RICH Sbus Monitoring

Nick Brönn

School of Physics, Georgia Institute of Technology, Atlanta, GA 30332

Abstract

In the CLEO III experiment at the Wilson Synchrotron Laboratory at Cornell University, the detector used to measure the velocity of charged particles is the Ring Imaging Cherenkov (RICH) detector. The RICH detector measures the conic section of the Cherenkov light cone emitted by a charged particle in a LiF radiator. This is especially useful for distinguishing between charged kaons and charged pions. The Sbus monitors critical parameters of the RICH detector. In the past this information was not useful because it was not recorded or displayed continuously by any software. Two existing software programs were then expanded to allow for this kind of monitoring. Now the data read from the Sbus is recorded and displayed allowing behavior of these critical parameters to be examined.

Introduction

Momentum is the simplest property of a particle to measure in high-energy physics. However in order to unambiguously determine the identity of the particle observed one must combine this measurement of momentum with the velocity or energy of the particle. Thus the mass of the particle is identified. This data can be used to eliminate wrong track hypotheses (Fig. 1). RICH is the detector in CLEO used to record a particle's velocity. It does this by measuring the opening angle of the Cherenkov light cone emitted by a relativistic particle during its passage through a LiF crystal radiator. The opening angle determines the velocity by

$$cos\Theta_{Ch} = \frac{c}{vn} \tag{1}$$

where n is the index of refraction.

When a Cherenkov photon enters the wire chamber in the RICH detector, a small signal consisting of only a few electrons is created. This signal is then amplified in an avalanche process around the wires. The induced electronic signal is then amplified in preamplifiers mounted on the detector. From there the signal is read in with a receiver and digitized on databoards in the VME standard. Stability of the electronic operating parameters is required for the correct performance of the RICH detector. RICH Sbus monitoring ensures the crucial operating parameters of the detector are stable.

Overview of RICH Sbus

The software that establishes a connection with the Sbus databoards is a C++ program called the RICH Sbus Component. The Component reads data from all the channels in the Sbus and sends alarms to the Alarm Manager if these values are out of specified limits. The Component also sets up an interface with a Graphical User Interface (GUI) written in Java called the RICH Sbus GUI. The GUI displays the most current values of the data when a certain crate and board are selected.

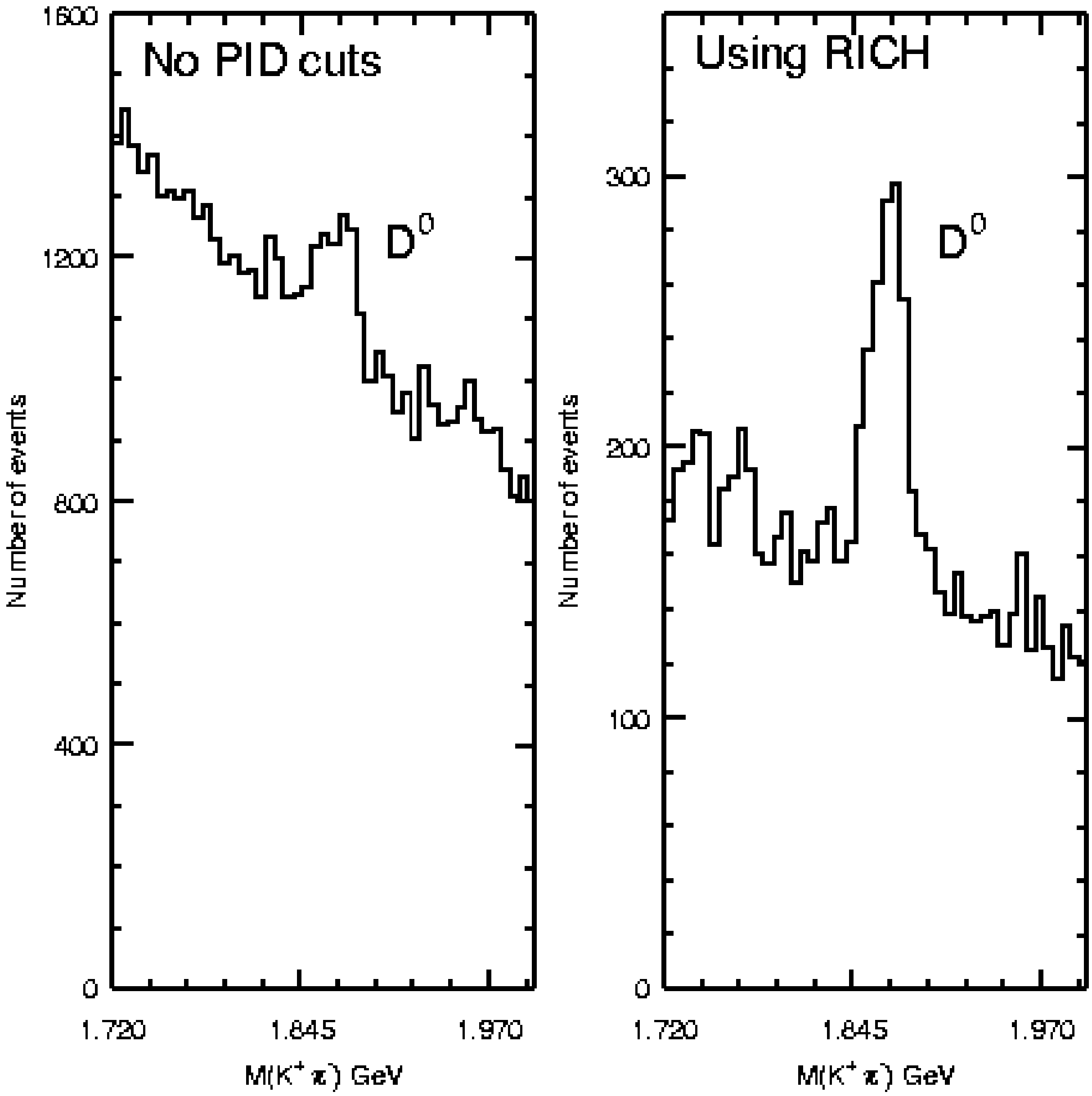

FIGURE 1. Event to mass before and after RICH cuts[1].

This monitoring software was not completely developed and did not provide much use in monitoring. The Component only sent one type of alarm (fatal) whenever a channel's data was out of range. The data was not kept past displaying it on the GUI. The GUI was not updated when it received new data from the Component. It also did not notify which channels were out of range, and how much out of range they were. And finally the GUI was not written in the Object Oriented paradigm that Java was built for.

The goal of this project is to provide easy monitoring of critical RICH parameters. This is accomplished by many software modifications to the two existing programs. Now RICH Sbus monitoring is incorporated into the CLEO online data acquisition and control system, and shifters can easily check for out of range channels and respond to them. Monitoring these critical parameters ensures the quality of the data obtained by the RICH detector.

Software Modifications

The first area that was improved upon was the recording of the Sbus data. The GUI was programmed to write a file of data to disk after each scan was completed by the component. This data was formatted such that it could be read by the data analysis program mn_fit. Later it was decided that reliability would be increased if instead the Component wrote the data to disk. This change was implemented.

The Component also needed the capability to send different kinds of alarms to the Alarm Manager, depending on how much out of range a channel's value is. This involved going deeper into the source code of the component. Finally, it was found that different kinds of alarms could be sent by each channel if the Alarm ID List was an array. The appropriate changes where made so that each different kind of alarm was initialized for each channel. These alarms are, in order of increasing severity: Info, Warning, Error, Severe, and Fatal. Each alarm corresponds to a specified range in the parameters.

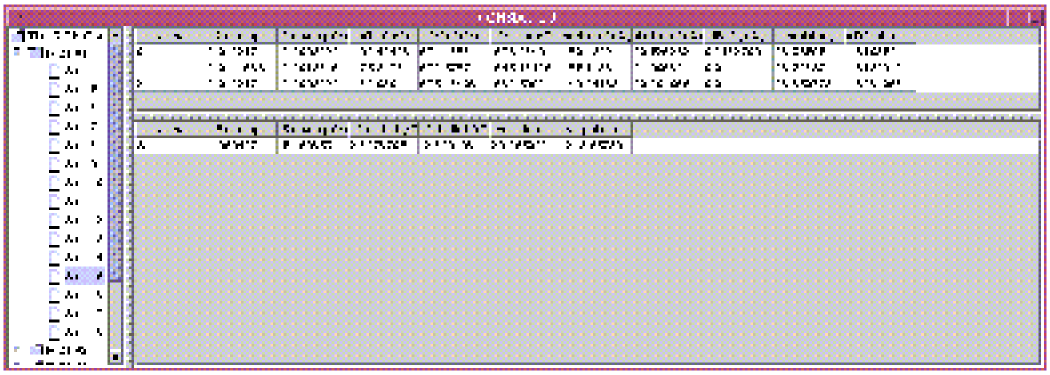

FIGURE 2. The original RICH Sbus GUI.

Then began the modification of the RICH Sbus GUI. A drawback of the original GUI (Fig. 2) that warranted a complete overhaul was the fact that the original did not utilize the Object Oriented paradigm. In the original the same class was used to establish a connection with the Component and display the GUI. This kind of design would not benefit the addition of extra features. Hence the class RICHSbusGUI was rewritten to only establish the CORBA connection with the Component, and then call an instance of a class that would display the appropriate information.

Class Monitor was written to set up the layout of the GUI (Fig. 3), as well as display alarms. The basic layout behind this is each crate is represented as a grid (or double array) of buttons. Each button corresponds to a specific board and cell in that crate, and each is

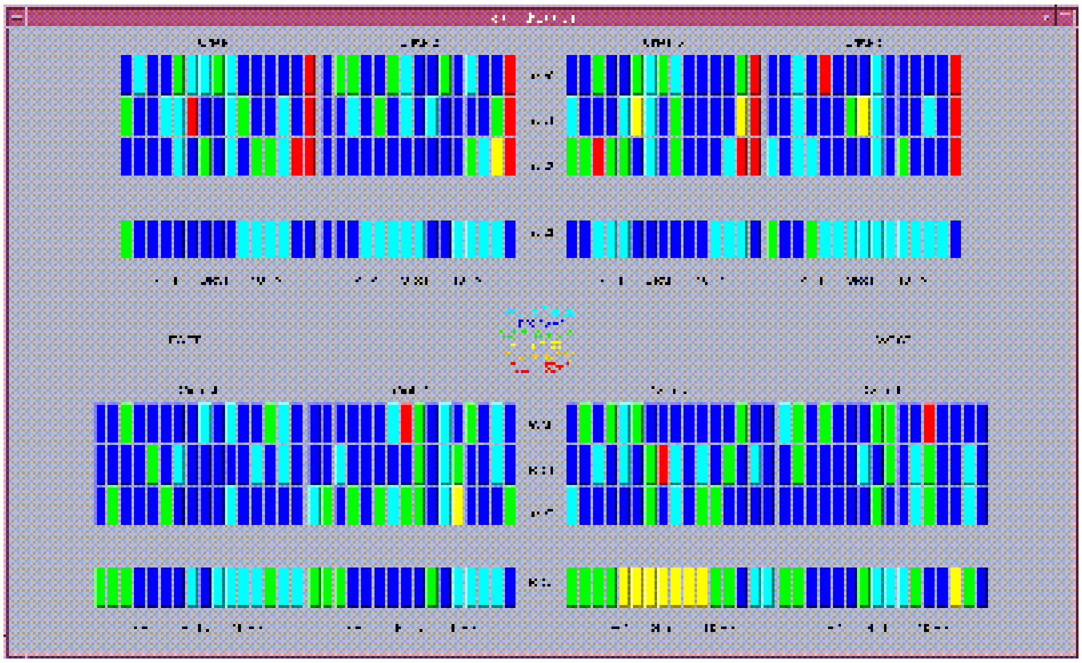

FIGURE 3. The layout of the modified GUI. Each button represents a cell, which contains either 6 or 10 channels. The colors correspond to different levels of alarm.

colored to the color of highest severity of alarm in that cell. Whenever Class RICHSbusGUI receives new data from the component, methods are called that update all the colors and store the current data in Class Monitor. Hence the modified software provides continuous error monitoring.

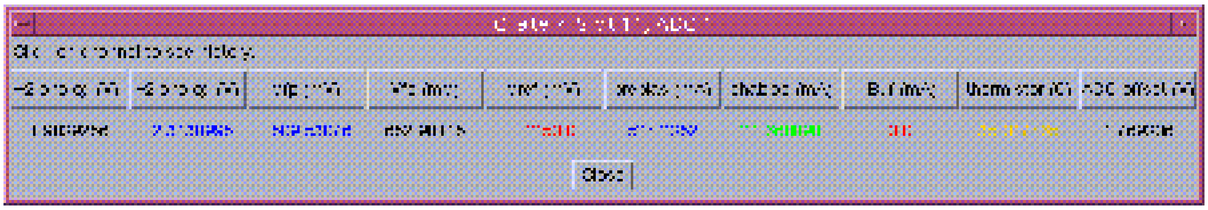

FIGURE 4. Window created by Class ADCWindow. Each button is a channel with its current data value below. Channel data is colored according to level of alarm.

When one of the buttons representing a cell is pressed, an instance of Class ADCWindow is created. This causes a window to pop up displaying the current data for each channel (Fig. 4). It also colors each value according to how much out of range it is, with black meaning the data is within range. Each one of the channel names can also be pressed to display the history of that channel. Because this software is written in an object oriented fashion, no code is necessary to keep track of these windows once they are open. Hence as many of them can be open as needed.

Class History extends from Canvas (API) to plot approximately the past 24 hours of history for a specific channel (Fig. 5). This class also performs the I/O necessary to extract data from the files written by the Component. This is done with some rudimentary parsing techniques in Java 1.1. This also makes use of Class DataPoint, which is essentially a data

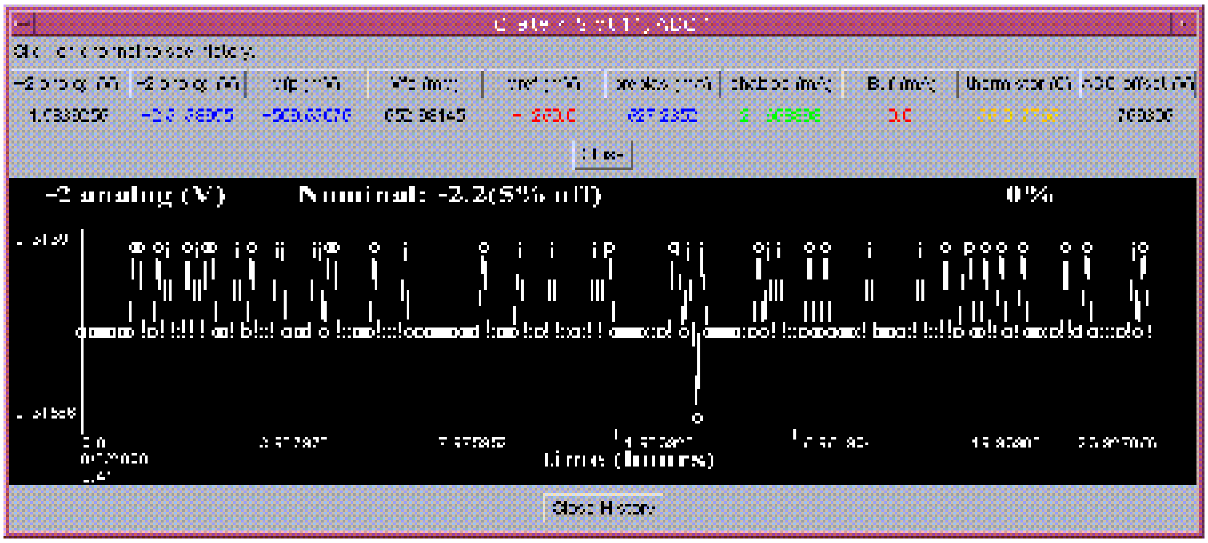

FIGURE 5. A Canvas with history plotted is added to the expanded ADC Window when a channel is clicked.

structure that has a channel's data value and the time it was recorded. From there Class History calculates where on the plot the data point should appear. It also calculates the overall variance in the past 24 hours, and the percent off the nominal value the last data value is.

In order to prevent hard coding all of the nominal data value and ranges into the GUI, Interface Alarm was created to store that specific data. Interface Alarm is implemented by Classes Monitor, ADCWindow, and History. The interface also keeps the kind of errors and their corresponding colors. Interface Alarm can be compiled by itself, simplifying the process of updating error data.

Results

These two modified pieces of software can now be used to provide online monitoring of readout electronics for the RICH detector. Strange behavior in some of these electronics, known as "hot chains," (Fig. 6) have been observed and must be remedied. These are chains of chips that experience a simultaneous fluctuation of their pedestal. This rules out the possibility of statistical fluctuation and are caused by a problem in temperature, setup, hardware, or software. The aim of monitoring is to discover where the hot chains originate from and then eliminate the problem.

Pedestals are the signal levels for each electronic channel if no input signal is applied, thus they give a "zero-line" for each channel. They are calibrated in calibration runs where the readout is analyzed with no input on the electronics. It must be made sure that the electronics are working within a good range in parameter space.

The first issue that was addressed with this new software was determined to be caused by the constants-loading process into the crates. This was due to an error in setup and has since been remedied. Although the GUI was not used in this resolution, the GUI will provide the same kind of information used to solve this problem, and it will provide it online without

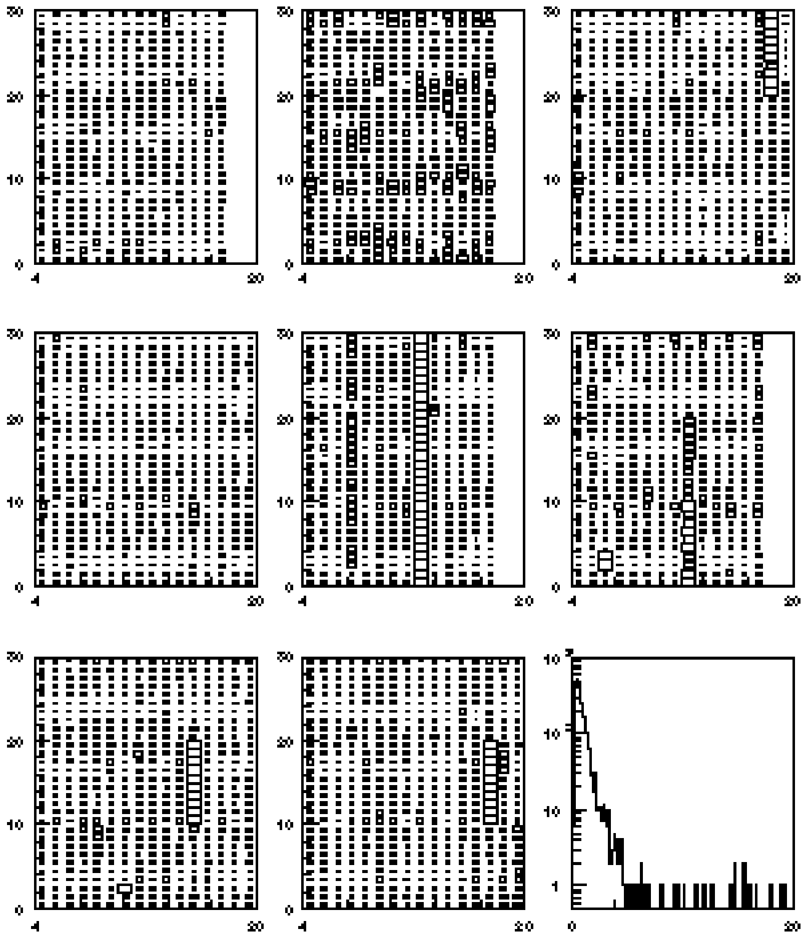

FIGURE 6. Hot chains can be seen as the vertical rows of larger rectangles for each of the eight crates. The last figure shows deviation from the "zero-line"[2].

the need for other data analysis tools.

Conclusion

It was the goal of this project to develop useful software that will aide in the monitoring of the RICH detector's critical parameters. This goal has been achieved. The source for the Component has already been uploaded to the data acquisition and monitoring system and the source for the GUI soon will be as well. The Component and GUI will provide invaluable monitoring instrument to ensure the correct and accurate operation of the RICH detector.

There is one unresolved issue left with RICH Sbus Monitoring. This is the problem of a memory error encountered sporadically during the operation of the GUI. The Component cannot pass such a huge amount of data to the GUI very easily, so instead it passes a reference to the location of the data in memory. However sometimes when the GUI starts, the initial reference to the data is invalid, causing the GUI to read garbage and thus experiencing a memory error. This error kills the monitoring thread in the GUI that receives data from the Component. Hence the specific communication between the Component and the GUI must be explored and corrected.

Acknowledgments

I would like to thank Georg Viehhauser and Ray Mountain of Syracuse University for their guidance and direction of this project, Andreas Wolf of Syracuse University for his computer expertise, and David Cassel of Cornell University for making this experience possible. This work was supported by the National Science Foundation REU grant PHY-9731882 and research grant PHY-9809799.

References

1. Figure courtesy of Tomasz Skwarnicki.
2. Figure courtesy of Ray Mountain.