rotation_Time();
      virtual void Set_Token_Rotation_Time(float value);

};
/*_____________________________________________________________________
* Class definition : Master                                           *
* Inheritance from : Node                                             *
* HISTORY: Date         Author       Comments                         *
*          ---------------------------------------------------------- *
*          19.06.96     S.M.Rolland  Creation                         *
_____________________________________________________________________*/
class Master : public Node{
      float local_arrival_rate;
      float local_hp_arrival_rate;
      float local_packet_length;
      float Token_Rotation_Time;
      public:
      Master(float rate, float hp_rate, float plength);
      ~Master();
         virtual   float Get_arrival_rate();
         virtual   float Get_hp_arrival_rate();
         virtual   float Schedule_next_arrival();
         virtual   float Schedule_next_hp_arrival();
         virtual float Get_packet_length();
         virtual void Set_arrival_rate(float rate);
         virtual void Set_hp_arrival_rate(float rate);
         virtual void Set_packet_length(float length);
         virtual void Describe(ostream& strm,unsigned char mode)
            {
            if (mode == VERBOSE)
               strm <<"Master\t AR = "<<local_arrival_rate<<"\t HP-AR = "<<local_hp_arrival_rate <<"\t
      P = "<<local_packet_length;
            else if (mode == TABULAR)
               strm <<"M";

            }
         virtual float Get_Token_Rotation_Time();
         virtual void Set_Token_Rotation_Time(float value);

      };

/*_____________________________________________________________________
* Class definition : Actuator                                         *
* Inheritance from : Slave                                            *
* HISTORY: Date         Author       Comments                         *
*          ---------------------------------------------------------- *
*          19.06.96     S.M.Rolland  Creation                         *
_____________________________________________________________________*/
class Actuator: public Slave{
      public:
      Actuator():Slave(0.0,0.0,48.0){}
      ~Actuator(){}
      //this is small packet size and low arrival rate
      virtual   float Schedule_next_arrival()
      {
      return infinite;// NO packet sent
      }
      virtual   float Schedule_next_hp_arrival()
      {
      return infinite; //No packets sent
      }
      virtual void Describe(ostream& strm,unsigned char mode)
               {
               if (mode == VERBOSE)
                  strm <<"Actuator\t AR = "<<local_arrival_rate<<"\t HP-AR =
      "<<local_hp_arrival_rate<<"\t P = "<<local_packet_length;
               else if (mode== TABULAR)
                  strm <<"A";
               }
      };
/*_____________________________________________________________________
* Class definition : Sensor                                           *
* Inheritance from : Slave                                            *
* HISTORY: Date         Author       Comments                         *
*          ---------------------------------------------------------- *
*          19.06.96     S.M.Rolland  Creation                         *
_____________________________________________________________________*/
class Sensor: public Slave{
      public:
      Sensor():Slave(7,0.001,80){}
         //this is small packet size and high arrival rate
```

APPENDIX C Network Simulation Software


```
     ~Sensor(){}
     virtual    float Schedule_next_arrival()        // Not a Poisson process!!!
         {
           return FACTOR/local_arrival_rate;
          }


     virtual    float Schedule_next_hp_arrival()    // Poisson Process
         {
           float x,result;
              for (;;)
              {
              x=ran0(inum);
              if (x!=0.0) break;
              }
           if (local_hp_arrival_rate> 0.000001)
              {
              result = -(float)log((double)x) * FACTOR / local_hp_arrival_rate;
              }
           else
              {
              result=infinite;
              }
           return result;
         }

     virtual void Describe(ostream& strm,unsigned char mode)
              {
              if (mode == VERBOSE)
                 strm <<"Sensor\t AR = "<<local_arrival_rate<<"\t HP_AR =
     "<<local_hp_arrival_rate<<"\t P = "<<local_packet_length;
              else if (mode == TABULAR)
                 strm <<"S";
              }
     };
/*___________________________________________________________________________
* Class definition : Tool                                                   *
* Inheritance from : Slave                                                  *
* HISTORY: Date          Author       Comments                              *
*          ------------------------------------------------------------------ *
*          19.06.96    S.M.Rolland  Creation                                *
___________________________________________________________________________*/
class Tool: public Slave{
     public:
     Tool():Slave(30,0.5,96){}
     //this is larger packet size and medium arrival rate
     ~Tool(){}
     virtual    float Schedule_next_arrival()        // Not a Poisson process
         {
                  return FACTOR/local_arrival_rate;
/*       float x,result; // Previous Poisson Process
              for (;;)
              {
              x=ran0(inum);
              if (x!=0.0) break;
              }
           if (local_arrival_rate> 0.000001)
              {
              result = -log(x) * FACTOR / local_arrival_rate;
              }
           else
              {
              result=infinite;
              }
           return result;                         */
         }


     virtual    float Schedule_next_hp_arrival()    // Poisson Process
         {
           float x,result;
              for (;;)
              {
              x=ran0(inum);
              if (x!=0.0) break;
              }
           if (local_hp_arrival_rate> 0.000001)
              {
              result = -log(x) * FACTOR / local_hp_arrival_rate;
              }
           else
              {
              result=infinite;
              }
              return result;
         }
     virtual void Describe(ostream& strm, unsigned char mode)
              {
              if (mode == VERBOSE)
                 strm <<"Tool\t\t AR = "<<local_arrival_rate<<"\t HP_AR =
     "<<local_hp_arrival_rate<<"\t P = "<<local_packet_length;
```

APPENDIX C Network Simulation Software


```
            else if (mode== TABULAR)
               strm <<"T";

            }

    };
/*_____________________________________________________________________
* Class definition : Net                                              *
* Friends  : Simu                                                     *
* Comments : contains instances of Node                               *
* HISTORY: Date         Author       Comments                         *
*          -----------------------------------------------------------*
*          19.06.96     S.M.Rolland  Creation                         *
______________________________________________________________________*/
class Net  {
friend class Simu;
private:

     int max_stations;
     int num_stations;
     Node ** station;


     public:
        Net(int size);
        ~Net();
        float Get_event_time(int i,int j);
        void Inc_queue_size(int i);
        void Dec_queue_size(int i);
        void Inc_hp_queue_size(int i);
        void Dec_hp_queue_size(int i);
        void Set_in(int i, int value);
        void Set_next_stn(int i, int value);
        int Get_next_stn(int i);
        void Set_previous_stn(int i, int value);
        int Get_previous_stn(int i);
        void Set_event_time(int i, int j, float value);
        void Set_start_time(int i, int queue, float value);
        void Set_hp_start_time(int i, int queue, float value);
     float Get_hp_start_time(int i,int queue);
        int Get_queue_size(int i);
        void Set_queue_size(int i, int value);
        int Get_hp_queue_size(int i);
        void Set_hp_queue_size(int i, int value);
        int Get_in(int i);
        float Get_start_time(int i, int queue);
        float Get_arrival_rate(int i);
        float Get_hp_arrival_rate(int i);
        float Schedule_next_arrival(int i);
        float Schedule_next_hp_arrival(int i);

        float Get_packet_time(int i);
        Node * Remove(int index);
        int Add(Node * n);
        void ListNodes(ostream & strm,unsigned char mode);
        void Edit_Node(int index, float rate, float length);
        void Set_corrupt_frame_flag(int i,unsigned char value);
        unsigned char Get_corrupt_frame_flag(int i);
        void Set_Token_Rotation_Time(int i, float value);
        float Get_Token_Rotation_Time(int i);
        void Set_skipped_flag(int i,unsigned char value);
        unsigned char Get_skipped_flag(int i);

    }; // Net class

/*_____________________________________________________________________
* Class definition : Medium                                           *
* Friends  : Simu                                                     *
* HISTORY: Date         Author       Comments                         *
*          -----------------------------------------------------------*
*          19.06.96     S.M.Rolland  Creation                         *
______________________________________________________________________*/
class Medium{
friend class Simu;
     private:

     int ring_or_bus; // flag to choose topology
     float packet_time; // average packet transmission time
     float stn_latency; //station latency in time units
     float token_time; // token transmission time
     float tok_ack_time; // token acknowledge transmission time*/
     float tau; // end to end propagation delay

     public:
        Medium();
        ~Medium();

     };// Medium class

/*_____________________________________________________________________
* Class definition : Simu                                             *
```

APPENDIX C Network Simulation Software


```
* Comments : contains instances of Net and Medium - This is a high *
*           level class                                              *
* HISTORY: Date         Author        Comments                       *
*          ------------------------------------------------------ *
*          19.06.96     S.M.Rolland   Creation                       *
*          30.09.96     "              Each node has its own stat    *
_________________________________________________________________*/
class Simu{
     Medium * mymedium;
     Net * net;
     float arrival_rate; // arrival rate in packets per sec per station
     float rho, clock, next_event_time;
     int *no_pkts_departed;
     float *delay, *total_delay, *average_delay, walk_time;
     float *delay_sum, *delay_sqr, *delay_var, *delay_sdv, *delay_con_int;
     int *no_hp_pkts_departed;
     float *hp_delay, *hp_total_delay, *hp_average_delay;
     float *hp_delay_sum, *hp_delay_sqr, *hp_delay_var, *hp_delay_sdv, *hp_delay_con_int;
     float trt,temp,trt_sum;
     long trt_count;
     int degrees_fr;
     int ic, flag, next_station, previous_station;
     float x, logx, rand_size, infinite;
     float **delay_ci;
     float **hp_delay_ci;
     int temp_flag;
     int stn_to_add, ring_size, next_event;
     int master_index;
     float Frame_Error_Rate;
     int error_count;
     float token_count;
     float TTRT;

     public :
        Simu();
        ~Simu();
        void Init();
        void Increase_Arrival_Rate();
        void Increase_ic_index();
        void Run();
        void Result();
        int Add_Node(Node* n);
        void List_Nodes(ostream& strm,unsigned char mode);
        int Delete_Node(int n);
        void Edit_Node(int n, float rate, float length);
     }; // Simu class
// End Of File
```

APPENDIX C Network Simulation Software


```
/*****************************************************************
* File : simu.cpp                                                *
* Description : OO network simulation                            *
* HISTORY: Date         Author       Comments                    *
*         ------------------------------------------------------ *
*         19/06/96     S.M.Rolland  Creation                     *
*****************************************************************/
#include "simu.hpp"
#include <fstream.h>
#include <io.h>
#include <fcntl.h>
#include "random.hpp"
/*________________________________________________________________
* Class Member definitions : Node                                *
* HISTORY: Date         Author       Comments                    *
*         ------------------------------------------------------ *
*         19.06.96     S.M.Rolland  Creation                     *
________________________________________________________________*/
Node::Node() //constructor
{
int i;
i=1;
inum=&i;
queue_size = 0;
hp_queue_size=0;
corrupt_frame_flag = FALSE;
skipped_flag=FALSE;
for(i=0;i<MAX_Q_SIZE; i++)
    start_time[i]=0.0;
/* assuming bus */
next_stn =-1;
previous_stn=-1;
in=0;
infinite= 1.0 * pow(10.0, 30.0);
for(i=0;i<3; i++)
    {
    event_time[i]=0.0;
    if (i!=0) event_time[i]=1.0 * pow(10.0,30.0);
    }
}
                    ////////////////////////////
Node::~Node(){} // destructor
float Node::Get_arrival_rate()
    {return 0.0;}
float Node::Get_hp_arrival_rate()
    {return 0.0;}
float Node::Schedule_next_arrival()
    {return 0.0;}
float Node::Schedule_next_hp_arrival()
    {return 0.0;}

float Node::Get_packet_length()
        {return 0.0;}
void Node::Set_arrival_rate(float rate)
    {}
void Node::Set_hp_arrival_rate(float rate)
    {}

void Node::Set_packet_length(float length)
    {}

float Node::Get_Token_Rotation_Time()
        {
        return 0.0;
        }
void Node::Set_Token_Rotation_Time(float value)
        {
        }

/*________________________________________________________________
* Class Member definitions : Slave                               *
* HISTORY: Date         Author       Comments                    *
*         ------------------------------------------------------ *
*         19.06.96     S.M.Rolland  Creation                     *
________________________________________________________________*/
Slave::Slave(float rate, float hp_rate, float plength){
            local_arrival_rate=rate;
            local_packet_length=plength;
            local_hp_arrival_rate=hp_rate;
            }
Slave::~Slave(){}
float Slave::Get_arrival_rate()
        {return local_arrival_rate;}
float Slave::Get_hp_arrival_rate()
        {return local_hp_arrival_rate;}
float Slave::Schedule_next_arrival()
    {return 0.0;}
float Slave::Schedule_next_hp_arrival()
    {return 0.0;}
```

APPENDIX C Network Simulation Software


```
float Slave::Get_packet_length()
            {return local_packet_length;}
void Slave::Set_arrival_rate(float rate)
    {local_arrival_rate=rate;}
void Slave::Set_hp_arrival_rate(float rate)
    {local_hp_arrival_rate=rate;}
void Slave::Set_packet_length(float length)
    {local_packet_length=length;}
float Slave::Get_Token_Rotation_Time()
            {
            return Token_Rotation_Time;
            }
void Slave::Set_Token_Rotation_Time(float value)
            {
            Token_Rotation_Time= value;
            }
/*_____________________________________________________________________
* Class Member definitions : Master                                  *
* HISTORY: Date         Author        Comments                        *
*          -------------------------------------------------------- *
*          19.06.96     S.M.Rolland   Creation                        *
_____________________________________________________________________*/
Master::Master(float rate, float hp_rate, float plength){
                local_arrival_rate=rate;
                local_hp_arrival_rate=hp_rate;
                local_packet_length=plength;
                }
Master::~Master(){}
float Master::Get_arrival_rate()
            {return local_arrival_rate;}
float Master::Get_hp_arrival_rate()
            {return local_hp_arrival_rate;}
float Master::Schedule_next_arrival()       // Poisson process
        {
        float x,result;
                for (;;)
                {
                x=ran0(inum);
                if (x!=0.0) break;
                }
            if (local_arrival_rate> 0.000001)
                {
                result = -log(x) * FACTOR / local_arrival_rate;
                }
            else
                {
                result=infinite;
                }
            return result;
         }


float Master::Schedule_next_hp_arrival()  // Poisson Process
        {
            float x,result;
                for (;;)
                {
                x=ran0(inum);
                if (x!=0.0) break;
                }
            if (local_hp_arrival_rate> 0.000001)
                {
                result = -log(x) * FACTOR / local_hp_arrival_rate;
                }
            else
                {
                result=infinite;
                }
                return result;
         }


float Master::Get_packet_length()
            {return local_packet_length;}
void Master::Set_arrival_rate(float rate)
    {local_arrival_rate=rate;}
void Master::Set_hp_arrival_rate(float rate)
    {local_hp_arrival_rate=rate;}

void Master::Set_packet_length(float length)
    {local_packet_length=length;}
float Master::Get_Token_Rotation_Time()
            {
            return Token_Rotation_Time;
            }
void Master::Set_Token_Rotation_Time(float value)
            {
            Token_Rotation_Time= value;
```

```
              }
/*_____________________________________________________________
* Class Member definitions : Net                              *
* HISTORY: Date         Author        Comments                *
*          -------------------------------------------------- *
*          19.06.96     S.M.Rolland   Creation                *
_____________________________________________________________*/
Net::Net(int size)
{
max_stations=size;
num_stations=0;
station=new Node *[size];
for (int i=0; i<size; ++i)
    station[i]=NULL;

}
Net::~Net()
{
}
                       ////////////////////////////
float Net::Get_event_time(int i,int j)
{
return station[i]->event_time[j];
}
                       ////////////////////////////
float Net::Get_start_time(int i,int queue)
{
return station[i]->start_time[queue];
}
                       ////////////////////////////
void Net::Inc_queue_size(int i)
{
station[i]->queue_size++;
if (station[i]->queue_size > MAX_Q_SIZE)
    {
    station[i]->queue_size--;
/*  cout << "Queue size too large";*/
/*  exit(1);*/
    }
}
                       ////////////////////////////
void Net::Dec_queue_size(int i)
{
station[i]->queue_size--;
}
                       ////////////////////////////
void Net::Inc_hp_queue_size(int i)
{
station[i]->hp_queue_size++;
if (station[i]->hp_queue_size > MAX_Q_SIZE)
    {
    station[i]->hp_queue_size--;
/*  cout << "Queue size too large";*/
/*  exit(1);*/
    }
}
                       ////////////////////////////
void Net::Dec_hp_queue_size(int i)
{
station[i]->hp_queue_size--;
}

                       ////////////////////////////
void Net::Set_in(int i, int value)
    {
        station[i]->in=value;
    }
                       ////////////////////////////
int Net::Get_in(int i)
    {
    return (station[i]->in);
    }

                       ////////////////////////////
void Net::Set_next_stn(int i, int value)
    {
        station[i]->next_stn=value;
    }
                       ////////////////////////////
int Net::Get_next_stn(int i)
    {
    return (station[i]->next_stn);
    }

                       ////////////////////////////
void Net::Set_previous_stn(int i, int value)
    {
        station[i]->previous_stn=value;
    }
```

APPENDIX C Network Simulation Software


```
                    /////////////////////////////
int Net::Get_previous_stn(int i)
    {
    return (station[i]->previous_stn);
    }

                    /////////////////////////////
void Net::Set_event_time(int i, int j, float value)
    {
    station[i]->event_time[j]=value;

    }
                    /////////////////////////////
void Net::Set_start_time(int i, int queue, float value)
    {
    station[i]->start_time[queue]=value;
    }
                    /////////////////////////////
int  Net::Get_queue_size(int i)
    {
/*  if (i==2)
        return 1;*/
    return (station[i]->queue_size);
    }
void Net::Set_queue_size(int i, int value)
                {
    station[i]->queue_size = value;
    }
                    /////////////////////////////
int  Net::Get_hp_queue_size(int i)
    {
    return (station[i]->hp_queue_size);
    }
void Net::Set_hp_queue_size(int i, int value)
                {
    station[i]->hp_queue_size = value;
    }
                    /////////////////////////////
float Net::Get_arrival_rate(int i)
    {
    return station[i]->Get_arrival_rate();
    }
                    /////////////////////////////

float Net::Get_hp_arrival_rate(int i)
    {
    return station[i]->Get_hp_arrival_rate();
    }
                    /////////////////////////////
float Net::Schedule_next_arrival(int i)
    {return station[i]->Schedule_next_arrival();}
                        /////////////////////////////
float Net::Schedule_next_hp_arrival(int i)
    {return station[i]->Schedule_next_hp_arrival();}
                        /////////////////////////////
float Net::Get_packet_time(int i)
    {
    return (station[i]->Get_packet_length()  * FACTOR /RATE);
    }
                    /////////////////////////////
Node *  Net::Remove(int index)
    {
    if (index>max_stations)
        return 0;
    if (station[index]!=NULL)
        {
        Node * temp=station[index];
        station[index]=NULL;
        --num_stations;
        return temp;
        }
    else
        return NULL;
    }
                    /////////////////////////////
int  Net::Add(Node* n)
    {
    if (num_stations == max_stations)
        return 0;
    ++ num_stations;
    int i=0;
    while (station[i]!=NULL)
        ++i;
    station[i]=n;
    return i+1;
    }
                    /////////////////////////////
void  Net::ListNodes(ostream& strm, unsigned char mode)
    {
```

```
    if (num_stations >0)
        for(int i=0;i<num_stations;++i)
            if (station[i] !=NULL)
                {
                if (mode == VERBOSE)
                    strm << "\nNode "<<i << " is ";
                station[i]->Describe(strm,mode);
                }
           else
            {
                if (mode == VERBOSE)
                    strm << "\nNode "<<i << " is  NULL";
            }
    }
                    ///////////////////////////
void Net::Edit_Node(int index, float rate, float length)
    {
    station[index]->Set_arrival_rate(rate);
    station[index]->Set_packet_length(length);
    }
void Net::Set_corrupt_frame_flag(int i, unsigned char value)
    {
    station[i]->Set_corrupt_frame_flag(value);
    }
unsigned char Net::Get_corrupt_frame_flag(int i)
    {
    return    station[i]->Get_corrupt_frame_flag();
    }
void Net::Set_skipped_flag(int i, unsigned char value)
    {
    station[i]->Set_skipped_flag(value);
    }
unsigned char Net::Get_skipped_flag(int i)
    {
    return    station[i]->Get_skipped_flag();
    }

void Net::Set_Token_Rotation_Time(int i, float value)
    {
    station[i]->Set_Token_Rotation_Time(value);
    ;
    }
float Net::Get_Token_Rotation_Time(int i)
    {
    return station[i]->Get_Token_Rotation_Time();
    }

float Net::Get_hp_start_time(int i,int queue)
{
return station[i]->hp_start_time[queue];
}
void Net::Set_hp_start_time(int i, int queue, float value)
    {
    station[i]->hp_start_time[queue]=value;
    }

/*______________________________________________________________________________
* Class Member definitions : Medium                                            *
* HISTORY: Date          Author       Comments                                 *
*          ---------------------------------------------------------------     *
*          19.06.96     S.M.Rolland  Creation                                 *
______________________________________________________________________________*/
Medium::Medium(){
          ring_or_bus = 0;
          packet_time = 56.0 * FACTOR / RATE; // this is the average packet time
          stn_latency = 0.0085 * FACTOR;     // from measurements
          token_time = 48.0 * FACTOR / RATE;
          tok_ack_time=48.0 * FACTOR / RATE;
          tau = MEDIUM_LENGTH * FACTOR * 5.0 * pow(10.0,-9.0);
            }
                ///////////////////////////
Medium::~Medium() {};
/*______________________________________________________________________________
* Class Member definitions : Simu                                              *
* HISTORY: Date          Author       Comments                                 *
*          ---------------------------------------------------------------     *
*          19.06.96     S.M.Rolland  Creation                                 *
______________________________________________________________________________*/
Simu::Simu()
{
   mymedium = new Medium();
   net = new Net(MAX_NODES);
   degrees_fr=5;
   delay_ci= new float*[5];          // STEP 1: SET UP THE ROWS.
   for (int j = 0; j <= 5; j++)
           delay_ci[j] = new float[MAX_NODES];  // STEP 2: SET UP THE COLUMNS
   hp_delay_ci= new float*[5];          // STEP 1: SET UP THE ROWS.
   for (j = 0; j <= 5; j++)
           hp_delay_ci[j] = new float[MAX_NODES];  // STEP 2: SET UP THE COLUMNS
```

```

    no_pkts_departed=new int[MAX_NODES];
    delay=new float[MAX_NODES];
    total_delay =new float[MAX_NODES];
    average_delay =new float[MAX_NODES];
    delay_sum =new float[MAX_NODES];
    delay_sqr=new float[MAX_NODES];
    delay_var=new float[MAX_NODES];
    delay_sdv=new float[MAX_NODES];
    delay_con_int=new float[MAX_NODES];
    no_hp_pkts_departed=new int[MAX_NODES];
    hp_delay=new float[MAX_NODES];
    hp_total_delay=new float[MAX_NODES];
    hp_average_delay=new float[MAX_NODES];
    hp_delay_sum=new float[MAX_NODES];
    hp_delay_sqr=new float[MAX_NODES];
    hp_delay_var=new float[MAX_NODES];
    hp_delay_sdv=new float[MAX_NODES];
    hp_delay_con_int=new float[MAX_NODES];

    arrival_rate=0.5; // this is the global arrival rate
    rho=0.0;
    clock=0.0;
    for (int i=0;i<MAX_NODES;i++)
        {
        no_pkts_departed[i] = 0;
        total_delay[i]=0.0;
        average_delay[i]=0.0;
        no_hp_pkts_departed[i] = 0;
        hp_total_delay[i]=0.0;
        hp_average_delay[i]=0.0;
        }
    flag=1.0;
    next_event_time = 0.0;
    next_event=-1;
    ic=-1;
    rand_size = 0.5 * pow(2.0,8.0* sizeof(int));
    infinite= 1.0 * pow(10.0, 30.0);
    master_index=0;
    Frame_Error_Rate=1000.0; /* one in Frame_Error_Rate frame will be corrupted*/
    error_count=0;
    token_count=0;
    for (i=0;i<5;i++)
                {
                for (j=0;j<MAX_NODES;j++)
                    {
                    delay_ci[i][j]=0;
                    hp_delay_ci[i][j]=0;
                    }
                }

}
                ////////////////////////////

Simu::~Simu()
{
delete (mymedium);
delete (net);
}
                ////////////////////////////
void Simu::Init()
{
    trt=0.0;
    temp=0.0;
    trt_sum=0.0;
    temp=0.0;
    trt_count=0;
    degrees_fr=5;
    arrival_rate=20.0; // this is the global arrival rate
    rho=0.0;
    clock=0.0;
    for (int j=0;j<MAX_NODES;j++)
        {
        no_pkts_departed[j] = 0;
        total_delay[j]=0.0;
        average_delay[j]=0.0;
        no_hp_pkts_departed[j] = 0;
        hp_total_delay[j]=0.0;
        hp_average_delay[j]=0.0;
        }

    flag=1.0;
    next_event_time = 0.0;
    next_event=-1;
    ic=-1;
    rand_size = 0.5 * pow(2.0,8.0* sizeof(int));
    infinite= 1.0 * pow(10.0, 30.0);
    error_count=0;
    token_count=0;
```

```
    cout <<"\nEnter the Frame Error Rate (float) : " << flush;
    cin >> Frame_Error_Rate;
    cout <<"\nEnter TTRT : "<<flush;
    cin >> TTRT;
}

                  ////////////////////////////
void Simu::Increase_Arrival_Rate()
{
arrival_rate= arrival_rate + 20.0;
}
                  ////////////////////////////
int Simu::Add_Node(Node *n)
{
if (net->Add(n)==0)
    {
    cout <<"\nCould not add a node\n";
    return (0);
    }
else
    return 1;
}
                  ////////////////////////////
void Simu::List_Nodes(ostream& strm,unsigned char mode)
{
if (net->num_stations ==0)
    {
    if (mode == VERBOSE)
       strm <<"\nEmpty Network!";
    }
net->ListNodes(strm,mode);
if (mode == VERBOSE)
    strm << "\nTotal of " << net->num_stations <<" nodes.";
}

                  ////////////////////////////
int Simu::Delete_Node(int n)
{
Node *temp=net->Remove(n);
if ((temp==NULL) || (temp ==0))
    return 0;
for (int i=n+1;i<=net->num_stations;i++) // shift down the rest of the nodes
    {
    temp=net->Remove(i);
    if ((temp==NULL) || (temp ==0))
       return 0;
    Add_Node(temp);
    }
return 1;
}
void Simu::Edit_Node(int n, float rate, float length)
{
if (n<net->num_stations)
    {
    net->Edit_Node(n,rate,length);
    cout <<"\nNode modified";
    }
else
    cout << "\nThis node does not exist";
}

                  ////////////////////////////
void Simu::Increase_ic_index()
{
int i,j;
if (ic<=degrees_fr)
{
    ic=ic+1;
    rho=0.0;
    clock=0.0;
    temp=0;
    for (i=0;i<MAX_NODES;i++)
        {
        no_pkts_departed[i] = 0;
        total_delay[i]=0.0;
        average_delay[i]=0.0;
        no_hp_pkts_departed[i] = 0;
        hp_total_delay[i]=0.0;
        hp_average_delay[i]=0.0;
        }
    flag=1.0;
    next_event_time = 0.0;
    rho=arrival_rate * 48.0 * net->num_stations / RATE; // using global arrival rate
    if (rho >=1.0)
        {
/*      cout <<"Warning Traffic intensity is too high"<<"\n";// not necessarily true!!!! obsolete
/*      exit(1);*/
        }
    for(i = 0;i<net->num_stations; i++)
```

```
        {
        net->Set_queue_size(i,0);
        net->Set_hp_queue_size(i,0);
        for (j=0;j<MAX_Q_SIZE;j++)
                {
                net->Set_start_time(i,j,0.0);
                net->Set_hp_start_time(i,j,0.0);
                }
        }
}
}
                ////////////////////////////
void Simu::Run()
{
int i, j;
//float trt,temp;
int temp_stn,next;
int * inum;
float error_gen;
unsigned char end;
i=1;
inum=&i;
end=FALSE;
//ofstream tst("test.log",ios::out|ios::app);// output file
ofstream tlog("token.log",ios::out|ios::app);// output file
if (mymedium->ring_or_bus ==1) // RING
    {
    ring_size=net->num_stations;
    walk_time=mymedium->token_time + mymedium->stn_latency + mymedium->tau/net->num_stations;
    }
else // BUS
    {
    ring_size=0;
    walk_time= mymedium->token_time +mymedium->stn_latency+  mymedium->tau/3.0+mymedium->tau/3.0 +
mymedium->tok_ack_time;
    }
for(i=0;i<net->num_stations;i++)
    {
    net->Set_next_stn(net->num_stations-1,0);
    net->Set_previous_stn(0,net->num_stations-1);
    net->Set_previous_stn(net->num_stations-1,net->num_stations-2);
    net->Set_next_stn(0,1);
        if ((i<(net->num_stations-1)) && (i>0))
            {
            net->Set_next_stn(i,i+1);
            net->Set_previous_stn(i,i-1);
            }
    }
for(i=0;i<(net->num_stations);i++)
    {
    net->Set_Token_Rotation_Time(i,0.0);
    for (j=0;j<5;j++)
        {
        net->Set_event_time(i,j,0.0);
        if ((j!=0)  && (j!=4))
            net->Set_event_time(i,j,infinite);
        }
    }

while (end==FALSE)
    {
    next_event_time=infinite;
    for(i=0;i<(net->num_stations);i++)
        {
        if (no_pkts_departed[i] > MAX_PACKETS)
            end = TRUE;
        for(j=0;j<5;j++)
            {
            if (next_event_time >  net->Get_event_time(i,j))
                {
                next_event_time = net->Get_event_time(i,j);
                next_station=i;
                next_event=j;
                }
            }
        }
    clock=next_event_time;
    if (next_event > 4)
        {
        cout <<"Check the Event list";
        exit(1);
        }
    // SCAN THE EVENT LIST
    switch (next_event)
        {
        case 0:// arrival event
            {
//          tlog <<"\nNA "<< next_station <<" "<<clock;
//          net->Inc_queue_size(next_station); // INCREASE NORMAL PRIORITY QUEUE
```

```
            if (flag ==1.0)
               {
               flag=0.0;
               net->Set_event_time(next_station,2,clock);
               }

            // schedule for next arrival

           x=net->Schedule_next_arrival(next_station);
//         if (next_station==0)
  //           tst <<x<<"\n";
            if (net->Get_arrival_rate(next_station) !=0.0)
               {
               net->Inc_queue_size(next_station); // INCREASE QUEUE
               net->Set_event_time(next_station, next_event,(clock+x));
               net->Set_start_time(next_station, (net->Get_queue_size(next_station) -1),
                                                                clock);
               }
            else
               {
               net->Set_event_time(next_station, next_event,x+clock);
               }
            break;
            }
         case 1 : // departure event
            {
                     tlog <<"\nND "<< next_station <<" ";
            if (net->Get_queue_size(next_station) >0)    // QUEUE SIZE CHECK
               {
               if (net->Get_corrupt_frame_flag(next_station) != TRUE) // ERROR GEN
                  {
                  if (net->Get_skipped_flag(next_station)!=TRUE) // TTRT CHECK
                     {
                     net->Dec_queue_size(next_station);
                     no_pkts_departed[next_station] ++;
                     if (next_station==0) // added to simulate the fact that PC NOT on interrupts
                        {
                           if ((clock - net->Get_start_time(next_station,0)) < 40) // PC dead time
                              {
                                 delay[next_station]=40;
                              }
                           else

                           delay[next_station]=clock - net->Get_start_time(next_station,0);// COMPUTE DELAY
                        }
                     else
                     delay[next_station]=clock - net->Get_start_time(next_station,0);// COMPUTE DELAY
                     total_delay[next_station] +=delay[next_station];
                     // push the queue forward
                     for(i=0;i<net->Get_queue_size(next_station);i++)
                        net->Set_start_time(next_station,i,
                                              net->Get_start_time(next_station,i+1));
                     net->Set_start_time(next_station,net->Get_queue_size(next_station),0.0);
                     net->Set_event_time(next_station,next_event,infinite);
                     tlog <<" Txed";
                     }
                  else // TTRT CHECK TRUE
                     {
                                 tlog <<" Skipped";
                     net->Set_skipped_flag(next_station,FALSE);
                     net->Set_event_time(next_station,next_event,infinite);
                     }
                  }
               else // ERROR GEN TRUE
                  {
                  net->Set_corrupt_frame_flag(next_station,FALSE);
                  net->Set_event_time(next_station,next_event,infinite); // THe station doesn't know it was corrupt
                                                      tlog <<" Error";
                  }
               }
               else // QUEUE SIZE CHECK EMPTY
                  {
                  net->Set_event_time(next_station,next_event,infinite);
                  tlog <<" EMPTY";

                  }
               // Modified logical ring management
               next=net->Get_next_stn(next_station);
               // Token Passing
               net->Set_event_time(next,2,clock+walk_time);
               error_gen=ran0(inum);
               if ((error_gen * Frame_Error_Rate) <1.0)
                  {
                  // error during token passing => reopeat procedure????*/
                  error_count++;
```

APPENDIX C Network Simulation Software


```
                    net->Set_event_time(next,2,clock+2*walk_time);
                    }
            break;
             }
        case 2: // This is a token arrival event
            {
//                      tlog <<"\nTOK "<< next_station <<" "<<clock;
            if (next_station == 0)
                {

                if (temp !=0.0)
                    {
                    trt=net->Get_event_time(next_station,2)-temp; // TRT STATISTICS
                    trt_count++;
                    }
                temp=net->Get_event_time(next_station,2);
                trt_sum=trt_sum + trt;

                }
                net->Set_event_time(next_station,2,infinite);

                if (net->Get_hp_queue_size(next_station) >0)    // QUEUE SIZE CHECK
                    {
                    error_gen=ran0(inum);
                    if ((error_gen * Frame_Error_Rate) <1.0)
                        {
                        // Corrupt frame => not actually transmitted*/
                        error_count++;
                        net->Set_corrupt_frame_flag(next_station,TRUE);
                        }
                    net->Set_event_time(next_station,3,clock + net->Get_packet_time(next_station));
                    }
                else
                    {
                    net->Set_event_time(next_station,3,clock);
                    }
                if( (clock-net->Get_Token_Rotation_Time(next_station)) <TTRT)   // PREPARE FOR TTRT CHECK
                    {
//                  cout <<" T "<<clock<<" "<<(net->Get_Token_Rotation_Time(next_station))<< " "<<clock-net->Get_Token_Rotation_Time(next_station);
                                        net->Set_skipped_flag(next_station, FALSE);
                    }
                    else
                    {
//                  cout <<" S "<<clock-net->Get_Token_Rotation_Time(next_station);
                    net->Set_skipped_flag(next_station, TRUE);
                    }

                net->Set_Token_Rotation_Time(next_station,clock);
            break;
            }
        case 3 : // departure of high priority frame
            {
//                      tlog <<"\nHPD "<< next_station <<" "<<clock;
            if (net->Get_hp_queue_size(next_station) >0)    // QUEUE SIZE CHECK
                {
                net->Dec_hp_queue_size(next_station);
                if (net->Get_corrupt_frame_flag(next_station) != TRUE) // ERROR GEN
                    {
//                                                          tlog <<" Txed";
                    no_hp_pkts_departed[next_station] ++;
                    hp_delay[next_station]=clock - net->Get_hp_start_time(next_station,0);// COMPUTE DELAY
                    hp_total_delay[next_station] +=hp_delay[next_station];
                    // push the queue forward
                    for(i=0;i<net->Get_hp_queue_size(next_station);i++)
                        net->Set_hp_start_time(next_station,i,
                                            net->Get_hp_start_time(next_station,i+1));
                    net->Set_hp_start_time(next_station,net->Get_hp_queue_size(next_station),0.0);
                    net->Set_event_time(next_station,next_event,infinite);
                    }
                else // ERROR GEN TRUE
                    {
//                                                          tlog <<" Error";
                    net->Set_corrupt_frame_flag(next_station,FALSE);
                    net->Set_event_time(next_station,next_event,infinite); // THe station doesn't know it was corrupt
                    }
                }
                else // QUEUE SIZE CHECK EMPTY
                    {
//                                                          tlog <<" Empty";
                    net->Set_event_time(next_station,next_event,infinite);
                    }
             // TRIGGER LP_EVENT
            if (net->Get_queue_size(next_station) >0)   // QUEUE SIZE CHECK
            {
            if (net->Get_skipped_flag(next_station) !=TRUE)
```

```
                {
//                 net->Set_event_time(next_station,1,clock + net->Get_packet_time(next_station));
                   net->Set_event_time(next_station,1,clock );
                   error_gen=ran0(inum);
                   if ((error_gen * Frame_Error_Rate) <1.0)
                      {
                      // Corrupt frame => not actually transmitted*/
                      error_count++;
                      net->Set_corrupt_frame_flag(next_station,TRUE);

                      }
                }
                else
                {
                net->Set_event_time(next_station,1,clock);
                }
             }
             else
                {
                net->Set_event_time(next_station,1,clock);
                }
             break;
             }
        case 4:// HP arrival event
             {
//           tlog <<"\nHP "<< next_station <<" "<<clock;
          x=net->Schedule_next_hp_arrival(next_station);
            if (net->Get_hp_arrival_rate(next_station) !=0.0)
                {
                net->Inc_hp_queue_size(next_station); // INCREASE HP PRIORITY QUEUE
                net->Set_event_time(next_station, next_event,(clock+x));
                net->Set_hp_start_time(next_station, (net->Get_hp_queue_size(next_station) -1),
                                                                               clock);
//               tlog <<" Added";
                }
            else
                {
                net->Set_event_time(next_station, next_event,x+clock);
//              tlog <<" No HP Frame";
                 }
            break;
            }

        } // end of switch
       } // end of while
       for (i=0;i<net->num_stations;i++)
       {
        if (no_pkts_departed[i] == 0)
          {
          average_delay[i] =0.0;
          }
        else
          {
          average_delay[i]= total_delay[i] /(no_pkts_departed[i] *FACTOR);
          if (no_hp_pkts_departed[i] ==0)
             hp_average_delay[i]=0;
          else
             hp_average_delay[i]= hp_total_delay[i] /(no_hp_pkts_departed[i] *FACTOR);
          }
        delay_ci[ic][i]=average_delay[i];
        hp_delay_ci[ic][i]=hp_average_delay[i];
       }

  }

void Simu::Result()
{
int  i,j;

ofstream ostrm("simu.log",ios::out|ios::app);// output file
             if (net->num_stations <=1)
                cout << "\nA network needs at least two stations ! Will not run simulation";
             else
             {
             for (j=0;j<=degrees_fr;j++)
                {
                Increase_ic_index();
                Run();
                }

             for (i=0;i<net->num_stations;i++)
             {
             delay_sum[i] =0.0;
             delay_sqr[i] = 0.0;
             hp_delay_sum[i] =0.0;
             hp_delay_sqr[i] = 0.0;

             for (j=0;j<=degrees_fr;j++)
                {
```

University of Southampton Library

```
                    delay_sum[i] +=delay_ci[j][i];
                    delay_sqr[i] += pow(delay_ci[j][i], 2.0);
                    hp_delay_sum[i] +=hp_delay_ci[j][i];
                    hp_delay_sqr[i] += pow(hp_delay_ci[j][i], 2.0);
                    }
                cout <<"\nNODE "<<i<<"*************";
                if (delay_sum[i]!=0)
                    {
                    delay_sum[i] = delay_sum[i] / (degrees_fr +1);
                    delay_sqr[i] = delay_sqr[i] / (degrees_fr +1);
                    delay_var[i] =delay_sqr[i] - pow(delay_sum[i], 2.0);
                    if (delay_var[i] >0)
                        {
                        delay_sdv[i] = sqrt(delay_var[i]);
                        delay_con_int [i]= delay_sdv[i] * T_dist_par[degrees_fr-1]/sqrt (degrees_fr);
                        }

                    cout << "\nThe average delay is " << delay_sum[i] << "+/-" << delay_con_int[i];
                    List_Nodes(ostrm,TABULAR);
                    if (delay_con_int[i] >0)
                        {
                        cout << " Validity check " << degrees_fr <<">= "<<delay_sdv[i] * T_dist_par[degrees_fr-1]/sqrt (delay_con_int[i]);
                        ostrm <<","<<i<< ","<<delay_sum[i] << "," << delay_con_int[i] <<"," << net->num_stations <<","<<
                        delay_sdv[i] * T_dist_par[degrees_fr-1]/sqrt (delay_con_int[i])<<","<< trt_sum/trt_count<<","<<Frame_Error_Rate<<
                        ","<<TTRT<<"\n" ;
                        }
                    else
                        {
                        cout << "\nUnable to compile confidence interval check";
                        ostrm <<","<<i<< ","<<delay_sum[i] << "," << delay_con_int[i] <<"," << net->num_stations <<","<<
                        "Failed"<<","<< trt_sum/trt_count<<","<<Frame_Error_Rate<<","<<TTRT<<"\n" ;
                         }
                    }
                else
                    cout <<"\nNo information gathered "<<delay_sum[i];
                if (hp_delay_sum[i]!=0)
                    {
                    hp_delay_sum[i] = hp_delay_sum[i] / (degrees_fr +1);
                    hp_delay_sqr[i] = hp_delay_sqr[i] / (degrees_fr +1);
                    hp_delay_var[i] =hp_delay_sqr[i] - pow(hp_delay_sum[i], 2.0);
                    if (hp_delay_var[i] >0)
                        hp_delay_sdv[i] = sqrt(hp_delay_var[i]);
                    else
                        hp_delay_sdv[i]=0;
                    hp_delay_con_int[i] = hp_delay_sdv[i] * T_dist_par[degrees_fr-1]/sqrt (degrees_fr);
                    cout << "\nThe average hp_delay is " << hp_delay_sum[i] << "+/-" << hp_delay_con_int[i] ;
                    if (hp_delay_con_int[i] >0)
                        cout << " Validity check " << degrees_fr <<">= "<<hp_delay_sdv[i] * T_dist_par[degrees_fr-1]/sqrt (hp_delay_con_int[i]);
                    }
                else
                    cout <<"\nNo High Priority Frame information gathered "<<hp_delay_sum[i];
                }
                cout << "\nAverage trt "<< trt_sum/trt_count ;
                cout <<"\n Generated "<< error_count <<" errors. ";

}
}
// End of File
```

APPENDIX C Network Simulation Software


```
/******************************************************************
* File : netsim.cpp                                              *
* Description : OO network simulation                            *
* HISTORY: Date         Author       Comments                    *
*          ---------------------------------------------------- *
*          13/06/96     S.M.Rolland  Creation                    *
*          17/06/96     "            Tested against simtp.c      *
*****************************************************************/
#include <math.h>
#include <stdlib.h>
#include <iostream.h>
#include <conio.h>
const MAX_Q_SIZE =100;
const MAX_PACKETS = 10000;
float RATE = 10000000.0;
float FACTOR = 1000.0;
float T_dist_par[10] = {12.706, 4.303, 3.182, 2.776, 2.571, 2.447, 2.635, 2.306, 2.262, 2.228};
/* T distribution parameters*/

/////////////////////////////////////////////////////////////////////
class Node{
friend class Net;
   int queue_size; // queue size at station
   int next_stn; // identifies next station
   int previous_stn; // identifies previous station
   int in; // status of station
   float start_time[MAX_Q_SIZE]; // starting time of packets
   float event_time[3]; // time of occurence of an event

   public:
      Node();
      ~Node();

   }; // Node class
                 ////////////////////////////
Node::Node() //constructor
{
int i;
queue_size = 0;
for(i=0;i<MAX_Q_SIZE; i++)
   start_time[i]=0.0;
/* assuming bus */
next_stn =-1;
previous_stn=-1;
in=0;
for(i=0;i<3; i++)
   {
   event_time[i]=0.0;
   if (i!=0) event_time[i]=1.0 * pow(10.0,30.0);
   }
}
                 ////////////////////////////
Node::~Node(){}; // destructor

/////////////////////////////////////////////////////////////////////
class Net  {
friend class Simu;
private:

   int max_stations;
   Node *station;

   public:
      Net(int size);
      ~Net();
      float Get_event_time(int i,int j);
      void Inc_queue_size(int i);
      void Dec_queue_size(int i);
      void Set_in(int i, int value);
      void Set_next_stn(int i, int value);
      int Get_next_stn(int i);
      void Set_previous_stn(int i, int value);
      int Get_previous_stn(int i);
      void Set_event_time(int i, int j, float value);
      void Set_start_time(int i, int queue, float value);
      int Get_queue_size(int i);
      void Set_queue_size(int i, int value);
      int Get_in(int i);
      float Get_start_time(int i, int queue);


   }; // Net class

                 ////////////////////////////
Net::Net(int size)
{
max_stations=size;
station=new Node[size];

```

```
}
                    /////////////////////////////
Net::~Net()
{
delete[] station;
}
                    /////////////////////////////
float Net::Get_event_time(int i,int j)
{
return station[i].event_time[j];
}
                    /////////////////////////////
float Net::Get_start_time(int i,int queue)
{
return station[i].start_time[queue];
}
                    /////////////////////////////
void Net::Inc_queue_size(int i)
{
station[i].queue_size++;
if (station[i].queue_size > MAX_Q_SIZE)
   {
   cout << "Queue size too large";
   exit(1);
   }
}
                    /////////////////////////////
void Net::Dec_queue_size(int i)
{
station[i].queue_size--;
}

                    /////////////////////////////
void Net::Set_in(int i, int value)
   {
      station[i].in=value;
   }
                    /////////////////////////////
int Net::Get_in(int i)
   {
   return (station[i].in);
   }

                    /////////////////////////////
void Net::Set_next_stn(int i, int value)
   {
      station[i].next_stn=value;
   }
                    /////////////////////////////
int Net::Get_next_stn(int i)
   {
   return (station[i].next_stn);
   }

                    /////////////////////////////
void Net::Set_previous_stn(int i, int value)
   {
      station[i].previous_stn=value;
   }
                    /////////////////////////////
int Net::Get_previous_stn(int i)
   {
   return (station[i].previous_stn);
   }

                    /////////////////////////////
void Net::Set_event_time(int i, int j, float value)
   {
   station[i].event_time[j]=value;

   }
                    /////////////////////////////
void Net::Set_start_time(int i, int queue, float value)
   {
   station[i].start_time[queue]=value;
   }
                    /////////////////////////////
int  Net::Get_queue_size(int i)
   {
   return (station[i].queue_size);
   }
void Net::Set_queue_size(int i, int value)
   {
   station[i].queue_size = value;
   }
//////////////////////////////////////////////////////////////////////
class Medium{
friend class Simu;
   private:
```

APPENDIX C Network Simulation Software


```

    int ring_or_bus; // flag to choose topology
    float packet_time; // packet transmission time
    float stn_latency; //station latency in time units
    float token_time; // token transmission time
    float tau; // end to end propagation delay

    public:
       Medium();
       ~Medium();

    };// Medium class
                    ////////////////////////////
Medium::Medium(){
          ring_or_bus = 0;
          packet_time = 1000.0 * FACTOR / RATE;
          stn_latency = 0.0;
          token_time = 50.0 * FACTOR / RATE;
          tau = 0.01;
             }
                    ////////////////////////////
Medium::~Medium() {};

///////////////////////////////////////////////////////////////////////
class Simu{
    Medium * mymedium;
    Net * net;
    float arrival_rate; // arrival rate in packets per sec per station
    float rho, clock, no_pkts_departed, next_event_time;
    float delay, total_delay, average_delay, walk_time;
    float delay_sum, delay_sqr, delay_var, delay_sdv, delay_con_int;
    int degrees_fr;
    int ic, flag, next_station, previous_station;
    float x, logx, rand_size, infinite;
    float *delay_ci;
    int temp_flag;
    int stn_to_add, ring_size, next_event;
    public :
       Simu();
       ~Simu();
       void Increase_Arrival_Rate();
       void Increase_ic_index();
       void Run();
       void Result();
    }; // Simu class
                    ////////////////////////////
Simu::Simu()
{
    mymedium = new Medium();
    net = new Net(50);
    degrees_fr=5;
    delay_ci = new float[5];
    arrival_rate=0;
    rho=0.0;
    clock=0.0;
    no_pkts_departed = 0.0;
    total_delay=0.0;
    average_delay=0.0;
    flag=1.0;
    next_event_time = 0.0;
    next_event=-1;
    ic=-1;
    rand_size = 0.5 * pow(2.0,8.0* sizeof(int));
    infinite= 1.0 * pow(10.0, 30.0);
}
                    ////////////////////////////

Simu::~Simu()
{
delete delay_ci;
delete (mymedium);
delete (net);
}
                    ////////////////////////////
void Simu::Increase_Arrival_Rate()
{
arrival_rate= arrival_rate + 20.0;
}
                    ////////////////////////////
void Simu::Increase_ic_index()
{
int i,j;
if (ic<=degrees_fr)
{
    ic=ic+1;
    rho=0.0;
    clock=0.0;
    no_pkts_departed = 0.0;
    total_delay=0.0;
```

```
    average_delay=0.0;
    flag=1.0;
    next_event_time = 0.0;
    rho=arrival_rate * 1000.0 * net->max_stations / RATE;
    if (rho >=1.0)
       {
       cout <<"Traffic intensity is too high"<<"\n";
       exit(1);
       }
    for(i = 0;i<net->max_stations; i++)
       {
       net->Set_queue_size(i,0);
       for (j=0;j<MAX_Q_SIZE;j++)
             net->Set_start_time(i,j,0.0);
       }
}
}
                  ////////////////////////////
void Simu::Run()
{
int i, j;
int temp_stn,next;
if (mymedium->ring_or_bus ==1) // RING
    {
    ring_size=net->max_stations;
    walk_time=mymedium->token_time + mymedium->stn_latency + mymedium->tau/net->max_stations;
    }
else // BUS
    {
    ring_size=0;
    walk_time= mymedium->token_time +  mymedium->tau/3.0;
    }
for(i=0;i<net->max_stations;i++)
    {
    if(mymedium->ring_or_bus ==1) // RING
    {
    net->Set_next_stn(net->max_stations-1,0);
    net->Set_previous_stn(0,net->max_stations-1);
    net->Set_previous_stn(net->max_stations-1,net->max_stations-2);
    net->Set_next_stn(0,1);
       if ((i<(net->max_stations-1)) && (i>0))
          {
          net->Set_next_stn(i,i+1);
          net->Set_previous_stn(i,i-1);
          }
    }
    else
    {
    net->Set_in(i,0);
    net->Set_next_stn(i,-1);
    net->Set_previous_stn(i,-1);
    }
    }
for(i=0;i<(net->max_stations);i++)
    {
    for (j=0;j<3;j++)
       {
       net->Set_event_time(i,j,0.0);
       if (j!=0)
          net->Set_event_time(i,j,infinite);
       }
    }

while (no_pkts_departed < MAX_PACKETS)
    {
    next_event_time=infinite;
    for(i=0;i<(net->max_stations);i++)
       {
       for(j=0;j<3;j++)
          {
          if (next_event_time >  net->Get_event_time(i,j))
             {
             next_event_time = net->Get_event_time(i,j);
             next_station=i;
             next_event=j;
             }
          }
       }
    clock=next_event_time;
    if (next_event > 2)
       {
       cout <<"Check the Event list";
       exit(1);
       }
    // SCAN THE EVENT LIST
    switch (next_event)
       {
       case 0:// arrival event
          {
```

```
          net->Inc_queue_size(next_station);
          if (mymedium->ring_or_bus ==1) //RING
             {
             if (flag == 1.0)
                {
                flag = 0.0;
                net->Set_event_time(next_station,2,clock);
                }
             }
          else
             {
             if (flag==1.0)
                {
                flag=0.0;
                ring_size=1;
                net->Set_in(next_station,1);
                net->Set_next_stn(next_station,next_station);
                net->Set_previous_stn(next_station,next_station);
                net->Set_event_time(next_station,2,clock);
                }
             }

          // schedule for next arrival
          for (;;)
             {
             x=(float) rand();
             if (x!=0.0) break;
             }
          logx = -log(x/rand_size) * FACTOR / arrival_rate;
          net->Set_event_time(next_station, next_event,(clock+logx));
          net->Set_start_time(next_station, (net->Get_queue_size(next_station) -1),
                                                                            clock);
          break;
          }
       case 1 : // departure event
          {
          net->Dec_queue_size(next_station);
          no_pkts_departed ++;
          delay=clock - net->Get_start_time(next_station,0);
          total_delay +=delay;
          // push the queue forward
          for(i=0;i<net->Get_queue_size(next_station);i++)
             net->Set_start_time(next_station,i,
                                          net->Get_start_time(next_station,i+1));
          net->Set_start_time(next_station,net->Get_queue_size(next_station),0.0);
          net->Set_event_time(next_station,next_event,infinite);
          if (mymedium->ring_or_bus == 0)
             {
             stn_to_add =-1;
             for( i=next_station+1;i<net->max_stations;i++)
                {
                if((net->Get_queue_size(i)>0) && (net->Get_in(i)==0))
                      stn_to_add=i;
                if (stn_to_add !=-1) continue;
                }
             if (stn_to_add == -1)
                {
                for(i=0; i<next_station -1; i++)
                   {
                   if   ((net->Get_queue_size(i)>0) && (net->Get_in(i) ==0))
                      stn_to_add=i;
                   if (stn_to_add !=-1) continue;
                   }
             }
          if (stn_to_add !=-1)
             {
             temp_stn = net->Get_next_stn(next_station);
             net->Set_next_stn(next_station, stn_to_add);
             net->Set_next_stn(stn_to_add, temp_stn);
             net->Set_previous_stn(stn_to_add,next_station);
             net->Set_previous_stn(temp_stn,stn_to_add);
             ring_size++;
             net->Set_in(stn_to_add,1);
             }
          if (net->Get_queue_size(next_station)==0)
             {
             ring_size--;
             net->Set_in(next_station,0);
             if (ring_size==0)
                {
                net->Set_next_stn(next_station,-1);
                net->Set_previous_stn(next_station,-1);
                flag=1.0;
                }
             else
                {
                next=net->Get_next_stn(next_station);
                net->Set_event_time(next,2,clock+walk_time);
                net->Set_next_stn(net->Get_previous_stn(next_station),
```

```
                                        net->Get_next_stn(next_station));
                   net->Set_previous_stn(next,net->Get_previous_stn(next_station));
                   }
                 }
             else // queue size not 0
                 {
                 next=net->Get_next_stn(next_station);
                 net->Set_event_time(next,2,clock+walk_time);
                 }
              }
             if (mymedium->ring_or_bus ==1) //RING
                 {
                 next=net->Get_next_stn(next_station);
                 if (( next==0) && (net->Get_queue_size(next_station) == 0))
                    {
                    temp_flag =1 ;
                    for(i=0; i<net->max_stations; i++)
                       {
                       if (net->Get_queue_size(i) != 0)
                          {
                          net->Set_event_time(next,2,clock+walk_time);
                          temp_flag=0;
                          break;
                          }
                       }
                    if (temp_flag ==1)
                       {
                       flag = 1.0;
                       net->Set_event_time(next,2,infinite);
                       }
                    }
                 else
                    {
                    net->Set_event_time(next,2,clock+walk_time);
                    }
             break;
             }
             break;
             }
        case 2: // This is a token arrival event
            {
            net->Set_event_time(next_station,2,infinite);
            if (net->Get_queue_size(next_station) >0)
                {
                         net->Set_event_time(next_station,1,clock + mymedium->packet_time);
                }
            else
                cout <<"There is something wrong (BUS)";

                // assuming bus
            break;
            }
         } // end of switch
        } // end of while
         if (no_pkts_departed == 0.0)
            {
            average_delay =0;
            cout << "wrong answer\n";
            }
         else
            average_delay= total_delay /(no_pkts_departed *FACTOR);
         delay_ci[ic]=average_delay;

  }

void Simu::Result()
{
int i, j;
          for (i=0;i<10;i++)
              {
              ic=-1;
              Increase_Arrival_Rate();
              for (j=0;j<=degrees_fr;j++)
                 {
                 Increase_ic_index();
                 Run();
                 }

              delay_sum =0.0;
              delay_sqr = 0.0;
              for (ic=0;ic<=degrees_fr;ic++)
                 {
                 delay_sum +=delay_ci[ic];
                 delay_sqr += pow(delay_ci[ic], 2.0);
                 }
              delay_sum = delay_sum / (degrees_fr +1);
              delay_sqr = delay_sqr / (degrees_fr +1);
              delay_var =delay_sqr - pow(delay_sum, 2.0);
              delay_sdv = sqrt(delay_var);
```

APPENDIX C Network Simulation Software


```
                delay_con_int = delay_sdv * T_dist_par[degrees_fr-1]/sqrt (degrees_fr);
                cout << " For an arrival rate of " << arrival_rate << " the average delay is "
                << delay_sum << "+/-" << delay_con_int << "\n";
            }
}

void main (void)
{

Simu * mysimulation;
cout << "Starting simulation" << "\n";

mysimulation = new Simu();
mysimulation->Result();
delete mysimulation;
cout << "*******************END*********************"<< "\n";
getch();

}
```

Appendix D   Random number generation

This presents different ways of generating a random number X, from its probability distribution F(x). Two techniques commonly used are:

1. Inverse transformation (or direct method)

This is based by inverting the cumulative probability function $F(x) = P(X \leq x)$, which is associated with the random variable X. We know that $0 \leq F(x) \leq 1$.

By generating a random number U uniformaly distributed between 0 and 1, we can produce a random sample X from the distribution by inversion:

$$U = F(x)$$

$$X = F^{-1}(U)$$

$$\text{e.g. if } F(x) = 1 - e^{-x/\mu} \text{ with } 0 < x < \infty$$

$$\text{then } X = -\mu \ln(1 - U)$$

Assuming that the inverse transformation exists, this method is good. However, there is a problem if it does not exist, as in a Gaussian distribution.

2. Rejection method

This method can be applied to any bounded variable. With the probability density function of the random variable noted as $f(x)$.

$$\text{letting } f(x) = 0 \text{ for } a > x > b \quad \text{and} \quad f(x) \leq M$$

It is possible to generate random variates by

a) generating two random numbers U1 and U2 in the interval (0,1)

b) computing two random numbers with uniform distribution in (a,b) and (0,M) respectively so that :

X=a+(b-a)U1 (scale on the X axis)

Y=U2 M (scale on the Y axis)

c) if Y<=f(X1) accept X the next random variate otherwise reject and go back to a)

All points falling above $f(x)$ are rejected, and the points falling on or below are utilised to generate X.

APPENDIX E Simulation Self tuning software

selftune.c

```
// File : selftune.c   Author : S.Rolland September '97
// selftuning PID simulation software

//Other module included is matrix computation library (matlib)
#include "matlib.h"

#include <dos.h>
#define TRUE   1
#define FALSE 0


#define N 5

//-------------------

// Main procedure

//-------------------

void main()
{
//variable declarations
float ** theta_k,**phi_k,**theta_old,**phi_old,**P,**P_old,**L,**temp,**temp2,**P_temp,**P_temp2;
float ** used_theta,** used_phi,** P_temp3,**P_temp4,**phi_temp;
float ** phi_log3,** phi_log4,** phi_log5,** phi_log6;
static float den;
static float lamba=0.99;
static float estimate_error;
static float dyk;//current output
static double wo=10e7;
static double A,B,C,delta,root1,root2;
int i,j;
int tint;
float alpha1,alpha2,b0,b1,b2;// the actual parameters to estimate
float d; // estimate values
float temp_float1,temp_float2;
FILE *in;
FILE *pid;
float newy=0.0;
float epsilon;
float a1,a2,tau,delta2,mu,h;
float K,Ti,Td;
float PID_error,PID_output,PID_integral,PID_derivative,PID_old_error;
float *f_pointer;
float tf;
int RLS_step,old_RLS_step;
unsigned char PID_ON;
static float uk;
static float old_uk;
float command[10];
float output[10];
unsigned char locked;

// variable initialisation
locked = FALSE;
_stklen=0x2000;
command[1]=-1;
output[1]=-1;
A=0;
B=0;
C=0;
delta=0;
a1=0;
a2=0;
tau=0;
mu=0;
d=1;
h=0.04;

// file opening
// event.log logs parameters estimation values
if ((in = fopen("event.log", "wt")) == NULL)
    {
        fprintf(stderr, "Cannot open input file.\n");
        exit(1);
    }
// pid.log logs the PID coefficients and output values
if ((pid = fopen("pid.log", "wt"))  == NULL)
    {
        fprintf(stderr, "Cannot open pid file.\n");
        exit(1);
    }
// values used for generating model response
alpha1=0.975;
b0=0;
b1=2.56;
b2 =0;
// matrix memory allocation
printf("\nThe is stack: %u\tstack pointer: %u", stackavail(), _SP);
```

APPENDIX E Simulation Self tuning software

selftune.c

```
used_theta=matrix(1,1,1,N);
theta_k=matrix(1,1,1,N);
theta_old=matrix(1,1,1,N);
used_phi=matrix(1,N,1,1);
phi_k=matrix(1,N,1,1);
phi_old=matrix(1,N,1,1);
phi_log3=matrix(1,N,1,1);
phi_log4=matrix(1,N,1,1);
phi_log5=matrix(1,N,1,1);
phi_log6=matrix(1,N,1,1);
phi_temp=matrix(1,N,1,1);
L=matrix(1,1,1,N);
P=matrix(1,N,1,N);
P_old=matrix(1,N,1,N);
temp=matrix(1,1,1,N);
temp2=matrix(1,1,1,1);
P_temp=matrix(1,N,1,N);
P_temp2=matrix(1,N,1,N);
P_temp3=matrix(1,N,1,N);
P_temp4=matrix(1,N,1,N);
// matrix initialisation
zero(P,1,N,1,N);
eye(P,1,N,1,N);
zero(P_old,1,N,1,N);
eye(P_old,1,N,1,N);
zero(theta_k,1,1,1,N);
zero(theta_old,1,1,1,N);
zero(L,1,1,1,N);

theta_k[1][4]=1; // b1!=0
theta_old[1][4]=1; // b1!=0
theta_k[1][1]=1;
theta_old[1][1]=1;
theta_k[1][2]=1;
theta_old[1][2]=1;
used_theta[1][1]=-alpha1;
used_theta[1][2]=b2;
used_theta[1][3]=b1;
used_theta[1][4]=b0;
used_theta[1][5]=0;
zero(used_phi,1,N,1,1);
zero(phi_k,1,N,1,1);
zero(phi_old,1,N,1,1);
phi_k[N][1]=1;
phi_old[N][1]=1;
zero(L,1,1,1,N);
zero(temp,1,1,1,N);
zero(temp2,1,1,1,1);
printf("\nNow, the stack: %u\tstack pointer: %u", stackavail(), _SP);

//PID coeffs default values
K=0.5;
Ti=0.1;
Td=0.01;
PID_integral=0;
PID_error=0;
PID_old_error=0;
PID_derivative=0;

RLS_step=0;
// write header line in files
fprintf(pid,"K Ti Td uk dyk PID_output");
fprintf(in,"Step dyk esterror estimate den espilon theta1 2 3 4 5 6 d");
for(j=0;j<2000;j++) // loop for 800 steps
{


    if (j==250)          //  introduce delay at step 250 by shifted used model
       {

       used_theta[1][1]=-alpha1;
       used_theta[1][2]=0;
       used_theta[1][3]=b2;
       used_theta[1][4]=b1;
       used_theta[1][5]=0;
       }
    if (j==650)          // back to original st step 450
       {
       used_theta[1][1]=-alpha1;
       used_theta[1][2]=b2;
       used_theta[1][3]=b1;
       used_theta[1][4]=b0;
       used_theta[1][5]=0;
       }
    if (j==0)  // command changes
       {
       uk=1000;
       }
    else if (j==250)
```

APPENDIX E Simulation Self tuning software

selftune.c

```
        {
        uk=2000;
        }
     else if (j==350)     {
        uk=3000;
        }

     else if (j==450)
        {
        uk=1000;
        }
     else if (j==650)
        {
        uk=4000;
        }
     else if (j==850)
        {
        uk=3000;
        }

     else if (j==1050)
        {
        uk=1000;
        }
     else if (j==1250)
        {
        uk=2000;
        }
     else if (j==1450)
        {
        uk=3000;
        }
     else if (j==1650)
        {
        uk=1000;
        }
     else if (j==1850)
        {
        uk=2000;
        }
     else if (j==2050)
        {
        uk=3000;
        }
// PID calculations
if (locked == TRUE) // only uses PID when RLS estimator has converged
{
     printf("\n%+e %+e %e  " ,K,Ti,Td);
     PID_old_error=PID_error;
     PID_error=uk-dyk;
     PID_integral= PID_error + PID_integral; // wind up needed????
     if (PID_integral>5000)
        PID_integral=5000;
     if (PID_integral<-5000)
        PID_integral=-5000;
     PID_derivative=(PID_error-PID_old_error);
     fprintf(pid," PID_integral= %e ",PID_integral);
     PID_output=  (fabs(K)*PID_error)+((h*PID_integral)/(fabs(Ti)*10))
                                 +(rand()*10/RAND_MAX);;

     }
     else
     {
        PID_output=(rand()*10/RAND_MAX);;
        fprintf(pid, " NO PID");
     }
     fprintf(pid,"\n%+e %+e %e %+e %+e %+e = %e +%e +random  "
   ,K,Ti,Td,uk,dyk,PID_output,(fabs(K)*PID_error*h),((PID_integral *h)/(fabs(Ti)*10)));
// open loop force PID_output=uk

     printf("\n %d:",j);
     fprintf(in,"\n%d",j);

     copy_m(phi_k,phi_temp,N,1);
     pmm(used_theta,phi_temp,1,1,1,N,1,N,1,1);
     dyk=phi_temp[1][1];
     fprintf(in," %e ",dyk);
// calculate estimation error
     tint=(int)d;
     switch(tint)
        {
        case 1:
           copy_m(phi_k,phi_temp,N,1);
           break;
        case 2:
           copy_m(phi_old,phi_temp,N,1);
           break;
        case 3:
           copy_m(phi_log3,phi_temp,N,1);
```

APPENDIX E Simulation Self tuning software

selftune.c

```
            break;
         case 4:
            copy_m(phi_log4,phi_temp,N,1);
            break;
         case 5:
            copy_m(phi_log5,phi_temp,N,1);
            break;
         case 6:
            copy_m(phi_log6,phi_temp,N,1);
            break;
         }
     pmm(theta_k,phi_temp,1,1,1,N,1,N,1,1);
     estimate_error=dyk-phi_temp[1][1];
     fprintf(in," %e %e",estimate_error,phi_temp[1][1]);
// if estimate outside +/- 2% band, RLS should be on
old_RLS_step=RLS_step;
     if ((fabs(estimate_error)>fabs(dyk)*0.02) && (RLS_step<50))
         {
         RLS_step++;
         locked = FALSE;
         lamba=pow(0.99,RLS_step);
         P_old=P;
         tint=(int)d;
         switch(tint)
            {
            case 1:
                copy_m(phi_k,P_temp3,N,1);
                break;
            case 2:
                copy_m(phi_old,P_temp3,N,1);
                break;
            case 3:
                copy_m(phi_log3,P_temp3,N,1);
                break;
            case 4:
                copy_m(phi_log4,P_temp3,N,1);
                break;
            case 5:
                copy_m(phi_log5,P_temp3,N,1);
                break;
            case 6:
                copy_m(phi_log6,P_temp3,N,1);
                break;
            }
         ptranspose(P_temp3,1,N,1,1);
         copy_m(P_old,P_temp2,N,N);
         pmm(P_temp3,P_temp2,1,1,1,N,1,N,1,N);
         if (d==1)
            copy_m(phi_k,phi_temp,N,1);
         else
            copy_m(phi_old,phi_temp,N,1);
         pmm(P_temp2,phi_temp,1,1,1,N,1,N,1,1);
         den=lamba+phi_temp[1][1];
         fprintf(in," %e",den);
         if (d==1)
            copy_m(phi_k,phi_temp,N,1);
         else
            copy_m(phi_old,phi_temp,N,1);
         pmm(P_old,phi_temp,1,N,1,N,1,N,1,1);
         if (den!=0)
            {
            *f_pointer=1/den;
            pkm(phi_temp,f_pointer,1,N,1,1);
             }
         else
            fprintf(in,"Error den=0");
         temp=theta_k;
         *f_pointer=estimate_error;
         pkm(phi_temp,f_pointer,1,N,1,1);
         theta_k=transpose(phi_temp,1,N,1,1);
         theta_old=temp;
         paddm(theta_old,theta_k,1,1,1,N,1,1,1,N);
         //calculate new P
         //P=(P_old-((P_old*phi_k*phi_k'*P_old)/(lambda+phi_k'*P_old*phi_k)))/lamda
         if (d==1)
            copy_m(phi_k,phi_temp,N,1);
         else
            copy_m(phi_old,phi_temp,N,1);
         pmm(P_old,phi_temp,1,N,1,N,1,N,1,1);
         if (d==1)
            copy_m(phi_k,P_temp2,N,1);
         else
            copy_m(phi_old,P_temp2,N,1);
         ptranspose(P_temp2,1,N,1,1);
         pmm(phi_temp,P_temp2,1,N,1,1,1,1,1,N);
         copy_m(P_old,P_temp3,N,N);
         pmm(P_temp2,P_temp3,1,N,1,N,1,N,1,N);
         if (den!=0)
            {
```

APPENDIX E Simulation Self tuning software

selftune.c

```
            * f_pointer=-1/den;
            pkm(P_temp3,f_pointer,1,N,1,N);
            }
        else
            printf("den=0");
        paddm(P_old,P_temp3,1,N,1,N,1,N,1,N);
        if (lamba !=0)
            {
            *f_pointer = 1/lamba;
            pkm(P_temp3,f_pointer,1,N,1,N);
            copy_m(P_temp3,P,N,N);
            }
        else
            printf("lamba=0");
    }// end of RLS
// estimate is within 2% no RLS
else
    {
// reset RLS parameters
    locked = TRUE;
    printf(" NO-RLS");
    fprintf(in," NORLS");
    zero(P,1,N,1,N);
    eye(P,1,N,1,N);
    zero(P_old,1,N,1,N);
    eye(P_old,1,N,1,N);
    zero(L,1,1,1,N);
    RLS_step=0;
    }

// Shifting Phi k record
copy_m(phi_log5,phi_log6,N,1);
copy_m(phi_log4,phi_log5,N,1);
copy_m(phi_log3,phi_log4,N,1);
copy_m(phi_old,phi_log3,N,1);
copy_m(phi_k,phi_old,N,1);
if (RLS_step ==1)
    {
    zero(phi_k,1,N,1,1);
    zero(phi_old,1,N,1,1);
    phi_k[N][1]=1;
    phi_old[N][1]=1;
    phi_k[2][1]= PID_output;
    phi_k[1][1]=-dyk;
    }
else
    {
    phi_k[5][1]=phi_k[4][1];
    phi_k[4][1]=phi_k[3][1];
    phi_k[3][1]=phi_k[2][1];
    phi_k[2][1]=PID_output;
    phi_k[1][1]=-dyk;
    }

phi_log6[1][1]=-dyk;
phi_log5[1][1]=-dyk;
phi_log4[1][1]=-dyk;
phi_log3[1][1]=-dyk;
phi_old[1][1]=-dyk;
fprintf(in," %e %e %e %e %e ",theta_k[1][1],theta_k[1][2],theta_k[1][3],theta_k[1][4],theta_k[1][5]);
fprintf(in,"d= %e",d);
fprintf(in," phi= %e %e %e %e %e",phi_k[1][1],phi_k[2][1],phi_k[3][1],phi_k[4][1],phi_k[5][1]);

epsilon=(theta_k[1][4]-theta_k[1][2])/(theta_k[1][4]+theta_k[1][3]+theta_k[1][2]);
fprintf(in," %e",epsilon);

if ((epsilon < - 0.8) && (RLS_step==0)) // if RLS has locked on and epsilon<-0.8
    {
    printf("< -0.8");
    fprintf(in," <-0.8");
    temp_float1=theta_k[1][4];//b0
    d=d-1;
    theta_k[1][4]=theta_k[1][3]+3*temp_float1; //b0=b1+3b0
    theta_k[1][3]=theta_k[1][2]-3*temp_float1; //b1=b2-3b0
    theta_k[1][2]=temp_float1; //b2=b0
    }
else  if ((epsilon > 0.8) && (RLS_step==0))// if RLS has locked on and epsilon>0.8
    {
    printf("> 0.8");
    fprintf(in,">0.8");
    d=d+1;
    temp_float1=theta_k[1][3]; //b1
    temp_float2=theta_k[1][2]; //b2
    theta_k[1][3]=theta_k[1][4]-3*temp_float2;//b1=b0-3b2
    theta_k[1][2]=temp_float1+3*temp_float2;//b2=b1+3b2
    theta_k[1][4]=temp_float2;//b0=b2
    }

tau=h*(d+epsilon);
```

APPENDIX E Simulation Self tuning software

selftune.c

```
// calculate PID coeffs
if (theta_k[1][1]<0)
    a1=-log(-theta_k[1][1])/h;
if ((1+theta_k[1][1]) !=0)
    {
    mu=(theta_k[1][2]+theta_k[1][3]+theta_k[1][4])/(1+theta_k[1][1]);
        fprintf(in," %e/%e",(theta_k[1][2]+theta_k[1][3]+theta_k[1][4]),(1+theta_k[1][1]));
        }
else
    fprintf(in," NOMU");
// Using Haalman tuning rules
if (a1!=0)
    {
    Ti=(1/a1);
    Td=0; // PI only
    K=2/(a1*6*mu*tau);
    fprintf(in," theta_k[1][1]=%e a1=%e mu=%e Ti=%e\tTd=%e\tK=%e",theta_k[1][1],a1,mu,Ti,Td,K);
    }
else
    fprintf(in," skip");


}
// end for loop
// cleaning memory and closing files
fclose(in);
fclose(pid);
free_matrix(used_theta,1,1,1,N);
free_matrix(theta_k,1,1,1,N);
free_matrix(theta_old,1,1,1,N);
free_matrix(phi_k,1,N,1,1);
free_matrix(phi_old,1,N,1,1);
free_matrix(phi_log3,1,N,1,1);
free_matrix(phi_log4,1,N,1,1);
free_matrix(phi_log5,1,N,1,1);
free_matrix(phi_log6,1,N,1,1);
free_matrix(used_phi,1,N,1,1);
free_matrix(P,1,N,1,N);
free_matrix(P_old,1,N,1,N);
free_matrix(L,1,1,1,N);
free_matrix(temp,1,1,1,N);
free_matrix(temp2,1,1,1,1);
free_matrix(P_temp,1,N,1,N);
free_matrix(P_temp2,1,N,1,N);
free_matrix(P_temp3,1,N,1,N);
free_matrix(P_temp4,1,N,1,N);
free_matrix(phi_temp,1,N,1,1);


printf(" end.");
getch();
exit(1);
}
```

APPENDIX F Matrix Library

matlib.h

```
// File : matlib.h   Author : S.Rolland     August '97
// matrix libary header file
#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
// standard error handler
void nrerror(char error_text[]);
// allocates a float vector range [nl..nh]
float *vector(int nl,int nh);
//allocates an int vector range [nl..nh]
int *ivector(int nl,int nh);
//allocates a double vector range [nl..nh]
double*dvector(int nl,int nh);
// allocates a float matrix with rane [nrl..nrh][ncl..nch]
float **matrix(int nrl,int nrh,int ncl,int nch);
// allocates an int matrix with rane [nrl..nrh][ncl..nch]
int **imatrix(int nrl,int nrh,int ncl,int nch);
// allocates a double matrix with rane [nrl..nrh][ncl..nch]
double **dmatrix(int nrl,int nrh,int ncl,int nch);
// returns a submatrix with range [newrl..newrl+(oldrh-oldrl)] [newcl..newcl+(oldch-oldcl)]
float **submatrix(float **a,int oldrl,int oldrh,int oldcl,int oldch,int newrl,int newcl);
//frees a float vector
void free_vector(float*v,int nl,int nh);
//frees an int vector
void free_ivector(int*v,int nl,int nh);
//frees a double vector
void free_dvector(double*v,int nl,int nh);
//frees a matrix
void free_matrix(float **m ,int nrl,int nrh,int ncl,int nch);
//frees an int matrix
void free_imatrix(int **m ,int nrl,int nrh,int ncl,int nch);
//frees a double matrix
void free_dmatrix(double **m ,int nrl,int nrh,int ncl,int nch);
// frees a sumatrix
void free_submatrix(float **b,int nrl,int nrh,int ncl,int nch);
//allocate a float matrix that points to the matrix a
float ** convert_matrix(float *a,int nrl,int nrh,int ncl,int nch);
//frees a matrix allocated by covert_matrix()
void free_convert_matrix(float **b,int nrl,int nrh,int ncl,int nch);
// copies contents of a matrix to the other
// dest can be larger than dest
void copy_m(float **src,float **dest,int nr,int nc);
// mukltiplies two matrices , result returned in b
// must not multiply two same matrices
void pmm(float**a,float**b,int nral,int nrah,int ncal,int ncah,int nrbl,int nrbh,int ncbl,int ncbh);
// adds two matrices, result returned in b
void paddm(float**a,float**b,int nral,int nrah,int ncal,int ncah,int nrbl,int nrbh,int ncbl,int ncbh);
// multiplies matrix by float
// float passed as a pointer
//result returned in a
void pkm(float**a,float* f,int nral,int nrah,int ncal,int ncah);
// calculates determinant of 2X2 matrix
float det22(float**a,int nral,int nrah,int ncal,int ncah);
// fills a matrix with zero
void zero(float**a,int nral,int nrah,int ncal,int ncah);
// fills a matrix with identity matrix
void eye(float**a,int nral,int nrah,int ncal,int ncah);
// calculates cofactor
float cofactor(float**a,int nral,int nrah,int ncal,int ncah,int i,int j);
// calculates determinant
float det (float**a,int nral,int nrah,int ncal,int ncah);
// transposes a matrix
// result returned as pointer to float
// WARNING : this function allocates memory that isn't freed afterwards
// use ptranspose instead
float **transpose(float**a,int nral,int nrah,int ncal,int ncah);
// transposes a matrix
// result returned in a
void ptranspose(float**a,int nral,int nrah,int ncal,int ncah);
// calculate adjoint matrix
float **adjoint(float**a,int nral,int nrah,int ncal,int ncah);
// calculate inverse matrix
float **inverse(float**a,int nral,int nrah,int ncal,int ncah);
// displays  matrix contents on screen
void display(float **m,int nrl,int nrh,int ncl,int nch);
// displays matrix contenets on file
void fdisplay(FILE *stream,float **m,int nrl,int nrh,int ncl,int nch);
```

APPENDIX F Martrix Library

matlib.c

```
// File : matlib.c   Author : S.Rolland August '97
// matrix libary
#include "matlib.h"

// standard error handler
void nrerror(char error_text[])
{
fprintf(stderr,"Matrix calculation run time error\n");
fprintf(stderr,"%s\n",error_text);
fprintf(stderr,"...Now exiting system...\n");
getch();
exit(1);
}
// allocates a float vector range [nl..nh]
float *vector(int nl,int nh)
{
float *v;
v=(float*)malloc((unsigned )(nh-nl+1)*sizeof(float));
if (!v)nrerror("allocation failure in vector()");
return v-nl;
}
//allocates an int vector range [nl..nh]
int *ivector(int nl,int nh)
{
int *v;
v=(int*)malloc((unsigned )(nh-nl+1)*sizeof(int));
if (!v)nrerror("allocation failure in ivector()");
return v-nl;
}
//allocates a double vector range [nl..nh]
double*dvector(int nl,int nh)
{
double *v;
v=(double*)malloc((unsigned )(nh-nl+1)*sizeof(double));
if (!v)nrerror("allocation failure in dvector()");
return v-nl;
}

// allocates a float matrix with rane [nrl..nrh][ncl..nch]
float **matrix(int nrl,int nrh,int ncl,int nch)
{
int i;
float **m;
// rows
m=(float**)malloc((unsigned)(nrh-nrl+1)*sizeof(float*));
if (!m) nrerror("allocation failure 1 in matrix()");
m-=nrl;
for(i=nrl;i<=nrh;i++){
    m[i]=(float*)malloc((unsigned)(nch-ncl+1)*sizeof(float));
    if (!m[i])nrerror("allocation failure 2 in matrix()");
    m[i]-=ncl;
    }
return m;
}
// allocates an int matrix with rane [nrl..nrh][ncl..nch]
int **imatrix(int nrl,int nrh,int ncl,int nch)
{
int i;
int **m;
// rows
m=(int**)malloc((unsigned)(nrh-nrl+1)*sizeof(int*));
if (!m) nrerror("allocation failure 1 in imatrix()");
m-=nrl;
for(i=nrl;i<=nrh;i++){
    m[i]=(int*)malloc((unsigned)(nch-ncl+1)*sizeof(int));
    if (!m[i])nrerror("allocation failure 2 in imatrix()");
    m[i]-=ncl;
    }
return m;
}
// allocates a double matrix with rane [nrl..nrh][ncl..nch]
double **dmatrix(int nrl,int nrh,int ncl,int nch)
{
int i;
double **m;
// rows
m=(double**)malloc((unsigned)(nrh-nrl+1)*sizeof(double*));
if (!m) nrerror("allocation failure 1 in dmatrix()");
m-=nrl;
for(i=nrl;i<=nrh;i++){
    m[i]=(double*)malloc((unsigned)(nch-ncl+1)*sizeof(double));
    if (!m[i])nrerror("allocation failure 2 in dmatrix()");
    m[i]-=ncl;
    }
return m;
}
// returns a submatrix with range [newrl..newrl+(oldrh-oldrl)] [newcl..newcl+(oldch-oldcl)]
float **submatrix(float **a,int oldrl,int oldrh,int oldcl,int oldch,int newrl,int newcl)
{
```

APPENDIX F Martrix Library

matlib.c

```
int i,j;
float **m;
m=(float**)malloc((unsigned)(oldrh-oldrl+1)*sizeof(float*));
if (!m)nrerror("allocation failure in submatrix()");
m-=newrl;
for(i=oldrl,j=newrl;i<=oldrh;i++,j++)
    m[j]=a[i]+oldcl-newcl;
return m;
}

//frees a float vector
void free_vector(float*v,int nl,int nh)
{
free((char*)(v+nl));
}

//frees an int vector
void free_ivector(int*v,int nl,int nh)
{
free((char*)(v+nl));
}

//frees a double vector
void free_dvector(double*v,int nl,int nh)
{
free((char*)(v+nl));
}
//frees a matrix
void free_matrix(float **m ,int nrl,int nrh,int ncl,int nch)
{
int i;
for(i=nrh;i>=nrl;i--)
    free((char*) (m[i]+ncl));
free ((char*)(m+nrl));
}
//frees an int matrix
void free_imatrix(int **m ,int nrl,int nrh,int ncl,int nch)
{
int i;
for(i=nrh;i>=nrl;i--)
    free((char*) (m[i]+ncl));
free ((char*)(m+nrl));
}

//frees a double matrix
void free_dmatrix(double **m ,int nrl,int nrh,int ncl,int nch)
{
int i;
for(i=nrh;i>=nrl;i--)
    free((char*) (m[i]+ncl));
free ((char*)(m+nrl));
}

// frees a sumatrix
void free_submatrix(float **b,int nrl,int nrh,int ncl,int nch)
{
free ((char*)(b+nrl));
}
//allocate a float matrix that points to the matrix a
float ** convert_matrix(float *a,int nrl,int nrh,int ncl,int nch)
{
int i,j,nrow,ncol;
float **m;
nrow=nrh-nrl+1;
ncol=nch-ncl+1;
m=(float **) malloc((unsigned)(nrow)*sizeof(float*));
if (!m) nrerror("allocation failure in convert_matrix()");
m-=nrl;
for(i=0,j=nrl;i<=nrow-1;i++,j++)
    m[j]=a+ncol*i-ncl;
return m;
}

//frees a matrix allocated by covert_matrix()
void free_convert_matrix(float **b,int nrl,int nrh,int ncl,int nch)
{
free((char*)(b+nrl));
}
// copies contents of a matrix to the other
// dest can be larger than dest
void copy_m(float **src,float **dest,int nr,int nc)
{
int i,j;
float **m;
m=(float**)malloc((unsigned)(nr+1)*sizeof(float*));
if (!m) nrerror("allocation failure 1 in copym()");
m-=1;
for(i=1;i<=nr;i++){
    m[i]=(float*)malloc((unsigned)(nc+1)*sizeof(float));
```

matlib.c

```
    if (!m[i])nrerror("allocation failure 2 in copym()");
    }

for(i=1;i<=nr;i++)
    for(j=1;j<=nc;j++)
        {
        m[i][j]=src[i][j];
        }
for(i=1;i<=nr;i++)
    for(j=1;j<=nc;j++)
        {
        dest[i][j]=m[i][j];
        }
free_matrix(m,1,nr,1,nc);
}
// mukltiplies two matrices , result returned in b
// must not multiply two same matrices
void pmm(float**a,float**b,int nral,int nrah,int ncal,int ncah,int nrbl,int nrbh,int ncbl,int
ncbh)
{
int i;
int j;
int k;
int nra,ncb,nca;
float **m;
if ((ncah-ncal)!=(nrbh-nrbl))
    {
    printf("\n%d != %d\n",ncah-ncal,nrbh-nrbl);
    nrerror("Matrix Muliplication error");
    }
nca=ncah-ncal+1;
nra=nrah-nral+1;
ncb=ncbh-ncbl+1;


    // size of resulting matrix (nrah-nral+1) rows X (ncah-ncbk+1) columns
m=(float**)malloc((unsigned)(nra)*sizeof(float*));
if (!m) nrerror("allocation failure 1 in pmm()");
m-=nral;
for(i=nral;i<=nrah;i++){
    m[i]=(float*)malloc((unsigned)(ncb)*sizeof(float));
    if (!m[i])nrerror("allocation failure 2 in pmm()");
    }

for(i=1;i<=nra;i++)
    {
    for(j=1;j<=ncb;j++)
        {
        m[i][j]=0;
        for (k=1;k<=nca;k++)
            {
                m[i][j]=m[i][j]+(a[i][k]*b[k][j]);
            }
        }
    }
for(i=1;i<=nra;i++)
    for(j=1;j<=ncb;j++)
        b[i][j]=m[i][j];
free_matrix(m,1,nra,1,ncb);
}

// adds two matrices, result returned in b
void paddm(float**a,float**b,int nral,int nrah,int ncal,int ncah,int nrbl,int nrbh,int ncbl,int
ncbh)
{

int i,j;
int nra,nrb,ncb,nca;
float **m;
nca=ncah-ncal+1;
nra=nrah-nral+1;
ncb=ncbh-ncbl+1;
nrb=nrbh-nrbl+1;
if ((nra!=nrb) || (nca!=ncb))
    nrerror("Matrix Addition error error");


    // size of resulting matrix (nrah-nral+1) rows X (ncah-ncbk+1) columns
m=(float**)malloc((unsigned)(nra)*sizeof(float*));
if (!m) nrerror("allocation failure 1 in mm()");
m-=nral;
for(i=nral;i<=nrah;i++){
    m[i]=(float*)malloc((unsigned)(ncb)*sizeof(float));
    if (!m[i])nrerror("allocation failure 2 in mm()");
    }
for(i=1;i<=nra;i++)
    for(j=1;j<=ncb;j++)
        {
        m[i][j]=a[i][j]+b[i][j];
```

APPENDIX F Martrix Library

matlib.c

```
        }

for(i=1;i<=nra;i++)
    for(j=1;j<=ncb;j++)
        b[i][j]=m[i][j];
free_matrix(m,1,nra,1,ncb);

}
// multiplies matrix by float
// float passed as a pointer
//result returned in a
void pkm(float**a,float* f,int nral,int nrah,int ncal,int ncah)
{
int i;
int j;
int nra,nca;
float **m;
nca=ncah-ncal+1;
nra=nrah-nral+1;
    // size of resulting matrix (nrah-nral+1) rows X (ncah-ncbk+1) columns
m=(float**)malloc((unsigned)(nra)*sizeof(float*));
if (!m) nrerror("allocation failure 1 in mm()");
m-=nral;
for(i=nral;i<=nrah;i++){
    m[i]=(float*)malloc((unsigned)(nca)*sizeof(float));
    if (!m[i])nrerror("allocation failure 2 in mm()");
    }

for(i=1;i<=nra;i++)
    {
    for(j=1;j<=nca;j++)
        {
        m[i][j]=(a[i][j])*(*f);
        }
    }
for(i=1;i<=nra;i++)
    for(j=1;j<=nca;j++)
        a[i][j]=m[i][j];
free_matrix(m,1,nra,1,nca);

}
// calculates determinant of 2X2 matrix
float det22(float**a,int nral,int nrah,int ncal,int ncah)
{
float f;
f=(a[nral][ncal]*a[nrah][ncah])-(a[ncal][nrah]*a[ncah][nral]);
return f;
}

// fills a matrix with zero
void zero(float**a,int nral,int nrah,int ncal,int ncah)
{
int nra,nca,i,j;
nra=nrah-nral+1;
nca=ncah-ncal+1;
for(i=1;i<=nra;i++)
    for(j=1;j<=nca;j++)
        a[i][j]=0.0;
}
// fills a matrix with identity matrix
void eye(float**a,int nral,int nrah,int ncal,int ncah)
{
int nra,nca,i;
nra=nrah-nral+1;
nca=ncah-ncal+1;
if (nra!=nca)
    nrerror("Unable to create non square I matrix");
for(i=1;i<=nra;i++)
        a[i][i]=1.0e6;
}
// calculates cofactor
float cofactor(float**a,int nral,int nrah,int ncal,int ncah,int i,int j)
{
float f;
float**m;
int nra,nca;
int ii,jj,k,l;


nra=nrah-nral+1;
nca=ncah-ncal+1;

if ((nra>3) ||(nca>3))
    nrerror("Unable to find cofactor of big matrix");
m=matrix(1,nra-1,1,nca-1);
k=1;l=1;
for(ii=1;ii<=nra;ii++)
    {
    if (ii!=i)
```

matlib.c

```
            {
            l=1;
            for(jj=1;jj<=nca;jj++)
               {
               if (jj!=j)
                  {
                  m[k][l]=a[ii][jj];
                //    printf("\nminor is m[%d][%d]=%e %d %d",k,l,m[k][l],ii,jj);
                  l++;
                  }
               }
            k++;
            }
     }
f=pow(-1,i+j)*det22(m,1,2,1,2);
return f;
}
// calculates determinant
float det (float**a,int nral,int nrah,int ncal,int ncah)
{
int i;
float f;
int nra,nca;
nra=nrah-nral+1;
nca=ncah-ncal+1;
if (nra!=nca)
   nrerror("unable to calculate det of non square matrix");
f=0;
for(i=1;i<=nra;i++)
   f=f+a[i][1]*cofactor(a,nral,nrah,ncal,ncah,i,1);
return f;
}
// transposes a matrix
// result returned as pointer to float
// WARNING : this function allocates memory that isn't freed afterwards
// use ptranspose instead
float **transpose(float**a,int nral,int nrah,int ncal,int ncah)
{
int i;
int j;
float **m;
int nra,nca;
nra=nrah-nral+1;
nca=ncah-ncal+1;
m=(float**)malloc((unsigned)(nca)*sizeof(float*));
if (!m) nrerror("allocation failure 1 in transpose()");
m-=ncal;
for(i=ncal;i<=ncah;i++){
   m[i]=(float*)malloc((unsigned)(nra)*sizeof(float));
   if (!m[i])nrerror("allocation failure 2 in transpose()");
   }
for(i=1;i<=nra;i++)
   {
   for(j=1;j<=nca;j++)
      {
      m[j][i]=a[i][j];
      }
   }
return m;
}
// transposes a matrix
// result returned in a
void ptranspose(float**a,int nral,int nrah,int ncal,int ncah)
{
int i;
int j;
float **m;
int nra,nca;
nra=nrah-nral+1;
nca=ncah-ncal+1;
m=(float**)malloc((unsigned)(nca)*sizeof(float*));
if (!m) nrerror("allocation failure 1 in transpose()");
m-=ncal;
for(i=ncal;i<=ncah;i++){
   m[i]=(float*)malloc((unsigned)(nra)*sizeof(float));
   if (!m[i])nrerror("allocation failure 2 in transpose()");
   }
for(i=1;i<=nra;i++)
   {
   for(j=1;j<=nca;j++)
      {
      m[j][i]=a[i][j];
      }
   }
for(i=1;i<=nca;i++)
   {
   for(j=1;j<=nra;j++)
      {
      a[i][j]=m[i][j];
```

matlib.c

```
        }
    }

free_matrix(m,ncal,ncah,nral,nrah);
}

// calculate adjoint matrix
float **adjoint(float**a,int nral,int nrah,int ncal,int ncah)
{
int i;
int j;
float **ma;
int nra,nca;
nra=nrah-nral+1;
nca=ncah-ncal+1;
ma=(float**)malloc((unsigned)(nra)*sizeof(float*));
if (!ma) nrerror("allocation failure 1 in transpose()");
ma-=nral;
for(i=nral;i<=nrah;i++){
    ma[i]=(float*)malloc((unsigned)(nca)*sizeof(float));
    if (!ma[i])nrerror("allocation failure 2 in transpose()");
    }
for(i=1;i<=nra;i++)
    {
    for(j=1;j<=nca;j++)
        {
        ma[i][j]=cofactor(a,nral,nrah,ncal,ncah,i,j);
        }
    }
return(transpose(ma,nral,nrah,ncal,ncah));

}
// calculate inverse matrix
float **inverse(float**a,int nral,int nrah,int ncal,int ncah)
{
float **mm;
int nra,nca;
float f;
float * fp;
int i;
nra=nrah-nral+1;
nca=ncah-ncal+1;
mm=(float**)malloc((unsigned)(nra)*sizeof(float*));
if (!mm) nrerror("allocation failure 1 in inverse()");
mm-=nral;
for(i=nral;i<=nrah;i++){
    mm[i]=(float*)malloc((unsigned)(nca)*sizeof(float));
    if (!mm[i])nrerror("allocation failure 2 in inverse()");
    }

f=1/det(a,nral,nrah,ncal,ncah);
*fp=f;
mm=adjoint(a,nral,nrah,ncal,ncah);
pkm(mm,fp,nral,nrah,ncal,ncah);
return(mm);

}
// displays  matrix contents on screen
void display(float **m,int nrl,int nrh,int ncl,int nch)
{
int i,j;
for (i=1;i<=nrh-nrl+1;i++)
    {
    printf("\n Row %d : ",i);
    for(j=1;j<=nch-ncl+1;j++)
        printf(" %e\t",m[i][j]);
    }
}
// displays matrix contents on file
void fdisplay(FILE *stream,float **m,int nrl,int nrh,int ncl,int nch)
{
int i,j;
for (i=1;i<=nrh-nrl+1;i++)
    {
    fprintf(stream,"\n Row %d : ",i);
    for(j=1;j<=nch-ncl+1;j++)
        {
        fprintf(stream," %e\t",m[i][j]);
        }
    }
}
```

APPENDIX G Experimental Estimation software

estbase.c

```c
// File : estbase.c   Author : S.Rolland     August '97
//Estimator software for experimental trials

#include "c:\stef\control\matlib.h"

#include "sdlc.h"
#include <stdio.h>
#include <conio.h>
#include <float.h>
#include <math.h>
#include <dos.h>
#include <process.h>
#include "psdrv.h"
#include "token.h"

#define OFF FALSE
#define ON TRUE

#define MAX_ESTIMATION_ERROR  15.0
#define MIN_ESTIMATION_ERROR  -15.0

#define INIT_CETREK 0x09
#define SETUP_UART 0x0A
#define DATA_HEADER 0x0B
#define STATUS_REQ 0x0C
#define VIDEO 0x0D
#define RELAY_COMM 0x01
#define PAN_COMM 0x02
#define TILT_COMM 0x03
#define LIGHT_COMM 0x04

#define N 5
/*_______________________________________________
                      Global Variables
_________________________________________________*/
extern long Inactivity_timer;
extern long Trt_timer;
char  msg[30];
signed char RECV_BUFFER[64];
int RBL;
int heading, depth;
struct  time t; /* time structure*/
int TTRT; /*TTRT in msec*/
int INACTIVITY_TIME_OUT ;
int TTRT_PC_TICKS;
unsigned char speed,th,tl,command;
float Small_Delay_Test;
FILE *log_file;
int setpoint;
float calculated_speed;
int error;
int old_error;
float derivitive;
float integral;
float PIDcommand;
unsigned char Tilt_Com;
float f_tilt_com;
float temp_k,temp_Ti;
extern unsigned  _stklen = 80000;
unsigned char locked;
long temp_long;
float temp_f;
/*_______________________________________________
                      Functions and Procedures
_________________________________________________*/
void km(float**a,float f,int nral,int nrah,int ncal,int ncah)
{
int i;
int j;
int nra,nca;
float **m;
nca=ncah-ncal+1;
nra=nrah-nral+1;
    // size of resulting matrix (nrah-nral+1) rows X (ncah-ncbk+1) columns
m=(float**)malloc((unsigned)(nra)*sizeof(float*));
if (!m) nrerror("allocation failure 1 in mm()");
m-=nral;
for(i=nral;i<=nrah;i++){
    m[i]=(float*)malloc((unsigned)(nca)*sizeof(float));
    if (!m[i])nrerror("allocation failure 2 in mm()");
    }

for(i=1;i<=nra;i++)
    {
    for(j=1;j<=nca;j++)
        {
        m[i][j]=(a[i][j])*(f);
        }
    }
```

APPENDIX G Experimental Estimation software

estbase.c

```
for(i=1;i<=nra;i++)
    for(j=1;j<=nca;j++)
        a[i][j]=m[i][j];
free_matrix(m,1,nra,1,nca);

}

/****************************************************************
* Procedure Receive()                                           *
* Input: none                                                   *
* Output : none                                                 *
* Action : receives frame                                       *
* HISTORY: Date         Author       Comments                   *
*          ---------------------------------------------------- *
*          22.05.95     S.M.Rolland  Creation                   *
*          08.09.95     "             ARCOM card                *
*           11.09.95    "              aligned with ssdrv.c       *
****************************************************************/
void Receive()
{
int i;
int j;
int temp;
float f;
RSD[Station_Number].Buffer_Status=BUFFER_READY;
if ( (msg[0]==MASTER_ADDRESS) && ((msg[1] & 0xEF) == TOKEN)) /* The Token */
       {
       Send_TOK_A(RSD[0].Predecessor,0);
       RSD[0].Station_State=TOKEN_ACK;


       }

else if (msg[1]  == UI) /* I frame*/
    {
     if (msg[2]==VIDEO)
          {
          gotoxy(50,13);
          printf("Update");

          speed=msg[3];
          command=msg[4];
          th=msg[5];
          tl=msg[6];
              old_error=error;
        if (speed<40)
           calculated_speed=0;
        else
           calculated_speed=25.741*speed-953.36;
        //PID calculations
        error=setpoint-calculated_speed;
        if (error!=0)
           derivitive=(old_error-error)/error;
        else
           derivitive=0;
        integral=integral+error;
        if (integral<-200)
           integral=-200;
        if (integral>200)
           integral =200;
        if (locked == TRUE)
           {
           if (temp_k<0)
               temp_k=-temp_k;

           PIDcommand=temp_k*error+temp_Ti*integral;
           }
        else
        PIDcommand=setpoint;
        if (PIDcommand>255)
           {
           Tilt_Com=255;
           f_tilt_com=255.0;
           }
        else if (PIDcommand<0)
           {
           Tilt_Com=0;
           f_tilt_com=0.0;
           }
        else
           {
           Tilt_Com=(unsigned char)PIDcommand;
           f_tilt_com=PIDcommand;
           }
        RSD[1].Info_Length = 4;
        RSD[1].Buffer_Status= BUFFER_READY;
        RSD[1].Data[0]= TILT_COMM;
        RSD[1].Data[1]= Tilt_Com;

```

APPENDIX G Experimental Estimation software

estbase.c

```
            }


      }/* End UI Frame*/
else if ( (msg[1] & 0xEF) == SS) /* Set Successor*/
        {
        RSD[Station_Number].Successor=msg[2];
        /* ACknowledge*/
        }
else if ( (msg[1] & 0xEF) == SP) /* Set Predecessor*/
        {
        RSD[Station_Number].Predecessor=msg[2];
        /* ACknowledge*/
        }
else if ( (msg[1] & 0xEF) == WFM) /* Who Follows Me */
        {
        for (i=0;i<=NUMBER_OF_STATIONS;i++)
            {
            if (RSD[i].Station_Address == (msg[2] & 0xFF))
                {

                if (i== 0)
                {
                RSD[0].Station_State=TOKEN_ACK;
                Station_Number=0;
                return;
                }
                else
                {
                if (Recovery(i,0)!=TRUE)
                    {
                    RSD[i].Station_State=ERASED;
                    temp=FALSE;
                    for(j=1;j<NUMBER_OF_STATIONS;j++)
                    {
                    if (RSD[j].Station_State != ERASED)
                        temp=TRUE;
                    }
                    if (temp == FALSE)
                        {
                        printf("\nAll nodes now erased - automatic shutdown, check main umbilical");
                        getch();
                        exit(1);
                        }

                    /* REGENERATE TOKEN FROM MASTER*/
                    RSD[0].Station_State=TOKEN_ACK;
                    Station_Number=0;
                    for (temp=0;temp<NUMBER_OF_STATIONS;temp++)
                        {
                            if (RSD[temp].Successor== RSD[i].Station_Address)
                                {
                                RSD[temp].Successor=RSD[i].Successor;
                                }
                        }
                    for (temp=0;temp<NUMBER_OF_STATIONS;temp++)
                        {
                            if (RSD[temp].Predecessor== RSD[i].Station_Address)
                                {
                                RSD[temp].Predecessor=RSD[i].Predecessor;
                                }
                        }
                        /* should reduce TRT*/
                          RSD[i].Station_State=ERASED;
                          RSD[0].Station_State=TOKEN_ACK;
                          Set_Logical_Ring(1, 0,TTRT);

                    return;
                    }
                return;
                }
                }
            }
        }
else if ( (msg[1] & 0xEF) == CTF) /* Claim token frame */
        {
        for (i=0;i<=NUMBER_OF_STATIONS;i++)
            {
            if (RSD[i].Station_Address == (msg[2] & 0xFF))
                {
                for(temp=0;temp<100;temp++); /*  delay ?*/
                RSD[i].Station_State=DISCONNECT_S;
                RSD[i].Station_State=DISCONNECT_S;

                }
            }
    for(i=0;i<NUMBER_OF_STATIONS;i++)
```

APPENDIX G Experimental Estimation software

estbase.c

```
    {
    if ((i>=1) && (i<NUMBER_OF_STATIONS-1))
        {
        RSD[i].Successor=RSD[i+1].Station_Address;
        RSD[i].Predecessor=RSD[i-1].Station_Address;
        }
    else if (i==0)
        {
        RSD[0].Successor=RSD[1].Station_Address;
        RSD[0].Predecessor=RSD[NUMBER_OF_STATIONS-1].Station_Address;
        }
    else if (i==NUMBER_OF_STATIONS-1)
        {
        RSD[i].Successor=RSD[0].Station_Address;
        RSD[i].Predecessor=RSD[i-1].Station_Address;
        }

    }
    Set_Logical_Ring(1, 0,TTRT);
    RSD[0].Station_State=TOKEN_ACK;
    }
}

/*_____________________________________________________________________
* Procedure:main                                                      *
* Input: void                                                         *
* Output : void                                                       *
* Action :main program loop                                           *
* HISTORY: Date         Author        Comments                        *
*          ---------------------------------------------------------- *
*          11.08.95     S.M.Rolland   Creation                        *
*          13.10.95     "             Improved                        *
______________________________________________________________________*/
void main()
{
    int counts;
    int channel, byte;
    signed char Command[4];
    char flag;  /* thruster command change*/
    char status_flag; /* station state flag*/
    int end;
    int i,j;
    int car;
    char ctemp;
    long old_trt;
    unsigned char successor;
    static float den;
    static float lamba=0.99;
    static float estimate_error;
    double RLS_step;

    int tint;
    float temp_float1,temp_float2;


    FILE *in;
    FILE *pid;
    float **
 theta_k,**phi_k,**theta_old,**phi_old,**P,**P_old,**L,**temp,**temp2,**P_temp,**P_temp2;
    float ** used_theta;
    float ** used_phi,** P_temp3,**P_temp4,**phi_temp;
    float ** phi_log3,** phi_log4,** phi_log5,** phi_log6;

    float epsilon;
    float d;
    float Ti,Td,K;
    float a1,a2,tau,delta2,mu,h;
    float *f_pointer;
    unsigned char test;

    locked-=FALSE;
     end=0;
    temp_float1=0;
    temp_float2=0;
    // Communication card detection
     printf("\n<<< Demo Program >>>\n\n");
     printf("Assumes PCSER4 switches set to 180\n");
     printf("Testing for PCSER4\n");
     byte = ioread(ID);
     INACTIVITY_TIME_OUT=10000;
     TTRT=255;
     for (i=0;i<6;i++)
        Command[i]=0;
     setpoint=0;
     calculated_speed=0;
     byte=ioread(ID);
     Tilt_Com=0;
     f_tilt_com=0;
     if (byte == 16)
```

APPENDIX G Experimental Estimation software

estbase.c

```
    {
        printf("PCSER4 found OK\n");
        iowrite(GRLED,0); /* LED on */
        delay(0x100);
        iowrite(GRLED,1); /* LED off */
        delay(0x100);
        channel = 0xFF;
        if ((log_file = fopen("otl.log", "at")) == NULL)
                                {
                                printf("Cannot open otl.log");
                                return;
                                };
        if ((in = fopen("event.log", "wt"))
                == NULL)
                                {
                                fprintf(stderr, "Cannot open input file.\n");
                                   exit(1);
                                }
        if ((pid = fopen("pid.log", "wt"))
                == NULL)
                                {
                                fprintf(stderr, "Cannot open pid file.\n");
                                exit(1);
                                }
        // matrix initialisation
        used_theta=matrix(1,1,1,N);
        theta_k=matrix(1,1,1,N);
        theta_old=matrix(1,1,1,N);
        used_phi=matrix(1,N,1,1);
        phi_k=matrix(1,N,1,1);
        phi_old=matrix(1,N,1,1);
        phi_log3=matrix(1,N,1,1);
        phi_log4=matrix(1,N,1,1);
        phi_log5=matrix(1,N,1,1);
        phi_log6=matrix(1,N,1,1);
        phi_temp=matrix(1,N,1,1);
        L=matrix(1,1,1,N);
        P=matrix(1,N,1,N);
        P_old=matrix(1,N,1,N);
        temp=matrix(1,1,1,N);
        temp2=matrix(1,1,1,1);
        P_temp=matrix(1,N,1,N);
        P_temp2=matrix(1,N,1,N);
        P_temp3=matrix(1,N,1,N);
        P_temp4=matrix(1,N,1,N);
        zero(P,1,N,1,N);
        eye(P,1,N,1,N);
        zero(P_old,1,N,1,N);
        eye(P_old,1,N,1,N);
        zero(theta_k,1,1,1,N);
        zero(theta_old,1,1,1,N);
        zero(L,1,1,1,N);
        theta_k[1][3]=1; // b1!=0
        theta_old[1][3]=1; // b1!=0
        theta_k[1][1]=1;
        theta_old[1][1]=1;
        theta_k[1][N]=1;
        theta_old[1][N]=1;
        RLS_step=0.0;
        zero(used_phi,1,N,1,1);
        zero(phi_k,1,N,1,1);
        zero(phi_old,1,N,1,1);
        phi_k[N][1]=1;
        phi_old[N][1]=1;
        zero(L,1,1,1,N);
        zero(temp,1,1,1,N);
        zero(temp2,1,1,1,1);
        // PID default values
        K=0.5;
        Ti=0.1;
        Td=0.01;
        d=1;
        // Evaluate PC speed
        printf("Evaluating PC Speed..");
        Evaluate_PC();
        printf("\nPC speed evaluated to %f",Small_Delay_Test);
        fprintf(log_file,"Feeback,Time,Sync\n");
        fprintf(in,"Measured Estimated Estimate_Error Step den Theta_k[1][1] Theta_k[1][2]
    Theta_k[1][3] Theta_k[1][4] Theta_k[1][5]");
        while((channel < 0) || (channel > 3))
            {
             printf("Enter a channel number (0-3)");
             scanf("%x",&channel);
            }
        setpoint=1500;
        printf("Initialising Channel %x\n",channel);
        initscc(channel);  /* initialise SCC */
        initDPLL(channel);
        Power_On();
```

APPENDIX G Experimental Estimation software

estbase.c

```
        /* set comm to thruster card*/
        Station_Number=0;
        Update_Address(RSD[Station_Number].Station_Address,channel);
        RSD[Station_Number].Station_State=TOKEN_ACK;
        clrscr();
        Set_Logical_Ring(1, channel,TTRT);
        gotoxy(10,1);
        printf("Estimator Test - Logging software ");
        printf("Node 3  : ");
        gotoxy(5,5);
        printf("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^");
        Inactivity_timer=biostime(0,0L);
        Trt_timer=biostime(0,0L);
        while (end==0)  /* loop until q pressed */
            {
            while   (RSD[0].Station_State!=TOKEN_ACK)
                {
                fprintf(pid," W ");
                Listen(channel);
                if ((biostime(0,0L)-Inactivity_timer) >= INACTIVITY_TIME_OUT)
                            {

                            Inactivity_timer=biostime(0,0L);
                            ctemp=FALSE;
                            fprintf(pid," X ");
                            for(j=1;j<NUMBER_OF_STATIONS;j++)
                                {
                                if (RSD[j].Station_State != ERASED)
                                ctemp=TRUE;
                                }
                            if (ctemp == FALSE)
                                {
                                printf("\nAll nodes now erased - automatic shutdown, check main umbilical");
                                getch();
                                exit(1);
                            }
                            RSD[0].Station_State=TOKEN_ACK;

                            }
                        }

        if (RSD[0].Station_State== TOKEN_ACK)
        {
        fprintf(pid," TOK ");
        //PID Calculation is implemented in Receive()
        tint=(int)d;
        switch(tint)
            {
            case 1:
                copy_m(phi_k,phi_temp,N,1);
                break;
            case 2:
                copy_m(phi_old,phi_temp,N,1);
                break;
            case 3:
                copy_m(phi_log3,phi_temp,N,1);
                break;
            case 4:
                copy_m(phi_log4,phi_temp,N,1);
                break;
            case 5:
                copy_m(phi_log5,phi_temp,N,1);
                break;
            case 6:
                copy_m(phi_log6,phi_temp,N,1);
                break;

            }
        pmm(theta_k,phi_temp,1,1,1,N,1,N,1,1);
        estimate_error=calculated_speed-(phi_temp[1][1]);
        fprintf(in,"\n%e %e",calculated_speed,phi_temp[1][1],estimate_error);
        fprintf(log_file,"%d,%d,%d\n",speed,Tilt_Com,(th*256)+tl);
        // CHECK RLS CONVERGENCE
        if ((int)RLS_step<50)
            {
            fprintf(in," <50");
            if ( ((int)fabs(estimate_error)>(int)(calculated_speed*0.1))  )
                {
                test=TRUE;
                fprintf(in," OUT%e %d ",estimate_error,(int)fabs(estimate_error));
                }
             else
                {
                test=FALSE;
                fprintf(in," IN%e %d ",estimate_error,(int)estimate_error);
              }
            }
        else
```

APPENDIX G Experimental Estimation software

estbase.c

```
            test=FALSE;


        if (test==TRUE )
           {
           // RLS algorithm
           fprintf(in," RLS");
           RLS_step++;
           locked=FALSE;
           lamba=pow(0.99,RLS_step);
           P_old=P;
           tint=(int)d;
           switch(tint)
              {
           case 1:
              copy_m(phi_k,P_temp3,N,1);
              break;
           case 2:
              copy_m(phi_old,P_temp3,N,1);
              break;
           case 3:
              copy_m(phi_log3,P_temp3,N,1);
              break;
           case 4:
              copy_m(phi_log4,P_temp3,N,1);
              break;
           case 5:
              copy_m(phi_log5,P_temp3,N,1);
              break;
           case 6:
              copy_m(phi_log6,P_temp3,N,1);
              break;
           }

           ptranspose(P_temp3,1,N,1,1);
           copy_m(P_old,P_temp2,N,N);
           pmm(P_temp3,P_temp2,1,1,1,N,1,N,1,N);
           tint=(int)d;
           switch(tint)
              {
           case 1:
              copy_m(phi_k,phi_temp,N,1);
              break;
           case 2:
              copy_m(phi_old,phi_temp,N,1);
              break;
           case 3:
              copy_m(phi_log3,phi_temp,N,1);
              break;
           case 4:
              copy_m(phi_log4,phi_temp,N,1);
              break;
           case 5:
              copy_m(phi_log5,phi_temp,N,1);
              break;
           case 6:
              copy_m(phi_log6,phi_temp,N,1);
              break;
           }

           pmm(P_temp2,phi_temp,1,1,1,N,1,N,1,1);
           den=lamba+phi_temp[1][1];
           fprintf(in," %e",den);
           copy_m(phi_k,phi_temp,N,1);
           pmm(P_old,phi_temp,1,N,1,N,1,N,1,1);
           km(phi_temp,1/den,1,N,1,1);
           copy_m(theta_k,temp,1,N);
           km(phi_temp,estimate_error,1,N,1,1);
           theta_k=transpose(phi_temp,1,N,1,1);
           copy_m(temp,theta_old,1,N);
           paddm(theta_old,theta_k,1,1,1,N,1,1,1,N);
           //calculate new P
           //P=(P_old-((P_old*phi_k*phi_k'*P_old)/(lambda+phi_k'*P_old*phi_k)))/lamda
           tint=(int)d;
                 switch(tint)
           {
           case 1:
              copy_m(phi_k,phi_temp,N,1);
              break;
           case 2:
              copy_m(phi_old,phi_temp,N,1);
              break;
           case 3:
              copy_m(phi_log3,phi_temp,N,1);
              break;
           case 4:
              copy_m(phi_log4,phi_temp,N,1);
              break;
           case 5:
```

APPENDIX G Experimental Estimation software

estbase.c

```
                copy_m(phi_log5,phi_temp,N,1);
                break;
            case 6:
                copy_m(phi_log6,phi_temp,N,1);
                break;

            }
            pmm(P_old,phi_temp,1,N,1,N,1,N,1,1);
            tint=(int)d;
            switch(tint)
                {
            case 1:
                copy_m(phi_k,P_temp2,N,1);
                break;
            case 2:
                copy_m(phi_old,P_temp2,N,1);
                break;
            case 3:
                copy_m(phi_log3,P_temp2,N,1);
                break;
            case 4:
                copy_m(phi_log4,P_temp2,N,1);
                break;
            case 5:
                copy_m(phi_log5,P_temp2,N,1);
                break;
            case 6:
                copy_m(phi_log6,P_temp2,N,1);
                break;
            }
            ptranspose(P_temp2,1,N,1,1);
            pmm(phi_temp,P_temp2,1,N,1,1,1,1,1,N);
            copy_m(P_old,P_temp3,N,N);
            pmm(P_temp2,P_temp3,1,N,1,N,1,N,1,N);
            km(P_temp3,-1/den,1,N,1,N);
            paddm(P_old,P_temp3,1,N,1,N,1,N,1,N);
            km(P_temp3,1/lamba,1,N,1,N);
            copy_m(P_temp3,P,N,N);
            }// end of RLS
            else
            {
            // RLS has converged
            // reset RLS parameters
            locked = TRUE;
            fprintf(in," NORLS");
            zero(P,1,N,1,N);
            eye(P,1,N,1,N);
            zero(P_old,1,N,1,N);
            eye(P_old,1,N,1,N);
            zero(L,1,1,1,N);
            RLS_step=0.0;
            }
copy_m(phi_log5,phi_log6,N,1);
copy_m(phi_log4,phi_log5,N,1);
copy_m(phi_log3,phi_log4,N,1);
copy_m(phi_old,phi_log3,N,1);
copy_m(phi_k,phi_old,N,1);
copy_m(phi_log5,phi_log6,N,1);
copy_m(phi_log4,phi_log5,N,1);
copy_m(phi_log3,phi_log4,N,1);
copy_m(phi_old,phi_log3,N,1);
copy_m(phi_k,phi_old,N,1);
fprintf(in," TestY %e",theta_k[1][1]);
if ((int)RLS_step ==1)
    {
    phi_k[N][1]=1;
    phi_old[N][1]=1;
    phi_k[2][1]=f_tilt_com;
    phi_k[1][1]=-calculated_speed;
    }
else
    {
    phi_k[5][1]=phi_k[4][1];
    phi_k[4][1]=phi_k[3][1];
    phi_k[3][1]=phi_k[2][1];
    phi_k[2][1]=f_tilt_com;
    phi_k[1][1]=-calculated_speed;

    }




phi_log6[1][1]=-calculated_speed; // this does not need shifting
phi_log5[1][1]=-calculated_speed;
phi_log4[1][1]=-calculated_speed;
phi_log3[1][1]=-calculated_speed;
phi_old[1][1]=-calculated_speed;
fprintf(in," TestZ %e",theta_k[1][1]);
```

APPENDIX G Experimental Estimation software

estbase.c

```
fprintf(in," %e %e %e %e %e
",theta_k[1][1],theta_k[1][2],theta_k[1][3],theta_k[1][4],theta_k[1][5]);
fprintf(in," %e %e %e %e %e ",phi_k[1][1],phi_k[2][1],phi_k[3][1],phi_k[4][1],phi_k[5][1]);
fprintf(in," d= %e",d);
// Epsilon based delay estimation
epsilon=(theta_k[1][4]-theta_k[1][2])/(theta_k[1][2]+theta_k[1][3]+theta_k[1][4]);
fprintf(in," epsilon= %e",epsilon);

if (((int)epsilon*10 <-8) && ((int)RLS_step==0)) // if RLS has locked and espilon <-0.8
    {
    temp_float1=theta_k[1][4]; //b0
    d=d-1;
    theta_k[1][4]=theta_k[1][3]+3*temp_float1; // b0=b1+3b0
    theta_k[1][3]=theta_k[1][2]-3*temp_float1; // b1=b2-3b0
    theta_k[1][2]=temp_float1; // b2=b0
    }
else if (((int)epsilon*10>8) && ((int)RLS_step==0)) // if RLS has locked and espilon >0.8
    {
    d=d+1;
    temp_float1=theta_k[1][3]; //b1
    temp_float2=theta_k[1][2]; //b2
    theta_k[1][3]=theta_k[1][4]-3*temp_float2;//b1=b0-3b2
    theta_k[1][2]=temp_float1+3*temp_float2; //b2=b1+3*b2
    theta_k[1][4]=temp_float2; //b0=b2
    }
    if ((int)(epsilon*1000) !=0)
       tau=h*epsilon;
    else
       tau =0.001;
// calculate PID coeffs
if ((locked==TRUE) && ((int)theta_k[1][1]!=1) )
    a1=-log(-theta_k[1][1])/h;
else
    a1=-log(0.9)/h;
    mu=(theta_k[1][2]+theta_k[1][3]+theta_k[1][4])/(1+theta_k[1][1]);
// using Haalman tuning rules
    Ti=a1;
    K=2/(a1*6*mu*tau);
    temp_k=K;
    temp_Ti=Ti;
    fprintf(pid,"\n %e %e %e %e %e",a1,mu,tau,K, Ti);
    delay(50); // this is  the additional delay in msec
    old_trt=Trt_timer;
    /* Initialise TRT Timer*/
    Trt_timer=biostime(0,0L);
    h=(Trt_timer-old_trt)/_BIOS_CLK_TCK;
    fprintf(pid," h=%lu  %e",Trt_timer-old_trt,h);
    gotoxy(50,13);
    printf("token");
    gotoxy(50,13);
    printf("                        ");
    gotoxy(50,5);
    flag=0;
    for (i=1;i<NUMBER_OF_STATIONS;i++)
             {
             fprintf(pid," S%d",i);
             if(RSD[i].Station_State== ERASED)
                {
                   /*              break; skip to next station*/
                }
             else if (RSD[i].Station_State== DISCONNECT_S)
                {
                   gotoxy(50,i+1);
                   printf("OFF ");

                Send_Snrm(i,channel);
                if (RSD[i].Station_State== I_T_S)
                   {
                   gotoxy(50,i+1);
                   printf("ON ");
                   }
                }
             else if (RSD[i].Station_State== GO_TO_DISC)
                {
                Send_Disc(RSD[i].Station_Address,channel);
                }
             else if ( (RSD[i].Info_Length>0) &&(RSD[i].Buffer_Status==BUFFER_READY) )
                {
                      gotoxy(50,13);
                printf("Xmit 1");

                Xmit_I_T_S(i,T_I_FRAME,channel);
                RSD[i].Info_Length=0;
                gotoxy(50,13);
                printf("Xmit 1b");
                }
             } /* end of for*/

             fprintf(pid," Passing");
```

APPENDIX G Experimental Estimation software

estbase.c

```
                if (Pass_TOKEN(RSD[Station_Number].Successor,channel)!=TRUE)
                {
                   gotoxy(50,13);
                   printf("Failed");

                for(ctemp=1;ctemp<NUMBER_OF_STATIONS;ctemp++)
                   {
                   if (RSD[ctemp].Station_Address== RSD[Station_Number].Successor)
                        successor=ctemp;

                   }
                   Listen(channel);
                   Listen(channel);
                if (Recovery(successor,0)!=TRUE)
                   {

                      gotoxy(50,successor+1);
                      printf("OFF ");

                         for (ctemp=0;ctemp<NUMBER_OF_STATIONS;ctemp++)
                            {
                            if (RSD[ctemp].Successor== RSD[successor].Station_Address)
                            {
                               RSD[ctemp].Successor=RSD[successor].Successor;
                            }
                            }
                         for (ctemp=0;ctemp<NUMBER_OF_STATIONS;ctemp++)
                            {
                               if (RSD[ctemp].Predecessor== RSD[successor].Station_Address)
                                  {
                                  RSD[ctemp].Predecessor=RSD[successor].Predecessor;
                                  }
                             }
                         RSD[successor].Station_State=ERASED;
                         RSD[Station_Number].Station_State=TOKEN_ACK;
                         Set_Logical_Ring(1, channel,TTRT);

                    }

    }
       }  //
          if (kbhit())
             car=getch();
          else
             car=0;
          switch (car)
          {
          case 113:
          case 81:            /* Q _>end*/
                {end=1; break;}
             }  /* end of switch statement*/

    }  /* end of while*/
    // closing files
    fclose(log_file);
    fclose(in);
    fclose(pid);
    // cleaning matrix allocations
    free_matrix(used_theta,1,1,1,N);
    free_matrix(theta_k,1,1,1,N);
    free_matrix(theta_old,1,1,1,N);
    free_matrix(phi_k,1,N,1,1);
    free_matrix(phi_old,1,N,1,1);
    free_matrix(phi_log3,1,N,1,1);
    free_matrix(phi_log4,1,N,1,1);
    free_matrix(phi_log5,1,N,1,1);
    free_matrix(phi_log6,1,N,1,1);
    free_matrix(used_phi,1,N,1,1);
    free_matrix(P,1,N,1,N);
    free_matrix(P_old,1,N,1,N);
    free_matrix(L,1,1,1,N);
    free_matrix(temp,1,1,1,N);
    free_matrix(temp2,1,1,1,1);
    free_matrix(P_temp,1,N,1,N);
    free_matrix(P_temp2,1,N,1,N);
    free_matrix(P_temp3,1,N,1,N);
    free_matrix(P_temp4,1,N,1,N);
    free_matrix(phi_temp,1,N,1,1);
    }

     else // no communication card found
          printf("PCSER4 not found: ID = $%x\n",byte);/* Arcom board not detected*/
          getch();
}
```

Appendix H RLS Estimation

The Recursive Least Square estimation algorithm used is a classic one [29]. It allows to estimate unknown model parameters, by using observations from the experiment. In our case, the model is $H(Z) = Z^{-(d+1)} \frac{b0 + b1Z + b2Z^2}{Z - p}$. The unknown parameters can be put in vector notation has follows:

$$\vartheta = [-p \quad b2 \quad b1 \quad b0 \quad C]$$

The observations are in the following vector:

C is a constant set to 1.

$$\Psi_{(k+1)} = [y_{(k+1)} \quad u_{(k+1)} \quad u_{(k)} \quad u_{(k-1)} \quad C]$$

$$\hat{\vartheta}_{(k+1)} = \hat{\vartheta}_{(k)} + \Gamma_{(k+1)}\left[y_{(k+1)} - \hat{\vartheta}_{(k)}\Psi_{(k+1}\right]$$

$$\text{with } \left[y_{(k+1)} - \hat{\vartheta}_{(k)}\Psi_{(k+1}\right] \text{the estimation error at step k} + 1$$

$$\Gamma_{(k)} = P_{(k-1)}\Psi'_{(k)}\left[\lambda + \Psi_{(k)}P_{(k-1)}\Psi'_{(k)}\right]^{-1}$$

$$P(k) = \frac{\left[I - \Gamma_{(k)}\Psi_{(k)}\right]}{\lambda}P_{(k-1)}$$

The recursive least square algorithm with forgetting factor used is:

Where λ is the forgetting factor and is calculated as $\lambda=0.99^{step}$, where step is the number of iteration of the RLS estimator.

Those calculations are implemented in the C program in appendix G.

APPENDIX I Probability of spurious flag

Funk [14] shows that in an assumed memory-less, binary symmetric channel model, each bit position is inverted independently with the bit error probability *p,* and is received correctly with the probability *q.* With this simplified model we get :

$$P(FLAG) = \frac{1}{248} q^2 \left(\binom{6}{1} pq^5 + \binom{6}{2} p^2 q^4 + \binom{6}{3} p^3 q^3 + \binom{6}{4} p^4 q^2 + \binom{6}{5} p^5 q + p^6 \right)$$

$$P(FLAG) = \frac{1}{248} q^2 \left(1 - q^6\right) \text{ for } q = 1 - p$$

$$P(FLAG) = \frac{6}{248} p = 0.024p \text{ for } p << 0.5$$

For example, with a bit error probability p = $1e^{-6}$, the probability of having a spurious flag is $2.41e^{-8}$.

However, this does not take into account that the probability distribution for '0' and '1' is different, because of bit insertion. The average distance of a '0' bit insertion is 62 bits, since '0' bits are inserted in patterns of type 011111 or 011111 11111 etc.... Considering these patterns as exclusive events occurring with probability 2^{-6}, 2^{-11}..., the resulting probability of '0' bit insertion is:

$$P('0'\text{insertion}) = 2^{-6} + 2^{-11} + ...$$

$$P('0'\text{insertion}) = 2^{-6} / (1 - 2^{-5}) = 1/62$$

This means that an HDLC text of arbitrary length contains 32/63 "0's" and 31/63 "1's".

Within and HDLC frame, things are slightly different : within the first five bits after the FLAG, the probabilities of "0's" and "1's" are 0.5. Funk [14] shows that the probability of a spurious flag caused by a single bit error then becomes :

$$P(FLAG) \approx \frac{2}{63} pq^7 \approx 0.0317p \quad \text{for } p << 0.5$$

For example, with a bit error probability p = $1e^{-6}$, the probability of having a spurious flag is $3.17e^{-8}$, slightly higher than with the simplified model.

A spurious flag divides the transmitted message in two received frames. However, SDLC rejects short frames. Thus the chances of receiving a shortened frame without detecting it supposes that:

- The spurious flag occurs after the address and control field are transmitted.

- The 16 bits Frame Check Sequence (FCS) happens to be correct. The probability of this happening is 2^{-16}.

The resulting residual error probability caused by a spurious flag is for a n–bit message is:

$$R(FLAG) = 2^{-16}\left(1-\left(1-P(FLAG)\right)^{n-16}\right)$$