FIM

The development of a Fault Injection and Monitoring application for work simulation support

Per Pascal Grube, Vikas Sidhavatula

School of Engineering
Blekinge Institute of Technology
Box 520
SE - 372 25 Ronneby
Sweden
This thesis is submitted to the Department of Interaction and System Design, School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science (Ubiquitous Computing). The thesis is equivalent to 40 weeks of full time studies.

Contact Information:
Authors:
Per Pascal Grube
Address: Folkparksvägen 15:18, 37240 Ronneby, Sweden
E-mail: pascal.grube@flightmare.net

Vikas Sidhavatula
Address: Karlskronavägen 25, 37237 Ronneby, Sweden
E-mail: vikas.vikky@gmail.com

University advisors:
Hans Kyhlbäck
School of Engineering, BTH

Björn Stille
School of Management, BTH

Department of Interaction and System Design
Blekinge Institute of Technology
Box 520
SE 372 25 Ronneby
Sweden

Internet : www.bth.se/tek
Phone : +46 457 38 50 00
Fax : + 46 457102 45
ABSTRACT

In this thesis work, we try to enhance an industry work simulator by developing a Free and Open source fault injecting and monitoring application. We present our experiences and ideas while participating in a work simulation. We implement some of these ideas to further enhance the simulation. We also try to connect these experiences to the fields of Human Computer Interaction (HCI), Computer Supported Cooperative Work (CSCW) and Ubiquitous Computing.

**Keywords:** work simulation, CSCW, interaction, Free and Open-Source Software (FOSS).
Contents

1 Introduction to the field and our approach .................................................. 3

2 The Industry Work Simulator ........................................................................... 6
   2.1 Hardware .................................................................................................... 6
       2.1.1 Industrial machines ........................................................................... 6
       2.1.2 PC .................................................................................................... 9
       2.1.3 DAQ card ......................................................................................... 10
       2.1.4 Video camera ................................................................................... 10
   2.2 Software .................................................................................................... 10
       2.2.1 Ubuntu ............................................................................................. 10
       2.2.2 COMEDI ......................................................................................... 10
       2.2.3 Python ............................................................................................ 11
       2.2.4 GTK and PyGTK .............................................................................. 11
       2.2.5 Gstreamer and PyGST ..................................................................... 11

3 The Industry Live Game .................................................................................... 12
   3.1 Idea .......................................................................................................... 12
   3.2 Working ..................................................................................................... 12
   3.3 Rules ......................................................................................................... 12
   3.4 Material used in the game ......................................................................... 13

4 The FIM ............................................................................................................ 14
   4.1 Program requirements .............................................................................. 14
       4.1.1 Installing COMEDI ......................................................................... 14
       4.1.2 Setting up the DAQ card .................................................................. 15
       4.1.3 Using the DAQ card ........................................................................ 15
   4.2 General considerations ............................................................................. 17
   4.3 TUI and GUI ............................................................................................ 19
       4.3.1 TUI - The Textmode User Interface ............................................... 19
       4.3.2 GUI - The Graphical User Interface .............................................. 21
   4.4 The COMEDI Triad .................................................................................. 22
   4.5 Logging ...................................................................................................... 23
   4.6 Video ......................................................................................................... 24
       4.6.1 The basic way of data in Gstreamer ................................................. 24
       4.6.2 Muxing and Demuxing of the Video stream .................................... 25
       4.6.3 Decoding and Encoding the Video Stream ...................................... 26
       4.6.4 Adding the channel status ................................................................ 29
1 Introduction to the field and our approach

In today’s world, industries are striving for better work environments and conduct a lot of research in many fields of which some relate to improvements of human interaction at the shop floor (see [26]). There is also an interest, both in industry and school, to explore and develop programs to train more knowledgeable workers. These training programs in turn also support research on improving work environments by providing e.g. data required by managers to better understand the environment.

This masters thesis report is about our project of further development and improving one such environment for training research. The project has a bias on technical issues but in this writing we also address some concepts concerning HCI (Human Computer Interaction) and Ubiquitous computing. The research question to be answered in this report is, if it is possible to replace the existing proprietary fault injection application by a free and open source application. Depending on the outcome of this question, it will be discussed to what extent the current way of conducting experiments in the TELL lab can be enhanced. This question proposes the use of action research kind of research methodology, since we as researchers will actively participate in further development of the environment (for a description of Action Research, see [19], p. 215). The outcome of this research type is described in chapter 5 which clearly shows that going for this approach was valid, as many details of the setup are only visible while conducting experiments which includes playing the Industry Live Game. It is a game involving pupils or students playing different kinds of roles in a simulated industrial work environment. Along with those questions, we structured the report. Looking at the website of the software to use suggested that it should be feasible to get a solution to the first question, but there still was some risk that it might fail at that very early stage and all ideas and theories we could have written down would be needless. Thus the report first focuses on the development work which shows the steps that have been taken from the original idea towards the current application. Afterwards, it introduces the theories behind the steps taken. The order allows the reader to see the theory in relation to the work done for the thesis. Using a different order could have cause the situations which would have meant unnecessary work for the reader. If the first question would not have been possible to prove, he/she would have been required to read a lot of theory and afterwards figure out that everything is just theory and we failed to prove its usability. The second situation would have been that we explain the theory without any relation to our work. It would have been possible, but in our opinion, giving the reader a real world example to relate, is a better option, as it helps understanding the theories and doesn’t require to explain the relation to our setup a second time after the Industry Work Simulator (IWS) has been explained. Nevertheless, we added readers guide at the end of this introduction, to help the reader to find the relevant chapter.
Another question that broadens the context of this project, is to find ways of further improving work environment using the Industry Work Simulator (IWS) and our new FIM application at TEschnology and Learning Lab (TELL). To actively change the environment in which the research is conducted is one of the characteristics of action research. This specific kind of IWS is supposed to be used at grammar or vocational schools to possibly bridge the gap between classroom learning and an industry work environment. It is intended to help pupils to participate in simulations, understand a work environment and finally suggest possible improvements helping industries to implement these new ideas as stated in [15]. The current IWS is a set of machines as shown in figure 1.1 and a computer to control the machines.

The idea originally tested is that pupils or students participate in the Industry Live game at TELL. They use the IWS to act as in a real industry environment where they monitor the machines while taking notes of the occurring events. In order to cause a fault, a person supporting the game, uses the controller to inject faults. As an important feature of the simulation, a video camera records the actions of the people involved in the game, providing the possibility to review and analyze the gathered data afterwards.

![Figure 1.1: A picture of the TELL lab](image)

We assumed that enhancing the existing IWS does not concern the existing machine but improving the fault injection feature. To achieve this a new fault injecting application had to be developed with more features to improve the Industry Live game, which led to the first research question mentioned above. Structuring the project in two questions required to be answered sequentially together with the possibility to interact with open source communities enabled an iterative development style. This allowed to react quickly on ongoing changes both in the open source area as well as changes in the TELL lab, which might make both be induced by outside events as well as our activities during the research.
To facilitate easy reading, the following guide can be used to find one's way through the text. As mentioned above, the first chapters 2-5 present the practical parts, followed by a review of the theories in chapter 6 and 7. Chapter 8 and 9 finally draw a conclusion and point to areas and ideas for further research.

Chapter 2: The Industry Work Simulator (IWS)
In this chapter the details of the IWS setup are explained. These details are rather technical and can be skipped.

Chapter 3: The Industry Live Game
In this chapter, the industry simulation game is explained and also the idea, rules and roles played by the participants is discussed.

Chapter 4: The FIM
In this chapter, the FIM application development stages and working is presented. These are again rather technical details with programming and design concepts, and can be skipped.

Chapter 5: Review of the 4th Industry Live Game
In this chapter, we present our experiences from the 4th Industry live game conducted at TELL and the ideas that emerged from it. The enhancement of the output data from the FIM during different stages of its development are also presented.

Chapter 6: Work simulation in relation to HCI
Vikas Sidhavatula presents the implications of this thesis work in the fields of HCI and CSCW.

Chapter 7: Ubiquitous control of the game
Per Pascal Grube presents the implications of this thesis work in the field of Ubiquitous computing.

Chapter 8: The authors’ mutual conclusion
The conclusions and achievements of this project work are presented here.

Chapter 9: Future enhancements
Here we discuss the enhancements for the future which weren’t possible to be implemented during this thesis work due to the limited time available.
2 The Industry Work Simulator

In the following chapter, the setup used in the TELL is described. The first part will present the hardware used to conduct the experiments. It will introduce the two components representing the industrial workplace as well as the off the shelf parts used to control, interact and observe the workplace. The second part will focus on the software used to implement the controller software.

2.1 Hardware

The required hardware can be split into two main parts: the industrial workplace and off the shelf components. The different components are described in the following sections.

2.1.1 Industrial machines

The workplace consists of two different machines. The first one simulates a system making different kinds of cylinders. The second one is then used to check the quality of the “produced” cylinders and sort them. Here onwards a “produced cylinder” is referred as “bit” and cylinder means the air cylinder which allows movement to avoid confusion between them.

First Machine: Bit Production

This machine represents an environment where a PID controller is used to keep a fluid at a certain level. A schematic overview is given in figure 2.1.

The following parts can be found in the setup:

1 a water tube
2 a tube with the possibility to change the flow
3 a valve to control the water flow
4 a connection from the PID to valve 3
5 a sensor to measure the water level in the tube
6a,b connections from the level sensor to the tube
7 connection from the flow sensor to the PID with the possibility to modify the signal
8 the PID controller
connection from the computer to the switchable resistor in connection 7

The PID controller(8) in this setup is used to keep the water level in tube (1) at a constant value. The height of the water is dependent on the amount of water flowing in or out of the tube. The way to change the amount flowing into the tube is through the controllable valve (3), which is connected to the PID controller via wire (4). The last component required to have a functioning PID controller is the input from a sensor, in this case the level sensor (5). It provides the PID controller with a signal representing the height. For simulating a fault the connection (7) from the sensor to the PID can be modified by adding an additional resistor and thus changing the current. This additional resistor is one possible fault that can be managed by the controller card in the computer.

Second Machine: Autoportal

The second part of the workplace is the pneumatic portal system used to check the height of the bits and then sort them according to their material and color. Figure 2.2 gives a schematic birdeye view on the portal itself. It only contains the main parts. The pneumatic valves as well as the PLC used to control are not shown, as they do not provide any additional information at this point.

The main movable parts are those to control the movement of the portal in the three axes, which are labeled 1A, 2A and 3A. They are used to move the bit in x, y and z-direction. The actual position is controlled by the corresponding sensors 1S1-1S3, 2S1-2S3 and 3S1. Those sensors only allow to recognize if the portal is at one of the sensors and don't provide any relative position to the sensors. But for this setup this information is sufficient as the following description of a bit on its way though the portal will show.
When the portal is turned on, the PLC will move the portal arm above the feeder, where the bit is put by the operator of the system. Thus the suction device (presented as the black square at 3A in figure 2.2) will be able to pick up the bit. To reach this point the arm will move in x-direction until until 1S1 is triggered, in y-direction, until 2S1. There it will remain until sensor 6S1 is triggered, which happens if a bit is inserted into the feeder. Then the suction device will move in z-direction using the pneumatic cylinder 3A until 3S1 tells that its at its lower position. Then the bit will be picked up. The next step is to move to the height measurement unit. This required moving 3A to it upper position, moving in y-direction until 2S2 is triggered, moving 3A to its lower position and finally releasing the bit from the suction device.

Releasing the bit will trigger sensor 4S1, which is responsible for recognizing if a bit is in the measurement unit. This will then cause 4A to push the bit into the unit, where the sensors 5S1 and 5S2 are used to check the height. After this 4A will remove the bit from the measurement unit. The movement of 4A is controlled by the sensor 4S2. Depending
on the outcome of the test, the PLC controlling the Autoportal will either drop the bit into the scrap or continue the sorting process.

Assuming that the bit has passed the test, it will now be moved to the material detection unit. This is done by picking it up with the suction device again, moving 3A to the upper position and moving in y-direction until the sensor 2S3 is triggered. Then the suction device will be lowered again. The three sensors 7S1-7S3(an inductive, a capacitive and an opto-electric sensor) will let the PLC determine if the bit is black, white or metal. Once this is detected, the bit will be moved to its drop off place. Thus the pneumatic cylinder 3A is moved to the upper position again. Afterwards the portal bridge (the part which moves in y and z-direction and hold the suction device) will move in x-direction until 1S3 is triggered. Finally the suction device will move in y-direction until 2S3 (for a metal bit), 2S2 (for white) or 2S1 (for black) are triggered and then release the bit. The Autoportal will then return into its starting position above the feeder and wait for a new bit to be inserted.

In case the bit did not pass the test it will be put into the scrap. For this the suction device mounted on 3A will be moved to the upper position again. Then the portal bridge will move in x-direction until 1S2 is triggered, followed by a movement in y-direction until the sensor 2S1 indicates that the portal has reached its y-position. Then the bit is released by the suction device and the portal will move back to its starting position.

Some of the connection between the sensors and the PLC controlling the pneumatic cylinders can be “broken” by the card installed in the computer and thus producing wrong PLC in-and-out-results. E.g. the wire connecting the sensor 6S1(feeder) and the PLC is broken, it will never realize that a bit should be checked and thus it will not start testing. Appendix B contains a complete list about the possible errors.

2.1.2 PC

The requirements for the fault injection computer depends very much on what features of the control programs are used. During the development of the project we identified the following technical demands:

**text console mode only** the hardware requirements are very low. Any system being able to run Python \(^1\) is enough. Nearly any CPU power is required. Also only very little throughput to the diskdrive is needed.

**gui without video** the requirements are nearly as low as for the text console. The only requirement is that the computer is able to run an Xserver. Anything equal to a Pentium2 is sufficient for this task.

**gui including video** the requirements for this mode are rather high. As the program will encode a video stream in realtime, a lot of CPU power is necessary. Depending on the used video codec and the used resolution, a fast dualcore CPU can be required. It can be stated that in this operation mode, more CPU power cannot do any harm.

---

\(^1\)Python is an object-oriented scripting language
For the harddrive one should consider that 1h of raw dv-stream uses around 12GB of disk space, so a constant write speed of 4MB/s is required.

### 2.1.3 DAQ card

An Advantech PCIU 1762 controller providing 16 digital i/o-channels is used. Any other card providing 16 i/o channel and being supported by the COMEDI interface (see §2.2.2) should also be sufficient. There are currently no requirements on the minimum frequency the card should support.

### 2.1.4 Video camera

To use the Video Mode of the Graphical User Interface (GUI), it is required to get a live video of the workplace environment. This can be either achieved by connecting the camera directly to the computer running the GUI or by providing a realtime video stream stream through the network.

### 2.2 Software

One of the main requirements of this work is that the used software should be available as opensource. This step was taken to provide not only control over the game itself, but also control the software running the game. This provides the possibility to not only change the software in a way one likes, but also look how it works. Another benefit of open source software is that it generally provides fast and direct support by the communities of developers. In [18], Raymond present the benefit of this development method. The section about COMEDI shows the benefits for this project.

#### 2.2.1 Ubuntu

Ubuntu[23] is currently used as an operating system running on the system used to control the game. It was decided to use Ubuntu, because it was already used in the working group on other computers. Any Linux based operating system running a kernel 2.6 and being able to run Python 2.5 is sufficient to run this project.

#### 2.2.2 Comedi

COMEDI(see [5]) package is an open source project providing a unified interface to many DAQ cards. The package consists of three different parts: the drivers for the different card, a realtime extension for the rt-linux kernel and a userspace library. Since there are no critical timings in this project, the realtime extension is not used. The current version available in the Ubuntu package repository cannot be used as it contains a bug in the driver for the used DAQ card. For this project, the used kernel driver and the userspace library have been checked out from the COMEDI CVS repository.

Table 2 in [22], shows that CVS is a very common tool used for a collaborative development in open source project. All the observed open source projects used it. For this
project it is of great value that COMEDI is open source and that direct communication with the developers via the COMEDI mailing list is possible, as it allows to resolve the issues within 24h. At the time of this writing, the latest release is 0.7.75, which was released on 28.10.2007. The driver for this card was fixed on the 21.11.2007, which can be seen on the COMEDI community mailing list.

For this reason, no attempt was made to create a deb-package for ubuntu, even if the current package in the ubuntu package repository is not working. Since no experience was at hand how to create proper packages, it was decided to leave this task open until the next release of the COMEDI kernel driver.

2.2.3 Python

During the initial phase it was decided to go for Python as a programming language to implement the GUI. Looking at section 4 of suggests this step. Another reason for choosing Python over another scripting language like Perl is that the COMEDI project provides bindings for the Python language. Another benefit of Python is the large amount of available components.

2.2.4 GTK and PyGTK

Displaying video requires a graphical interface. This suggests that also the other parts of the application should be implemented using a graphical interface as well. For this application The GIMP Toolkit(GTK+) was used. The reason for choosing this toolkit is that Gnome, the default desktop environment used by Ubuntu also uses it. This has the benefit that all required libraries are already installed and doesn’t require further setup by the user. PyGTK is one example of the many available components for Python.

2.2.5 Gstreamer and PyGST

During the project it became evident that there was a need to also include video data. It was decide to use Gstreamer framework, as it is already used by many applications of the Gnome desktop. This ensures that the requirements needed are available on the system used for development. A more detailed description about Gstreamer and the way it is used follows in chapter 4

\textsuperscript{2}deb is the extension of the Debian software package format and the most often used name for such binary packages.
3 The Industry Live Game

The Industry Live game is a simulation of a work environment in an industry to study interaction between workers. The participants of the game could be anyone e.g: students of industrial economy at higher education, pupils from grammar or vocational schools, etc. It is conducted at the TELL lab. The two machines represent the production environment with one being the manufacturing and the other being the sorting unit. The game facilitates interaction among the players while it also provides data about the imagined production work.

3.1 Idea

The idea of the game is to involve students in simulating work to let them understand the work environment as well as to get and archive the information that could be useful for the next following work shift in finding faults from the machines. From this game an idea is obtained about what data could be useful for all the people in the game and also the data that could be useful in developing a better simulation which could help in further research about the interaction in the simulation. It also helps to design a proper training procedures for work environment.

3.2 Working

The two machines involved in the game are actually independent from each other. But as it’s assumed that the first machine which is a PID controller produces a bit for every one minute of stable liquid level at 300 mm. The second machine is the quality control and sorting machine called Autoportal. This machine is fed with a bit and it puts the bit into the height measurement section. If the bit fails the height test it is taken and dropped into the scrap. If it passes the test it is carried on to the material determination section. The material is detected using a combinations of different sensors. Finally it is put in the corresponding holder.

3.3 Rules

There can be two or more teams with three people forming a work shift of which the first hands over the production to the second, etc. One at each shift acts as a white collar worker who is a manager and two are blue collar workers who actually work on the machines as shown in Figure 3.1. It is assumed that for every one minute of stable water height a bit or a small cylinder is produced. And then the workers have to take out a bit and put
it into the sorting and quality control machine. Each shift works for 25 minutes. The next team will arrive in the lab after 20 minutes so that they can get information about the experience with the machines from the previous shift. Managers exchange managerial data and workers exchange the problems faced during their shift and the cause of it. There is also a coach who monitors the entire team and takes the assumptions by the workers about the problem and also gives them hints about what could the fault. Then there is a fault injector who injects the faults through the open source fault injector that has been developed. When the shift suggests the cause of the fault or at least gets close to it the coach asks the fault injector to release the fault.

3.4 Material used in the game

The material used by the students during the game is the archived data from previous sessions of industry live game which includes notes by the participants and from previous shift during the current game. The electric circuit diagrams from the manufacturer, electrical functionality of each component on the machines, general usage manuals to start and stop the entire setup and guidance from the coach is also available during the game.
4 The FIM

As mentioned before, the main task for this project is to develop a program which supports controlling the Industry Live game. Before this project the GUI provided by the manufacture of Digital/Data Acquisition Board (DAQ) card was used. Transferring this GUI to the Linux operating system was the first step towards the design of the current GUI. This chapter can be read as both an account of what we implemented, but hopefully also as a manual for others to learn and follow.

4.1 Program requirements

Figure 4.1 shows the original GUI. Creating something equivalent was the first task to accomplish. This involved several steps to complete.

![Figure 4.1: The GUI to control the card in Windows](image)

4.1.1 Installing COMEDI

As mentioned before it is necessary to checkout the source code from the repository first. This requires the `cvs` command to be installed. Running Ubuntu this can be done by `sudo apt-get install cvs`. Once cvs is installed, it is necessary to checkout the code by issuing the following command into a terminal:

```
cvs -d :pserver:anonymous@cvs.comedi.org:/cvs/comedi login
cvs -d :pserver:anonymous@cvs.comedi.org:/cvs/comedi co comedilib
```
This step will create two new directories: comedi and comedilib. The first is the kernel part containing the drivers for the different cards. The second is the userspace library used to interact with the kernel driver and for writing user applications. To use those parts, one needs to install them. As this step required root privileges, it is from now on assumed that a root shell is used, which can be acquired on Ubuntu by running `sudo -i`. The common three commands used for many software packages today, can be used to compile and install comedi and comedilib:

```
./configure
make
make install
```

It should be noted that this step will install comedilib in `/usr/lib` and which could produce problems if one later tries to install the comedilib package from the repository. But as stated before, waiting for a stable release was no option so this step was taken, knowing that it might cause problems later.

### 4.1.2 Setting up the DAQ card

The next step required is to connect the DAQ card to the comedidriver so it can be used. This requires loading the required modules and setting up comedi. For the used Advantech DAQ card, this is done by the following command:

```
modprobe adv_pci_dio
comedi_config /dev/comedi0 pci1762
```

As a final step, the access writes for the device should be changed to access it from a normal user account and not require root privileges. Depending on the used environment, this might require creating groups, etc. Since only one user is using the system, this was simply done by executing:

```
chmod 666 /dev/comedi0
```

For convenience these commands should be added to the `/etc/rc.local` file or the equivalent depending on the distribution to set up the card when the system is started.

### 4.1.3 Using the DAQ card

After these steps, one can use the card. Using the tools provide with comedilib reveal that the card has the following structure through the comedi interface:

Figure 4.2 shows how comedi internally handles cards. As some cards support different types of input and output ports, mixed between digital and analog, the card is split up in subdevices. The used card in the project is split up into three different devices. Each subdevice is then split into different channel, in this case 16 for subdevice 0 and 1 and four channels for subdevice 2.
subdevice 0 this subdevice is connected to the 16 input channel of the card. It can only be used to read data from the card. Reading 1 form the channel means that the channel is on a high value, 0 means the channel is on a low value.

subdevice 1 this subdevice is connected to the 16 output channel of the card. It can be read and written. The channel can be switch between on and off by writing either 1 or 0 to the corresponding channel. The status of a channel can be seen by reading a value from the corresponding channel, which will either return 1 or 0, depending on its state.

subdevice 2 the 4 channel in this subdevice represent the id of the board. The main usage of this subdevice is to identify different cards in case multiple [DAQ] card of the same kind would be installed in the computer. As this was not the case in this project, the existence of this subdevice is ignored.

For convenience, subdevice 0 will from now on be regarded as input device, subdevice 1 as output device.

Comedi and comedilib are both written in C. To actually use comedilib in Python, it is necessary to have some language bindings. Comedilib contains an input file for Simplified
Wrapper and Interface Generator (SWIG) [21], which will then create the languages bindings for python, which is used to develop the application. As the Python interface reflects very much the C API, it is possible to use the documentation provided on the COMEDI website [5].

4.2 General considerations

A basic architecture for creating interactive user applications is to use a Model-View-Controller (MVC) based design. The basic concepts are described in [3]. The basic concept usually involves three parts. A view, the part taking care of displaying the information, a controller receiving and processing input and a model to handle the data.

![Model-View-Controller Architecture](image)

Figure 4.3: Typical Model-View-Controller Architecture

Figure 4.3 shows the basic structure. Solid lines represent associations between the components which are used for directly accessing the associated component. This could be e.g. realized by having a variable. Dashed lines indicate an association realized by some kind of callback mechanism, to inform the associated component about changes. A common way to implement this structure is to use the observer pattern (see [9], chapter 5, Observer). This will become more concrete later during the description of the program, where the different parts become directly visible. As Burbeck states:

Unlike the model, which may be loosely connected to multiple MVC triads, each view is associated with a unique controller and vice versa. [3]

For the Fault Injection and Monitoring (FIM) application, several triads are used sharing one common model. This is presented in figure 4.4. The arrow means that the model is used by the corresponding triad. The model used is very easy. Figure 4.5 shows that data model is modeled closely after the structure used by comedi. Each channel of the card is stored in an own instance of Channel. Currently only the current state is stored in the model. But it is possible to add further functions to the model without any problems as
long as the current interface is maintained. It might be required for further development to keep an internal log of the occurred events.

At the moment of this writing four triads sharing the same model are used.

- a COMEDI triad
- a GUI triad
- a logging triad
- a Video triad

While the first two implement both a controller and a view, the logging and video triad only implement a view. The communication between the different triads is done by writing the information into the model and triggering an update. This is done by using the standard observer model, which closely relates to the MVC pattern.

Figure 4.6 illustrates the relation between the different observers and the subject. In the implementation the original pattern as shown in figure 4.3 has been slightly altered to deal with need to synchronize the three different triads. If the original design would have been chosen, a lot of synchronization code would have been added to the model and thus making it more complicated to adapt for future development. To finally implement the notification of the observers, the Event class from the Python library is used, as it provides a fast and reliable way for notifying objects about changes.
4.3 TUI and GUI

The [MVC] pattern also proved itself very useful when during the different development phases of this project. The development of the two different user interfaces proved that the overhead of implementing the [MVC] pattern is justified.

4.3.1 TUI - The Textmode User Interface

The first version of the program we created only contained a Textmode User Interface [TUI]¹, which is shown in figure 4.7. This interface was modeled after the GUI available for Windows. It was kept very simple, as it was designed to not consume a lot of processing power. It also reflects a very early stage of the development, where the requirements for an advanced version where not very clear. Different ideas existed at this point, from which many were discharged as it can be seen in section 4.7. The section also explains the various reasons for the decisions.

The triad for this MVC system visualized is figure 4.7. The implementation uses the ncurses² bindings provided by Python.

Using ncurses for the implementation provides the possibility to use a component-oriented programming style for the TUI which is common for many modern user interface systems. It provides different elements like menus, panels and windows allowing fast creation of the UI. It also provides methods to handle user input from the mouse and

¹The term Textmode means that no graphical environment like X11 is needed. Everything (even the boxes) is drawn using ascii characters.
²ncurses is a widget system for textmode, similar to the Java Swing Library or the MFC for Windows.
As specified in the pattern the entire processing of input events is done in the class KeyboardHandler. It contains a loop, waiting for input from the keyboard. The current mapping of key events to channels is shown in figure 4.8. For simple applications it is enough to update the TUI information if some user input has occurred. This means that for only showing if a channel is switched or not, it is sufficient to update the TUI if a user presses a key as model used to store would not be updated otherwise. For this application this is obviously not enough, as the model is used in more then one triad. Thus, it must be provided that the information visible in the TUI can be updated independently from the keyboardcontroller. This is achieved by using different threads for each component of the triad, which is realized by simply utilizing the Thread class provided by Python.

The view for the TUI triad is implemented in the class TUI. It uses an own thread so it can update itself whenever it is notice by the model that the data changed. This is done by adding the TUI to FIM.channelEvent. FIM.channelEvent is an instance of the python Event class, which allows to easily implement an observer pattern in the FIM application. Every time something in the model changes all classes registered at FIM.channelEvent get noticed and take the appropriate action. For the TUI this is reading the state of all channels and updating the displayed information accordingly.
CHAPTER 4. THE FIM

4.3.2 GUI - The Graphical User Interface

Using the TUI during the fourth Industry Live games (see chapter 3) revealed that having a logfile as created by the logging triad (see section 4.5) is useful for reviewing the game later but it also has a major drawback. It creates another information artifact which needs to be handled by the people reviewing the game. It would e.g. require the reviewers watching the recorded video and having a piece of paper containing the information at the same time. The pros and cons of this are discussed later in section 7.3.

As the design of the entire program was chosen with extendability in mind, it was decided to implement another GUI using PyGTK. It was necessary to use a graphical interface to be able to display the video during the game. The text mode interface doesn’t provide the possibility to display the video stream and thus caused the entire redesigning of the GUI.

Figure 4.9: Implementation of the MVC components for the GUI
Figure 4.9 shows that our design does not split anymore into the three parts model, view and controller as nicely as it did for the TUI. It might look as a bad design at first, but taking a closer look reveals that this is due to using the PyGTK framework and thus splitting the view and controller are already performed by the framework. This allows to have only a few lines of code to perform the required actions. What might look confusing is that the view part is split up into two different classes. The main reason for this step is to provide a simple implementation. The different parts of the GUI like the buttons are created in the class GUI, which also contains the code for reacting to the different input events from the user. A lot of work is in this case performed by the framework without requiring the user to handle low level events. As the TUI interface is very simple, it was sufficient to only react on certain keyboard events. For dealing with a GUI, this is far more complicated. Reacting on mouse events e.g. requires to check if the user clicked on something clickable in the GUI and in this case react to something. All this tedious work is done by PyGTK and thus it only require to connect e.g. a button with a method that should be executed in case somebody pressed the button. The second class implementing

![Figure 4.10: The GUI without Video](image)

a part of the view is ButtonThread. As figure 4.10 shows the buttons show the current state of the channel. Again the reasons that already applied for the TUI and resulted in the two different threads also apply here. The model is changed without interaction of the user, so only handling input from the user in one thread is not sufficient. Thus changing the view according to the model needs in its own loop to process. It would be possible to use the different threads already provided by the framework to update the view from the model, but it would have required a lot more work and would be harder to change in case of changing requirements.

Adding the video stream is done by adding another thread representing the another view on the model. An in depth presentation of the video processing follows in section 4.6.

### 4.4 The Comedi Triad

Even if it doesn’t look like a part of the application one would implement using the MVC pattern, it should be done this way. The structure provided by comedi as present in section 4.1.3 helps applying this pattern since it allows direct mapping of the view and controller component to the card. Figure 4.11 shows the structure of the implementation. It shows that it can be clearly split up into a controller and a view. The controller ComediHandler is implemented as an endless loop reading the current state of the card and in case of a
change modifying the model, which will in return notify all the observers that it has been updated. *ComediWriter* uses this standard observer pattern using *FIM.channelEvent* to react on channel events and in case the state of channel has been toggle applies the changes through the comedi interface. Thus the *ComediWriter* can be be seen as a limited view on the model, as only parts of the model are presented.

In addition to the real COMEDI interface, a dummy implementation is available by using the *--dummy* commandline switch. This dummy mode simulates input from the controller and discharges any output information. This dummy mode is very usefull during development, as is remove the need of having a [DAQ] card in the computer and thus allows development on the project without being bound to one maschine.

### 4.5 Logging

The previews sections always contained full triads implementing the MVC pattern. The logging mechanism in comedi is also realized by a [MVC] pattern, but it does not contain any controller, as only the current state needs to be recorded. Figure 4.12 shows that

![Diagram](image)

**Figure 4.12: Implementation of the MVC components for Logging**

the structure used is the same as for the implementation of the Comedi triad. The main
difference is is lack of a controller to modify the model. Each time one of the channels change, a line in the form 093343,4,“[16, 0, 0, 0]” is written into the log file. The first number is the time written as hour minute and second, the second is channel that changed. The 4 numbers in the [ ] are the number of the channels as represented by Comedi ($2^{\text{channelNumber}}$) followed by the states of the different subdevices of that channel.

4.6 Video

The last component is the view for video processing. The design can be split up into two parts. One parts acts as a view on the model, reading the status of the channels and creating a textual representation of it. The string consists of 16 0’s or 1’s, depending on the state of the channel. The string 0011 0000 0000 0000 means that there is a signal on channel 2 and 3. The representation of the output channel is generated in the same way. It provides a simple way to display the information in the video. If this representation is suitable in this way is a topic for later research and is not discussed at the moment.

The second part of this triad it the thread processing the video itself. It is not running within the current model as it does not work well with the used event model. The number of events is very low, whereas the video thread needs to be executed very fast to process the data.

The entire video processing in this project is done by GStreamer. It is multimedia-framework suitable for video processing. The core of the framework provides basic elements needed like the pipelines, filters, generic input and output plugins. The far greater part is implemented in plugins, which actually do the work. This opens the framework for a great number of applications, which ranges from simple tools to converting audio file over streaming servers to complete video editing solutions. It provides the user with a large variety of different input and output elements.

In this project it is used to record a video stream from a firewire card, add the two strings containing the state of the channels as well as a time and finally save it one the harddrive. In the following the different steps of processing the data will be shown together with a small introduction to GStreamer, to allow the reader to use this section as a reference to changing this part of the program later.

4.6.1 The basic way of data in GStreamer

The easiest way to think of the data in GStreamer is to compare it with the flow of water. It will come from a source and will disappear in a sink. This means that a minimal GStreamer pipeline must at least consist of those two parts, which is visualized in figure 4.13. The figure shows that in this application two different sinks are used. One for storing the video information on the disk, the second one for showing the video on the screen. To achieve this, GStreamer contains a so called tee element, which has one sink and an unlimited number of sources, which allows the FIM application to view the video at the same time as saving to the disk. This situation is presented in figure 4.13. This setup allows to display the video at the same time as it is recorded to the hard disk.
4.6.2 Muxing and Demuxing of the Video stream

The last sentence is in fact not totally correct, as it simplifies the process too much, but it is due to the very rough view on GStreamer until now. Taking a closer look at the stream reveals that it is made up of two parts: the video signal and the audio signal. To transfer those two parts together they are usually combined together in a process called muxing. If this step would be left out, one would be e.g. required to have two files to watch a movie instead of just one .avi file containing both.

So the first step to work with such video stream received from the camera is to demux it and thus retrieve both parts as separated streams of information. This step is not required to actually save the stream onto the disk file, but it is required to view the video signal on the screen. This is due to the fact that pads (the black square and round elements in figure 4.14) usually implement some kind of filter and only accept some kind of information, as it would be e.g. pretty useless to send an audio signal to a screen.

The required plugins for demux e.g. the signal received from a video camera connected via IEEE1394 (Firewire,iLink) are included in the GStreamer framework. Figure 4.15 shows what signals are available after demuxing the video stream. For just presenting the video and audio data this step is sufficient, but as mentioned earlier, it is not suitable for storing the information on the disk. The two signals(audio and video) would each be stored in a different file and this resulting in two information artifacts. As this is not advisable, the information artifacts must be combined in a processes called muxing. This step does the opposite of the previous demuxing step. It combines the audio and video
signal into one stream again, which can then be stored in one file. Figure 4.16 then shows
the structure of the system up to this point. This setup now allows to transform the
different signals individually. Each signal can be transformed as adding text to the video
signal, resizing the video signal or compressing the signals for storing them on the disk.

Since adding more elements to the current diagrams would only add more complexity
and spoil the effect of providing a fast overview, only parts of it will be shown. A complete
overview can be found in figure A. The diagrams will always start and end some elements
that are contained in figure 4.16 and thus allowing the reader to easily figure out where
the components are situated.

4.6.3 Decoding and Encoding the Video Stream

One common step in video processing is to transcode the video and audio signals from one
format into another. Usually the incoming signal is encoded in some format that cannot
be directly used and thus it must be decoded before using it. The outgoing signal is then
usually encoded again in some format, usually in a format that compresses the signal to
save disk space or bandwidth because saving audio e.g. in an uncompressed format like
wav would just be waste as it could as well be compressed to mp3 without much loss of
information.

Another reason for transcoding the video stream might be to make it available to a
broader audience. The signal received from the camera uses the \textit{dv} stream format. It is possible to save it directly to the harddisk without transcoding it, but it would limit the usage, as it is not widely supported. Since the video should also be used for reviewing e.g. the participants of the game, it is not possible to force everybody to install a corresponding software for their operating system. For this reason it is necessary to convert into a common format like \textit{avi}.

**Decoding the Stream**

The video stream received from the camera contains two parts which need to be treated separately after demuxing the stream. The audio signal is encoded as a raw audio stream already and thus doesn’t need to be transformed into anything before it can be used e.g. to play it back on the speakers or to record it as a \textit{wav}-file. The video signal is encoded as a \textit{dv}-video stream and thus needs to decode to raw image frames before it can be further used. This requires to add another element in the processing pipeline. Figure 4.17 shows the pipeline with the inserted decoder. The resulting audio and video signal could then be directly transferred to the audio and video sink.

**Encoding the Stream**

It is also possible to directly mux those two signals and send them to a disk sink, but it would result in huge files and thus waste a lot of disk space. Also, it would not be possible
to directly play such files with standard media player. To achieve this it is necessary to add an audio and video encoder to the pipeline, which is shown in figure 4.18. The GStreamer framework contains a lot of different audio and video encoders, which could be used. Tests have been done with different audio and video codecs. The following list shows the varying results and gives possible uses of the used combinations.

<table>
<thead>
<tr>
<th>A/V-Codec</th>
<th>Results and usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>raw/raw</td>
<td>This combination doesn’t apply any encoding on the audio and video signal. The</td>
</tr>
<tr>
<td></td>
<td>main benefit is that no information is lost by compression and thus it is</td>
</tr>
<tr>
<td></td>
<td>useful for post processing the file in a later step after the game. It can be</td>
</tr>
<tr>
<td></td>
<td>read by the most video editing tools. It works very well, but the produced files</td>
</tr>
<tr>
<td></td>
<td>are very huge and thus require some transformation later.</td>
</tr>
<tr>
<td>a52/mpeg2</td>
<td>This combination is the used on DVDs. Its main benefit it that it easily</td>
</tr>
<tr>
<td></td>
<td>allows to create standard DVDs which can also be used on standalone DVD players.</td>
</tr>
<tr>
<td></td>
<td>It works very well and depending on the later usage it might be very useful for</td>
</tr>
<tr>
<td></td>
<td>archiving also.</td>
</tr>
</tbody>
</table>
This is the currently most used format to store movies on computers. It provides a good trade-off between quality and size. The resulting files are playable on every computer and supported by many modern standalone players. Depending on the available CPU this combination might cause some problems. High compression rates at full size of the video might cause some problems. The available system (a small core2 duo CPU) was able to perform this step without any problems, but already causes a system load of nearly 1.0.

Both are free open source high quality codecs. The main benefit is that they provide the same quality as the divx/mp3 combination at a lower bitrate and thus requiring less disk space afterwards. The drawback is that it is not directly viewable on some systems and thus might require some 3rd party video player like VLC[24]. The higher compression ratio comes along with a higher CPU usage and thus might cause problems depending on the available hardware.

h264 is a standard high performance codec for video compression, providing even better compression than theora. The main problem of this codec is that the available performance of the used system wasn’t high enough to allow real-time compression, and thus it was not usable for further testing. One possible step would be to reduce the size of the video, but as the video should be archived this option has not been considered. One possible solution would be to save the video as raw/raw-stream and convert after the game.

The current version of the software uses the vorbis/theora combination to store the video.

4.6.4 Adding the channel status

The last step is to add the channel information to the video signal. Using this text string might not be the best option, but it was chosen during this stage of development because the GStreamer framework provides a module for adding text on the video stream, thus making development a lot easier than trying to write a new overlay implementation. The section about unrealized ideas (see 4.7) gives an idea of how it might look if it would be implemented.

Adding the text to the video signal is simply adding another plugin to the pipeline. Figure 4.19 shows the part of the pipeline where the plugins are added. It is required to use two TextOverlay plugins to add two different lines of text. The current status of the channel is added on the top of the video, the set channel on the bottom.

4.6.5 Video Design Conclusion

The design of GStreamer and the availability of the plugins proved that it was good choice to use this framework. The implementation in FIM actually uses some more plugins like
Figure 4.19: Pipeline with text and clock overlay

queues, videoscalers, etc which are not discussed here. Their documentation can be found on the project website (see [10]). They are only required to construct the pipeline without adding additional functionality. Figure A contains all the used elements together with the names of the their variable names of the program to facilitate the later maintenance and enhancement of the application. Figure 4.20 finally shows the application running with video enabled.

Figure 4.20: GUI with video of FIM

4.7 Unrealized Ideas

During the initial design phase of the software different ideas have been thought of. Some of them are collected in this section to show and argue why they were not realized.
4.7.1 16-line wave diagram

One of the first ideas after implementing the TUI version of the application was to create a GUI which would allow to see the state of the different channels. This idea originates before participating in the game. Figure 4.21 shows a sketch of the GUI. The main idea behind is that this kind of diagram gives the operator maximum control over the game as it displays all the information that is available to the computer. Without providing a video this GUI might be very useful. At the moment there is always an operator present in the room. Considering that this must not always be the case, it will be required to give the person controlling the game from a remote machine some kind of feedback and it might not be possible to transmit the video in real-time. For this the graph will provide a better overview of the current state of the machine than the buttons which only show the current state without any relation to time.

Figure 4.21: Unrealized gui 1

4.7.2 Statemachine representation

Figure 4.22 shows the second idea of a GUI implementation. The idea is to separate the way of a bit in different phases. Start would be to turn on Autoportal without any inserted bits. If the user would place a bit in the pickup position, it would enter phase 2, etc. This GUI would be a different representation of the state of the machine compared to the previous one. The pros and cons are the same.
4.7.3 Graphical (and Clickable) overlay plugin

As mentioned earlier it was decided to implement a GUI version that shows the video with additional information about the different channels. Despite time constraints limiting the possibility to exploit this concept any further, it was realized that the current presentation of the status of the channels is a subject for further research. One idea is to integrate the status of the buttons into the video as graphical representation. In a second step this could even be enhanced by defining clickable areas in the video and thus removing the need of having the extra buttons to change the state of a channel all together. The main problem of this approach was the need to implement a plugin to display the information in the video. Another problem might be caused by the required CPU power to process the video and the buttons.

Also different other information about the game might be presented in the video during runtime. One idea is to show when a fault is set or released. Looking at the broadcast of a soccer game on TV can give an hint how it could be realized. When a player scores a goal, additional information like the number of goals in the current season etc are displayed. This concept could be used to give additional information about the set fault because the information like “fault 4 is set” is not very useful for the participants of the game and thus won’t be of much help if the video is reviewed later.
Figure 4.23: The GUI with buttons in the video
5 Playing The Game

During our participation as fault injectors in the fourth industry live game with our new open source TUI, we tried to study the usability of it. The new application creates a log file containing the time stamps when a fault was injected. This would assist researchers and students/pupils trying to study the video data generated during the game because they can now know exactly when a fault is injected and can see its effects in the video exactly with a precision of seconds. When the researcher analyzes the data generated during the game, he has to study the video, the log file and the notes by the participants. But somewhere it was felt that more can be done to this setup than just an automatically created log file. It’s observed that there could be something that can be done with the video and then an idea occurred about embedding the channel active information and the fault injected information into the video on the fly. This started our study about how to embed the data into the video. As the current version of our software proves, we have successfully embedded the data on the video creating a better form of data available to the researchers.

5.1 Recorded Data

The recorded data has changed during the different stages of development. After the first developed TUI we had the log file shown in figure 5.1 recorded video and the notes of the participants as the recorded data. After our development of the GUI in which an injected fault/error data entry is embedded into the video, different data is positioned together in the recorded video as shown in figure 5.2.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>93333</td>
<td>3</td>
<td>[8, 0, 0, 0]</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>93333</td>
<td>11</td>
<td>[2048, 0, 0, 0]</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>93334</td>
<td>3</td>
<td>[8, 0, 0, 0]</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>93334</td>
<td>6</td>
<td>[64, 0, 0, 0]</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>93334</td>
<td>11</td>
<td>[2048, 0, 0, 0]</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>93334</td>
<td>3</td>
<td>[8, 0, 0, 0]</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>93334</td>
<td>6</td>
<td>[0, 0, 0, 0]</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>93334</td>
<td>11</td>
<td>[2048, 0, 0, 0]</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>93334</td>
<td>3</td>
<td>[8, 0, 0, 0]</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>93334</td>
<td>7</td>
<td>[128, 0, 0, 0]</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.1: View of the log file in Excel
5.2 Reviewing Previous GUI Ideas

When we try to review our previous TUI idea of an open source application which only had channel active information and the fault injected information, the current one has lot more data at hand in the form of video with embedded log. So the focus of the subjects playing the game is not diverted onto too many things and almost all the necessary data is available on one screen which realizes a very good principle of usability.

5.3 Combining Video and DAQ board Data

The new idea of combining the video and the data obtained from the DAQ board seemed promising. This enhance version of the FIM application has both, the important data on one screen facilitating much more convenience and also simplifying the analysis work of the researcher. The FIM application has buttons with multiple functions, showing the activation and deactivation of the channels of the DAQ board with green and red colour, showing the status of a channel if an error has been induced on that particular channel and also when one of them is clicked that particular channel becomes active or inactive. This change in information is reflected in the video with two rows of data, one corresponding to the channel activation and deactivation and the other corresponds to the fault injected.

According to Harris:

> Video allows repeated review of complex and fleeting events (in both real world and laboratory settings) at varying levels of granularity, bringing opportunity to check and amend interpretations of observed user actions in the light of data from other sources (log files, interviews, verbal protocols, etc.) or new analytic insights. [11]
Thus adding video to the data obtained after the industry live game eases repeated reviewing of the data.Possibly this facility also opens up for the combination of some subjects playing the game on the lab site and others co-operating on distance using network and streaming video.
The FIM we have developed for this thesis work, involves several HCI issues such as usability, human interaction and CSCW. Based on the experiences during the development and experimentation, it has also proved some of the above concepts as valid. A model of interaction between different persons and objects in the game is shown in the figure 6.1. Interaction between the computer and fault injector is basically for using the FIM (during simulation FIM is accessible only to the fault injector role). Usability concepts involved in this interaction are presented in section 6.1. Human interaction between the workers is presented in section 6.2. Interaction between the workers and the machine models is also discussed in section 6.2.

Figure 6.1: A model of interaction in the game.

6.1 Usability concepts of HCI

The idea of FIM itself was to make an application which could better support simulation experiments and researchers during analysis work. The last aspect of the idea correlates with the first principle of usability in being able to do things using an appropriate amount of effort as described in [1]. Although there is no measure for the appropriate amount of effort, but according to our perceptions it clearly indicates that the new FIM would allow for faster analysis of the data available.

The next thing we have tried to achieve was to place all the necessary functionality on one screen in an understandable manner. This idea also correlates with another usability...
principle that all the appropriate functions and information is available to the user in an understandable manner. It is also very easy to use and easy to remember how to use it even after sometime of not using the FIM as stated in [1] because the user interface is pretty simple with not many pages of menu and buttons. It only has one button representing the each channel number which is also displayed on the buttons. All that is needed is just a click of a button to inject a fault on a particular channel. Although it is not tested out in live experiment but during the development and testing process, it has shown considerable amount of ease to operate and understand.

### 6.2 Human interaction

During the game we have made an effort to observe and understand the trends of interaction. The rules of the game are very important to promote interaction. We have seen that the rule of two shifts overlapping for five minutes adds better practical knowledge to the next shift. This was noticed when we injected the same fault injected during the previous shift and the current shift quickly identified the fault due to the knowledge gained from the previous ones. This trend can also be used in an industry situation where, for a small amount of time the people from both the shifts interact with each other and share their experiences. This can prepare the workers to be ready for a particular problem and builds a more capable community that makes their job of finding any fault easier. So the current worker can use the experiences of the previous shift people. Figure 6.2 shows the students of one shift exchanging their experiences with the students of the next shift. This concept has been tested and has been proved valid during the fourth Industry Live game.

![Figure 6.2: Students exchanging their knowledge during the overlapped time.](image)
The environment of this IWS, the setup, the rules of the game and the FIM are designed to enforce better interaction between the pupils/students playing the game. One of the main aims of this setup is to study the interaction between them. The setup with models of industrial machines gives an impression of a real-time industry to the participants and while monitoring these machines the participants interact to understand and notice the events happening. In the handing over process, talk is supported by taken notes and by direct references to details of the machinery. They also use the information artifacts available to them such as the manuals, archives, etc. together with physical artifacts such as labels on the machine components, etc. to interact and find the cause of the fault. In a similar interaction as described in [20], a civil engineer describes her activities in a project to replace a bridge using the computer screen, paper graphs, maps and also she uses her hand to show a particular angle of a road pointing with her pencil to a particular point to properly communicate her vision to the author. According to [27] most of the work place interactions between two people usually involve information artifacts such as documents, reports, notes, archives, etc. Also in [2], the author writes of the interaction between the captain and the pilot of the ship, where they use different information artifacts such as the radar, sea-map and the view out of the front pane to confirm the identity of foreign ships. Similarly, the workers in the game also use information and physical artifacts by pointing to them using a pen or their finger as seen in figure 6.3 to communicate their ideas to each other while trying to find the fault. All this proves that the set-up of the Industry Work Simulator is successful in implementing the best concepts of interaction. Thus giving a good understanding of the interaction principles.

Figure 6.3: Pointing to information artifacts.

The interaction between the machine models and the workers is important because the
machines should be able to communicate to the workers about a fault occurrence. If a PID controller is considered, it is very important to get attention of the worker when a fault is injected. In the case of the Autoportal where a fault causes it to either not work or work irregularly, is not possible in a PID controller because when a fault is injected the water level rises and comes back after a period of time to normal because of the extensive compensation power in the mechanism. The placing of the PID controller is very important so that it can easily catch the attention of the worker, who in turn notes down the fault and tries to find the cause or reports it. This could avoid unexpected failure of the machines in a real industry scenario. In [13], the author explains similar situation with an experiment on the primary flight display on an Airbus A320. He mentions that it is very important to let the pilot know that an error occurred while in auto-pilot mode, through a signal in the flight display and it is also equally important to make him understand the meaning of this signal clearly and quickly so that he immediately tries to correct it. Thus avoiding greater problems by better interaction between the machines and the humans.

The different kinds of interaction, between human-human ans human-machine respectively, are both prominent side by side within the IWS simulating work. All these factors together make the interaction between the students/pupils very interesting, challenging and tempting in terms of learning and developing a work environment through interaction in the IWS. Thus also satisfying the principles of interaction and confirming some of the existing principles.

6.3 Computer Supported Co-operative Work (CSCW)

During our development of the FIM we used and participated in bug forums, Subversion (SVN), etc. and realized the ideas of distributed CSCW. SVN is used for co-ordinating the work done by the two authors. This software as described in [4] is a version control system that is used to maintain current and historical versions of source code, web pages, documentation, reports, etc. This I find a very useful tool when two or more people are working on a project. It helps a lot in team management when all the people are editing the same resource. It also merges data from two authors, helps resolve conflicts, and maintains a revertible record of modifications and additions. At any point of time it can be reverted to a previous version. This is to me a very good example of distributed CSCW in our masters thesis work. Also as can be seen in figures 6.4 to 6.6 other people such as thesis advisors have continuous access to the accumulated versions/revisions of documents.

Although the ideas of CSCW are originally addressing office work, when SVN is used to co-operate during the thesis work it is actually showing that computers do support co-operative work in a distributed environment. Even though the team is not located under one roof they are able to connect to a central computer located in the lab using Secure SHell (SSH) connections and view, add or modify data. Using a SSH client and a Xserver on the local computer, anyone in the team can even run certain applications on the server and check the results. Thus using a computer in the lab to co-operate during the work is on the same lines of CSCW except that it is in a distributed environment. This also
confirms that the concept of CSCW is valid and also been successfully tested during this thesis work.

The bug forums such as COMEDI mailing lists we have participated in, is also a kind of CSCW in distributed environment. When the DAQ card was not working as expected due to some bug, we have written a mail in the COMEDI mailing list for help (see figure 6.7). It was surprising that within a day we got replies from many people trying to help us and one among them has given a patch to the COMEDI package, after this the problem was fixed. This gives an idea of the efficiency of distributed CSCW through mailings lists.

We also want to contribute to the open-source communities that have helped us during our thesis work by making our work available under GNU General Public License (GPL) (see in [8]). This allows others to use our work for further development and freely share it. We have put up our work which includes the source code and the report at \url{http://tellstud.tekproj.bth.se/} for public access.

The whole setup of the IWS at TELL is also promoting CSCW by giving the opportunity, on still an experimental level, to train pupils, students and workers efficiently and also making them understand how to co-operate and work together. Further, the enhanced IWS and Industry live will open up for e.g: teachers and pupils at vocational education investigating new forms of learning. They will not only be able to play the game (with many interaction features) but also, with thanks to open source software, be able to further develop and change the FIM application. The participants of the game and the teachers who are actually users of the FIM application may also come up with new ideas and become the designers/developers. Also, the ability to reflect on the game with the use of
Figure 6.5: Archive of different versions of the report.

video recordings, logs, notes, etc. makes way for more research.
Figure 6.6: A view of the report online.

Figure 6.7: Communication with the COMEDI mailing list.
7 Ubiquitous Control of the Game

As pointed out earlier, understanding the ongoing activities in today’s workplaces plays an important role. Looking at studies of different workplaces for instance wastewater plants and the bridge of a ship [2] point out that a deeper understanding is required to take proper decisions both in use and design of computer systems. In [2], this means that one always tries to achieve the “Ultimate goal”. In their examples these were complex tasks like running a wastewater plant. But even the simple setup of the IWS allows to apply this concept.

7.1 Towards the Ultimate Goal

The goal of this game can be fairly easy defined as producing as many bits as possible. If the machine would run without any introduced errors, it would be 60 bits a hour. As this would be desirable for a production environment, one objective of this lab is to obtain more information on how people work together in case errors occur. For this reason FIM can be used. Assuming that normally the operators would more or less watch the water flow and Autoportal work, this radically changes in case an error appears, where the component that is suspected to have failed draws the main attention. This effect is described in [2] as the shift between different levels of abstraction of the workplace (Tx refers to an error appearing in Microsoft Word):

In the example of level shift given earlier, Tx refers to more than one style, is an annoying interruption that made the level shift very evident.

As one usually figures out after some times running into the same error how to work around it. In the case of the IWS this would mean to know that different errors occur because of e.g. a defect sensor. Thus more knowledgeable workers would be able to produce more bits and this sticks more to the ultimate goal than trying to fix a single fault.

One of the benefits of the IWS is that it allows to control this shift awareness between the levels of work by means of FIM. Having this simulated environment also allows to validate this by giving the same tasks to different groups. In a company this is rather undesirable as a not working machine only costs money and thus doesn’t allow any experiments in opposition to the lab. A company’s main interest is to transfer the knowledge of one worker to the next one.

As presented in [14] the physical proximity plays an important role for research. Their main research is conducted in collaboration with scientific research and states that:

We believe that scientific collaboration provides a model of the way professionals in many fields construct research collaboration may help to specify the
Looking at figure 7.1 shows that this is also true for the IWS game. It is important for the two workers to have a view at the same machine. This helps both to easily transfer their knowledge and resolve the problem. Looking at figure 6.2 also shows that this is important as the takeovers take place at the IWS and not e.g. outside the lab where it would be more quiet. This finding fits nicely with table 1 in [14] where they show that the communication between collaborators is higher if they have the possibility of informal conversation. It would be possible for the two teams to just hand over the system without reporting about the occurred errors or to hand it over at a different place and as such not having the possibility to point e.g. at a sensor that failed during the game.

The importance of having the possibility to use different sources of information and juxtaposing them is also discussed in [2]. Having the possibility to use the radar of the ship and looking out of the front window provides to use the two sources at the same time. This possibility becomes even more important when the captain and the helmsman are steering the ship. Another interesting point stated in [2] is that:

When things get hot in trafficked waters, the helmsman is routinely called in to handle the wheel while the captain takes care of the machine telegraph (cf. the example in Section 4.2), whereas in open waters with little traffic one person controls a 350-m ship. [2]

This situation is easily compared to the IWS. When everything is running fine, one operator is enough to overlook both machines at the same time. In case something breaks, this
will be not be possible anymore. Referring to the idea of “levied design” in [2], it shows that the error will draw full attention of at least one of the operators, but the second one will still be able to overlook e.g. the other machine. At the same time, the physical proximity of the two machines allows the two operators to work together.

7.2 Making FIM interactive

Currently FIM does not do anything on its own without somebody interfering. The actions to keep the operator busy are provided by the person operating FIM. At the moment FIM just acts as an interface between the DAQ card and the operator. It would be possible to enable FIM to introduce errors by itself. The current design of the software allows to implement this feature in a rather simple way, but it has two major implications which need to be addressed.

1. make FIM ware of what is happening
2. make the operator aware of what FIM does

As the second issue is easier to solve, it will be addressed first.

Currently either the coach of the teams or the operator of FIM will issue the errors. Due to the physical proximity it will be possible for the coach to always know what will happen next. In case an operator issued the error he can easily inform the coach. If FIM introduces these, this will not be possible anymore. It would require more attention from either coach or operator to watch the display to stay updated. A similar kind of problem is discussed in [13], where the pilot needs to be made aware of the actions of an autopilot. Despite the problem to make the user aware of some system, the situation in TELL is more complicated. In the cockpit of the plane it is o.k if everybody is aware of the autopilot. It might even be beneficial. In the lab this effect is undesirable as e.g. a noise indicating an inserted error would also be realized by the participants and thus cause them to pay extra attention. This would then spoil the idea of having a good simulation because in a production environment machines will not inform the user that they will cause any problems.

One possible solution to this problem would be to change the current display of the application to make it more informative for the coach. He will not use the video output anyway, as it is much more convenient to watch the game in real than on a small screen. A second option would be to add another triad presenting the information in a way suitable for usage by the coach. One possible realization would be to add a second display showing the information. This could be kept very simple e.g. by a small LCD display just showing the number of the error attached to the USB port of the computer. A second option is to provide a wireless headset to the coach and provide information. Both options have their pros and cons. The realization is possible in both case since the Linux kernel supports bluetooth headsets as well as USB attached LCD displays. The LCD display will always show the information to the coach and thus removing the burden from the coach to remember the current error. But the information is fixed to location of the display. Using a wireless headset like the ones commonly used for mobil phones would resolve
this problem but on the other hand requires the coach to remember the messages and thus draws much attention from his original task, coaching the participants. Maybe a combination will provide a proper solution. Discovering this proposes more experiments for the future.

To conduct these experiments, one major task has to be done first: making FIM aware of what is going on. As the name of the application shows one of its main tasks is to show the information. This is done by presenting the current information, but it does not require any interpretation on the same. It would of course be possible to just inject random faults. But this could lead to undesirable results as it would be unpredictable when it would occur. To avoid this it is required to make FIM aware what is happening.

The first step would be to recognize if there is a bit in the Autoportal. This step is rather easy to achieve as there is a sensor which recognizes when a bit is put into the feeder. It is also possible to determine when the bit is dropped from the combination of different sensors. This would at least permit not to introduce faults while a bit is being checked.

Using the different sensors allows to implement a state machine, allowing FIM to always know what is happening. This would allow e.g. to implement the checking which error code to be inserted in the different states using the state pattern (see §5, chapter 5, State). Using this pattern removes the problem of not knowing when to insert which error. It also allows to insert more interesting errors like moving every second bit to scrap, even if it is o.k. But still FIM only knows what is going on by the sensors.

One possibility to make FIM even more aware of what is going on in the game would be to make FIM know what kind of bit is inserted. For a human watching the game discovering if the bit is black, white or metal is done when one sees it. This information is not available to FIM until the bit has passed the material detection. The same goes for height measurement. After some time a coach can see by just looking at the bit if its good or not. For FIM this information is only available after the height measurement.

One possibility to overcome this problem is by using RFID. It would require to put a tag on every bit and a RFID reader at the feeder of the Autoportal. This would allow FIM to take decisions even before starting. It would allow to create more advanced scenarios like only sorting white bits wrong. From a technical view, adding this functionality is rather easy by adding a triad controlling a RFID reader.

A second possibility would be by adding e.g. a barcode on top of the bits and adding a barcode scanner at the feeder. Even if that would make it more obvious to the people playing the game that some information is gathered it has the benefit that it relies on everyday technology. Barcodes are used in our everyday life at the grocery stores to identify items. With the upcoming of the self-scanning service like the ones shown in figure 7.2 the users are coming into direct contact with this technology. Also barcodes are commonly used to track other information like packages at the postoffice, which shows that it had proved its usability. So maybe it won’t interfere with the game at all, because as Marc Weiser stated

\[ \text{The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.} \]
Thus, using an obvious technology that everybody is used to might even provide some benefit.

But before trying to answer this question, another one should be answered: is having more information about the game useful? The first step before adding another technology to this project needs to be an evaluation if the game play could be enhanced even further by just making FIM aware of the situation. Just having a lot more possibilities does not mean having more or better results. It even might make impossible to identify anything, as to many different control sources will create too much noise or even cause reactions by the participants that will cover the ones that one would like to discover.

Another reason against adding more technology is the possibility of more errors. As Dijkstra stated

> Program testing can be used to show the presence of bugs, but never to show their absence![^7]

The project hasn’t shown yet that it has successfully used the input data from the already available sensors, so adding more sensors might cause more problems, especially using upcoming technology like RFID. If there occurs an error while reading e.g. the id of the bit, the Autoportal should still be able to function normally.

### 7.3 Information artifacts

In this project several different pieces of information are created: log files, video recordings, etc. The previous section introduced even more artifacts like the status of FIM and the
corresponding actions. One problem of creating more and more artifacts is that they need to be made available in a suitable way. Also one important artifact is missing. As Heath and Luff discovered in their studies of the London Underground [12], it is important to overhear the actions of the collaborators, which is in line with Bødker and Andersen on their studies about the bridge of a ship. Due to the noise of the machine it is currently not possible to capture it. As those interactions are in important point of the game as shown in section 6.2, it might be important to directly capture those artifacts. Again, this leaves room for further research, as it is not yet understood, if this information would provide additional information or if it would just create another artifact which requires more information to be combined and thus add to the overall complexity of the project without any gain.

7.4 Reached the Ultimate Goal?

As pointed out earlier, it is possible to define an ultimate goal for the participants of the game, which, given enough time to train them can easily be achieved. Towards the end of this project, it was possible for us as researchers to directly identify the faults when the system stopped working. But can an ultimate goal also be defined for this thesis project? Looking back to the research questions given in the introduction, this can be done. As for the first question, the ultimate goal was to create FIM, using open source and different sources of information. As the current version provides the required functionality, it can be concluded that this goal has been achieved.

For the second question, the goal is more difficult. Gaining knowledge and improving a workplace environment is nothing that can be as easily evaluated as if a program is properly working. Also, regarding learning as a life long process, this goal is a long term process, without a clear end. But looking back at the previous chapters, it can be concluded that a considerable amount of effort has been put into reaching this goal. The overall design of FIM has been made as flexible as possible to deal with current and upcoming requirements to gain knowledge. In this version, the integration of video and the errors has been done to facilitate reviewing the games and drawing conclusions from it. Section 7.2 above contains further suggestions of what can be done in the future for FIM, depending on the requirements for further development.

Thus, the question if the goals, stated as research questions have been reached cannot finally be answered. As the first question could be answered, trying to answer the second question by enhancing the application only lead to more questions. Not only the question, what knowledge should be gained is not answered, FIM broadens this area and allows more different “how” methods to be used, from which none could be answered due to time constraints. Hopefully the fifth Industry Live game will help to evaluate the currently implemented “how”-methods of video and thus allow the introduction of another iteration of development of FIM and reduce the open questions.
CHAPTER 7. UBIQUITOUS CONTROL OF THE GAME

7.5 Ubiquitous Control

This leaves that final question open for this chapter: What has ubiquitous computing to do with the control of the game? Two major aspects of ubiquitous computing have been and will need to be considered when enhancing FIM: situation awareness and technology becoming imperceptible from the environment. As the first is discussed above, the second goes beyond the issues mentioned above. Using new technologies is one aspect of ubiquitous computing, but they have to merge themselves into the environment. As using them in the IWS might help to discover new possibilities for using technology in a workplace environment, it might as well make it impossible to discover current problems of workplace, as the problems with new technologies will cover the current issues.

Thus, even if it does not appear directly as a major aspect of this project, issues concerning ubiquitous computing can play important role in this project. But like technology should, they merge themselves nicely into the overall project.
8 Conclusion

During the process of this thesis work, several development iterations of the FIM application have been done, each addressing different aspects towards making it more suitable for using it in TELL. After finishing most of this development work and providing a properly working version of it, one should now reflect if those steps achieved the desired effects or not and have they answered the research questions stated in the introduction.

The first step was to get the DAQ card to work with Linux. This was required to be able to do any further experiments. Having worked with Linux for several years, DAQ cards are not the kind of everyday hardware that needs to be integrated and thus it required some time to get it work. Never the less it was possible to help the Comedi community to improve the software by running over a bug and providing an error report. Thus this step turned out to be successful, as it did not only provide the basis for FIM but also helped other people that would try to use this card.

The second step was to develop FIM. The Windows GUI allowed to control the game in a very primitive way by just allowing to insert a fault. As one of the requirements for this project was that it should be entirely built on open source it was required to build something similar starting with basic functionality. As shown in section 4.3.1 this step was accomplished and also successfully tested as shown in chapter 5. So this answers the first research question as we have successfully replaced the proprietary application. This step improved the way it was possible to record the data, but there was more to be done.

To answer the second question we used the result gathered from the first version of FIM and we tried to improve it according to some usability considerations which are stated in section 6.1. The primary test with the authors as participants showed that the result looks promising. Yet, there hasn’t been any testing done by some independent outsider proving that this concept of combining the video and data is useful in this way. This task to prove the second research question discussed in the introduction is left open to a next round of the Industry Live game.

However, it is possible to conclude that this project, even if some parts haven’t been tested due to time constraints, has proved that it was successful. It provides a solid base on which further experiments can be based on and thus helps to gain more insight on the interaction in the TELL lab. Also the overall design of the application allows to easily extend it to support e.g. different game “models”.

During the development and the writing of this report, it was shown which kinds of open source tools can be used to support the co-operative development and writing. Thus this report does not only contain topics directly related to the development itself, but also shows the process of how it was created. This part might itself prove useful for later projects, as research today is commonly done in teams, thus providing knowledge how they can be supported by a computer in a proper way, which is also important.
Finally, this report puts up some challenges to be solved in the future version of FIM. To support this development efforts, a considerable amount of technical information is contained in this report to be usable not only for reviewing what has been achieved so far, but also to be used as a developer’s guide in the future.
9 Future Enhancements

Although many good achievements are made during this thesis work, there is always a chance of future enhancements. This could depend on the way the game is played and new experiments in it. It could also be the result of the research work that there is something else to be done to improve the game and the setup. During the course of our work we encountered many difficulties, problems and we have tried our best to solve these within the stipulated time. But we do agree that there are few more things which we wanted to add to this game setup, which we were unable to do due to the lack of time.

One such future enhancement which we thought would make the game more interesting was the introduction of RFID\footnote{Wikipedia: Radio-frequency identification (RFID) is an automatic identification method, relying on storing and remotely retrieving data using devices called RFID tags or transponders.} stickers on the bits that were used. This feature will increase the data available to the FIM application so that it can have more features and thus make the game more interesting. For more details about this feature refer to section 4.7. The knowledge of the bit being fed into the Autoportal in advance creates opportunity for more custom based errors to be injected.

Another possibility we have thought of is of a random bit dispenser. All the bits are placed into the dispenser and one bit is released for every one minute of stable water level in the PID controller. So the student/pupil that acts as a worker concentrates more on the PID controller also. Now because the bits are always available in the box, they are sometimes ignoring the rule, which states that a bit is produced for every one minute of stable water level at 300 mm.

In order to take the game to the next level we can create a remote playing environment where in the fault injector can be at any place and is not constrained by the lab space. He can view the live video of the game and inject faults using the FIM remotely. When the coach feels the team has found the fault he can just give a visual signal(eg: hand rised, hand wave) to the fault injector and he can release the fault. This feature possibly avoids the players from knowing the fault injector’s actions, which could give them some clue about a fault being injected.
## B Autoportal Errors

<table>
<thead>
<tr>
<th>Degree of difficulty: A, B, C, D</th>
<th>Advanced relay panel</th>
<th>Advanced relay description</th>
<th>Component reference feet</th>
<th>Fault description and restoring comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>R0</td>
<td>To Valve 1V1</td>
<td>Sequence breaks, Air-cylinder 1A doesn’t move in positive direction.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>From</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>R1</td>
<td>To Valve 2V1</td>
<td>Sequence breaks, air-cylinder 2A doesn’t move in negative direction, bit remains over material separation section</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>From</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2A goes in immediately in positive direction</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>R2</td>
<td>To Valve 3V1</td>
<td>Sequence breaks, lifting cylinder 3A doesn’t move in positive direction, bit remains where it is.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>From</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sequence restarts, lifting cylinder 3A moves in positive direction but drops the bit if already held.</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>R3</td>
<td>To Valve 3V2</td>
<td>Sequence breaks, no suction effect, eventually drops the bit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>From</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Restarts but the bit is dropped on the table.</td>
</tr>
<tr>
<td>A</td>
<td>4</td>
<td>R4</td>
<td>To Valve 4V1</td>
<td>Sequence breaks, cylinder 4 doesn’t move in positive direction, bit is not sent into the height measurement section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>From</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sequence restarts, cylinder moves in positive direction.</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>R5</td>
<td>To Valve 5V1</td>
<td>Bad height bits are treated as good height and sorted to output folder.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>From</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Restarts functioning but bit is wrongly sorted.</td>
</tr>
<tr>
<td>B</td>
<td>6</td>
<td>R6</td>
<td>To Magnet</td>
<td>Bad bit cannot reach the trash and hangs over the black output folder.</td>
</tr>
<tr>
<td>From</td>
<td>sensor 152</td>
<td>Reverts functionality but bad bit must be taken in hand.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------------</td>
<td>---------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>7</td>
<td>To Magnet-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sensor 152</td>
<td>Good bit hangs over output folders.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>7</td>
<td>From sensor 153 Reverts functionality but bit must be taken in hand.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>8</td>
<td>To Magnet-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sensor 252</td>
<td>Sequence misses height measurement.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>9</td>
<td>From sensor 252 Sequence restarts after manually restarting the machine.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>9</td>
<td>To Pressure-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sensor</td>
<td>Sequence stops when air-cylinder 3A is in positive position to take the bit.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>From sensor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>To Capacitive-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sensor 753</td>
<td>Bit remains in property determination section and then sequence breaks.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>11</td>
<td>To Opto-Electric-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sensor 753</td>
<td>Bit black is sorted correctly. Bit metal stops sequence. Bit white is sorted as black.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>12</td>
<td>From sensor 451</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>12</td>
<td>To Magnet-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sensor 451</td>
<td>After height measurement cylinder goes in and comes out, the sequence breaks.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>13</td>
<td>From &quot;start&quot; switch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>13</td>
<td>To &quot;start&quot; switch Air leakage sound and machine goes off, entire PLC program stops.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>14</td>
<td>From regulator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>14</td>
<td>To I/P signal-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>regulator</td>
<td>Fault in the PID controller. Fault in signal to controlling valve. Due to this the water level in the tank rises but slowly gets to normal due to the automatic device.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>regulator</td>
<td>After a long time it gets normal automatically. But if removed manually, the water level decreases suddenly.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>R15</td>
<td>To Not used</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>From</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


# Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSCW</td>
<td>Computer Supported Cooperative Work</td>
</tr>
<tr>
<td>HCI</td>
<td>Human Computer Interaction</td>
</tr>
<tr>
<td>IWS</td>
<td>Industry Work Simulator</td>
</tr>
<tr>
<td>TELL</td>
<td>TEachnology and Learning Lab</td>
</tr>
<tr>
<td>DAQ</td>
<td>Digital/Data Acquisition Board</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>SVN</td>
<td>Subversion</td>
</tr>
<tr>
<td>SWIG</td>
<td>Simplified Wrapper and Interface Generator</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>FIM</td>
<td>Fault Injection and Monitoring</td>
</tr>
<tr>
<td>TUI</td>
<td>Textmode User Interface</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure SHell</td>
</tr>
<tr>
<td>GPL</td>
<td>General Public License</td>
</tr>
<tr>
<td>FOSS</td>
<td>Free and Open-Source Software</td>
</tr>
</tbody>
</table>
D Acknowledgments

The work done and the progress achieved in the field of HCI, CSCW and CSCL are of great help for any researcher. For all this achievement we would first of all like to thank our supervisor and guide Hans Kyhlbäck for his continuous inputs, guidance and support. And also for providing this opportunity to work for them. We would also like to thank Björn Stille for his guidance and encouragement he has provided us with and kept us excited.

We would also like to thank our respective families for supporting us during our work.
Bibliography


