FFI RAPPORT

DATA ACQUISITION SOFTWARE FOR FABRY-PEROT BASED FIBER BRAGG GRATING INTERROGATION HARDWARE

SAGVOLDEN Geir

FFI/RAPPORT-2000/01293

FFIE/711/116

Approved
Kjeller 12 july 2000

Stian Løvold
Director of Research

DATA ACQUISITION SOFTWARE FOR FABRY-PEROT BASED FIBER BRAGG GRATING INTERROGATION HARDWARE

SAGVOLDEN Geir

FFI/RAPPORT-2000/01293

FORSVARETS FORSKNINGSINSTITUTT
Norwegian Defence Research Establishment
P O Box 25, NO-2027 Kjeller, Norway

FORSVARETS FORSKNINGSINSTITUTT (FFI)
Norwegian Defence Research Establishment

UNCLASSIFIED

P O BOX 25
N0-2027 KJELLER, NORWAY

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

REPORT DOCUMENTATION PAGE

1) PUBL/REPORT NUMBER FFI/RAPPORT-2000/01293 1a) PROJECT REFERENCE FFIE/711/116	2) SECURITY CLASSIFICATION UNCLASSIFIED 2a) DECLASSIFICATION/DOWNGRADING SCHEDULE -	3) NUMBER OF PAGES 37

4) TITLE
DATA ACQUISITION SOFTWARE FOR FABRY- PEROT BASED FIBER BRAGG GRATING INTERROGATION HARDWARE

5) NAMES OF AUTHOR(S) IN FULL (surname first)
SAGVOLDEN Geir

6) DISTRIBUTION STATEMENT
Approved for public release. Distribution unlimited. (Offentlig tilgjengelig)

7) INDEXING TERMS

IN ENGLISH:	IN NORWEGIAN:
a) Fiber optic sensors	a) Fiberoptiske sensorer
b) Data acquisition	b) Datainnsamling
c)	c)
d)	d)
e)	e)

THESAURUS REFERENCE: INSPEC 1995

8) ABSTRACT
This report documents the data acquisition software developed during project 711 “Fiberoptisk Skrogovervåkning” for interrogation of Fabry-Perot filter based fiber Bragg grating interrogation hardware. The report describes the interface with hardware, the RT Linux real time driver for communication with hardware, the software for processing the data, file formats and error correction procedures.

9) DATE	AUTHORIZED BY This page only	POSITION
12 july 2000	Stian Løvold	Director of Research

ISBN-82-464-0460-1

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

CONTENTS

DATA ACQUISITION SOFTWARE FOR FABRY–PEROT BASED FIBER BRAGG GRATING INTERROGATION HARDWARE

1 INTRODUCTION

This report documents version 1.0 of the **FabryPerot** software developed at Forsvarets Forskningsinstitutt (FFI) as a part of the Composite Hull Embedded Sensor System (CHESS) project. CHESS aims at developing a real-time structure monitoring system to be installed in the Royal Norwegian Navy's new Skjold-class Fast Patrol Boat, and is a collaboration between FFI and the Naval Research Laboratory (NRL)(1).

The **FabryPerot** software replaces the older MS–DOS based software previously used with the NRL fiber Bragg grating (FBG) interrogation hardware. It is designed with the requirements of CHESS in mind, but may also be used with the NRL hardware for other projects.

In CHESS, the primary sensors are fiber Bragg gratings (FBGs) measuring strain. A FBG is a periodic variation in the refraction index along an optical fiber (2, 3). When light from a broadband source (e g an Erbium-doped fiber amplifier (EDFA)) is sent through it, wavelengths matching the grating periodicity are reflected at high intensity, while all other wavelengths are transmitted. When used as a strain sensor, the fiber will stretch as the substrate it is fixed to elongates, increasing the reflected wavelength. The purpose of the hardware is therefore to measure the wavelength of the reflected light, and thereby the strain.

The software receives data from the hardware, and controls its operation. It provides a user interface that allows the operator to setup the hardware, view graphs of the data as it is received, and save data for post-processing.

The software must meet some additional requirements when acting as a data source for real-time structure monitoring in CHESS. Here, a stream of data is passed from the **FabryPerot** software and other data sources to analysis software monitoring the ship's behavior. The data should therefore be checked for errors at the source to eliminate the need for error handling further along the processing pathway. Since several data sources may be in operation simultaneously, high-resolution time tags must be provided for each data point to allow subsequent alignment of data streams. Means of passing data on a network must be included as the sources and data analysis may run on different computers.

These requirements made it necessary to replace the original data collection software. The RT–Linux real-time Unix operating system was chosen for this task.

The implementation consists of three parts; the **rt_epp_handler** kernel module, which handles all communication with the hardware, the **FabryPerot** program for data acquisition and the **Player** program for file playback.

This report describes the **FabryPerot** software. A technical description of the implementation is given in Chapter 2 to help further development of the software. A guide to its operation is given in Chapter 3, while a report on the testing and verification is given in Chapter 4.

2 IMPLEMENTATION

This chapter provides an overview of the **FabryPerot** implementation, and discusses the most important choices made. A description of the hardware is first given, followed by a discussion of the choice of operating system, an introduction to RT-Linux and a description of the program architecture. After this introduction, the main features of the different program parts are described.

This chapter is not intended as a complete description of the algorithms used, but as a guide to accompany the source code.

2.1 Hardware

The hardware and its operation is described elsewhere (2, 4), but a description of the concepts important to the software implementation is provided below for the reader's convenience.

The hardware scans a band of wavelengths by passing the light from an Erbium–doped fiber amplifier through a Fabry–Perot narrow bandpass filter. The filter is scanned using an internal 16-bit ramp generator. The ramp offset voltage is adjusted by the interface program to include all sensors in the scan. The resulting intensity spectrum has peaks at the wavelengths reflected by the sensor array. The intensity maxima are found from the zero crossings of the intensity derivative signal, and the ramp count at these crossings are stored in an internal memory buffer.

Data may be collected on four channels. Channels 1 and 3 are scanned on the ramp up-scan. At the end of the scan, an interrupt request (IRQ) is generated, allowing the computer to retrieve up to 16 maxima positions for each channel. This procedure is repeated for channels 2 and 4 on the downscan. The scan frequency is fixed at approximately 359 Hz.

Communication with the controlling PC is carried out using the EPP bi-directional parallel port standard. The controlling software therefore includes an interrupt handler which acts on the parallel port interrupt requests.

Data is only available in a short ($< 100\ \mu$s) time interval after the interrupt is generated. Fast interrupt handling by the operating system is therefore imperative.

2.2 Choice of operating system

Even though interrupt requests are generated by hardware, the operating system (OS) decides the priority and speed at which control is passed to the interrupt handling software. The quality of the OS scheduler also determines whether "background" tasks, such as saving data to disks and network communication, may be carried out without losing data points.

Attempts to port the old MS-DOS software to Windows NT failed both at FFI and the NRL. The author believes that the reason was that the NT operating system did not meet the criteria for fast interrupt handling, leading to failures in communication. An alternative explanation may be that printer support embedded in the NT operating system interfered with communication.

RT-Linux (5) is a patch to the increasingly popular Linux operating system, giving high priority to interrupt handling. Since Linux is a true multi-tasking system, applications such as Graphical User Interfaces, data saving and network communications may still run while time-critical tasks are serviced.

	RT-Linux	Win-NT	MS-DOS
System time resolution	1 μs	1 ms	55 ms
Typical IRQ reaction time	< 2 μs	—	fast
Supports NTP time protocol	yes	yes	no
Supports socket network communication	yes	yes	no

Table 2.1 Comparison of critical parameters for different operating systems

There are additional synchronization requirements when several data sources are working in parallel. In an offline system, data gathered on different sources should include a common time reference to properly align the data for later analysis. Synchronization is equally important in a real-time system.

Software that synchronizes computer clocks using the Network Time Protocol (6) (NTP) is available for both the Linux and NT operating systems. Here, a time server distributes a "true" time to its clients via the network. The time server may either be a GPS clock (7) offering absolute time (UTC) or a computer acting as a master clock. The clients use a feedback loop to lock their system clocks to the time server. This method gives time coordination better than 1 ms on a local network.

Data synchronization may then be carried out by reading the system time for every data point, which is a more effective synchronization method than sampling a common reference signal.

Although other operating systems that handle these tasks equally well may be available, RT-Linux was chosen because of the author's familiarity with Linux, its general availability, the availability of fast and cheap computer hardware, and its apparent suitability for the task.

2.3 Configuring the RT-Linux OS kernel

The Linux operating system is built around a small kernel into which operating system modules may be loaded when necessary. These modules are drivers that handle various tasks, such as communication with a particular network card or harddrive.

Some of these modules reference the parallel port. The data acquisition software uses the parallel port for communication with the hardware, thus all modules and program parts that may interfere with this communication must be removed from the kernel at compilation.[1] These are:

- Floppy, IDE: Parallel port IDE device support.
- Network: PLIP (parallel port) support.
- Character devices: Parallel printer support.

In RT-Linux, the time-critical task is implemented as a kernel module, and thus becomes a part of the operating system. The kernel module cannot access disk drives or accept user input, thus a data acquisition program must have two distinct parts; the kernel module and a user front-end.

These programs communicate using special first in first out (FIFO) memory buffers named **/dev/rtf0** to **/dev/rtf63**. The kernel module accesses these FIFOs by special commands, while the user front-end may access them as ordinary files.

2.4 The kernel module rt_epp_handler.o

The **rt_epp_handler.o** kernel module is ported from the MS-DOS software developed at the NRL. Its implementation is described in this section, covering initialization, communication with the parallel port and communication with the user front-end. Additional documentation may be found in the source code.

The module has two basic tasks. It retrieves data each time the parallel port IRQ fires, passing them on the FIFO to the user front-end, and acts on requests to set the ramp signal offset. These events are handled by the **irq_handler** and **control_handler** methods.

[1]For details on how to compile: See the Linux kernel compilation HOWTO and the RT-Linux documentation. The kernel was compiled using the standard kernel distribution (http://www.kernel.org/, version 2.0.36) with the RT-Linux patch applied, and a *.config* file taken from the RedHat source distribution with parallel port support removed. Later, the source was ported to RT-Linux version 2.0 and compiled using a pre-patched version of kernel 2.2.13

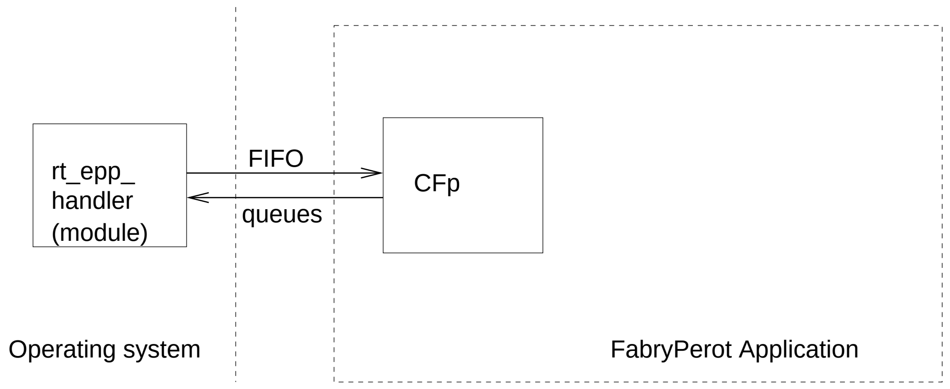


Figure 2.1 *The kernel module communicates using* FIFO *queues. In the FabryPerot application, the class CFp handles all communication with the kernel module and the general control of the hardware.*

2.4.1 Initialization

All kernel modules contain an **init_module()** and a **cleanup_module()** method, which are called upon initialization and removal of the module, respectively. In **rt_epp_handler.o**, the following initialization tasks are carried out:

1. Initialize two FIFO queues with sizes 128k (data) and 4k (control).
2. Set the parallel port register to *0x10*, enabling IRQs.
3. Set the **irq_handler** method to run at each parallel port IRQ.
4. Set the **control_handler** method to run when the user front-end passes requests on the control FIFO (/dev/rtf0).

2.4.2 Hardware communication protocol

Hardware communication is invariably carried out by first setting a message on the parallel port address latch, and then reading from or writing to the parallel port data latch. Each communication task is implemented as a separate method. The messages and data flow are shown in table 2.2.

2.4.3 Interrupt request handler

The **irq_handler** method is called every time the hardware generates an IRQ on the parallel port. Data is read into a *sample* structure, which subsequently is put on the FIFO queue. The following tasks are carried out:

Operation	Message	data	method
Read buffer 1	0	read word	ReadRam1
Read buffer 2	1	read word	ReadRam2
Get status	2	read word	ReadStatus
Set offset 1	0	write word	WriteOffset1
Set offset 2	1	write word	WriteOffset2
Write status	2	write byte	WriteStatus

Table 2.2 Parallel port communication protocol

1. System time is stored.
2. The number of intensity peaks is stored and ramp direction is retrieved.
3. A new ramp direction is set.
4. Data is retrieved.
5. The collected data are put on FIFO 1.

The Linux kernel was changed in versions 2.2.X, allowing the system time, controlled by the Network Time Protocol, to be non-monotonic. This is unacceptable for sequence consistency checks made later on.

To correct for this, the the RT-Linux time, which is monotonic, was included in the communication format in addition to the system time. This allows the true time to be estimated in the FabryPerot application by linear interpolation.

2.4.4 Control request handler

The **control_handler** method is called each time a request is put on FIFO 0. The request consists of a command (byte) and a message (word). The implemented requests are shown in table 2.3

Command	Message	Explanation
1	0	Turn off dataflow to FIFO 1
1	1	Turn on dataflow
2	offset	Set offset 1
3	offset	Set offset 2
4	—	Reset hardware

Table 2.3 Control commands passed to **rt_epp_handler.o** *on FIFO 0.*

2.5 Class CFp

The *CFp* class is the primary class in the data acquisition software. It contains methods for controlling the hardware, and provides data for the graphical frontend. CFp is independent of the graphical frontend, and is intended as a driver for the hardware.

2.5.1 Initialization

The **CFp()** constructor is called with the configuration file name as an argument. The configuration file contains information on operation of the hardware and the conversion from raw data to strain values. The information must be available to the constructor as a text file having the following format:

```
derivative limit (for Derivative Limit correction)
offset 1
offset 2
reference grating number on array 1 (counting from 0)
reference setpoint
M00  Ramp count to wavelength conversion
M01  polynomial factors (Up-scan)
M02
M03
M10  Down-scan
M11
M12
M13
64 sensor equilibrium wavelengths
```

After construction, the source is selected by calling the overloaded **Initialize** function. Both real-time and file playback sources are supported.

2.5.2 Controlling the hardware

Hardware control is carried out by the **FIFOstart(), FIFOstop(), SetOffset(int), GetOffset(int) and EPPreset()** methods. The methods check the source type, so calls to these methods are safe also when playing back a file.

2.5.3 Data retrieval

The **int GetData()** method loads data from a file or FIFO into the *samp*[2] structure. A

[2]see common.h

boolean TRUE (or 1) is returned if data is available, 0 is returned otherwise. When reading from a FIFO, GetData() waits for data available, and always returns 1.

Upon return, raw data is available in the *samp* structure, while calculated strains are available in the *strains* structure.

2.5.4 Data correction

If the *FlCorrect* flag is set, the **GetData()** methods will attempt to correct the data if sensors are missing by calling **CorrectArray**. These instances are detected when the hardware returns a lower sensor count than that stored in *strains.STcount[]*. The missing sensor(s) are then identified by minimizing the sum of the absolute value of the derivatives calling a recursive search procedure, **EvalDiff**. It is assumed that the hardware sensor count *nfound* is correct, and that the *nfound* first data points are valid. The best combination is stored in *min_index[]* upon return, while the sum of derivatives is stored in *min_sum*.

If *FlPersistent* is set, it is assumed that the same sensor is missing as long as the sensor count is constant. This speeds up the search algorithm, but tests have shown that the assumption is not valid in all cases (see 4.2).

This correction strategy relies on the hardware sensor count being correct. Some instances of errors in the returned sensor count have been observed. To identify these cases, the derivative of all data points is calculated by **DerivativeLimit()** if *FlDerivLimit* is set. When a derivative larger than the limit set in the configuration file is encountered, data for the remaining sensors on the channel is replaced with the previous reading.

Text messages with details of the corrections being made are printed to the console if *FlPrintCorrect* and *FlPrintDerivLimit* is set for fallout and derivative corrections respectively.

2.5.5 Reference feedback

The Fabry-Perot filter is a piezo–based mechanical filter that may drift during measurement. Channel 1 should therefore include a reference grating to which the ramp signal may be locked. This is achieved by changing the ramp signal offset by a fraction of the difference between the average reading on the reference grating and the target value. The reference grating and its target value are selected in the configuration file.

The reference grating feedback loop is enabled when *FlLocked* is set.

2.5.6 Reference protect

In some cases, for instance when the online correction routine is turned off, the feedback loop may attempt large corrections due to erroneous readings when sensors are missing. If *FlRefProtect* is set, changes to the ramp offset are not made if the deviation (in rampcount) is larger than *REF_ERROR_BOUND*. These instances are counted as reference errors.

2.5.7 Normalization

Strains are calculated relative to some value defined as neutral strain. Neutral strain is calculated from the average of *NORM_NUM_AVG* measurements[3] taken after *FlNormalize* is set. The average values may be written to the configuration file by calling **write_setupfile()**

2.5.8 Old data files

The sensor counts were not saved in the data format used before June 1999. Thus, they must be inferred from the data when playing back old files. This is done by setting absolute limits on the observed values and their derivatives. These values are read from the *bandwidth* file passed to the **Initialize()** method. The expected sensor count must be entered by the user.

The *bandwidth* file has the following format

```
max offset ch1 S0
max derivative ch1 S0
...
max offset ch1 S15
max derivative ch1 S15
max offset ch2 S0
...
max offset ch3 S0
...
max offset ch4 S0
...
```

The offset is calculated from the average of the first *RAW_NUM_AVG* data points. File playback should therefore not be started with the correction feature on. If correction of the entire run is desired, a part of the data file should first be played back with correction off to calculate the averages, before starting over again with the correction feature on.

[3]currently 1000

2.6 GUI frontend

The user frontend is designed as event-driven software built on the Qt (8) toolkit for programming graphical user interfaces (GUI). Events are generated either by timers or user interaction (i.e. pressing buttons etc).

The GUI uses two primary classes. *SetupWidget* handles the setup of the hardware, while *RunWidget* handles data acquisition. Both classes incorporate a data acquisition loop (see 2.6.1), together with additional methods to control the *CFp* class. In the **FabryPerot** application, *SetupWidget* is run first, and replaced by *RunWidget* when hardware setup is completed. Only *RunWidget* is used in the **Player** application, since no hardware setup is necessary for file playback.

2.6.1 The data acquisition loop

Qt is in principle an event handling toolkit with an extensive selection of library routines for generating graphical widgets such as buttons, sliders etc. It maintains a list of methods to be run following a specific event.

The software uses a Qt timer with 0 timeout to call **timerEvent**, which launches **CFp::GetData()**, to retrieve data as often as possible. This design works as long as **timerEvent** is called at a rate at least as fast as data is placed on the FIFO by **rt_epp_handler.o**. This gives an overhead, since a timer event and an event handling mechanism needs to be initiated for each call to **CFp::GetData()**.

An alternative strategy would be to use the Linux multi-thread feature,[4] allowing data acquisition to run as an infinite loop in a separate process.

2.6.2 The SetupWidget class

The purpose of SetupWidget is to provide the user with an interface to setup the hardware before data acquisition. Typical setup tasks are:

- Roughly adjust the ramp offset to bring all sensors within the scan range.
- Lock to the reference sensor.
- Normalize to define the zero-strain value.
- Calibrate the ramp-count to wavelength conversion.

All hardware control features are implemented in the *CFp* class, but *SetupWidget* provides an interface to control them. *SetupWidget* also displays graphics showing the position of the

[4]LinuxThreads

peaks identified by the hardware (see figure 3.2). The positions are given in raw rampcount values. Each channel is plotted by an instance of the *CFbgPos* class.

The execution of *SetupWidget* is controlled by calls to **SetupWidget::timerEvent** by Qt as described in 2.6.1. A counter, *irq_cnt*, is used to update the display of an alternating pair of channels every *UPDATE_FREQ* data points, and the current ramp offset value every *REF_UPDATE_FREQ* data points.

2.6.3 The RunWidget class

The *RunWidget* class offers real-time display of strain as well as control of the *CFp* real-time data correction features. *RunWidget* also saves data to binary (see 2.7) or ASCII text files. Calculated strains and raw data are also passed on the network using the *ChTrans* class(9).

Data retrieval in *RunWidget* is triggered by calls to **RunWidget::timerEvent** by the Qt event handler, as in *SetupWidget*. The *irq_cnt* counter controls the frequency of plotting and the frequency of screen updates.

A time tag consistency check is also carried out in **RunWidget::timerEvent**. In some cases,[5] its value is 0.01s less than expected and automatically updated.

In some cases where the load on the computer is high, one or more data points may be lost. Data from the previous reading is therefore filled in the gap if *FlSeqCorr* is set, to maintain a constant sampling rate. If the gap is larger than *SEQ_CORR_MAX*, the correction is not made.

Messages detailing all sequence corrections are printed to the console if *CRunStatus::FlPrintSeqErr* is set.

The methods **FileOpen(), FileClose()** and **FileAppend()** are wrapper functions that call the appropriate open, close and append functions for the selected file format. **NetworkAppend()** copies data to a buffer for later network transmits.

2.6.4 Graphical display tools

Graphical display of calculated strain data is carried out using the *CViewStrain* and *CStrainGraph* classes. *CViewStrain* contains methods to change the plotting scale of each graph, while *CStrainGraph* plots a graph of the strains.

CStrainGraph maintains a circular buffer of the plotting positions of 512 data points. New data are added to the buffer by calls to the **CViewStrain::StrainUpdate()** method in the *RunWidget* event loop. The display is updated by a call to the **CViewStrain::DoRepaint()** method.

[5]Typically every 3-4 s

The current status of the feedback loop and error counts are printed using the *CRunStatus* class. The information is updated by calls to **CRunStatus::timerEvent** every 0.5s. The class also contains methods to turn on and off printing of correction details.

2.6.5 The **FabryPerot** application

The **FabryPerot** application operates in either the *setup* or the *run* mode. Control of the *CFp* class is therefore given to either *SetupWidget* or *RunWidget* by the class *CContWidget*, depending on the operating mode. Control is always passed to the *SetupWidget* class first.

2.6.6 The **Player** application

Data file playback is implemented to allow using old data files for developing real-time analysis software for the CHESS project. In this case, a single *PlayControl* class may launch several instances of *RunWidget*, each acting as a separate data source.

Time-coordinated data streams are obtained by setting a maximum time limit for file playback by calling **CFp::SetPlayToTm**. If the next record in the file has a timetag larger than the target time, **CFp::GetData** will return 0. The time limit is updated by periodic calls to **PlayControl::timerEvent**. Accelerated and reduced playback speed is also implemented. The maximum playback speed depends on the number of sources, the amount of data correction done etc., so the actual playback speed may in some cases be slower than the target speed.

Although only **FabryPerot** data sources are implemented at the moment, *PlayControl* is designed to allow playback of all sources inheriting the *CSourceControl* class.

2.7 Binary file format

The binary file format saves both raw data and calculated strains, and is suitable for file playback. The file format is as follows:

```
<header>
char  Sensor Count MAX ch 1
          - " -           2
          - " -           3
          - " -           4

<Data block>
int   timetag (seconds)
int   timetag (usec)
```

```
short sensor count channel 1
          - " -            3
short strain   ch1 sensor 0
short raw data ch1 sensor 0
.
.
short strain   ch1 sensor MAX
short raw data ch1 sensor MAX
short strain   ch3 sensor 0
short raw data ch3 sensor 0
.
.
short strain   ch3 sensor MAX
short raw data ch3 sensor MAX
int   timetag (seconds)
int   timetag (usec)
short sensor count channel 2
          - " -            4
short strain   ch2 sensor 0
short raw data ch2 sensor 0
.
.
short strain   ch2 sensor MAX
short raw data ch2 sensor MAX
short strain   ch4 sensor 0
short raw data ch4 sensor 0
.
.
short strain   ch4 sensor MAX
short raw data ch4 sensor MAX

<Data block>
....

<Data block>
...
```

Storing calculated strains is not necessary for file playback or post processing since they may be found given the information in the configuration file. There is also no need to store the timetag as two integers, since the time is always used as a double precision floating variable in the system. One should therefore consider changing the file format to reduce filesize.

2.8 Network data packet format

All data received by the *RunWidget* class are distributed on the network. The network data packets contain time information and calculated strains in the following format:

```
<Data block>
double timetag
short strain ch1 S 0
...
short strain ch1 S MAX
short strain ch2 S 0
...
short strain ch3 S 0
...
short strain ch4 S 0
...
```

Raw rampcount values are transmitted as unsigned short integers using a similar data packet format.

The port number, data block size, and number of data blocks in each transmitted packet is selected at construction of the *ChTrans* objects in the *RunWidget* constructor.

2.9 Qt issues

The Qt library have some added features which may cause unexpected behavior for the unaware programmer.

- All objects inheriting QWidget are automatically destroyed upon calling the destructor of their parent.
- Calls to **repaint()** will call repaint for all children.

3 USER DOCUMENTATION

This chapter documents the operation of the **rt_epp_handler** kernel module, the **FabryPerot** program for data acquisition and the **Player** program for file playback. This documentation refers to several technical concepts which have been explained in Chapter 2.

3.1 Data acquisition using FabryPerot

3.1.1 Setting up the hardware

The hardware has potentiometers to adjust the amplification of the intensity signal and discrimination level of the derivative signal for each of the four channels. These should be adjusted whenever the broadband light source or the sensor configuration changes. The amplification should be set so that the peaks are as large as possible without saturating, while the discrimination level should be set between the noise level and the smallest negative peak in the derivative signal. These signals are found on the pin numbers given in table 3.1.

Signal	Pin assignment
Sum	Pin 6 on Op-Amp 1
Derivative	Pin 6 on Op-Amp 2
Discrimination level (Ch 1&3)	Pin 5 on comparator
Discrimination level (Ch 2&4)	Pin 12 on comparator

Table 3.1 Hardware pin configuration

3.1.2 Loading the **rt_epp_handler** kernel module

The kernel module, which carries out communication with the hardware, must be loaded together with some RT-Linux support modules prior to data collection. Only the super user may load modules. The following commands must therefore be run before starting the application:

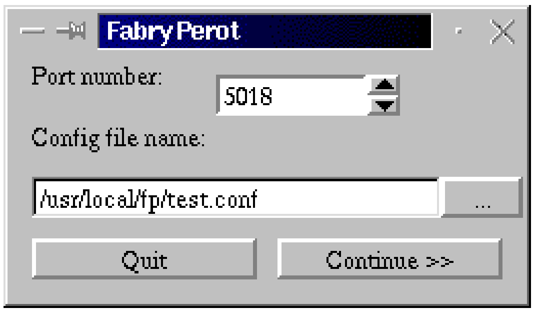
```
> su
<enter root passwd>

<Remove old instances of the module>

> /sbin/rmmod rt_epp_handler
> /sbin/rmmod rtl_fifo
> /sbin/rmmod rtl_sched

<Insert kernel modules>

> /sbin/insmod /usr/src/rtlinux/rtl/modules/rtl_fifo.o
> /sbin/insmod /usr/src/rtlinux/rtl/modules/rtl_sched.o
```



Port number: 5018

Config file name: /usr/local/fp/test.conf

Figure 3.1 Choosing the port number and configuration file

```
> /sbin/insmod ./rt_epp_handler.o
```

Usually, module loading and unloading are carried out in scripts automatically run at CHESS user login.

3.1.3 **FabryPerot** initialization

The user is expected to choose the configuration file and the port number network clients may connect when starting **FabryPerot**. Calculated strain values will be transmitted on the selected (odd) port number, while raw rampcounts are made available on the next port number.

Most parameters important to the operation of the hardware is stored in the configuration file. The file format is documented in section 2.5.1. The user is expected to have edited the following information:

line 1	Derivative limit. This number should be larger than the maximal derivative expected during measurement, but smaller than the separation (in rampcount) of neighbor sensors.
line 4	Reference grating number (gratings are counted 0,1,2,3..)
line 5	Reference setpoint. This is the rampcount number the reference is locked to. It should be chosen so that all gratings are included in the scan.
lines 6-11	Polynomial factors. The rampcount to wavelength conversion factors should be found by a third-order polynomial fit to wavelength standards (see 3.1.4).

3.1.4 The setup window

The setup window is used for initializing the hardware prior to data collection. The sequence of operation is as follows:

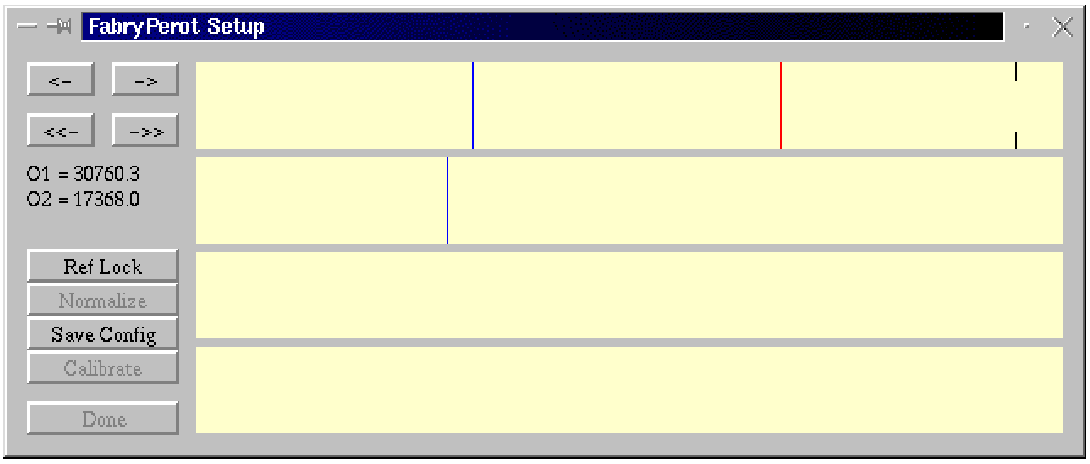


Figure 3.2 *The setup window. The rampcounts of the gratings on each array are shown as vertical lines. The reference grating is shown in red. The broken line indicates the reference setpoint. The current values of the ramp offsets are shown as O1 and O2.*

1. Adjust ramp offset voltage. The ramp offset voltages O1 and O2 are adjusted using the "$->$" and "$->>$" buttons respectively. The purpose of the adjustment is to position the reference grating close to its setpoint while retaining sufficient dynamic range of O1 to allow the feedback loop to follow the reference. Ideally, O1 should be close to 30000.

2. Lock the reference grating to the setpoint. The reference feedback control is started by pressing *Ref Lock*. It is not possible to proceed to the data acquisition window when reference lock is off. The sensor count is also stored when pressing this button.

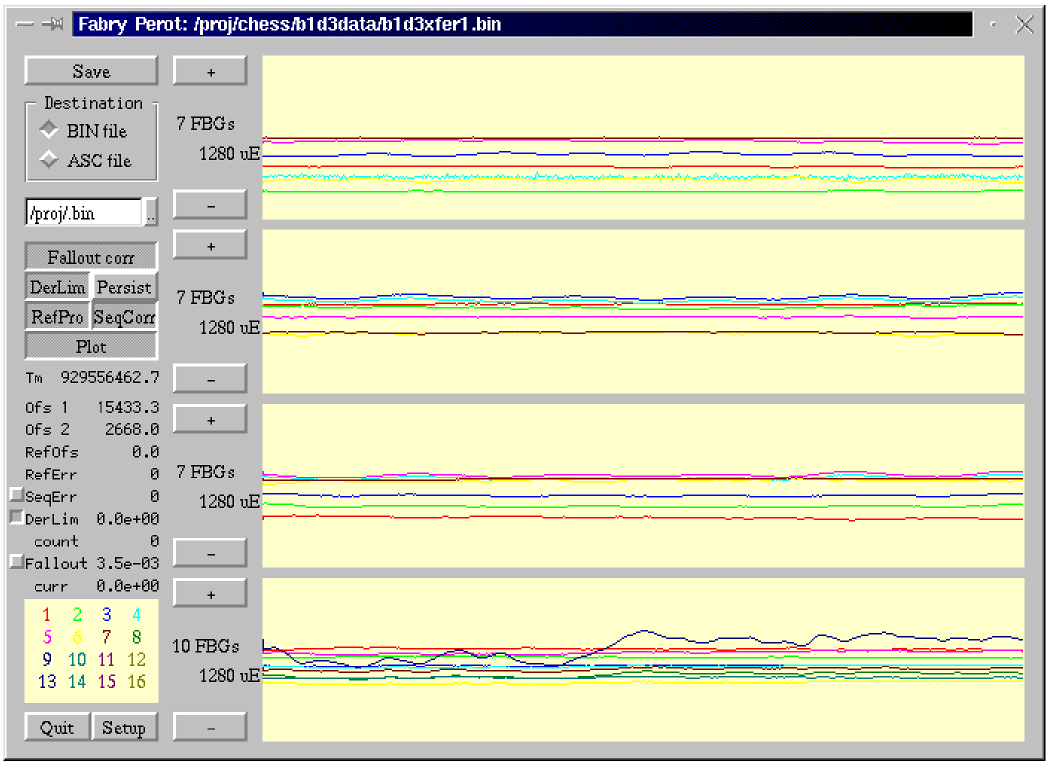
3. Normalize the strain (optional). By pressing normalize, an average of each sensor reading is calculated. This average is used as the 0-strain value. These values are obtained from the configuration file if no normalization is done.

4. Save configuration (optional). Saves the normalization and ramp offset values to the configuration file.

5. Calibrate (optional). This function is used for calibrating the ramp count to wavelength conversion, printing the average rampcount for each grating.

When the hardware is properly set up and the reference feedback loop makes only small adjustments (typically < 1 rampcount), the user may proceed to the data acquisition window by pressing *Done*.

3.1.5 The data acquisition window



```
Tm      929556462.7
Ofs 1       15433.3
Ofs 2        2668.0
RefOfs          0.0
RefErr            0
SeqErr            0
DerLim      0.0e+00
count             0
Fallout     3.5e-03
curr        0.0e+00
```

Figure 3.3 The data acquisition window

The data acquisition window provides and interface for the acquisition, network distribution and storage of data. The window is separated in a control part to the left and a graphical display of strain values to the right.

Data storage control is located in the upper left corner. The program supports native binary and ASCII file formats. The binary filetype saves all information necessary to play back files and re-calculate the strains (see 2.7). This should be the standard filetype, since error detection and correction is possible during post-processing. The ASCII filetype saves time and strain data in human readable form.

The *Save* button turns saving on and off. The destination filename may be entered in the box. The small button to the right of the box brings up a file selection window. The application does not allow overwriting old files.

The buttons below the *Save* group control the error correction routines. The routines are described in 2.5.4 and 2.6.3, but are outlined briefly below.

- *Fallout Correct* turns on and off the correction routines for hardware detected sensor fallout. Data for the previous measurement is used for the missing sensors.

- *Derivative limit.* This error check is a software consistency check of the returned data. If the derivative (in rampcount) of a sensor exceeds a threshold set in the configuration file, it is assumed that there was an error in hardware-software communication, and the missing data points are replaced by the previous readings.

- *Persistent.* This controls the behavior of the fallout correction routine. If set, the software will only search for the missing sensors each time the sensor count changes.
- *Reference Protect.* Erroneous readings of the reference sensor may destabilize the feedback loop. Reference protect imposes an upper limit on the magnitude of ramp offset corrections.
- *Sequence Correct.* High load on the computer (such as turning on/off plotting) may lead to missing data points. If sequence correct is set, missing sequences will be filled with previous readings. If the missing sequence is too long, the sequence will not be filled in.
- *Plot* turns on/off data plotting.

Some key parameters are displayed below these control buttons. *Tm* is the sample time (in seconds UTC). *Ofs 1* and *Ofs 2* are the current value of the ramp offsets, *RefOfs* is the correction last made to O1 by the reference feedback loop. *RefErr* is the number of times the attempted reference correction was larger than the discrimination level, *SeqErr* is the accumulated number of missing data points, *DerLim* is the communication error rate[1] while *count* is the accumulated number, *Fallout* is the accumulated fraction of readings with missing data, and *curr* is the current rate.

The small buttons to the left of these status numbers are used to turn on error message printing to the console.

Buttons for returning to the setup menu and to quit the application are provided in the lower left corner.

The strain values for all sensors in an array are shown in four plotting windows. To the left, the number of gratings and plot scale are shown. The scale is changed using the “+” and “–” buttons.

To successfully record data, the following procedure should be followed.

1. Check that the number of sensors in each channel is correct.
2. Select the level of data correction. It is recommended that *Ref Protect* is set. *Fallout correct* and *Derivative limit* should be set when data is processed real-time. Data may also be filtered for errors during post-processing. It is, however, important to note that the reference feedback loop may not function well if the data in channel 1 is of poor quality. Correction should therefore always be used in this case.
3. Select file format and name.

[1]As a fraction of the total number of data points

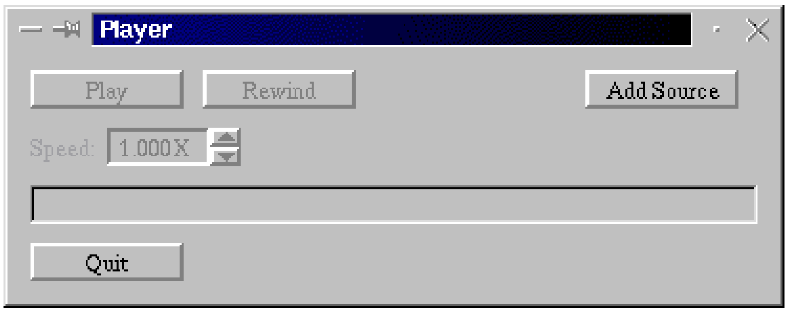


Figure 3.4 *The* **Player** *startup window*

4. Start saving.

Data are transmitted on the network as long as the data acquisition window is running. Thus, to transmit data, only steps 1 and 2 have to be completed.

3.2 File playback

Player, the file playback application, is mainly used for transmitting real data for the development and testing of components in the CHESS real-time structure surveillance system. File playback may also be used for offline error filtering and converting the native binary file format to text files.

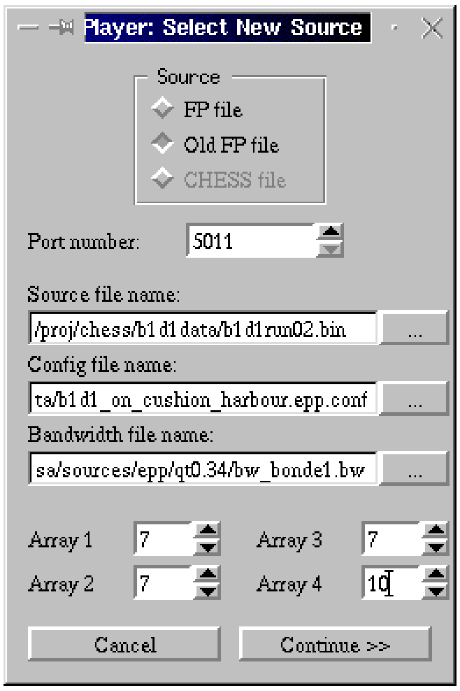
3.2.1 Adding a source

Data sources are added by clicking the *AddSource* button in the **Player** startup window (figure 3.4). This action brings up a source selection window (figure 3.5). Currently, binary **FabryPerot** files are supported. The user should enter the data file name, the configuration file name and the port number for the data stream. If the old file format is chosen, the user also has to enter the sensor count for each channel and the name of the *bandwidth* file (See 2.5.8).

The selection process should be repeated for those data files that should be played concurrently. Since playback is controlled by time-of-collection, only files recorded simultaneously should be used.

3.2.2 The Play interface

When the desired number of sources have been selected, playback is started by pressing the *Play* button (figure 3.7). The files will start playing at the beginning at maximum speed



Port number: 5011

Source file name: /proj/chess/b1d1data/b1d1run02.bin

Config file name: ta/b1d1_on_cushion_harbour.epp.conf

Bandwidth file name: sa/sources/epp/qt0.34/bw_bondel.bw

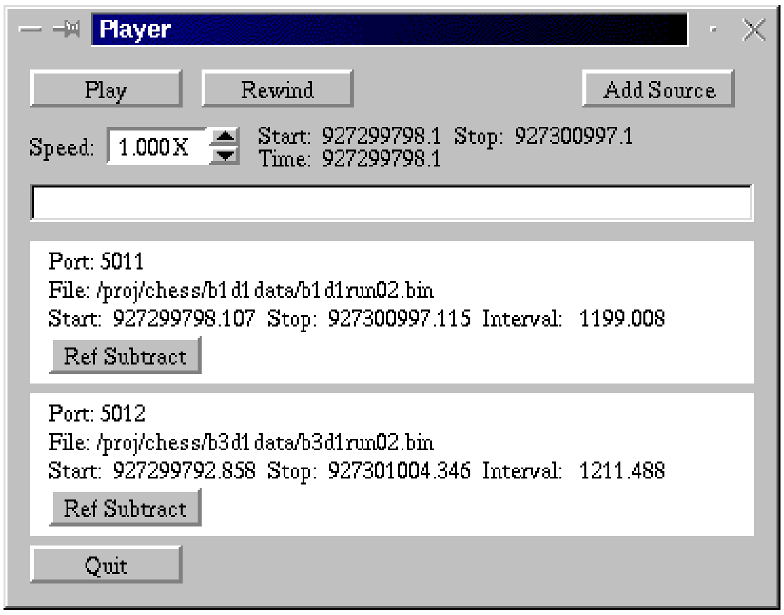
Array 1 7

Array 2 7

Array 3 7

Array 4 10

Figure 3.5 *The* **Player** *source selection window*



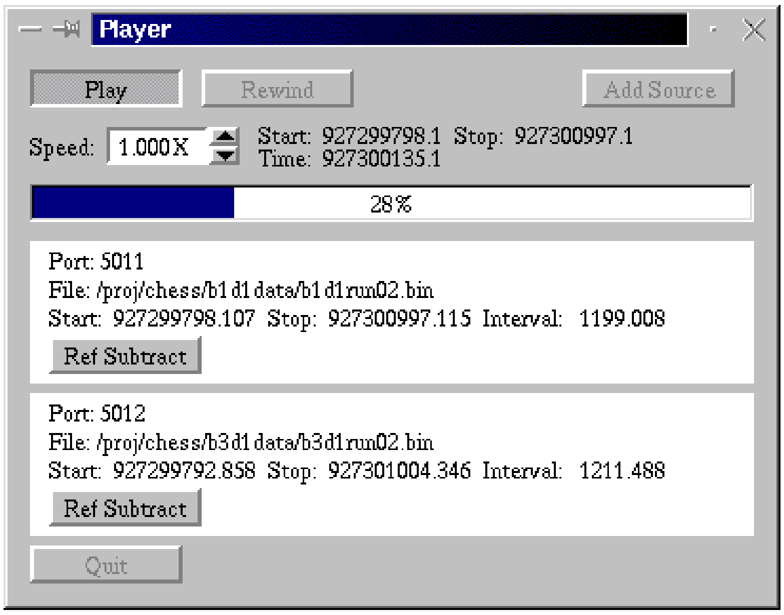
Speed: 1.000X

Start: 927299798.1 Stop: 927300997.1
Time: 927299798.1

Port: 5011
File: /proj/chess/b1d1data/b1d1run02.bin
Start: 927299798.107 Stop: 927300997.115 Interval: 1199.008

Port: 5012
File: /proj/chess/b3d1data/b3d1run02.bin
Start: 927299792.858 Stop: 927301004.346 Interval: 1211.488

Figure 3.6 *The Play control window with two sources selected*



Speed: 1.000X Start: 927299798.1 Stop: 927300997.1
Time: 927300135.1

28%

Port: 5011
File: /proj/chess/b1d1data/b1d1run02.bin
Start: 927299798.107 Stop: 927300997.115 Interval: 1199.008
Ref Subtract

Port: 5012
File: /proj/chess/b3d1data/b3d1run02.bin
Start: 927299792.858 Stop: 927301004.346 Interval: 1211.488
Ref Subtract

Figure 3.7 The Play control window during playback

until the target time, displayed above the progress bar, is reached. The target time begins at the largest common start time, and stops at the lowest common stop time. The playback speed multiplier indicates the desired playback speed, but the actual speed may be lower if all system resources are consumed. Data are supplied to the a data acquisition window for each source, which may be operated as described in 3.1.5.

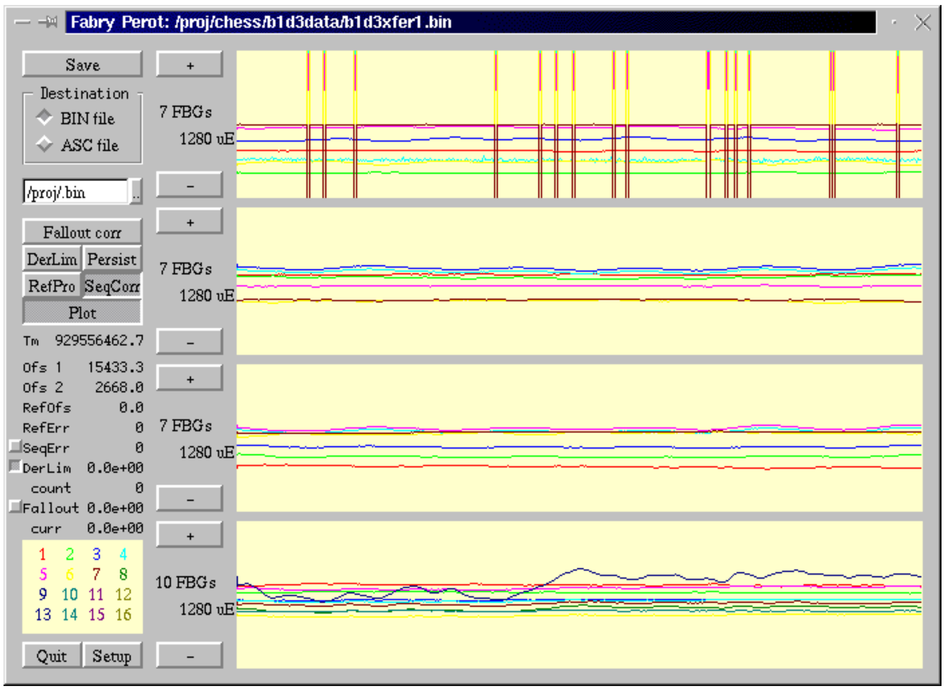
The play control window has a control button for each source. *Ref Subtract* turns on reference subtract, subtracting the reference error from all sensor raw data. This is useful in cases where data were collected without using *Reference protect*, and the feedback loop made large erroneous corrections.

The progress bar, as shown in figure 3.7, shows the progress of the target time.[2] The *Rewind* button is used to resume playing from the start.

4 TESTING AND VERIFICATION

The data acquisition programs have been extensively tested during field measurements in the CHESS project, and in laboratory tests both at FFI and the NRL. The programs work reliably, and have recorded gigabytes of data.

[2]Provided that the sources keep up with the desired playback speed



```
Tm   929556462.7
Ofs 1    15433.3
Ofs 2     2668.0
RefOfs       0.0
RefErr         0
SeqErr         0
DerLim  0.0e+00
 count         0
Fallout 0.0e+00
 curr   0.0e+00
```

Figure 4.1 Playback with correction off

4.1 Data verification

Strains have been measured independently using conventional strain gauges, showing a reasonable agreement with the fiber-optic data acquisition system (10).

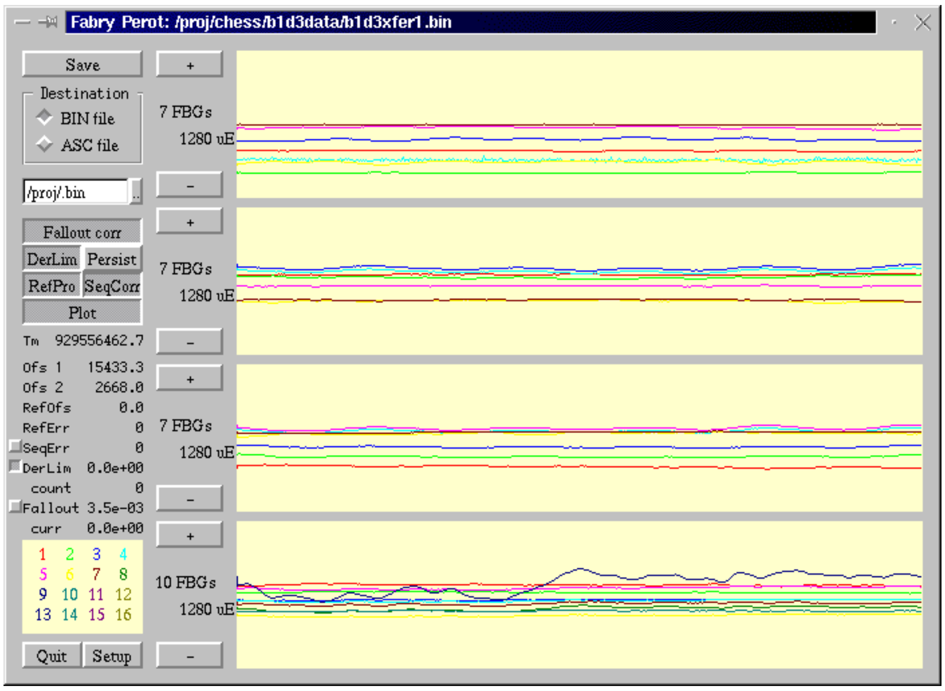
4.2 Sensor fallout correction

Most measurement errors may be ascribed to failures in hardware or problems with measurement design. These include missing sensors due to:

- Low returned intensity. These errors are often due to instabilities in the light source.
- Intensity peak collision. These errors occur when two neighboring peaks cannot be separated by the hardware.

These errors are detected by the hardware as a lower sensor count. The first error type is successfully corrected by the software, while the second type may give unpredictable results for the sensors in question. This is a limitation of the measurement system, and care should be taken in sensor design to keep the number of such collisions at a minimum.[1]

[1]Which is easier said than done....



```
Fabry Perot: /proj/chess/b1d3data/b1d3xfer1.bin

Tm   929556462.7
Ofs 1    15433.3
Ofs 2     2668.0
RefOfs       0.0
RefErr         0
SeqErr         0
DerLim   0.0e+00
count          0
Fallout  3.5e-03
curr     0.0e+00
```

Figure 4.2 Playback with correction on

Figures 4.1 and 4.2 show the results of data correction for a data set where the source intensity was deliberately turned low to produce numerous errors of the low intensity type. These tests show that the error correction feature works successfully. However, the assumption made for *Persistent* correction, i.e. that the same sensors are missing as long as the sensor count is constant, is invalid in cases where the error rate is very high. This is due to frequent flickering of several sensors, leading to events where the missing sensor returns at the same instant another sensor falls out. Hence, the sensor count is constant while the missing sensor changes, and correction fails. On the other hand, the *Persistent* option **may** be able to track one of the two peaks in a peak collisions, as well as reducing the computational demands on the computer.

Some cases of hardware communication failure have also been observed. Here, the sensor count indicates that all gratings have been found, while the data retrieved are clearly wrong. To correct for these events, data consistency checks should be carried out. This is the purpose of the *Derivative limit* correction.

The data correction features have been extensively tested on recorded data and during data acquisition in the lab, as well as in extensive field tests.

4.3 Timetag correction

Timetag errors, frequent in earlier versions of the data acquisition software, are extremely rare after the FIFO data buffer size was increased from 4k to 128k and the monotonic RT-linux timer was used as time reference.

4.4 Data recording

Data are normally recorded to a local harddisk. No instances have been observed where data points have been missing due to recording. Hence, the application is capable of recording a continuous data stream whose length is only limited by available harddisk space.

Data have been recorded at sea in very harsh vibrational environments. Although storage reliability ultimately depends on the harddrive specifications, the CHESS sea tests show that the measurement platform is sufficiently robust for demanding field experiments.

4.5 Network communication

Data may be transferred over the network for realtime analysis. Tests have shown that data is transferred reliably, and that communication does not interfere with data retrieval.

5 CONCLUSION

Extensive testing show that the **FabryPerot** software record, play back and correct fiber optic strain data with high accuracy and reliability. The software contains several features important to the CHESS system which were not available in the original MS-DOS version. The software is therefore suited as an element in a future real-time structure surveillance system.

The RT-Linux operating system is the key to the success, due to its combination of real-time features with graphics and network support.

References

(1) Pran K, Sagvolden G, Farsund Ø, Havsgård G B, Wang G (2000): A summary of project 711 ”Fiberoptisk skrogovervåkning” (CHESS), FFI/RAPPORT-2000/01298, Forsvarets forskningsinstitutt (Approved for public release. Distribution unlimited).

(2) Davis M A, Bellemore D G, Putnam M A, Kersey A D (1996): Interrogation of 60 fibre Bragg grating sensors with microstrain resolution capability, *Electronics Letters* **32**, 15, 1393–1394.

(3) Pran K (1995): Design of optical fibre Bragg gratings, FFI/RAPPORT-95/05818, Forsvarets forskningsinstitutt (Approved for public release. Distribution unlimited).

(4) Havsgård G B (1997): Besøk på Naval Research Laboratories, Washington DC, 25. august til 5. september 1997, , FFI travel report.

(5) Http://www.rtlinux.org/.

(6) Http://www.ntp.org/.

(7) Http://www.truetime.com/.

(8) Http://www.troll.no/.

(9) Sagvolden G (2000): Modular distributed signal processing network for CHESS, FFI/RAPPORT-2000/01294, Forsvarets forskningsinstitutt (Approved for public release. Distribution unlimited).

(10) Farsund Ø (1999): Strain measurements - a comparision of strain gauge and fiber Bragg grating measurements on KNM Skjold, FFI/RAPPORT-99/06462, Forsvarets forskningsinstitutt (Approved for public release. Distribution unlimited).

(11) Http://www.cygnus.com/.

(12) Http://www.gnu.org/.

APPENDIX

A AUXILIARY PROGRAMS

A.1 swapdata

swapdata is a command line program for swapping the byte order in the binary data files to make them readable by **matlab** running on mainframe computers. The command line options are documented by issuing a *-h* flag. The program was originally designed by Ole Henrik Waagaard, but is now re-written by the author. Typical use for new data files is

```
swapdata -f run00.bin > run00.cbin
```

while the sensor count has to be included for the old data file format

```
swapdata -s 7,7,7,10 -f run00.bin > run00.cbin
```

B FUTURE IMPROVEMENTS

B.1 Multi-interface support

Data acquisition tasks often require monitoring several sensors simultaneously. The hardware is limited to 63 sensors. Practical restraints, such as the dynamic range of sensors, may further reduce this number. Therefore, several Fabry-Perot FBG interfaces are used in parallel for large measurement tasks.

Currently, a dedicated computer is needed to control a single FBG interface. It is simple to re-write the software to support several interfaces running on a single computer, however, the limited time available for retrieving data, CPU-speed, and the number of available IRQ-lines sets a limit on the number of interfaces that may run concurrently.

B.2 Autostart

To be useful as a constantly running surveillance system, the application should be able to auto-start and auto configure. The Linux system may be configured to autostart the X windows system and the application. Autostart of options and auto-configuration may be carried out using a more extensive system of configuration files together with command-line options.

B.3 Performance

The system is now developed, compiled and tested using the GNU egcs compiler (11). Performance gains may be earned by using a commercial Pentium optimizing compiler for Linux, such as *Code Fusion* (11). Code fusion is benchmarked to produce code 20% faster than MS Visual C++ and twice as fast as egcs.

C LICENSING ISSUES

Most code written for Linux is licensed under the GNU (12) General Public License (GPL). This license is a protection for the writers of open source software, restricting others from patenting or incorporating their code into proprietary software, while developers of open source software may freely incorporate the code in their applications. The output generated by running GPL'd program (such as egcs) is not restricted under this license.

When compiling, the programs are linked, statically or dynamically, with the C library. Legally, this makes the binary output a combined work. To promote commercial use and standardization of the GNU C library, it is licensed under the Less General Public License (LGPL), which allows writers of proprietary code to link against LGPL'd libraries.

Qt (8) has a more restrictive license. The Q public license (QPL) grants the users the right to write open source software linked with the library. Developers of commercial applications must obtain the Qt Professional Edition, which grants the rights to distribute proprietary software linked with Qt. One license costs $1550. The licenses are sold on a per-programmer basis.

The difference between these licenses reflects that GNU is an organization promoting open source software development, while Troll TECH is a company developing a cross-platform GUI library for commercial proposes. However, Troll has decided to grant free use of their libraries for writing open source programs to promote the use of Qt.

Hence, to legally commercialize the **FabryPerot** application, i.e. as a part of a commercial structure surveillance system, a Qt Professional license must be obtained.

D SYSTEM REQUIREMENTS

The software was tested on machines with the following configuration:

CPU	PII-400 / P200
RAM	128 MB
Disk	Seagate ST36531A
Parallel	EPP
Par. IRQ	7
Par. IO addr	378h
OS	Mandrake 5.3
kernel	2.0.36
RT-Linux	v1.1

DISTRIBUTION LIST

FFIE **Dato:** 12 july 2000

RAPPORTTYPE (KRYSS AV) [✓ X] RAPP ☐ NOTAT ☐ RR	RAPPORT NR. 2000/01293	REFERANSE FFIE/711/116	RAPPORTENS DATO 12 july 2000
RAPPORTENS BESKYTTELSESGRAD UNCLASSIFIED		ANTALL EKS UTSTEDT 41	ANTALL SIDER 37
RAPPORTENS TITTEL DATA ACQUISITION SOFTWARE FOR FABRY-PEROT BASED FIBER BRAGG GRATING INTERROGATION HARDWARE		FORFATTER(E) SAGVOLDEN Geir	
FORDELING GODKJENT AV FORSKNINGSSJEF:		FORDELING GODKJENT AV AVDELINGSSJEF:	

EKSTERN FORDELING **INTERN FORDELING**

ANTALL	EKS NR	TIL	ANTALL	EKS NR	TIL
		Naval Research Laboratory	14		FFI-Bibl
1		Fiber Optic Smart Structures Section	1		Adm direktør/stabssjef
1		V/Gregg Johnson	1		FFIE
1		V/ Sandeep Vohra	1		FFISYS
1		V/ Gary Cogdell	1		FFIBM
		Code 5600	1		FFIN
		Washington DC 20375			
		USA	1		Gunnar Wang, FFIE
			1		Karianne Pran, FFIE
			1		Øystein Farsund, FFIE
1		SFK, Teknisk avdeling	1		Geir Sagvolden, FFIE
1		V/ Steinar Nilsen			
1		V/ Atle Sannes	10		Arkiv, FFIE
		Postboks 3, Haakonsvern	1		FFI-veven
		5086 BERGEN			