Communication between Robots and a Computer via the Internet

Final Report

Andreas Bernhard Max Hofmeier

Submission Date: 26.04.2005
Student ID: 2406108

This report has been submitted for assessment towards a Bachelor of Engineering Degree in Computer Systems and Networks (2388) BEng in the School of Engineering, South Bank University.

This report is written in the author's own words and all sources have been properly cited.

Author's signature: ______________________

Acknowledgements

I would like to thank my supervisor, Dr. Shuwo Chen, who supervised this project and was always ready to answer my questions.

Thanks are also given to other people from London South Bank University and the University of Applied Science Bremen, who offered their assistance to me while doing this project. Dr. Chris Merridan, the technical assistant and Mr. Jianzhong Shang, who supported me in dealing with the robot experiment. Dr. Thomas Risse who helps me to organise of my overseas-term. My German class mates stood on my side with advice and encouragement. All peoples who provided help in form of materials such as free software (GNU Linux), free documents (web-sites), articles, and books.

Abstract

This report describes the final year project done by Andreas Hofmeier. The project is aimed to develop the software tool for the communication between robots and computers on the Internet to realise the remote control of robots via the Internet.

The feasibility study of controlling robots on the Internet was re-examed, identifying the certain restrictions affecting the communication and hence control on the Internet. One conclusion has been drawn that the time delay is the most important restriction is caused by the local network but not by the Internet. Several approaches to the restrictions were studied and a promising method, the line monitor, was developed.

A software platform in form of a library for GNU Linux was developed to provide the necessary tools for implementing robot control on the Internet. The platform is successfully used to control a Cartesian robot in laboratory test.

The results have proven that design methodology for the project are correct and the theoretical results will benefit future development.

Contents

Aim and Objectives

Aim

The aim of the project is to develop a software platform on GNU Linux systems (in the form of a library) for the communication between a server and robots to realise remote control of robots over the Internet. The communication between robots is allowed as well.

Objectives

1. (a) Feasibility study of real-time control of robots on the Internet.
   (b) A block diagram of the architecture of an example system which can control a robot remotely
2. (a) Developing the library
   (b) Developing the simulator and the user interface (UI)
   (c) Demonstrating the platform by using a simulation. Showing that it is possible to control the simulated robot over an Internet connection.
3. (a) Developing an interface to the real robot
   (b) Demonstrating the platform by means of the real robot. Showing that it is possible to control the real robot over an Internet connection.

Deliverables

The objectives have been met and the deliverables include,

1. The Library (the Software Platform)
2. Documentation of the Library (API; description of the functions; how to use)
3. User Interface
4. Documentation of the User Interface (User manual)
5. Simulator
6. Documentation of the Simulator (User manual)
7. Interface between Library and real Robot, for Demonstration.
8. Interim Report
9. Presentation
10. Final Report

1 Introduction

Robots become more and more important because the technical progress allows economic and useful applications. Controlling robots over a short distance with cables or wirelessly is quite popular, but more often robots need to be controlled at a remote site far away from the scene. For example a robot as security guard or a pizza-robot is operating at home while controlled from the office. To meet this demand a communication network is required.

One of the most flexible and economical solutions is to control robots on the Internet that works packet-oriented. All data have to be fragmented before it can be transmitted. The transmission process on the Internet can be compared with the postal service (Ball et al. 1999). "The packet should be there within two days" is a possible answer to the question how long the delivery takes. This little word "should" is the problem. It should, but there is no guarantee. If many packets are handed in, it may take a week or more time to deliver them. The Internet has the similar problem. Even if a packet does not need a week to reach a recipient, it is still difficult to predict a delay time because huge amount of users are at random using the resources on the Internet. Therefore as a user of the Internet, a controlled robot may encounter a statistic time delay. (Elhaji et al. 2000).

Through this project the feasibility of controlling robots remotely over the Internet was studied. A software platform on GNU Linux which provides this functionality was developed.

This platform in form of a network library provides the tools to utilise the Internet as a communication link to control a robot. An example of using the network library is given by a simple demonstration on a real robot. The demonstration includes a user interface, a simulator, and an interface to the robot.

The analysis has shown that under certain circumstances it is possible to actually control a robot on the Internet. One of the possibilities is to observe the network connection and initiate appropriate actions when the line (connection) becomes unusable for the remote control. This idea was adapted from Andreu et al (2003) and implemented. In this report it is called the line monitor.

The report is structured in the following way: the first two sections define the aim, the objectives, and the deliverables of the project. The next section introduces the technical background. After this the technical approach will be explained in detail. The results of the analysis and the tests are given in the next chapter named "Results and Discussion". The conclusions and recommendations for further work are given in Chapter 5. After that the Bibliography is listed. It is followed by details about the planning of the project compared with its actual realisation. The last parts of the report are the appendices which include the software and its documentation (user's manual and application programmer interface).

2 Technical Background and Context

2.1 Modes to Control a Robot

Han et al (2004) distinguishes between three modes to control a robot which were adapted. This three points are extreme examples only. A robot in the "real world" will be somewhere between those extreme points. This depends highly on the application.

2.1.1 Direct Control

Within this control mode the hardware is controlled at lowest level over the network. There is no intelligence or data processing on the robot's side.

For example: the robot receives a bit stream which represents its outputs. The robot receives a package of n bits analogous to n output-bits. These outputs can be simple actors which can only be switched on or off (one bit) or complex solutions with digital-to-analogue converters which are controlling a DC-motor (maybe 12 bits).

This mode requires strict time constraints because the controlling is time based. If the robot should move one meter in a direction the corresponding motor for this direction has to be switched on for exactly the time which is necessary to cover this distance. If the motor is switched on longer the robot will cover a greater distance and vise versa. The engines has to be switched "in time" but this is almost impossible if the time for transmitting a bit (or a package if more than one motor has to be controlled) varys from one command to the next one. If the time-delay would have been constant, it could be simply subtracted in the calculation.

Another problem occurs if the connection breaks down. When the robot receives a "start moving" command just before the connection fails it may move until the battery is empty. This can be a hazard if, for example, the robot hits someone.

2.1.2 Supervisory Control

This control mode controls the robot on a much higher level. For this reason more intelligence is needed on the robot's side.

A target is transmitted to the robot. The robot has to evaluate this target and calculate the appropriate action to reach it. The target can be transmitted in relative (Δx, Δy) or in in absolute (x, y) coordinates (In this case it is assumed that the robot has two degrees of freedom – can move in two dimensions.) In the first case the robot has to calculate which actors have to be switched on and for how long to cover the given distance in the right direction. In the second case the robot has to know its current position to calculate the Δx and the Δy which can be used to move to the target.

The calculation of Δx and Δy may include considerations like:

- What is the fastest way?
- What is the way which needs the fewest resources (energy)?
- Which way causes no hazards or damages? Are they any hindrances?

In this mode the time constraints are not as strict as in the direct control because the robot will stop moving if the target is reached and no new targets are receive in time. Normally hazards only occur if something is moving or is "in action".

2.1.3 Job Scheduling

Within the Job Scheduling Mode a whole sequence of targets or jobs is transmitted to the robot at once.

2.2 Levels of Processing

To be able to "teleoperate" something (for example to control a robot from far away) at least three levels of data-processing are necessary. These three levels are the human operator, the User Interface (UI), and the robot control program. They are illustrated in figure 1.

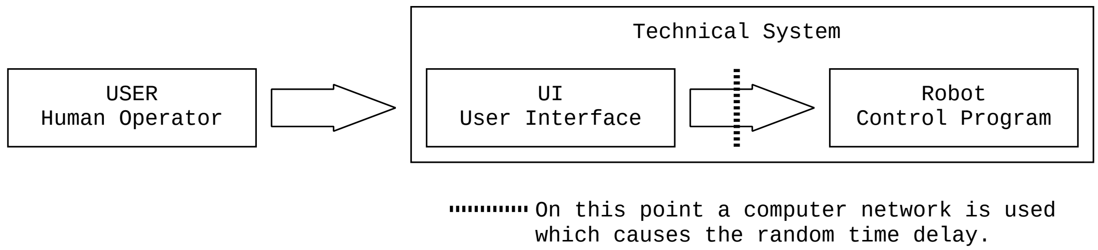


Figure 1: Different Levels of Processing which are necessary to Control a Robot Remotely.

- The human operator makes the decisions and gives the system its purpose. There will be no systems without a human operator at some level because there is no point in doing something without gaining an advantage. This human interaction can be within a wide range from “switching it on to get a cup of hot coffee out of it” to “control a space-explore-robot to explore what is out there”. The human operator always gives the commands to an User Interface.

- The user interface has to read the commands from the user and transmit it through some kind of network to the robot-control-program. The program which performs the necessary operations to handle this job runs on computer in front of the user. This can be a kind of robot-control-server if it is taken as a central control station.

- The robot control program receives the commands from the user interface and applies them to the hardware of the robot. This job is done by a piece of software which runs on a computer on (or near to) the robot.

2.3 Adaptability of the Program which Controls the Robot

In these days the requirement of multi-functionality becomes more important. The robot should be as flexible as possible to be used in a wide spectrum of applications.

To use a robot for a new application the program which controls it has to be changed. As discussed above this program is made up of three components. (If it is assumed, that there is some kind of “program” in our brain.)

In case that the "technical system" is very primitive it may be enough to train the human operator to do other things with it. An example could be a simple remote control of a crane which switches actors remotely.

An important improvement of our technology today is that it helps us to perform our tasks. The time of a human operator is valuable and should not be wasted in doing things which can be done by a computer. Many calculations can be processed much faster and more accurate by a computer than a human.

The consequence of this is that it may not be enough to train the human operator to do new things with the robot. In this case two components are left: The GUI[1] (Server) and the robot control program.

One solution is to keep the robot control program as simple as possible and transfer all intelligence into the GUI (the Server). In this case the direct control is in use. As discussed this can be a problem because of safety considerations. If the network link breaks down no server or human operator can stop the robot. There must be some processing on the robot's side. At least an emergency stop function has to be implemented. This solution makes it possible to change the behaviour of the technical system only by changing the server/GUI side. This can be an advantage.

On the other hand it is possible to perform one part of the processing on the robot's side. This moves the classification of the system closer to the supervisory control. The GUI/server transmits a job or a target to the robot and monitors its execution. It might be necessary to change both, the GUI/server and the robot control program in order to change the behaviour of the system. Advantages of this solutions are: distribution of processing work on both sides, probably less bandwidth requirements, and a possible gain in safety.

2.4 Feedback

Another important issue needs to be considered: the feedback. This is the difference between operating or affecting something and controlling it. Affecting means to do something without getting a response. There is no guarantee that everything happens the way as it was intended to happen. Controlling applies a feedback which closes this (response-)loop. For example, it can be seen what the robot does. In this case it is likely that the bandwidth of the feedback connection is much bigger than the control connection because of the video data. Figure 2

[1]GUI stands for Graphical User Interface.

illustrates this example. Unfortunately these meanings are often confused. In this report the word “controlling” is used for both.

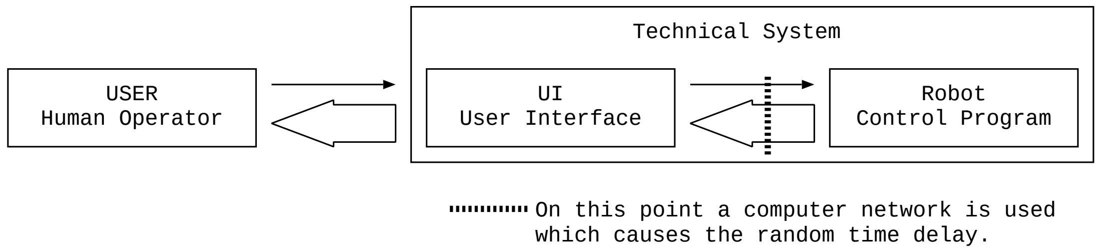


Figure 2: Different Levels of Processing which are necessary to Control a Robot Remotely: A Closed-Loop-System with Feedback.

2.5 Real Time and Bandwidth Constrains

The term “real time” is used for systems which have to complete a task in a certain time. This time depends on the application but it can be said that the (reaction-)time must be short enough to perform an in-time control of the environment. In this case the transfer of data over a network has to be finished within a certain time-limit.

Bandwidth describes how much data a network is able to transfer in a certain time. The time delay will increase if more data is transmit because the data has to be buffered until the line is free. If continuously more data is transfered the network can no longer transfer all the data without loosing some of it.

Both terms are interdependent. This will be explained in the following section.

2.5.1 How the Internet Works

Before it is possible to explain what the problem causes it is necessary to provide an overview about “how it works”. If detailed information is requested please refer to a textbook, for example, Forouzan (2001) from which this overview was condensed.

Figure 3 shows an overview of the different levels of data-processing which have to be passed before a communication is possible. The diagram illustrates these levels on a Hyper Text Transfer Protocol (HTTP) request. Those requests are generated if a web-page is opened. In the case of the example it was

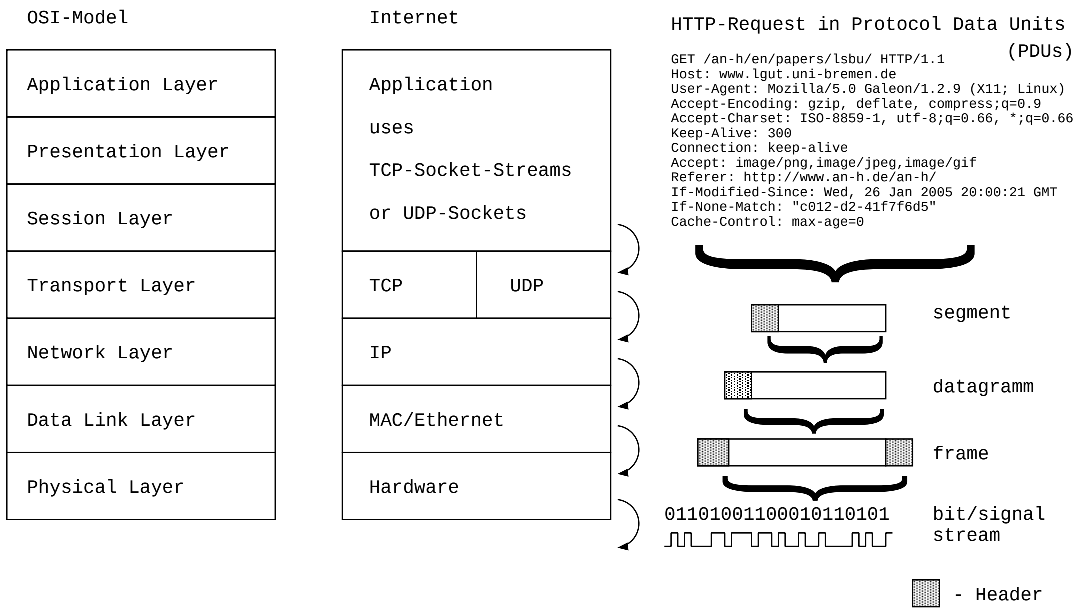


Figure 3: Comparison between OSI-Model and the real Internet

`http://www.lgut.uni-bremen.de/an-h/en/papers/lsbu/`. The OSI-model (left hand side) is a general description of the network layers. These OSI layers assigned to real layers which are used in the Internet. The right hand side lists the different types of protocol data units (PDUs) (packages) which are created by each layer. Each layer adds management information (a header) which are shown as gray extension. In this report the general term package is used for different PDUs.

To understand the idea behind these levels of layers it can be compared to telling someone a long story who is on the other side of the world. The narrator (the application) starts writing it and gives the script its secretary (the transport layer, TCP). She knows that the postal service accepts letters up to a maximum of 80g. For this reason the story has to be segmented into pieces which fit on less than 80 grams of paper. In order to make it easier to recombine the story on the other side of the world, supporting information like a segment-number is added to each letter. This letter is given to the next secretary (the network layer, IP). In this step the letter will be placed in an envelope. This envelope on which the source and destination address is written will be handed in to the nearest post office (the data link layer). Within the post office the letter will be packed into a bag which is transported to another post office. This office may resort the bags and send it to the following post office. This process (which can be compared with the routing and transferring data over a line [physical layer]) will be continuing until the letter reaches the mail box of the receiver. Then the reverse process takes place.

2.5.2 Changing Behaviour and Delay

The problems which arise from this process are: the system must work transparently. That means that the higher levels do not know what the lower levels are doing. For this reason the behaviour of the lower levels may vary. An example for this is a replacement of one secretary.

The same problem occurs during the transport. There is no guarantee that the letter will always take the same route. It is unknown which way a letter will take. This depends on the environment and on the network load or utilisation. If, for example, an earthquake destroys a road, the mail must take another way. The workload of the system may change faster than the environment. During the rush-hour it takes longer to cover a distance than on a "normal" daytime. This can be compared with school hours – all students using the network. This both phenomena cause the random time delay.

2.5.3 Carrier Sense Multiple Access with Collision Detection

The next problem arises from the way in which the physical layer transmits data. Ethernet is used in most of the end-user networks of the Internet. Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is in use within Ethernet. This protocol tries to broadcast when nobody else is sending data. If the network load increases it becomes harder to find a gap to send the data. Because of physical limitations (transmission speed) computers do not recognise fast enough if data is sent and start sending by them selfs. This causes collisions. Data is destroyed and has to be sent again. Because of this the delay increases with rising workload. These both are reasons for the unpredictability of the transport time. This is not applicable to modern switching networks. (Fairhurst 2004)

2.5.4 Level to Use

To use the high-level Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP) to control robots is not the most efficient way. It is possible to bypass the official transport layer by replacing it with an own protocol that is optimised for real time traffic. Liu et al (2004), for example, developed the "trinomial protocol". This is a semi-real-time-protocol. It works much better than the TCP or UDP but it cannot change the lower levels. These levels have to be used if the Internet should transport the data. They are prescribed by the Internet itself as a standard.

2.5.5 Monitor the Line

Another possibility, which was adapted from Andreu et al (2003), is to observe the network-connection and take appropriate actions if the connection becomes unusable for the purpose of remote controlling. This strategy assumes that the network is normally usable for the job. Without any further actions this method is not well suited for direct control. Instead it is very suitable for an additional usage.

2.5.6 Buffer to Compensate Random time Delay

Andreu et al (2003) explored the possibility of using a buffer or a stack as a “Delay Regulator” to smooth up the random component of the transport delay. This is done by delaying the data on the receiver side (buffer) in a way that the overall delay stays constant. For example, if one command has to be processed in one second, one command per second has to be ready for processing on the robot’s side. Assuming that the maximal time delay for the transmission is 5 seconds, the sender starts transmitting 5 seconds before the robot starts executing commands. All data which is received earlier will be stored in a buffer. If the sender sends one command in one second, the robot reads one command a second from the buffer, and the maximal time-delay does no exceed 5 seconds, the commands are constantly delayed by 5 seconds. This is the way to suppress the random component of the time delay by extending the delay to a maximal value.

This solution is suited to be used for direct control but has the disadvantage that the transmission takes longer than necessary. If everything works well all commands are delayed by the same time. It makes sense to combine this solution with a line-monitor to take appropriate actions if the maximal delay was exceeded.

2.5.7 Use a Simulator

One idea to fit a robot-control-system into existing bandwidth constrains is to prevent the large bandwidth-consumption of the video-data-stream by using a simulator. This simulator simulates the environment of the robot on the GUI/server side. Figure 4 gives an example of this. (Belousov 2004; Han et al. 2001)

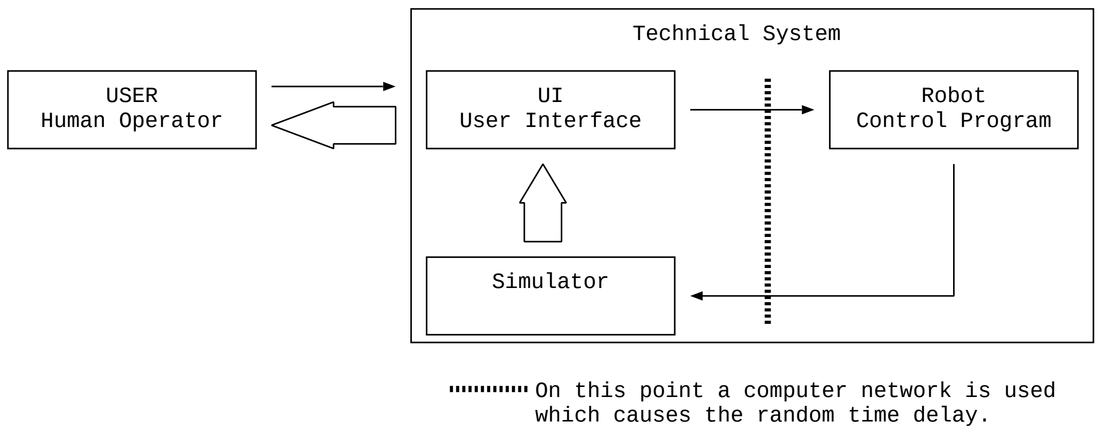


Figure 4: Different Levels of Processing which are necessary to control a Robot remotely: A Closed-Loop-System with Feedback through a Simulator.

This solution makes a simulation of the reality necessary. However, the best simulation is still just a simulation and not reality itself. For this reason reality and simulation may differ. Maybe the physical attributes of an object are calculated in a wrong way or the feedback is not accurate enough and an object is pictured in an other place than it really is. This can be a safety risk which has to be considered.

2.6 Safety

According to the British Standard (1992) Industrial robots – Recommendations for safety, a single point of failure must not cause any hazards. For this reason it is very important to stop the robot if the network link breaks down. The technique of monitoring the line is well suited to this application.

3 Technical Approach

3.1 The Ping Measurement

To explore the feasibility of using the Internet for remote control of robots the ping measurement was conducted. Over one week the round trip time (RTT) to the destinations was measured every minute. The gathered data was analysed statistically. A second ping measurement was conducted to proof the assumption that an overload of the local network causes the majority of the time delay.

3.1.1 Ping

Ping is a tool which sends ICMP[2] echo requests to the destination. The destination computer echos the request package (sends it back to the sender). The initiator of the ping request measures the time between sending the ping and receiving its echo. This is the RTT, the time which is necessary to transfer data to the destination and back. The additional processing time on the destination is immanent. This value can be neglected because it is very short in comparison to the transfer time. The fault caused by neglecting this time minimises when the whole ping time increases.

By default a ping package has an overall size of 84 bytes and contains the following parts: the IP header (20 Bytes) which specifies the source and target (IP) address and some network management information. The ICMP header (8 Bytes) contains a sequence-number, a checksum, and an identifier. The last part is the data part (56 Bytes) which contains a timestamp and filling-bytes. (Forouzan 2001; Kozierok 2004; Berkeley 1996) Over one week 10080 pings were sent which correspond to 846720 Bytes or 826 kByte in one direction per destination. The same amount of data was sent back.

3.1.2 Source – Destinations

The pings were sent from the author's private server which is located in the 'Schulzentrum Utbremen' a school in Bremen, Germany. It is connected through

[2]The Internet Control Message Protocol works at the same level as the UDP and TCP – it uses the Network Layer, the Internet Protocol (IP). It is used for troubleshooting and to announce network errors and timeouts. Please refer to RFC 792 (Postel 1981) for details.

the local school network and the network of the University of Bremen to the Internet. This server was used because the network of the LSBU does not allow to send pings to external computers.

All distances were calculated by using Global Positioning System (GPS) coordinates. The short overview about GPS coordinates given by Guthrie (2004) was used. All GPS coordinates were obtained from Maptech (2005) except the Unisa one which was adapted from Kennington (2000). It was not possible to obtain any exact GPS coordinates from the servers which were utilised. Coordinates from the home city or near objects were used. For this reason the accuracy lays far beyond the GPS accuracy. In addition to this the calculation results are air-line distances and not the real lengths of the cable of the transmissions.

To calculate the distance the following steps needed to be performed. Calculation of the difference between the latitude of the destination and the latitude of Bremen and the difference between the longitudes. After this the Pythagorean Theorem was used to calculate the distance (the hypotenuse). The following formula was used:

$$distance = \sqrt{(latitude_1 - latitude_2)^2 + (longitude_1 - longitude_2)^2}$$

The destinations were selected to give a wide spectrum of distances. It was assumed, that the servers of the destination organisations were located within the organisation's main building or near to it. It was not possible to locate a LSBU server which echoes pings. For this reason the LSBU did not become a destination.

Servername	Name of Organisation	Distance
www.unisa.edu.au	University of South Australia	17,000 km
www.harvard.edu	Harvard University (USA)	8,800 km
www.nationalgallery.org.uk	National Gallery of United Kingdom	1,000 km
www.tu-dresden.de	University of Dresden (Germany)	460 km
mail.hs-bremen.de	Mailserver of the University of Applied Science Bremen (Germany)	2 km

Table 1: Destinations which were used in the Ping Measurement.

For details about the results and analyse of the ping measurement please refer to the section 4.1 (page 28).

3.2 Basic Concept

The first thing which was developed during this project was the basic concept. The network library provides the communication tools to the robot and to the server control program. A user-command takes the following way:

A user enters a command into the user interface of the GUI/server. The server control program reads a command from the user through the user interface, evaluates it, and sends it through the IP network (the Internet) to the other side by using the functions of the network library. The network library on the robot's side receives the command and hands it over to the robot control program. The robot control program executes the command and controls the robot. The feedback from the robot follows this way in the other direction.

It is difficult to clearly distinguish between these levels. For example: there is no boarder between the GUI and the server control program in this project. In addition to this the robot's side includes a GUI to. This GUI is implemented to simulate the assumed position of the robot. (In the real world at least an emergency stop key has to be implemented on this side.) The name of this program (robot and its GUI) is `guirobot`. The name of the server control program (and its GUI) is `guiserver`.

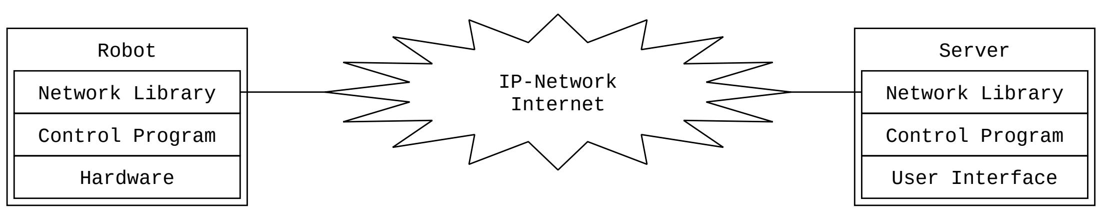


Figure 5: Basic Structure (Concept) which is assumed in this Project

This communication has to be bidirectional. It must be possible to request data from the robot. This could be the acknowledgement for a command, obtaining data of the robots environment, and to monitor the robot.

3.3 Implementation of the Library

The library was implemented on the free operating system GNU Linux because of the following reasons:

- It is free software. The term "free software" is defined by the Free Software

Foundation (2004) as these four rights:

1. "The freedom to run the program, for any purpose.
2. The freedom to study how the program works, and adapt it to your needs. Access to the source code is a precondition for this.
3. The freedom to redistribute copies so you can help your neighbour.
4. The freedom to improve the program, and release your improvements to the public, so that the whole community benefits. Access to the source code is a precondition for this."

- Because of these four rights a complete Linux system is available for free (without charge).
- Special versions of Linux which are working on small computers, like Real-Time-Linux (RTLinux) on embedded systems, are available. "A Linux system can actually be adapted to work with as little as 256 KB ROM and 512 KB RAM" (Addison 2001). This is essential because the robot control program often has to run on this kind of computer.

3.3.1 Network Layers

After the basic concept was clarified, the network library was implemented. The picture 5 does not show the entire truth. The network library by itself cannot transfer data through a network. It has to use lower levels.

During this project it was decided to used TCP for the data transmission. (Please refer section 2.5.1 on page 6.) The TCP and the levels underneath had to be implemented as well. This is not part of this project. Fortunately this was done before and it is now possible to use the implementation in the Linux-Network-Stack. In addition to the network implementation in Linux there must be some hardware in form of a Network Interface Card (NIC) and some network facilities like cables and hubs. Figure 6 gives an overview about the network layers which were used in this project.

3.3.2 Usage of the Linux Network Implementation

The Linux kernel provides an interface in form of system calls to allow (user mode) programs to use its facilities. This section gives an overview about these

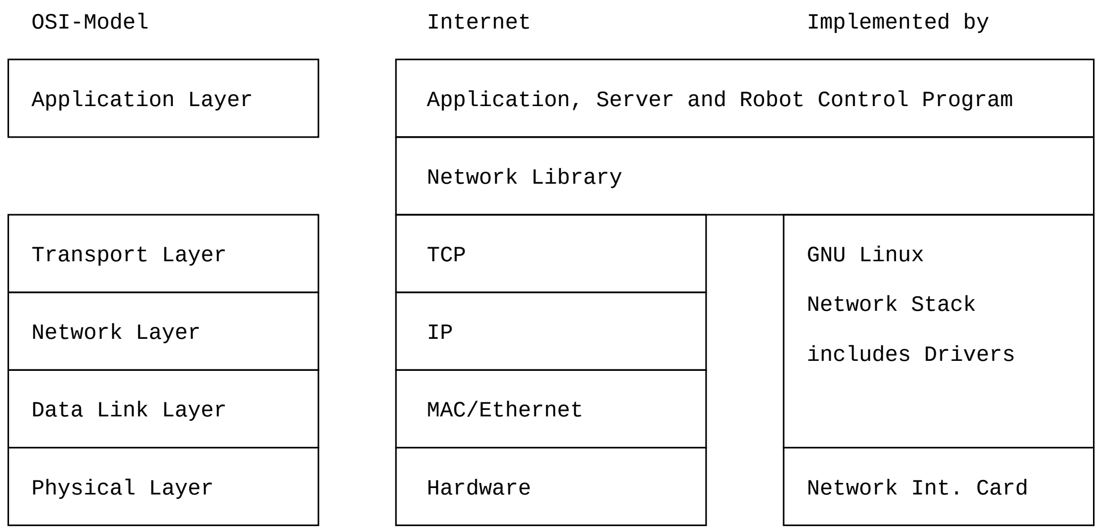


Figure 6: Overview of the used Network Layers

system calls and the way in which they were used in this project. For more detailed information please refer to IBM (1995).

Figure 7 shows on outline of the steps which are necessary to establish, to use, and to disconnect a communication link – a TCP socket stream.

It is possible to `accept()` more than one connection on a bound port. For this reason it is necessary to distinguish between the socket which is bound to the port and those for incoming connections. For each new connection a new socket is generated and given back by this system call.

The communication (in `Open / Usage`) between both sides is duplex (bidirectional). The duplex mode (pseudo-, half-, or full-duplex) depends on the underlying network equipment. There is no prescribed order in which the sides have to call `send()` and `recv()`.

The functions `accept()`, `recv()`, and `send()` will block[3] by default if there is no connection to accept, no data to receive (no data was sent), or the sent-buffer is full (cannot absorb more data). This behaviour can be changed with `fcntl()`.

For a detailed description of the system calls used please refer to the manual pages within the 'Linux Programmer's Manual'.

[3]If a system call cannot be completed because not all necessary data is received it waits until it can be completed. This causes a suspending of the calling function. This is called: "the function is blocked".

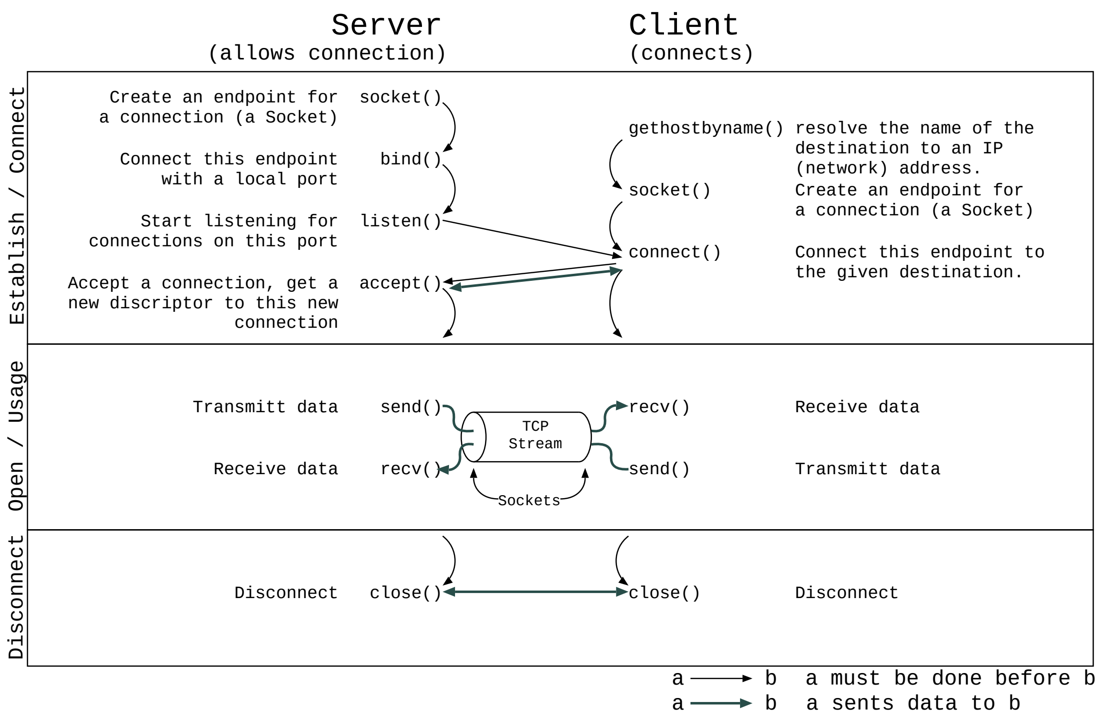


Figure 7: Life Cycle of a TCP Socket Stream

To simplify the process of establishing a socket-stream connection the following functions were implemented:

- `socket_accept()`: Start a new thread, wait for connections, and call a function when someone connects (server side).
- `socket_bind()`: Bind a socket to a port (server side).
- `socket_connect()`: Connect a TCP-stream to a server (client side).

In addition to this it was necessary to implement several sub-functions. All these functions can be found in the file `src/lib/libcomm.c` (appendices, section C.2 on page 69).

For a detailed description with parameters and return values of the listed functions please refer to the Application Programmer Interface (API) of the network library – libcomm – in the appendices section A.4 (page 53).

Please refer to section 4.3.1 (page 32) for details about the tests which were conducted to proof the correct behaviour of this functions.

3.3.3 Block Transfer Functions

Any data which is sent to a socket that is connected to a TCP stream will be transfered to the socket on the other side of this stream. The lower levels take care about the integrity of the data. The TCP monitors and corrects the order of the data and its integrity. This is important because data packages may follow different routes through the network or packages are lost and must be retransmitted. In both cases the packages need to be re-sorted on the receiver's side. If the connection is broken due to a network fault, `recv()` and `send()` will return an error.

For control purposes often blocks need to be transfered. A block in this context is a unit of data. For example: target coordinates if form of two integers for x and y. If the size of the datablock is constant it is simple to receive or transmit it:

```
recv(fd, buf, n, MSG_WAITALL);

send(fd, (void *) buf, n, 0);
```

In this example `fd` describes the used socket, `buf` is a pointer to the block, and `n` the number of bytes to be sent or received.

`MSG_WAITALL` tells the function to wait until all n bytes are received. A problem may occur at this point: TCP is a stream protocol and acts in this manner. It guarantees that the bytes are in the right order. But it does not guarantee that if m * n bytes were sent, it will be received as m * n bytes. If, for example, two blocks with 10 bytes each were transmitted it is possibly received in one block of 20 bytes, two blocks with 5 and 15 bytes, or three blocks ...

This problem is caused by the transparency of the network stack. The higher levels do not know what the lower ones do. In addition to this TCP buffers incoming and outgoing data. Once the `send()` is called the behaviour of the network stack depends on many things. For example: buffer size, network load, and speed. On the receiver's side all received data is stored in a buffer. `recv()` can load already received data from this buffer or it has to wait for some data to be received. This behaviour can be configured as mentioned earlier.

If different size blocks are possible it can be tricky to distinguish between two blocks because there is no way to know what block-size was used. To bypass this problem the block functions were implemented.

The block functions are transferring blocks according the following protocol. This list shows the transmitted parts and their order.

1. 2 Bytes*: Type: type of the datablock, can be chosen by the user of the function.
2. 2 Bytes*: Length: length of the datablock.
3. n Bytes: The datablock itself.

*) These are two byte-values used as a 16 bit integer. For this reason these values can vary in the range between 0 and 65535. A consequence of this is that the maximal size of a datablock is 65535 bytes. In addition to this the integer must be organised in the same way on both sides (GUI/server and robot). This can be tested be the test-program `src/tests/test002integer.c` (appendices, section G.2 on page 118).

In the programming language C data blocks are handled as pointers to the first unit (in this case a byte). There is no possibility to know how many units need to be processed if only the pointer is given. For this reason the block functions need to handle the size as well. Figure 8 gives a schematic of the basic block function.

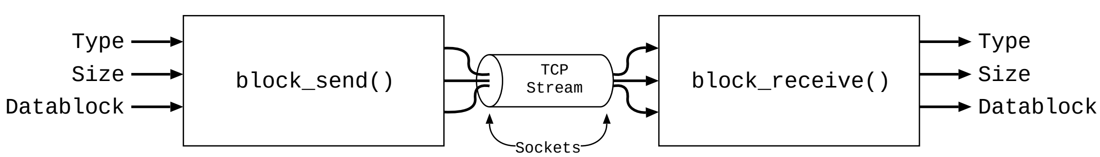


Figure 8: Basic Concept of the Block Functions

Implemented functions:

- `block_send()`: Send a datablock. The function blocks until the whole block is transfered to the buffer. If the buffer is full, data has to be sent first before it can continue.
- `block_receive()`: Receive a block. This function blocks until a whole block is received.
- `block_ifdata()`: Tests if there is data in the receiving buffer. The result of this test is returned.

- `block_receive_poll()`: Starts receiving a block if there is any data in the buffer. The function waits until the whole block is received. If there is no data in the buffer, an error-code is returned.
- `block_call()`: Starts a thread (goes to background, the calling function can continue) and waits for a block to be received. When this event occurs a given function is called to process this received datablock.

In addition to this several subfunctions were implemented. All these functions can be found in the file `src/lib/libcomm.c` (appendices, section C.2 on page 69).

For a detailed description with parameters and return values of the listed functions please refer to the API of the network library – libcomm – in the appendices section A.4 (page 53).

Please refer to section 4.3.2 (page 33) for details about the tests which were conducted to proof the correct behaviour of this functions.

3.3.4 Line Monitoring Functions

In this project it was decided to implement a set of functions to observe the network link (line). As explained in section 2.5.5 (page 9) this functions should be able to recognise if the network link becomes too slow to be used for remote controlling. In this case a function which takes appropriate actions (stop the robot) must be called.

The basic concept of these functions was adapted from ping. A data packet is sent to the other side which echoes it (sends it back to the original sender). The time between the "ping" launch and the arrival of its echo is measured. If this time exceeds a specific value a exception-function is called.

This implementation uses a TCP socket streams (not ICMP which is used by the original "ping") to transmit one-byte messages. Because of the different layers which are in use (TCP, IP, and Ethernet) the size of the data package increases to 67 bytes (1 byte data, 32 byte TCP, 20 byte IP, and 14 byte Ethernet).

It can be helpful to distinguish between two levels of real time exceptions (timeouts):

1. Soft Real Time Exception: If this time was exceeded no serious consequences can occur. It can be ignored but it is an indicator that something is going wrong, possibly a forewarn. If too many of these timeouts occur together they can become a hard real time exception.
2. Hard Real Time Exception: If this time limit is exceeded an uncorrectable error is assumed. An appropriate action is to shut the system down (in particular the robot) to a safe state.

The following functions were implemented:

- `linemonitor()` this function connects the given server on the given port and starts sending "pings". After one byte (used as a ping) was lunched, the function calls `poll()`[4] to determine if the answer (echo) arrives within the soft-timeout. If this was not the case a specified exception function is called and `poll()` will be called again. It determines if the answer is received within hard-timeout (hard-timeout is used as an offset value based on soft-timeout). If this answer was received in time the function waits a specified time before is sends the next ping. If this does not happen the exception function is called.

  In addition to this the exception-function is called if the connection breaks down, an emergency-stop-code was received, or an invalid answer (answer (echo) differs from request (ping)) was received. This function should run on the dangerous side (robot side) because the real-time-timeouts are more accurate than within the server function.

- `linemonitor_server()` this function is the server counterpart to the earlier mentioned function. It opens a specified port, waits for a connection and echoes (sends back) all received data. This function is less accurate in recognising timeouts because the wait-time (time before the next ping is sent by the `linemonitor()`-function) has to be included. As mentioned this function should run on the less dangerous side because of this fact. This function does not send pings by itself, it only echoes the received data.

  After one ping is echoed the function calls `poll()` to determine if the next ping arrives within wait-time plus soft-timeout. If the time-limit was exceeded it calls the exception function and repeats this procedure for the hard-timeout. The function repeats this until it is terminated.

- `linemonitor_emergencystop()` sends an emergency-stop code which causes an exceptions within the `linemonitor()`-function.

[4]This is a system call which suspends the current function until data is received or a given timeout is exceeded. Please refer to the 'Linux Programmer's Manual' for a detailed description.

In addition to this, the function `linemonitor_thread()` as "background"-part of the `linemonitor()`-function was implemented. All these functions can be found in the file `src/lib/libcomm.c` (appendices, section C.2 on page 69).

Problems with the implementation It was planned to provide the accurate round trip time (RTT) which was taken by the "ping". The system call `select()` which waits for an event (for example incoming data) was used to realise this. This system call provides a possibility to request the time which the calling function was suspended. This functionality can be used to determine an exact value for RTT but it did not work. Maybe the function is not compatible with sockets. There was no direct hint about this in its manual page. For this reason the system call `poll()` was used instead. This system call does not allow to determine the exact time. It only states if the timeout was exceeded or not.

For a detailed description with parameters and return values of the listed functions please refer to the API of the network library – libcomm – in the appendices section A.4 (page 53).

Please refer to section 4.3.3 (page 33) for details about the tests which were conducted to proof the correct behaviour of these functions.

3.3.5 Authentication

It is essential that the robot can only be controlled by an authorised person. It must not be possible for anyone else to give commands to the robot. For this reason it is necessary to implement some kind of authentication. This was done by including the following functions in the library: `socket_md5auth()`, `getauthinfo()`, and `free_authinfo()`. These functions were tested (please refer to section 4.3.2 (page 33) for details) but not used in the demonstration. Because of the last point it was forgone to describe the used protocol in detail.

For a detailed description with parameters and return values of the listed functions please refer to the API of the network library – libcomm – in the appendices section A.4 (page 53).

The mentioned functions can be found in the file `src/lib/libcomm.c` (appendices, section C.2 on page 69).

3.4 Demonstration with a Real Robot

One objective of this project was to demonstrate the function of the library on a real robot. The following section describes how this was done.

3.4.1 The Robot

It was decided to use a Cartesian robot with two degrees of freedom. The robot itself is attached to a certain place but can move a platform in horizontal (x) and vertical (y) direction. The robot is driven by a pneumatic system which is controlled through electronic valves. There are four valves, one for each direction in both dimensions. To move the platform in a direction the assigned valve has to be opened. A valve opens at an operation voltage of 24V. The figure 9 gives a basic overview about the structure of the robot.

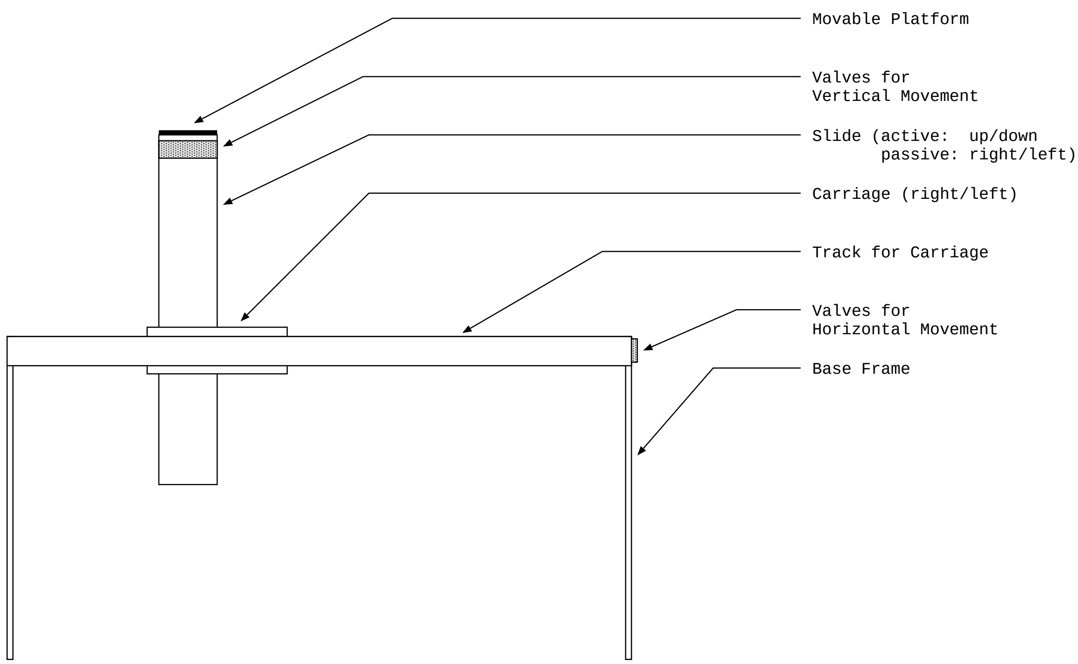


Figure 9: Basic Structure of the Robot

Unfortunately there was no interface, neither hardware nor software to control the robot with a Linux machine. Both was implemented in this project.

3.4.2 Hardware-Interface to the Robot

It was decided to use the parallel port to control the robot because only four actors needed to be switched on or off. A feedback from the robot was not intended. As mentioned earlier the valves to control the robot are driven by 24V. The valves consume about 100mA. This value was measured under operation at 24V.

The parallel port is neither able to deliver 100mA nor 24V. It works with 5V and can provide a few milliamperes. To connect the valves to the parallel port an amplifier is required.

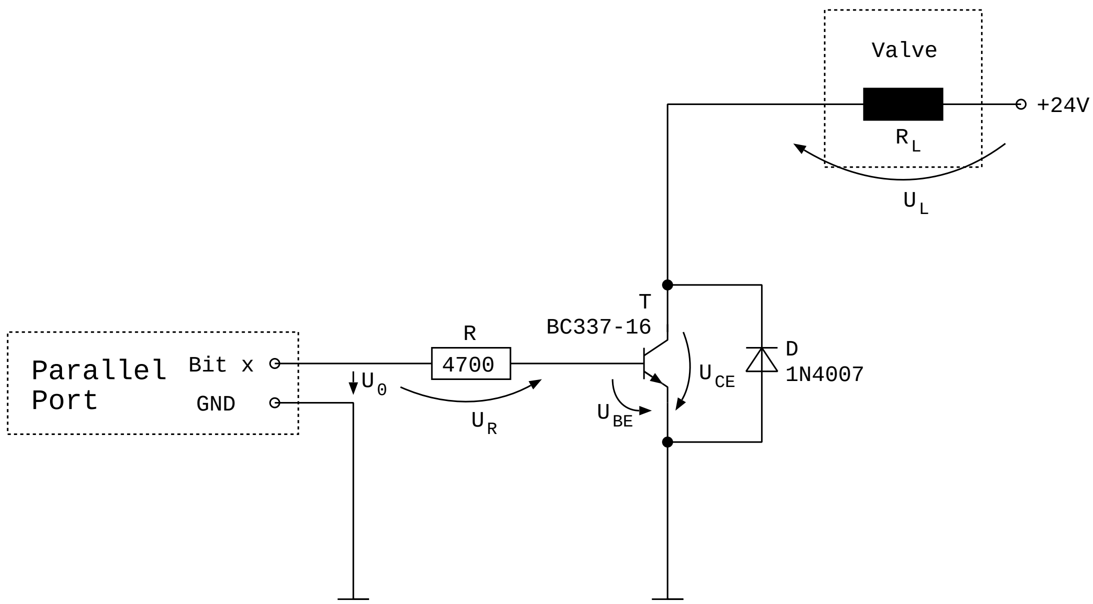


Figure 10: Circuit of the Hardware-Interface

Figure 10 shows the amplifier circuit which was designed to control the valves by using the parallel port. If the output of the parallel port is low (0 = 0V), U_R is zero either. If U_R is zero, no current flows into the base of the transistor. The transistor is closed, $U_{CE} \approx 24V$ and $U_{Vlave} \approx 0V$. The valve is closed. If the output is high (1 = 5V), $U_R \approx 4.3V$, $U_{BE} \approx 0.7V$. The transistor is open: $U_{CE} \approx 0V$ and $U_{Vlave} \approx 24V$. The valve is open. (Please refer to the following calculations.)

Calculation of the substitution resistor for the values. Used in the simulation. This value is only an approximation because of the inaccuracy cased by the power supply which generates U_L and the measurement of I_L.

$R_L = \frac{U_L}{I_L} = \frac{24V}{100mA} = 240\Omega$

Calculation of R. Assumption: parallel port is operating (Hight = 1) and generates $U_0 = 5V$; 1mA is sufficient to open the transistor entirely.

$$R = \frac{U_0 - U_{BC}}{I_B} = \frac{4.3V}{1mA} = 4300\Omega$$

The calculated resistor value is not available (in E1 series). For this reason it was decided to use 4700Ω. The reverse calculation:

$$I_B = I_R = I_{parallel-port} = \frac{U_0 - U_{BC}}{R} = \frac{4.3V}{4700} = 0.91mA$$

This value is acceptable.

Worst case calculation: if the transistor generates a short-circuit between C and B, 24V on B. (Parallel port delivers zero, 0V.)

$$I_R = I_{parallel-port} = -\frac{24V}{4700V} = -5.11mA$$

The parallel port should not be damaged by this current if this happens.

The diode (D) is used to protect the transistor in case of a high self-induction voltage. The magnetic field in an inductive element depends on the current through it and vice versa. If an inductive load is switched off, the magnetic field (which does not disappear in an infinitely short time) forces a current. If the transistor is closed the current cannot flow and charges are divided. A high voltage is generated which can destroy the transistor. The diode allows the current to flow, no charges are divided, no problem occurs. The induced current flows in the opposite direction as the operation current. The diode allows the current only in this direction to pass. Because of this the transistor is not bypassed during normal operation.

The circuit as shown was built four times, one time for each valve. The main challenge in this part of the project was to built this four amplifiers small enough to fix them info the parallel port plug. Table 2 explains in which way the interface to the robot is wired.

PIN	Bit	Operation
2	0	Move Up
3	1	Move Down
4	2	Move Right
5	3	Move Left
18	-	GND[a]

Table 2: Connection between the Parallel Port and the Robot's Actors.

- [a]) GND is an abbreviation for Ground which describes the common 0V-level.

Please refer to section 4.4.1 (page 34) for details about the tests which were conducted to proof the correct behaviour of this functions.

3.4.3 Software-Interface to the Robot

The software part of the interface was implemented to control the robot by using the hardware-interface.

According to Messmer and Dembowski (2003) the basis IO-port of the parallel port is by default located on the IO-address `0x378`. The eight output bits can be directly accessed through this address. The following eight addresses can be used to control other features of the parallel port.

Normally these IO-ports are only accessed by kernel drivers. These drivers provide an interface (for example, some special files in `/dev`) to user level programs. User programs access the hardware only through the kernel. This is done due to security aspects. If any program could access the hardware directly, it could bypass the access permission management of the system. For example: copying private files by directly accessing the harddrive.

To be able to directly access IO-ports under Linux the program has to have the right to do this. This right is reserved for programs which run with root (system administrator) privileges. Those programs can enable the access to the IO-ports by calling the system-call `iopl(3)`. After this was successful the IO-ports can be accessed by using `inb()` and `outb()`. `inb(p)` reads one byte from port p and returns it. `outb(v, p)` writes the value v to the port p. (Linux Programmer's Manual)

To prevent collisions between the software interface and conventional Linux drivers these drivers have to be unloaded:

- `parport_pc`: low level driver for the parallel port of a PC
- `parport`: general driver for parallel ports
- `lp`: driver for Line Printers

The software interface calculates the Δx and the Δy on the basis of given target coordinates and the stored position. The interface has to remember the last coordinates of the robot's platform because there is no feedback from the robot. There is no possibility for it to request the current position of the platform.

To know the start position of the platform the interface moves the y-axis to zero during the initialisation process. Because the position in y-direction is not known the interface assumes $y = 100\%$. If y is not 100%, the platform will hit its physical limit. This does not cause any problem. This procedure is not applied to the x-axis because it would physical damage the component if it is pushed beyond the given limit. It is for that reason why the x-axis is not moved during the initialisation process. $x = 50\%$ is assumed.

The interface assumes linear behaviour of the robot. This means that 50% of the time necessary to cover the whole distance is required to cover exactly 50% of the total range (independent from start-point and direction). Unfortunately the robot is not accurate enough. For this reason the process of moving the robot's platform to some target coordinates will produce a great discrepancy between the stored and the real values. This discrepancy increases with every movement because of the assumption that the stored coordinates (the result of the previous movement) were correct.

The following functions were implemented:

- `interface_init()` to initialise the interface
- `interface_driveto()` drives the robot to absolute coordinates.
- `interface_stop()` shuts the interface down.

Several subfunctions needed to be implemented to realize this functionality. These functions can be found in the file `src/example/interface.c` (appendices, section E.1 on page 95).

Please refer to section 4.4.2 (page 35) for details about the tests which were conducted to proof the correct behaviour of this functions.

3.4.4 GUI and Simulator

Some kind of user interface is necessary to control the robot. After Allegro (Hargreaves 2004), TCL (Unknown 2004), and GTK were considered, it was decided to use `GTK-2.0` to implement a Graphical User Interface (GUI). GTK is the GIMP Toolkit, a set of tools and libraries to implement GUIs. GIMP is the free GNU Image Manipulating Program. Both, GIMP and GTK are under LGPL[5]. (Blandford et al. 2004)

After a basic understanding of the functionality of GTK was gained an illustration of the robot was implemented. This illustration is used as an input to control the robot and as a simulation. This was realized as two GUIs: one on the server side (`guiserver`) which allows to manipulate the position of the robot's platform by clicking on in and moving it. The other on the client/robot side (`guirobot`) which shows (simulates) the current (assumed) position of the platform. There is no possibility to influence the position of the platform on this side.

The GUIs are using the functions of the network library to transfer the commands (destination position) over a network from the server to the robot. In addition to this the line monitor is used to observe the quality of the network connection. An emergency stop can be applied through the line monitor. If an emergency stop code is transmitted or the connection performance falls below a certain level (hard timeout occurs) the interface of the robot is shut down. The robot stops all movements immediately.

These function can be found in the following files:

- `src/example/guicommon.c` (appendices, section F.1 on page 101) draws the sketch of the robot and calculates the new coordinates which were given by mouse-inputs.
- `src/example/guirobot.c` (appendices, section F.2 on page 103): implementation of the robot control program and the simulator. This implementation uses the interface to the robot.
- `src/example/guiserver.c` (appendices, section F.3 on page 110): implementation of the remote control program. It reads commands from the user and transmits them over the network to the `guirobot`.

Please refer to the section 7.4 (page 50) for a more detailed description of the implemented GUIs.

The main challenge during the implementation of the GUIs was the re-drawing of the simulation (on the robot's side). A thread independent from GTK receives the new position from the server and calls the function which plots the

[5]GNU Lesser General Public License, please refer Free Software Foundation (1999)

sketch of the robot. Because of this GTK does not recognise that something was changed. The result of this is that the changes were not applied to the screen. It was difficult to find an appropriate solution to this problem. Many redraw-function do not work and the others were causing a 'Xlib: unexpected async reply' – a crash of the program. This happens because the GTK-thread and the independent thread were not synchronised. This synchronisation is now restored by calling `gdk_thread_enter()` before drawing the sketch of the robot.Then `gdk_window_process_all_update()` forces all components to be redrawn. After this `gdk_thread_leave()` unlocks the main thread. To use this functions the gdk-library (extension of GTK for platform independence) needs to be loaded and initialised.

Please refer to section section 4.5 (page 35) for details about the tests which were conducted to proof the correct behaviour of these functions.

4 Results and Discussion

4.1 Analysis of the Ping Measurement

The ping measurement was conducted twice:

1. from Mon, 01. November 2004 00:00 to Sun, 07. November 2004 23:59 – normal school week.
2. from Mon, 26. December 2004 00:00 to Sun, 01. January 2005 23:59 – holiday period.

The table 3 outlines the average of the results. It faces the distance to the destination with the minimum (`Min`), average (`Avg`) and maximum (`Max`) values for each destination and conducted measurement (`Try`). In addition to this the number of lost pings (`Lost [n]`) and the percentage related to the total number of pings (`Lost [%]`) is given for each destination and measurement.

The data was displayed in a diagram (on page 31) to gain a better understanding. The one week time span of the measurement is plotted on the x-axis with a main interval of one day. The unit of the y-axis is milliseconds. This axis represents the time which was required by the ping (RTT) to travel to the destination and back. The graphs were shifted on the y-axis in order to show all destinations in

Servername Organisation	Distance [km]	Try	Min [ms]	Avg [ms]	Max [ms]	Lost [n]	Lost [%]
www.unisa.edu.au	17,000	1st	358	431	9326	1432	14.21
Uni South Australia		2nd	350	370	712	40	0.40
www.harvard.edu	8,800	1st	120	158	1754	16	0.16
Harvard University (USA)		2nd	121	137	470	2	0.02
www.nationalgallery.org.uk	1,000	1st	47	83	1673	15	0.15
National Gallery of UK		2nd	48	64	401	3	0.03
www.tu-dresden.de	460	1st	21	59	1571	2	0.02
Uni Dresden (Germany)		2nd	21	36	374	2	0.02
mail.hs-bremen.de Uni	2	1st	6	39	1632	145	1.44
A.S. Bremen (Germany)		2nd	6	11	204	61	0.61

Table 3: Overview of the Results of the Ping Measurement.

one diagram. The arrangement of the graphs is equal to the order of the above listed destinations. The upper graphs where shifted by 6000*, 4000*, 2000*, and 1000.

*) The 0-level of these graphs were shifted to a grid line.

The first expected result of this measurement was that the time delay depends on the distance to the destination. This can be seen in the average values (table on page 29) as well as in the first diagram on page 31. The minimum, average, and maximum values on each measurement increase with the distance. The graphs in the diagram represent this by being shifted higher according to the distance.

The second observation was that the time delay for all destinations increased to very high values from around 8am to about noon and decreases from noon to 7pm to “normal” values. All graphs are following almost the same pattern. It was assumed that there is a change in the time delay depending on the daytime. However, the increases should have been shifted by the time difference to the time zone of the destination if the destination was to causes a countable amount of the time delay.

All curves have almost the same shape. Because of this it was assumed that the same reason caused the time delay for all destinations. When translating this to network-language, it means that all pings went through the same sub-network. There is only one sub-network which matches this characteristic: the local school and university sub-network through which the server sending the pings is connected to the Internet. After this network the pings took different routes.

To proof this assumption, that the workload of local school and university network was causing the majority of the time delays, a second measurement was conducted. To exclude the possibility of high work loads this measurement was conducted during the holiday period.

In comparison to the first measurement the time delay is almost stable. Except of some peaks on Monday which might have been caused by network maintenance.

4.2 Result of the Ping Measurement

As a conclusion of the ping measurement it can be said that the Internet can be used for remote control quite well as long as some assumptions are made:

1. The network link and the possible time delay must be explored before a statement about its usability can be made. The levels of time delay are changing from network link to network link and often even from hour to hour. This has to be well considered. After this analysis the behaviour (time delay) becomes well known. However, there is still a large random component because it is unpredictable how the unknown part of the network link will behave. Most of the network link is unknown.

2. There must be a possibility to observe the network link quality. Appropriate actions must be taken if the network link becomes unusable for remote control purposes. This is essential for safety reasons.

3. The bandwidth to the Internet must be wide enough to carry the workload without causing unacceptable time delays. The definition of unacceptable depends on the real time requirements of the remote control system. It is not a good idea to share the network-access with other parties because these parties may cause unpredictable workloads and time delays.

4. If the bandwidth is shared with some other parties, it may help to implement some priority system.

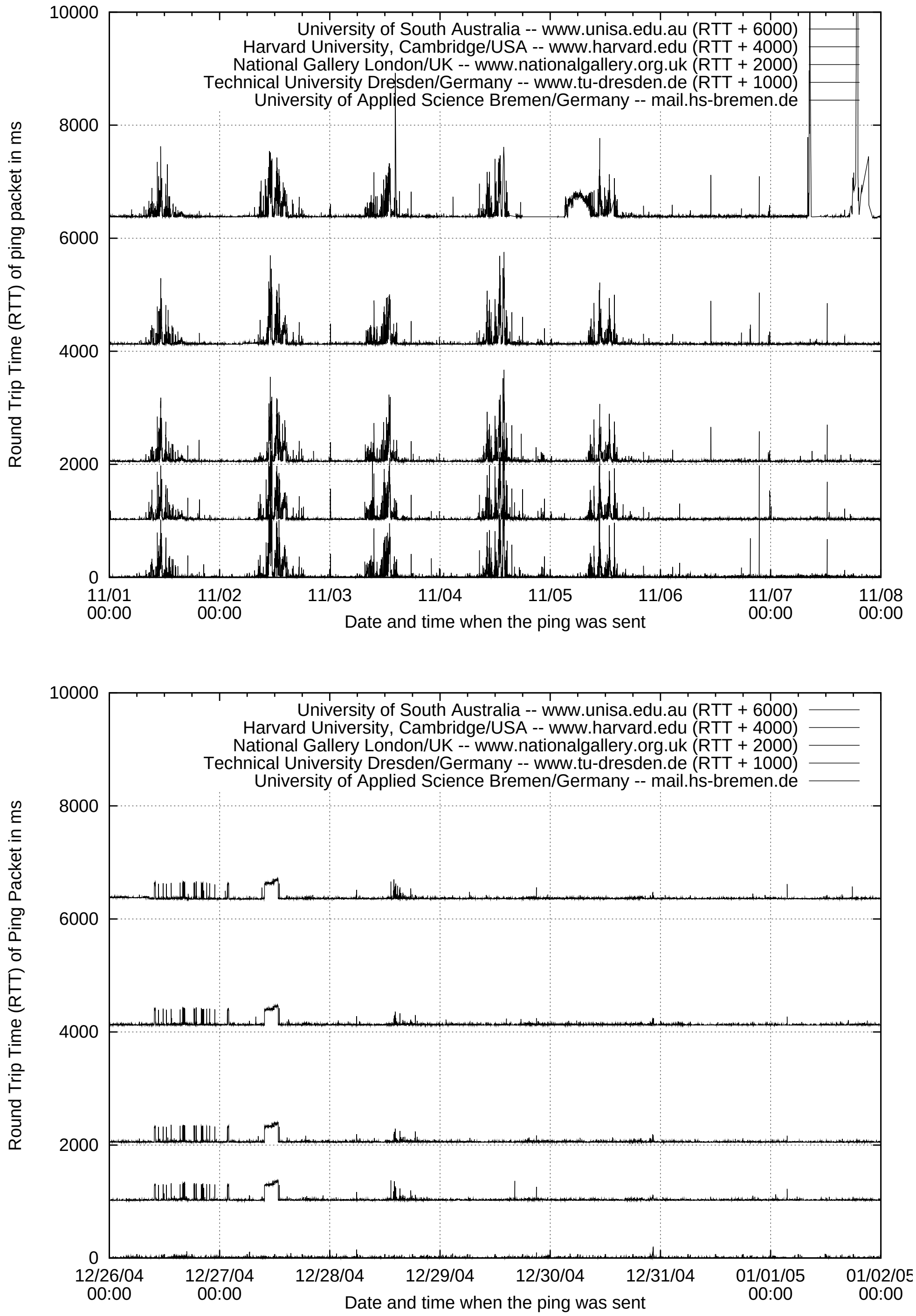


Figure 11: Results of the Ping Measurements. Top: First Measurement, Normal School Week. Bottom: Second Measurement during Holiday period.

4.3 Test of the Library

4.3.1 Basic Functions

To test the basic functions (`socket_bind()` and `socket_connect()`) the test-program `src/tests/test001sockets.c` (appendices, section G.1 on page 115) was implemented. The function `socket_accept()` was tested by the program `src/tests/test003block.c` (appendices, section G.3 on page 119), please refer to section 4.3.2 (page 33).

The program `test001sockets.c` implements a server and a client to test the network-functions on the loop-back-network[6] of the local machine. The server binds a port, receives one 8192-byte-block, inverts it, and sends it back. The client side connects to the server, sends a random-block, inverts it, receives a second block, and compares both. If both blocks (received and local inverted one) are equal it is assumed that the test was successful.

Destination	Test	Result
127.0.0.1[a]	connect to IP	OK
localhost[b]	resolution of a local name[c]	OK
lblacky[d]	resolution of a local name[c]	OK*
hofmeira.student.sbu.ac.uk[e]	Resolution by DNS[f]	OK*

Table 4: The Test-Results of the Basic (Socket) Function of the Network Library

*) These tests will only work, if the name of the local machine is equal to the mentioned name.

a) This IP-address exists on every computer and points to the loop-back-network to the local machine.

b) This name should exist on every computer and is in any case an alias of the local machine.

c) Uses /etc/hosts a host-IP-table to resolve the name of a computer to its IP address.

d) Name of the local computer (on which the tests were executed).

e) Name which is allocated to the local computer by the DNS[f] of the LSBU network.

[6] This network is in use if a computer establishes a connection to itself.

[f]) DNS stand for Domain Name Server. This is a system to manage unique global names.

4.3.2 Block Transfer Functions

The test-program `src/tests/test003block.c` (appendices, section G.3 on page 119) which tests block transfer functions works in almost the same way as the test-program for the basic socket functions. Changes are: the program connects localhost and uses the block transfer functions to transfer blocks.

The server-side program uses and tests this functions (in the following order)

1. `block_receive()`,
2. `block_receive_poll()`,
3. `block_receive_call()`, and
4. `socket_accept()` (which calls `block_receive_call()`)

to receive a block. `block_receive_send()` is used to send this block back. The client side only uses `block_receive_send()` and `block_receive_receive()`.

After these four tests are done, the authentication (functions `socket_md5auth()`) is tested as well. This is done by running one test and authenticating the connection before the block-transfer starts.

After some troubleshooting all functions worked properly. During the test 3 and 4 the error "recv(): Bad file descriptor" occurs because the thread still tries to receive after the client closes the connection. The thread will recognise (through this exception) if the connection is closed and terminate. This event is documented by the message "(server: connection terminated.)", which was perceived during the test.

4.3.3 Line Monitoring Functions

A small test-program `src/tests/tes005realtime.c` (appendices, section G.4 on page 125) which only implements the linemonitor-functions was used to test these functions. This program starts either the server or the client of the linemonitor-system depending on the parameters which were given:

- Client Mode:
  `run_tes005realtime server port soft_msec hard_msec wait_msec`
- Server Mode:
  `run_tes005realtime port soft_msec hard_msec wait_msec`

The client needs to know which `server` on which `port` has to be connected, while the server only needs to know which `port` to bind (and wait for incoming connections). Please refer to section A.4 (page 51) for a description of the remaining parameters.

To test if the line monitor detects when the time-limit was exceeded, the quality of the line was lessened by overloading the connection with pings[7] and by breaking the network connection trough unplugging it.

4.4 Test of the Interface to the Robot

4.4.1 Hardware-Interface

Before the hardware-interface was implemented the circuit was tested with Multi-SIM^{TM} 2001. The result of the simulation validated the results of the calculations in section 3.4.2 (page 23). This simulation used a substitution resistor (calculated as R_L) to simulate the valve.

Table 5 lists the measurements which were conducted to test if the hardware was implemented properly.

After this test was completed, the valves were connected and the hardware was tested by setting the bits on the parallel port manually. In this test the robot was not moved, the pneumatic supply was off-line. It was tested if the valves were switching on when a bit was set. The valves indicate this by a red light and by switching-noise. The result of these tests was that the interface worked properly. All valves can be controlled.

[7] The ping-program can be configured to send pings without the default time delay of one second. This was used to overload the network link.

Test	Result
Connection from +24V to the valves	OK (low-resistance)
Connection from +24V to other components (Valves not connected)	OK (none)
Connections between the GNDs	OK (low-resistance)
Connection between parallel output bits and GND (both directions because of the diode in the transistor)	OK (high-resistance)
Connection between parallel output bits and the input of the valves (C of transistor), both directions	OK (high-resistance)
Connection between parallel output bits	OK (none)
Connection between valve inputs (Cs)	OK (none)

Table 5: Results of the First Test of the Hardware-Interface.

4.4.2 Software-Interface

The software-interface was tested by monitoring the bits of the parallel port. At this time the hardware-interface was not connected. After some troubleshooting the interface seemed to work properly.

To be able to run the final test on the entire interface a test-program (`src/example/test001interface.c`) (appendices, section G.5 on page 127) was written. This program initialises the interface (`interface_init()`) first. After this it reads (x,y) coordinates from the keyboard and hands them over to the interface (`interface_driveto()`). After a few mistakes were eliminated the interface worked properly.

The major mistake in this phase of the development was a misinterpretation of the parameter of `usleep()`[8]. Milliseconds instead of microsecond were used. As mentioned earlier, the movements of the robot's platform are not linear. For this reason the interface does not work accurately.

4.5 Test of the GUI and the Simulator

The implementation and the testing of the GUI were running almost at the same time. All newly included details were checked when they were ready. These tests were conducted by the author personally because there is no point in writing

[8] System-call which suspends the calling function for a given time (unit: microseconds).

a test-program to test the interface to the user. The user has to decide if the interface works properly or not.

The GUI was implemented and tested in these steps:

1. Open a (program) window and draw the sketch of the robot in it.
2. Read the commands from the user. The new position of the robot's platform can be entered by moving the sketch (with the mouse) on the screen.
3. Transmit the new coordinates over a network to the other side and apply them to the simulated sketch.
4. Apply the new coordinate to the robot by using the interface.

All these steps are fully implemented now. The system works.

5 Conclusions and Recommendations for Further Work

5.1 Project Conclusions

- During this project the possibility of using the Internet for remote controlling of robots was re-examed. This was done by conducting an analysis of an example connection over the Internet. One conclusion has been drawn that the majority of the time delays (the most important restriction) were caused by the local network but not by the Internet. This theoretical result may be useful to future study in this field.
- Several approaches to the restrictions were studied and a promising method, the line monitor, was implemented.
- This project implements a network library which makes it possible to control a robot over the Internet. This library was demonstrated on a real robot by implementing a Cartesian robotic system. It includes a server side which reads commands from the user and transmits them through the Internet to the robot's side, where the commands are received to control the robot.

5.2 Personal Conclusions

- The three most challenging aspects for me during this project were to organise myself (take responsibility), to document my work well (write logbook and reports), and to do this in English. I have gained the confidence to pursue future studies in the proper ways.
- After completing this project I have realised that an aim and plan is vital for a project. The good plan will help the student to elaborate the study direction.

5.3 Recommendations for Further Work

- Porting the library to other platforms. All system calls which were used should be available within GDK (platform independent development library, extension of GTK). For this reason it should be possible to use this library and make the network library platform independent. But this has to be well considered because of the performance. It may not be recommendable to use a platform independent implementation on the robot's side because of the overhead which is caused by this independency. If the robot uses an embedded system with Real-Time-Linux, maybe there are not enough resources to use GDK. On the other side – the server side – this is completely different because of the performance of today's computer.
- Implementation of other strategies to bypass the random time delay. This can help to make the library more useful for additional applications.
- Students spent a lot of work and time to produce reports like this. It might be a great contribution to provide them on the Internet and it could help a lot of other people.

6 Bibliography and References

Addison, D. (2001) Embedded Linux applications: An overview [Online] Available at http://www-106.ibm.com/developerworks/linux/library/l-embl.html (accessed 20. October 2004)

Andreu, D.; Fraisse, P.; Roqueta, V.; Zapata, R. (2003) Internet enhanced teleoperation toward a remote supervised delay regulator *IEEE International Conference on Industrial Technology* 10-12 December 2003 p663-668 volume 2 [Online] Available at http://0-ieeexplore.ieee.org.lispac.lsbu.ac.uk/iel5/9059/28746/01290733.pdf (accessed 9. December 2004)

Ball, P.; Farnham, J; Iraca, S. (1999) Transmission Control Protocol/Internet Protocol TCP/IP [Online] Available at http://cne.gmu.edu/itcore/internet/tcpip/tcpip.html (accessed 11. December 2004)

Belousov, I. (2004) Internet Robotics [Online] Available at http://www.keldysh.ru/pages/i-robotics/operidea.html (accessed 26. November 2004)

Berkeley Distribution (1996) Ping Manual Page [Online] Available at http://snowhite.cis.uoguelph.ca/course_info/27420/ping.html (accessed 12. November 2004)

Blandford, J; Clasen, M; Janik, T; Lillqvist, T; Quintero, F.M.; Rietveld, K; Sandmann, S; Taylor, O; Wilhelmi, S (2005) GTK+ – The GIMP Toolkit [Online] Available at http://www.gtk.org/ (accessed 28. March 2005)

British Standard (1992) Industrial robots – Part 6: Recommendations for safety; BS 7228-6:1992; EN 775:1992; ISO 10218:1992 [Online] Available at http://bsonline.techindex.co.uk/ (accessed 28. October 2004)

Chen, Y.-M.; Chen, Y.-B. (2004) Research reform on real-time operating system based on Linux *WCICA 2004. Fifth World Congress on Intelligent Control and Automation* 5-19. June 2004 p3916-3920 Volume 5 [Online] Available at http://0-ieeexplore.ieee.org.lispac.lsbu.ac.uk/iel5/9294/29576/01342230.pdf (accessed 20. October 2004)

Elhaji, I.; Tan, J.; Xi, N.; Fung, W.K.; Liu, Y.H.; Kaga, T.; Hasegawa, Y.; Fukuda, T. (2000) Multi-site Internet-based cooperative control of robotic operations. *Proceedings 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2, 31. Oct - 5. Nov 2000 p826-831 [Online] Available at `http://0-ieeexplore.ieee.org.lispac.lsbu.ac.uk/iel5/7177/19309/00893121.pdf` (accessed 20. October 2004)

Engelen, P. (2004) GTK-Drawing Demo on Win32 [Online] Available at `http://mail.gnome.org/archives/gtk-app-devel-list/2004-October/msg00105.html` (accessed 28. March 2005)

Fairhurst, G. (2004) Carrier Sense Multiple Access with Collision Detection (CSMA/CD) [Online] Available at `http://www.erg.abdn.ac.uk/users/gorry/course/lan-pages/csma-cd.html` (accessed 11. December 2004)

Feibel, W. (1990) Using ANSI C in Unix. Berkeley: Osborne McGraw-Hill

Forouzan, B. A. (2001) Data Communication and Networking, 2^{nd} edition. New York: McGraw-Hill.

Free Software Foundation (1999) GNU Lesser General Public License. [Online] Available at `http://www.gnu.org/copyleft/lesser.html` (accessed 28. March 2005)

Free Software Foundation (2004) The Free Software Definition [Online] Available at `http://www.gnu.org/philosophy/free-sw.html` (accessed 28. March 2005)

Gale, T.; Main, I. (2000) GTK+ 1.2 Tutorial: Chapter 25. Scribble, A Simple Example Drawing Program [Online] Available at `http://www.johnmalone.org/gtk/tutorial/sec-eventhandling.html` (accessed 28. March 2005)

Guthrie, J (2004) Understanding GPS Coordinates [Online] Available at `http://www.co.lincoln.wa.us/GIS%20Data/Understanding%20GPS%20Coordinates.pdf` (accessed 04. February 2005)

Han, K.-H.; Kim, S.; Kim, Y.-J.; Lee, S.-E.; Kim, J.-H. (2001) Implementation of Internet-based personal robot with Internet control architecture. *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation* 2001 p217-222 volume 1 [Online] Available at
`http://0-ieeexplore.ieee.org.lispac.lsbu.ac.uk/iel5/7423/20179/00932556.pdf` (accessed 20. October 2004)

Hargreaves, S. (2004) Allegro – A game programming library.
[Online] Available at
`http://www.talula.demon.co.uk/allegro/` (accessed 28. November 2004)

Heesch, D. (2004) Doxygen: Introduction [Online] Available at
`http://www.stack.nl/ dimitri/doxygen/` (accessed 20. December 2004)

IBM Corporation (1995) TCP/IP Tutorial and Technical Overview: Ports and Sockets [Online] Available at
`http://www.auggy.mlnet.com/ibm/3376c210.html` (accessed 22. December 2004)

Kennington, A (2000) Alan Kennington's recommendations and suggestions
[Online] Available at
`http://www.topology.org/reco/` (accessed 04. February 2005)

Kernighan, B. W.; Ritchie, D. M. (1988) The C Programming Language, 2^{nd} edition. London: Prentice Hall ISBN 0-131-10362-8

Kozierok, C. M. (2004) The TCP/IP Guide [Online] Available at
`http://www.tcpipguide.com/free/index.htm` (accessed 12. November 2004)

Liu, P. X.; Meng, M.Q.-H.; Gu, J.; Yang, S.X.; Hu, C. (2003) Control and data transmission for Internet robots *Proceedings ICRA 2003. IEEE International Conference on Robotics and Automation* 14-19 September 2003 p1659-1664 volume 2 [Online] Available at
`http://0-ieeexplore.ieee.org.lispac.lsbu.ac.uk/iel5/8794/27834/01241832.pdf` (accessed 21. October 2004)

Liu, Y.; Chen, C.; Meng, M. (2000) A study on the teleoperation of robot systems via WWW *Canadian Conference on Electrical and Computer Engineering* 7-10. March 2000 p836-840 volume 2 [Online] Available at
`http://0-ieeexplore.ieee.org.lispac.lsbu.ac.uk/iel5/6844/18402/00849583.pdf` (accessed 20. October 2004)

LRAV/AVDAY (1985) Computer in Control, 01: Introducing the Robot. [Video]

Maptech, Inc. (2005) Online Maps: Map Server [Online] Available at
`http://mapserver.maptech.com/` (accessed 04. February 2005)

Mattis, P. (1998) The GIMP Toolkit (GTK Documentation) [Online] Available
`http://www.csa.iisc.ernet.in/old-website/Department_Resources/Hypertext/gtk/gtk_toc.html` (accessed 28. March 2005)

McKerrow, P. J. (1991) Introduction to Robotics. Singapore: Addison-Wesley. ISBN: 0-201-18240-8

Messmer, H.-P.; Dembowski, K (2003) PC-Hardwarebuch (German: PC-Hardwarebook) 7^{th} edition. Munich: Addison-Wesley. ISBN 3-827-32014-3.

Mitchell M.; Oldham J.; Samuel A. (2001) Advanced Linux Programming, New Riders Publishing [Online] Available at
`http://docs.linux.cz/programming/other/ALP/advanced-linux-programming.pdf` (accessed 20. December 2004)

Plauger, P.J.; Brodie, J. (1989) Standard C - A Reference. London: Prentice Hall. ISBN 0-134-36411-2

Postel, J. (1981) RFC 792 – Internet Control Message Protocol
[Online] Available at
`http://www.freesoft.org/CIE/RFC/792/index.htm`
(accessed 11. November 2004)

Sato, H.; Yakoh, T. (2000) A real-time communication mechanism for RTLinux *IECON 2000. 26th Annual Conference of the IEEE Industrial Electronics Society* 22-28. October 2000 p2437-2442 volume 4 [Online] Available at
`http://0-ieeexplore.ieee.org.lispac.lsbu.ac.uk/iel5/7662/20956/00972379.pdf`
(accessed 19. October 2004)

Schwarzenbach, J.; Gill, K.F. (1992) System Modelling and Control, 3rd edition. London: Edward Arnold. ISBN: 0-340-54379-5

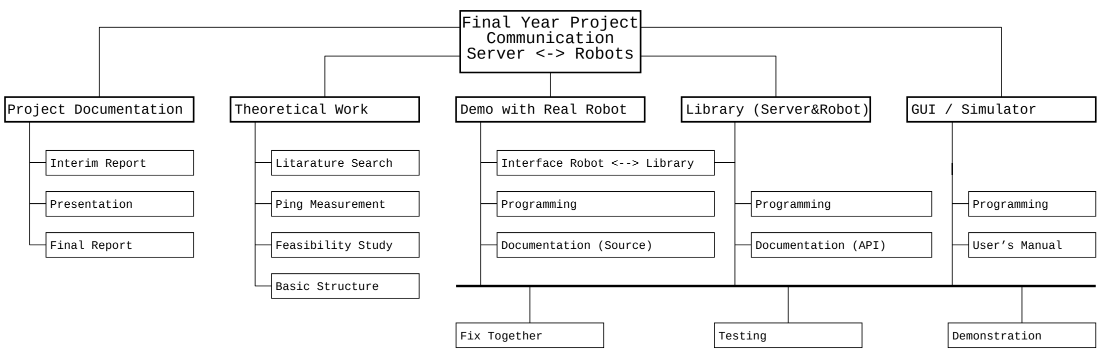
Unknown (2004) TCL Developer Xchange: tcl/tk [Online] Available at
`http://www.tcl.tk/software/tcltk/` (accessed 28. November 2004)

Wilson, G. (2001) OSI Model Layers [Online] Available at
`http://www.geocities.com/SiliconValley/Monitor/3131/ne/osimodel.html`
(accessed 11. December 2004)

7 Project Planning

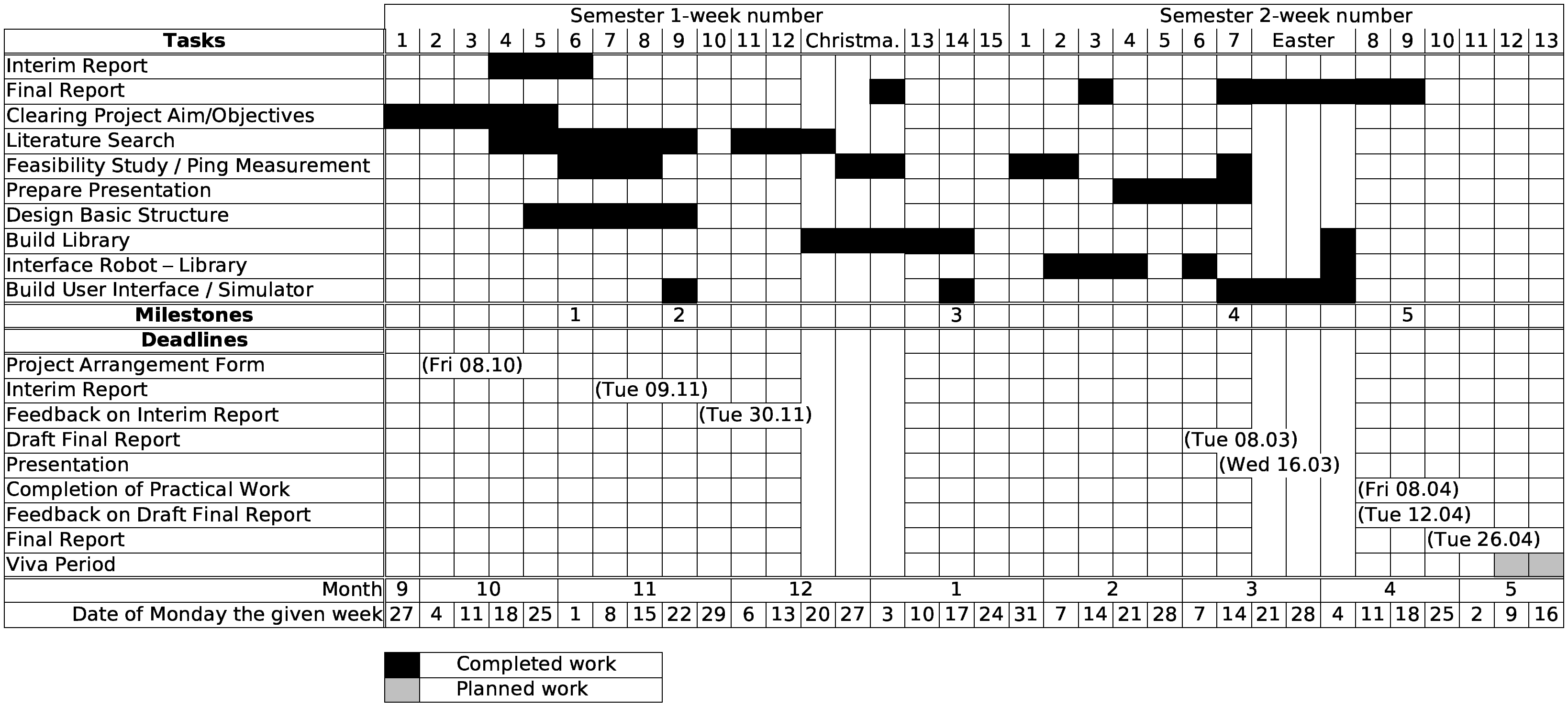
This section compares the planing of the project with its actual realisation. After the work breakdown structure is given, the final version of the Gantt char is shown. This is followed by the comparison of the Gantt charts and action plans from the beginning, the middle, and the end of the project. In the end the milestones and some explanations about the project and it's planning are given.

7.1 Work Breakdown



7.2 Gantt Chart of Final Stage

The Project Schedule – Actual Realisation



7.3 Project Schedule

7.3.1 Comparison: Pre and After Interim Stage

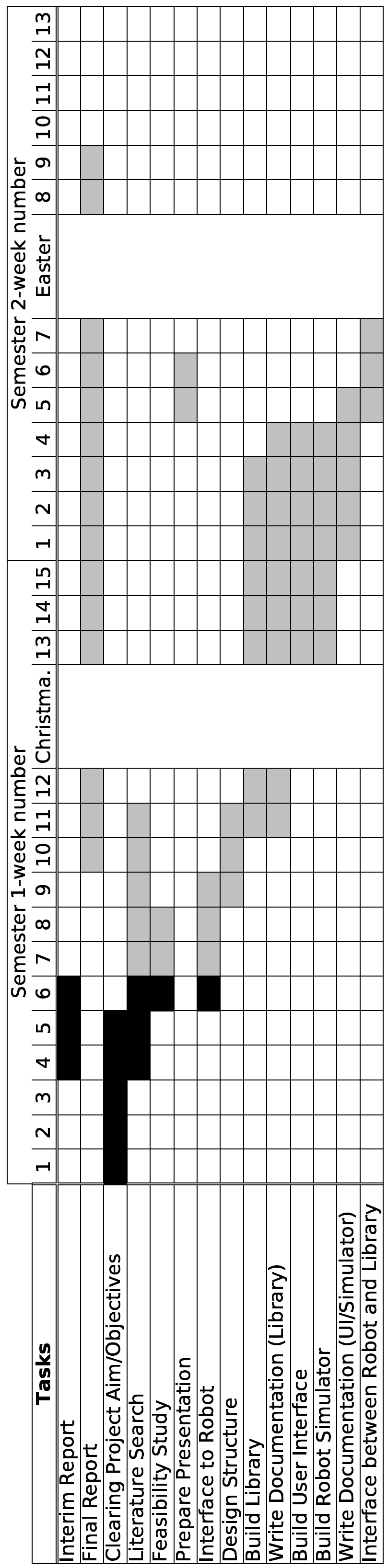
Action Plan Pre Interim Report

Task		Estimated Duration in Weeks	Precedence
A	Interim Report	3	-
B	Final Report	15	Feedback A
C	Clearing Project Aim/Objectives	5	-
D	Literature Search	8	-
E	Feasibility Study	3	C
F	Prepare Presentation	2	C,(E)
G	Interface to Robot	4	C
H	Design Structure	4	C
I	Build Library	8	(G)
J	Write Documentation (Library)	9	(I)
K	Build User Interface	7	(I)
L	Build Robot Simulator	7	(I)
M	Write Documentation (UI/Simulator)	5	(K),(L)
N	Interface between Robot and Library	3	G,(M)

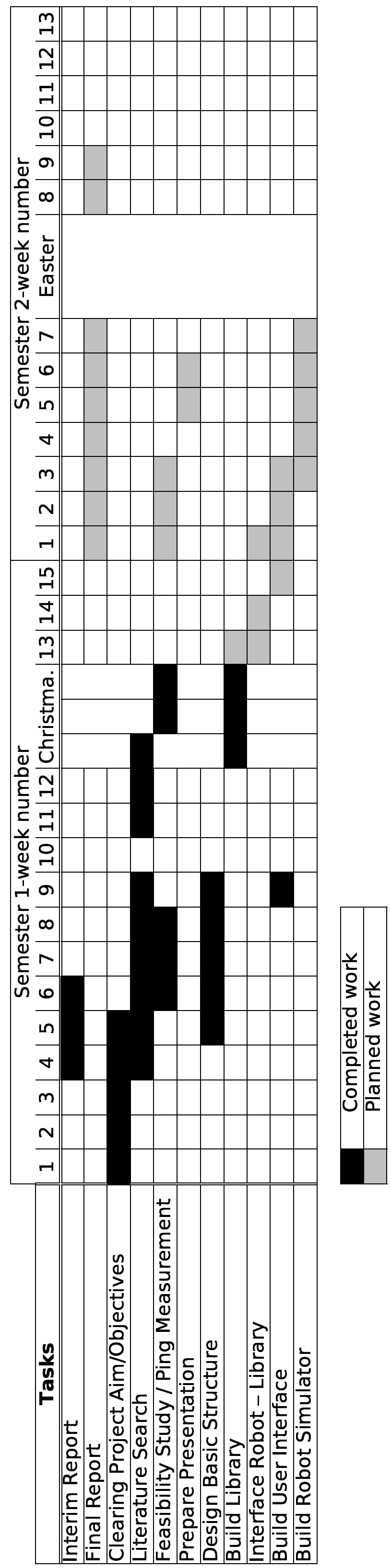
Action Plan After Interim Report

Task		Estimated Duration in Weeks	Precedence
A	Interim Report	3	-
B	Final Report	8	Feedback A
C	Clearing Project Aim/Objectives	5	-
D	Literature Search	9	-
E	Feasibility Study / Ping Measurement	8	C
F	Prepare Presentation	2	C,(E)
I	Build Library	4	C
K	Build User Interface	5	(I),(N)
L	Build Robot Simulator	5	(K)
N	Interface between Robot and Library	3	(I)

The Project Schedule – Interim Report



The Project Schedule – Christmas



7.3.2 Comparison: After Interim Stage and Final Stage

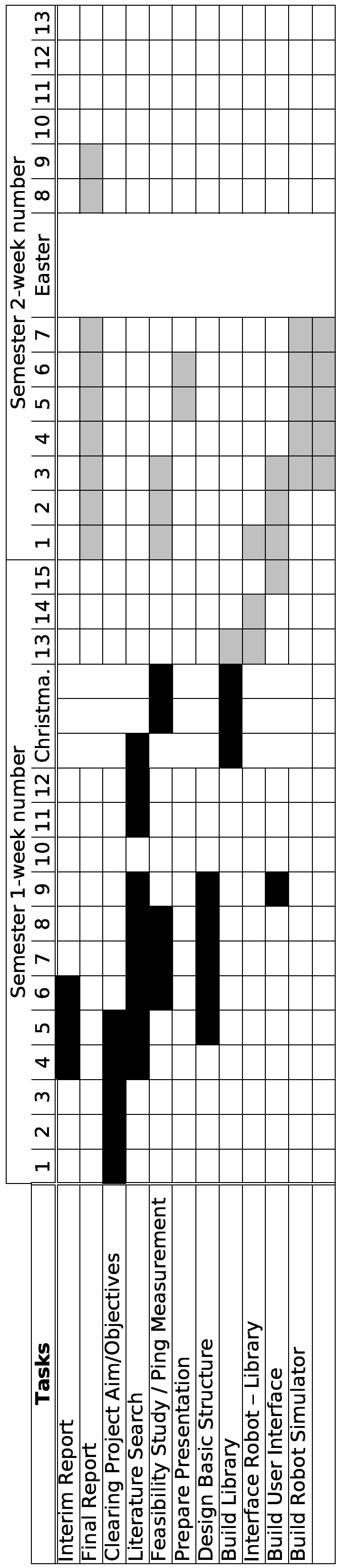
Action Plan After Interim Report

Task		Estimated Duration in Weeks	Precedence
A	Interim Report	3	-
B	Final Report	8	Feedback A
C	Clearing Project Aim/Objectives	5	-
D	Literature Search	9	-
E	Feasibility Study / Ping Measurement	8	C
F	Prepare Presentation	2	C,(E)
I	Build Library	4	C
K	Build User Interface	5	(I),(N)
L	Build Robot Simulator	5	(K)
N	Interface between Robot and Library	3	(I)

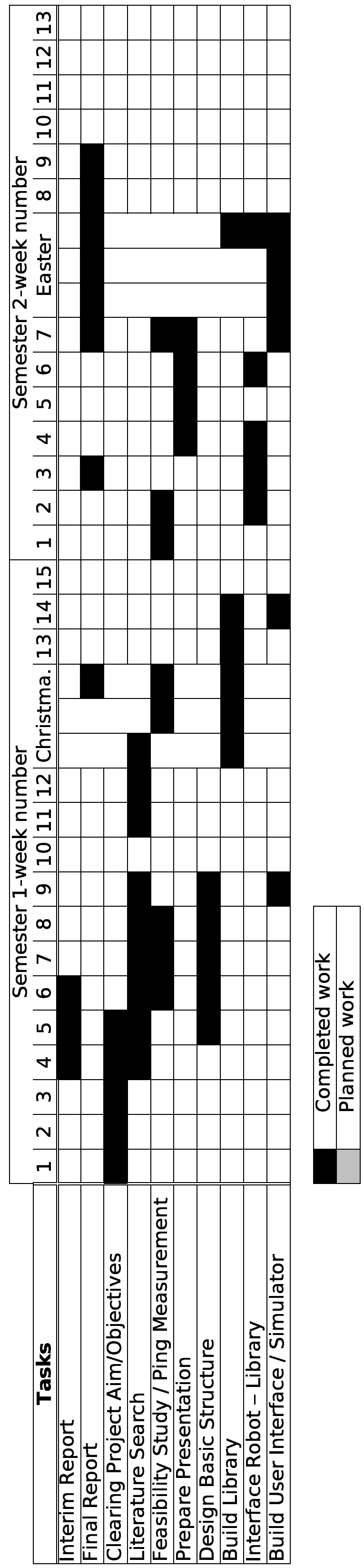
Final Action Plan

Task		Estimated Duration in Weeks	Precedence
A	Interim Report	3	-
B	Final Report	8	Feedback A
C	Clearing Project Aim/Objectives	5	-
D	Literature Search	9	-
E	Feasibility Study / Ping Measurement	8	C
F	Prepare Presentation	5	C,(E)
I	Build Library	6	C
K	Build User Interface / Simulator	6	(I),(N)
N	Interface between Robot and Library	5	(I)

The Project Schedule – Christmas



The Project Schedule – Actual Realisation



- Precedence (in action plans):

  X: Task X has to be completed before the task can start.

  (X): Task X has to be semi-completed before the task can start. That means that task X has to be in a state in which it is possible to start a new task simultaneous. Tasks which run simultaneous can have an influence among each other.

- The scale which was used to measure the duration (weeks) is inaccurate because of the fact that the work load was not allotted constantly over the project time. Often the time was shared between different tasks which were conducted at the same time. These tasks included project tasks, other study related tasks, and private tasks. However, it can be said, in all conscience, that at least 300 hours were spent in this project.

- The holidays were scheduled as a reserve in case that the project-work would take longer than expected. During the project it was decided to move some work into the holiday periods.

- It was chosen to condense the number of tasks to simplify the project planning. During this process the “write documentation”-tasks were included into the corresponding programming (building) task, the design of the structure was embedded into the library building process, and the “Interface to Robot” development task was included in the task “Interface Robot – Library”.

- The development of the “Interface to Robot” was moved from the beginning of the project to its middle because the decision about which robot should be used for the demonstration was delayed.

- After beginning the development of the GUI it was decided to use almost the same GUI for both sides. One (server-side) to input the new position of the robot’s platform and the other (robot’s side) to show (simulate) the behaviour of the robot. For this reason, both development phases were condensed to one.

7.4 Milestones

1. On Tuesday, 9 November 2004, the project is defined by now and has been started. This is reflected by the interim report which is completed and handed in.
2. At this point (9^{th} week of first semester) of the project it is possible to control the robot with a little experimental program. The interface to the robot is well understood.
3. In week 14 (first semester) an early simulation with library, user interface and simulator shows the basic function of the system.
4. At the end of week seven (second semester) the system works. It is now possible to control the robot over the Internet by using the library. This will be demonstrated. The simulation works as well.
5. On Tuesday, 26 April 2005, the project and final report are completed and handed in.

The milestones one and three were met. The last milestone will be also be met in time.

The milestones two and four were missed. As mentioned earlier the decision about the robot was delayed. This caused the missing of the second milestone. The fourth milestone was missed by almost three weeks because of the fact that the work was delayed by some neglected factors. The development of the interface to the robot was delayed. The time which was necessary to prepare the presentation was underestimated. And external events like exams interfered with the initial plan.

Appendix A: User's Manual GUI for Robot and Server

A.1 Both Programs (`guirobot` and `guiserver`) explained

- `guirobot`: This is the robot control program. It can run in two modes:
  - Simulation only: If no access to the hardware is possible (the program does not run under `root` (with system administrator rights), access to the IO-ports is not possible. The program recognise this and shows (simulates) the robot's movements only on the screen.
  - Simulation and Controlling: The IO-ports can be accessed, the interface is fully active. The program will show the new position of the robot on the screen and drive the robot to this position.
- `guiserver`: This is the server or the remote control station. It allows the user to input the new position of the robot. This new position will be transmitted to the `guirobot` by using `libcomm`.

The robot is shown (simulated) in both programs. The black rectangle pictures the platform of the robot which can change its position.

The `guirobot` has to be started first. In the second step the `guiserver` connects to the `guirobot`. After this connection is established, the `guirobot` connects to the `guiserver` to monitor the stability and the speed of the line with the `linemonitor()`. Both programs are looking almost identical:

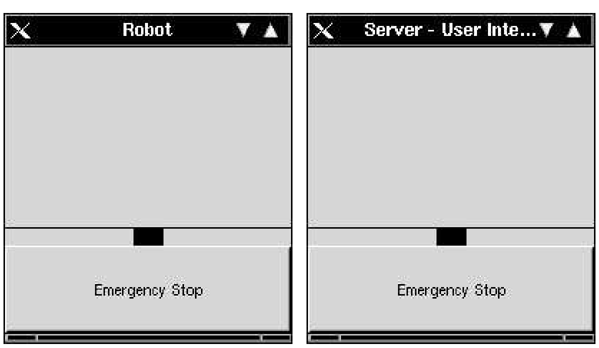


Figure 12: Both Programs `guirobot` (left) and `guiserver` (right) directly After Startup.

Their lookout (title line, buttons) depends on the used window manager and its configuration. In this case AfterStep is in use.

The difference between these two Graphical User Interfaces (GUIs) is that only the `guiserver` allows manipulations of the position of the robot's platform. The `guirobot` shows the actual position and drives the platform of the robot to the required position if the interface is active and can access the hardware.

A.2 Manipulate the Position of the Robot's Platform

Click (with the left mouse-button) on the robot's platform (sketched as black rectangle) and move it with held mouse button to the new position. Release the button. An example of the result of this can be seen in the following screenshot:

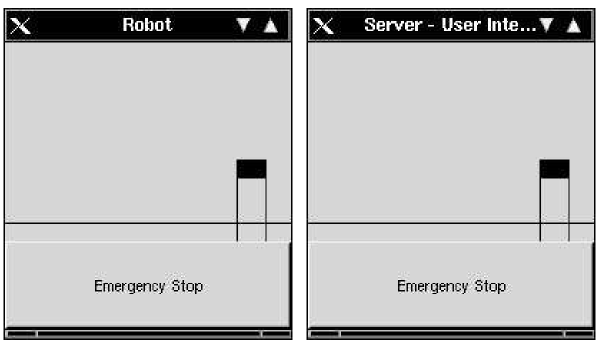


Figure 13: Both Programs `guirobot` (left) and `guiserver` (right) After a Movement of the Robot.

A.3 Stopping the Robot

Both programs are able to stop all movements of the robot by clicking "Emergency Stop". This emergency stop is also executed if the window is closed or the program receives a terminate signal.

In addition to this the linemonitor stops the robot if the the connection breaks down or if the server does not answer within a given time period.

A.4 Starting both Programs, Parameter

As mentioned the `guirobot` has to be started first. This program receives the following parameters:

```
guirobot port-to-bind soft_msec hard_msec wait_msec
```

- `port-to-bind` describes the port on which `guirobot` has to listen for connections from the `guiserver`. The `guirobot` will connect to the home address of the `guiserver` and `port-to-bind + 1` to establish a linemonitor connection.
- `soft_msec` tells the program how long it has to wait before a soft-timeout is assumed. A soft-timeout causes a message on the terminal. The value is specified in milliseconds.
- `hard_msec` tells the program how long it has to wait after a soft-timeout has occurred, before a hard-timeout is assumed. A hard-timeout causes an emergency-stop of the robot. The hard-timeout is assumed if there is no response after `soft_msec + hard_msec`. The value is specified in milliseconds.
- `wait_msec` specifies the time which has to past after the last response was received before a new enquiry is sent. The value is specified in milliseconds.

The interface initialises the robot. That means driving the robot's platform to the (x=50,y=0) coordinates. The interface assumes that the robot is already in the position x=50. The x-position will not change. However, this takes some time. The program is ready when the window is displayed.

After the window is displayed the `guiserver` should be started with the following parameters:

```
guiserver robot-address port soft_msec hard_msec wait_msec
```

- `robot-address` specifies the address of the computer on which the `guirobot`-program runs. This can be an IP-Address (Internet Protocol Address) or the name of the computer which must be resolvable by the used Domain Name Server (DNS).
- `port` correlates to `port-to-bind` from `guirobot` and must be the same.
- `soft_msec` correlates to `soft_msec` from `guirobot` and should be the same.
- `hrad_msec` correlates to `hard_msec` from `guirobot` and should be the same.
- `wait_msec` correlates to `wait_msec` from `guirobot` and should be the same.

Appendix B: API of the Network Library

libcomm.c(3) libcomm.c(3)

NAME

libcomm.c - Main part of libcomm.

SYNOPSIS

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <unistd.h>
#include <sys/poll.h>
#include <string.h>
#include 'libcomm.h'
#include 'md5.h'
#include <pthread.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
```

Functions

void socket_accept_thread (struct LIBCOMMPTHREADP *libcommpthreadp)
: This is a part of socket_accept() and must not called from the user.

int socket_accept (int sockport, int id, void(*socket_accept_do)(int fd, int id, char *pip, struct sockaddr_in their_addr))
: Start a new thread, wait for connections and start socket_accept_do() when someone connects.

int socket_bind (int port, int cqueue)
: Bind a socket to a port (Server side).

int socket_connect (char *host, int port)
: Connect a TCP-stream to a server (Client side).

char * block_random (char *buf, int size)
: Get random numbers/bytes.

void thread1 (struct LIBCOMMPTHREADS

*libcommpthreads)

This is a part of block_call() and must not called from the user.

int block_call (int fd, int id, int term, void(*block_call_do)(int fd, int id, unsigned int type, char *buf, unsigned int size, int term), void(*block_call_term)(int fd, int id))

Waits in a new thread for a datablock to be received and calls the function block_call_do() if this event occurs or block_call_term() when the connection terminates.

int block_ifdata (int fd)

This function tests if new data is available to read on a stream.

char * block_receive_poll (int fd, unsigned int *type, char *buf, unsigned int *size, unsigned int maxsize, int term)

Test if is there data available on the socket's input buffer and starts receiving a block if there is.

char * block_receive (int fd, unsigned int *type, char *buf, unsigned int *size, unsigned int maxsize, int term)

Receive a block (composition of: type, size of datablock and datablock) from a socket.

int block_receive_integer (int fd, unsigned int *recvi)

Receive an integer (two bytes; 16Bit) from the socket.

int block_receive_nbytes (int fd, char *buf, int n)

Receive n bytes from socket.

int block_send (int fd, unsigned int type, char *buf, unsigned int size)

Send a block (composition of: type, size of datablock and datablock (buf)) to a socket.

void free_authinfo (struct AUTHINFO *destroy)

Free the memory space which is used by an AUTHINFO structure.

int socket_md5auth (int fd, char *netname, char *name, struct AUTHINFO **plocallogin, struct AUTHINFO **premotelogin)

Do both side authentification.

AUTHINFO * getauthinfo (char *netname, char *name)

Load authentication informations (netname, name, passwd, keyencrypt, keydecrypt) from authfile.

void linemonitor_server_thread (struct

LINEMONITOR_THREAD_DATA
*linemonitor_thread_data)

Thread used by linemonitor_server() NOT for direct usage.

int linemonitor_server (int port, int soft_msec, int hard_msec, int wait_msec, void(*linemonitor_exception)(char *server, int port, int type))

Monitor if the 'line' is fast enough: Server Application.

void linemonitor_emergencystop (int sock)

Sends an 'Emergency Stop' to the client's side, linemonitor() will produce an 'Emergency Stop' exception (type 4).

int linemonitor_thread (struct
LINEMONITOR_THREAD_DATA
*linemonitor_thread_data)

Thread used by linemonitor() NOT for direct usage.

int linemonitor (char *server, int port, int soft_msec, int hard_msec, int wait_msec, void(*linemonitor_exception)(char *server, int port, int type))

Monitor if the 'line' is fast enough: Client/Robot Application.

DETAILED DESCRIPTION

Main part of libcomm.

FUNCTION DOCUMENTATION

int block_call (int fd, int id, int term, void(* block_call_do)(int fd, int id, unsigned int type, char *buf, unsigned int size, int term), void(* block_call_term)(int fd, int id))

Waits in a new thread for a datablock to be received and calls the function block_call_do() if this event occurs or block_call_term() when the connection terminates.

Parameters:

fd (int) descriptor of socket

id (int) arbitrary id of background process / thread term (int) 0: do not terminate the buffer, 1: terminate the buffer by appending a 0x00.

block_call_do
(int fd, int id, unsigned int type, char *buf, unsigned int size, int term) (function) this function is called if a datablock was received. fd, id and term are the same as in block_call(). type describes the type of the received datablock, buf is a pointer to this datablock and size is the number of bytes of the datablock

block_call_term
(int fd, int id) (function) this function is called if the connection terminates. fd and id are the same as in block_call().

Returns:
If all right zero otherwise non zero.

int block_ifdata (int fd)

This function tests if new data is available to read on a stream.

Parameters:

fd (int) discriptor of stream to test

Returns:
(int) 1: Data to read; 0: No data to read

char* block_random (char * buf, int size)

Get random numbers/bytes.

This function reads random numbers/bytes from /dev/urandom and stores this bytes in a buffer.

Parameters:

buf (char *) in which the bytes will be stored. If this parameter is equal to NULL dynamic memory will be allocated.

size an integer, specifies ths size of the buffer (the number of the random bytes). WARNING: If buf is not equal to null, n*(size) bytes will be stored in this buffer without any check of ths size of this buf.

Returns:
(char *) a pointer to the buffer in which the random bytes are stored.

char* block_receive (int fd, unsigned int * type, char * buf, unsigned int * size, unsigned int maxsize, int term)

Receive a block (composition of: type, size of datablock and datablock) from a socket.

Waits for a block to be received completely. WARNING: The integers (type and size; excluding fd) are only 16 bit values (0 - 65535).

Parameters:

fd (int) descriptor of socket

type (unsigned int *) pointer to integer, this value can be used as buyer's option

buf (char *) buffer for datablock. Memory will be allocated if this parameter is equal to null.

size (unsigned int *) pointer to integer in which the size of the received datablock is saved.

maxsize (unsigned int *) describes size of buf. This parameter will be ignored if buf is equal to null.

term (int) 0: do not terminate the buffer, 1: terminate the buffer by appending a 0x00.

Returns:
(char *) pointer to buffer which contains the received datablock; NULL if fail.

int block_receive_integer (int fd, unsigned int * recvi)

Receive an integer (two bytes; 16Bit) from the socket.

Parameters:

fd (int) descriptor of socket

recvi (unsigned int *) pointer to integer in which the received integer is saved.

Returns:
(int) 2: OK; -1: fail

int block_receive_nbytes (int fd, char * buf, int n)

Receive n bytes from socket.

Parameters:

fd (integer) descriptor of socket

buf (char *) buffer for saving the received bytes

n (integer) number of bytes to receive

Returns:
(integer) n: OK; -1 fial

char* block_receive_poll (int fd, unsigned int * type, char * buf, unsigned int * size, unsigned int maxsize, int term)

Test if is there data available on the socket's input buffer and starts receiving a block if there is.

WARNING: The integers (type and size; excluding fd) are only 16 bit values (0 - 65535).

Parameters:

fd (int) descriptor of socket

type (unsigned int *) pointer to integer, this value can be used as buyer's option

buf (char *) buffer for datablock. Memory will be allocated if this parameter is equal to null.

size (unsigned int *) pointer to integer in which the size of the received datablock is saved.

maxsize
: (unsigned int *) describes size of buf. This parameter will be ignored if buf is equal to null.

term
: (int) 0: do not terminate the buffer, 1: terminate the buffer by appending a 0x00.

Returns:

(char *) pointer to buffer which contains the received datablock; NULL if fail; 1 if no data available.

int block_send (int fd, unsigned int type, char * buf, unsigned int size)

Send a block (composition of: type, size of datablock and datablock (buf)) to a socket.

The function blocks until the whole block is transfered to the buffer. If the buffer is full, data has to be sent first. WARNING: The integers (type and size; excluding fd) are only 16 bit values (0 - 65535).

Parameters:

fd
: (int) descriptor of the socket to which buf should send

type
: (unsigned int) This value can be used as buyer's option

buf
: (char *) which should be send

Returns:

number of sent bytes, -1 if an error is occurt.

void free_authinfo (struct AUTHINFO * destroy)

Free the memory space which is used by an AUTHINFO structure.

Parameters:

struct AUTHINFO *) pointer to structure to destroy.

struct AUTHINFO* getauthinfo (char * netname, char * name)

Load authentication informations (netname, name, passwd, keyencrypt, keydecrypt) from authfile.

Parameters:

<u>netname</u>
: (char *) specify the network name (may IP). NULL not specified.

<u>name</u>
: (char *) specity the login name. NULL not specified.

Returns:
: (struct AUTHINFO *) the first entry from authfile which matches network name OR login name. If both values are NULL, the first entry of the authfile is given back.

int linemonitor (char * server, int port, int soft_msec, int hard_msec, int wait_msec, void(* linemonitor_exception)(char *server, int port, int type))

Monitor if the 'line' is fast enough: Client/Robot Application.

This function opens a socket stream, sents pings/bytes and wait for them to come back. The soft-timeout will called after soft_msec is timeouted. The hard-timeout will called after soft-timeout was called AND hard_msec is timeouted. wait_msec specifies the time which is waited after a ping is received befor the next one will be launched.

Parameters:

<u>server</u> (char *) server to be connected

<u>port</u> (int) port to be connected

<u>soft_msec</u>
: (int) timeout in milliseconds which causes soft-real-time exception.

<u>hard_msec</u>
: (int) timeout in milliseconds which causes hard-real-time exception.

<u>wait_msec</u>
: (int) timeout for resent -- sending of the next

ping.

linemonitor_exception
: (pointer to function) This function will be called if an exception occurs. It becomes the following parameters: server name (char *) which is always null, port (int): listend port and type (int) of exception which can be: 0: Connicion Fault, 1: Soft Real Time Exception, 2: HARD Real Time Exception, 3: Transmission Fault, 4: Emergency Stop.

void linemonitor_emergencystop (int sock)

Sends an 'Emergency Stop' to the client's side, linemonitor() will produce an 'Emergency Stop' exception (type 4).

int linemonitor_server (int port, int soft_msec, int hard_msec, int wait_msec, void(* linemonitor_exception)(char *server, int port, int type))

Monitor if the 'line' is fast enough: Server Application.

This function opens a port and wait for the first connection on this port. All data/pings which is sent by this first connection will be sent back. The soft-timeout will called after wait_msec AND soft_msec is timeouted. The hard-timeout will called after soft-timeout was called AND hard_msec is timeouted.

Parameters:

port
: (int) port which should be listend

soft_msec
: (int) timeout in milliseconds which causes soft-real-time exception.

hard_msec
: (int) timeout in milliseconds which causes hard-real-time exception.

wait_msec
: (int) timeout for resent -- sending of the next ping.

linemonitor_exception
: (pointer to function) This function will be called if an exception occurs. It becomes the following parameters: server name (char *) which is always null, port (int): listend port and type (int) of exception which can be: 0: Connicion Fault, 1: Soft Real Time Exception, 2: HARD Real Time Exception.

Returns:
: (int) Filediscriptor to the used socket. Only for usage with linemonitor_emergencystop().

void linemonitor_server_thread (struct LINEMONITOR_THREAD_DATA * linemonitor_thread_data)

Thread used by linemonitor_server() NOT for direct usage.

int linemonitor_thread (struct LINEMONITOR_THREAD_DATA * linemonitor_thread_data)

Thread used by linemonitor() NOT for direct usage.

int socket_accept (int sockport, int id, void(* socket_accept_do)(int fd, int id, char *pip, struct sockaddr_in their_addr))

Start a new thread, wait for connections and start socket_accept_do() when someone connects.

Parameters:

sockport
: (int) descriptor of a tcp socket/port from socket_bind()

id
: (int) arbitrary id of background process / thread. (May be it is a good idea to use the portnumber.)

aocket_accept_do
: (int fd, int id, char *pip, struct sockaddr_in their_addr) (function) this function is called if somebody connects. fd is the descriptor of the new socket to the connected tcp-tream. id is the same as in socket_accept(). pip contains the ip-address

of the connected client. The structure their_addr contails all known information about the connected client.

Returns:

If all right zero otherwise non zero.

void socket_accept_thread (struct LIBCOMMPTHREADP * libcommpthreadp)

This is a part of socket_accept() and must not called from the user.

This function is the thread which is started from socket_accept() and runs in background.

int socket_bind (int port, int cqueue)

Bind a socket to a port (Server side).

This function creates a socket and binds it to a local port.

Parameters:

port an integer which specifies the port

cqueue an integer how many pending connections queue will hold in the waiting queue.

Returns:

The File Descriptor (FD) which allows access to the bound port.

int socket_connect (char * host, int port)

Connect a TCP-stream to a server (Client side).

Creates a socket and connect it over a TCP-stream to the specified port on the specified server.

Parameters:

host a string (char *) which specifies the name or the IP-address of the server.

port an integer which specifies the port on the server.

Returns:

```
The File Descriptor (FD) which allows access to the
TCP-stream-socket or -1 if the connection fails.
```

int socket_md5auth (int fd, char * netname, char * name, struct AUTHINFO ** plocallogin, struct AUTHINFO ** premotelogin)

```
Do both side authentification.

This function is usually called just after a socket stream
is established. The function must be called on both sides.

WARNING: This authentication can be bypassed simply by
using the multiple session attack if multiple session are
allowd and the same password is used for both sides.

Both sides following these steps:

1. get auth info ([login] name, passwd) by using
getauthinfo() from name or netname for remote login

2. generate random numbers

3. exchange (first send, then receive) login names

4. exchange random numbers

5. calculate md5 checksum over the random numbers
(received from other side) and the remote passwd.

6. exchange md5 checksums

7. get auth info from name (received from other side) for
local login

8. calculate md5 checksum over the local random numbers
and the local passwd.

9. check login -- compare the received md5sum (6.) with
the generated one (8.); send acknowledgement

10. receive remote acknowledgement

11. return suitable values
```

Parameters:

fd (int) describes the socket on which the authentication has to be done

netname
(char *) use netname to resolve [login] name and passwd of the remote machine (NULL: not specified)

netname
(char *) use [login] name to resolve passwd of the remote machine (NULL: not specified; both NULL use first entry in file, see getauthinfo())

plocallogin
(struct AUTHINFO **) (pointer to pointer to an AUTHINFO struct) in this (double pointed) struct the local authinfo will be loaded, if the parameter is not null.

premotelogin
(struct AUTHINFO **) in this (double pointed) struct the remote authinfo will be loaded, if the parameter is not null.

Returns:
(int) 0: Authentication/Login OK; -1: remote login error; -2: login error on both sides; -3: local login error; -4: other (network) error; -5: cannot load remote auth info; -6: cannot load local auth info;

void thread1 (struct LIBCOMMPTHREADS * libcommpthreads)
This is a part of block_call() and must not called from the user.

This function is the thread which is started from block_call() and runs in background.

Parameters:

libcommpthreads
(struct LIBCOMMPTHREADS *) holds pointers to the functions to be call, fd (socket discriptor) and

```
       id.
```

AUTHOR

```
       Generated automatically by Doxygen for
       Hofmeier_FYP:libcomm from the source code.



Hofmeier_FYP:libcomm       25 Apr 2005              libcomm.c(3)
```

Appendix C: Source Code of the Network Library

C.1 src/lib/libcomm.h

```
/**
   @file

   Definitions for libcomm.
*/

/*
  Copyright (c) Andreas Hofmeier
  (www.an-h.de, www.an-h.de.vu, www.lgut.uni-bremen.de/an-h/)

  This program is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 2 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful, but
  WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
  General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program; if not, write to the Free Software
  Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*/



//#include "md5.h"
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define true 1
#define false 0

```

```
#ifndef nothread
  #include <pthread.h>
#endif

// declaration

// md5auth
// the first authfile, which is fould will be used
// first authfile.
#define authfile0 "./libcomm_md5auth.pwd"
// second authfile
#define authfile1 "/etc/libcomm_md5auth.pwd"
// one char/byte which seperates the field in the authfile
#define authfilefieldseperator ':'
// maxinam lenght of a line in the authfile
#define authfilemaxlinelenght 4096
// how much random bytes are generated for the authentication
#define authrandomstringsize (int) 16
// Define the message-type for the auth blocks
#define authmessagetype 65535

// structure to stroe the authentication infromationen
struct AUTHINFO {
  // network name
  char *netname;
  // login name
  char *name;
  // login passwd
  char *passwd;
  // Key for encryption (not used yet)
  char *keyencrypt;
  // Key for decryption (not used yet)
  char *keydecrypt;
};

int socket_md5auth(int fd, char *netname, char *name,
                   struct AUTHINFO **locallogin,
                   struct AUTHINFO **remotelogin);
struct AUTHINFO *getauthinfo(char *netname, char *name);



// socket_acceept
#ifndef nothread
struct LIBCOMMPTHREADP {
  void (*socket_accept_do)(int fd, int id, char *pip,
                           struct sockaddr_in their_addr);

  pthread_t        thrd_2;
  pthread_attr_t   thrd_2_attr;
  int              sockport;
  int              id;
};
void socket_accept_thread(struct LIBCOMMPTHREADP *libcommpthreadp);
int socket_accept(int sockport, int id,
           void (*socket_accept_do)(int fd, int id, char *pip,
                           struct sockaddr_in their_addr));
#endif


// block_receive
#ifndef nothread
struct LIBCOMMPTHREADS {
  void (*block_call_do)(int fd, int id, unsigned int type,
                        char *buf, unsigned int size, int term);
```

```
  void (*block_call_term)(int fd, int id);

  pthread_t        thrd_1;
  pthread_attr_t   thrd_1_attr;
  int              fd;
  int              id;
  int              term;
};
void thread1(struct LIBCOMMPTHREADS *libcommpthreads);
int block_call(int fd, int id, int term,
           void (*block_call_do)(int fd, int id, unsigned int type,
                                 char *buf, unsigned int size,
                                 int term),
           void (*block_call_term)(int fd, int id));
#endif
char *block_receive_poll(int fd, unsigned int *type, char *buf,
                         unsigned int *size, unsigned int maxsize,
                         int term);
char *block_receive(int fd, unsigned int *type, char *buf,
                    unsigned int *size, unsigned int maxsize,
                    int term);
int block_receive_integer(int fd, unsigned int *recvi);
int block_receive_nbytes(int fd, char *buf, int n);



// block_send
int block_send(int fd, unsigned int type, char *buf, unsigned int size);



// block_random
char *block_random(char *buf, int size);



// socket_bind
int socket_bind(int port, int cqueue);



// socket_connect
int socket_connect(char *host, int port);



// line monitor
struct LINEMONITOR_THREAD_DATA {
  char *server;
  int port;
  int soft_msec;
  int hard_msec;
  int wait_msec;
  void (*linemonitor_exception)(char *server, int port, int type);
  int sock;
};


void linemonitor_server_thread(struct LINEMONITOR_THREAD_DATA
                         *linemonitor_thread_data);
int linemonitor_server(int port,
               int soft_msec, int hard_msec, int wait_msec,
               void (*linemonitor_exception)(char *server, int port,
                                             int type));
void linemonitor_emergencystop(int sock);
```

```
int linemonitor_thread(struct LINEMONITOR_THREAD_DATA
                        *linemonitor_thread_data);
int linemonitor(char *server, int port,
                int soft_msec, int hard_msec, int wait_msec,
                void (*linemonitor_exception)(char *server, int port,
                                              int type));


```

C.2 src/lib/libcomm.c

```
/**
   @file

   Main part of libcomm.
*/

/*
  Copyright (c) Andreas Hofmeier
  (www.an-h.de, www.an-h.de.vu, www.lgut.uni-bremen.de/an-h/)

  This program is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 2 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful, but
  WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
  General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program; if not, write to the Free Software
  Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*/


#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <unistd.h>
#include <sys/poll.h>
#include <string.h>


#include "libcomm.h"
#include "md5.h"

#ifndef nothread
  #include <pthread.h>
#endif


#ifndef nothread
/**
   This is a part of socket_accept() and must not called from the
   user. This function is the thread which is started from
   socket_accept() and runs in background.
*/
void socket_accept_thread(struct LIBCOMMPTHREADP *libcommpthreadp) {
```

```
  char *buf;
  unsigned int type;
  unsigned int size;
  /* connector's address information */
  struct sockaddr_in their_addr;
  int sin_size;
  int fd;

  while (1) {
    // wait and accept a incomming connection
    sin_size = sizeof(struct sockaddr_in);
    if ((fd = accept(libcommpthreadp -> sockport,
                     (struct sockaddr *) &their_addr,
                     &sin_size)) != -1) {
      char *pip = inet_ntoa(their_addr.sin_addr);

      libcommpthreadp -> socket_accept_do(fd,
                                          libcommpthreadp -> id,
                                          pip, their_addr);
    }
  }
  //  pthread_exit(NULL);
}


/**
   Start a new thread, wait for connections and start
   socket_accept_do() when someone connects.

   @param sockport (int) descriptor of a tcp socket/port from
   socket_bind()

   @param id (int) arbitrary id of background process / thread. (May
   be it is a good idea to use the portnumber.)

   @param aocket_accept_do(int fd, int id, char *pip, struct
   sockaddr_in their_addr) (function) this function is called if
   somebody connects.  fd is the descriptor of the new socket to the
   connected tcp-tream. id is the same as in socket_accept(). pip
   contains the ip-address of the connected client. The structure
   their_addr contails all known information about the connected
   client.

   @return If all right zero otherwise non zero.
*/
int socket_accept(int sockport, int id,
           void (*socket_accept_do)(int fd, int id, char *pip,
                                    struct sockaddr_in their_addr)) {

  // allocate memory for thread configuration
  struct LIBCOMMPTHREADP *libcommpthreadp;
  libcommpthreadp = (struct LIBCOMMPTHREADP *)
    malloc(sizeof(struct LIBCOMMPTHREADP));
  if (libcommpthreadp == NULL) {
    perror("malloc()");
    return -1;
  }

  // store all necessary data in it
  libcommpthreadp -> sockport = sockport;
  libcommpthreadp -> id = id;
  libcommpthreadp -> socket_accept_do = socket_accept_do;

  // starting thread
  pthread_attr_init(&(libcommpthreadp -> thrd_2_attr));
```

```
  return pthread_create(&(libcommpthreadp -> thrd_2),
                        &(libcommpthreadp -> thrd_2_attr),
                        (void *) socket_accept_thread,
                        libcommpthreadp);

}

#endif


/** Bind a socket to a port (Server side). This function creates a
    socket and binds it to a local port.

   @param port an integer which specifies the port

   @param cqueue an integer how many pending connections queue will
   hold in the waiting queue.

   @return The File Descriptor (FD) which allows access to the bound
   port.
*/

#include <netinet/in.h>

int socket_bind(int port, int cqueue) {
  // FD of the new socket to the bound port
  int sock;
  // address information
  struct sockaddr_in ad;

  // create a socket
  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    // cannot created socket, return error
    perror("socket");
    return -1;
  }

  // make ensure that the memory is initiated
  memset(&ad, 0, sizeof(ad));

  // address family: AF_INET: IPv4 Internet protocols
  ad.sin_family = AF_INET;
  // convert and copy port in structure
  ad.sin_port = htons(port);
  // bind to all interfaces -- the port will accept connections to all
  // addresses of the local machine
  ad.sin_addr.s_addr = INADDR_ANY;
  // bind socket
  if (bind(sock, (struct sockaddr *) &ad, sizeof(struct sockaddr)) == -1) {
    // cannot bind, return error
    perror("bind");
    return -1;
  }
  // listen for connections on bound port
  if (listen(sock, cqueue) == -1) {
    // cannot listen, return error
    perror("listen");
    return -1;
  }
  // all right, port is listening. Return the FD as
  // reference for use.
  return sock;
}
```

```
#include <netinet/in.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>



/**
  Connect a TCP-stream to a server (Client side). Creates a socket and
  connect it over a TCP-stream to the specified port on the specified
  server.

  @param host a string (char *) which specifies the name or the
  IP-address of the server.

  @param port an integer which specifies the port on the server.

  @return The File Descriptor (FD) which allows access to the
  TCP-stream-socket or -1 if the connection fails.
*/
int socket_connect(char *host, int port) {
  // FD of the new socket to the TCP-stream
  int sock;
  // The IP address in binary form
  in_addr_t inaddr;
  // address information to connect other side (syscall: connect() )
  struct sockaddr_in ad;
  // contains the result of the resolution of a network-name.
  struct hostent *hp;

  // make ensure that the memory is initiated
  memset(&ad, 0, sizeof(ad));
  // address family: AF_INET: IPv4 Internet protocols
  ad.sin_family = AF_INET;
  // Try to convert the given IP-address into binary data...
  inaddr = inet_addr(host);
  if (inaddr != INADDR_NONE) {
    // if the IP address was converted copy it in the parameter
    // structure (ad) for later use
    memcpy(&ad.sin_addr, &inaddr, sizeof(inaddr));
  } else {
    // if this is not possible (the name and not the IP address is
    // given), try to resolve the name to a binary IP address
    hp = gethostbyname(host);
    if (hp == NULL) {
      // name cannot resolved, return error
      perror("gethostbyname()");
      return -1;
    }
    // copy address in the parameter structure (ad) for later use
    memcpy(&ad.sin_addr, hp->h_addr, hp->h_length);
  }
  // convert and copy port-number in the parameter structure (ad) for
  // later use
  ad.sin_port = htons(port);
  // create a socket
  sock = socket(AF_INET, SOCK_STREAM, 0);
  if (sock < 0) {
    // cannot created socket, return error
    perror("socket()");
    return -1;
  }
  // connect the socket over an TCP-stream to the port and the server,
  // which are stored in ad.
```

```c
  if (connect(sock, (struct sockaddr *) &ad, sizeof(ad)) < 0) {
    // connection is not possible, return error
    perror("connect()");
    return -1;
  }
  // all right, socket is connected an can be used. Return the FD as
  // reference for use.
  return sock;
}


/**
   Get random numbers/bytes. This function reads random numbers/bytes
   from /dev/urandom and stores this bytes in a buffer.

   @param buf (char *) in which the bytes will be stored. If this
   parameter is equal to NULL dynamic memory will be allocated.

   @param size an integer, specifies ths size of the buffer (the
   number of the random bytes). WARNING: If buf is not equal to null,
   n*(size) bytes will be stored in this buffer without any check of
   ths size of this buf.

   @return (char *) a pointer to the buffer in which the random bytes
   are stored.
*/
char *block_random(char *buf, int size) {
  FILE *f;

  // If no momory allocated, allocate memory
  if (buf == NULL) {
    if ((buf = malloc(size)) == NULL) {
      perror("malloc");
      return NULL;
    }
  }

  // Read Random numbers from /dev/urandom and stroe this these in the
  // buffer

  if ((f = fopen("/dev/urandom", "ro")) == NULL) {
    perror("fopen(/dev/urandom)");
    return NULL;
  }

  fread(buf, 1, size, f);

  fclose(f);

  return buf;
}






#ifndef nothread
/**
   This is a part of block_call() and must not called from the
   user. This function is the thread which is started from
   block_call() and runs in background.

   @param libcommpthreads (struct LIBCOMMPTHREADS *) holds pointers to
   the functions to be call, fd (socket discriptor) and id.
```

```
*/
void thread1(struct LIBCOMMPTHREADS *libcommpthreads) {
  char *buf;
  unsigned int type;
  unsigned int size;

  while (1) {
    // try to receive a datablock...
    buf = block_receive(libcommpthreads->fd, &type, NULL, &size, 0,
                        libcommpthreads->term);
    // failed: call block_call_term() and terminate thread
    if (buf == NULL) {
      libcommpthreads->block_call_term(libcommpthreads->fd,
                                       libcommpthreads->id);
      break;
    }
    // datablock OK: call block_call_do(), after this wait for the
    // next datablock
    libcommpthreads->block_call_do(libcommpthreads->fd,
                                   libcommpthreads->id,
                                   type, buf, size,
                                   libcommpthreads->term);
  }
  pthread_exit(NULL);
}


/**
   Waits in a new thread for a datablock to be received and calls the
   function block_call_do() if this event occurs or block_call_term()
   when the connection terminates.

   @param fd (int) descriptor of socket

   @param id (int) arbitrary id of background process / thread

   @param term (int) 0: do not terminate the buffer, 1: terminate the
   buffer by appending a 0x00.

   @param block_call_do(int fd, int id, unsigned int type, char *buf,
   unsigned int size, int term) (function) this function is called if
   a datablock was received. fd, id and term are the same as in
   block_call(). type describes the type of the received datablock,
   buf is a pointer to this datablock and size is the number of bytes
   of the datablock

   @param block_call_term(int fd, int id) (function) this function is
   called if the connection terminates. fd and id are the same as in
   block_call().

   @return If all right zero otherwise non zero.
*/
int block_call(int fd, int id, int term,
           void (*block_call_do)(int fd, int id, unsigned int type,
                                 char *buf, unsigned int size,
                                 int term),
           void (*block_call_term)(int fd, int id)) {

  // allocate memory for thread configuration
  struct LIBCOMMPTHREADS *libcommpthreads;
  libcommpthreads = (struct LIBCOMMPTHREADS *)
    malloc(sizeof(struct LIBCOMMPTHREADS));
  if (libcommpthreads == NULL) {
    perror("malloc()");
    return -1;
```

```
  }

  // store all necessary data in it
  libcommpthreads -> fd = fd;
  libcommpthreads -> id = id;
  libcommpthreads -> block_call_do = block_call_do;
  libcommpthreads -> block_call_term = block_call_term;

  // starting thread
  pthread_attr_init(&(libcommpthreads -> thrd_1_attr));
  return pthread_create(&(libcommpthreads -> thrd_1),
                        &(libcommpthreads -> thrd_1_attr),
                        (void *) thread1, libcommpthreads);
}

#endif



/**
   This function tests if new data is available to read on a stream.

   @param fd (int) discriptor of stream to test

   @return (int) 1: Data to read; 0: No data to read
*/
int block_ifdata(int fd) {
  struct pollfd polld;

  polld.fd = fd;
  polld.events = POLLIN | POLLPRI;

  if (poll(&polld, 1, 0)) {
    return 1;
  }
  return 0;
}


/**
   Test if is there data available on the socket's input buffer and
   starts receiving a block if there is. WARNING: The integers (type
   and size; excluding fd) are only 16 bit values (0 - 65535).

   @param fd (int) descriptor of socket

   @param type (unsigned int *) pointer to integer, this value can be
   used as buyer's option

   @param buf (char *) buffer for datablock. Memory will be allocated
   if this parameter is equal to null.

   @param size (unsigned int *) pointer to integer in which the size
   of the received datablock is saved.

   @param maxsize (unsigned int *) describes size of buf. This
   parameter will be ignored if buf is equal to null.

   @param term (int) 0: do not terminate the buffer, 1: terminate the
   buffer by appending a 0x00.

   @return (char *) pointer to buffer which contains the received
   datablock; NULL if fail; 1 if no data available.

*/
```

```
char *block_receive_poll(int fd, unsigned int *type, char *buf,
                         unsigned int *size, unsigned int maxsize,
                         int term) {
  // new data available
  if (block_ifdata(fd)) {
    return block_receive(fd, type, buf, size, maxsize, term);
  } else {
  // no new data available
    return (char *) 1L;
  }
}



/**
   Receive a block (composition of: type, size of datablock and
   datablock) from a socket. Waits for a block to be received
   completely. WARNING: The integers (type and size; excluding fd) are
   only 16 bit values (0 - 65535).

   @param fd (int) descriptor of socket

   @param type (unsigned int *) pointer to integer, this value can be
   used as buyer's option

   @param buf (char *) buffer for datablock. Memory will be allocated
   if this parameter is equal to null.

   @param size (unsigned int *) pointer to integer in which the size
   of the received datablock is saved.

   @param maxsize (unsigned int *) describes size of buf. This
   parameter will be ignored if buf is equal to null.

   @param term (int) 0: do not terminate the buffer, 1: terminate the
   buffer by appending a 0x00.

   @return (char *) pointer to buffer which contains the received
   datablock; NULL if fail.

*/
char *block_receive(int fd, unsigned int *type, char *buf,
                    unsigned int *size, unsigned int maxsize,
                    int term) {
  // do not trust any user!
  if (term > 1) {
    term = 1;
  }
  if (term < 0) {
    term = 0;
  }

  // receiving type
  if (block_receive_integer(fd, type) < 0) {
    return NULL;
  }
  // receiving size
  if (block_receive_integer(fd, size) < 0) {
    return NULL;
  }

  if (buf == NULL) {
    if ((buf = (char *) malloc(*size + term)) == NULL) {
      perror("malloc()");
      return NULL;
```

```
    }
  } else {
    if ((*size + term) > maxsize) {
      fprintf(stderr, "Try to receive more than fit in the buffer\n");
      return NULL;
    }
  }
  // receiving data
  if (block_receive_nbytes(fd, buf, *size) < 0) {
    return NULL;
  }

  if (term) {
    buf[*size] = 0;
  }

  return buf;
}




/**
   Receive an integer (two bytes; 16 Bit) from the socket.

   @param fd (int) descriptor of socket

   @param recvi (unsigned int *) pointer to integer in which the
   received integer is saved.

   @return (int) 2: OK; -1: fail
*/
int block_receive_integer(int fd, unsigned int *recvi) {
  int i, r;
  //  unsigned int recvi;
  int sizeofint = 2; /* sizeof(int); */

  // reset value
  *recvi = 0;

  // receive value
  if (recv(fd, ((char *) recvi), sizeofint, MSG_WAITALL) != sizeofint) {
    perror("recv()");
    return -1;
  }

  return 2; //recvi;
}




/**
   Receive n bytes from socket.

   @param fd (integer) descriptor of socket

   @param buf (char *) buffer for saving the received bytes

   @param n (integer) number of bytes to receive

   @return (integer) n: OK; -1 fial
*/
int block_receive_nbytes(int fd, char *buf, int n) {
  int i, r;
```

```
  unsigned int recvi;
  int sizeofint = 2; /* sizeof(int); */

  // receive n bytes to buffer
  if (recv(fd, buf, n, MSG_WAITALL) != n) {
    perror("recv()");
    return -1;
  }

  return n;
}



/**
   Send a block (composition of: type, size of datablock and datablock
   (buf)) to a socket. The function blocks until the whole block is
   transfered to the buffer. If the buffer is full, data has to be
   sent first. WARNING: The integers (type and size; excluding fd) are
   only 16 bit values (0 - 65535).

   @param fd (int) descriptor of the socket to which buf should send

   @param type (unsigned int) This value can be used as buyer's option

   @param buf (char *) which should be send

   @return number of sent bytes, -1 if an error is occurt.
*/

int block_send(int fd, unsigned int type, char *buf,
               unsigned int size) {
  // add up the number of sent byte, for checking.
  int i, r;

  // send the type of the data
  i = r = 0;
  while (r < 2) {
    if ((i = send(fd, (void *) &type + r, 2 - r, 0)) < 0) {
      return -1;
    }
    r += i;
  }
  // send the size of the buffer
  i = r = 0;
  while (r < 2) {
    if ((i = send(fd, (void *) &size + r, 2 - r, 0)) < 0) {
      return -1;
    }
    r += i;
  }
  // send the data in the buffer it self
  i = r = 0;
  while (r < size) {
    if ((i = send(fd, (void *) buf + r, size - r, 0)) < 0) {
      return -1;
    }
    r += i;
  }

  /*
  // send the type of the data
  if ((r = send(fd, (void *) &type, 2, 0)) < 0) {
    return -1;
  }
```

```
  // send the size of the buffer
  if ((i = send(fd, (void *) &size, 2, 0)) < 0) {
    perror("send0()");
    return -1;
  }
  r += i;
  // send the data in the buffer it self
  if ((i = send(fd, buf, size, 0)) < 0) {
    perror("send1()");
    return -1;
  }
  r += i;

  // not the comlete messages was sent.
  if (r = (size + 4)) {
    perror("send2()");
    return -1;
  }
  */

  return r;
}






/**
   Free the memory space which is used by an AUTHINFO structure.

   @param (struct AUTHINFO *) pointer to structure to destroy.
*/
void free_authinfo(struct AUTHINFO *destroy) {
  free(destroy -> netname);
  free(destroy -> name);
  free(destroy -> passwd);
  free(destroy -> keyencrypt);
  free(destroy -> keydecrypt);
  free(destroy);
}



/**
   Do both side authentification. This function is usually called
   just after a socket stream is established. The function must be
   called on both sides.

   WARNING: This authentication can be bypassed simply by using the
   multiple session attack if multiple session are allowd and the same
   password is used for both sides.

   Both sides following these steps:

   1. get auth info ([login] name, passwd) by using getauthinfo() from
   name or netname for remote login

   2. generate random numbers

   3. exchange (first send, then receive) login names

   4. exchange random numbers

   5. calculate md5 checksum over the random numbers (received from other
```

```
    side) and the remote passwd.

    6. exchange md5 checksums

    7. get auth info from name (received from other side) for local login

    8. calculate md5 checksum over the local random numbers and the
    local passwd.

    9. check login -- compare the received md5sum (6.) with the
    generated one (8.); send acknowledgement

    10. receive remote acknowledgement

    11. return suitable values

    @param fd (int) describes the socket on which the authentication
    has to be done

    @param netname (char *) use netname to resolve [login] name and
    passwd of the remote machine (NULL: not specified)

    @param netname (char *) use [login] name to resolve passwd of the
    remote machine (NULL: not specified; both NULL use first entry in
    file, see getauthinfo())

    @param plocallogin (struct AUTHINFO **) (pointer to pointer to an
    AUTHINFO struct) in this (double pointed) struct the local authinfo
    will be loaded, if the parameter is not null.

    @param premotelogin (struct AUTHINFO **) in this (double pointed)
    struct the remote authinfo will be loaded, if the parameter is not
    null.

    @return (int) 0: Authentication/Login OK; -1: remote login error;
    -2: login error on both sides; -3: local login error; -4: other
    (network) error; -5: cannot load remote auth info; -6: cannot load
    local auth info;
*/
int socket_md5auth(int fd, char *netname, char *name,
                   struct AUTHINFO **plocallogin,
                   struct AUTHINFO **premotelogin) {
  char rstr0[authrandomstringsize], rstr1[authrandomstringsize];
  char rstr0sum[35], rstr1sum[35];

  int otype;
  char *oname;
  int onamesize;
  char *randblock;
  char *orandblock;
  int orandblocksize;

  struct MD5Context context;
  unsigned char md5_prs[16];
  unsigned char omd5_prs[16];

  int login_ok = 0;

  struct AUTHINFO *locallogin;
  // 1.
  struct AUTHINFO *remotelogin = getauthinfo(netname, name);
  if (remotelogin == NULL) {
    return -5;
  }
  if (premotelogin != NULL) {
```

```
    *premotelogin = remotelogin;
  }

  // 2. generate random block for local login
  if ((randblock = block_random(NULL, authrandomstringsize))
      == NULL) {
    if (premotelogin == NULL) {
      free_authinfo(remotelogin);
    }
    return -4;
  }

  // 3. exchange login name
  if (block_send(fd, authmessagetype, remotelogin -> name,
                 strlen(remotelogin -> name)) <= 0) {
    if (premotelogin == NULL) {
      free_authinfo(remotelogin);
    }
    free(randblock);
    return -4;
  }
  oname = block_receive(fd, &otype, NULL, &onamesize, 0, true);
  if ((oname == NULL) ||
      (otype != authmessagetype)) {
    if (premotelogin == NULL) {
      free_authinfo(remotelogin);
    }
    free(randblock);
    return -4;
  }


  // 4. exchange random block
  if (block_send(fd, authmessagetype, randblock,
                 authrandomstringsize)
      <= 0) {
    if (premotelogin == NULL) {
      free_authinfo(remotelogin);
    }
    free(randblock);
    free(oname);
    return -4;
  }
  orandblock = block_receive(fd, &otype, NULL,
                             &orandblocksize, 0, false);
  if ((orandblock == NULL) ||
      (otype != authmessagetype)) {
    if (premotelogin == NULL) {
      free_authinfo(remotelogin);
    }
    free(randblock);
    free(oname);
    free(orandblock);
    return -4;
  }

  // 5. calculate md5 checksum over the random numbers (received from
  // other side) and the remote passwd.
  MD5Init(&context);
  MD5Update(&context, orandblock, orandblocksize);
  MD5Update(&context, remotelogin -> passwd,
            strlen(remotelogin -> passwd));
  MD5Final(md5_prs, &context);

  // 6. exchange md5 checkumms
```

```
  if (block_send(fd, authmessagetype, md5_prs, 16) <= 0) {
    if (premotelogin == NULL) {
      free_authinfo(remotelogin);
    }
    free(randblock);
    free(oname);
    free(orandblock);
    return -4;
  }
  if ((block_receive(fd, &otype, omd5_prs, &orandblocksize,
                      16, false)
      == NULL) ||
    (otype != authmessagetype) ||
    (orandblocksize != 16)) {
    if (premotelogin == NULL) {
      free_authinfo(remotelogin);
    }
    free(randblock);
    free(oname);
    free(orandblock);
    return -4;
  }
  usleep(1);

  // 7. get auth info from name (received from other side) for local login
  locallogin = getauthinfo(NULL, oname);
  if (locallogin == NULL) {
    if (premotelogin == NULL) {
      free_authinfo(remotelogin);
    }
    free(randblock);
    free(oname);
    free(orandblock);
    return -6;
  }
  if (plocallogin != NULL) {
    *plocallogin = locallogin;
  }

  // 8. calculate md5 checksum over the local random numbers and the
  // local passwd.
  MD5Init(&context);
  MD5Update(&context, randblock, authrandomstringsize);
  MD5Update(&context, locallogin -> passwd,
            strlen(locallogin -> passwd));
  MD5Final(md5_prs, &context);

  // 9. check login -- compare the received md5sum (6.) with the
  // generated one (8.); send acknowledgement
  if (memcmp(md5_prs, omd5_prs, 16) == 0) {
    login_ok = 1;
    if (block_send(fd, authmessagetype, "OK", 2) <= 0) {
      if (plocallogin == NULL) {
        free_authinfo(locallogin);
      }
      if (premotelogin == NULL) {
        free_authinfo(remotelogin);
      }
      free(randblock);
      free(oname);
      free(orandblock);
      return -4;
    }
  } else {
    if (block_send(fd, authmessagetype, "FAIL", 4) <= 0) {
```

```
      if (plocallogin == NULL) {
        free_authinfo(locallogin);
      }
      if (premotelogin == NULL) {
        free_authinfo(remotelogin);
      }
      free(randblock);
      free(oname);
      free(orandblock);
      return -4;
    }
  }

  // 10. receive remote acknowledgement
  free(orandblock);
  orandblock = block_receive(fd, &otype, NULL, &orandblocksize,
                             0, true);
  if ((orandblock == NULL) ||
      (otype != authmessagetype) ||
      (orandblocksize != 2) ||
      (strcmp(orandblock, "OK") != 0)) {
    if (login_ok) {
      if (plocallogin == NULL) {
        free_authinfo(locallogin);
      }
      if (premotelogin == NULL) {
        free_authinfo(remotelogin);
      }
      free(randblock);
      free(oname);
      return -1;
    } else {
      if (plocallogin == NULL) {
        free_authinfo(locallogin);
      }
      if (premotelogin == NULL) {
        free_authinfo(remotelogin);
      }
      free(randblock);
      free(oname);
      return -2;
    }
  }


  if (plocallogin == NULL) {
    free_authinfo(locallogin);
  }
  if (premotelogin == NULL) {
    free_authinfo(remotelogin);
  }
  free(randblock);
  free(oname);
  free(orandblock);

  if (!login_ok) {
    return -3;
  }

  // all right!
  return 0;
}
```

```
/**
   Load authentication informations (netname, name, passwd,
   keyencrypt, keydecrypt) from authfile.

   @param netname (char *) specify the network name (may IP). NULL not
   specified.

   @param name (char *) specity the login name. NULL not
   specified.

   @return (struct AUTHINFO *) the first entry from authfile which
   matches network name OR login name. If both values are NULL, the
   first entry of the authfile is given back.
*/
struct AUTHINFO *getauthinfo(char *netname, char *name) {
  // Descriptor for authfile
  FILE *f;
  // buffer for reading one line of the authfile
  char buf[authfilemaxlinelenght];
  // number of fields in the authfile
  int fields = 5;
  // pointer buffer for the five parts of the line
  char *bufsplit[fields];
  // char **bufsplit;
  // control variable, count variable for field
  int i, j;
  struct AUTHINFO *load;
  // temporary pointer
  char *s;

  //  bufsplit = (char **) malloc(sizeof(char *) * fields);

  // allocate memory for auth-structure
  load = (struct AUTHINFO *) malloc(sizeof(struct AUTHINFO));
  if (load == NULL) {
    perror("malloc(sizeof(struct AUTHINFO))");
    return NULL;
  }

  // open authfile
  if ((f = fopen(authfile0, "ro")) == NULL) {
    perror(authfile0);
    if ((f = fopen(authfile1, "ro")) == NULL) {
      perror(authfile0);
      free(load);
      return NULL;
    }
  }

  // read as long as ther is no more data
  while (!feof(f)) {
    // read one line
    fgets(buf, authfilemaxlinelenght - 1, f);

    // split the line into it five components
    j = 0;
    bufsplit[j++] = buf;
    //    load -> netname = buf;
    for (i = 0; i < authfilemaxlinelenght; i++) {
      if (buf[i] == authfilefieldseperator) {
        buf[i] = 0;
        if (j == fields) {
          break;
        }
        bufsplit[j++] = buf + i + 1;
```

```
      }
    }

    // if this the right entry? Compare with parameter.
    if (
        ((    name != NULL) && (strcmp(    name, bufsplit[1]) == 0))
        ||
        ((netname != NULL) && (strcmp(netname, bufsplit[0]) == 0))
        ||
        ((netname == NULL) && (name == NULL))
       ) {

      // allocate Memory
      for (i = 0; i < fields; i++){
        s = NULL;
        s = (char *) malloc(strlen(bufsplit[i]) + 1);
        if (s == NULL) {
          perror("malloc()");
          free(load);
          return NULL;
        }
        strcpy(s, bufsplit[i]);
        bufsplit[i] = s;
      }

      // store the pointers in the struct
      load -> netname =    bufsplit[0];
      load -> name =       bufsplit[1];
      load -> passwd =     bufsplit[2];
      load -> keyencrypt = bufsplit[3];
      load -> keydecrypt = bufsplit[4];

      // return the pointer to this struct
      return load;
    }
  }
  free(load);
  return NULL;
}




/**
   Thread used by linemonitor_server() NOT for direct usage.
*/
void linemonitor_server_thread(struct LINEMONITOR_THREAD_DATA
                               *linemonitor_thread_data) {
  unsigned char buf;

  // configuration of poll -- waiting for an event of the socket.
  struct pollfd polld;
  polld.fd = linemonitor_thread_data -> sock;
  polld.events = POLLIN | POLLPRI;

  while (1) {
    // Receive a Ping/Byte
    if (recv(linemonitor_thread_data -> sock, ((char *) &buf), 1,
             MSG_WAITALL) != 1) {
      linemonitor_thread_data -> linemonitor_exception(
                        linemonitor_thread_data -> server,
                        linemonitor_thread_data -> port, 0);
      break;
    }
    // And Send it Back
```

```
      if (send(linemonitor_thread_data -> sock, &buf, 1, 0) != 1) {
        linemonitor_thread_data -> linemonitor_exception(
                          linemonitor_thread_data -> server,
                          linemonitor_thread_data -> port, 0);
        break;
      }

      // Test, if next Ping is received within the reload-time plus
      // soft-timeout
      if (poll(&polld, 1, linemonitor_thread_data -> wait_msec)
          <= 0) {

        if (poll(&polld, 1, linemonitor_thread_data -> soft_msec)
            <= 0) {
          // If not, call exception-function
          linemonitor_thread_data -> linemonitor_exception(
                            linemonitor_thread_data -> server,
                            linemonitor_thread_data -> port, 1);

          // and test if the data is received within the hard-timeout
          if (poll(&polld, 1, linemonitor_thread_data -> hard_msec)
              <= 0) {
            // If not, call exception-function
            linemonitor_thread_data -> linemonitor_exception(
                              linemonitor_thread_data -> server,
                              linemonitor_thread_data -> port, 2);
          }
        }
      }
    }

    pthread_exit(NULL);
}



/**
   Monitor if the "line" is fast enough: Server Application. This
   function opens a port and wait for the first connection on this
   port. All data/pings which is sent by this first connection will
   be sent back. The soft-timeout will called after wait_msec AND
   soft_msec is timeouted. The hard-timeout will called after
   soft-timeout was called AND hard_msec is timeouted.

   @param port (int) port which should be listend

   @param soft_msec (int) timeout in milliseconds which causes
   soft-real-time exception.

   @param hard_msec (int) timeout in milliseconds which causes
   hard-real-time exception.

   @param wait_msec (int) timeout for resent -- sending of the next
   ping.

   @param linemonitor_exception (pointer to function) This function
   will be called if an exception occurs. It becomes the following
   parameters: server name (char *) which is always null, port (int):
   listend port and type (int) of exception which can be: 0: Connicion
   Fault, 1: Soft Real Time Exception, 2: HARD Real Time Exception.

   @return (int) Filediscriptor to the used socket. Only for usage
   with linemonitor_emergencystop().
*/
int linemonitor_server(int port,
```

```
                    int soft_msec, int hard_msec, int wait_msec,
                    void (*linemonitor_exception)(char *server, int port,
                                                  int type)) {
  int  sock;

  /* connector's address information */
  struct sockaddr_in their_addr;
  int sin_size;
  int fd;

  // ID and atributes for the threads
  pthread_t        thrd_2;
  pthread_attr_t   thrd_2_attr;

  // allocate memory to store parameter for the
  // linemonitor_server_thread() function.
  struct LINEMONITOR_THREAD_DATA *linemonitor_thread_data;
  linemonitor_thread_data = (struct LINEMONITOR_THREAD_DATA *)
    malloc(sizeof(struct LINEMONITOR_THREAD_DATA));
  if (linemonitor_thread_data == NULL) {
    perror("malloc()");
    return -1;
  }

  // store all necessary parameters in this memory
  linemonitor_thread_data -> server = NULL;
  linemonitor_thread_data -> port = port;
  linemonitor_thread_data -> soft_msec = soft_msec;
  linemonitor_thread_data -> hard_msec = hard_msec;
  linemonitor_thread_data -> wait_msec = wait_msec;
  linemonitor_thread_data -> linemonitor_exception =
    linemonitor_exception;

  // Bind a port
  sock = socket_bind(port, 10);

  // wait for the first connection
  // only accept the first connection
  sin_size = sizeof(struct sockaddr_in);
  if ((fd = accept(sock, (struct sockaddr *) &their_addr,
                   &sin_size)) != -1) {
    char *pip = inet_ntoa(their_addr.sin_addr);

    linemonitor_thread_data -> sock = fd;

    // starting linemonitor_server_thread()
    pthread_attr_init(&thrd_2_attr);
    if (pthread_create(&thrd_2,
                       &thrd_2_attr,
                       (void *) linemonitor_server_thread,
                       linemonitor_thread_data) != 0) {
      return -1;
    }

    return fd;
  }

  return -1;
}



/**
   Sends an "Emergency Stop" to the client's side, linemonitor() will
   produce an "Emergency Stop" exception (type 4).
```

```
*/
void linemonitor_emergencystop(int sock) {
  unsigned char data = 254;
  send(sock, &data, 1, 0);
}



/**
   Thread used by linemonitor() NOT for direct usage.
*/
int linemonitor_thread(struct LINEMONITOR_THREAD_DATA
                       *linemonitor_thread_data) {
  // buffer for sending a ping
  unsigned char counter;
  // buffer for receiving a ping
  unsigned char rcounter;

  // configuration of poll -- waiting for an event of the socket.
  struct pollfd polld;
  polld.fd = linemonitor_thread_data -> sock;
  polld.events = POLLIN | POLLPRI;


  while (1) {
    // increase counter, prevent "Emergency Stop"-Code 254
    counter++;
    if (counter == 254) {
      counter = 0;
    }

    // send a ping
    if (send(linemonitor_thread_data -> sock, &counter, 1, 0) != 1) {
      linemonitor_thread_data -> linemonitor_exception(
                              linemonitor_thread_data -> server,
                              linemonitor_thread_data -> port, 0);
      break;
    }

    // Test, if Ping returns within the soft-timeout time, if not
    // cause exception
    if (poll(&polld, 1, linemonitor_thread_data -> soft_msec) <= 0) {
      linemonitor_thread_data -> linemonitor_exception(
                              linemonitor_thread_data -> server,
                              linemonitor_thread_data -> port, 1);
      // Test, if Ping returns within the soft-timeout plus
      // hard-timeout time, if not cause exception
      if (poll(&polld, 1, linemonitor_thread_data -> hard_msec) <= 0) {
        linemonitor_thread_data -> linemonitor_exception(
                              linemonitor_thread_data -> server,
                              linemonitor_thread_data -> port, 2);
      }
    }

    // receive the ping
    if (recv(linemonitor_thread_data -> sock, ((char *) &rcounter), 1,
             MSG_WAITALL) != 1) {
      linemonitor_thread_data -> linemonitor_exception(
                              linemonitor_thread_data -> server,
                              linemonitor_thread_data -> port, 0);
      break;
    }

    // If "Emergency Stop" code was received, call "Emergency Stop"
    // exception and retry to receive a ping
```

```
      if (rcounter == 254) {
        linemonitor_thread_data -> linemonitor_exception(
                               linemonitor_thread_data -> server,
                               linemonitor_thread_data -> port, 4);

        if (recv(linemonitor_thread_data -> sock, ((char *) &rcounter),
                 1, MSG_WAITALL) != 1) {
          linemonitor_thread_data -> linemonitor_exception(
                               linemonitor_thread_data -> server,
                               linemonitor_thread_data -> port, 0);
          break;
        }
      }

      // Test on right transmission code and call "Transmission Fault"
      // exception if the data is currupted
      if (counter != rcounter) {
        linemonitor_thread_data -> linemonitor_exception(
                               linemonitor_thread_data -> server,
                               linemonitor_thread_data -> port, 3);
      }

      // Wait before sendin next ping
      if (poll(&polld, 1, linemonitor_thread_data -> wait_msec) <= 0) {
      }
    }

    pthread_exit(NULL);
}



/**
   Monitor if the "line" is fast enough: Client/Robot Application. This
   function opens a socket stream, sents pings/bytes and wait for them
   to come back. The soft-timeout will called after soft_msec is
   timeouted. The hard-timeout will called after soft-timeout was
   called AND hard_msec is timeouted. wait_msec specifies the time
   which is waited after a ping is received befor the next one will be
   launched.

   @param server (char *) server to be connected

   @param port (int) port to be connected

   @param soft_msec (int) timeout in milliseconds which causes
   soft-real-time exception.

   @param hard_msec (int) timeout in milliseconds which causes
   hard-real-time exception.

   @param wait_msec (int) timeout for resent -- sending of the next
   ping.

   @param linemonitor_exception (pointer to function) This function
   will be called if an exception occurs. It becomes the following
   parameters: server name (char *) which is always null, port (int):
   listend port and type (int) of exception which can be: 0: Connicion
   Fault, 1: Soft Real Time Exception, 2: HARD Real Time Exception, 3:
   Transmission Fault, 4: Emergency Stop.
*/
int linemonitor(char *server, int port,
                int soft_msec, int hard_msec, int wait_msec,
                void (*linemonitor_exception)(char *server, int port,
                                              int type)) {
```

```

    // ID and atributes for the threads
    pthread_t         thrd_2;
    pthread_attr_t    thrd_2_attr;

    // allocate memory to store parameter for the
    // linemonitor_thread() function.
    struct LINEMONITOR_THREAD_DATA *linemonitor_thread_data;
    linemonitor_thread_data = (struct LINEMONITOR_THREAD_DATA *)
      malloc(sizeof(struct LINEMONITOR_THREAD_DATA));
    if (linemonitor_thread_data == NULL) {
      perror("malloc()");
      return -1;
    }

    // connect to server
    linemonitor_thread_data -> sock = socket_connect(server, port);
    if (linemonitor_thread_data -> sock <= 0) {
      linemonitor_exception(server, port, 0);
      return -1;
    }

    // store all necessary parameters in this memory
    linemonitor_thread_data -> server = server;
    linemonitor_thread_data -> port = port;
    linemonitor_thread_data -> soft_msec = soft_msec;
    linemonitor_thread_data -> hard_msec = hard_msec;
    linemonitor_thread_data -> wait_msec = wait_msec;
    linemonitor_thread_data -> linemonitor_exception =
      linemonitor_exception;

    // launch linemonitor_thread()
    pthread_attr_init(&thrd_2_attr);
    return pthread_create(&thrd_2,
                          &thrd_2_attr,
                          (void *) linemonitor_thread,
                          linemonitor_thread_data);
}

```

Appendix D: API of the Interface

interface.c(3) interface.c(3)

NAME

interface.c - Implementation of an example interface to a simple robot with two independent axies.

SYNOPSIS

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <asm/io.h>
```

```
#include <time.h>
#include <sys/wait.h>
```

Defines

`#define` interface_ymax_time `20000`
: `Time it rakes to drive to robot form y_min to y_max.`

`#define` interface_xmax_time `20000`
: `Time it rakes to drive to robot form x_min to x_max.`

`#define` interface_ymax `100`
: `Maximal Y value (cosidered as 100 percent).`

`#define` interface_xmax `100`
: `Maximal X value (cosidered as 100 percent).`

`#define` interface_ioport `0x378`
: `IO Port to which lowest nible to robot is connected.`

Functions

`unsigned char` input `(int addr)`
: `Read a byte from an IO port.`

`unsigned char` output `(int addr, unsigned char out)`
: `Write a byte to an IO port.`

`void` msleep `(int msec)`
: `Wait a specified number of milliseconds.`

`int` getmax `(int a, int b)`
: `Get to highest number out of two input numbers.`

`int` getmin `(int a, int b)`
: `Get to lowest number out of two input numbers.`

`void` interface_drive `(int h, int v)`
: `Drive the robot in a specified direction.`

`void` interface_init `(int mode)`
: `Initialze the interface and drive the robot to the start position.`

`void` interface_driveto `(int x, int y)`
: `Drive to robot to absolute coordinates.`

`void` interface_stop `()`
: `Stop the interface, switch all off.`

Variables

`int` interface_x
: `current X possition of robot (global).`

`int` interface_y
: `current X possition of robot (global).`

`int` interface_mode
: `mode of interface: 0: normal, 1: simulation (do all`

except to drive the robot), 2: simulation with position-output 3: Blocked: Do nothing.

DETAILED DESCRIPTION

Implementation of an example interface to a simple robot with two independent axies.

The robot has four inputs whish are connected to the lower nible on IO port 'interface_ioport'. The bits are connected in this way (the signals a high-active):

Bit 0: Drive Up wires: switch to GND: yellow-green; +24V: gray-black

Bit 1: Dirve down wires: switch to GND: red-green; +24V: orange-black

Bit 2: Dirve right wires: switch to GND: green-red; +24V: yellow-black

Bit 3: Dirve left wires: switch to GND: white-red; +24V: red-blue

power wires: GND: blue; +24V: red

This interface assumes a linear dependency betwenn the coverence of distance and moving time. The 0,0 coordinates is left,bottom.

User functions are:

interface_init() - Initialze the interface and drive the robot to the start position (X=undefined; Y=0).

interface_driveto(int x, int y) - Drive the robot to the absolute coordinates x,y.

DEFINE DOCUMENTATION

define interface_ioport 0x378

IO Port to which lowest nible to robot is connected.

define interface_xmax 100

Maximal X value (cosidered as 100 percent).

define interface_xmax_time 20000

Time it rakes to drive to robot form x_min to x_max.

define interface_ymax 100

Maximal Y value (cosidered as 100 percent).

define interface_ymax_time 20000

Time it rakes to drive to robot form y_min to y_max.

FUNCTION DOCUMENTATION

int getmax (int a, int b)

Get to highest number out of two input numbers.

Parameters:

a (int) first input number

b (int) second input number

Returns:

(int) the highest of the input numbers

int getmin (int a, int b)

Get to lowest number out of two input numbers.

Parameters:

a (int) first input number

b (int) second input number

Returns:

(int) the lowest of the input numbers

unsigned char input (int addr)

Read a byte from an IO port.

Parameters:

addr (int): address of port to read

Returns:

`(unsigned char) read byte`

void interface_drive (int h, int v)

```
Drive the robot in a specified direction.

Any axies can be zero, greater than zero or less than
zero, in this cases the robot will not driven, driven to
increase to position (up[v,y] or right[h,x]) and decrease
the position (down[-v,-y] or left[-h,-x]).

Parameters:
```

<u>h</u> `(int) horizontal or X axias.`

<u>v</u> `(int) vertical or Y axias.`

void interface_driveto (int x, int y)

```
Drive to robot to absolute coordinates.

Parameters:
```

<u>x</u> `(int): absolute x coordinate`

<u>y</u> `(int): absolute y coordinate`

void interface_init (int mode)

```
Initialze the interface and drive the robot to the start
position.
```

void interface_stop ()

```
Stop the interface, switch all off.
```

void msleep (int msec)

```
Wait a specified number of milliseconds.

Parameters:
```

<u>msec</u> `(int): milliseconds to wait`

unsigned char output (int addr, unsigned char out)

```
Write a byte to an IO port.
```

Parameters:

addr (int): address of port to write onto

out (unsigned char): byte to write

Returns:
(unsigned char) written byte

VARIABLE DOCUMENTATION

int interface_mode

mode of interface: 0: normal, 1: simulation (do all except to drive the robot), 2: simulation with position-output 3: Blocked: Do nothing.

int interface_x

current X possition of robot (global).

int interface_y

current X possition of robot (global).

AUTHOR

Generated automatically by Doxygen for Hofmeier_FYP:libcomm from the source code.

Hofmeier_FYP:libcomm 25 Apr 2005 interface.c(3)

Appendix E: Source Code of the Interface

E.1 src/example/interface.c

```
/**
   @file

   Implementation of an example interface to a simple robot with two
   independent axies. The robot has four inputs whish are connected to
   the lower nible on IO port "interface_ioport". The bits are
```

```
   connected in this way (the signals a high-active):

   Bit 0: Drive Up
     wires: switch to GND: yellow-green;    +24V: gray-black

   Bit 1: Dirve down
     wires: switch to GND: red-green;       +24V: orange-black

   Bit 2: Dirve right
     wires: switch to GND: green-red;       +24V: yellow-black

   Bit 3: Dirve left
     wires: switch to GND: white-red;       +24V: red-blue

     power wires:     GND: blue;            +24V: red

   This interface assumes a linear dependency betwenn the coverence of
   distance and moving time. The 0,0 coordinates is left,bottom.

   User functions are:

   interface_init() - Initialze the interface and drive the robot to
   the start position (X=undefined; Y=0).

   interface_driveto(int x, int y) - Drive the robot to the absolute
   coordinates x,y.
*/

/*
  Copyright (c) Andreas Hofmeier
  (www.an-h.de, www.an-h.de.vu, www.lgut.uni-bremen.de/an-h/)

  This program is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 2 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful, but
  WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
  General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program; if not, write to the Free Software
  Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*/


#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <asm/io.h>
#include <time.h>
#include <sys/wait.h>


/** Time it rakes to drive to robot form y_min to y_max */
#define interface_ymax_time 20000 // at 6 bar
/** Time it rakes to drive to robot form x_min to x_max */
#define interface_xmax_time 20000 // 34670
/** Maximal Y value (cosidered as 100 percent) */
#define interface_ymax 100
/** Maximal X value (cosidered as 100 percent) */
#define interface_xmax 100
/** IO Port to which lowest nible to robot is connected */
```

```
#define interface_ioport 0x378

/** current X possition of robot (global) */
int interface_x;
/** current X possition of robot (global) */
int interface_y;

/** mode of interface:
    0: normal,
    1: simulation (do all except to drive the robot),
    2: simulation with position-output
    3: Blocked: Do nothing */
int interface_mode;


/**
   Read a byte from an IO port
   @param addr (int): address of port to read
   @return (unsigned char) read byte
*/
unsigned char input(int addr) {
  if (interface_mode == 0) {
    return inb(addr);
  }
  return 255;
}


/**
   Write a byte to an IO port
   @param addr (int): address of port to write onto
   @param out (unsigned char): byte to write
   @return (unsigned char) written byte
*/
unsigned char output(int addr, unsigned char out) {
  if (interface_mode == 0) {
    outb(out, addr);
    return out;
  }
  return out;
}


/**
   Wait a specified number of milliseconds
   @param msec (int): milliseconds to wait
*/
void msleep(int msec) {
  int sec = msec / 1000;
  msec = msec - sec * 1000;
  sleep(sec);
  usleep(msec * 1000);
}


/**
   Get to highest number out of two input numbers
   @param a (int) first input number
   @param b (int) second input number
   @return (int) the highest of the input numbers
 */
int getmax(int a, int b) {
  if (a > b) {
    return a;
  } else {
```

```
      return b;
    }
  }


  /**
     Get to lowest number out of two input numbers
     @param a (int) first input number
     @param b (int) second input number
     @return (int) the lowest of the input numbers
   */
  int getmin(int a, int b) {
    if (a < b) {
      return a;
    } else {
      return b;
    }
  }


  /**
     Drive the robot in a specified direction. Any axies can be zero,
     greater than zero or less than zero, in this cases the robot will
     not driven, driven to increase to position (up[v,y] or right[h,x])
     and decrease the position (down[-v,-y] or left[-h,-x]).

     @param h (int) horizontal or X axias.
     @param v (int) vertical or Y axias.
  */
  void interface_drive(int h, int v) {
    unsigned char buf;

    buf = 0;
    /* Y Axies */
    if (v > 0) {
      buf = buf | 1;
    }
    if (v < 0) {
      buf = buf | 2;
    }
    /* X Axies */
    if (h > 0) {
      buf = buf | 4;
    }
    if (h < 0) {
      buf = buf | 8;
    }

    /* Apply */
    buf = buf | (input(interface_ioport) & 240);
    output(interface_ioport, buf);
  }


  /**
     Initialze the interface and drive the robot to the start position.
   */
  void interface_init(int mode) {
    if ((mode < 0) || (mode > 3)) {
      mode = 2;
    }
    interface_mode = mode;

    // enable access to IO ports (need root previleges)
    if (interface_mode == 0) {
```

```
    if (iopl(3) < 0) {
      perror("iopl()");
      //      exit(1);
      interface_mode = 2;
    }
  }

  switch (interface_mode) {
  case 0:
    fprintf(stderr, "Interface full active.\n");
    break;
  case 1:
    fprintf(stderr, "Interface disabled.\n");
    break;
  case 2:
    fprintf(stderr, "Interface disabled: position output\n");
    break;
  }

  fprintf(stderr, "Initialze interface (moving to x=50,y=0)...\n");

  // drive to the coordinates ?, 0
  // x will not be driven, because it is too noisy and slowly
  interface_drive(0, -1);
  // code for "normal" drive-time for 0,0
  /*  if (interface_xmax_time > interface_ymax_time) {
    msleep(interface_xmax_time + interface_xmax_time/5);
  } else {
    msleep(interface_ymax_time + interface_ymax_time/5);
    } */
  // code for Y-drive-time only
  msleep(interface_ymax_time + interface_ymax_time/5);
  interface_drive(0, 0);

  // set coordinates to ?, 0
  interface_x = 50;
  interface_y = 0;
}


/**
   Drive to robot to absolute coordinates
   @param x (int): absolute x coordinate
   @param y (int): absolute y coordinate
*/
void interface_driveto(int x, int y) {
  int i;

  // absulute coordinates are in range?
  if (x > interface_xmax) {
    x = interface_xmax;
  }
  if (x < 0) {
    x = 0;
  }
  if (y > interface_ymax) {
    y = interface_ymax;
  }
  if (y < 0) {
    y = 0;
  }

  // calculate relative coordinates
  x = (x - interface_x);
  y = (y - interface_y);
```

```

  // store new coordinates
  interface_x += x;
  interface_y += y;

  //  if (interface_mode == 2) {
    fprintf(stderr, "x = %03d ; y = %03d\n",
            interface_x, interface_y);
    //  }

  // if any change has to be done...
  if ((x != 0) || (y != 0)) {
    // convert the relative distance into a time in which this
    // distance is covered
    x = (int) ((double)
               (((double) x * (double) interface_xmax_time)
                / (double) interface_xmax));
    y = (int) ((double)
               (((double) y * (double) interface_ymax_time)
                / (double) interface_ymax));


    // if the drive-time for both axies is not equal, drive the
    // greater time as long as the times are equal
    if ((x != 0) && (y != 0)) {
      interface_drive(x, y);
      msleep(getmin(abs(x), abs(y)));
      interface_drive(0, 0);

      i = getmin(abs(x), abs(y));
      x = x/abs(x) * (abs(x) - i);
      y = y/abs(y) * (abs(y) - i);
    }

    // if there is any time left, drive these
    if ((x != 0) || (y != 0)) {
      interface_drive(x, y);
      msleep(getmax(abs(x), abs(y)));
      interface_drive(0, 0);
    }
  }
}


/** Stop the interface, switch all off. */
void interface_stop() {
  fprintf(stderr, "interface_stop()\n");
  // block the interface
  interface_mode = 3;
  // switch all off.
  //  interface_drive(0, 0);

  //  interface_drive do not work, because "interface_mode = 3"
  //  switches to IO-access off. For this reason, the functions were
  //  directly used.
  outb((inb(interface_ioport) & 240), interface_ioport);
}
```

Appendix F: Source Code of the GUI

F.1 src/example/guicommon.c

```c
/** @file Function for drawing the robot and to calculate the absolute
    postion of the robot out of the mouse-position.

*/

/*
  Copyright (c) Andreas Hofmeier
  (www.an-h.de, www.an-h.de.vu, www.lgut.uni-bremen.de/an-h/)

  This program is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 2 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful, but
  WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
  General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program; if not, write to the Free Software
  Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*/


#include <time.h>
#include <math.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>
#include <glib.h>


/** Function for drawing the robot and to calculate the absolute
    postion of the robot out of the mouse-position.

    @param widget (GtkWidget *) pointer to affected object (drawing
    area).

    @param x (int *) pointer to absolute x position of the robot (will
    changed if mouseaction equal to 1 or 2)

    @param y (int *) pointer to absolute y position of the robot (will
    changed if mouseaction equal to 1 or 2)

    @param mx (int) x mouse position (only relevant if mouseaction
    equal to 1 or 2)

    @param my (int) y mouse position (only relevant if mouseaction
    equal to 1 or 2)

    @param mouseaction (int) current action: 1: start moving, 2:
    moving or finish moving, other: plot only

*/
void gui_common_drawrobot(GtkWidget *widget, int *x, int *y,
                          int mx, int my, int mouseaction) {
```

```
  // drawing area properties
  gint new_width, new_height;
  int height, width;

  // width and hieght of the illustration of the moving platform of
  // the robot.
  int hwidth, wwidth;

  // current position of the moving platform
  int x0, y0;

  // new position of the moving platform
  int x0n, y0n;

  // current action: 0: draw only, 1: during moving, 2: error: click
  // outside from platform
  static int moving = 0;

  // offset for mouse-moving
  static int ox, oy;

  // get size of the drawing area
  gdk_drawable_get_size(widget->window,
                        &new_width,
                        &new_height);
  width = (int) new_width;
  height = (int) new_height;

  // clear it
  gdk_window_clear_area(widget->window,
                        0,0,
                        width, height);

  // calculate the width and hieght of the illustration of the moving
  // platform of the robot.
  hwidth = 10 * height / 100;
  wwidth = 10 * width / 100;

  // moving or moving finished
  if (mouseaction == 2) {
    moving = 0;
  }
  if (moving == 1) {
    // calculate the new position of the platform and set the
    // variables
    *x = ((mx - ox) * 100) / (width - wwidth);
    *y = (((my + oy) * (-1) + (height - hwidth)) * 100) / (height - hwidth);

    // platform must not beyond the window bondaries
    if (*x > 100) {
      *x = 100;
    }
    if (*x < 0) {
      *x = 0;
    }
    if (*y > 100) {
      *y = 100;
    }
    if (*y < 0) {
      *y = 0;
    }
  }

  // draw the horizontal (X) axies track of the robot, whish is NOT
```

```
  // moving
  y0 = ((0 * (height - hwidth) / 100) - (height  - hwidth)) * (-1);
  gdk_draw_line(widget->window,
                widget->style->black_gc,
                0,y0,
                width,y0);
  gdk_draw_line(widget->window,
                widget->style->black_gc,
                0,y0 + hwidth,
                width,y0 + hwidth);

  // Draw the current possition of the movind platform of the
  // robot. Illustrated as a black rectangular.
  // +- calculate it's current position
  x0 = *x * (width - wwidth) / 100;
  y0 = ((*y * (height - hwidth) / 100) - (height  - hwidth)) * (-1);
  // +- draw lines under it.
  gdk_draw_line(widget->window,
                widget->style->black_gc,
                x0,y0,
                x0,height);
  gdk_draw_line(widget->window,
                widget->style->black_gc,
                x0 + wwidth,y0,
                x0 + wwidth,height);
  // +- draw the platform
  gdk_draw_rectangle(widget->window,
                     widget->style->black_gc,
                     1,
                     x0,y0,
                     wwidth, hwidth);

  // start moving the platform
  if ((moving == 0) && (mouseaction == 1)) {
    if ((mx > x0) && (mx < (x0 + wwidth))
        && (my > y0) && (my < (y0 + hwidth))) {
      // click inside form platform -- moving...
      moving = 1;
      // offset for moving
      ox = mx - x0;
      oy = y0 - my;
    } else {
      // error, click outside from platform
      moving = 2;
    }
  }
}


```

F.2 src/example/guirobot.c

```
/**
   @file

   Robot-Side application with GUI (simulator) to control the robot.
*/

#include <time.h>
#include <math.h>
#include <pthread.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <gtk/gtk.h>
#include <glib.h>
#include <unistd.h>
#include <signal.h>
#include <gdk/gdk.h>

#include "../lib/libcomm.h"
#include "guicommon.h"

// Pointer to the Main-Application-Object. Is necessary if a function
// of the GUI is called from a not-gui-object.
GtkWidget *gr_application;

// Pointer to the DrawinArea-Object. Is necessary if a function
// of the DrawingArea is called from a not-gui-object.
GtkWidget *da_global;

// Stores the absolut position of the robot.
int robot_x = 50, robot_y = 0;

// Configuration for LineMonitor. Stores the Soft- and Hard-Timeout
// and wait-times.
int soft_msec;
int hard_msec;
int wait_msec;

// Indicates the the "system" is shutting down. Do not launch further
// command to the GUI
int shut_down;

// Stores the portnumber on which the server operates. serverport:
// Datalink (local/bind); serverport + 1: Linemonitor (remote/connect).
int serverport;



/**

    @param widget (GtkWidget *) pointer to affected object (for
    example: window).

    @param event (GdkEventM *) pointer to structure which
    describes the event.

    @param data (gpointer) pointer to additional data from gtk.


*/
static gbooleandelete_event(GtkWidget *widget,
                            GdkEvent  *event,
                            gpointer   data) {
  interface_stop(); // ??
  fprintf(stderr, "Shut down...\n");
  gtk_main_quit();
  return FALSE;
}


/** This function will be called be gtk when the window gets a closing
    command, it makes sure that the interface is down and the program
    is finished.

    @param widget (GtkWidget *) pointer to affected object (for
    example: window).
```

```

    @param data (gpointer) pointer to additional data from gtk.
 */
void gr_delete_event(GtkWidget *widget, gpointer data) {
  interface_stop();
  shut_down = 1;
  fprintf(stderr, "Shut down...\n");
  gtk_main_quit();
  gdk_window_process_all_updates();
}


/** This function is called if a signal (line C-c, terminate)
    occurs. See in signal() -calls in main() for detals.

    @param sig (int) number of orrured signal
*/
static void finish(int sig) {
  gr_delete_event(gr_application, NULL);
}


/** This function is called by gtk if the drawing area (da) needs to
    be redrawn. For example if it becomes visible.

    @param widget (GtkWidget *) pointer to affected object (in this
    case the drawing area).

    @param data (gpointer) pointer to additional data from gtk.
*/
void gr_da_expose_event(GtkWidget *widget, gpointer data) {
  gui_common_drawrobot(widget, &robot_x, &robot_y, -1, -1, 0);
}



/** This function is called if a button is pressed

    @param widget (GtkWidget *) pointer to affected object (button).

    @param event (GdkEventMotion *) pointer to structure which
    describes the event.
*/

void gr_button_press_event(GtkWidget *widget, gpointer data) {
  if(strcmp("Emergency Stop", (char *)data) == 0) {
    gr_delete_event(gr_application, NULL);
    g_print("Emergency Stop Button pressed\n");
  }
}


/**
   Function which is called from block_call() if a
   message/datablock has received. See API of block_call().
*/
gr_block_call_do_test(int fd, int id, unsigned int type, char *buf,
                      unsigned int size, int term) {
  struct ROBOT_POSITION *robot_position;

  // do nothing, if the system is shutting down
  if (shut_down) {
    return;
  }

```

```
  if ((sizeof(struct ROBOT_POSITION) == size) && (type == 1)) {
    // load new position of the robot from the received package
    robot_position = (struct ROBOT_POSITION *) buf;
    robot_x = robot_position -> x;
    robot_y = robot_position -> y;
    free(robot_position);

    // drive robot to the new coordinates
    interface_driveto(robot_x, robot_y);

    // make sure, that thre is no conflict with the
    // gdk-main-loop. This is a locking-mechanism which privents
    // "Xlib: unexpected async reply"s
    gdk_threads_enter();

    // re-draw the robot with the new coordinates
    gui_common_drawrobot(da_global, &robot_x, &robot_y, -1, -1, 0);

    // make sure, that all changed item are really plotted n the screen
    gdk_window_process_all_updates();

    // unlock gdk-main-loop
    gdk_threads_leave();

    // some of the other tries to force a re-draw of the screen -- all
    // useless!
    /*    while(gtk_events_pending() && !shut_down) {
            gtk_main_iteration();
          } */
    //    gdk_flush();
    //    da_global->queue_draw();
    //    gtk_widget_draw(da_global, da_global);
    /*    gtk_widget_queue_draw(da_global);
          gtk_widget_queue_draw(gr_application);*/
    //    gtk_widget_queue_clear(da_global->window);
    //    da_global -> queue_draw();
    //    gtk_widget_draw(da_global, NULL);
    /*    gtk_signal_emit_by_name(da_global, "expose_event",
          NULL, 1);*/
    //    gtk_signal_emit_by_name(GTK_OBJECT(da_global), "changed");
    //    gtk_signal_emit_by_name(GTK_OBJECT(da_global), "expose_event");
    //    gtk_widget_show_all(da_global);
    //    gdk_window_hide(da_global->window);
    //    gdk_window_show(da_global->window);
    //    gdk_window_get_update_area(da_global->window);
    //    gtk_widget_queue_draw(da_global);
  } else {
    if (type != 1) {
      fprintf(stderr, "** WARNING: Unknown type of datablock received!\n");
    } else {
      fprintf(stderr, "** WARNING: Datablock with improper size received!\n");
    }
  }
}


/** Exception function for the linemonitor(), print exception code and
    its meaning on the screen and take further action if
    necessary. See API of linemonitor(). */
linemonitor_exception(char *server, int port, int type) {
  fprintf(stderr, "linemonitor_exception(%s, %d, %d): ",
          server, port, type);
  switch(type) {
  case 0:
    fprintf(stderr, "Connecion Fault\n");
```

```
      gr_delete_event(gr_application, NULL);
      break;
    case 1:
      fprintf(stderr, "Soft Real Time Exception\n");
      break;
    case 2:
      fprintf(stderr, "HARD Real Time Exception\n");
      gr_delete_event(gr_application, NULL);
      break;
    case 3:
      fprintf(stderr, "Transmission Fault\n");
      gr_delete_event(gr_application, NULL);
      break;
    case 4:
      fprintf(stderr, "Emergency Stop\n");
      gr_delete_event(gr_application, NULL);
      break;
    } /* switch() */
}
/**
   Function which ist called from block_call() if a the connection
   terminates: shut down system. See block_call() API.
*/
gr_block_call_term_test(int fd, int id) {
  gr_delete_event(gr_application, NULL);
}
/**
   Function which is called from socket_accept() if someone has
   connected. See socket_accept() API.
*/
gr_socket_accept_do(int fd, int id, char *pip,
                      struct sockaddr_in their_addr, int term) {
  // starting line-monitor
  char *pard = inet_ntoa(their_addr.sin_addr);
  linemonitor(pard, serverport + 1,
              soft_msec, hard_msec, wait_msec,
              linemonitor_exception);
  // waiting for incomming data in an other thread
  block_call(fd, id, false,
             (void *) gr_block_call_do_test,
             (void *) gr_block_call_term_test);
}
int main(int argc, char *argv[]) {
  // file discriptor for the local bind.
  int sock;
  // GTK-Objecte. This is necessary to create the buttons on the
  // screen and connect them to some actions
  GtkWidget *button1, *button2, *button3, *button4;
  // Sorting into tables.
```

```
  GtkWidget *table, *table2;

  // system is not shutting down now... (it shutting up ;-)
  shut_down = 0;

  // Init the gtk (GUI toolkit) and the gdk (threads) system
  g_thread_init(NULL);
  gdk_threads_init();
  gtk_init(&argc, &argv);

  // Connect the finish()-function to some signals which can cause a
  // system shut down.
  signal(SIGHUP, finish);   // 01  /  hangup      - close Window
  signal(SIGINT, finish);   // 02  /  Interrupt   - ^C - C-c
  signal(SIGQUIT, finish); // 03  /  Quit
  signal(SIGTERM, finish); // 15  /  Terminierung -- kill
  signal(SIGALRM, finish); // 14  /  Alarm

  // load and examine parameters
  if (argc == 6) {
    serverport = atoi(argv[1]);
    soft_msec = atoi(argv[2]);
    hard_msec = atoi(argv[3]);
    wait_msec = atoi(argv[4]);
    interface_init(2);
  } else {
    if (argc == 5) {
      serverport = atoi(argv[1]);
      soft_msec = atoi(argv[2]);
      hard_msec = atoi(argv[3]);
      wait_msec = atoi(argv[4]);
      interface_init(0);
    } else {
      fprintf(stderr, "%s port-to-bind soft_msec hard_msec wait_msec\n", argv[0]);
      //       exit(0);
    }
  }

  // bind local port and launch socket_accept() to wait for connection
  if ((sock = socket_bind(serverport, 10)) < 0) {
    error("bind()");
  } else {
    socket_accept(sock, 0, (void *) gr_socket_accept_do);
  }

  // create button(s)
  button1  = gtk_button_new_with_label("Emergency Stop");
  /*  button2  = gtk_button_new_with_label("Button 2");
  button3  = gtk_button_new_with_label("Button 3");
  button4  = gtk_button_new_with_label("Button 4"); */

  // connect the buttons to the function gr_button_press_event(). If
  // the botton is pressed, this function will be colled.
  gtk_signal_connect(GTK_OBJECT(button1), "clicked",
                     GTK_SIGNAL_FUNC(gr_button_press_event),
                     "Emergency Stop" );
  /*  gtk_signal_connect(GTK_OBJECT(button2), "clicked",
                     GTK_SIGNAL_FUNC(gr_button_press_event),
                     "Button 2");
  gtk_signal_connect(GTK_OBJECT(button3), "clicked",
                     GTK_SIGNAL_FUNC(gr_button_press_event),
                     "Button 3" );
  gtk_signal_connect(GTK_OBJECT(button4), "clicked",
                     GTK_SIGNAL_FUNC(gr_button_press_event),
                     "Button 4"); */
```

```

    // create tables to sort buttons and drawindarea in it
    table = gtk_table_new(2,2,FALSE);
    table2 = gtk_table_new(2,2,FALSE);

    // create new application (wiindow)
    gr_application = gtk_window_new(GTK_WINDOW_TOPLEVEL);

    // set window title
    gtk_window_set_title(GTK_WINDOW (gr_application), "Robot");

    // connect the function gr_delete_event() with the event of a
    // window-close.
    g_signal_connect(G_OBJECT (gr_application), "delete_event",
                     G_CALLBACK (gr_delete_event), NULL);

    // display window
    gtk_widget_show(gr_application);

    // create drawing area
    da_global = gtk_drawing_area_new();

    // set which events in the drawing area causing a expose-event
    gtk_widget_set_events(da_global, GDK_EXPOSURE_MASK
                          | GDK_LEAVE_NOTIFY_MASK
                          | GDK_BUTTON_PRESS_MASK
                          | GDK_POINTER_MOTION_MASK
                          | GDK_POINTER_MOTION_HINT_MASK);

    // set a minimum size of drawing area
    gtk_widget_set_size_request(da_global, 100, 100);

    // connect the function gr_delete_event() with the event of a
    // window-close.
    gtk_signal_connect(GTK_OBJECT(gr_application), "delete_event",
                       GTK_SIGNAL_FUNC(gr_delete_event), NULL);

    // connect the function gr_da_expose_event() with the expose-event
    // of the drawing area.
    gtk_signal_connect(GTK_OBJECT(da_global), "expose_event",
                       GTK_SIGNAL_FUNC(gr_da_expose_event), NULL);

    // fill the buttons in the table
    gtk_table_attach_defaults(GTK_TABLE(table2), button1, 0,1, 0,1);
    /*  gtk_table_attach_defaults(GTK_TABLE(table2), button2, 0,1, 1,2);
    gtk_table_attach_defaults(GTK_TABLE(table2), button3, 1,2, 0,1);
    gtk_table_attach_defaults(GTK_TABLE(table2), button4, 1,2, 1,2); */

    // fill the drawing area and the table into another table
    gtk_table_attach_defaults(GTK_TABLE(table), da_global, 0,1, 0,1);
    gtk_table_attach_defaults(GTK_TABLE(table), table2, 0,1, 1,2);

    // fill this table into the windos
    gtk_container_add(GTK_CONTAINER(gr_application),table);

    // display it
    gtk_widget_show_all(gr_application);

    // call gtk-main-loop
    gtk_main();

    return 0;
}


```

F.3 src/example/guiserver.c

```
/**
   @file

   Server/User-Side Application with GUI to control the robot.
*/

#include <time.h>
#include <math.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>
#include <glib.h>
#include <unistd.h>
#include <signal.h>

#include "../lib/libcomm.h"
#include "guicommon.h"

// Pointer to the Main-Application-Object. Is necessary if a function
// of the GUI is called from a not-gui-object.
GtkWidget *gs_application;

// Stores the absolut position of the robot.
int robot_x = 50, robot_y = 0;
// ... fill it into a structure to be able to transfer it
struct ROBOT_POSITION robot_position;

// Stores the portnumber on which the server operates. serverport:
// Datalink (remote/connect); serverport + 1: Linemonitor
// (local/bind).
int serverport;

// filediscriptor which pointes to the socket-stream which is
// connected to the robot's side and used fot the data-transfer.
int sock;

// Configuration for LineMonitor. Stores the Soft- and Hard-Timeout
// and wait-times.
int soft_msec;
int hard_msec;
int wait_msec;

// File discriptor to the linemonitor() connection. Used for Emergercy
// Stop command
int ems_fd;



/**

    @param widget (GtkWidget *) pointer to affected object (for
    example: window).

    @param event (GdkEventM *) pointer to structure which
    describes the event.

    @param data (gpointer) pointer to additional data from gtk.

*/
static gbooleandelete_event(GtkWidget *widget,
                            GdkEvent  *event,
```

```
                              gpointer   data) {

  linemonitor_emergencystop(ems_fd); // ??
  fprintf(stderr, "Shut down...\n");
  gtk_main_quit();
  gdk_window_process_all_updates();
  return FALSE;
}


/** This function is called if a signal (line C-c, terminate)
    occurs. See in signal() -calls in main() for detals.

    @param sig (int) number of orrured signal
*/
static void finish(int sig) {
  gs_delete_event(gs_application, NULL);
}


/** This function will be called be gtk when the window gets a closing
    command, it makes sure that the interface is down and the program
    is finished.

    @param widget (GtkWidget *) pointer to affected object (for
    example: window).

    @param data (gpointer) pointer to additional data from gtk.
 */
void gs_delete_event(GtkWidget *widget, gpointer data) {
  linemonitor_emergencystop(ems_fd);
  fprintf(stderr, "Shut down...\n");
  gtk_main_quit();
  gdk_window_process_all_updates();
}


/** This function is called by gtk if the drawing area (da) needs to
    be redrawn. For example if it becomes visible.

    @param widget (GtkWidget *) pointer to affected object (in this
    case the drawing area).

    @param data (gpointer) pointer to additional data from gtk.
*/
void gs_da_expose_event(GtkWidget *widget, gpointer data) {
  gui_common_drawrobot(widget, &robot_x, &robot_y, -1, -1, 0);
}


/** This function is called if an event occurs (mose motion or mouse
    button press) over the drawing area (da).

    @param widget (GtkWidget *) pointer to affected object (in this
    case the drawing area).

    @param event (GdkEventMotion *) pointer to structure which
    describes the event.
*/
static gint
motion_notify_event(GtkWidget *widget, GdkEventMotion *event) {
  // variables for the position of the mouse
  int x, y;

  GdkModifierType state;
```

```

  // member-variable to store if a moving action takes place or not
  static int move = 0;

  // get positoin of the mouse-pointer
  if (event->is_hint) {
    gdk_window_get_pointer (event->window, &x, &y, &state);
  } else {
    x = event->x;
    y = event->y;
    state = event->state;
  }

  // if left mouse button is pressed ...
  if (state & GDK_BUTTON1_MASK) {
    // ... re-draw robot and dertermine new position of the robot if
    // the click was on the robot
    gui_common_drawrobot(widget, &robot_x, &robot_y, x, y, 1);
    move = 1;
    // if the connection to the robot is established...
    if (sock != 0) {
      // ... and the position of it was changed ...
      if ((robot_position.x != robot_x) ||
          (robot_position.y != robot_y)) {
        robot_position.x = robot_x;
        robot_position.y = robot_y;

        // ... send the new coordinates to the robot
        block_send(sock, 1, (char *) &robot_position,
                   sizeof(struct ROBOT_POSITION));
      }
    }
  } else {
    // moving event is finished, drwa robot.
    if (move) {
      gui_common_drawrobot(widget, &robot_x, &robot_y, x, y, 2);
      move = 0;
    }
  }

  return TRUE;
}



/** This function is called if a button is pressed

    @param widget (GtkWidget *) pointer to affected object (button).

    @param event (GdkEventMotion *) pointer to structure which
    describes the event.
*/

void gs_button_press_event(GtkWidget *widget, gpointer data) {
  if(strcmp("Emergency Stop", (char *)data) == 0) {
    linemonitor_emergencystop(ems_fd);
    gs_delete_event(gs_application, NULL);
    g_print("Emergency Stop Button pressed\n");
  }
}



/** Exception function for linemonitor_server(), print exception
    code and meaning on the screen and initiate appropriate
```

```
     actions if necessary. See linemonitor_server() API. */
linemonitor_exception(char *server, int port, int type) {
  fprintf(stderr, "linemonitor_exception(%s, %d, %d): ",
          server, port, type);
  switch (type) {
  case 0:
    fprintf(stderr, "Connecion Fault\n");
    gs_delete_event(gs_application, NULL);
    break;
  case 1:
    fprintf(stderr, "Soft Real Time Exception\n");
    break;
  case 2:
    fprintf(stderr, "HARD Real Time Exception\n");
    gs_delete_event(gs_application, NULL);
    break;
  } /* switch() */
}


/**
   Start the linemonitor_server() -- waiting for a connection in a
   background-thread. This has to be a thread because otherwise the
   system would block.  It would wait for a connection while it is
   supposed to connect itself.
*/
thread_wait_for_linemonitor() {
  ems_fd = linemonitor_server(serverport + 1,
                              soft_msec, hard_msec, wait_msec,
                              (void *) linemonitor_exception);
  pthread_exit(NULL);
}



int main(int argc, char *argv[]) {
  // GTK-Objecte. This is necessary to create the buttons on the
  // screen and connect them to some actions
  GtkWidget *button1, *button2, *button3, *button4;
  // Sorting into tables.
  GtkWidget *table, *table2;
  // pointer to the DrawingArea-Object
  GtkWidget *da;

  // Init the gtk (GUI toolkit)
  gtk_init(&argc, &argv);

  // Connect the finish() -function to some signals which can cause a
  // system shut down.
  signal(SIGHUP, finish);  // 01  /  hangup     - close Window
  signal(SIGINT, finish);  // 02  /  Interrupt  - ^C - C-c
  signal(SIGQUIT, finish); // 03  /  Quit
  signal(SIGTERM, finish); // 15  /  Terminierung -- kill
  signal(SIGALRM, finish); // 14  /  Alarm

  // load and examine parameters
  if (argc != 6) {
    sock = 0;
    fprintf(stderr, "%s robot-address port soft_msec hard_msec wait_msec\n", argv[0]);
    //     exit(0);
  } else {
    // ID and atributes for the threads
    pthread_t        thrd_2;
    pthread_attr_t   thrd_2_attr;

```

```
    serverport = atoi(argv[2]);
    soft_msec = atoi(argv[3]);
    hard_msec = atoi(argv[4]);
    wait_msec = atoi(argv[5]);

    // launch thread_wait_for_linemonitor(), see
    // thread_wait_for_linemonitor().
    pthread_attr_init(&thrd_2_attr);
    pthread_create(&thrd_2,
                   &thrd_2_attr,
                   (void *) thread_wait_for_linemonitor,
                   NULL);

    // make sure, that the linemonitor_server() is ready before
    // connect to the robot.
    sleep(1);

    // connect to the robot.
    if ((sock = socket_connect(argv[1], serverport)) <= 0) {
      perror("connect()");
      sock = 0;
    }
  }

  // create button(s)
  button1  = gtk_button_new_with_label("Emergency Stop");
  //  button2  = gtk_button_new_with_label("Button 2");
  //  button3  = gtk_button_new_with_label("Button 3");
  //  button4  = gtk_button_new_with_label("Button 4");


  // connect the buttons to the function gr_button_press_event(). If
  // the botton is pressed, this function will be colled.
  gtk_signal_connect(GTK_OBJECT(button1), "clicked",
                     GTK_SIGNAL_FUNC(gs_button_press_event),
                     "Emergency Stop" );
  /*  gtk_signal_connect(GTK_OBJECT(button2), "clicked",
                     GTK_SIGNAL_FUNC(gs_button_press_event),
                     "Button 2");
  gtk_signal_connect(GTK_OBJECT(button3), "clicked",
                     GTK_SIGNAL_FUNC(gs_button_press_event),
                     "Button 3" );
  gtk_signal_connect(GTK_OBJECT(button4), "clicked",
                     GTK_SIGNAL_FUNC(gs_button_press_event),
                     "Button 4");*/

  // create tables to sort buttons and drawindarea in it
  table = gtk_table_new(2,2,FALSE);
  table2 = gtk_table_new(2,2,FALSE);

  // create new application (wiindow)
  gs_application = gtk_window_new(GTK_WINDOW_TOPLEVEL);

  // set window title
  gtk_window_set_title(GTK_WINDOW (gs_application),
                       "Server - User Interface");

  // connect the function gr_delete_event() with the event of a
  // window-close.
  g_signal_connect(G_OBJECT (gs_application), "delete_event",
                   G_CALLBACK (gs_delete_event), NULL);

  // display window
  gtk_widget_show(gs_application);

```

```
    // create drawing area
    da = gtk_drawing_area_new();

    // set which events in the drawing area causing a expose-event
    gtk_widget_set_events (da, GDK_EXPOSURE_MASK
                               | GDK_LEAVE_NOTIFY_MASK
                               | GDK_BUTTON_PRESS_MASK
                               | GDK_POINTER_MOTION_MASK
                               | GDK_POINTER_MOTION_HINT_MASK);

    // set a minimum size of da
    gtk_widget_set_size_request (da, 100, 100);

    // connect the function gr_delete_event() with the event of a
    // window-close.
    gtk_signal_connect(GTK_OBJECT(gs_application), "delete_event",
                       GTK_SIGNAL_FUNC(gs_delete_event), NULL);

    // connect the function gr_da_expose_event() with the expose-event
    // of the drawing area.
    gtk_signal_connect(GTK_OBJECT(da), "expose_event",
                       GTK_SIGNAL_FUNC(gs_da_expose_event), NULL);

    // connect the function motion_notify_event() with the
    // mouse-motion-event of the drawing area.
    gtk_signal_connect(da, "motion_notify_event",
                       GTK_SIGNAL_FUNC(motion_notify_event), NULL);

    // fill the buttons in the table
    gtk_table_attach_defaults(GTK_TABLE(table2), button1, 0,1, 0,1);
    /*  gtk_table_attach_defaults(GTK_TABLE(table2), button2, 0,1, 1,2);
    gtk_table_attach_defaults(GTK_TABLE(table2), button3, 1,2, 0,1);
    gtk_table_attach_defaults(GTK_TABLE(table2), button4, 1,2, 1,2); */

    // fill the drawing area and the table into another table
    gtk_table_attach_defaults(GTK_TABLE(table), da, 0,1, 0,1);
    gtk_table_attach_defaults(GTK_TABLE(table), table2, 0,1, 1,2);

    // fill this table into the windos
    gtk_container_add(GTK_CONTAINER(gs_application),table);

    // display it
    gtk_widget_show_all(gs_application);

    // call gtk-main-loop
    gtk_main();

    return 0;
}


```

Appendix G: Source Code of the Tests

G.1 src/test/test001sockets.c

```
/**
   @file

```

```
   This program tests the low-level tcp-socket-stream library function.

*/

/*
  Copyright (c) Andreas Hofmeier
  (www.an-h.de, www.an-h.de.vu, www.lgut.uni-bremen.de/an-h/)

  This program is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 2 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful, but
  WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
  General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program; if not, write to the Free Software
  Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*/


#include <stdio.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <signal.h>
#include "../lib/libcomm.h"

/** Port on which the server is listening and the client trys to
    connect. */
#define PORT 1234
/** How much random bytes should be transfered? */
#define bufsize 8192
//#define bufsize 8

/** test0001sockts: see test0001sockts.c.
*/

main(int argc, char *argv[]) {
  // PID of the child process
  int fpid;

  printf("running %s...\n", argv[0]);
  if (!(fpid = fork())) {
    server_test_program(PORT);
  }

  client_test_program("127.0.0.1", PORT);
  client_test_program("localhost", PORT);
  client_test_program("lblacky", PORT);
  client_test_program("hofmeira.student.sbu.ac.uk", PORT);

  // kill server terst programm
  kill(fpid, 15);
  sleep(1);
  kill(fpid, 9);

  exit(0);
}


/** test0001sockts: see test0001sockts.c, Client Test Programm.

```

```
    The client test program generates a block of n*(bufsize) random
    bytes, sents theses bytes to the server, receive a block from
    server, invert it and compare it with generated block.

    BUGS: Receives only one block. If not all data ready and function
    resv() do noct block, data get lost. Test fails.

    @param server a string which contains the server name
    @param port an integer which specifies the port on the server
*/
client_test_program(char *server, int port) {
  // ******************************************************
  // *** Client Test Programm

  int i;

  // FD of socket
  int sock;

  char buf[bufsize], buf2[bufsize];

  printf("  Generating random numbers...\n");
  if (block_random(buf, bufsize) == NULL) {
    fail("cannot generate random numbers", 1);
  }

  // Connect to server
  printf("  Try server %s...", server);
  if ((sock = socket_connect(server, port)) < 0) {
    fail("Cannot connect.", 1);
  }

  // send datablock to server
  send(sock, buf, bufsize, 0);

  // receive datablock from server
  recv(sock, buf2, bufsize, 0);

  close(sock);

  // invert the bits of the local datablock and
  // compare it with the result from the server
  // should be the same.
  for (i = 0; i < bufsize; i++) {
    if (~buf2[i] != buf[i]) {
      fail("some errors occure during the data transfer...");
    }
  }
  printf("OK.\n");
  sleep(1);
}


/** test0001sockts: see test0001sockts.c, Server Test Programm.

    The server test program binds a port and waits for connections. If
    someone connects it reads n*(bufsize) bytes, invert all bits of
    these bytes and send all back.

    BUGS: Receives only one block. If not all data ready and function
    resv() do noct block, data get lost. Test fails.

    @param port an integer which specifies the port to bind.
*/
server_test_program(int port) {
```

```
  // ***********************************************************
  // *** Server Test Program

  // buffer for storing data.
  char buf[bufsize];

  // FD of socket which is bounded to the port
  int sockport;
  // FD of socket
  int sock;

  /* connector's address information */
  struct sockaddr_in their_addr;
  int sin_size;


  // Bind port
  printf("  binding port %d on localhost...", port);
  if ((sockport = socket_bind(port, 10)) < 0) {
    fail("Cannot bind port.", 1);
  }

  while (1) {
    // accept connection
    sin_size = sizeof(struct sockaddr_in);
    if ((sock = accept(sockport, (struct sockaddr *) &their_addr,
                       &sin_size)) != -1) {
      int i, size;
      char *pard = inet_ntoa(their_addr.sin_addr);
      fprintf(stdout, "  got connection from %s\n", pard);

      // receive datablock from client
      size = recv(sock, buf, bufsize, 0);
      // invert the bits of thh whole datablock
      for (i = 0; i < size; i++) {
        buf[i] = ~buf[i];
      }
      // send datablock to client
      send(sock, buf, size, 0);

      close(sock);
    }
  }
}
```

G.2 src/test/test002integer.c

```
/**
   @file

   This program tests the size and the organisation of an integer on
   the local machine, with the local compiler.

*/

/*
  Copyright (c) Andreas Hofmeier
  (www.an-h.de, www.an-h.de.vu, www.lgut.uni-bremen.de/an-h/)

  This program is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 2 of the License, or
```

```
  (at your option) any later version.

  This program is distributed in the hope that it will be useful, but
  WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
  General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program; if not, write to the Free Software
  Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*/

#include <stdio.h>


main(int argc, char *argv[]) {
  unsigned int i;
  int size = sizeof(i);
  unsigned char *s, c;

  // make sure, that the size of the integer is greater or equal than
  // two bytes
  printf("testing integer...\n");
  printf("  sizeof(int) = %d Bytes (%d Bits)\n", size, size * 8);
  if (size < 2) {
    fail("The size of integer must be greater or equal that two " \
         "bytes (or 16 bits)", 1);
  }
  s = (char *) &i;
  printf("  organisation: ");
  for (i = 1; i <= 128; i = i * 2) {
    c = (unsigned char) i;
    if ((s[0] != c) ||
        (s[1] != 0)) {
      fail("organisation of Integer is on this machine different, " \
           "than the library expect.", 1);
    }
    printf("%d ", i);
  }
  for (i = 256; i <= 32768; i = i * 2) {
    c = (unsigned char) (i / 256);
    if ((s[0] != 0) ||
        (s[1] != c)) {
      fail("organisation of Integer is on this machine different, " \
           "than the library expect.", 1);
    }
    printf("%d ", i);
  }
  printf(" OK.\n");
  exit(0);
}
```

G.3 src/test/test003block.c

```
/**
   @file

   This program tests the low-level block transfer and authentication
   functions.

*/

```

```c
/*
  Copyright (c) Andreas Hofmeier
  (www.an-h.de, www.an-h.de.vu, www.lgut.uni-bremen.de/an-h/)

  This program is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 2 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful, but
  WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
  General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program; if not, write to the Free Software
  Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*/


#include <stdio.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <signal.h>
#include "libcomm.h"
#include <unistd.h>


/** Port on which the server is listening and the client trys to
    connect. */
#define PORT 1235
/** How much random bytes should be transfered? */
#define bufsize 8192
/** The ID which identifies the thread of the block_call()
    function. This value can be coosen arbitrary and is passwd to all
    called functions. */
#define threadid 1234
//#define bufsize 8

/** test0001sockts: see test0001sockts.c.
*/

main(int argc, char *argv[]) {
  // PID of the child process
  int fpid;
  int iport = PORT;

  printf("running %s...\n", argv[0]);

  printf("  trying blocked read mode... ");
  fflush(stdout);
  if (!(fpid = fork())) {
    server_test_program(iport, 0);
  }

  client_test_program("localhost", iport++, 0);

  // kill server terst programm
  kill(fpid, 9);


  printf("  trying non-blocked read mode... ");
  fflush(stdout);
  if (!(fpid = fork())) {
    server_test_program(iport, 1);
```

```
  }
  client_test_program("localhost", iport++, 0);

  // kill server terst programm
  kill(fpid, 9);

  printf("  trying function-call read mode... ");
  fflush(stdout);
  if (!(fpid = fork())) {
    server_test_program(iport, 2);
  }
  client_test_program("localhost", iport++, 0);

  // kill server terst programm
  kill(fpid, 9);

  printf("  trying accept-call and function-call read mode... ");
  fflush(stdout);
  server_test_program(iport, 3);
  sleep(2);
  client_test_program("localhost", iport++, 0);

  printf("  testing authentification... ");
  fflush(stdout);
  if (!(fpid = fork())) {
    server_test_program(iport, 4);
  }

  client_test_program("localhost", iport++, 1);

  // kill server terst programm
  kill(fpid, 9);


  exit(0);
}


/** test0001sockts: see test0001sockts.c, Client Test Programm.

   The client test program generates a block of n*(bufsize) random
   bytes, sents theses bytes to the server, receive a block from
   server, invert it and compare it with generated block.

   BUGS: Receives only one block. If not all data ready and function
   resv() do noct block, data get lost. Test fails.

   @param server (char *) contains the server name

   @param port (int) specifies the port on the server

   @param auth (int) 0: normale test; 1: do authentification before
   run test.

*/
client_test_program(char *server, int port, int auth) {
  // *******************************************************
  // *** Client Test Programm

  int i;

  // FD of socket
  int sock;

  char buf[bufsize], buf2[bufsize];
```

```
  int type, type2, rsize;

  // wait one second, give the fork enough time to bind and listen
  // the socket
  sleep(1);

  if (block_random(buf, bufsize) == NULL) {
    fail("cannot generate random numbers", 1);
  }
  if (block_random((char *) &type, 2) == NULL) {
    fail("cannot generate random numbers", 1);
  }

  // Connect to server
  if ((sock = socket_connect(server, port)) < 0) {
    fail("Cannot connect.", 1);
  }

  sleep(3);

  if (auth) {
    struct AUTHINFO *local, *remote;

    char a[] = "remote";
    char b[] = "loginname54321";

    //    if (socket_md5auth(sock, NULL, &b, &local, &remote) < 0) {
    if (socket_md5auth(sock, (char *) &a, NULL, &local, &remote) < 0) {
      fail("authentifications failed!", 1);
    } else {
      printf("(client OK [%s:%s]) ", local->name, local->passwd);
      fflush(stdout);
    }
  }

  // send datablock to server
  block_send(sock, type, buf, bufsize);

  // receive datablock from server
  block_receive(sock, &type2, buf2, &rsize, bufsize, false);
  if (rsize != bufsize) {
    fprintf(stderr, "WARNING: received less data from server than was" \
            " send.\n(%d:%d)\n", rsize, bufsize);
  }

  close(sock);

  // invert the bits of the local datablock and
  // compare it with the result from the server
  // should be the same.
  for (i = 0; i < bufsize; i++) {
    if (~buf2[i] != buf[i]) {
      fail("some errors occure during the data transfer...", 1);
    }
  }
  printf("OK.\n");
  fflush(stdout);
  sleep(1);
}


/**
   testfunction which ist called from block_call() if a
   message/datablock has received.
*/
```

```
block_call_do_test(int fd, int id, unsigned int type, char *buf,
                   unsigned int size, int term) {
  int i;

  if (id != threadid) {
    fprintf(stderr, "** WARNING: Thread-ID was not stored correctly!");
  }

  // invert the bits of the whole datablock
  for (i = 0; i < size; i++) {
    buf[i] = ~buf[i];
  }

  //  printf("block_call_do_test()");
  fflush(stdout);

  // send inverted datablock to client
  block_send(fd, type, buf, size);
  close(fd);
}


/**
   testfunction which ist called from block_call() if a
   the connection terminates.
*/
block_call_term_test(int fd, int id) {
  int i;

  if (id != threadid) {
    fprintf(stderr, "** WARNING: Thread-ID was not stored correctly!");
  }

  printf("(server: connection terminated.)\n");
  fflush(stdout);
}


/**
   testfunction which ist called from socket_accept() if someone has
   connected.
*/
socket_accept_do_test(int fd, int id, char *pip,
                      struct sockaddr_in their_addr, int term) {
  block_call(fd, id, false,
             (void *) block_call_do_test,
             (void *) block_call_term_test);
}


/** test0001sockts: see test0001sockts.c, Server Test Programm.

   The server test program binds a port and waits for connections. If
   someone connects it reads n*(bufsize) bytes, invert all bits of
   these bytes and send all back.

   BUGS: Receives only one block. If not all data ready and function
   resv() do noct block, data get lost. Test fails.

   @param port (int) specify the port to bind.

   @param mode (int) select the mode of receiving a message/datablock:
   0: blocked mode, use block_receive(); 1: poll mode, poll with
   block_receive_poll(); 2: call function block_call_do_test() if a
   block is received, use block_call(); 3: call
```

```
    socket_accept_do_test() if someone has connected. This function
    calls block_call() which does the same like in 2, use
    socket_accept(); 4: testing authenication by using
    socket_md5auth(). After this do the same as in 0.
*/
server_test_program(int port, int mode) {
  // ***********************************************************
  // *** Server Test Program

  // buffer for storing data.
  char *buf;

  // FD of socket which is bounded to the port
  int sockport;
  // FD of socket
  int sock;

  /* connector's address information */
  struct sockaddr_in their_addr;
  int sin_size;


  // Bind port
  if ((sockport = socket_bind(port, 10)) < 0) {
    fail("Cannot bind port.", 1);
  }

  if (mode == 3) {
    socket_accept(sockport, threadid, (void *) socket_accept_do_test);
    return;
  }

  // accept connection
  sin_size = sizeof(struct sockaddr_in);
  if ((sock = accept(sockport, (struct sockaddr *) &their_addr,
                     &sin_size)) != -1) {
    int i, size, type;
    char *pard = inet_ntoa(their_addr.sin_addr);
    //    fprintf(stdout, "  got connection from %s\n", pard);


    // receive datablock from client
    switch (mode) {
    case 0:
      // blocking function
      buf = block_receive(sock, &type, NULL, &size, 0, false);
      if (buf == NULL) {
        fprintf(stderr, "Can't receive block from client...\n");
        exit(1);
      }
      break;
    case 1:
      // polling function
      i = 0;
      do {
        usleep(2L);
        i++;
        buf = block_receive_poll(sock, &type, NULL, &size, 0, false);
      } while (buf == (char *) 1L);
      if (buf == NULL) {
        fprintf(stderr, "Error during receivion occured...\n");
        exit(1);
      }
      printf("(%d polls) ", i);
      fflush(stdout);
```

```
        break;
      case 2:
        // function call
        block_call(sock, threadid, false, (void *) block_call_do_test,
                   (void *) block_call_term_test);
        sleep(500); // simulate the running of the "normal" program...
        return;
        break;
      case 4:
        // authentification with blocking function
        {
          struct AUTHINFO *local, *remote;
          char a[] = "loginname12345";

          if (socket_md5auth(sock, NULL, (char *) &a, &local, &remote) == 0) {
            printf("(server OK [%s:%s]) ", local->name, local->passwd);
            fflush(stdout);
          }
          buf = block_receive(sock, &type, NULL, &size, 0, false);
          if (buf == NULL) {
            fprintf(stderr, "Can't receive block from client...\n");
            exit(1);
          }
        }
        break;
      }

      // invert the bits of the whole datablock
      for (i = 0; i < size; i++) {
        buf[i] = ~buf[i];
      }

      // send inverted datablock to client
      block_send(sock, type, buf, size);
      close(sock);
    }
    close(sockport);
    exit(0);
}


```

G.4 src/test/tes005realtime.c

```
/**
   @file Server- and Client-Testprogram for the linemonitor functions.

   @param server server to be connected (clientprogram only)

   @param port to be connected (client) or port to be listened (server)

   @param soft_msec (int) timeout in milliseconds which causes
   soft-real-time exception.

   @param hard_msec (int) timeout in milliseconds which causes
   hard-real-time exception.

   @param wait_msec (int) timeout for resent -- sending of the next
   ping.
*/

/*
```

```
  Copyright (c) Andreas Hofmeier
  (www.an-h.de, www.an-h.de.vu, www.lgut.uni-bremen.de/an-h/)

  This program is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 2 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful, but
  WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
  General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program; if not, write to the Free Software
  Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*/

#include <stdio.h>
#include "libcomm.h"


/** Exception Function, print exception code and meaning on the
    screen
*/
linemonitor_exception(char *server, int port, int type) {
  printf("linemonitor_exception(%s, %d, %d): ",
         server, port, type);
  switch (type) {
  case 0:
    printf("Connecion Fault\n");
    break;
  case 1:
    printf("Soft Real Time Exception\n");
    break;
  case 2:
    printf("HARD Real Time Exception\n");
    break;
  case 3:
    printf("Transmission Fault\n");
    break;
  case 4:
    printf("Emergency Stop\n");
    break;
  } /* switch() */
}


main(int argc, char *argv[]) {
  char buf[255];
  int fd = 0;

  // Client Mode: Server Port soft_msec hard_msec wait_msec
  if (argc == 6) {
    printf("Client Mode\n");
    linemonitor(argv[1], atoi(argv[2]),
                atoi(argv[3]), atoi(argv[4]), atoi(argv[5]),
                linemonitor_exception);
  }


  // Server Mode: Port soft_msec hard_msec wait_msec
  if (argc == 5) {
    printf("Server Mode\n");
    fd = linemonitor_server(atoi(argv[1]),
```

```
                        atoi(argv[2]), atoi(argv[3]), atoi(argv[3]),
                        linemonitor_exception);
  }

  if ((argc != 5) && (argc != 6)) {
    printf("Client Mode: %s server port soft_msec hard_msec wait_msec\n" \
           "Server Mode: %s port soft_msec hard_msec wait_msec\n\n", argv[0], argv[0]);
    exit(0);
  }

  while (1) {
    gets(buf);
    linemonitor_emergencystop(fd);
  }
}
```

G.5 src/example/test001interface.c

```
/**
   @file

   Testprogram for the interface to the robot. Interface will be
   initiated, than absolutes coordinates are asked for.
*/

#include <stdio.h>


main(int argc, char **argv[]) {
  int x, y;
  interface_init(0);

  while (1) {
    printf("Enter X: ");
    scanf("%d", &x);
    printf("Enter Y: ");
    scanf("%d", &y);

    interface_driveto(x, y);
  }
}
```