Überwachungs- und Steuerungssoftware der Betriebsparameter des VDC-Systems bei CMS

von

Lukas Koch

Bachelorarbeit in Physik

vorgelegt der

Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen

im August 2010

angefertigt im

III. Physikalischen Institut A

bei

Prof. Dr. Thomas Hebbeker

Zusammenfassung

Beim CMS-Detektor bei CERN kommen Gasdetektoren zum Myontracking zum Einsatz. Um die optimale Auflösung dieser Detektoren zu gewährleisten, wird an der RWTH Aachen zur Zeit ein VDC-System entwickelt, welches die Driftgeschwindigkeit von Elektronen in dem verwendeten Gas messen soll.

Im Rahmen dieser Bachelorarbeit wurden Programme zur Betriebsparameterüberwachung der VDCs entwickelt. Das Parameterkontrollsystem der VDCs wird skizziert. Die für die Überwachung der Betriebsparameter verantwortlichen Programme, sowie das zugrundeliegende Konzept werden vorgestellt. Anschließend werden einige, mit den neu entwickelten Programmen aufgenommene, Daten analysiert und aus ihnen Empfehlungen für Toleranzgrenzen für Gasfluss- und Druckschwankungen hergeleitet. Außerdem wird die Ventilpositionsauslese analysiert und gezeigt, dass sie bei den aktuellen Laborbedingungen gut funktioniert. Bei drei Ventilen wurde jedoch eine (sehr wahrscheinlich hardwarebedingte) Unregelmäßigkeit beobachtet, die das Bestimmen der Positionen bei diesen Ventilen teilweise verhindert.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation . . . 1
1.2 Die VDCs . . . 1

2 Das System **3**
2.1 Das Gassystem . . . 3
2.1.1 Fluss- und Druckregelung . . . 3
2.1.2 Die Ventilmatrix . . . 4
2.2 Die Temperatursensoren . . . 5
2.3 Die Hochspannungsversorgung . . . 7

3 Das Softwarekonzept **7**
3.1 Ziele . . . 7
3.2 Umsetzung . . . 7

4 Die Programme **8**
4.1 MASTERD & MINID . . . 8
4.2 HVREADOUT . . . 9
4.3 GASMATRIX . . . 10
4.4 FLOWBUS . . . 11
4.5 PBREADOUT . . . 11

5 Analyse aufgenommener Daten **12**
5.1 Druck- und Flussstabilität . . . 12
5.2 Die Ventilauslese . . . 14

6 Integration des Systems bei CMS **19**

7 Fazit und Ausblick **20**

Literatur **21**

Eidesstattliche Erklärung **22**

A Plots **23**
A.1 Druckstabilitätsmessung . . . 23
A.2 Flussstabilitätsmessung . . . 25
A.3 Ventilauslese . . . 27

B Quellcodes **71**
B.1 MASTERD & MINID . . . 71
B.2 GASMATRIX . . . 74
B.3 FLOWBUS . . . 89
B.4 PBREADOUT . . . 96

Abbildungsverzeichnis

1 Aufbau einer Driftkammer am CMS . . . 2
2 Prinzipskizze der VDCs . . . 2
3 Die Gasregelung . . . 4
4 Die Ventilmatrix . . . 5
5 Die Ventilauslese . . . 6
6 Hintergrundprozesskontrolle . . . 8
7 Informationsfluss: MASTERD . . . 9
8 Messung der Flussstabilität . . . 12
9 Messung der Druckstabilität . . . 13
10 Temperaturdifferenzverlauf während eines Heizzyklusses . . . 14
11 Temperaturdifferenzverläufe von 289 Heizzyklen überlagert . . . 15
12 Differenzen der Maximaltemperaturen der 289 Heizzyklen über die Zeit . . . 16
13 Histogramm der Differenzen der Maximaltemperaturen der 289 Heizzyklen . . . 16
14 Auffälliges Heizverhalten der Ventile 21, 46 und 64 . . . 17
15 Messschranktemperatur über die Zeit . . . 18
16 Momentaner (sehr früher) Stand der Entwicklung der grafischen Benutzeroberfläche . . . 19

Tabellenverzeichnis

1 Übersicht MASTERD . . . 8
2 Übersicht MINID . . . 9
3 Übersicht HVREADOUT . . . 9
4 Übersicht GASMATRIX . . . 10
5 Übersicht FLOWBUS . . . 11
6 Übersicht PBREADOUT . . . 11
7 Gemessene Gasfluss- und Druckstabilitäten . . . 13
8 Empfohlene Toleranzen für Gasfluss und -druck . . . 13
9 Mittlere Differenz der Maximaltemperaturen der verschiedenen Ventile . . . 15
10 Mittlere Differenz der Maximaltemperaturen der verschiedenen Ventile in anderer Position . . . 17
11 Mittlere Differenz der Maximaltemperaturen der verschiedenen Ventile in mittlerer Position . . . 18

1 Einleitung

1.1 Motivation

Beim CMS-Experiment[1] des LHCs[2] bei CERN[3] kommen Gasdetektoren zum Einsatz, welche zum Tracking von Myonen benutzt werden. Die eingesetzten Kathodenstreifenkammern[4] erhalten die Ortsinformation des Teilchendurchgangs direkt aus den Signalstärken an den Anodendrähten und Kathodenstreifen durch Gewichtung mit den Signalamplituden.

Für eine genaue Ortsbestimmung der Teilchendurchgänge in den Driftkammern[5] muss aber sowohl die Driftzeit, als auch die Driftgeschwindigkeit der Elektronen im Gas bekannt sein.[6]

Die Bestimmung der Driftzeit erfolgt durch die Messung der Zeitdifferenz zwischen einem Triggersignal[7] und dem eigentlichen Anodensignal der Driftkammern (Abb. 1).

Die Driftgeschwindigkeit der Elektronen im Gas der Kammer hängt von der lokalen Feldstärke und den Eigenschaften des Gases ab.

Das elektrische Feld lässt sich durch Simulationen sehr genau berechnen und durch feldformende Elemente in den Driftkammern gut festlegen.

Das Gas – bestehend aus einer Mischung aus 85% Argon und 15% Kohlendioxid – wird unter kontrollierten Bedingungen und leichtem Überdruck ca. einmal pro Tag ausgetauscht[8]. Dies sollte eine Verunreinigung verhindern.

Um Fluktuationen in der Driftgeschwindigkeit, welche die Messungen am CMS beeinflussen könnten, zu registrieren, wird Gas von verschiedenen Stellen[9] des Myonsystems zu VDCs[10] geleitet. Diese messen laufend die Driftgeschwindigkeit von Elektronen im Kammergas unter Bedingungen, die denen in den Driftkammern ähneln und können Alarm geben, wenn sich diese z.B. durch Gasverunreinigungen ändern.

Da es sich bei den VDCs um ein Überwachungssystem handelt, das kontinuierlich die Driftgeschwindigkeit im Gas registriert und damit den reibungslosen Ablauf der Messungen am DT-System garantieren soll, muss die Selbstkontrolle des VDC-Systems so ausgelegt sein, dass Störungen und Unregelmäßigkeiten zuverlässig erkannt werden. Außerdem müssen im Falle einer Störung alle nötigen Informationen bereitstehen, um die Ursache der Störung schnell ausfindig zu machen. Das Durchführen von Testmessungen und Diagnose soll soweit wie möglich aus der Ferne möglich sein. Die vorliegende Arbeit befasst sich mit der Software, die die Betriebsparameter des VDC-Systems steuert und überwacht.

1.2 Die VDCs

Abbildung 2 zeigt das Funktionsprinzip der VDC. Die Kammer ist mit dem zu untersuchenden Gas gefüllt. Zwischen Kathode und Anode wird eine Spannung angelegt, die ein elektrisches Feld erzeugt. Mit Hilfe der Feldformungselektroden wird dieses in der sensitiven Region möglichst homogen eingestellt.

Zwei Betastrahler (Sr90) sind in die Wand der VDC eingelassen und ihre Strahlung wird mit Kollimatoren fokussiert. Ein Teil der Elektronen durchquert die Kammer und löst in der Szintillator-Faser am anderen Ende

[1] Compact Muon Solenoid – Für nähere Informationen siehe z.B. [1].

[2] Large Hadron Collider

[3] Conseil Européen pour la Recherche Nucléaire – Europäische Organisation für Kernforschung

[4] Auch Cathode Strip Chambers (CSC) genannt

[5] Auch Drift Tubes (DT) genannt – Bei CMS gibt es 250 große DTs mit insgesamt ca. 170000 Driftzellen, wie sie in Abb. 1 dargestellt ist. Sie sind in 5 Gruppen von je 50 DT in 5 unabhängigen radförmigen Segmente des Magnetjochs montiert. Für jedes Rad gibt es eine eigene Gasregelung und eigene Gasanalysegeräte zur Messung der Feuchte, des Sauerstoffs und der Driftgeschwindigkeit im Gas. Letzteres ist kommerziell nicht erhältlich und wird deshalb in Form des VDC-Systems zur Zeit an der RWTH Aachen entwickelt.[2]

[6] Für ausführlichere Informationen zur Teilchendetektion mit Driftkammern mit siehe z.B. [3].

[7] Resistive Plate Chambers (RPC), CSCs, DTs und Kalorimeter erzeugen Triggersignale, welche von einer übergeordneten Logik bewertet werden. Diese liefert dann das finale Triggersignal.

[8] Das gebrauchte Gas wird gereinigt und wieder ins System zurückgeführt. Pro Umlauf muss etwa 10% des Gases durch Frischgas ersetzt werden.

[9] Das Gas kann sowohl vor als auch hinter den verschiedenen Kammern entnommen werden, um genauer einzugrenzen wo eine eventuelle Verunreinigung des Gases stattfindet. Es werden insgesamt bis zu 6 VDCs[10] messen, so dass eine gleichzeitige Kontrolle aller 5 Räder des Myonsystems und des Frischgases möglich ist.

[10] Velocity Drift Chambers

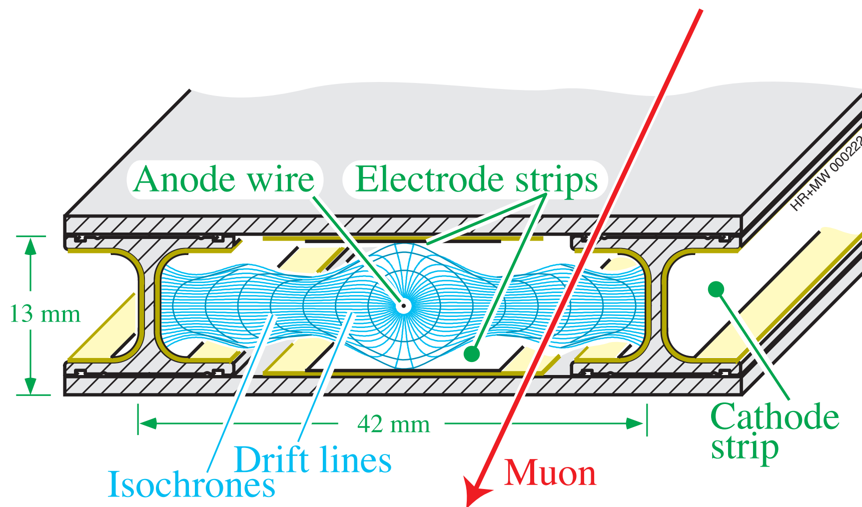


Abbildung 1: Aufbau einer Driftkammer am CMS[2]

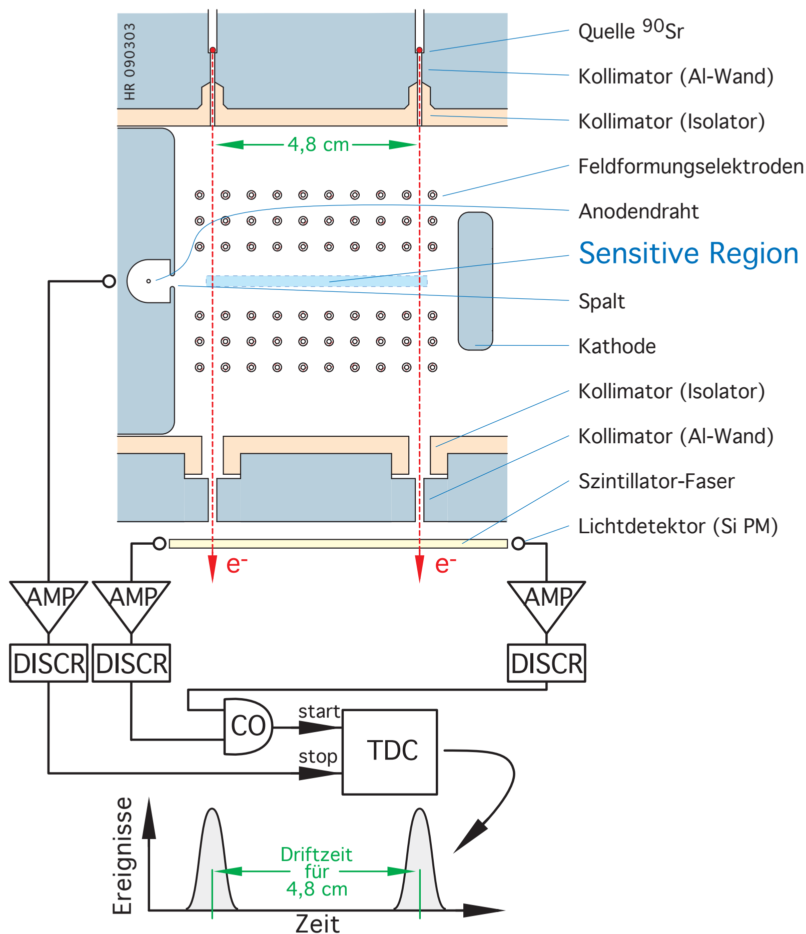


Abbildung 2: Prinzipskizze der VDCs[2] – AMP = Verstärker, DISCR = Diskriminator, CO = Koinzidenz, TDC = Zeit-Digital-Wandler

einen Lichtblitz aus. Wird dieser von den beiden SiPMs registriert, so wird am TDC das Startsignal ausgelöst und er beginnt die Zeit zu messen. Durch die Verwendung von zwei SiPMs[11] kann ihr Rauschen mit einer Koinzidenzschaltung stark unterdrückt werden. Nur Signale, die von beiden SiPMs gleichzeitig an der Koinzidenz antreffen, werden als Startsignal an den TDC[12] weitergegeben.

Auf seinem Weg durch das Gas hat das von der Strahlenquelle emittierte Elektron Gasmoleküle ionisiert. Die erzeugten Elektron-Ionen-Paare werden durch das elektrische Feld von einander getrennt und die Elektronen bewegen sich in Richtung der Anode und die Ionen in Richtung der Kathode. Elektronen, die außerhalb der sensitiven Region herausgeschlagen wurden, erreichen nicht den 2,5 mm breiten Schlitz vor dem Anodendraht und werden von den Feldformungselektroden oder der geerdeten Kammerwand aufgenommen. Elektronen, die in der sensitiven Region herausgeschlagen wurden, bewegen sich bis zur Anode, wo sie durch das dort starke Feld, eine Elektronenlawine auslösen und ein messbares Signal produzieren. Dieses Signal stoppt den TDC und damit wurde die Flugzeit der herausgeschlagenen Elektronen gemessen.

Je nach dem, aus welcher der beiden Quellen das emittierte Elektron kam, haben die herausgeschlagenen Elektronen eine unterschiedlich weite Strecke bis zur Anode zurückzulegen und ihre Flugzeiten sind dementsprechend auch unterschiedlich lang. Trägt man alle gemessenen Flugzeiten in einem Histogramm auf, so zeigen sich zwei Häufungen, die den beiden Strahlenquellen entsprechen.

Für die Driftgeschwindigkeit der Elektronen im Gas gilt $v_{drift} \propto E/N$ [4][5]. Hierbei ist N die Teilchendichte des Gases, welche innerhalb der Kammer konstant ist. Um nun die in der sensitiven Region konstante[13] Driftgeschwindigkeit zu messen, benötigen wir nur die Differenz (!) der Flugzeiten der den beiden Quellen Entsprechenden Elektronen Δt und die Streckendifferenz Δs. Mit $v_{drift} = \Delta s/\Delta t$ lässt sich die Geschwindigkeit einfach ausrechnen.

Der Vorteil dieser Methode ist, dass bei der Verwendung der Zeitdifferenz viele der systematischen Fehler auf die Flugzeitmessung[14] und Inhomogenitäten nahe der Anode neutralisiert werden, weil sie die Zeitmessungen bei beiden Quellen gleichermaßen beeinflussen und bei der Differenzbildung herausfallen.

Für weitere Informationen über die Funktion und Entwicklung der VDCs siehe z.B. [6], [7] und [5].

2 Das System

2.1 Das Gassystem

2.1.1 Fluss- und Druckregelung

Da das Gas ungeregelt vom CMS-Myonensystem an das VDC-System geliefert wird, ist jede der VDCs mit einem Fluss[15]- und einem Druckregler[16] ausgestattet (siehe Abb. 3). Die Flussregelung sorgt für einen konstanten Gasaustausch in den Kammern, während die Druckregelung für konstante Druckbedingungen in den Testkammern sorgt, womit reproduzierbare Geschwindigkeitsmessungen erst möglich gemacht werden. Es wird ein leichter Überdruck gegenüber der Atmosphäre eingestellt, um eine Verunreinigung des zu testenden Gases durch eventuelle Undichtigkeiten in der Kammer zu vermeiden.

Der Fluss kann durch die lokale Regelung im Bereich zwischen 0 l/h und 20 l/h eingestellt werden. Der Solldruck kann zwischen 0 mbar und 1100 mbar (absolut) gewählt werden, wobei natürlich kein Druck unterhalb des Atmosphärendrucks erreicht werden kann. Zur Zeit ist ein Regelbetrieb bei 5,00 l/h und 1000 mbar vorgesehen. Während des Probebetriebs in Aachen ist der Fluss auf 3,00 l/h eingestellt.

Sämtliche Fluss- und Druckregler werden von einer Basisstation[17] angesteuert. Diese wiederum ist per serieller Schnittstelle an den PC angeschlossen und wird mittels der – im Rahmen dieser Arbeit entwickelten – Software FLOWBUS bedient. FLOWBUS dient neben dem Einstellen der Sollwerte auch dem Auslesen und Aufzeichnen der Istwerte, welche ebenfalls von den internen Sensoren der Regler bereitgestellt werden (siehe Abschnitt 4.4).

[11] Silizium Photomultiplier – Ein Lichtdetektor auf Halbleiterbasis
[12] Time to Digital Converter – Ein Zeitmesser mit einer Auflösung von ca. 0,8 ns
[13] Das Feld ist in diesem Bereich ja homogen.
[14] Z.B. Laufzeiten in der Elektronik
[15] Bronkhorst EL-FLOW, HI-TEC Modell F-201C-RGD-33-Z Multi-Bus DMFC
[16] Bronkhorst EL-PRESS, HI-TEC Modell P-702C-1K1A-RGD-33-Z Multi-Bus DEPC
[17] Bronkhorst E-7400 und E-7500-05 Auswertesystem

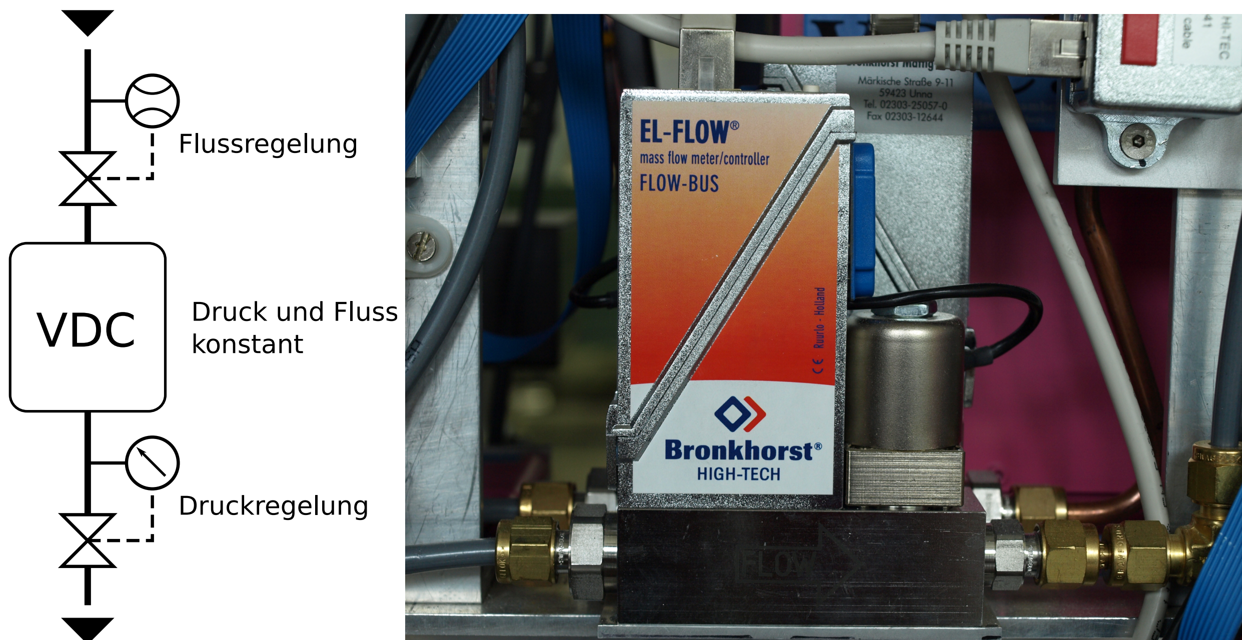


Abbildung 3: Die Gasregelung – Auf dem Foto ist im Vordergrund die Fluss- und dahinter die Druckregelung zu sehen.

Zusätzlich zu den digital auslesbaren Sensoren in den Druckreglern sind zur Drucküberwachung bei jeder Kammer noch zwei Differenz- und ein Absolutdrucksensor installiert[18]. Diese liefern als Signal eine Spannung, welche mittels eines Analog-Digital-Wandlers[19] erfasst und anschließend von der hierfür entwickelten Software PBREADOUT (siehe Abschnitt 4.5) in einen Druckwert umgewandelt wird.

Die analogen Sensoren weisen untereinander Abweichungen auf, die im Bereich der vom Hersteller angegebenen Genauigkeit liegt (bis zu 20% vom Maximalwert). Ein Sensor für sich genommen liefert aber reproduzierbar eine Spannung, die bei gegebenem Druck deutlich weniger schwankt. Nimmt man also Kalibrationskurven für die einzelnen Sensoren auf und wandelt diese in Lookup-Tabellen um, kann man die Genauigkeit der Druckmessung deutlich erhöhen (siehe [8]).

2.1.2 Die Ventilmatrix

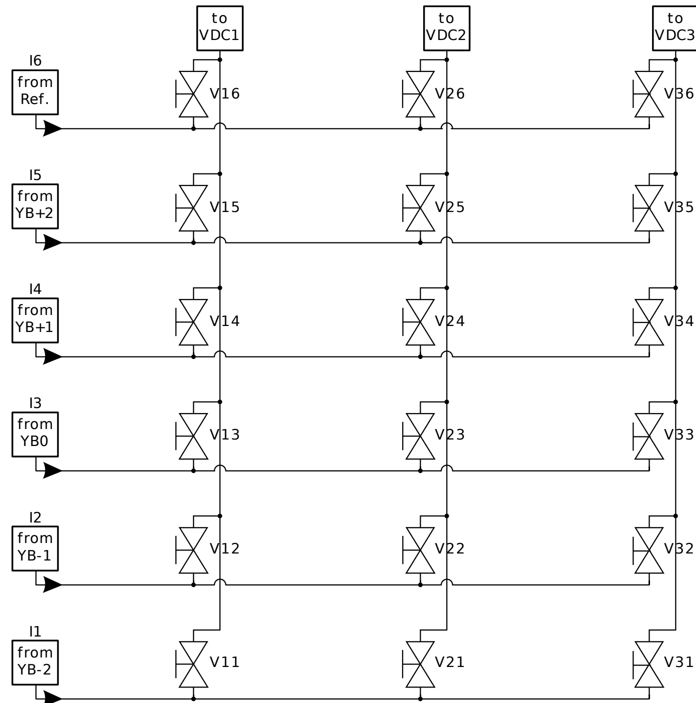
Das VDC-System ist dazu ausgelegt simultan Gas aus bis zu 6 verschiedenen Quellen analysieren zu können. Die ersten 5 Quellen entsprechen den Rädern -2 bis 2 des CMS-Myonsystems (siehe 1.1) und bei der 6. Quelle kann zwischen einem Referenzgas mit genau bekannter Zusammensetzung und dem Frischgas, welches auch in das Gassystem eingespeist wird, gewählt werden.

Um beim Ausfall einer Kammer flexibel zu sein und um eventuelle systematische Unterschiede der Kammern von tatsächlichen Unterschieden der Driftgeschwindigkeiten unterscheiden zu können, kann mit Hilfe einer Ventilmatrix an der Rückseite des Messschranks die Gasquelle jeder einzelnen Kammer individuell von Hand ausgewählt werden (siehe Abb. 4). Es ist also nicht nötig das Rohrsystem zu öffnen und dabei mit Luft zu verunreinigen, wenn man das Gas von einer Quelle zu einer anderen VDC umleiten will.

Bei dieser Flexibilität ist es jedoch wichtig sicherzustellen, dass jede Kammer auch genau das Gas erhält, welches sie untersuchen soll. Die Ventile können zwar nur vor Ort per Hand umgestellt werden, durch Flüchtigkeit oder Missverständnissen kann es aber trotzdem dazu kommen, dass die Ventile nicht so eingestellt werden, wie es im aktuellen Messvorgang vorgesehen ist. Um solche Fehler dennoch zu entdecken wurde ein System entwickelt, welches trotz eventuell vorhandener Magnetfelder (keine magnetische Auslese möglich) oder Staubansammlungen (keine optische Auslese möglich) eine zuverlässige Bestimmung der Ventilpositionen ermöglicht.

An jedem Ventil befinden sich zwei Temperatursensoren[20], einer dort, wo sich der Hebel bei geöffnetem Ventil

[18] Freescale Semiconductors, Inc. MPX5010DP (-100 mbar bis 100 mbar), MPX5050DP (-500 mbar bis 500 mbar) und MPXA6115AC6U (bis 1150 mbar absolut) – Die gleichen Differenzdrucksensoren werden auch zur Drucküberwachung an den DTs des Myonsystems benutzt.
[19] National Instruments NI-PCI-6229
[20] Dallas DS1820



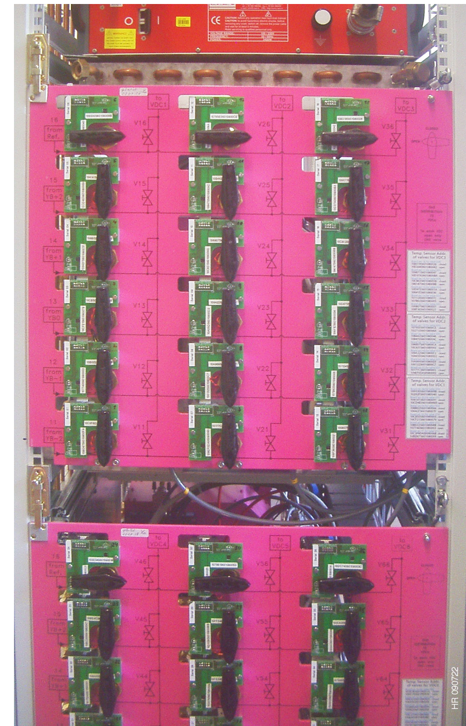


Abbildung 4: Die Ventilmatrix[2] – Jede Zeile entspricht einer Gasquelle (Referenz-/Frischgas und die fünf Räder des Myonsystems), jede Spalte einer VDC. Im Schema sind nur die ersten 3 VDCs abgebildet. Die übrigen 3 sind analog dazu angeschlossen. Außerdem befinden sich Ventile am Auslass jeder VDC (Ventil 17, 27, 37, 47, 57 und 67) und es gibt 2 Ventile um zwischen Frischgas vom DT-System und hochreinem Referenzgas wählen zu können (Ventil 101 und 102).

befindet, genannt Opentemp, und einer dort, wo sich der Hebel bei geschlossenem Ventil befindet, genannt Closetemp (siehe Abbildung 5). Im Hebel selbst ist ein SMD-Widerstand eingelassen, welcher in regelmäßigen Abständen beheizt wird. Anhand der gemessenen Temperaturverläufe an den beiden Temperatursensoren kann nun festgestellt werden, über welchem der beiden Sensoren sich der Hebel befindet, also in welcher Position sich das Ventil befindet. Auch eine undefinierte Ventilstellung (Hebel weder über dem einen noch dem anderen Temperatursensor) lässt sich so feststellen.

Die Temperatursensoren werden über einen 1-Wire-Bus ausgelesen. Die Anbindung an den PC geschieht über einen USB-1-Wire-Adapter[21]. Auch das Heizen wird über den 1-Wire-Bus gesteuert. Digitale IO-Bausteine[22] starten den Heizvorgang, welcher dann eine in der Hardware festgelegte Zeit andauert (monostabile Kippstufe). Eine ebenfalls in der Hardware festgelegte Totzeit, in der keine weiteren Heizvorgänge gestartet werden können, verhindert, dass die Widerstände in den Hebeln durch einen Softwarefehler permanent beheizt und evtl. zerstört werden. Sämtliche Elektronik ist in dem VME-Einschub „VALVE READOUT“ zusammengefasst.[9]

Das Heizen, die Temperaturauslese und die Bestimmung der Ventilpositionen werden von dem Programm GASMATRIX durchgeführt (siehe Abschnitt 4.3).

2.2 Die Temperatursensoren

Zusätzlich zu den Temperatursensoren in der Ventilmatrix sind über das VDC-System noch weitere Sensoren verteilt, die die Temperatur an verschiedenen Stellen überwachen sollen. Sie sind über denselben 1-Wire-Bus an den PC angeschlossen wie die Sensoren in der Matrix und werden auch über die Software GASMATRIX ausgelesen. Es wird für jeden Sensor ein Temperaturbereich definiert, in dem sich die Temperatur befinden darf und eventuelle Abweichungen werden gemeldet (siehe 4.3).

[21]Dallas DS9490R
[22]Dallas DS2413

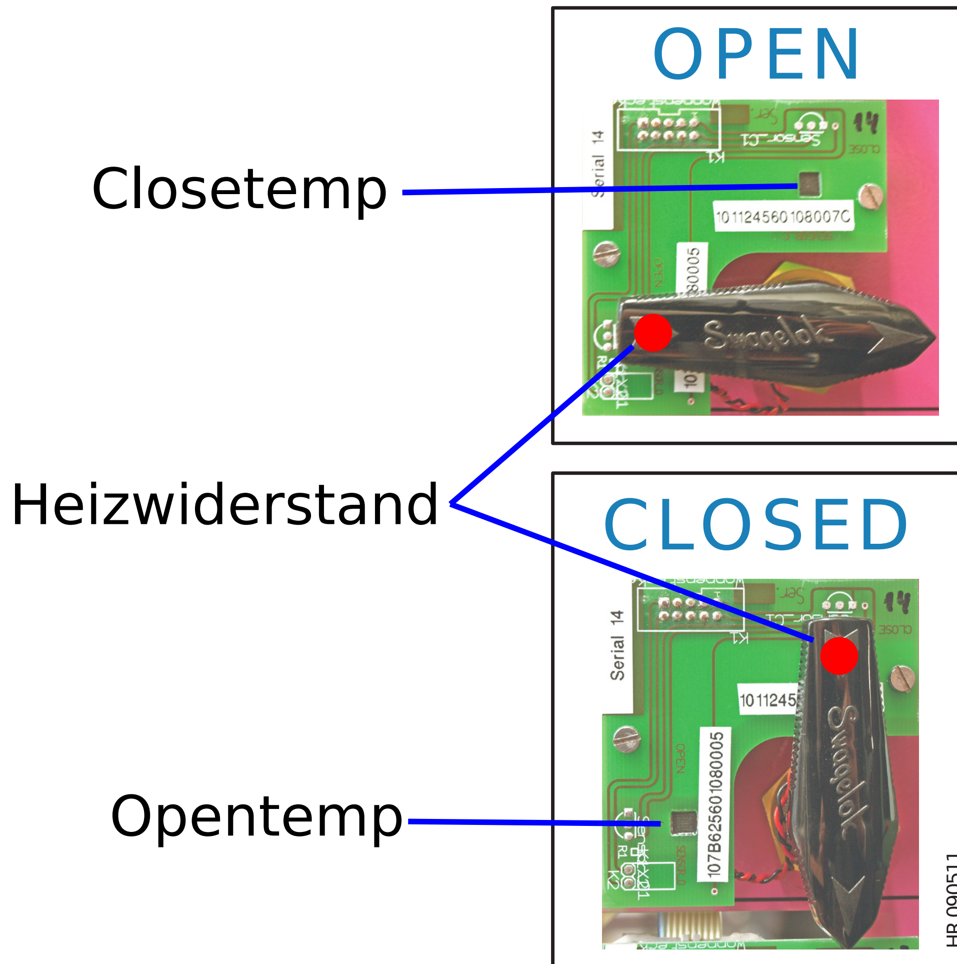


Abbildung 5: Die Ventilauslese[2] – Im Ventilhebel ist ein Heizwiderstand eingelassen, der in regelmäßigen Abständen eingeschaltet wird und je nach Ventilstellung den Sensor Opentemp oder Closetemp beheizt. Aus den unterschiedlichen Temperaturverläufen der Sensoren lässt sich dann die Ventilstellung bestimmen.

2.3 Die Hochspannungsversorgung

Die Spannungsversorgung für die Anoden und Kathoden der VDCs, sowie für die SiPMs wird von einem digital steuerbaren Netzteil[23] bereitgestellt.

Das Setzen der Sollwerte, sowie das Überwachen der Spannungen und Ströme auf mögliche Einbrüche oder Spitzen geschieht mit dem Programm HVREADOUT (siehe Abschnitt 4.2).

3 Das Softwarekonzept

3.1 Ziele

Da es sich bei dem VDC-System um ein Überwachungssystem handelt, welches den regulären Betrieb der 250 DTs des CMS-Myonsystems gewährleisten soll, ist es wichtig, dass es selbst stabil und zuverlässig funktioniert. Außerdem muss das System alle nötigen Informationen bereitstellen, um die Fehlerquelle bei einer eventuellen Störung schnell zu lokalisieren und möglichst zu beheben.

Um dies sicherzustellen wurden folgende Ziele für die Softwareentwicklung beim VDC-System formuliert:

Modularität Die einzelnen Subsysteme (Temperaturauslese, Druckauslese, Darstellung für den Benutzer etc.) operieren weitgehend unabhängig von einander. Wenn ein Subsystem aufgrund eines Softwareproblems ausfällt, bleiben die anderen Subsystem davon unbeeinflusst.

Transparenz Der Informationsfluss zwischen den Programmen geschieht über Textdateien, die jederzeit eingesehen werden können. Dies erleichtert es festzustellen an welcher Stelle im System ein Fehler auftritt.

Zurückverfolgbarkeit Das Loggen von Parametern und eventuellen Fehlern in Textdateien ermöglicht es Unregelmäßigkeiten im Nachhinein zu untersuchen.

Fehlertoleranz Sollte ein unerwarteter Softwarefehler auftreten, der ein Subsystem zum Absturz bringt, so wird dies erkannt und das entsprechende Subsystem automatisch neu gestartet.

3.2 Umsetzung

Sämtliche dauerhaft laufende Programme sind in C/C++ geschriebene Kommandozeilenprogramme, welche im Normalbetrieb als Daemon[24] laufen. Da diese Programme teilweise besondere Rechte benötigen, die der normale Benutzer aus Sicherheitsgründen nicht haben sollte (Zugriff auf einige Systemdateien und die Hardware), laufen sie als eigener Benutzer „messung“.

Jedes der Programme hat eine eigene Konfigurationsdatei, über die es gesteuert wird und welche es regelmäßig ausliest. Alle Konfigurationsdateien benutzen dieselbe Syntax:

```
# Kommentar
$BEFEHL, Anzahl_der_Argumente, Argument_1, Argument_2, ...
$STOP,0
```

Befehle beginnen immer mit einem $ als erstes Zeichen einer Zeile, gefolgt von dem Befehlsnamen, der Anzahl der Argumente, die mit dem Befehl übergeben werden und den Argumenten selbst.

Der Befehl „$STOP“ hat z.B. 0 Argumente und wird von allen Prozessen verstanden. Wird er in die Konfigurationsdatei geschrieben, signalisiert dies dem Programm es soll sich regulär beenden.

Ausgaben erfolgen, je nach Bedarf, über ständig aktualisierte Textdateien, die den aktuellen Status des Subsystems enthalten und über Logdateien, welche den Verlauf des Systems festhalten. Abbildung 6 zeigt das Prinzip der Hintergrundprozesskontrolle.

[23] Caen SY1527

[24] Hintergrundprozesse ohne direkte Benutzerinteraktion unter unixartigen Betriebssystemen wie Scientific Linux

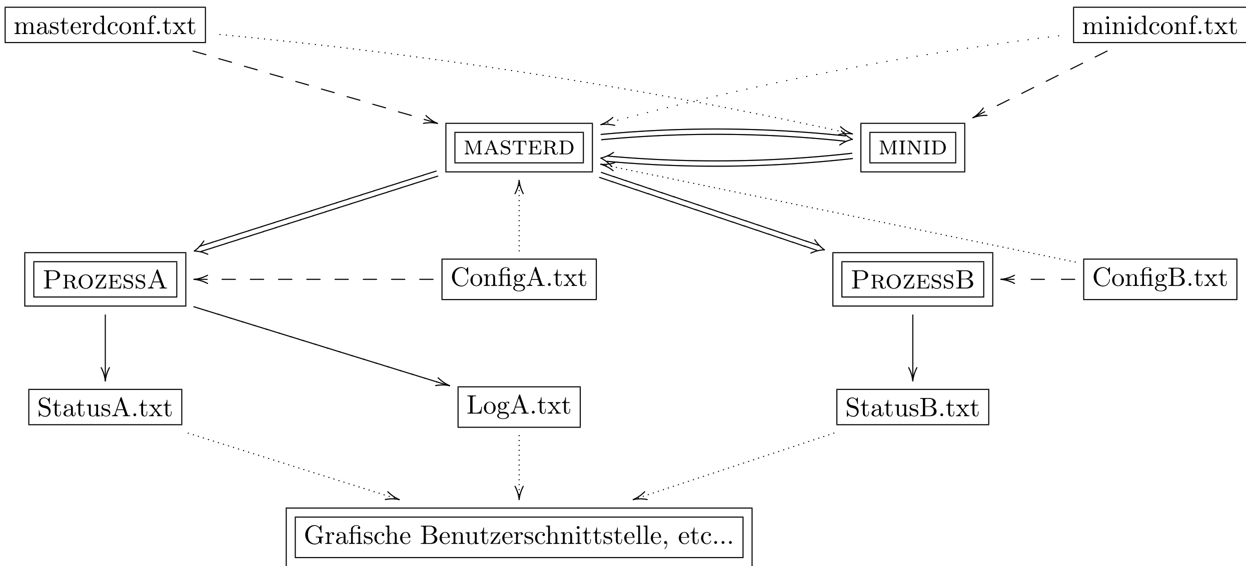


Abbildung 6: Hintergrundprozesskontrolle – Prozesse werden durch Konfigurationsdateien gesteuert (⇢). MASTERD und MINID starten überwachte Prozesse neu (⇒), wenn diese laut Konfigurationsdatei eigentlich laufen müssten (··· ≻). Ausgaben geschehen in Textdateien (→), welche wiederum von anderen Programmen ausgewertet werden können. MINID ist eine Kopie von MASTERD welche allerdings einzig den Prozess MASTERD überwacht. Dies stellt sicher, dass immer alle nötigen Prozesse zur Überwachung des Systems laufen.

Einen besonderen Stellenwert nehmen hierbei die Prozesse MASTERD und MINID ein. Der Masterdaemon (MASTERD) überwacht alle anderen Prozesse und startet sie neu, falls einer irregulär beendet worden sein sollte. Ein irreguläres Beenden wird daran erkannt, dass die Prozesse laut ihrer Konfigurationsdatei noch laufen sollten. Dies wird natürlich auch in einer Logdatei vermerkt, so dass man Softwareausfälle zurückverfolgen kann. Der Minidaemon (MINID) hat im Grunde die selbe Aufgabe wie der Masterdaemon, nur dass er ausschließlich den Masterdaemon überwacht und evtl. neu startet. MASTERD und MINID überwachen sich also gegenseitig und stellen so sicher, dass immer alle nötigen Prozesse laufen.

4 Die Programme

Eine Übersicht über die Dateien und Parameter der Programme befindet sich in den Tabellen 1 bis 6. Der Quellcode der im Rahmen dieser Arbeit entstandenen Programme kann in Anhang B eingesehen werden.

4.1 MASTERD & MINID

Name	MASTERD
Kommandozeilenargumente	-d1 Debuglevel 1
Ausführbare Datei	/home/messung/masterdaemon/masterd
Konfigurationsdatei	/home/messung/masterdaemon/masterdConfig.txt
Logdatei	/home/messung/masterdaemon/log/masterdLog.txt

Tabelle 1: Übersicht MASTERD

Die Masterdaemons MASTERD und MINID überwachen die anderen Prozesse und starten sie neu, wenn es nötig sein sollte. Da MINID identisch zu MASTERD ist und der einzige Unterschied darin besteht, dass MINID alleine

Name	MINID
Kommandozeilenargumente	-d1 Debuglevel 1
Ausführbare Datei	/home/messung/minidaemon/minid
Konfigurationsdatei	/home/messung/minidaemon/minidConfig.txt
Logdatei	/home/messung/minidaemon/log/minidLog.txt

Tabelle 2: Übersicht MINID

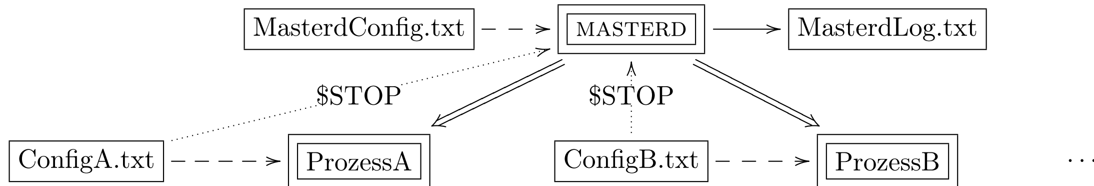


Abbildung 7: Informationsfluss: MASTERD

zur Überwachung des MASTERD-Prozesses konfiguriert ist, wird im Folgenden nur von MASTERD die Rede sein. Alle Angaben gelten analog für MINID.

In der Konfigurationsdatei von MASTERD werden die zu überwachenden Prozesse definiert. Es müssen der Prozessname, der auszuführende Befehl und die Konfigurationsdatei des Prozesses angegeben werden.

In regelmäßigen Abständen werden die angegebenen Konfigurationsdateien nach dem $STOP-Befehl durchsucht. Wird dieser nicht gefunden, überprüft MASTERD ob der zugehörige Prozess noch läuft, indem er die aktuelle Prozessliste[25] nach dem angegebenen Namen durchsucht. Wird auch dieser nicht gefunden, führt das Programm den Befehl aus, der in der Konfigurationsdatei angegeben wurde. In einem solchen Fall wird dies auch in der Log-Datei 'MasterdLog.txt' vermerkt (Abb. 7).

Auf diese Art und Weise können sämtliche vom Masterdaemon überwachten Programme alleine über ihre Konfigurationsdatei gestartet, gesteuert und gestoppt werden. Da MASTERD beim Systemstart automatisch als Benutzer „messung“ gestartet wird und der Benutzer an alle aufgerufenen Prozesse vererbt wird, werden die so gesteuerten Prozesse auch automatisch mit den richtigen Berechtigungen gestartet.

Zur Fehleranalyse kann MASTERD auch mit der Option -d1 aufgerufen werden. Sie verhindert, dass der Prozess als Daemon startet, so dass er Debuginformationen[26] in der Kommandozeile ausgeben kann.

4.2 HVREADOUT

Name	HVREADOUT
Kommandozeilenargumente	-D Als Daemon starten
Ausführbare Datei	/home/messung/hv/hvreadout
Konfigurationsdatei	/home/messung/hv/HVconfig.txt
Logdateien	/home/vdc/data/hv/data/
	/home/vdc/data/hv/log/
Statusdatei	/home/messung/currentstateHV.txt

Tabelle 3: Übersicht HVREADOUT

Das Programm HVREADOUT steuert und überwacht die Spannungsversorgung der Kathoden, Anoden und SiPMs. Es spricht die steuerbare Hochspannungsversorgung[27] über Ethernet an, setzt Sollspannungen entsprechend der Einstellung in der Konfigurationsdatei und loggt die Istwerte. Für nähere Angaben siehe z.B. [5].

[25] Die Prozessliste wird mit dem Systemaufruf „ps -A“ erzeugt.
[26] z.B. über gefundene oder nicht gefundene Prozesse
[27] Caen SY1527

4.3 GASMATRIX

Name	GASMATRIX	
Kommandozeilenargumente	-D	Als Daemon starten
	-d1	Debuglevel 1
	-T s	Messintervall auf s (4 bis 10) Sekunden stellen.
	-v	Verbose Ausgabe jedes Messwertes zur Analyse der Heizvorgänge.
Ausführbare Datei	/home/messung/gasmatrix/gasmatrix	
Konfigurationsdatei	/home/messung/gasmatrix/gasmatrixconfig.txt	
Logdateien	/home/vdc/data/gasmatrix/log/ConfigLog.txt	
	/home/vdc/data/gasmatrix/log/ValveLog.txt	
	/home/vdc/data/temperatures/log/TempWarning.txt	
	/home/vdc/data/temperatures/log/TempLog.txt	
Statusdateien	/home/messung/currentstateValves.txt	
	/home/messung/currentstateTemperatures.txt	

Tabelle 4: Übersicht GASMATRIX

GASMATRIX ist für die Bestimmung der Ventilpositionen und das Auslesen sämtlicher Temperatursensoren zuständig. Der Zugriff auf die Temperatursensoren und die digitalen Schalter geschieht über den 1-Wire-Bus. Die Routinen für den Zugriff auf diesen wurden von [5] übernommen.

Ein Auslesezyklus sieht dabei immer gleich aus:

Es wird eine Spannung an die Heizwiderstände an den Ventilhebeln (siehe Abschnitt 2.1.2) angelegt und sie heizen sich auf. Die Heizdauer ist in der Hardware fest eingestellt und kann nicht von der Software beeinflusst werden. Zur Zeit beträgt die Heizdauer ca. 30 Sekunden. Der Start jedes Heizvorgangs wird in der Logdatei festgehalten.

Nun werden in regelmäßigen Intervallen (einstellbar über das Kommandozeilenargument -T; standardmäßig 10 Sekunden[28]) die Temperaturen aller Sensoren in der Matrix und aller im System verteilten Einzelsensoren ausgelesen. Die Werte der Einzelsensoren werden in die Statusdatei currentstateTemperatures.txt geschrieben. Wurde das Programm im verbosen Modus aufgerufen (Argument -v), so werden außerdem alle Temperaturen in der Datei TempLog.txt protokolliert. Die erreichten Maximaltemperaturen der Sensoren in der Matrix werden bestimmt.

Nach ca. 4 Minuten ist ein Zyklus beendet. Für jedes Ventil wird die Differenz der Maximaltemperaturen der Sensoren Opentemp und Closetemp (siehe Abschnitt 2.1.2) gebildet und aus ihr die Position des Ventils bestimmt:

$$\Delta T = T_{max,o} - T_{max,c}$$

$$\begin{aligned} \Delta T > \Theta &\implies \text{Ventil offen} \\ \Delta T < -\Theta &\implies \text{Ventil geschlossen} \\ |\Delta T| \leq \Theta &\implies \text{Ventilposition unbekannt} \end{aligned}$$

Θ ist hierbei eine festgelegte[29] Temperaturschwelle, unterhalb derer die Temperaturdifferenz als zu gering betrachtet wird, um eine eindeutige Aussage über die Ventilposition zu treffen. Dies tritt z.B. dann auf, wenn sich das Ventil nicht in einer eindeutigen Position befindet oder wenn die Beheizung der Ventilhebel nicht korrekt funktioniert. Zur Zeit ist diese Schwelle auf 1 Kelvin gesetzt. Ob dies ein sinnvoller Wert ist, muss – zusammen mit der Heizdauer – noch einmal überprüft werden, wenn der Messschrank geschlossen und die Wasserkühlung eingeschaltet ist (siehe auch Abschnitt 5.2).

[28] Ein Messintervall von mehr als 10 Sekunden ist nicht vorgesehen, da das Maximum der Temperatur an dem beheizten Sensor nach 30 Sekunden dann verpasst werden könnte.

[29] Die entsprechende Variable im Quellcode heißt „mindiff". Um sie zu ändern, muss der Code neu kompiliert werden.

Die so bestimmten Ventilpositionen werden nun in die Statusdatei currentstateValves.txt geschrieben und eventuelle Abweichungen vom konfigurierten Sollwert in der Datei ValveLog.txt vermerkt. Befindet sich das Programm nicht im verbosen Modus, werden außerdem die über den Heizzyklus gemittelten Werte der Einzelsensoren in TempLog.txt gespeichert.

Misst ein Sensor eine Temperatur außerhalb des für ihn festgelegten Temperaturbereiches, so wird dies in der Logdatei TempWarnings.txt vermerkt. Fehler, die beim Auswerten der Konfigurationsdatei auftreten, werden in der Datei ConfigLog.txt gespeichert.

4.4 FLOWBUS

Name	FLOWBUS
Kommandozeilenargumente	-D Als Daemon starten
	-d1 Debuglevel 1
Ausführbare Datei	/home/messung/flowbus/flowbus
Konfigurationsdatei	/home/messung/flowbus/fbConfig.txt
Logdatei	/home/vdc/data/gas/log/flowbusLog.txt
Statusdatei	/home/messung/currentstateFlowbus.txt

Tabelle 5: Übersicht FLOWBUS

Das Programm FLOWBUS dient dazu über die serielle Schnittstelle[30] des PCs mit dem Gasregelsystem (siehe Abschnitt 2.1.1) zu kommunizieren[31]. In der Konfigurationsdatei werden für jede der Kammern ein Solldruck (absolut) und ein Sollfluss angegeben.

FLOWBUS liest diese Werte in Intervallen von ca. 10 Sekunden aus und setzt die Werte im Gasregelsystem entsprechend. Anschließend werden werden die tatsächlich gesetzten Sollwerte auch wieder ausgelesen, um eventuelle Fehler in der Kommunikation zu erkennen. Abweichungen vom gewünschten Sollwert werden sowohl in der Logdatei als auch in der Statusdatei vermerkt.

Außerdem werden im gleichen Intervall sämtliche Istwerte ausgelesen und in Status- und Logdatei gespeichert. Auch hier werden Abweichungen vom Sollwert vermerkt. Bei diesen Werten handelt es sich allerdings um reale Messwerte, die immer etwas um den tatsächlichen Wert schwanken. Selbst der tatsächliche Wert ist als Teil eines Regelkreises Schwankungen unterworfen, die aber keinesfalls eine Fehlfunktion der Regelung bedeuten müssen.

Um dem Rechnung zu tragen, wird nur eine Warnung ausgegeben, wenn die Abweichung vom Sollwert größer ist als ein bestimmter Wert. Ad hoc wurde dieser auf 1% des Sollwertes gesetzt. Ob diese erlaubte Abweichung zu eng oder zu weit gefasst ist, muss durch Messung der Stabilität des Druckes und des Flusses bestimmt werden (siehe Abschnitt 5.1).

4.5 PBREADOUT

Name	PBREADOUT
Kommandozeilenargumente	-D Als Daemon starten
	-d1 Debuglevel 1
Ausführbare Datei	/home/messung/pbreadout/pbreadout
Konfigurationsdatei	/home/messung/pbreadout/pbrConfig.txt
Logdatei	/home/vdc/data/gas/log/pbreadoutLog.txt
Statusdatei	/home/messung/currentstatePressureBox.txt

Tabelle 6: Übersicht PBREADOUT

[30] /dev/ttyS0

[31] Als Protokoll dient hier das namensgebende FLOW-BUS Protokoll. Es wird die ASCII-Variante verwendet. Für Details siehe [10].

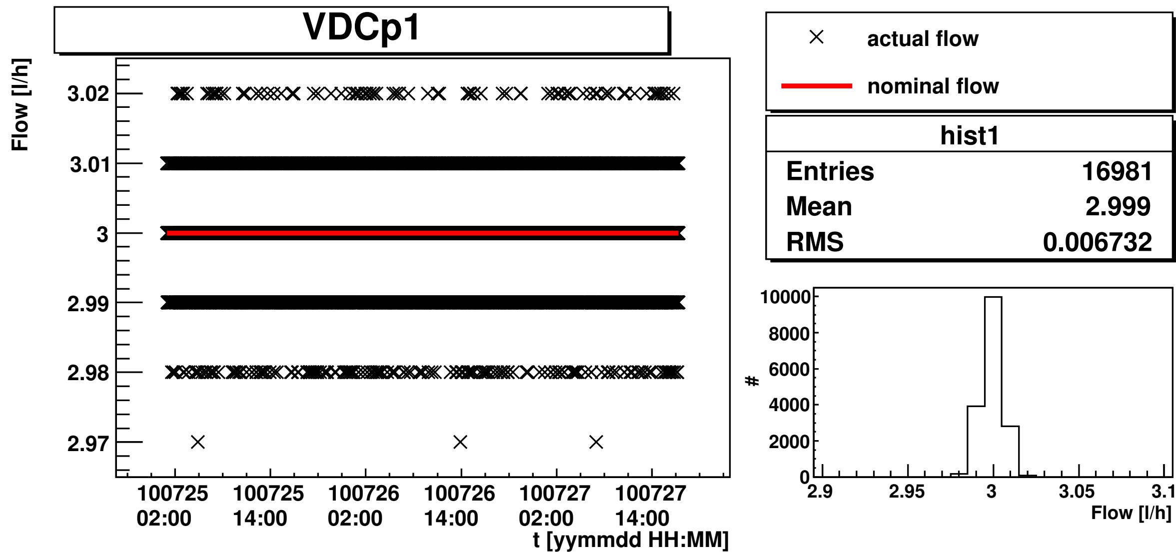


Abbildung 8: Messung der Flussstabilität – Die Ausgeprägte Abstufung der Werte rührt daher, dass FLOWBUS die Flusswerte nur mit einer Genauigkeit von 2 Nachkommastellen ausgibt.

PBREADOUT dient dem Auslesen der zusätzlichen Differenz- und Absolutdrucksensoren der Kammern (siehe Abschnitt 2.1.1). Diese liefern ein analoges Spannungssignal und sind zur Auslese an den Analog-Digital-Wandler NI-PCI-6229 von National Instruments angeschlossen.

Da die Treiber, welche von National Instruments angeboten werden, nicht mit 64-Bit Linuxsystemen, wie dem hier verwendeten, kompatibel sind, kommt stattdessen der open-source Treiber COMEDI zum Einsatz, welcher die Hardware ebenfalls direkt unterstützt.[11]

Die Sensoren werden in Intervallen von ca. 10 Sekunden ausgelesen. Eine Messung umfasst hierbei das Aufnehmen und Mitteln von 1000 Werten. Dies ist möglich, da der A/D-Wandler bis zu 250000 Werte pro Sekunde aufnehmen kann.

In der Konfigurationsdatei werden für jeden Sensor der entsprechende Kanal am A/D-Wandler, der normale Wertebereich und der Sensortyp angegeben. Anhand des Sensortyps geschieht die Zuordnung von Druck und Spannung, wie sie im Datenblatt des Sensors angegeben ist.

Eine Einbindung von sensorspezifischen Lookup-Tabellen, mit deren Hilfe man die Genauigkeit der Druckmessung stark erhöhen kann[8], ist vorgesehen, aber noch nicht implementiert.

5 Analyse aufgenommener Daten

Alle im Folgenden gezeigten Auswertungen und Plots wurden mithilfe von ROOT-Skripten[12] aus den Klartext-Logdateien der hier vorgestellten Programme erstellt.

5.1 Druck- und Flussstabilität

Um eine vernünftige Grenze festzulegen, ab derer eine Abweichung vom Solldruck und Sollfluss als Fehlfunktion eingestuft wird, muss zunächst einmal bestimmt werden, welchen Schwankungen diese beiden Werte unterliegen.

Hierfür kann man die Logdatei des Programms FLOWBUS auswerten[32]. Abbildung 8 zeigt exemplarisch die aufgezeichneten Ist- und Sollflusswerte für Kammer 1. Abbildung 9 zeigt die entsprechenden Druckwerte.

Die Messungen der anderen Kammern ergeben sehr ähnliche Werte[33]. Wie man aus Tabelle 7 sieht, entsprechen die 1% maximale Abweichung, welche ad hoc für die Feststellung einer Fehlfunktion ausgewählt wurden, bei

[32] Die Werte, die PBREADOUT liefert, sind wegen ihrer fehlenden Kalibrierung noch nicht geeignet quantitativ die Druckstabilität zu bestimmen.

[33] Die zugehörigen Plots befinden sich im Anhang A.1 und A.2

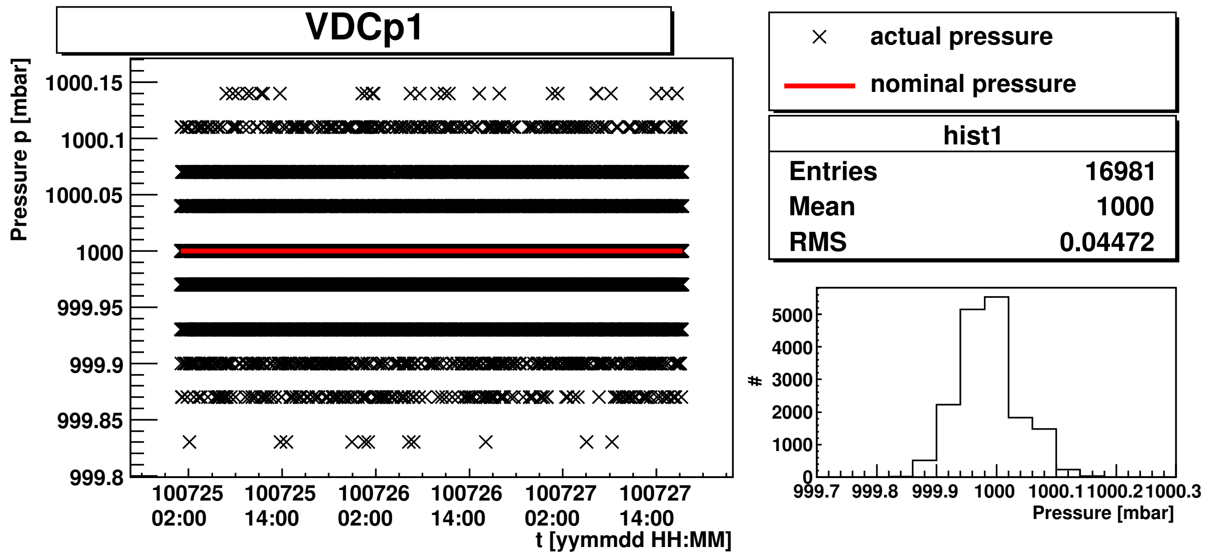


Abbildung 9: Messung der Druckstabilität – Die Abstufung der Werte ist hier hauptsächlich durch die Auflösung des Druckregelsystems gegeben. Drücke werden als digitale Ganzzahlenwerte übertragen, wobei ein Druck von 1100 mbar einem Wert von 32000 entspricht. Die minimale Schrittweite beträgt also 1100 mbar/32000 $\approx 0,034$ mbar. FLOWBUS rechnet die Ganzzahlenwerte mit diesem Faktor in mbar um und gibt sie mit 2 Nachkommastellen Genauigkeit aus, weswegen die Digitalisierungsschritte hier nicht alle denselben Abstand haben.

Kammer	(Fluss ± RMS) / l/h	# Flussmesspunkte	(Druck ± RMS) / mbar	# Druckmesspunkte
1	3,00 ± 0,01	16981	1000,00 ± 0,04	16981
2	3,00 ± 0,01	16981	1000,00± 0,07	16981
3	3,00 ± 0,01	16981	1000,00± 0,05	16981
4	3,00 ± 0,01	16981	1000,00± 0,06	16981
5	3,00 ± 0,01	16981	1000,00± 0,06	16981
6	3,00 ± 0,01	16981	1000,00± 0,05	16981

Tabelle 7: Gemessene Gasfluss- und Druckstabilitäten – Daten wurden über einen Zeitraum von ca. 64 Stunden aus den Bronkhorst-Reglern gelesen.

dem Gasfluss in etwa dem Dreifachen des RMS. Das heißt, es würden noch ca. 0,3% aller gemessenen Werte fälschlicherweise als anomale Abweichung vom Sollwert interpretiert werden. Bei tausenden von Messungen pro Tag[34] gäbe es also täglich einige Fehlwarnungen. Um diese Fehlwarnungen zu beseitigen oder zumindest so unwahrscheinlich zu machen, dass sie den Betriebsablauf nicht mehr stören, bietet es sich an die Toleranz der Abweichung auf das Fünffache des RMS zu erhöhen. Dies entspricht einer Flussabweichung von ca. 0,05 l/h.

Beim Druck sieht die Situation etwas anders aus. Hier entspricht eine Abweichung von 1% – also 10 mbar – schon mehr als dem 100-fachen des RMS. Die Toleranz der Druckabweichungen kann also deutlich reduziert werden. Legt man auch hier die Grenze auf das Fünffache des RMS, so entspricht dies einer Druckabweichung von 0,25 bis 0,35 mbar.

Empfohlene Toleranzen	Fluss (bestimmt bei 3 l/h)	Druck (bestimmt bei 1000 mbar)
absolut	0,05 l/h	0,35 mbar
relativ	1,7 %	0,035 %

Tabelle 8: Empfohlene Toleranzen für Gasfluss und -druck beim Gasregelsystem – Bestimmt unter Laborbedingungen in Aachen

Tabelle 8 zeigt die hier bestimmten empfohlenen Toleranzen für die Gasfluss- und Druckabweichungen. Bei diesen Betrachtungen muss jedoch berücksichtigt werden, dass die Rahmenbedingungen im Labor – wie z.B. der Druck

[34] Es wird ca. alle 10 Sekunden gemessen.

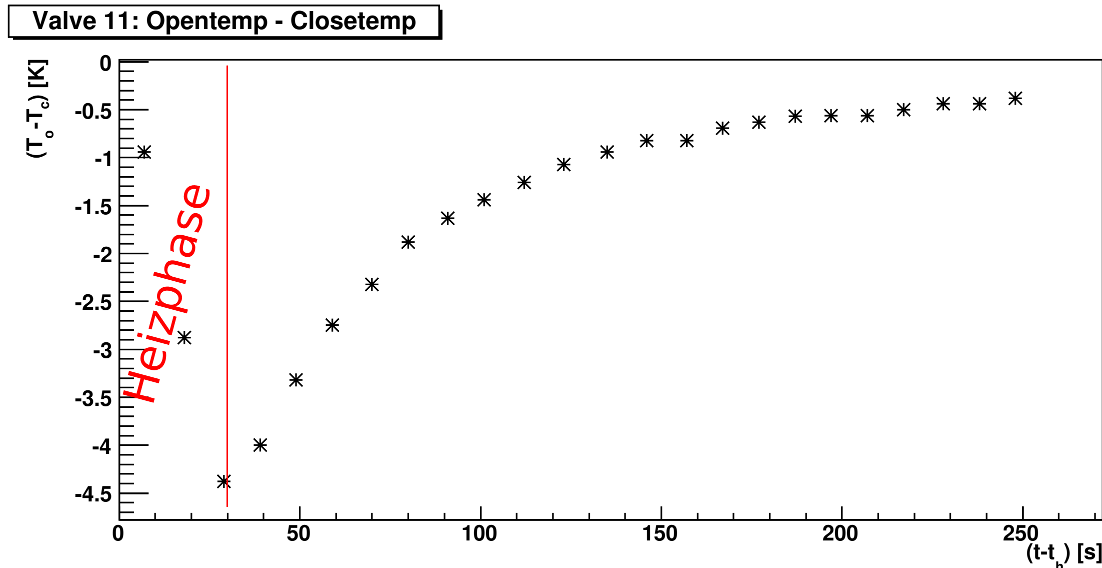


Abbildung 10: Temperaturdifferenzverlauf während eines Heizzyklus – Aufgetragen ist die Differenz der Temperaturen der Sensoren Opentemp (T_o) und Closetemp (T_c) gegen die Zeit nach t nach Start des Heizvorganges t_h. Da das Ventil geschlossen ist, wird nur der Sensor Closetemp beheizt und die Differenz wird negativ. Nach ca. 30 Sekunden wird nicht mehr geheizt und die Temperaturen gleichen sich wieder an.

des Gases vor dem Regelsystem – sehr konstant sind. Bei CMS können diese Bedingungen größeren Schwankungen unterlegen sein, was sich dann auch auf die Schwankungen von Druck und Fluss auswirken wird. Wenn das VDC-System bei CMS integriert ist, sollte also noch einmal eine Stabilitätsmessung zur Bestimmung von Toleranzgrenzen durchgeführt werden. Außerdem sollte für die von PBREADOUT ausgewerteten Drucksensoren eine eigene Toleranzgrenze bestimmt werden, sobald die Lookup-Tabellen in das Programm integriert sind, da diese Sensoren wahrscheinlich eine andere intrinsische Schwankung aufweisen werden als die Sensoren in der Gasregelung.

5.2 Die Ventilauslese

Um die Heizvorgänge an der Ventilmatrix (siehe Abschnitt 2.1.2) zu analysieren, kann die Temperaturlogdatei des Programms GASMATRIX ausgewertet werden, wenn sie im verbosen Modus aufgezeichnet wurde (Option -v, siehe Abschnitt 4.3), da hier der gesamte Temperaturverlauf aufgezeichnet wird. Dies soll hier anhand von Daten, die während des regulären Betriebs des übrigen VDC-Systems aufgezeichnet wurden, demonstriert werden.

Abbildung 10 zeigt einen aufgenommenen Heizvorgang, beispielhaft für Ventil 11. Die entsprechenden Plots der übrigen Ventile befinden sich im Anhang A.3. Da wir nicht an den absoluten Temperaturen, sondern nur an den unterschiedlichen Temperaturverläufen der Sensoren Opentemp und Clostemp interessiert sind, betrachten wir erst einmal nur die Differenz dieser. Aufgetragen ist hier also die Temperaturdifferenz $T_o - T_c$ gegen die Zeit nach dem Starten des Heizvorganges.

Man sieht gut, wie der Betrag der Temperaturdifferenz mit Beginn des Zyklus ansteigt und dann nach ca. 30 Sekunden, wenn die von der Hardware vorgegebene Heizzeit abgelaufen ist, wieder abfällt.

Da der Beginn jedes Heizzyklus in der Logdatei vermerkt ist, ist es möglich sämtliche Heizvorgänge übereinander zu legen und so eine Aussage über die Reproduzierbarkeit der Heizkurve zu machen. Weil wir hier nur die Differenz der Temperaturen Opentemp und Closetemp betrachten, fallen tageszeit- oder wetterabhängige Schwankungen, die ja beide Sensoren gleichermaßen beeinflussen, nicht ins Gewicht. Abbildung 11 zeigt den überlagerten Verlauf der Temperaturdifferenz in Abhängigkeit von der Zeit, die nach dem Starten des jeweiligen Heizzyklus verstrichen ist.

Die Messpunkte, welche über einen ganzen Tag aufgenommen wurden, liegen alle in etwa auf derselben Kurve. Dies spricht dafür, dass das Beheizen der Ventilhebel zuverlässig und reproduzierbar funktioniert.

Für die Bestimmung der Ventilpositionen wird aber nicht direkt der Verlauf der Temperaturdifferenz, sondern die Differenz der während eines Heizzyklusses gemessenen Maximaltemperaturen verwendet. Um die Stabilität

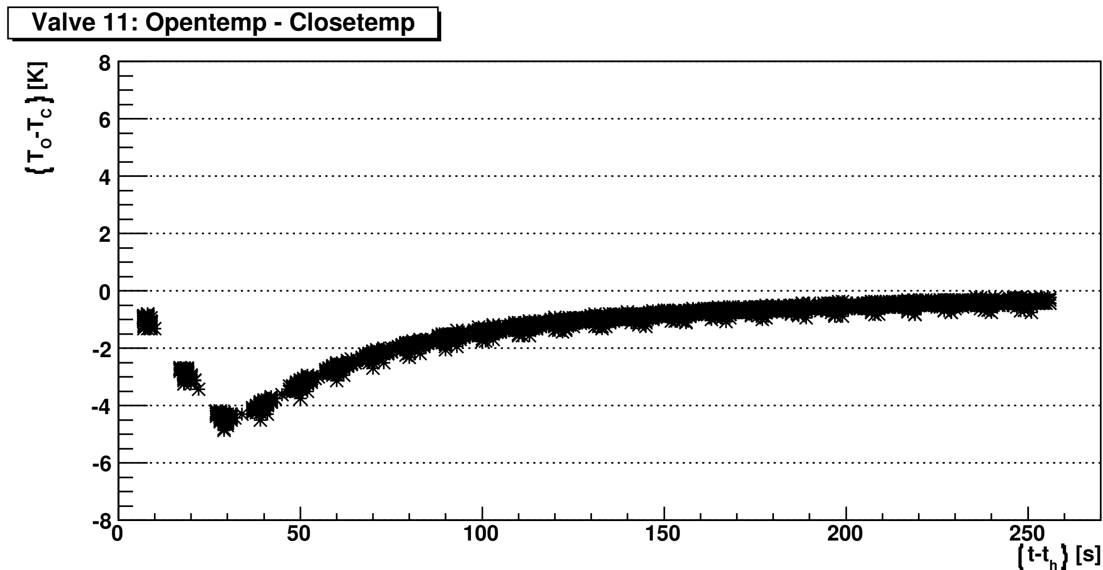


Abbildung 11: Temperaturdifferenzverläufe von 289 Heizzyklen überlagert – T_o = Temperatur des Sensors Opentemp; T_c = Temperatur des Sensors Closetemp; t_h = Start des Heizvorgangs

Ventil:	1y	2y	3y	4y	5y	6y
y=1	-4,362 ± 0,082	-4,864 ± 0,394	-2,748 ± 0,077	-4,48 ± 0,181	-3,489 ± 0,102	-4,376 ± 0,137
y=2	-2,988 ± 0,063	-3,383 ± 0,166	-2,909 ± 0,094	-2,921 ± 0,121	-3,711 ± 0,086	-3,024 ± 0,110
y=3	-2,439 ± 0,061	-3,945 ± 0,150	-2,864 ± 0,103	-4,76 ± 0,169	-3,219 ± 0,094	-2,022 ± 0,089
y=4	-3,423 ± 0,062	-4,826 ± 0,132	-3,446 ± 0,098	-2,823 ± 0,120	-3,584 ± 0,084	-0,355 ± 0,471
y=5	-2,811 ± 0,138	-4,024 ± 0,112	-2,47 ± 0,084	-3,094 ± 0,126	-3,675 ± 0,079	-2,509 ± 0,092
y=6	3,995 ± 0,089	4,686 ± 0,114	2,836 ± 0,102	3,538 ± 0,837	5,676 ± 0,080	2,845 ± 0,109
y=7	2,923 ± 0,070	2,882 ± 0,078	2,892 ± 0,086	3,016 ± 0,113	4,146 ± 0,081	3,858 ± 0,199
	Ventil 101:		4,113 ± 0,134	Ventil 102:		-3,243 ± 0,085

Tabelle 9: Mittlere Differenz der Maximaltemperaturen der verschiedenen Ventile: $((T_{\mathrm{max,o}} - T_{\mathrm{max,c}}) \pm RMS)$ [K] – Es handelt sich um Mittelwerte aus den 289 Heizzyklen. Die entsprechenden überlagerten Temperaturverläufe und Histogramme befinden sich im Anhang A.3. Auffällig sind die Ventile 21, 46 und 64, welche alle ein ungewöhnlich großes RMS, also eine große Streuung der Werte aufweisen. Bei Ventil 64 ist außerdem die im Mittel erreichte Temperaturdifferenz zu niedrig.

diese Messmethode zu untersuchen, können wir einerseits schauen, wie sich diese Differenz im Laufe der Zeit entwickelt (Abbildung 12), oder direkt die Verteilung der Differenzen betrachten (Abbildung 13).

Wie man sieht beträgt der Betrag der mittleren Differenz für dieses Ventil ca. 4,362 K, bei einem RMS von 0,081 K. Die gesetzte Grenze von 1 K (Siehe Abschnitt 4.3) ist also mehr als 30 RMS von mittleren Wert entfernt und damit geeignet, um falsche Fehlermeldungen auszuschließen.

Tabelle 9 zeigt die mittlere Differenz der Maximaltemperaturen der Heizzyklen aller Ventile. Positive Werte bedeuten ein offenes und negative Werte ein geschlossenes Ventil. Die Werte der einzelnen Ventile unterscheiden sich stark von einander, da der quantitative Verlauf der Heizkurven stark von der Geometrie des Ventilaufbaus abhängt. Die Abstände zwischen den Ventilhebeln und den Temperatursensoren sind nicht überall gleich und verursachen so die unterschiedlich hohe Erwärmung der Sensoren.

Bei den meisten Ventilen ist die mittlere Differenz größer als 2 K und das RMS kleiner als 0,2 K. Die mittleren Differenzen sind also mehr als 5 RMS vom vorgegebenen Schwellwert von 1 K entfernt.

Die Ventile 21, 46 und 64 fallen hierbei jedoch aus der Reihe. Ihr RMS ist deutlich höher als das der anderen Ventile und bei Ventil 64 liegt der Mittelwert mit -0,355 K sogar innerhalb des Bereichs, bei dem keine eindeutige Ventilposition bestimmt werden kann. Schaut man sich die entsprechenden Plots an, sieht man, dass das Heizen bei diesen Ventilen nicht so zuverlässig funktioniert, wie bei den anderen Ventilen (Abb. 14). Während die erreichte Temperaturdifferenz bei Ventil 21 trotz der auffälligen Heizkurven noch ausreicht, um die Ventilposition sicher zu bestimmen, scheint die Heizung bei Ventil 46 hin und wieder und bei Ventil 64 die meiste Zeit total zu

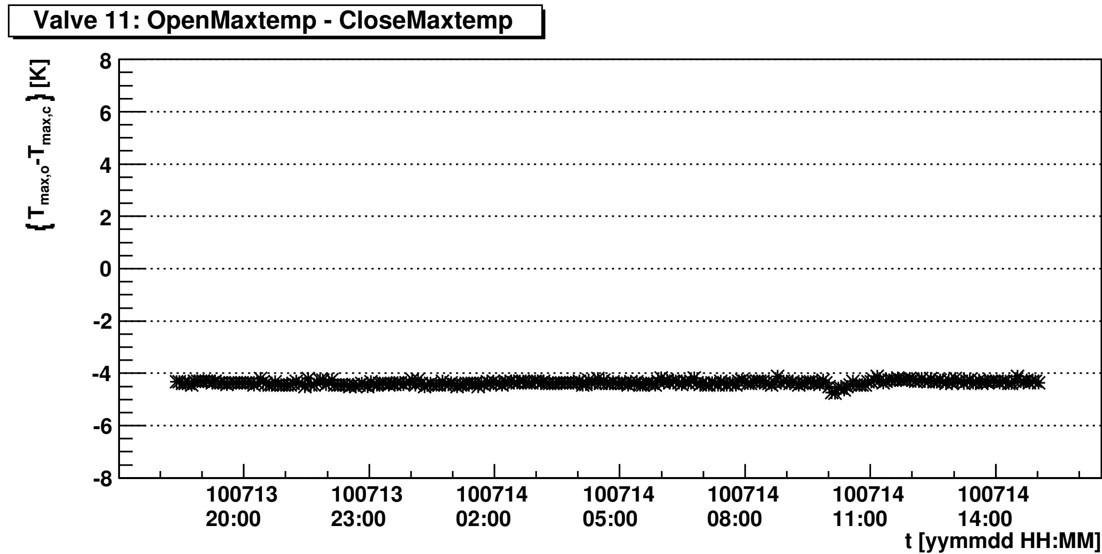


Abbildung 12: Differenzen der Maximaltemperaturen der 289 Heizzyklen über die Zeit – $T_{\mathrm{max,o}}$ = Maximale, während eines Heizzyklusses gemessene Temperatur des Sensors Opentemp; $T_{\mathrm{max,c}}$ = Maximale, während eines Heizzyklusses gemessene Temperatur des Sensors Closetemp

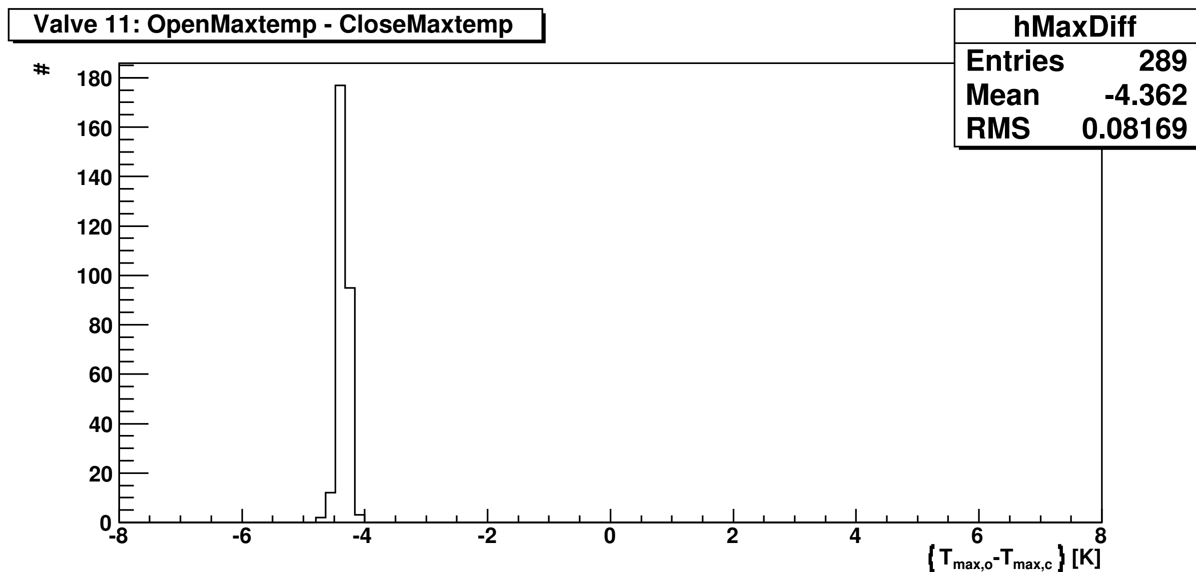


Abbildung 13: Histogramm der Differenzen der Maximaltemperaturen der 289 Heizzyklen – $T_{\mathrm{max,o}}$ = Maximale, während eines Heizzyklusses gemessene Temperatur des Sensors Opentemp; $T_{\mathrm{max,c}}$ = Maximale, während eines Heizzyklusses gemessene Temperatur des Sensors Closetemp

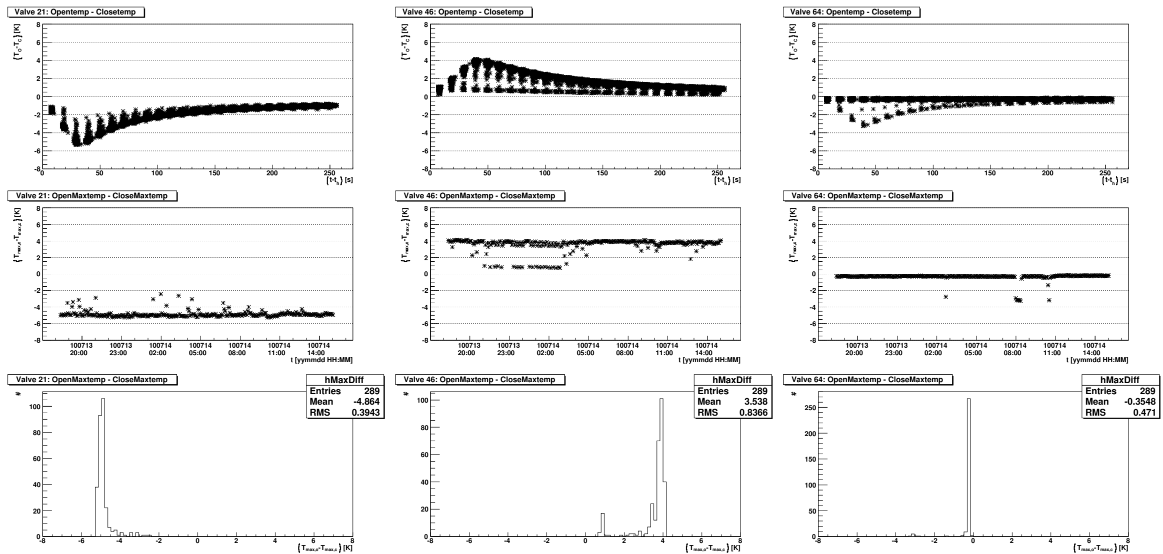


Abbildung 14: Auffälliges Heizverhalten der Ventile (von links nach rechts) 21, 46 und 64 – Gezeigt sind (von oben nach unten) die überlagerten Temperaturverläufe von 289 Heizzyklen ($(T_o - T_c)$ gegen $(t - t_h)$), die Differenzen der Maximaltemperaturen gegen die Zeit ($(T_{\text{max,o}} - T_{\text{max,c}})$ gegen t) und die Häufigkeitsverteilung der Differenzen der Maximaltemperaturen ($T_{\text{max,o}} - T_{\text{max,c}}$). Größere Darstellungen der Plots befinden sich im Anhang A.3. Die Heizung der drei Ventile scheint nicht zuverlässig zu funktionieren. Dies könnte an defekten Kontakten liegen.

Ventil:	1y	2y	3y	4y	5y	6y
y=1	4,762 ± 0,122	5,314 ± 0,092	3,234 ± 0,148	3,382 ± 0,162	4,179 ± 0,166	3,529 ± 0,136
y=2	3,069 ± 0,101	3,932 ± 0,120	3,174 ± 0,131	3,299 ± 0,156	4,398 ± 0,172	2,661 ± 0,130
y=3	3,132 ± 0,117	4,936 ± 0,132	3,279 ± 0,147	4,158 ± 0,187	3,432 ± 0,144	2,466 ± 0,123
y=4	4,104± 0,093	5,631 ± 0,121	4,519 ± 0,140	2,903 ± 0,101	3,714 ± 0,167	-0,2522 ± 0,037
y=5	3,331± 0,103	5,340 ± 0,124	3,382 ± 0,111	2,743 ± 0,118	5,241 ± 0,150	2,563 ± 0,095
y=6	-3,069± 0,087	-3,376 ± 0,088	-2,510 ± 0,340	-2,074 ± 0,145	-4,171 ± 0,104	-2,388 ± 0,079
y=7	-2,569± 0,072	-3,001 ± 0,087	-2,096 ± 0,096	-2,514 ± 0,084	-3,986 ± 0,173	-3,394 ± 0,093
	Ventil 101:		-2,111 ± 0,129	Ventil 102:		3,000 ± 0,168

Tabelle 10: Mittlere Differenz der Maximaltemperaturen der verschiedenen Ventile in anderer Position: $((T_{\text{max,o}} - T_{\text{max,c}}) \pm RMS)$ [K] – Es handelt sich um Mittelwerte aus 9 Heizzyklen mit im Vergleich zu Tabelle 9 umgekehrten Ventilstellungen. Hier fällt Ventil 36 mit einem erhöhten RMS auf.

versagen. Dies könnte unter anderem an fehlerhaften Kontakten liegen und muss untersucht werden.

Tabelle 10 zeigt die gemessenen mittleren Differenzen in gegenüber Tabelle 9 genau umgekehrter Ventilstellung und Tabelle 11 zeigt die entsprechenden Werte mit allen Ventilen in einer mittleren Position zwischen der offenen und geschlossenen Position. Wie man sieht, sind die aktuell gewählte Heizzeit von ca. 30 s und die Schwelle von 1 K unter den zur Zeit herrschenden Laborbedingungen geeignet, um die Ventilposition sicher zu bestimmen. Allerdings sollten die Parameter auch unter den später im Betrieb herrschenden Bedingungen (Seitenwände und Türen der Schränke, zusätzliche Lüftung, etc.) noch einmal mit mehr Heizzyklen überprüft werden, um eine definitive quantitative Aussage über die Stabilität der Ventilpositionsbestimmung treffen zu können.

Der Vollständigkeit halber sei hier auch der Verlauf der Raum- bzw. Schranktemperatur über die Messdauer angegeben (Abbildung 15). Diese wird als Mittelwert aus den unbeheizten Sensoren in der Ventilmatrix bestimmt. Die Veränderungen der Schranktemperatur sind langsam, aber von der selben Größenordnung wie die aufgezeichneten Heizzyklen. Dies verdeutlicht, wieso bei der Analyse der Ventilauslese lokal gebildete Temperaturdifferenzen zur Beseitigung der Schranktemperatur herangezogen werden.

Ventil:	1y	2y	3y	4y	5y	6y
y=1	0,328 ± 0,028	-0,316 ± 0,031	0,237 ± 0,036	0,166 ± 0,062	0,080 ± 0,080	-0,135 ± 0,071
y=2	0,136 ± 0,026	0,084 ± 0,028	-0,006 ± 0,033	0,072 ± 0,023	0,118 ± 0,073	0,017 ± 0,060
y=3	0,039 ± 0,039	-0,035 ± 0,032	0,067 ± 0,031	-0,169 ± 0,060	-0,001 ± 0,069	-0,023 ± 0,073
y=4	0,0346 ± 0,032	0,113 ± 0,034	0,081 ± 0,028	0,062 ± 0,065	0,097 ± 0,062	-0,237 ± 0,023
y=5	0,153 ± 0,033	0,085 ± 0,030	0,116 ± 0,024	0,090 ± 0,042	0,051 ± 0,074	-0,001 ± 0,052
y=6	-0,022 ± 0,029	-0,005 ± 0,031	-0,188 ± 0,004	0,023 ± 0,030	0,080 ± 0,029	0,121 ± 0,034
y=7	-0,117 ± 0,030	0,062 ± 0,026	-0,016 ± 0,027	-0,029 ± 0,042	0,093 ± 0,041	0,050 ± 0,038
	Ventil 101:		0,068 ± 0,017	Ventil 102:		0,034 ± 0,089

Tabelle 11: Mittlere Differenz der Maximaltemperaturen der verschiedenen Ventile in mittlerer Position: $((T_{\text{max,o}} - T_{\text{max,c}}) \pm RMS)$ [K] – Es handelt sich um Mittelwerte aus 11 Heizzyklen. Alle Ventile wurden in eine mittlere Position zwischen der offenen und geschlossenen Stellung gedreht. An diesen Messwerten erkennt man bei jedem Ventil deutlich, dass es in keiner der zwei regulären Stellungen (Tab. 9 und 10) ist. Daher wird diese Ventilstellung „undefiniert“ genannt.

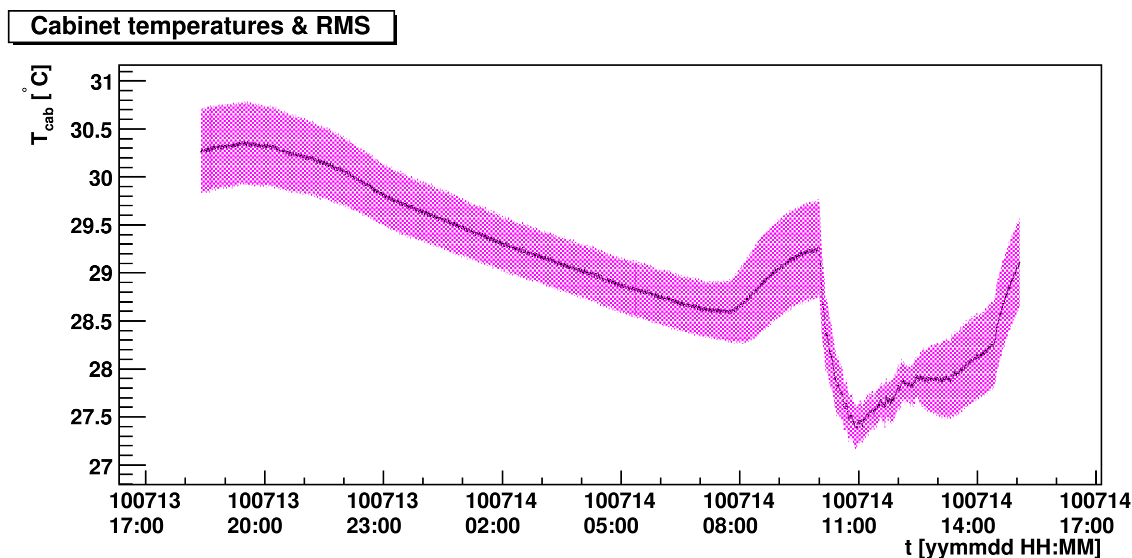


Abbildung 15: Messschranktemperatur über die Zeit – Mittelwert aus den unbeheizten Sensoren in der Ventilmatrix und RMS

6 Integration des Systems bei CMS

Außer den Programmen zur reinen Aufnahme, Überwachung und Aufzeichnung der Betriebsparameter müssen auch die Einbindung in das Detector Control System (DCS) bei CMS und die grafische Schnittstelle zum Benutzer realisiert werden.

Die Einbindung in das DCS existiert bereits für eine einzelne VDC. Sie muss jedoch für den Einsatz mit letztendlich 6 VDCs und mehr überwachten Parametern erweitert werden. Hieran arbeitet zur Zeit unter anderem [13].

Die Arbeit an der grafischen Benutzeroberfläche wurde ebenfalls bereits begonnen, befindet sich aber noch in einer sehr frühen Entwicklungsphase. Abbildung 16 zeigt den aktuellen Stand. Hier ist noch Entwicklungsarbeit notwendig.

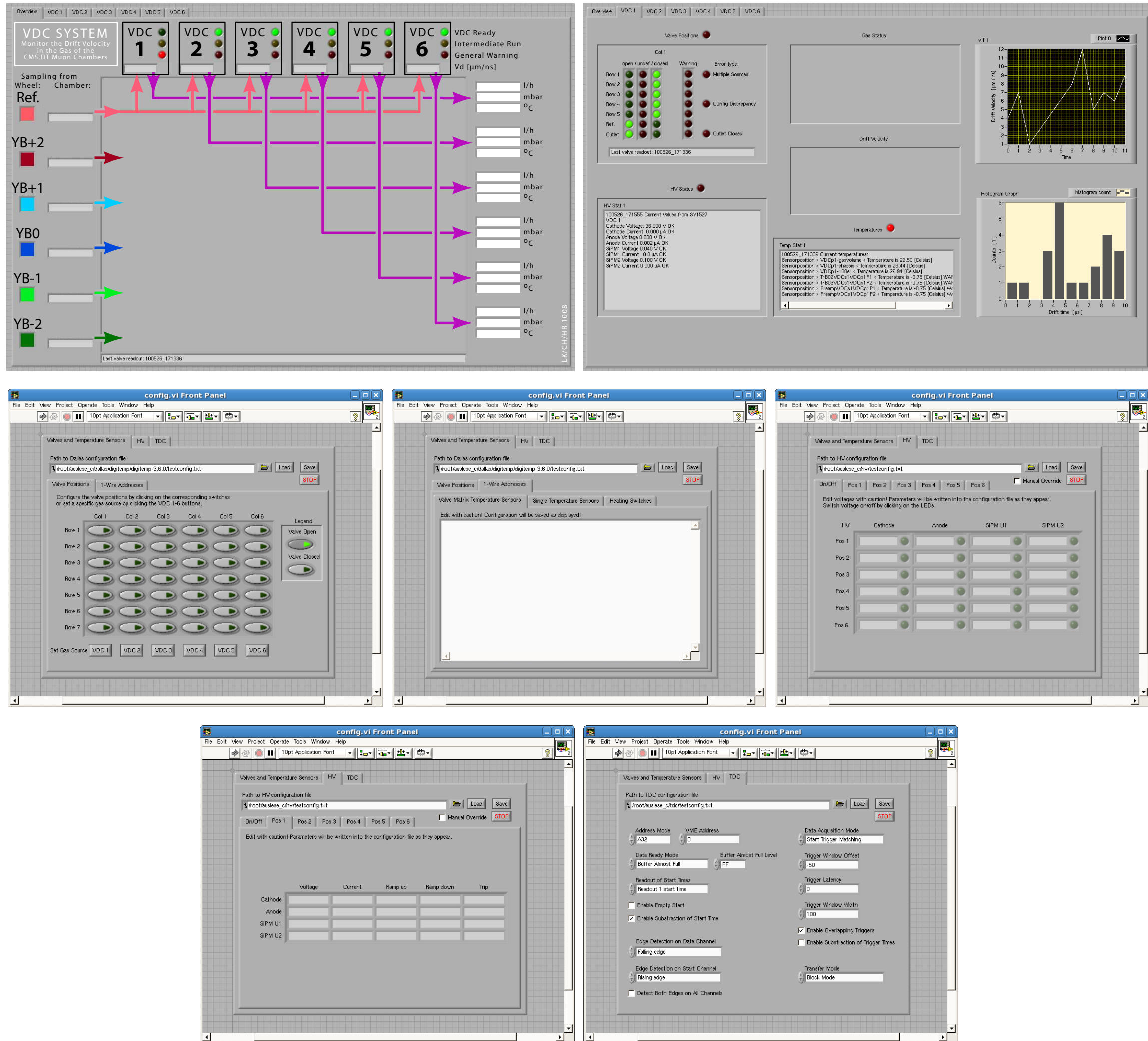


Abbildung 16: Momentaner (sehr früher) Stand der Entwicklung der grafischen Benutzeroberfläche – Es handelt sich um in LabVIEW[14] geschriebene Programme. Von links nach recht, von oben nach unten: Der Übersichtsbildschirm auf dem der Status der VDCs und aktuell gemessene Driftgeschwindigkeiten auf einem Blick erfassbar sein sollen[2]; der Detailbildschirm mit allen über eine Kammer gesammelten Informationen – wie Druck und Temperatur – und Warnungen; der Konfigurationseditor für die Soll-Ventilstellungen; der Konfigurationseditor für die Temperatursensoren; der Konfigurationseditor für die Spannungen der HV-Spannungsversorgung; Detaileinstellungen für die Spannungsversorgung einer Kammer; Konfigurationseditor für den TDC.

7 Fazit und Ausblick

Im Rahmen dieser Arbeit wurden die Programme MASTERD und MINID für die Prozesskontrolle des Systems entwickelt. Das Programm GASMATRIX zur Auslese der Ventilpositionen und Temperatursensoren wurde erweitert. Das Programm FLOWBUS wurde zur Gasfluss- und Drucksteuerung und -überwachung und PBREADOUT nur zur Drucküberwachung entwickelt.

Die hier vorgestellten Programme funktionieren und liefern Daten, welche man zur Überwachung des Systems verwenden kann. Allerdings sind die Sensoren, welche von PBREADOUT ausgewertet werden zwar schon kalibriert, aber ihre Lookup-Tabellen müssen noch erzeugt und in das Programm eingebunden werden. Dann sollten auch Messungen zur Toleranzgrenzenbestimmung dieser Sensoren stattfinden.

Aus den aufgenommenen Daten der Gasfluss- und Druckregelung konnten Toleranzgrenzen hergeleitet werden, welche ein zuverlässiges Erkennen von anomalen Zuständen erlauben sollten, ohne im regulären Betrieb falsche Warnungen zu erzeugen. Diese Messung sollte wiederholt werden, wenn das VDC-System bei CMS an das Gassystem angeschlossen wurde, um den veränderten Bedingungen – vor allem im Vergleich zum Laborbetrieb größere Schwankungen des Gasdrucks vor der Fluss- und Druckregelung – Rechnung zu tragen.

Die Analyse der Heizvorgänge der Ventile in der Ventilmatrix zeigt, dass die Bestimmung der Ventilpositionen mit der momentan verwendeten Methode zuverlässig funktioniert. Die Ventile 21, 46 und 64 bilden hier eine Ausnahme. Ihr unzuverlässiges Heizverhalten ist sehr wahrscheinlich hardwarebedingt und muss noch genauer untersucht werden. Es konnte jedoch gezeigt werden, dass die im Rahmen dieser Arbeit entwickelten Programme solche Fehler entdecken und sich gut für deren Untersuchung eignen.

Für eine komplette quantitative Validierung der Ventilauslese müssen mehr Heizverläufe von allen Ventilen in allen Ventilpositionen aufgenommen und ausgewertet werden. Dies sollte geschehen wenn der Messschrank seine endgültige Ausbauform (mit allen Seitenwände und Türen, zusätzlicher Lüftung, etc.) erreicht hat. Dann müssen die Parameter der Ventilpositionsbestimmung (Heizzeit und Erkennungsschwelle) nämlich sowieso noch einmal überprüft werden.

Danksagung

Ich danke Prof. Dr. Thomas Hebbeker für die Möglichkeit diese Bachelorarbeit zu schreiben. Außerdem gilt mein Dank allen Personen, die an diesem Projekt beteiligt sind oder waren und mit ihren Leistungen meine Arbeit erst möglich gemacht haben. Besonders möchte ich mich bei Carsten Heidemann und Dr. Hans Reithler bedanken, welche mich beim Erstellen dieser Arbeit sehr gut unterstützt haben.

Literatur

[1] EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH (CERN): *The Compact Muon Solenoid Experiment.* cms.web.cern.ch/cms/Detector/index.html.

[2] REITHLER, H.: *Private Mitteilung.* Physikalisches Institut IIIA der RWTH Aachen, 2010.

[3] W. BLUM, W. RIEGLER, L. ROLANDI: *Particle detection with drift chambers.* Springer, 2008.

[4] FERNOW, R. C.: *Introduction to Experimental Particle Physics.* Cambridge University Press, 1986.

[5] HEIDEMANN, C.: *Direkte Messung der Driftgeschwindigkeit im Gas der CMS-Myonenkammern mit VDC-Kammern.* Physikalisches Institut IIIA der RWTH Aachen, 2010.

[6] ALTENHÖFER, G.: *Development of a Drift Chamber for Drift Velocity Monitoring in the CMS Barrel Muon System.* Physikalisches Institut IIIA der RWTH Aachen, 2006.

[7] FRANGENHEIM, J.: *Measurements of the drift velocity using a small gas chamber for monitoring of the CMS muon system.* Physikalisches Institut IIIA der RWTH Aachen, 2007.

[8] SOWA, M.: *Aufbau, Kalibration und Anwendung einer Messapparatur zur Überwachung des Gasdrucks in den CMS-Myonkammern.* III. Physikalisches Institut A der RWTH Aachen, 2003.

[9] A. PAULUS, G. HILGERS, F. ADAMCZYK C. HEIDEMANN H. REITHLER: *VME-Einschub „VALVE READ-OUT“.* Entwickelt am Phys. Inst. IIIA, RWTH Aachen.

[10] BRONKHORST HIGH-TECH B.V.: *Instruction manual RS232 interface.* www.bronkhorst.com.

[11] *comedi - linux control and measurement device interface.* www.comedi.org.

[12] *ROOT - An Object-Oriented Data Analysis Framework.* root.cern.ch.

[13] TEYSSIER, D.: *Private Mitteilung.* Physikalisches Institut IIIA der RWTH Aachen.

[14] NATIONAL INSTRUMENTS: *LabVIEW.* www.ni.com/labview/.

Eidesstattliche Erklärung

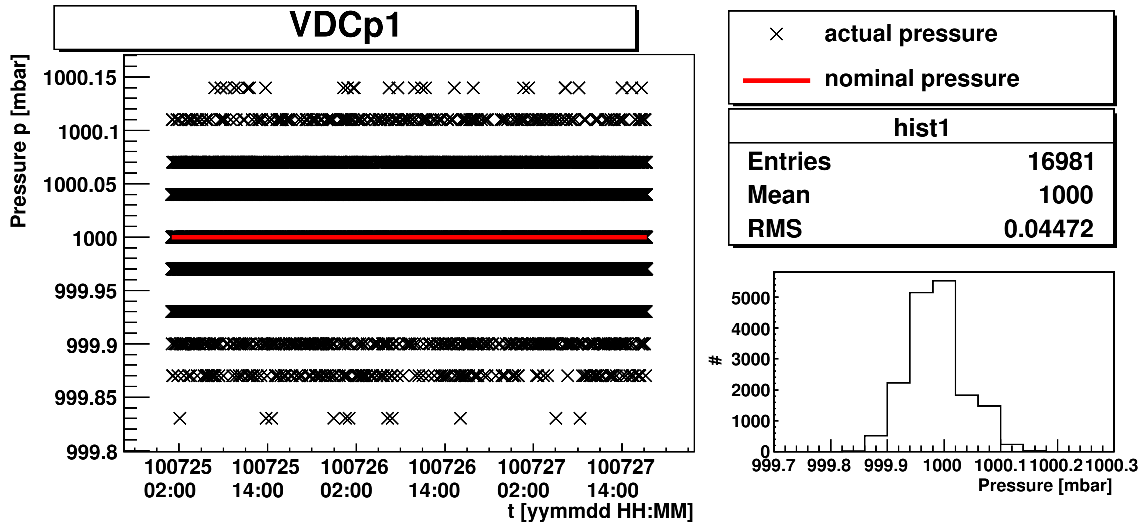
Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

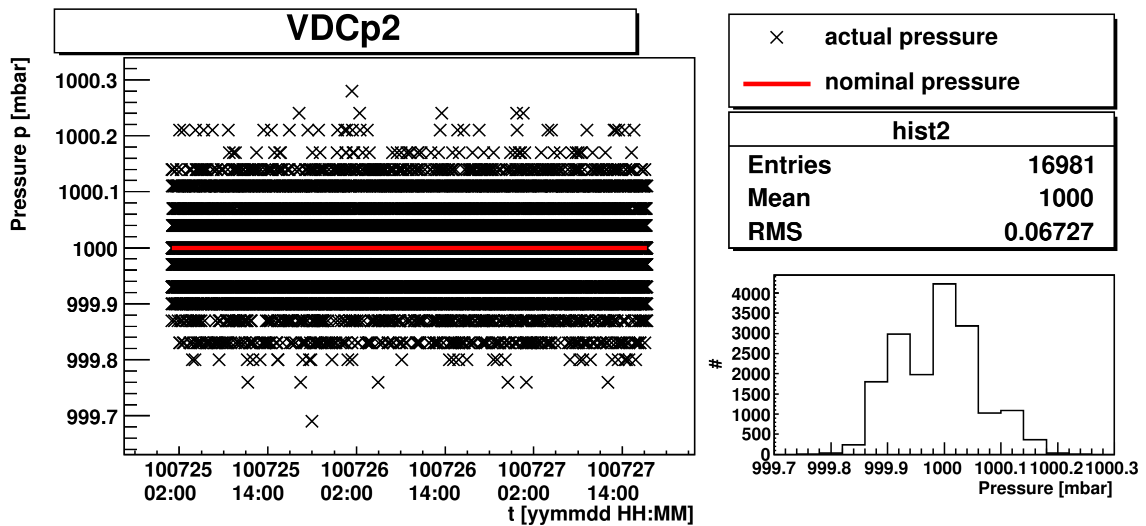
Aachen, den 11. August 2010

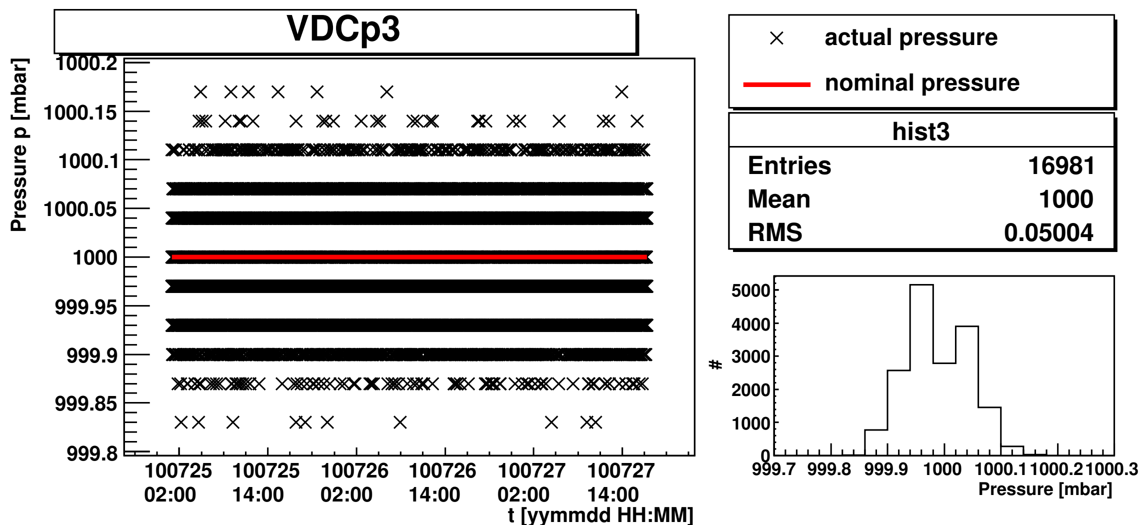
Lukas Koch

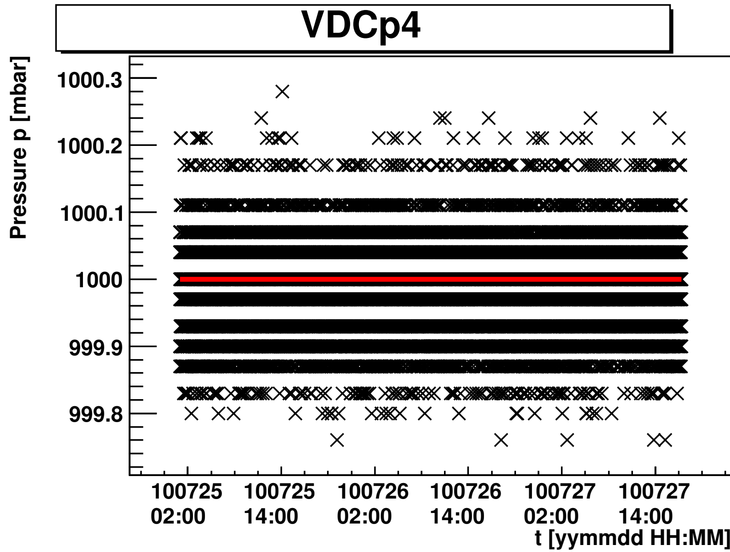
A Plots

A.1 Druckstabilitätsmessung





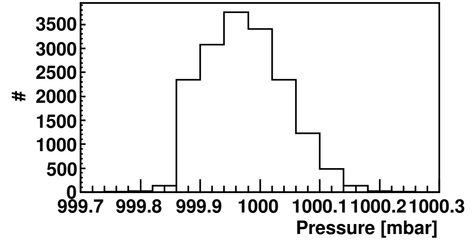


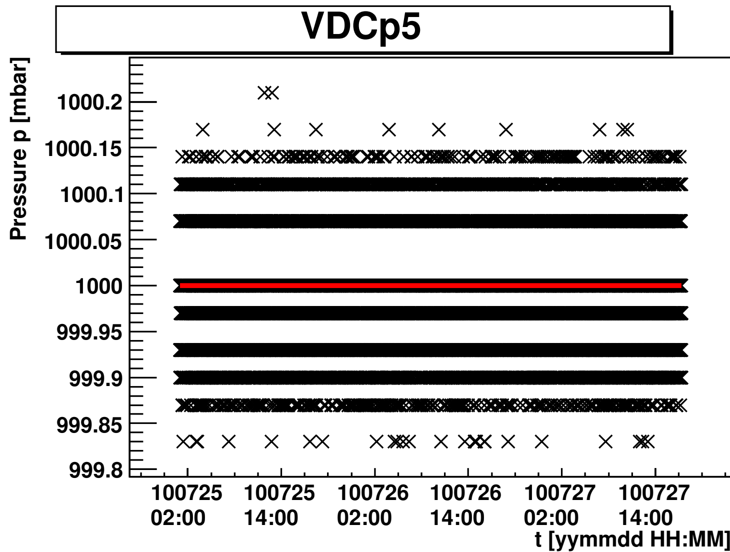


× actual pressure

nominal pressure

hist4	
Entries	16981
Mean	1000
RMS	0.06221

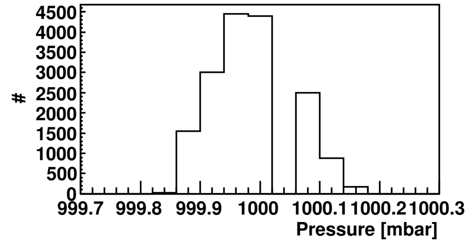


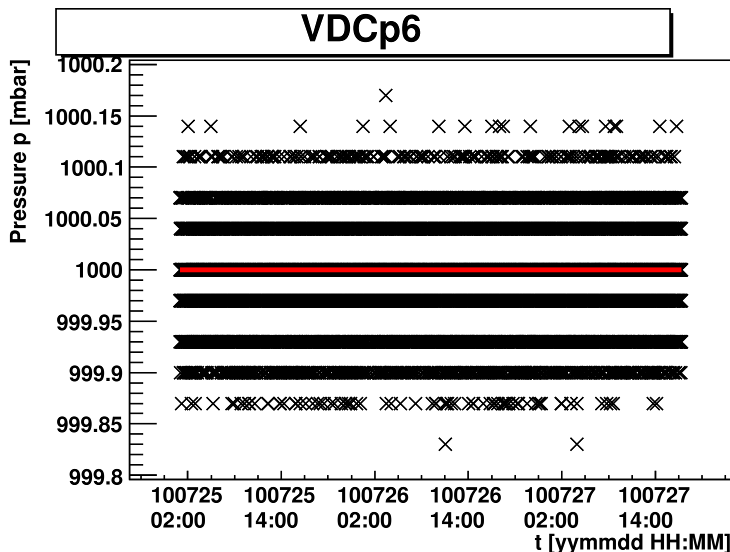


× actual pressure

nominal pressure

hist5	
Entries	16981
Mean	1000
RMS	0.05978

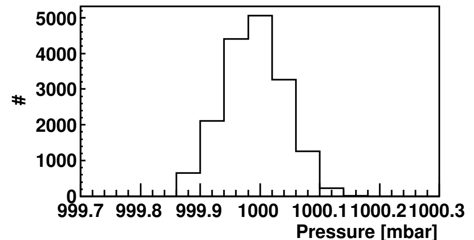




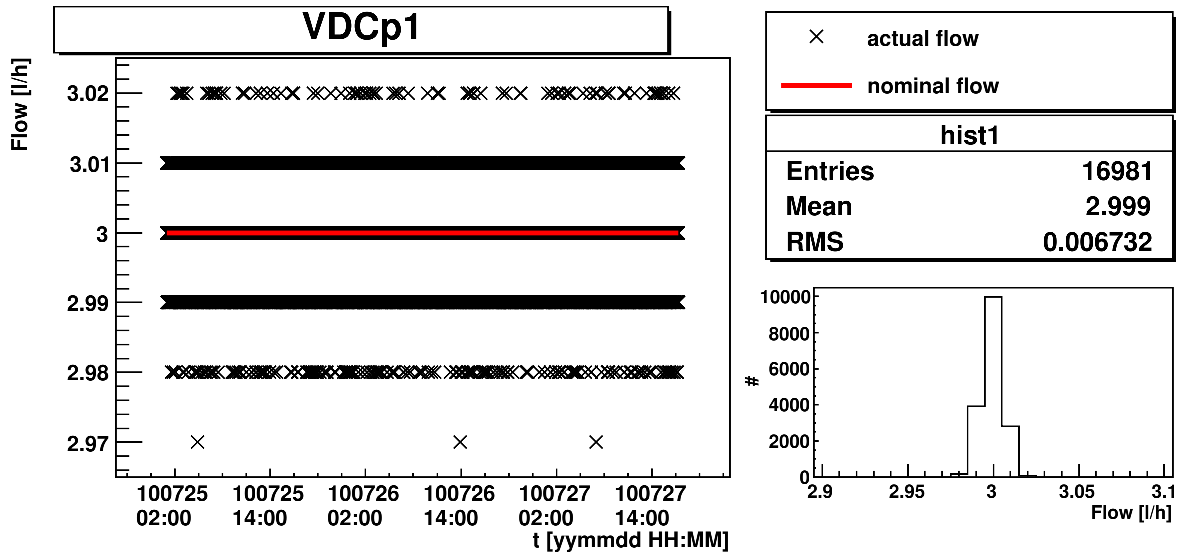
× actual pressure

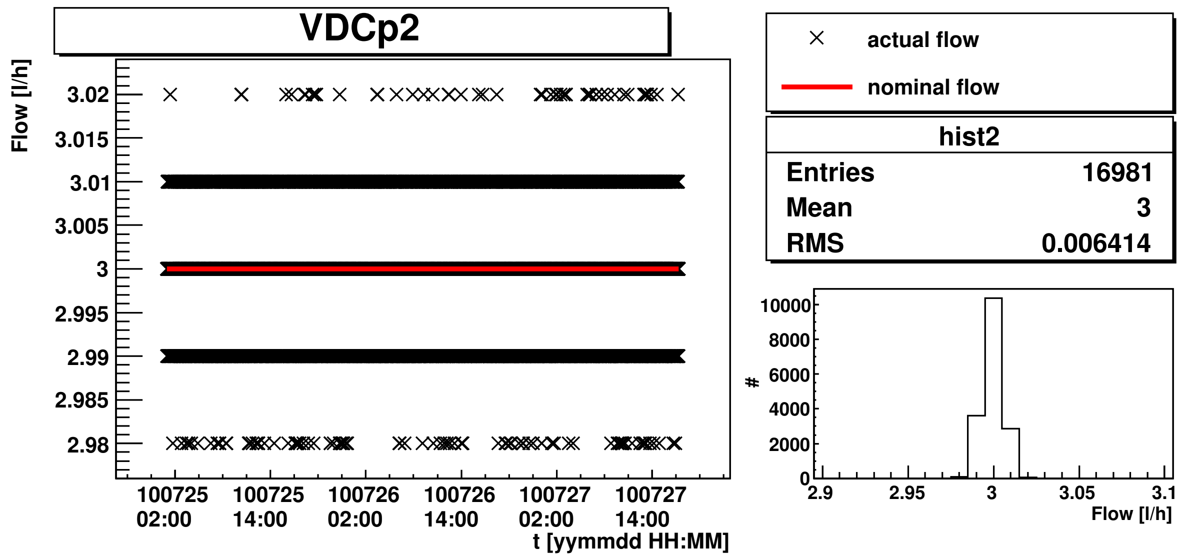
nominal pressure

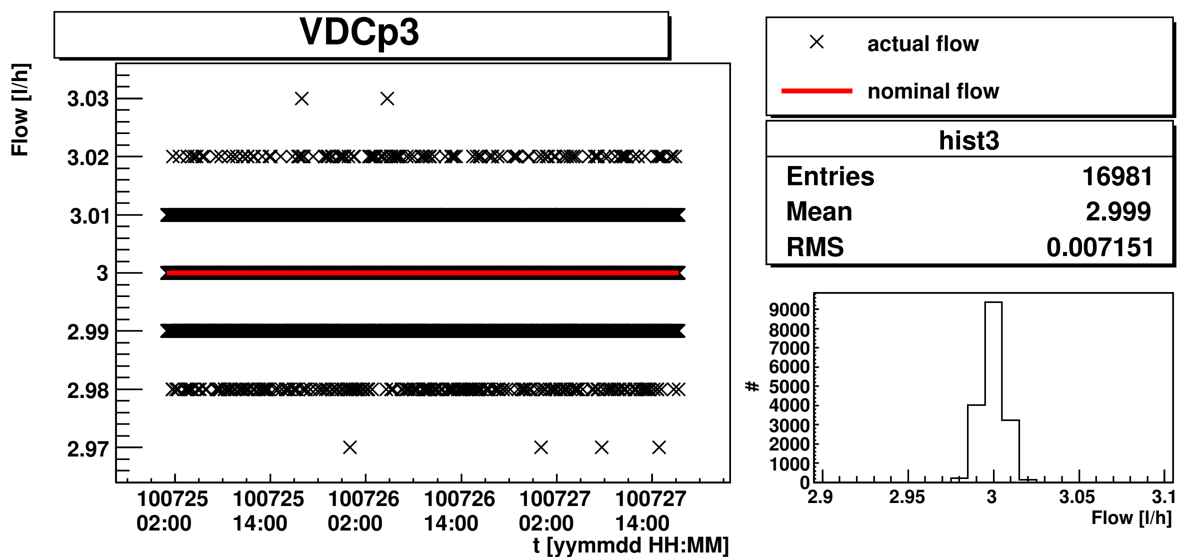
hist6	
Entries	16981
Mean	1000
RMS	0.04554

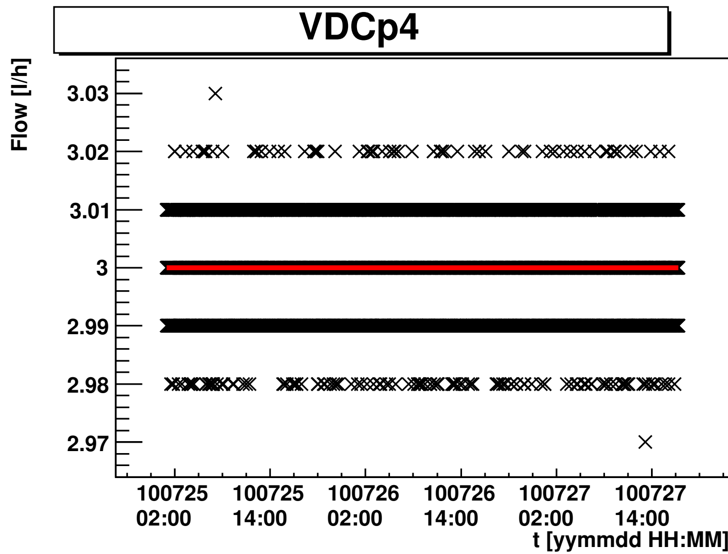


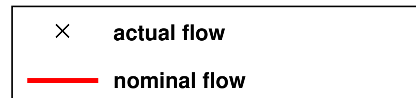
A.2 Flussstabilitätsmessung



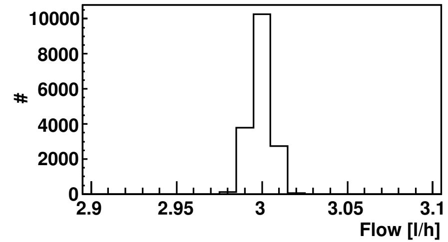


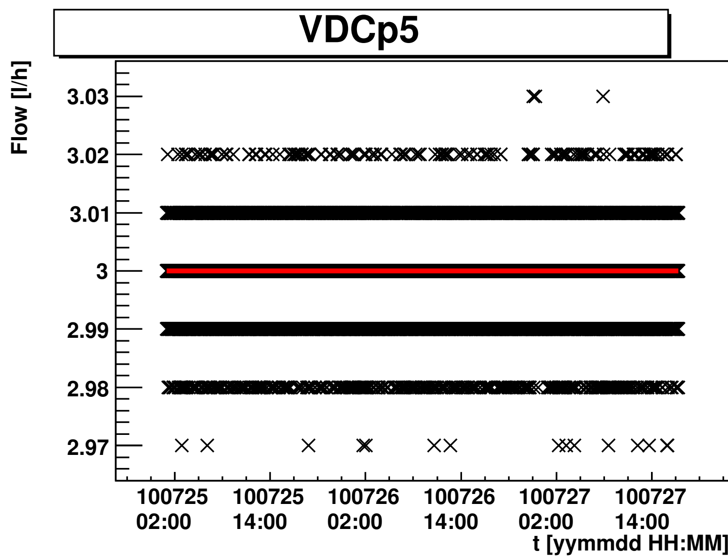


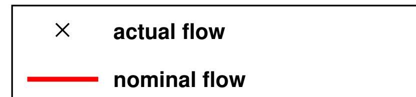




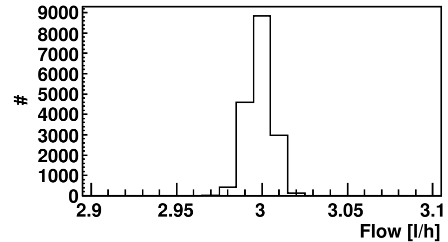
hist4	
Entries	16981
Mean	2.999
RMS	0.006513

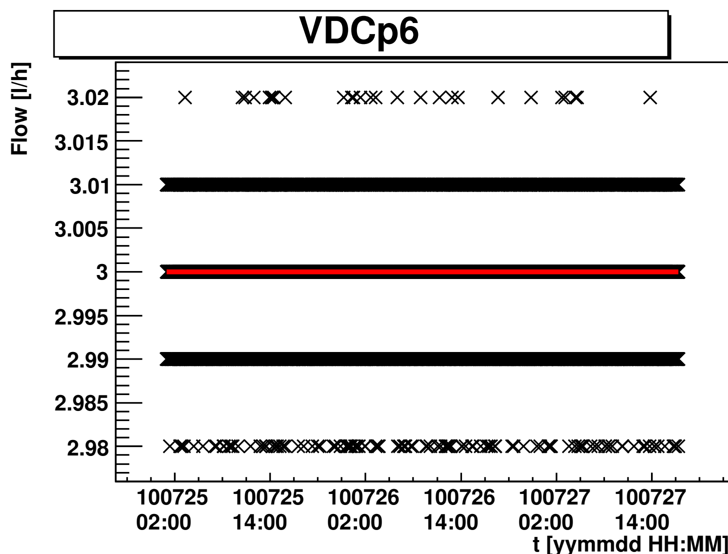


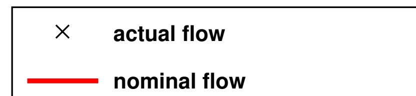




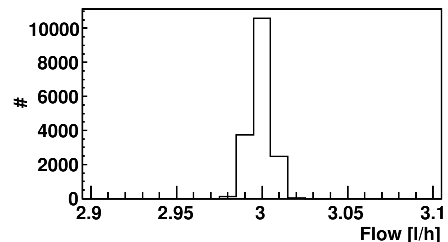
hist5	
Entries	16981
Mean	2.999
RMS	0.007538



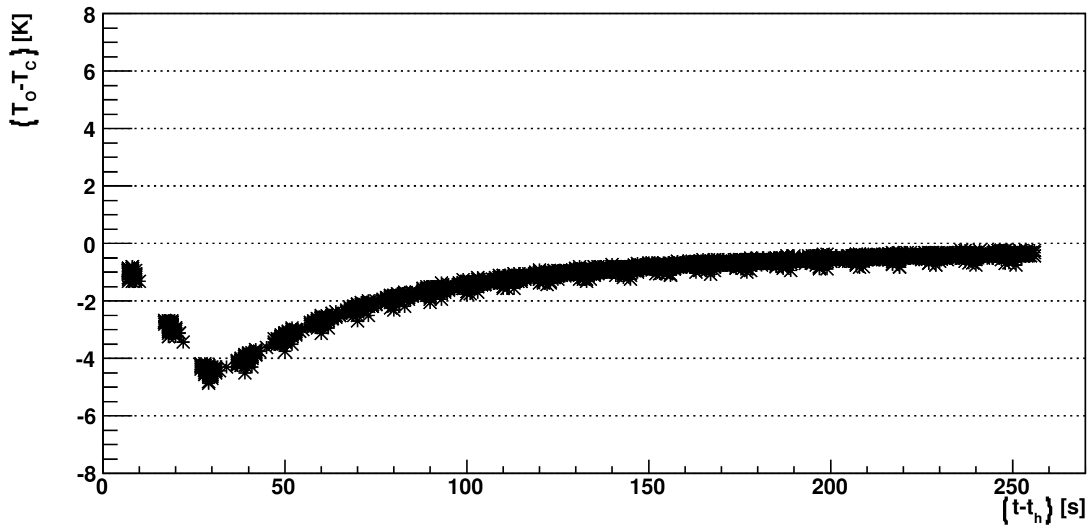


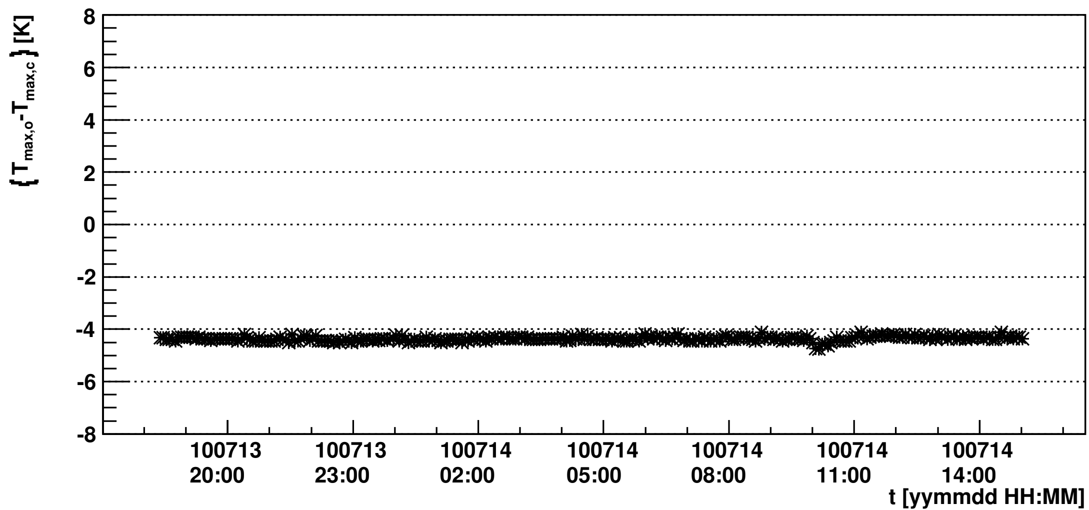


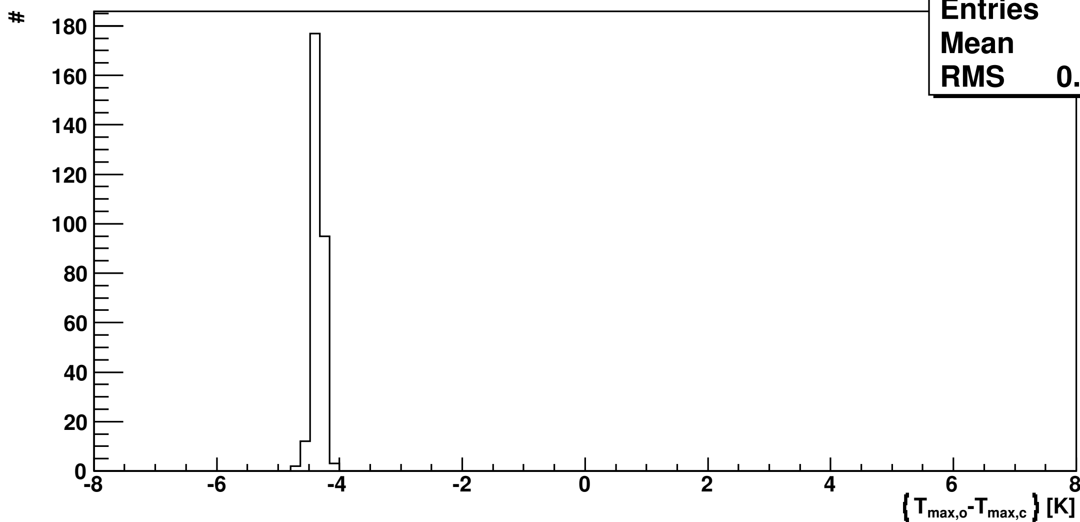
hist6	
Entries	16981
Mean	2.999
RMS	0.006282

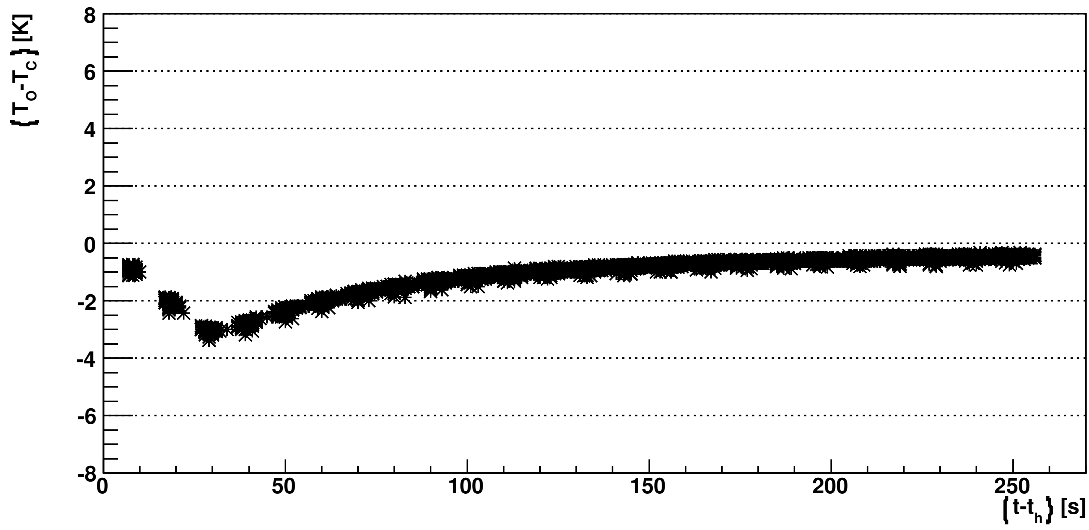


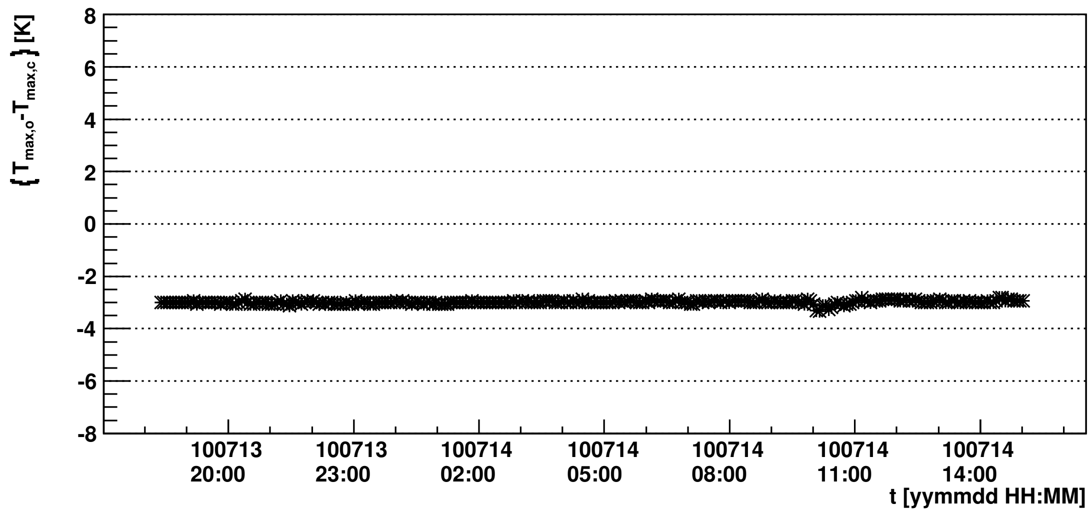
A.3 Ventilauslese

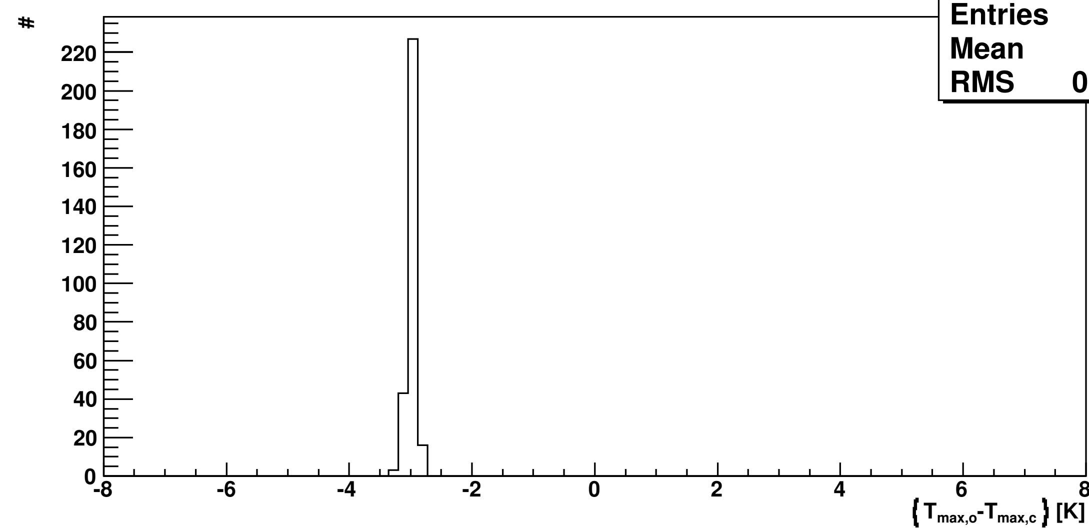


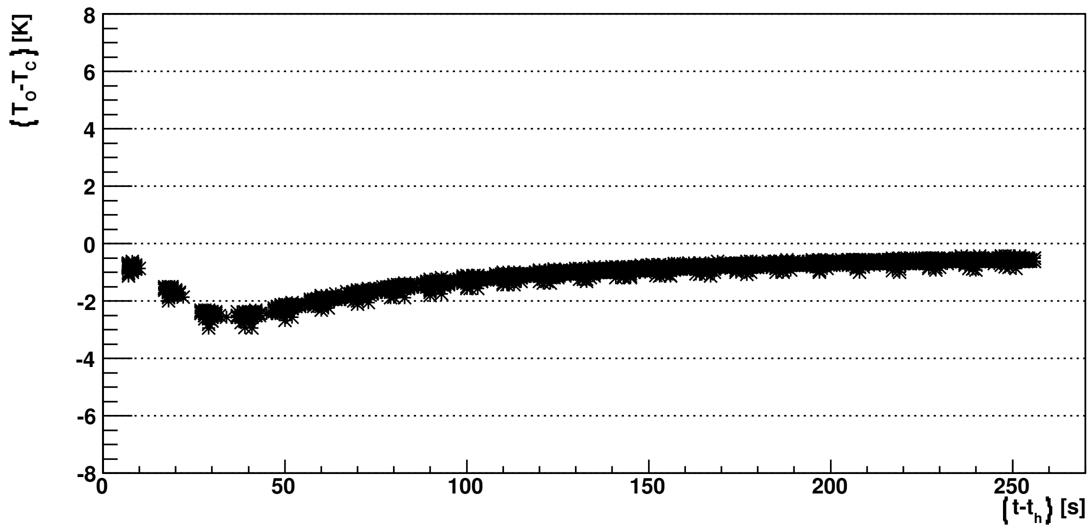


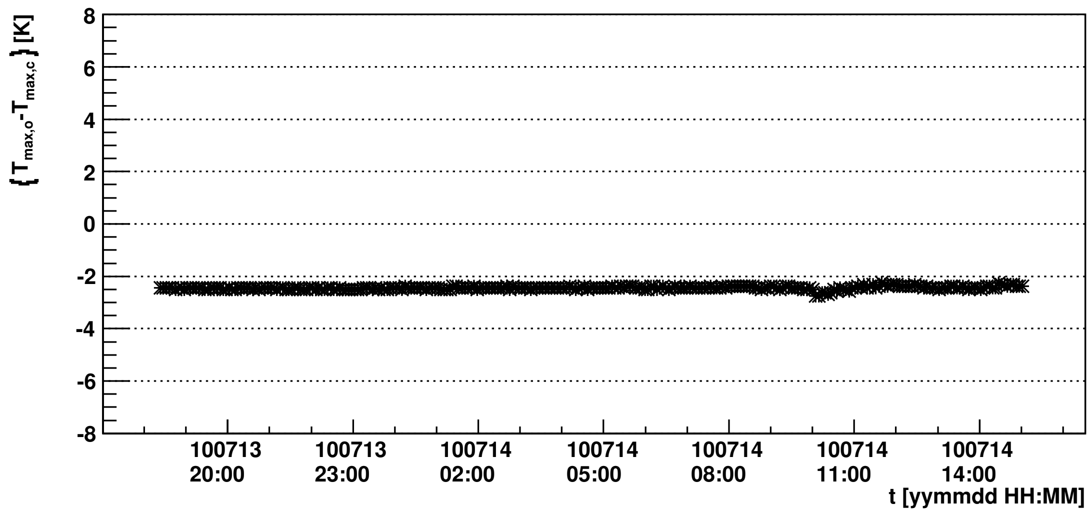


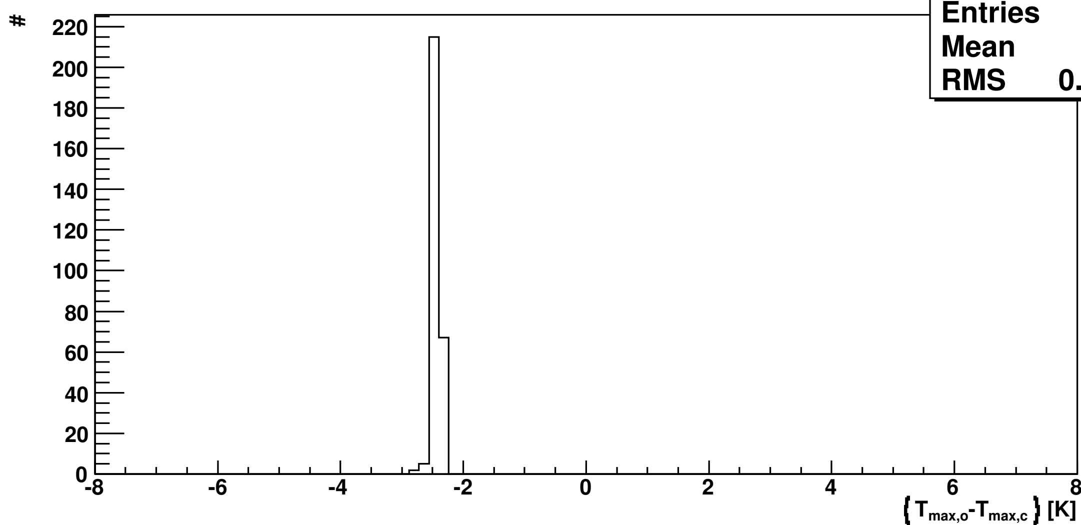

Valve 12: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

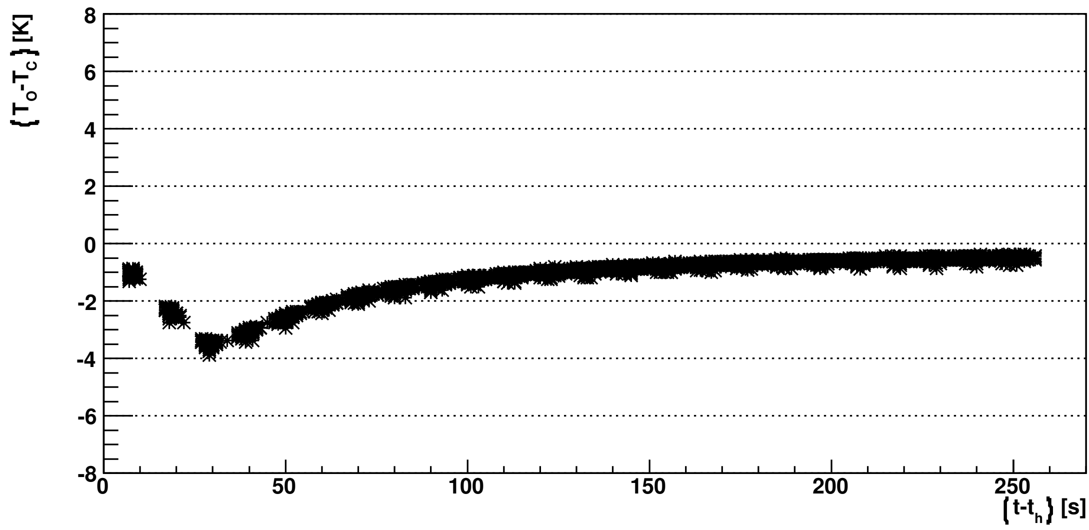

Valve 12: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

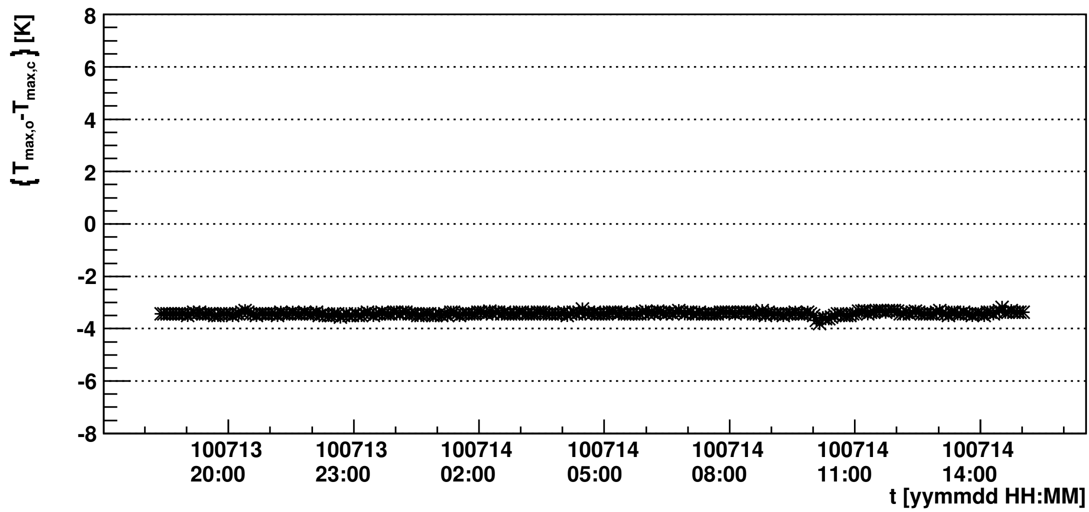

Valve 12: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -2.988
RMS 0.06311
#
220
200
180
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

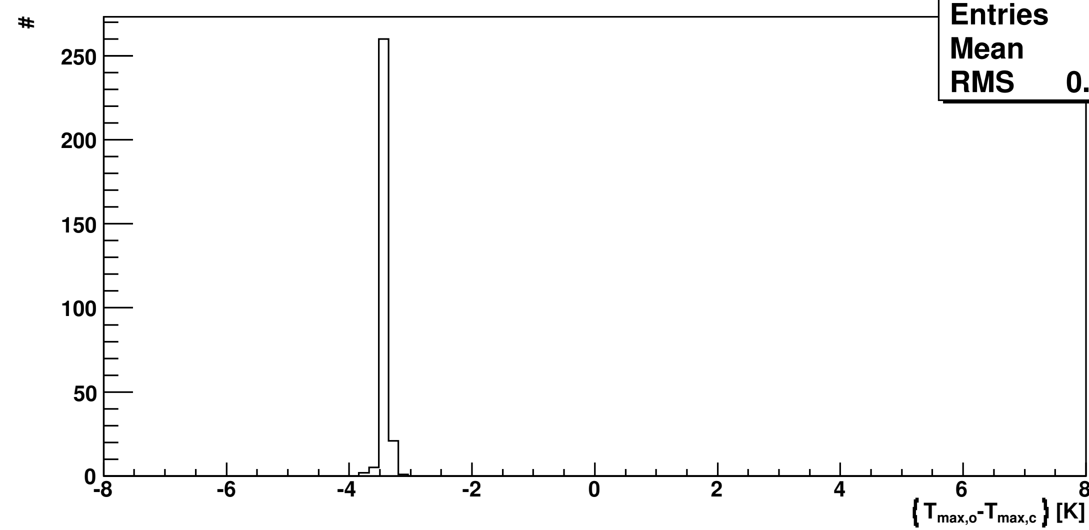

Valve 13: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

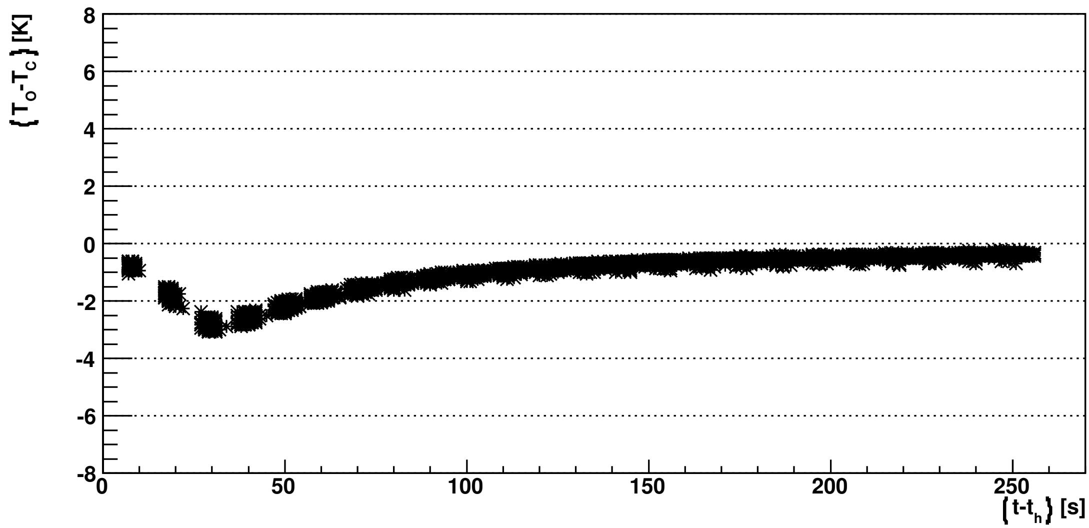

Valve 13: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

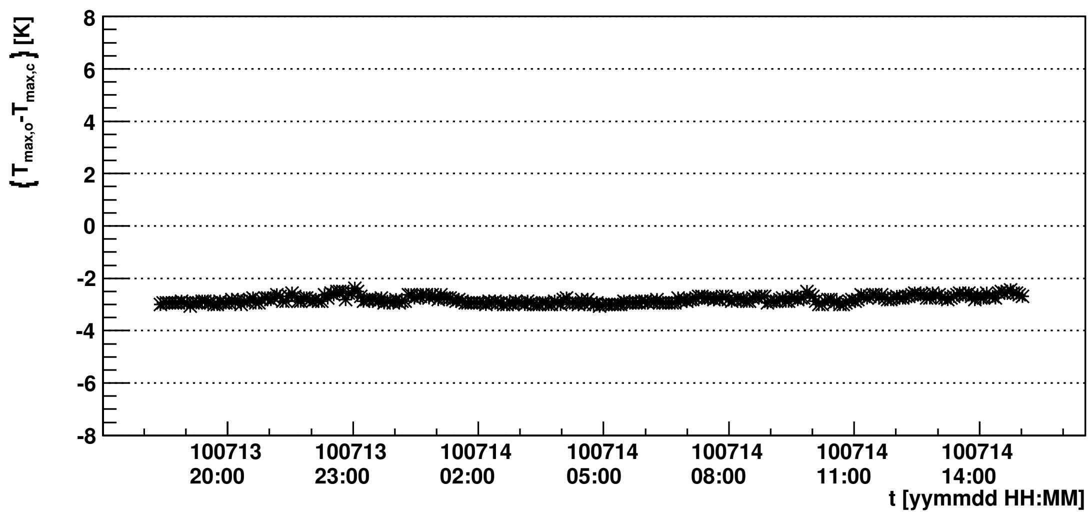

Valve 13: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -2.439
RMS 0.06113
#
220
200
180
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

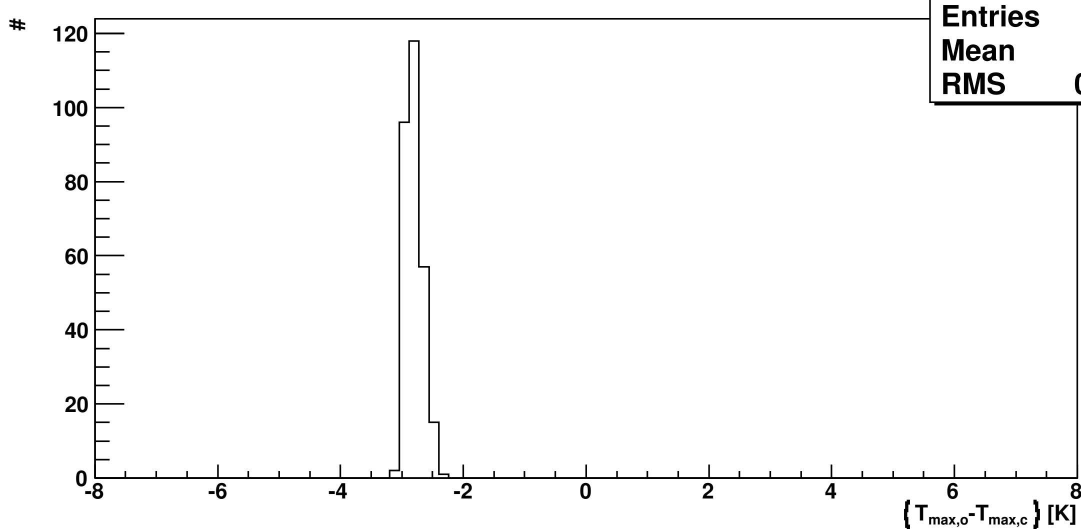

Valve 14: Opentemp - Closetemp
$\{T_O-T_C\}$ [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
$\{t-t_h\}$ [s]

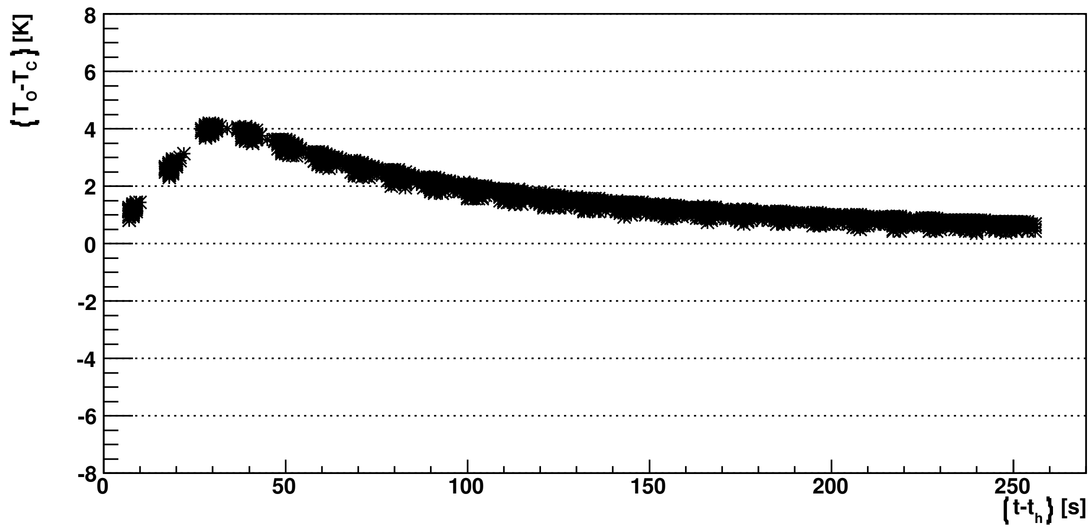

Valve 14: OpenMaxtemp - CloseMaxtemp
$\{T_{max,o}-T_{max,c}\}$ [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

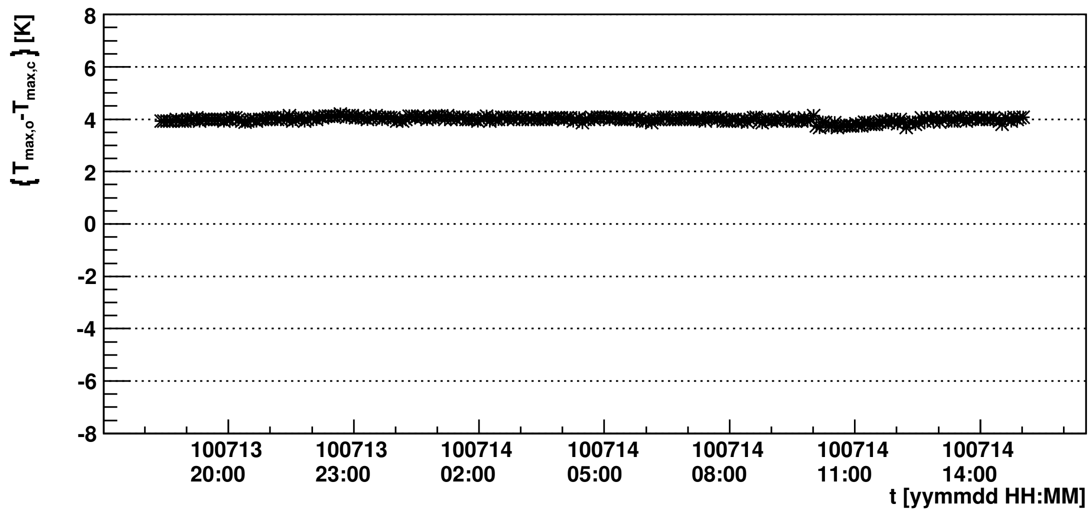

Valve 14: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -3.423
RMS 0.06156
#
250
200
150
100
50
0
-8
-6
-4
-2
0
2
4
6
8
$\{T_{max,o}-T_{max,c}\}$ [K]

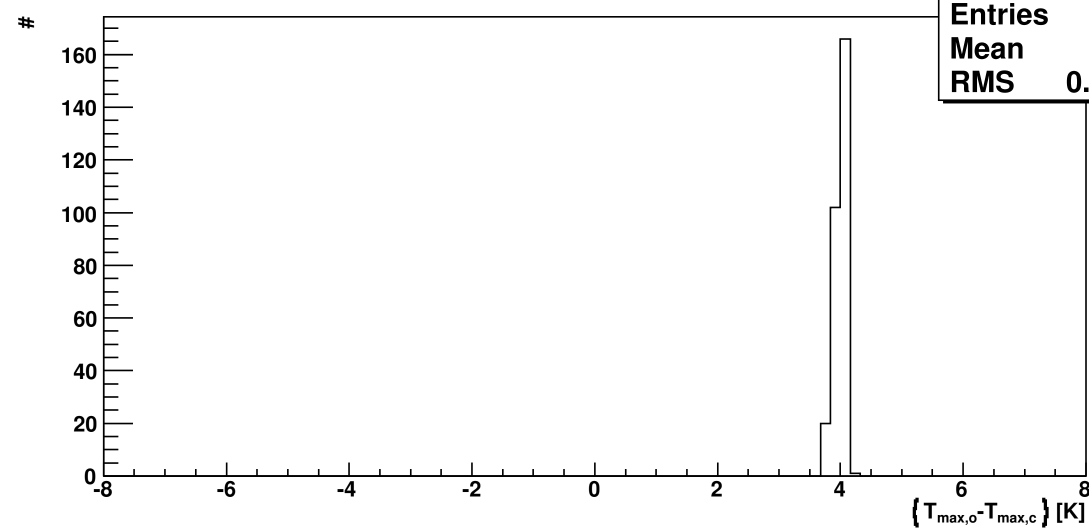

Valve 15: Opentemp - Closetemp
{T_O-T_C} [K]
{t-t_h} [s]

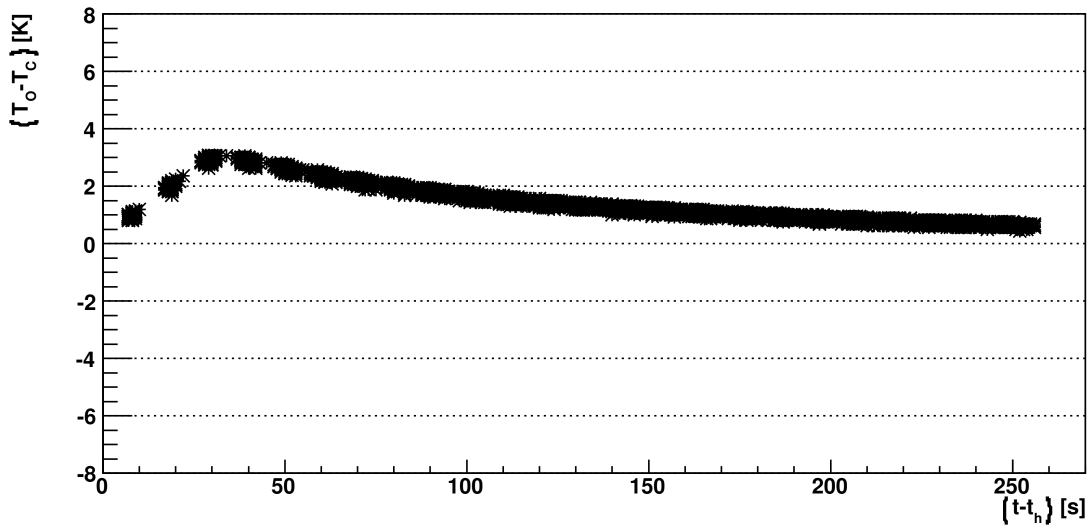

Valve 15: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

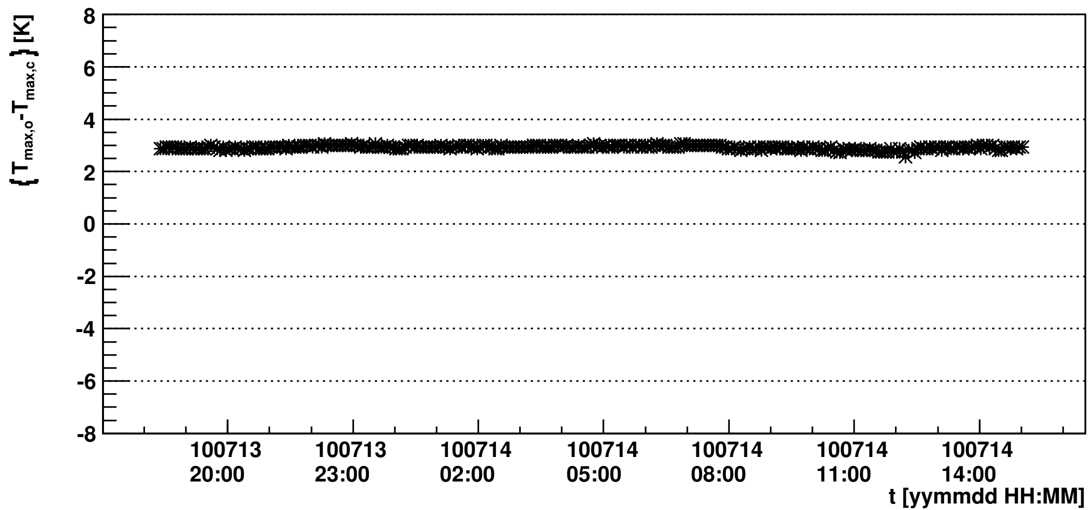

Valve 15: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -2.811
RMS 0.1383
#
{T_max,o-T_max,c} [K]

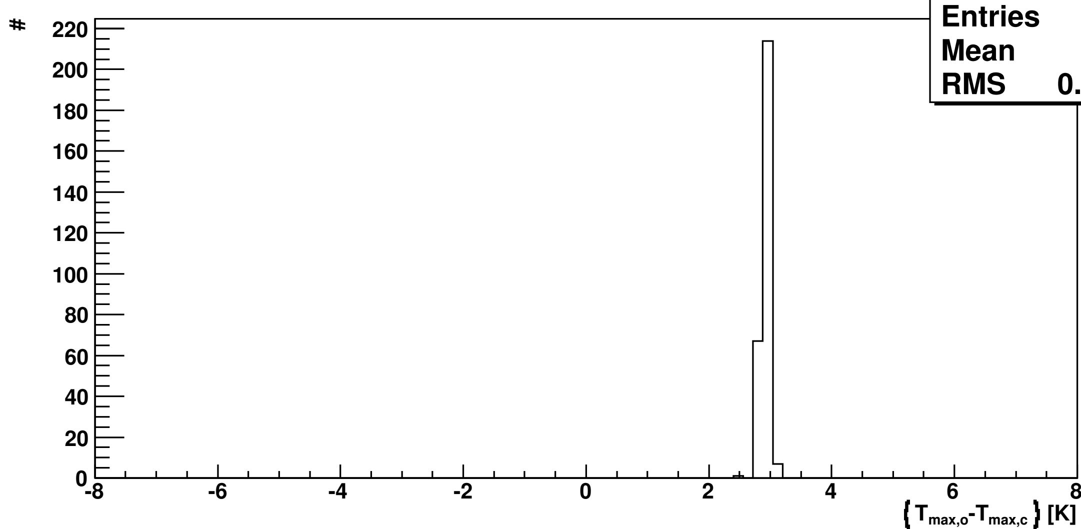

Valve 16: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

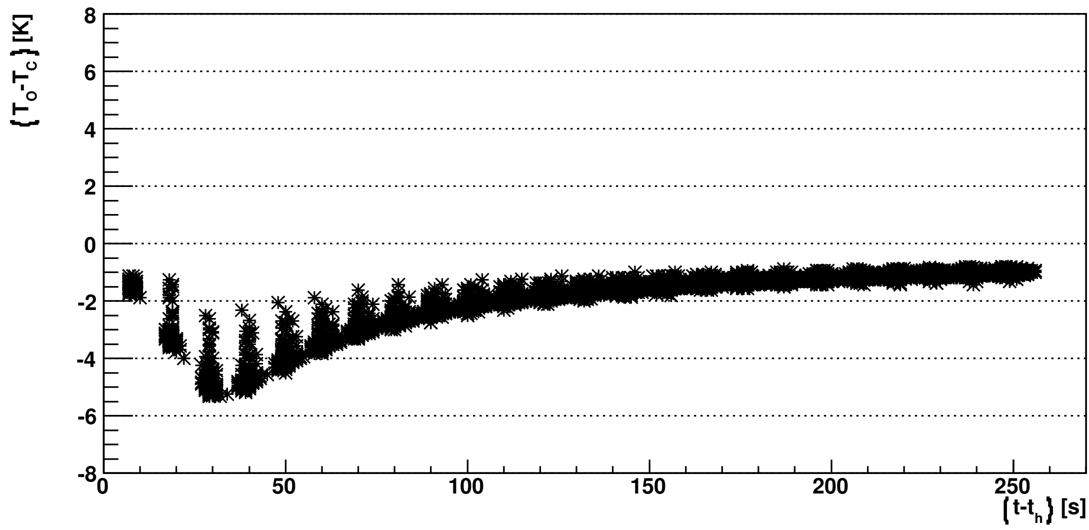

Valve 16: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

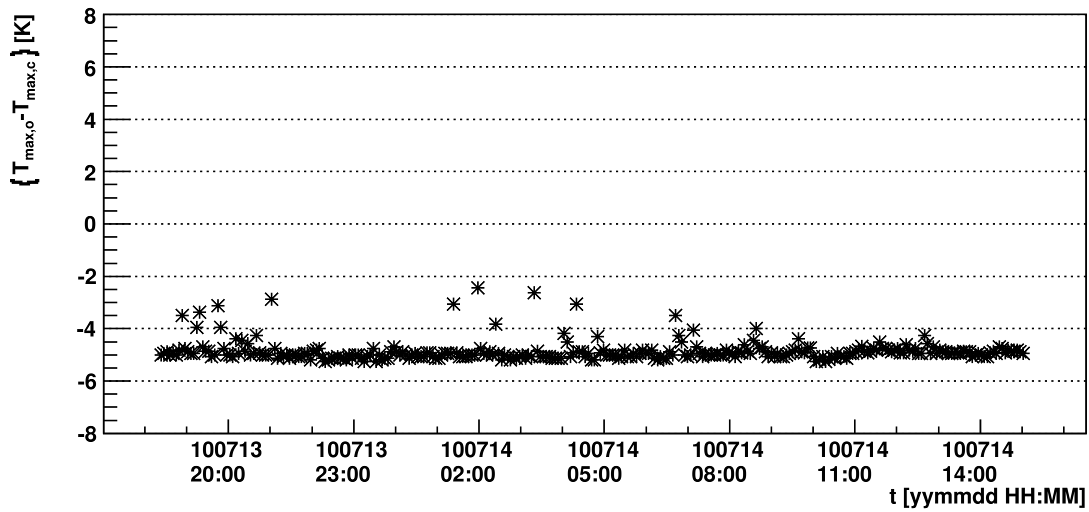

Valve 16: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean 3.995
RMS 0.08945
#
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

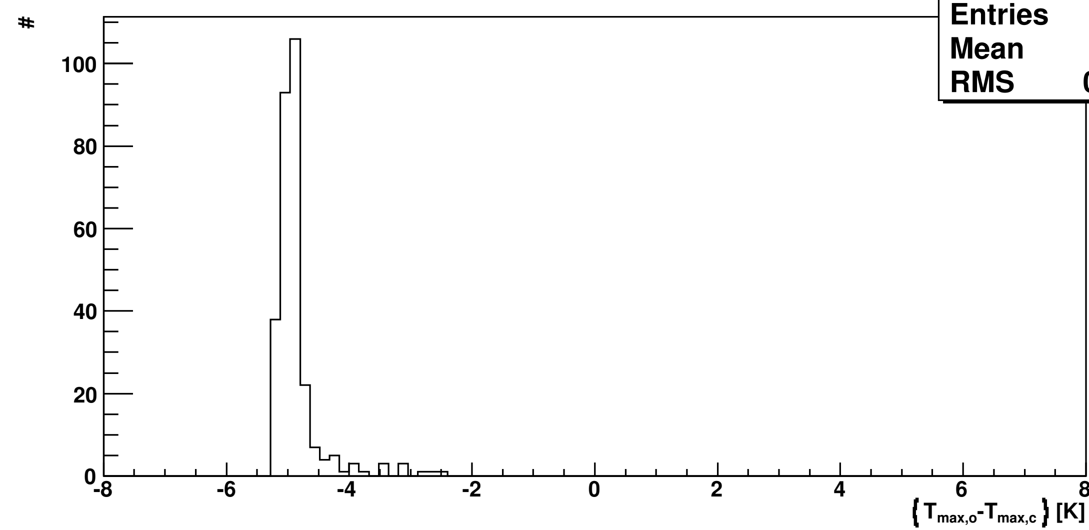

Valve 17: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

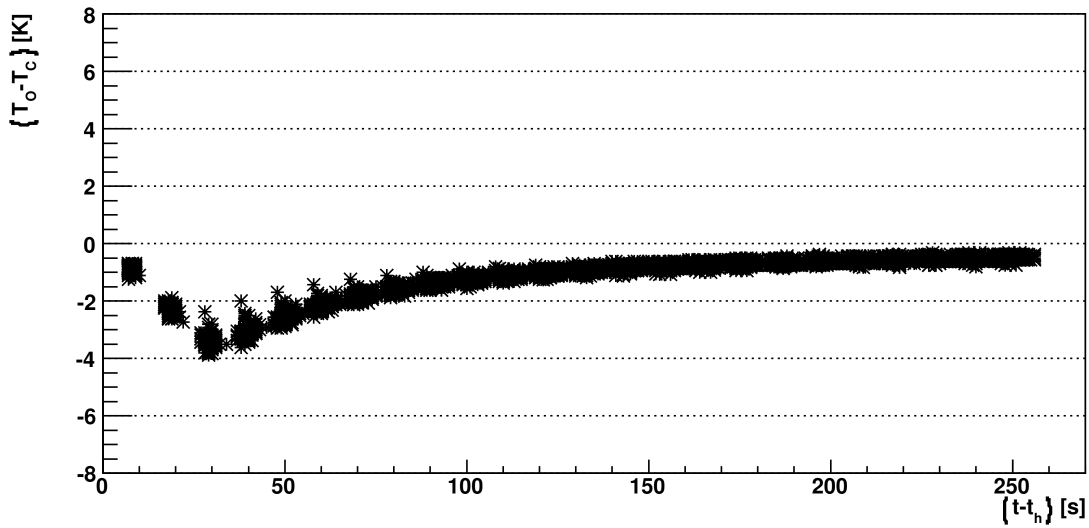

Valve 17: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

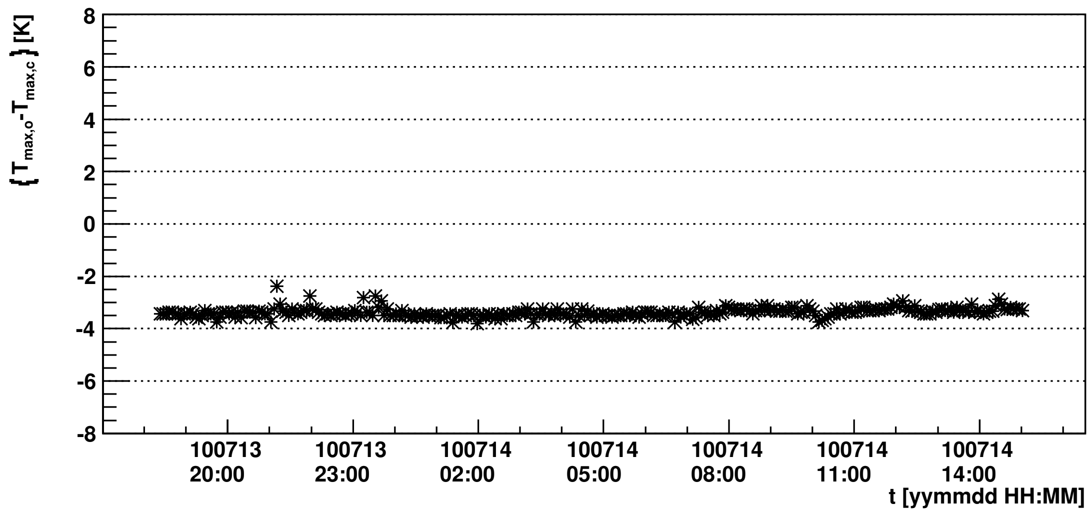

Valve 17: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean 2.923
RMS 0.07037
#
220
200
180
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

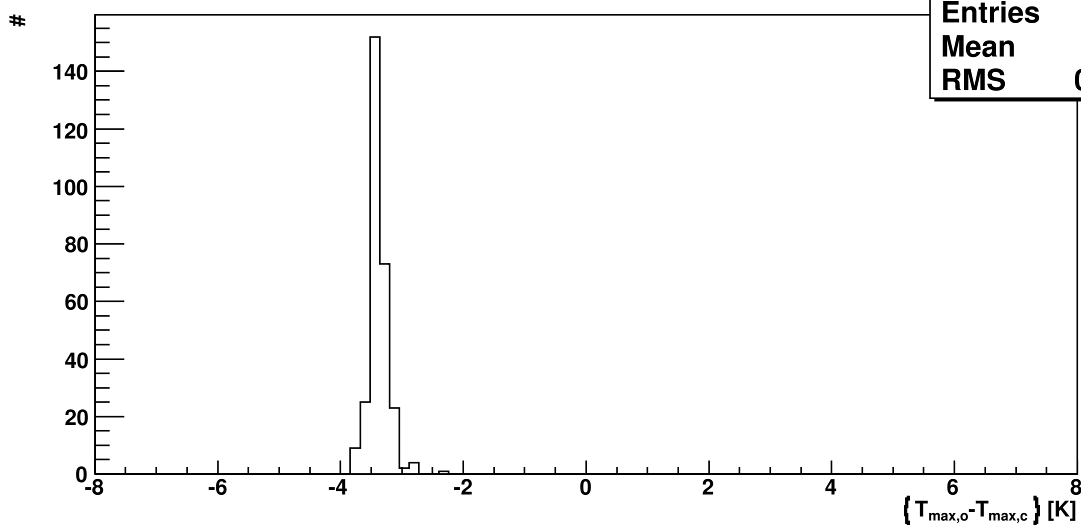

Valve 21: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

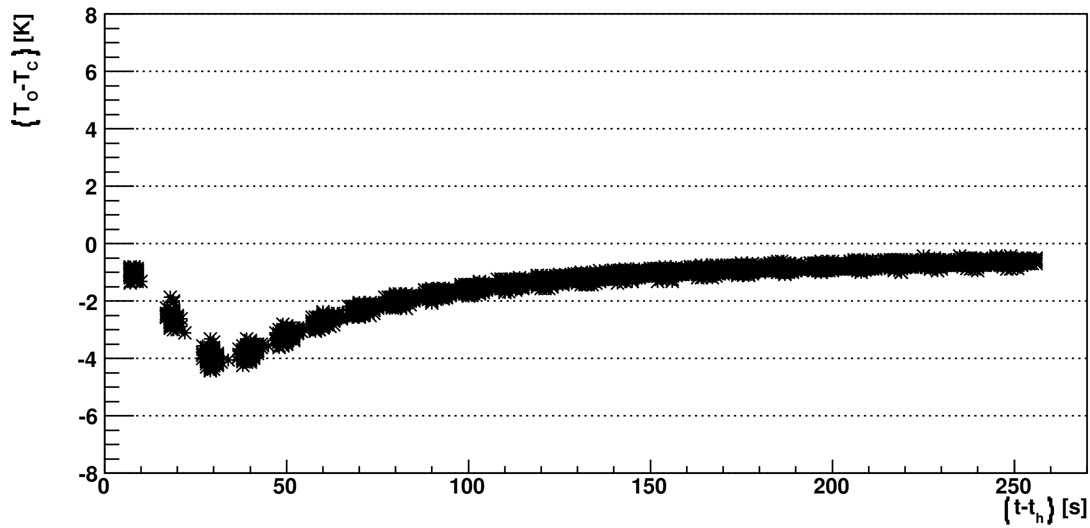

Valve 21: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

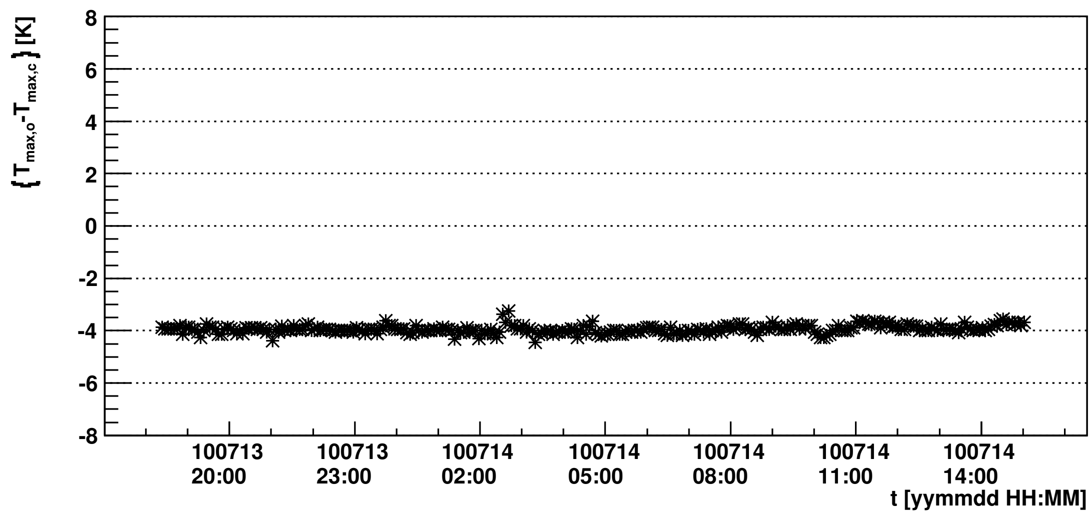

Valve 21: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -4.864
RMS 0.3943
#
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

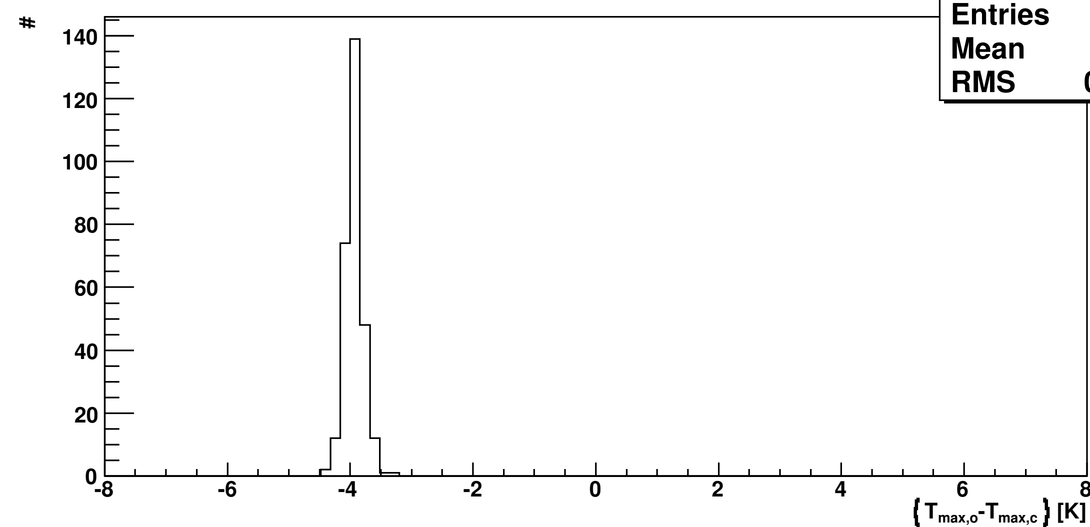

Valve 22: Opentemp - Closetemp
{T_o-T_c} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

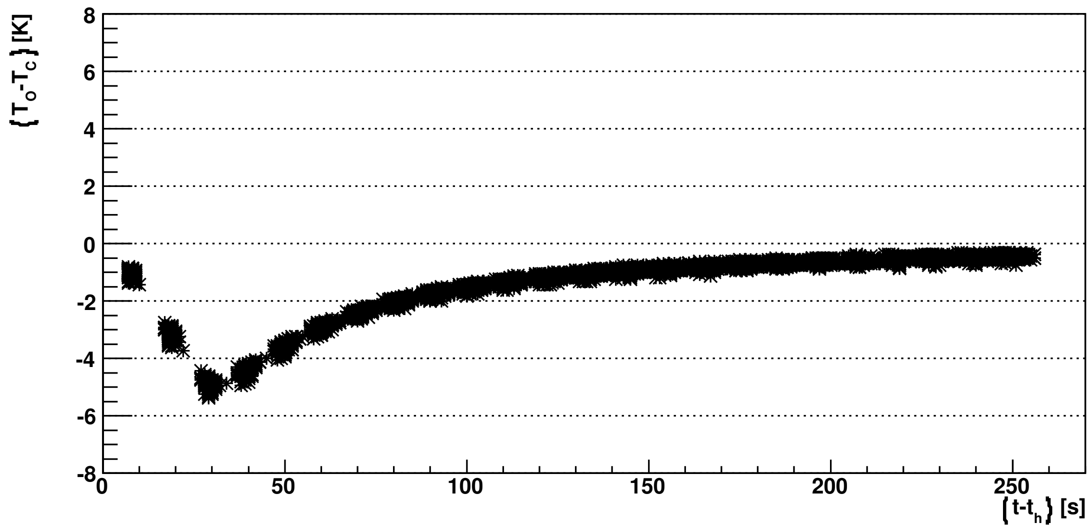

Valve 22: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

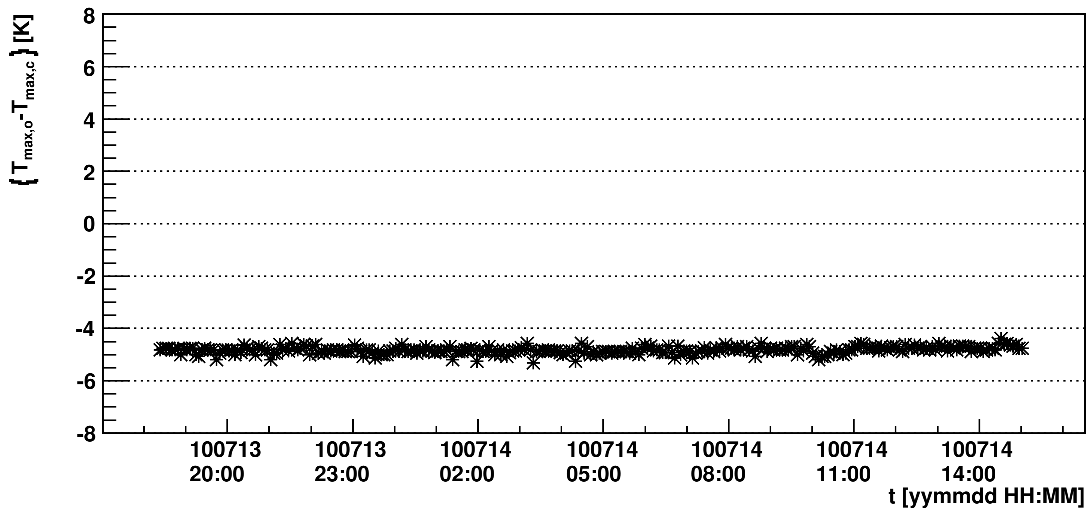

Valve 22: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -3.383
RMS 0.1659
#
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

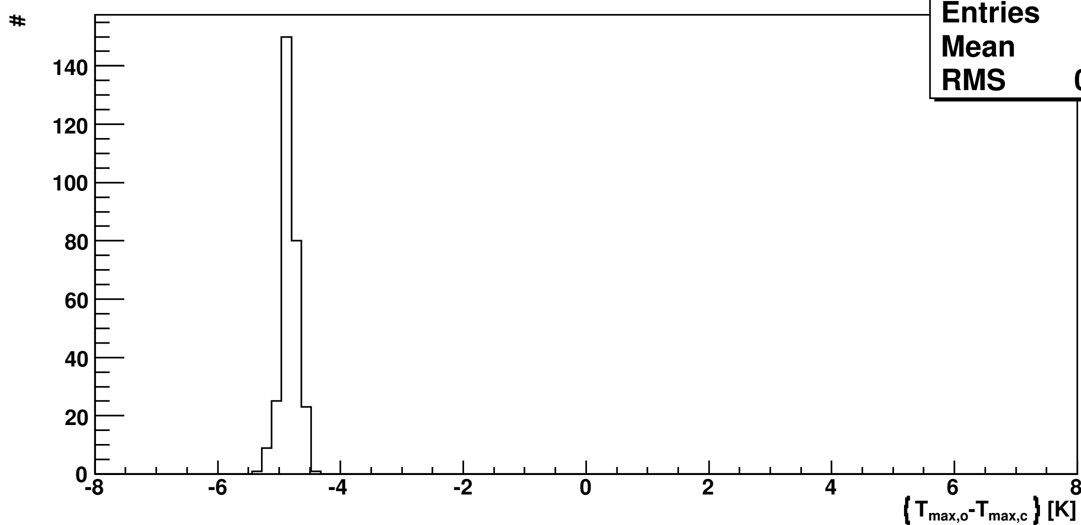

Valve 23: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

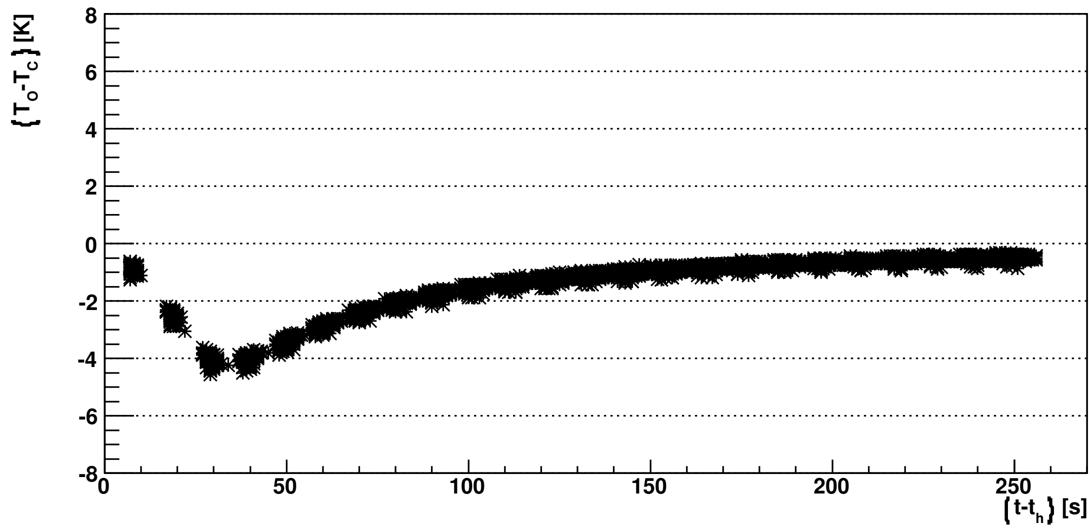

Valve 23: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

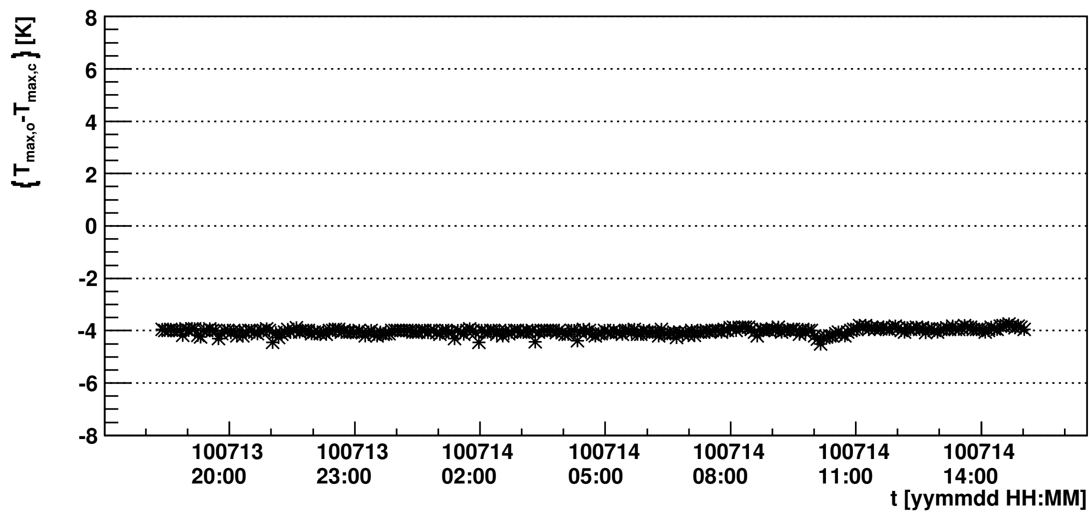

Valve 23: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -3.945
RMS 0.1495
#
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

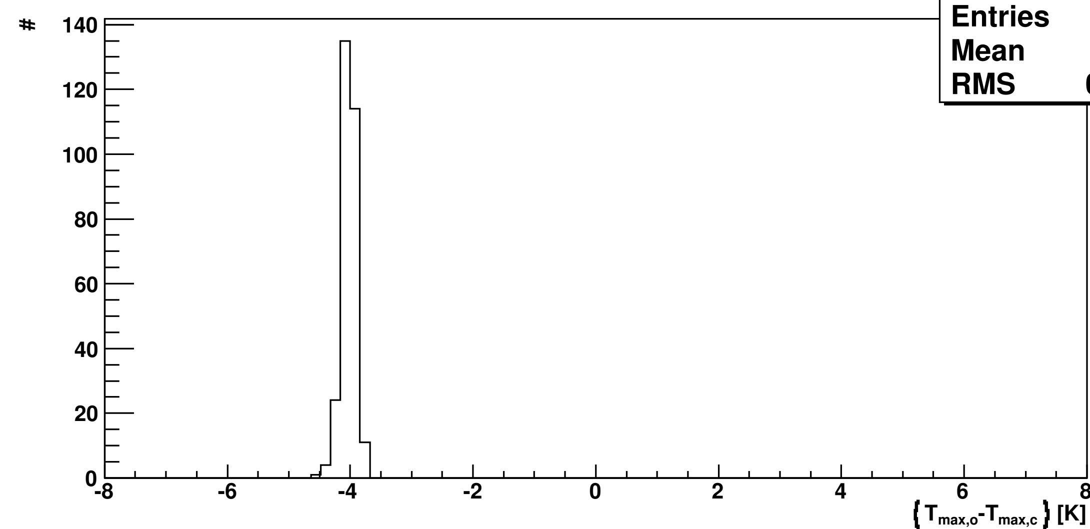

Valve 24: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

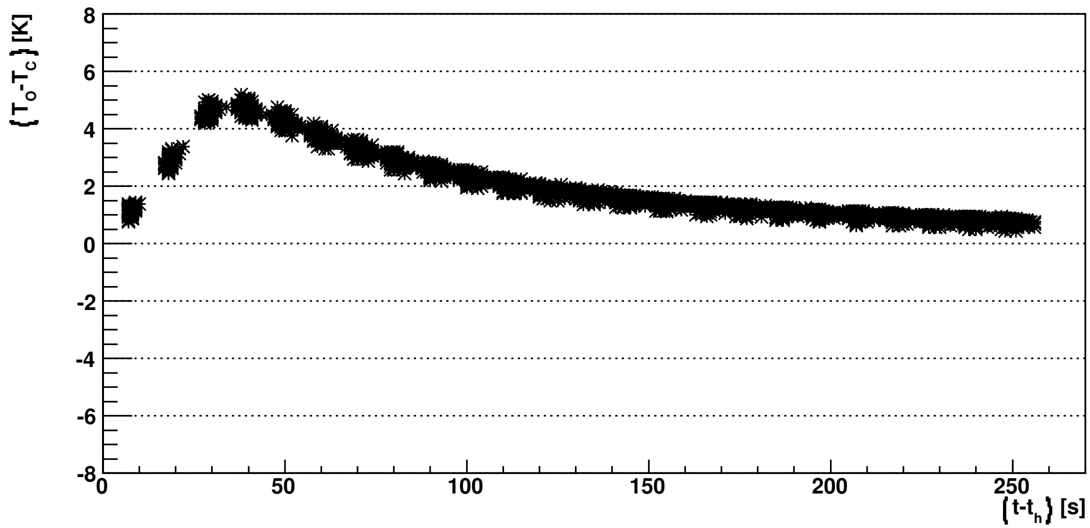

Valve 24: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

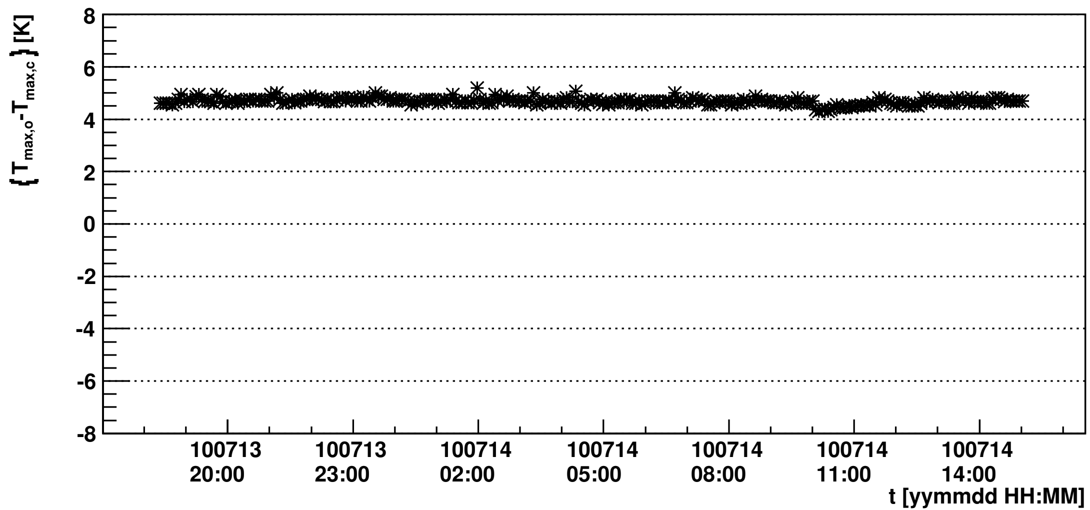

Valve 24: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -4.826
RMS 0.1323
#
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

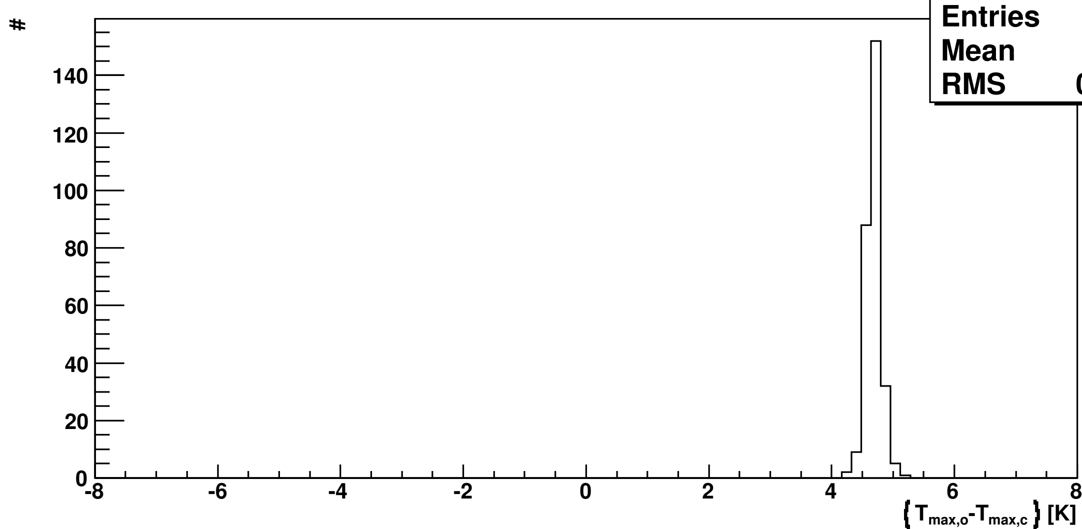

Valve 25: Opentemp - Closetemp
{T_o-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

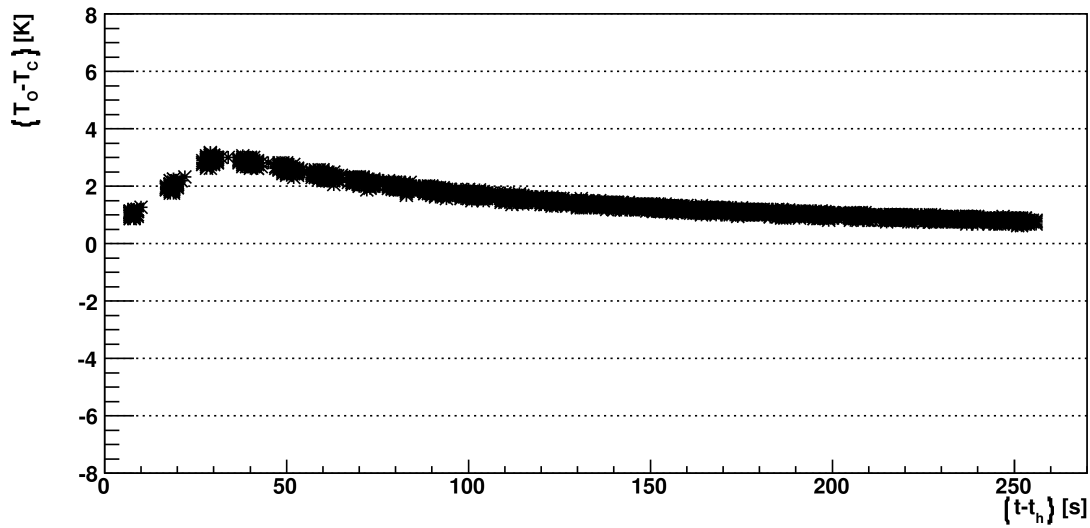

Valve 25: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

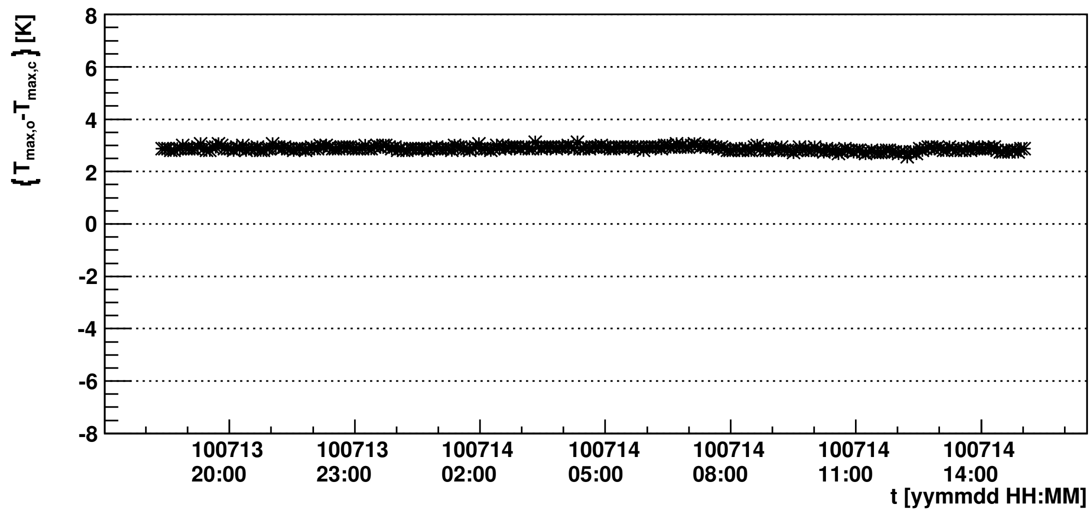

Valve 25: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -4.024
RMS 0.1122
#
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

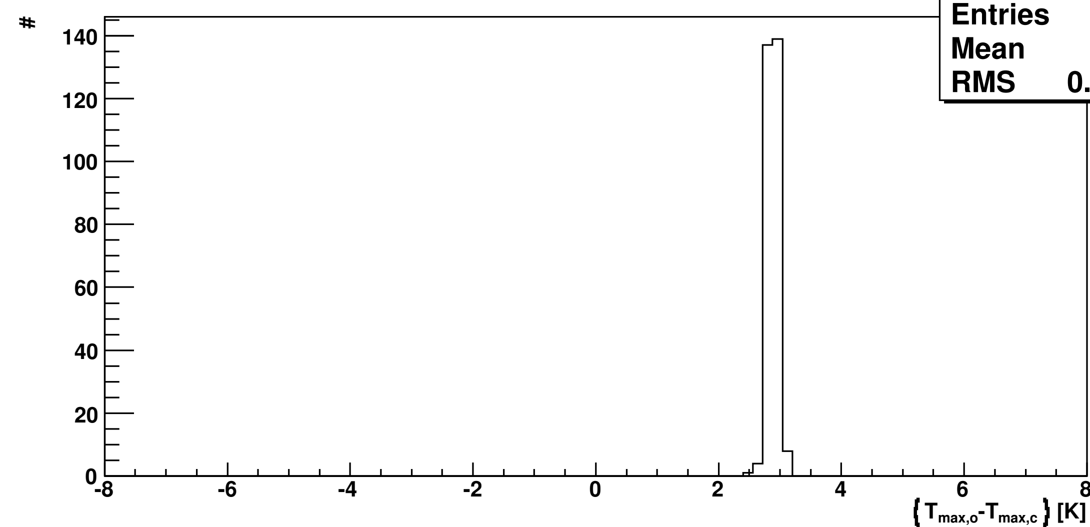

Valve 26: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

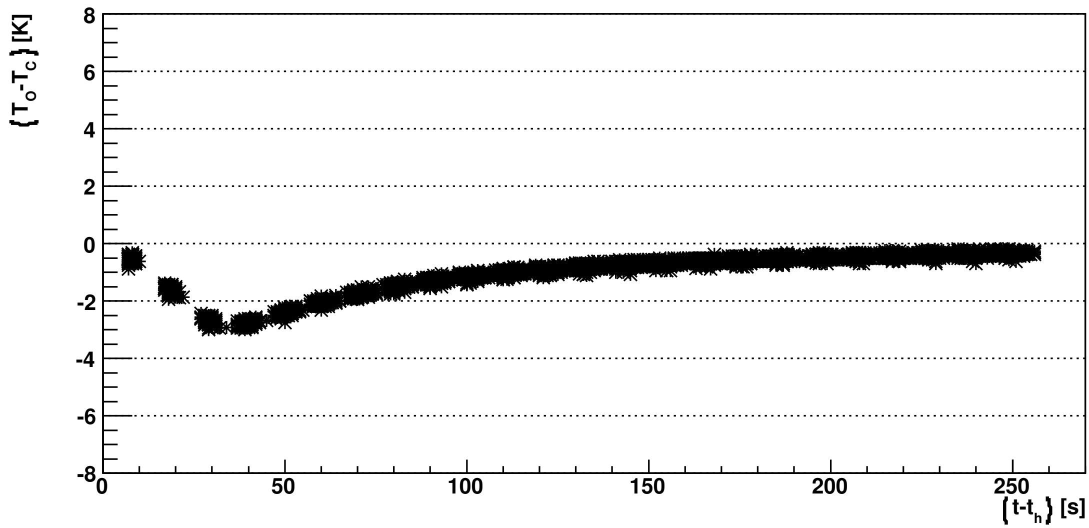

Valve 26: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

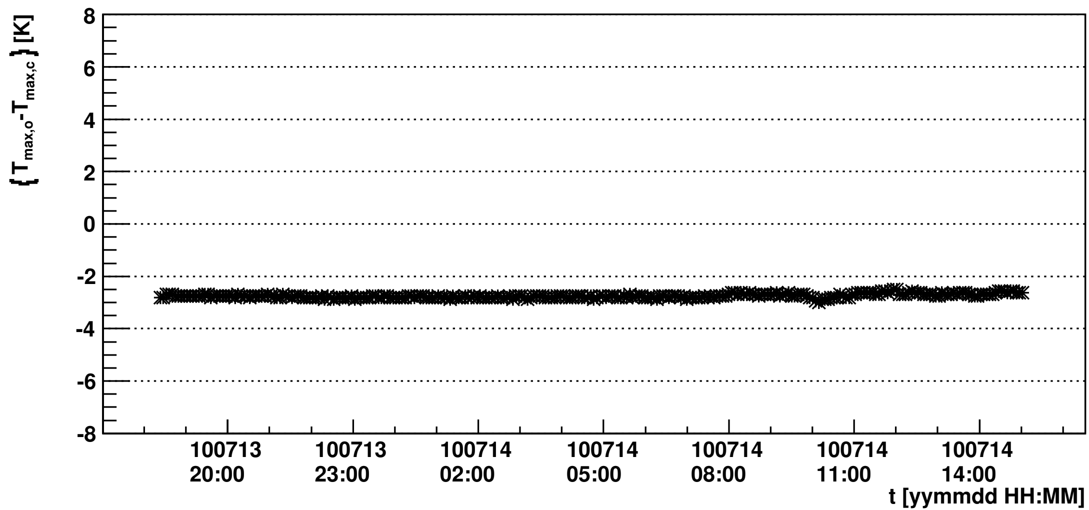

Valve 26: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean 4.686
RMS 0.1142
#
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

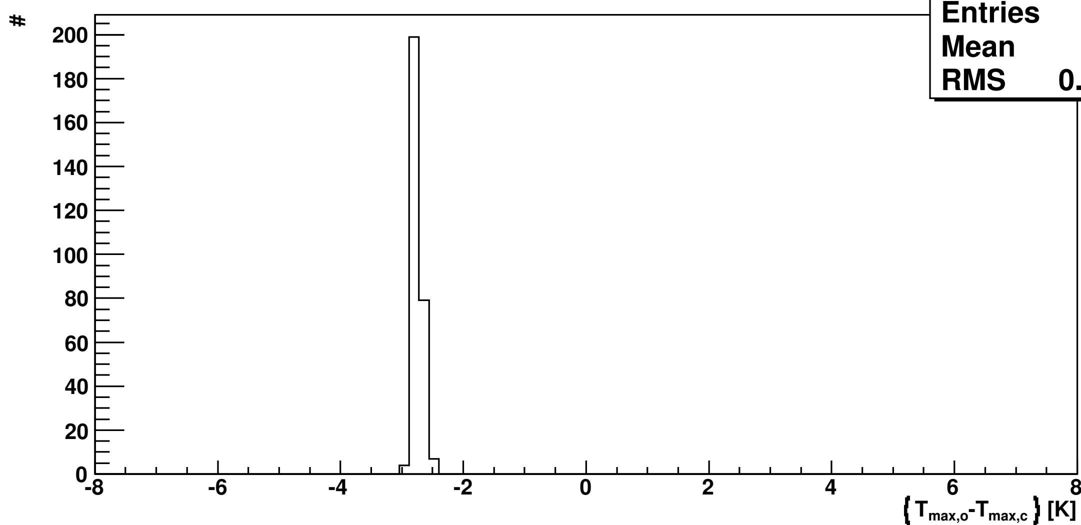

Valve 27: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

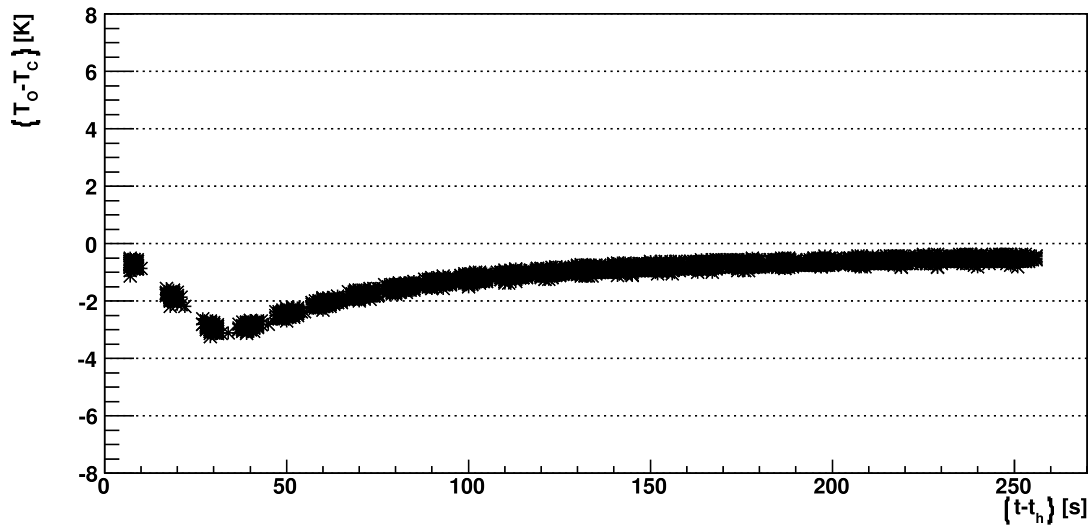

Valve 27: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

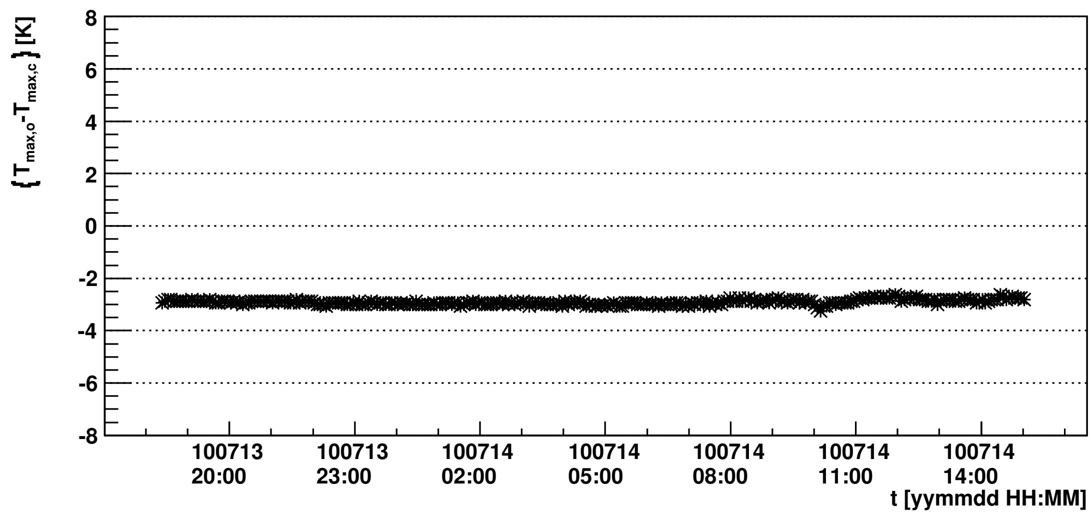

Valve 27: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean 2.882
RMS 0.07816
#
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

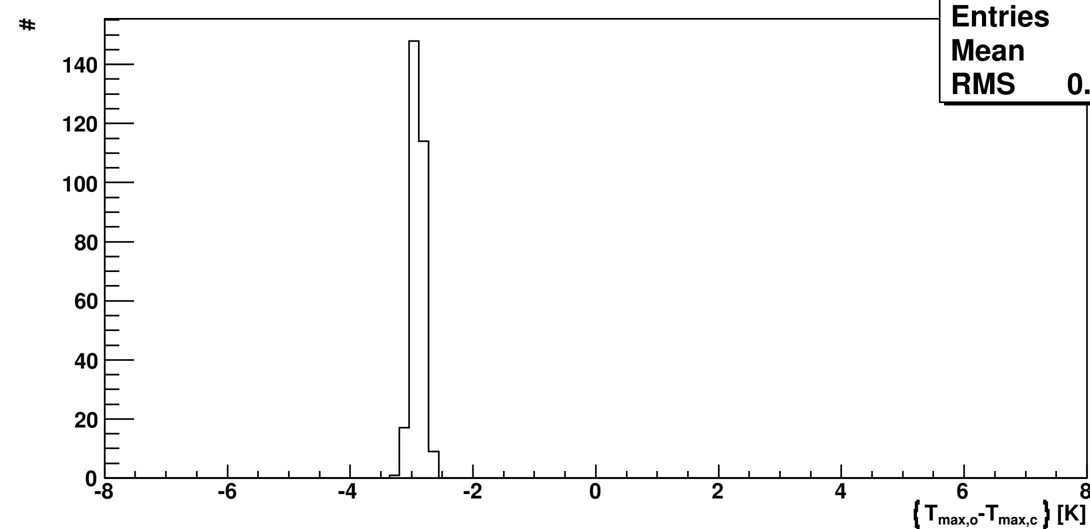

Valve 31: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

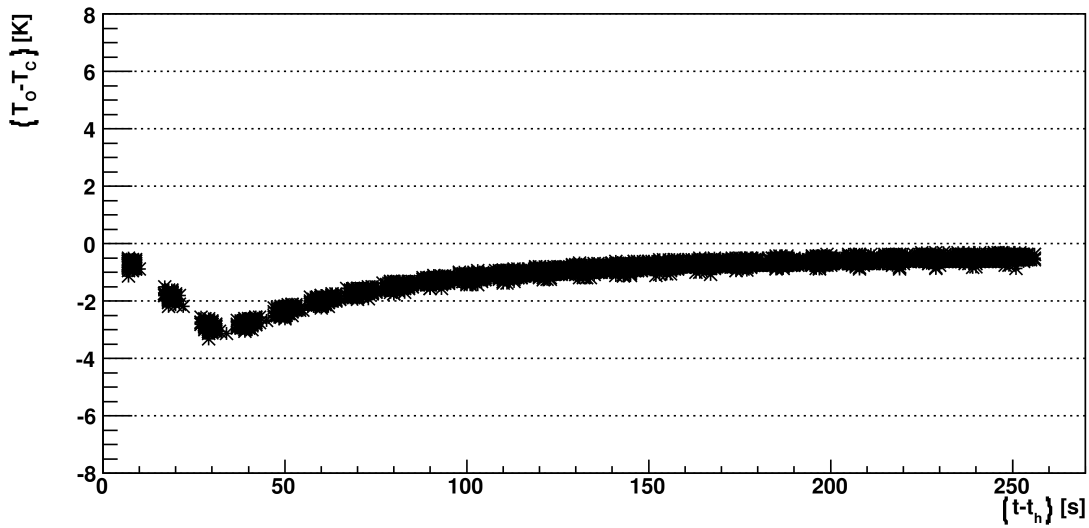

Valve 31: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

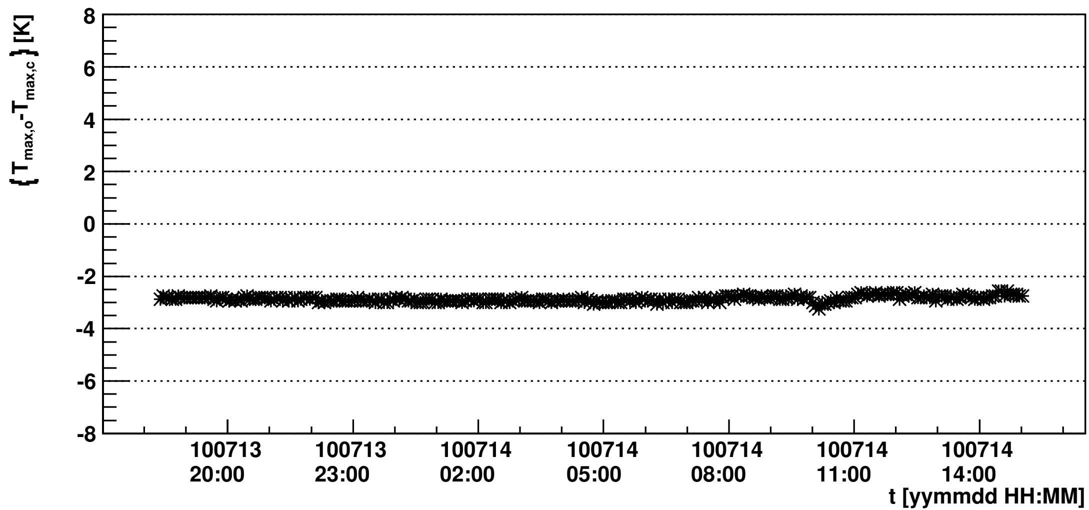

Valve 31: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -2.748
RMS 0.07652
#
200
180
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

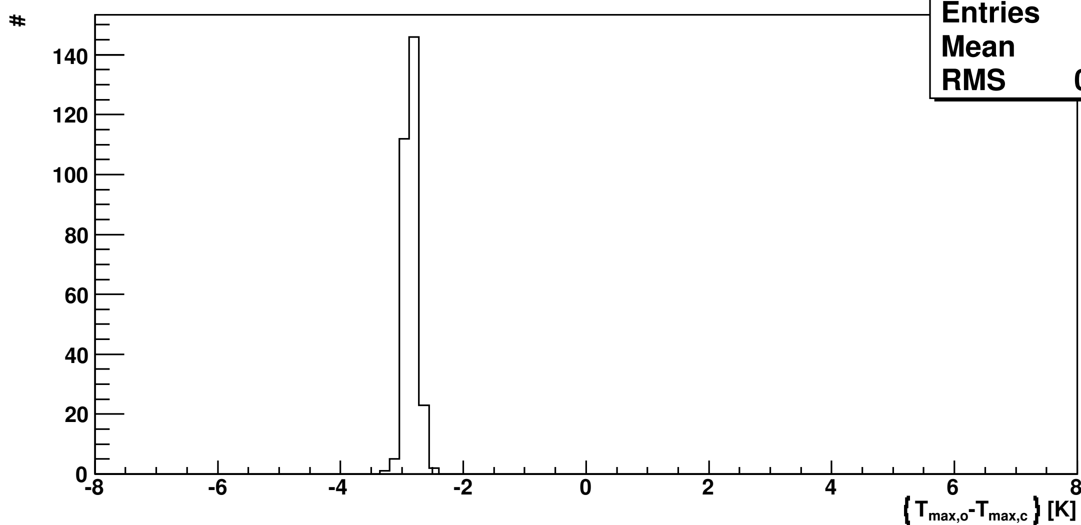

Valve 32: Opentemp - Closetemp
{T_O-T_C} [K]
{t-t_h} [s]

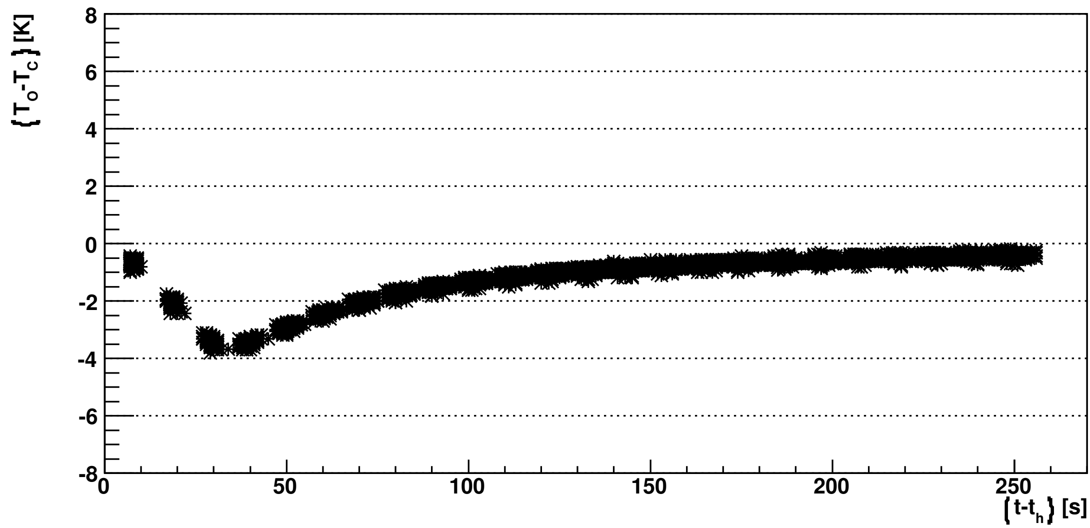

Valve 32: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
t [yymmdd HH:MM]

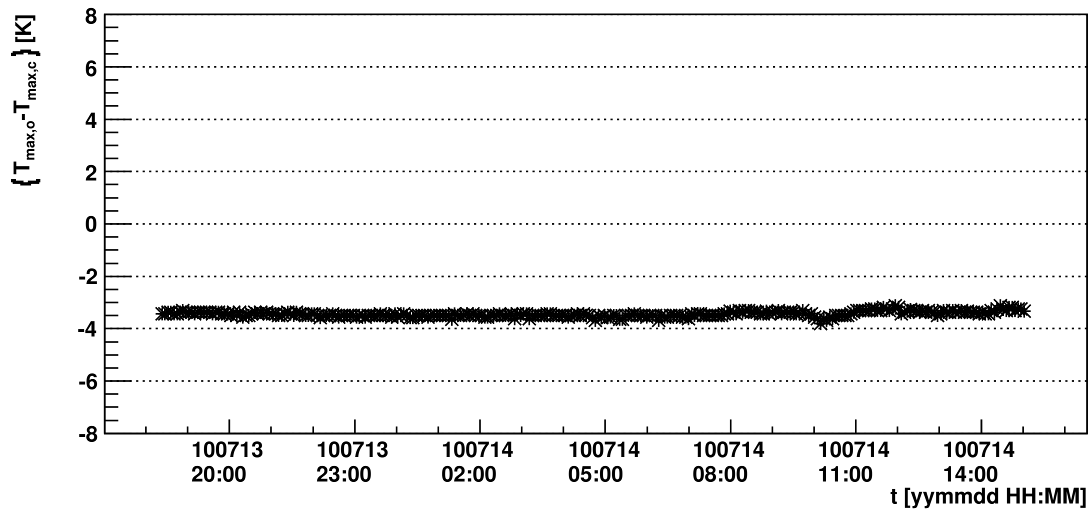

Valve 32: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -2.909
RMS 0.09373
#
{T_max,o-T_max,c} [K]

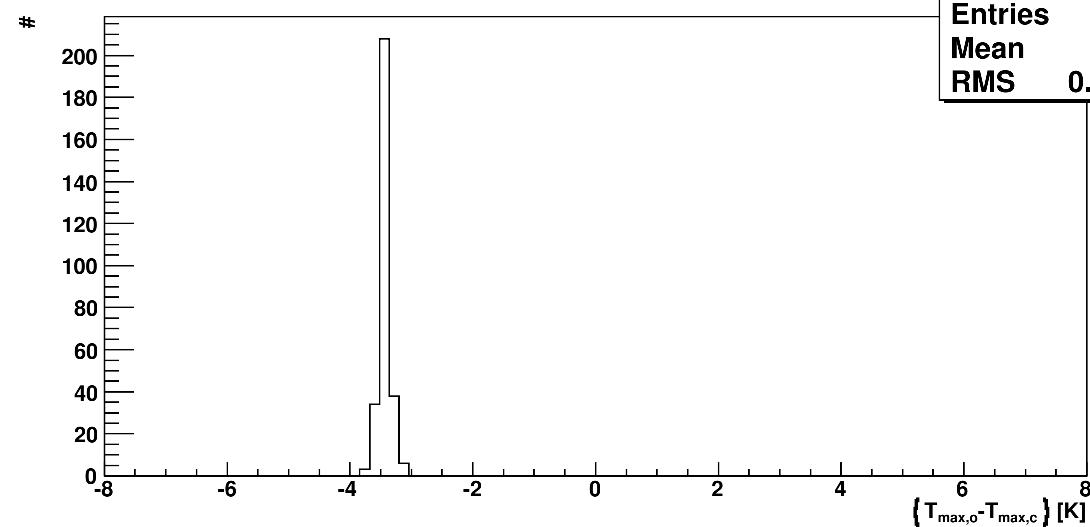

Valve 33: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

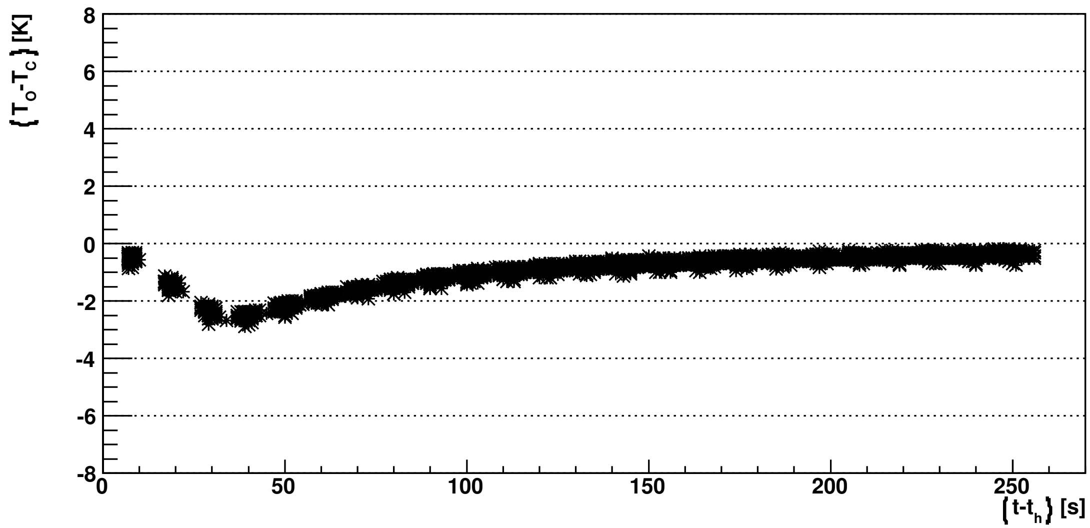

Valve 33: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]


Valve 33: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -2.864
RMS 0.1026
#
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

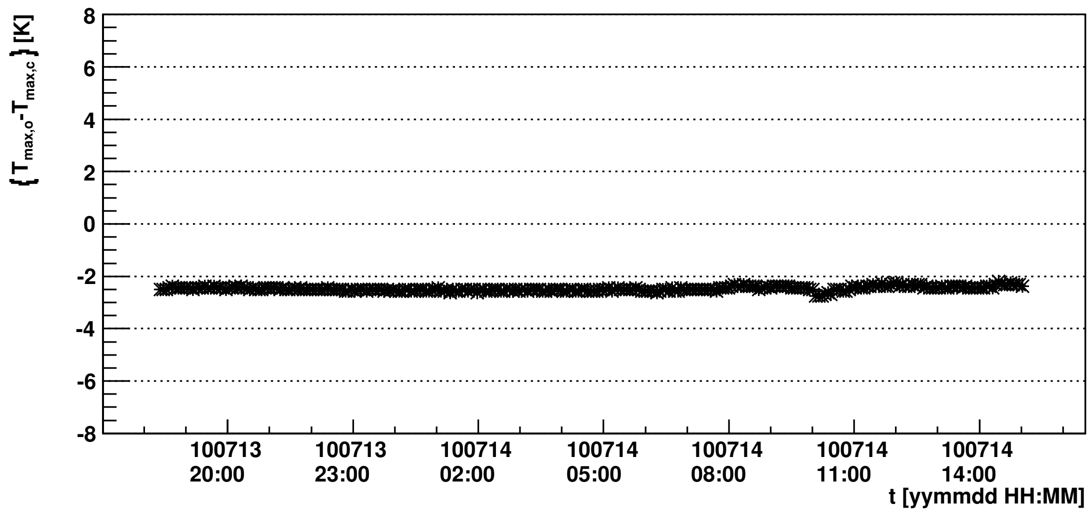

Valve 34: Opentemp - Closetemp
{T_o-T_C} [K]
{t-t_h} [s]


Valve 34: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
t [yymmdd HH:MM]

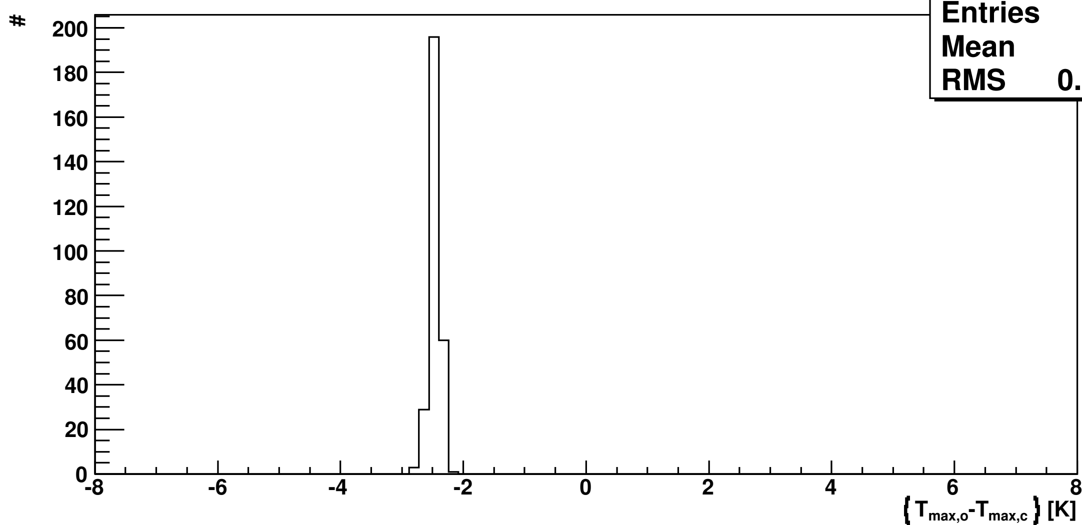

Valve 34: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -3.446
RMS 0.09842
#
{T_max,o-T_max,c} [K]



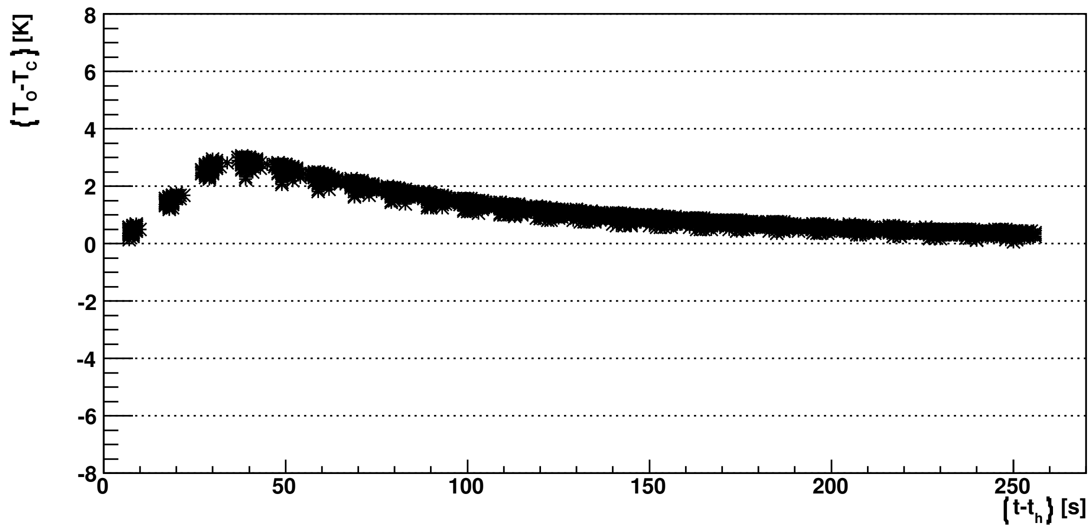
Valve 35: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]



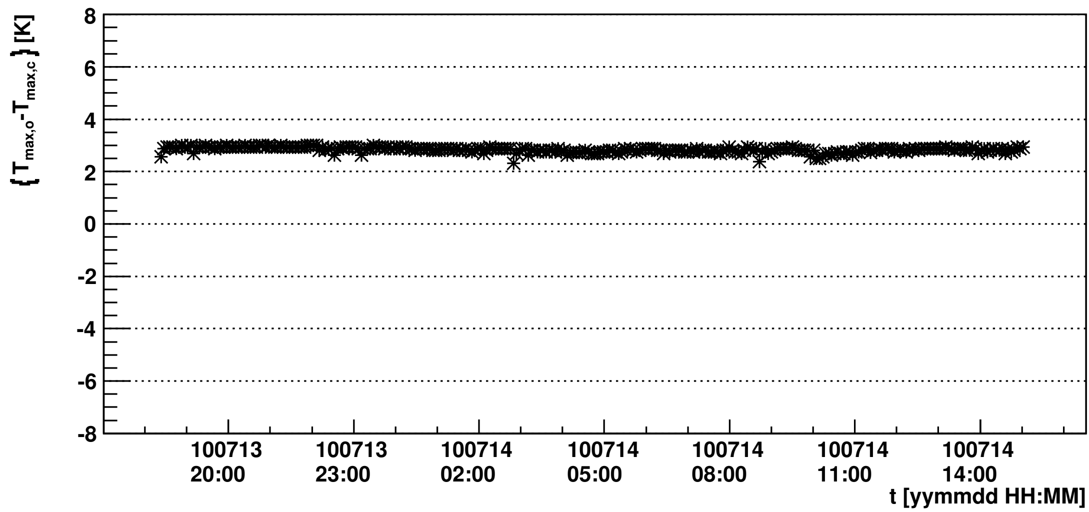
Valve 35: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]



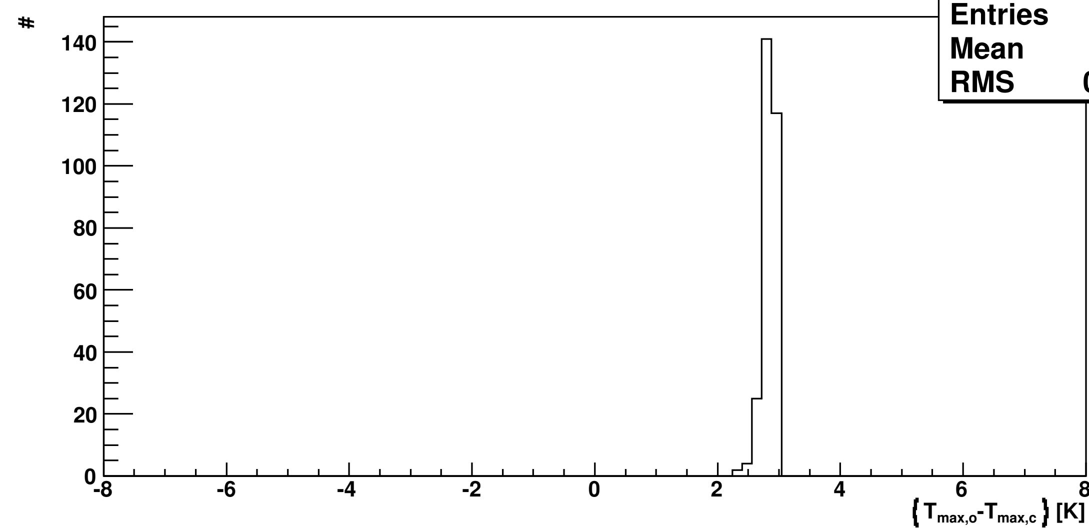
Valve 35: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -2.47
RMS 0.08365
#
200
180
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]



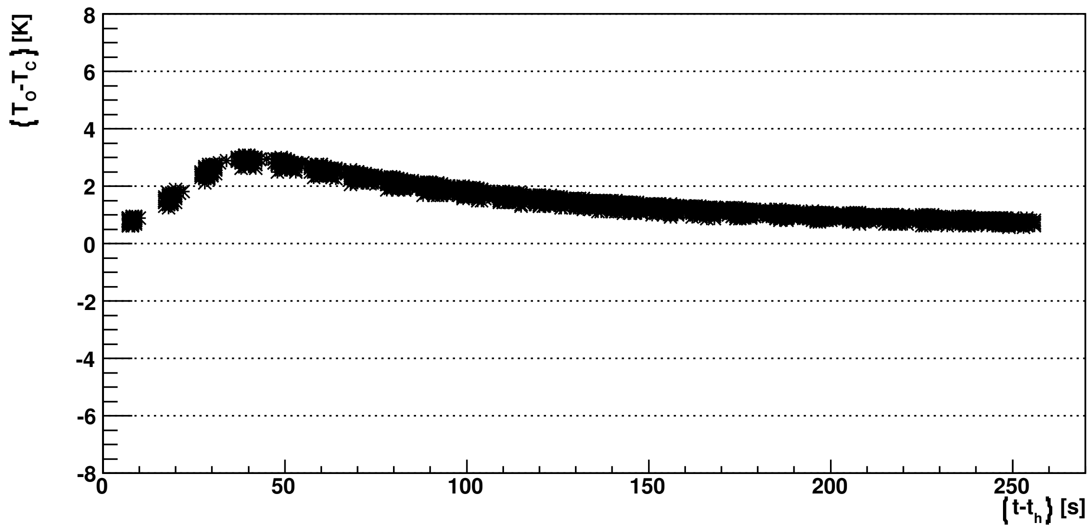
Valve 36: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

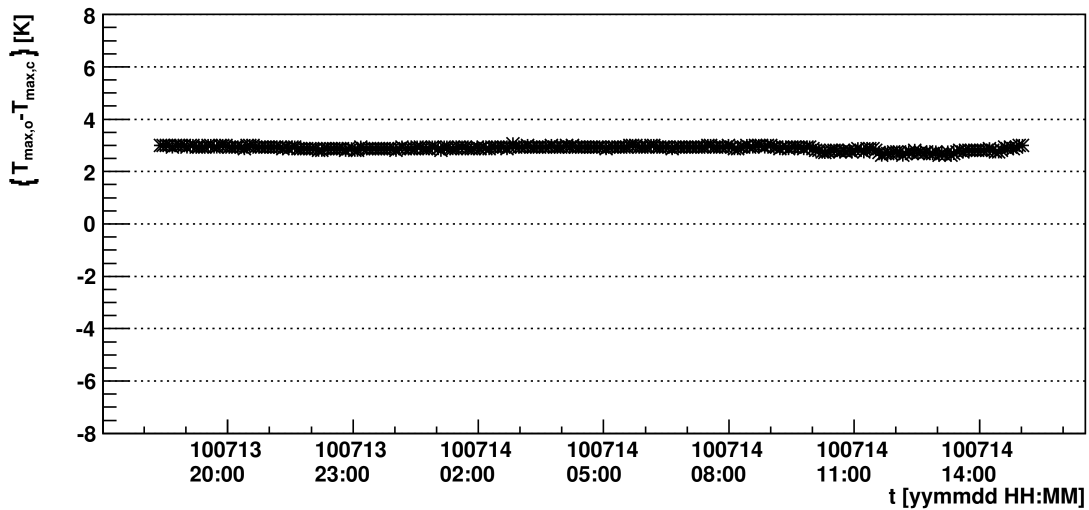


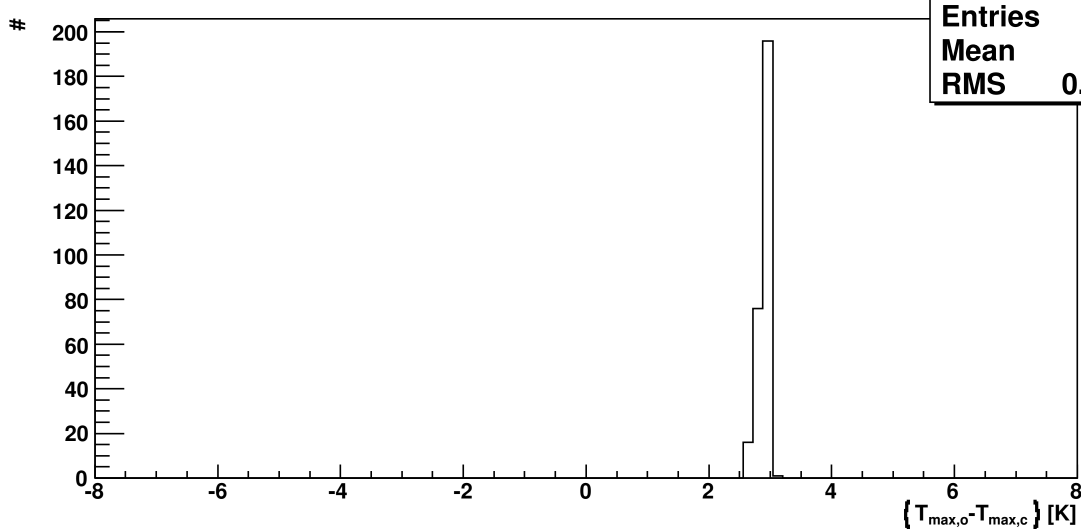
Valve 36: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

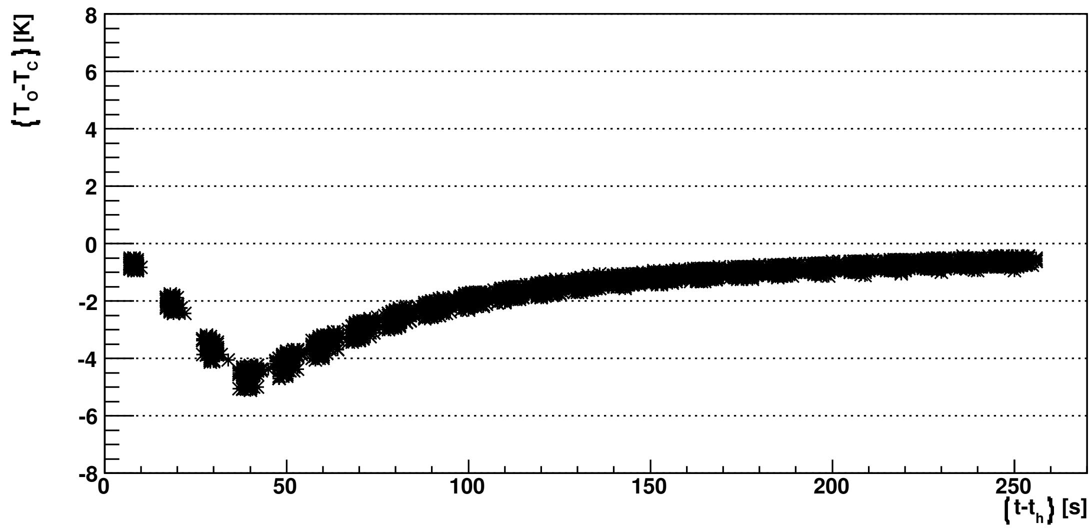


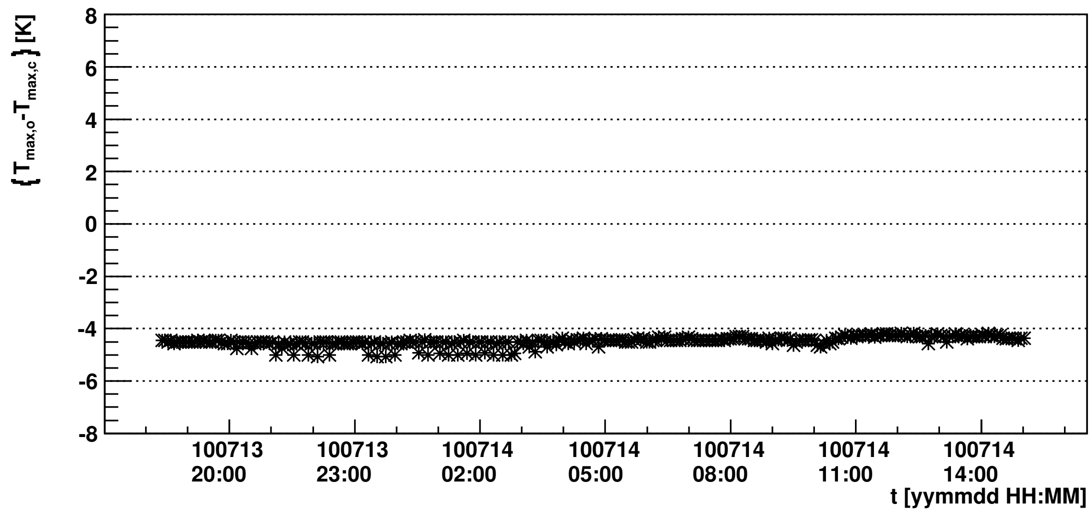
Valve 36: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean 2.836
RMS 0.1024
#
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

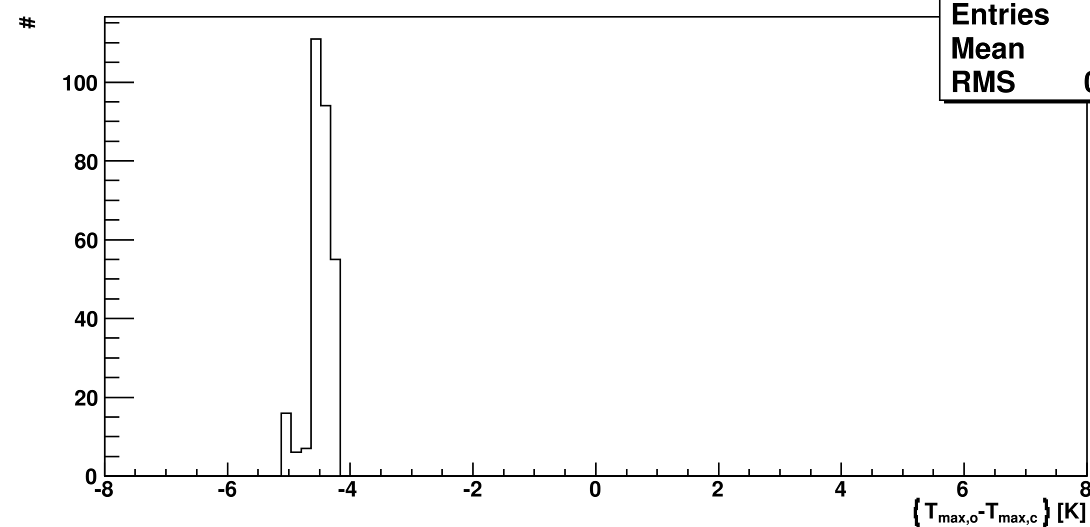

Valve 37: Opentemp - Closetemp
$\{T_O-T_C\}$ [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
$\{t-t_h\}$ [s]

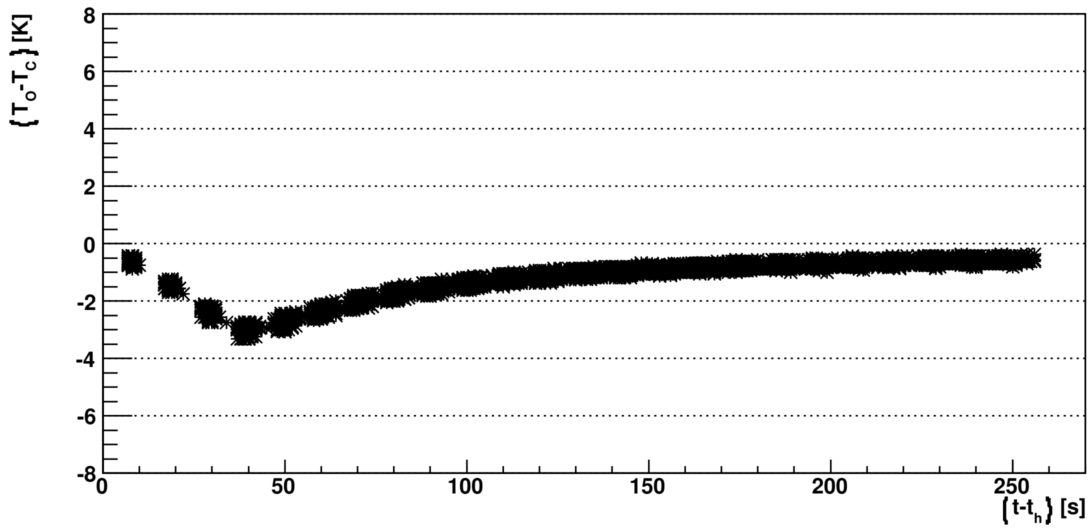

Valve 37: OpenMaxtemp - CloseMaxtemp
$\{T_{max,o}-T_{max,c}\}$ [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

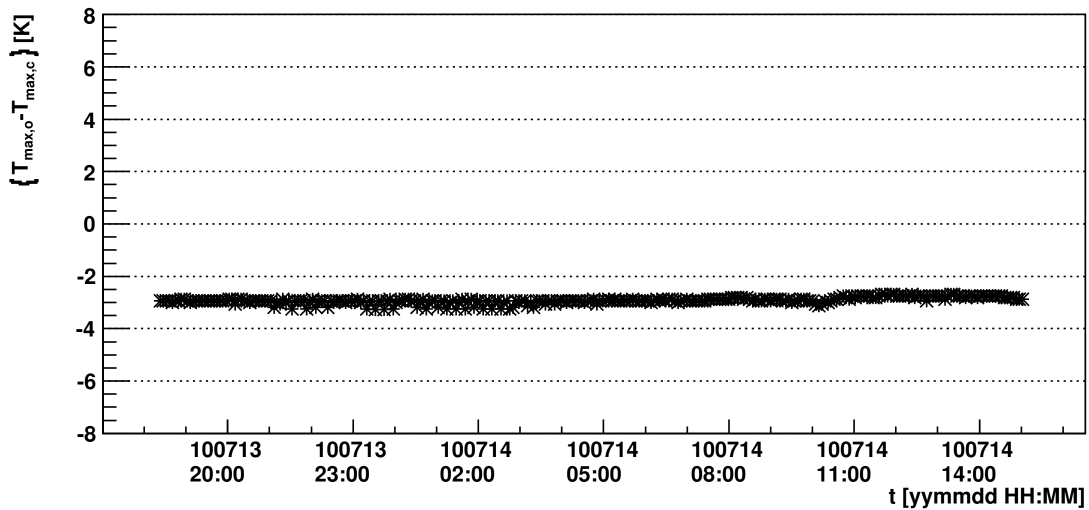

Valve 37: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean 2.892
RMS 0.08591
#
200
180
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
$\{T_{max,o}-T_{max,c}\}$ [K]

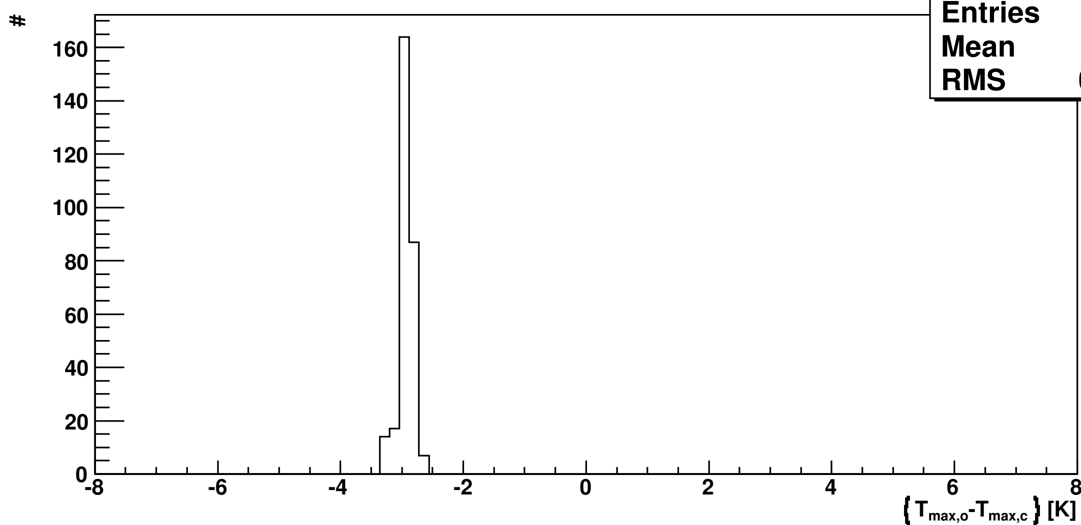

Valve 41: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

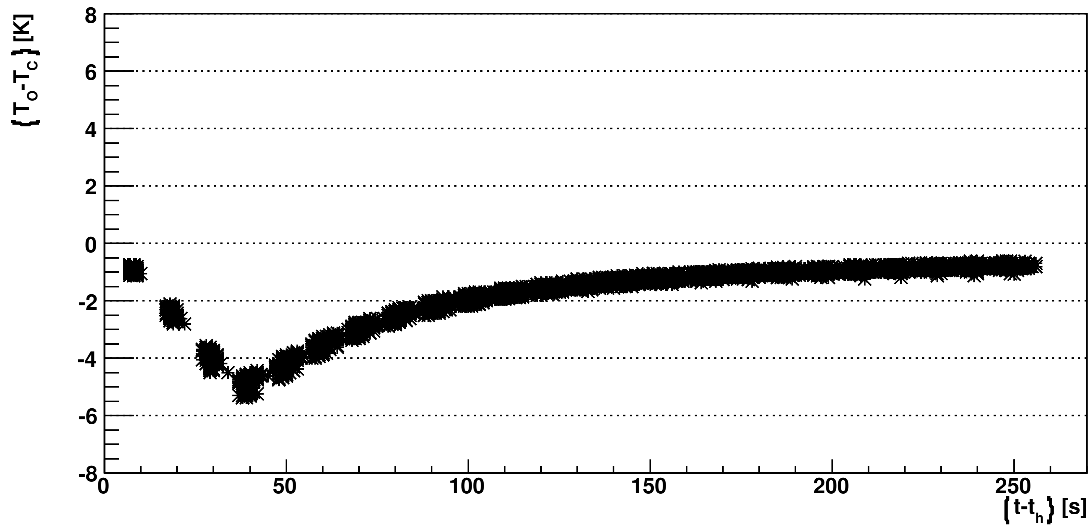

Valve 41: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

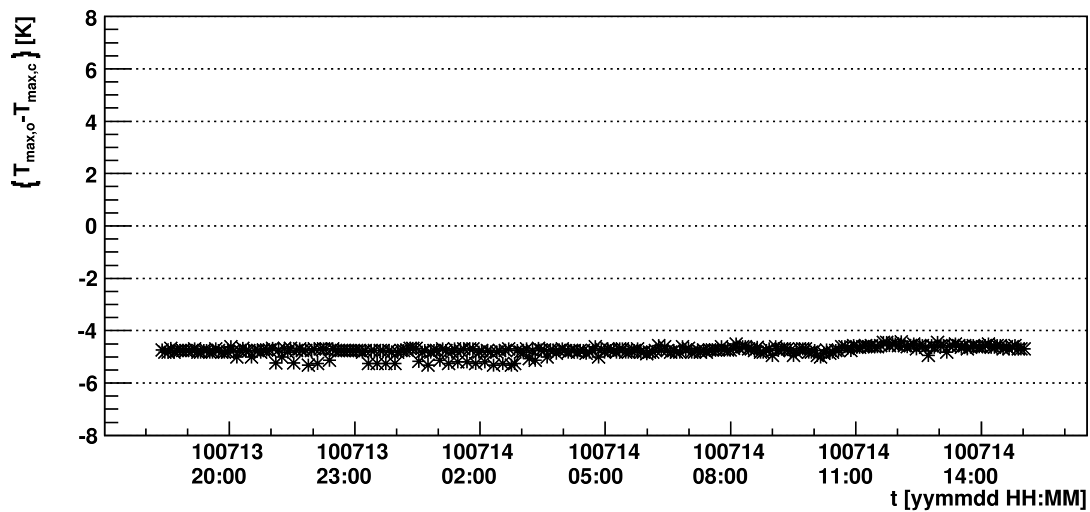

Valve 41: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -4.48
RMS 0.1813
#
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

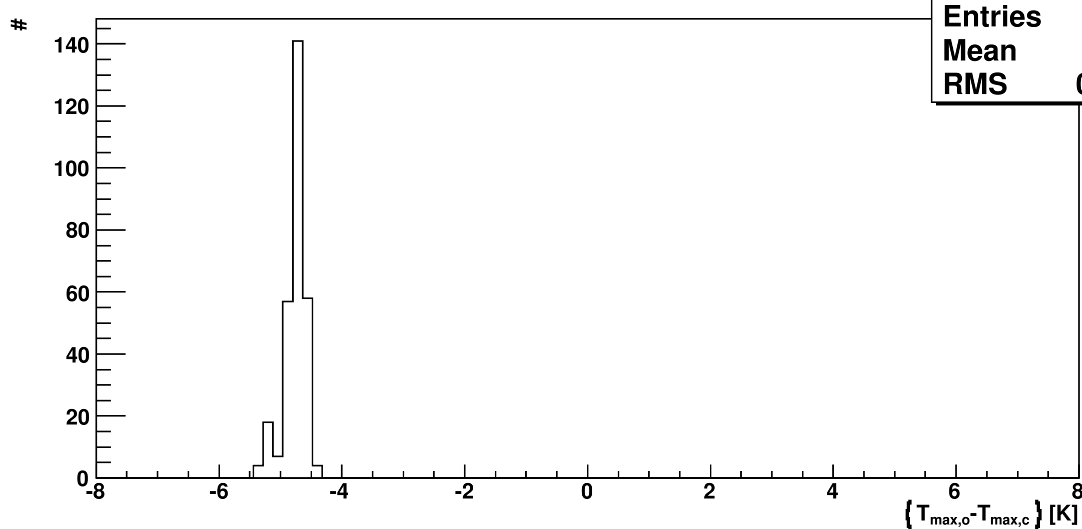

Valve 42: Opentemp - Closetemp
(T_O-T_C) [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
$(t-t_h)$ [s]

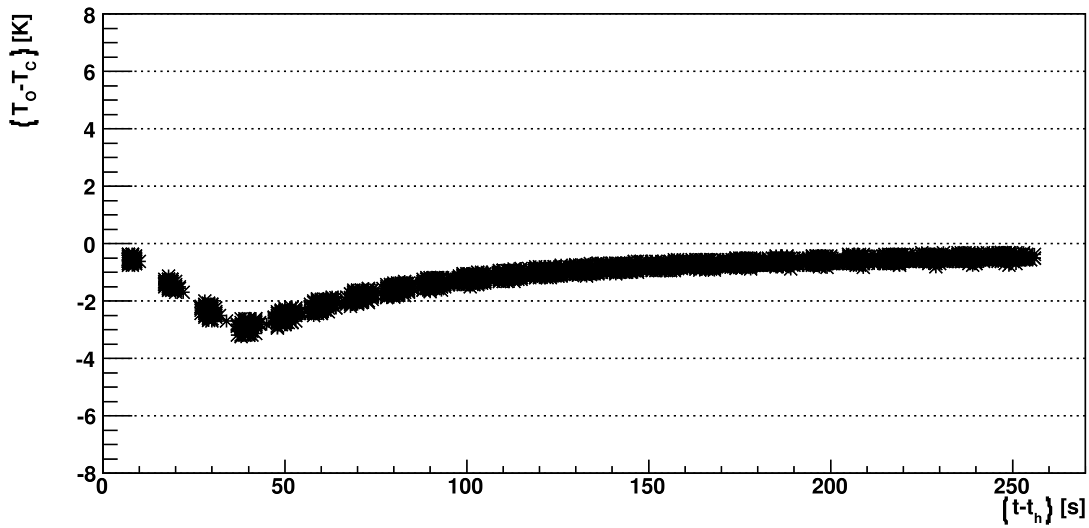

Valve 42: OpenMaxtemp - CloseMaxtemp
$(T_{max,o}-T_{max,c})$ [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

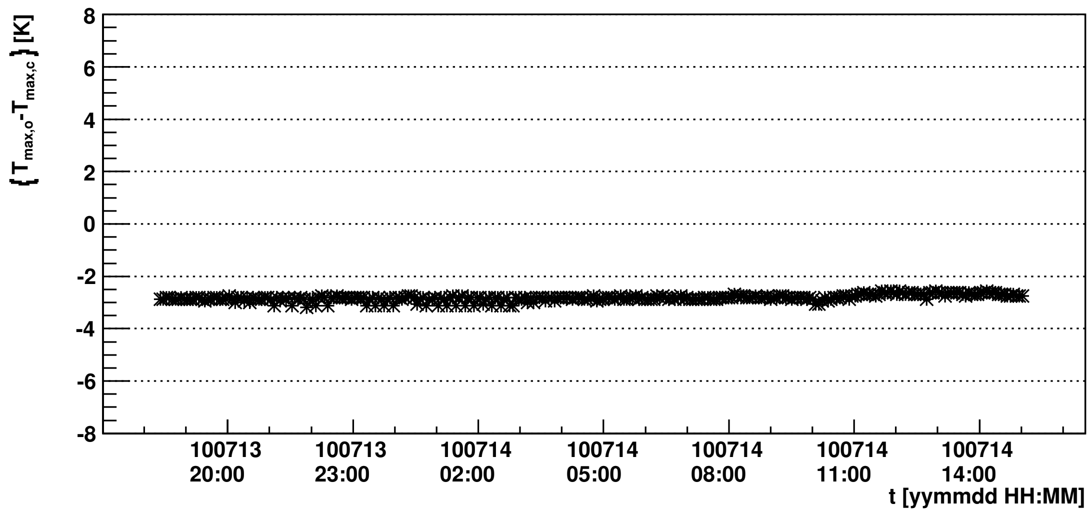

Valve 42: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -2.921
RMS 0.1211
#
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
$(T_{max,o}-T_{max,c})$ [K]

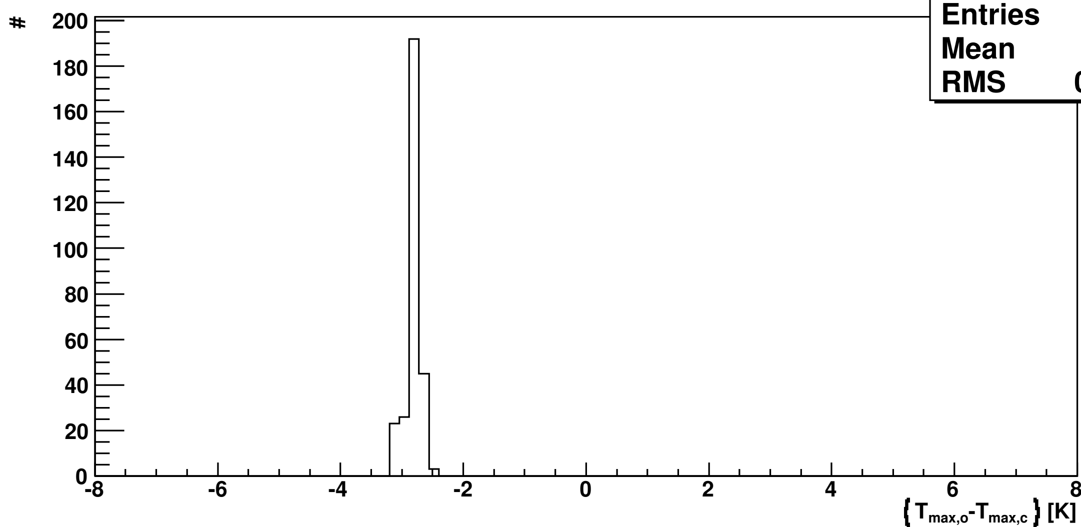

Valve 43: Opentemp - Closetemp
(T_O-T_C) [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
(t-t_h) [s]

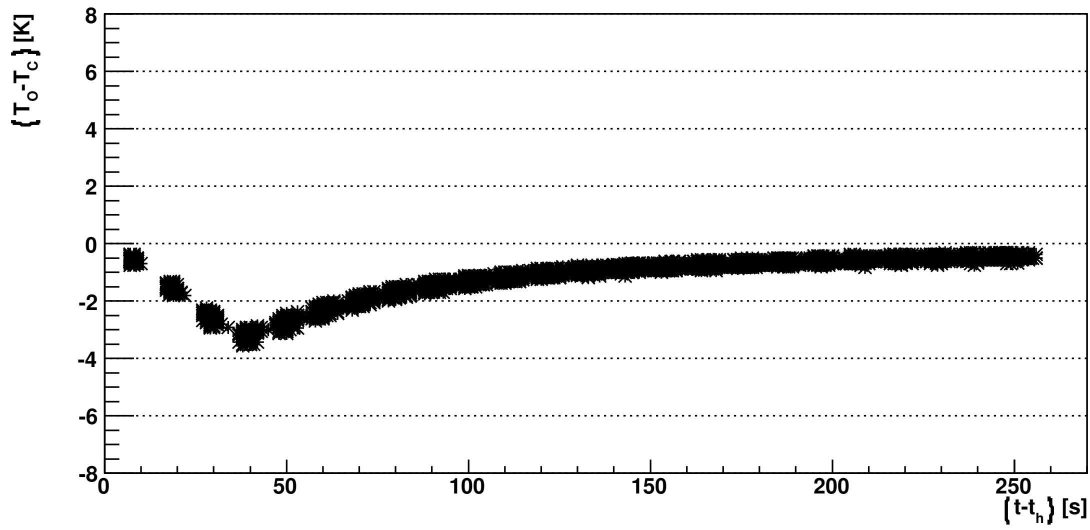

Valve 43: OpenMaxtemp - CloseMaxtemp
(T_max,o-T_max,c) [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

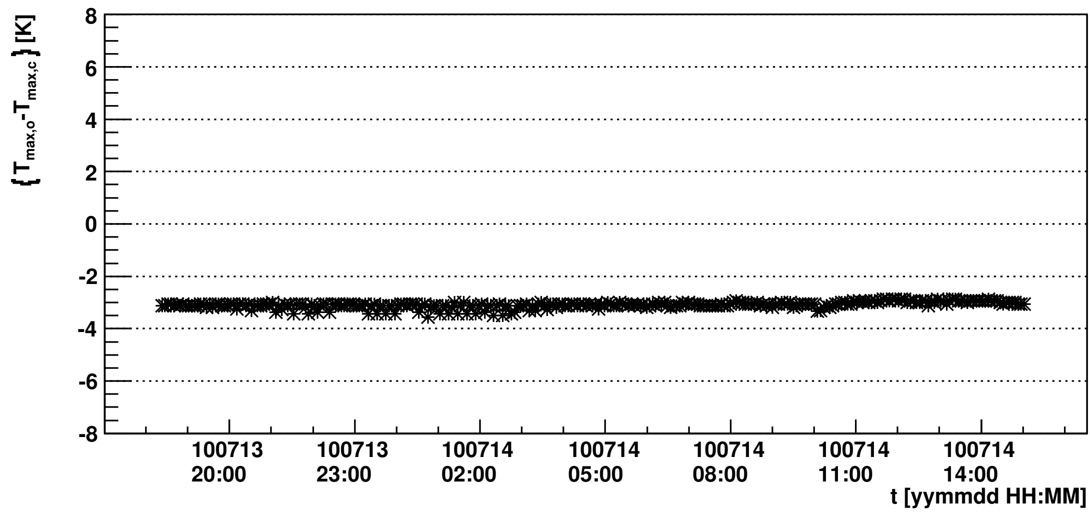

Valve 43: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -4.76
RMS 0.1694
#
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
(T_max,o-T_max,c) [K]

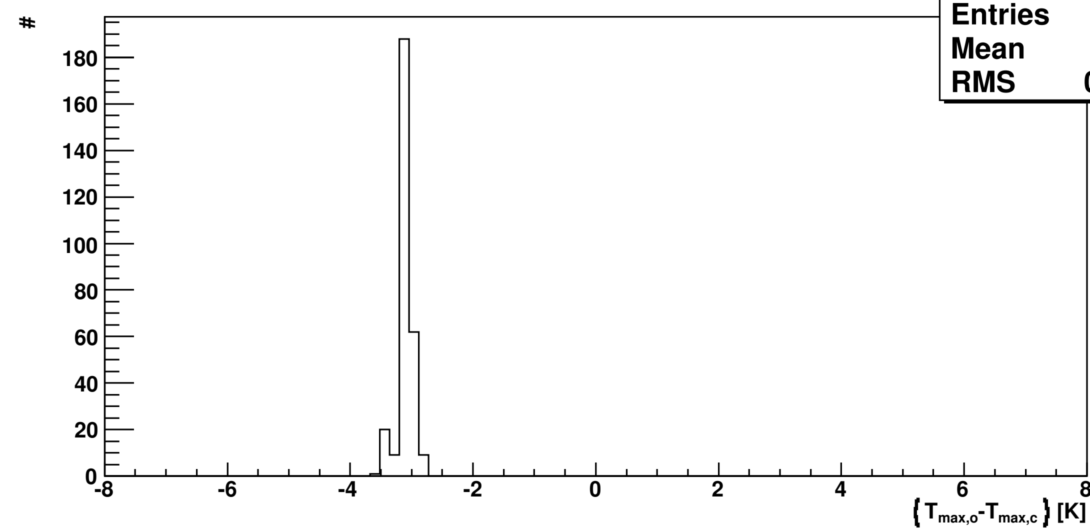

Valve 44: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

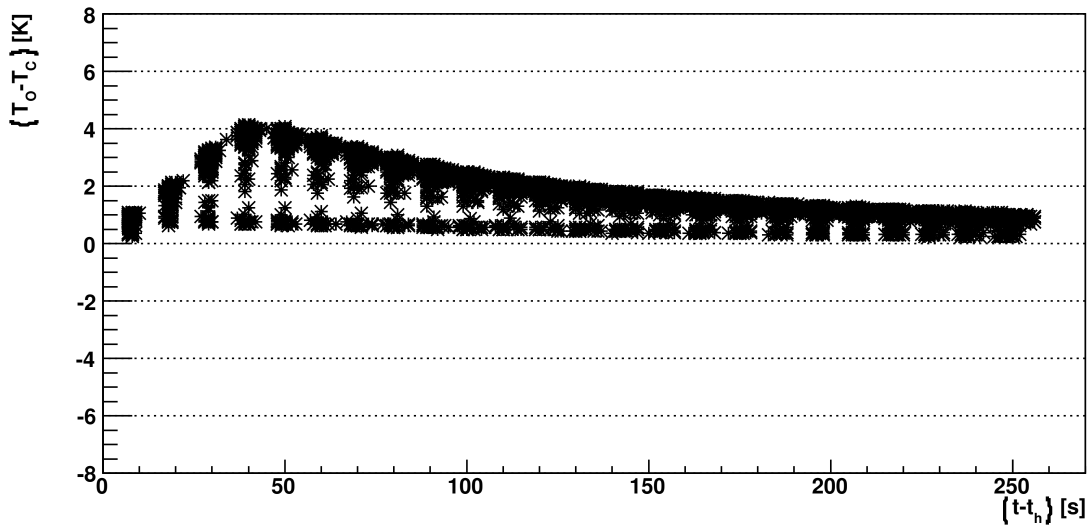

Valve 44: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

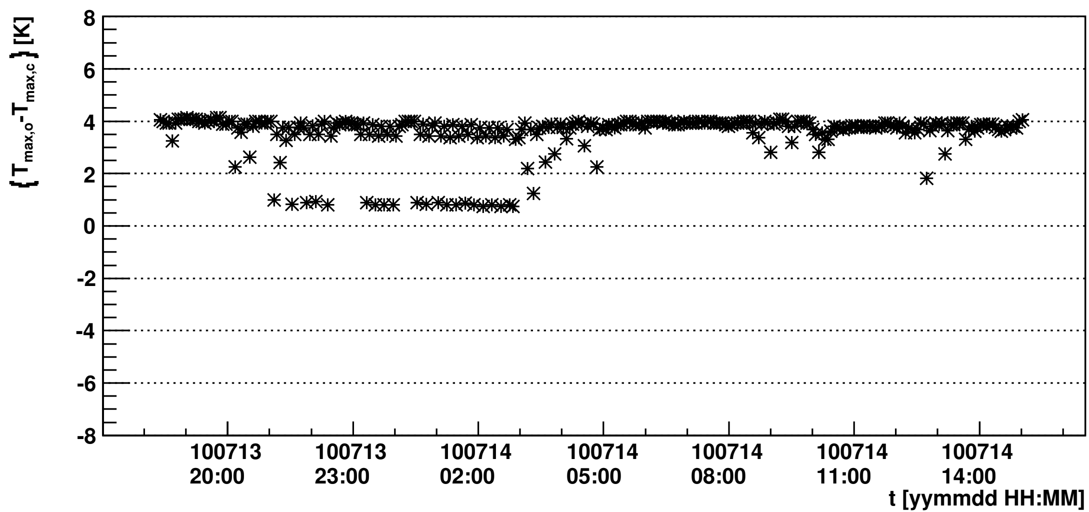

Valve 44: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -2.823
RMS 0.1206
#
200
180
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

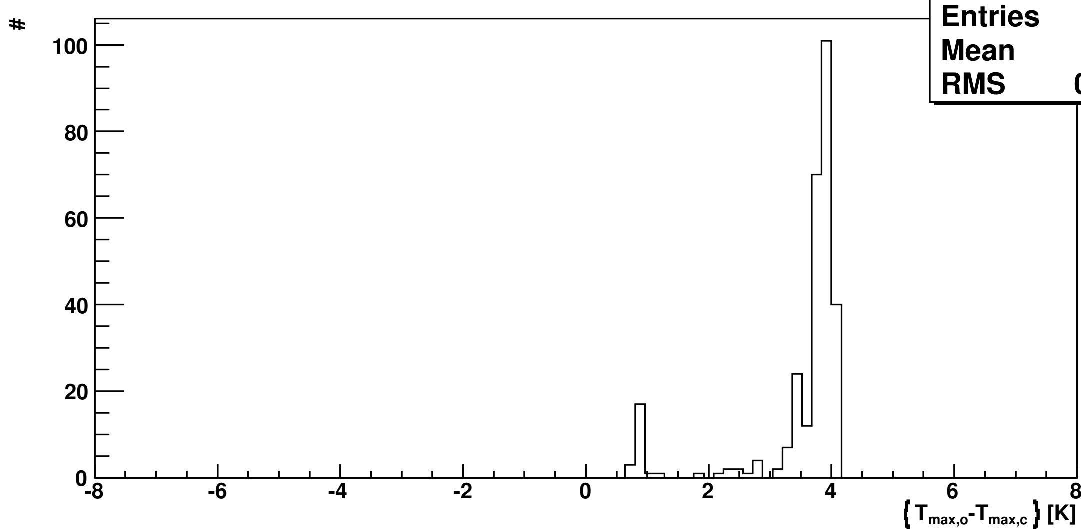

Valve 45: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

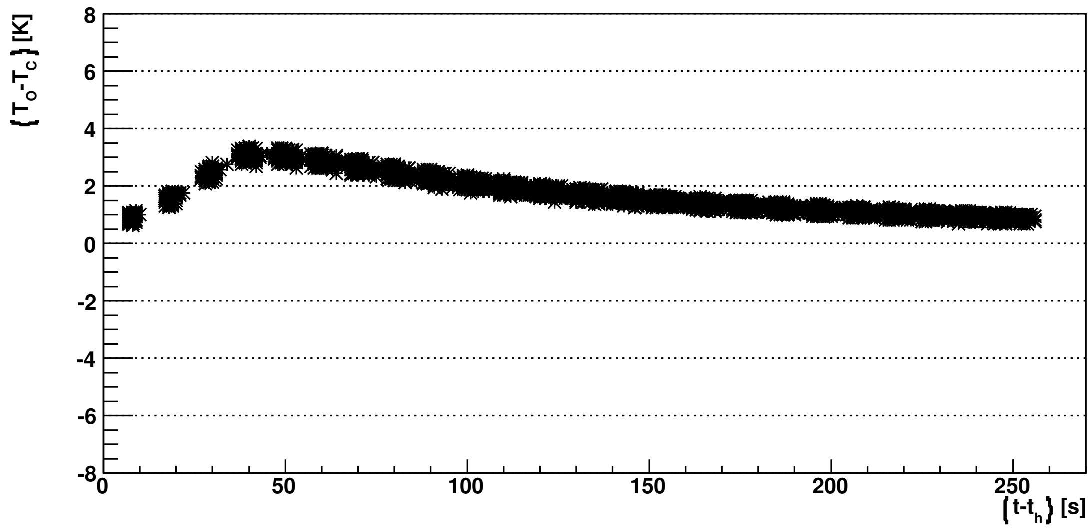

Valve 45: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

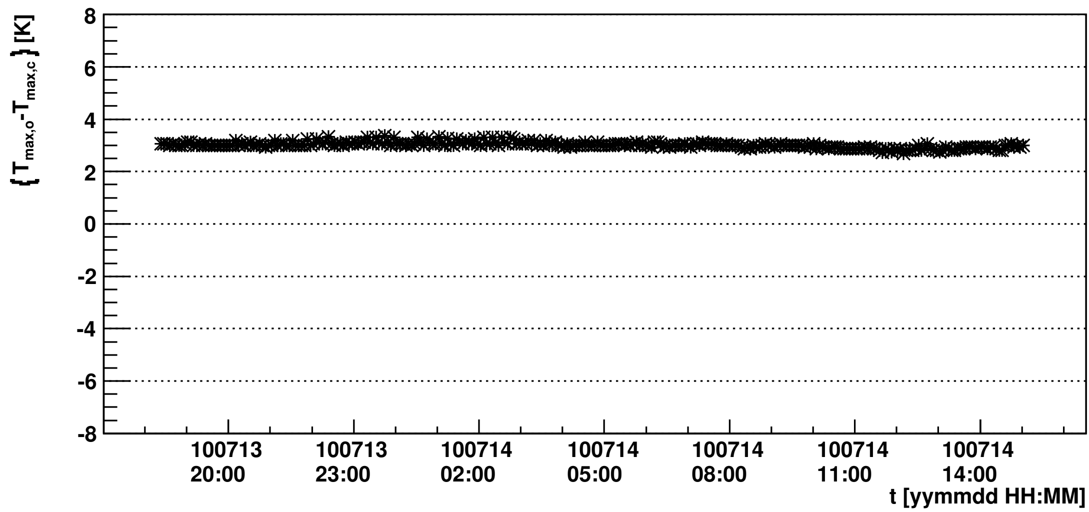

Valve 45: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -3.094
RMS 0.1255
#
180
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

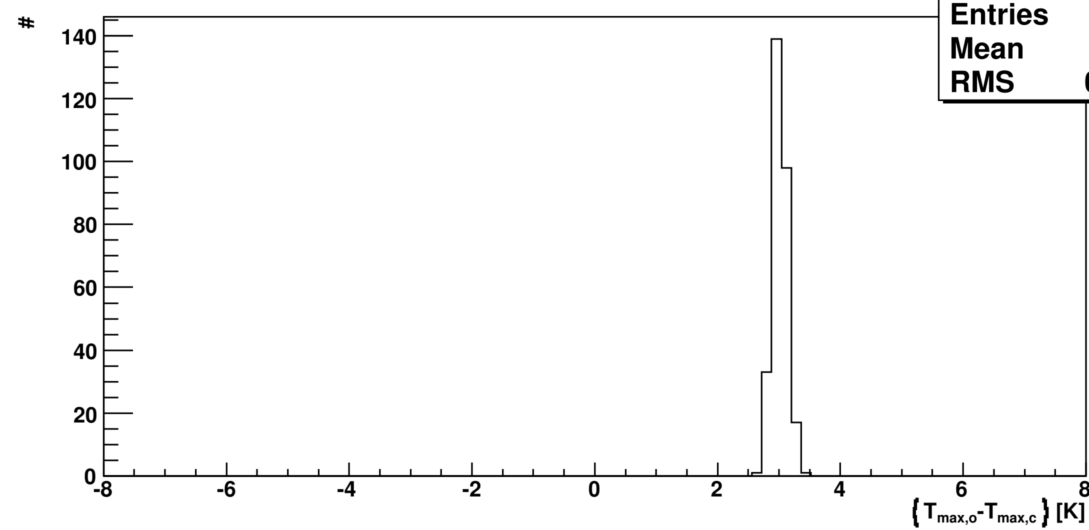

Valve 46: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

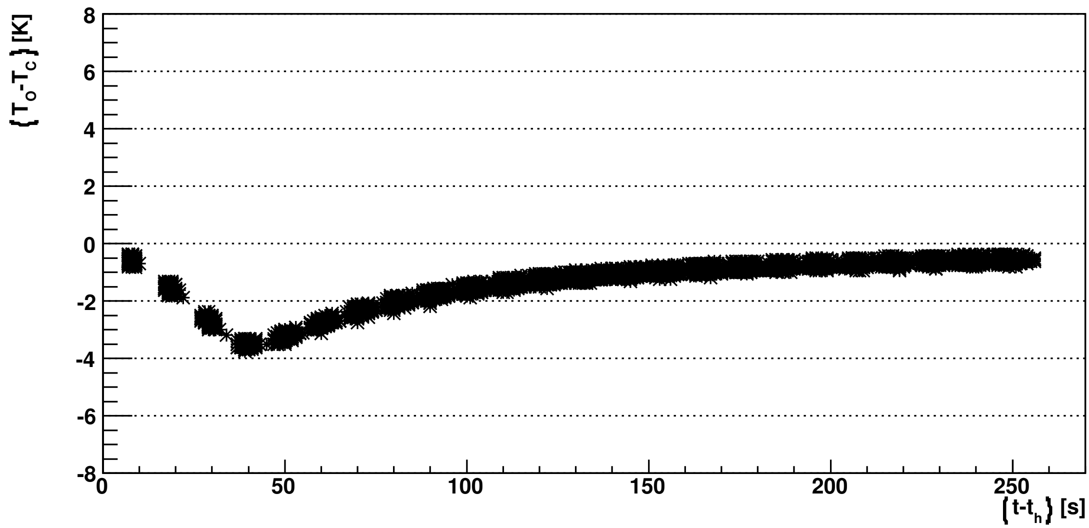

Valve 46: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

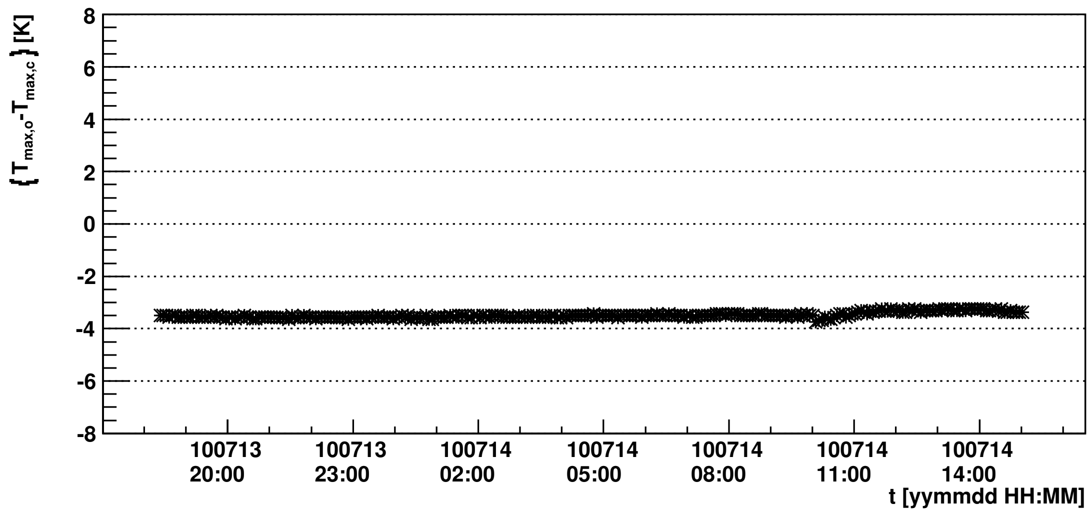

Valve 46: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean 3.538
RMS 0.8366
#
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

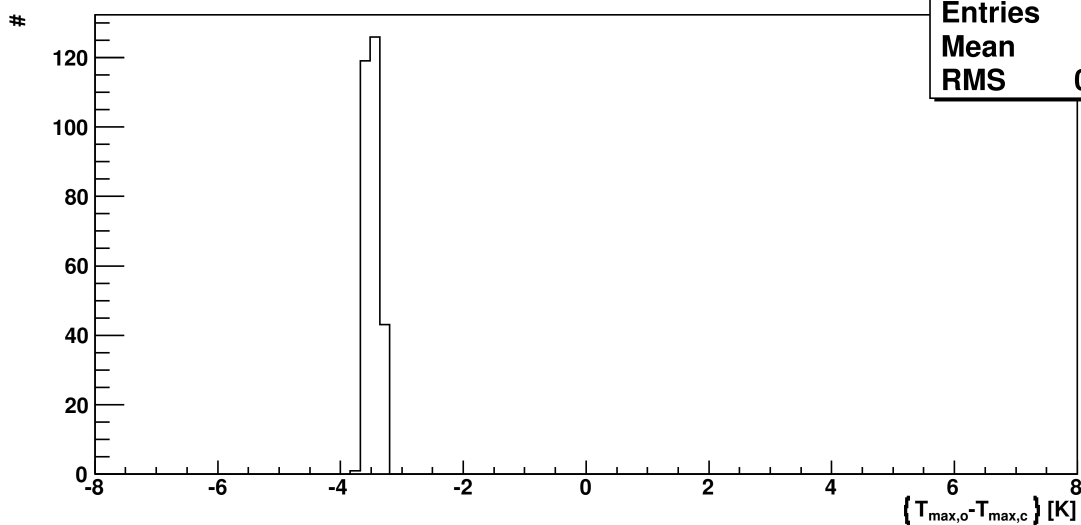

Valve 47: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

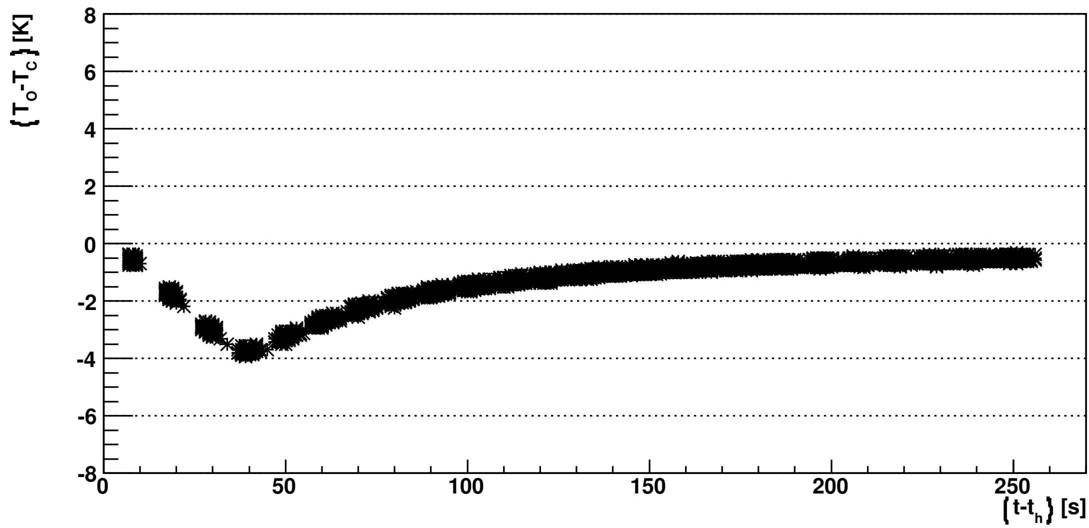

Valve 47: OpenMaxtemp - CloseMaxtemp
{$T_{max,o}$-$T_{max,c}$} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

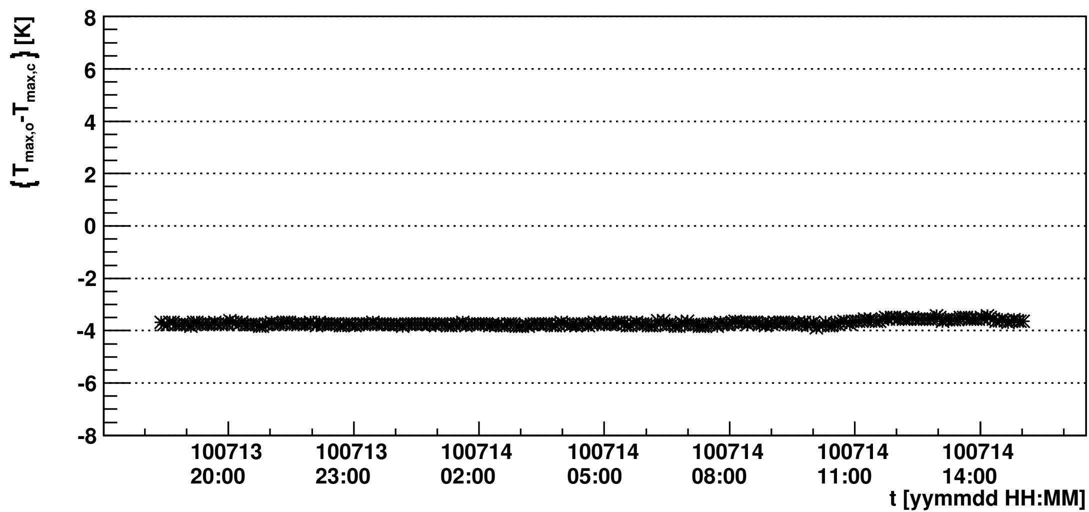

Valve 47: OpenMaxtemp - CloseMaxtemp
#
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{$T_{max,o}$-$T_{max,c}$} [K]
hMaxDiff
Entries 289
Mean 3.016
RMS 0.1134

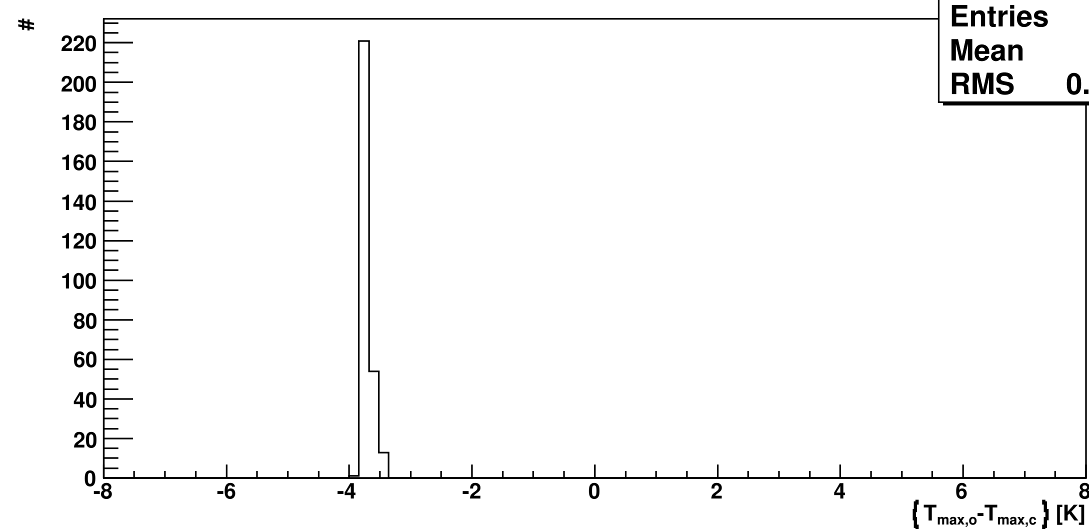

Valve 51: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

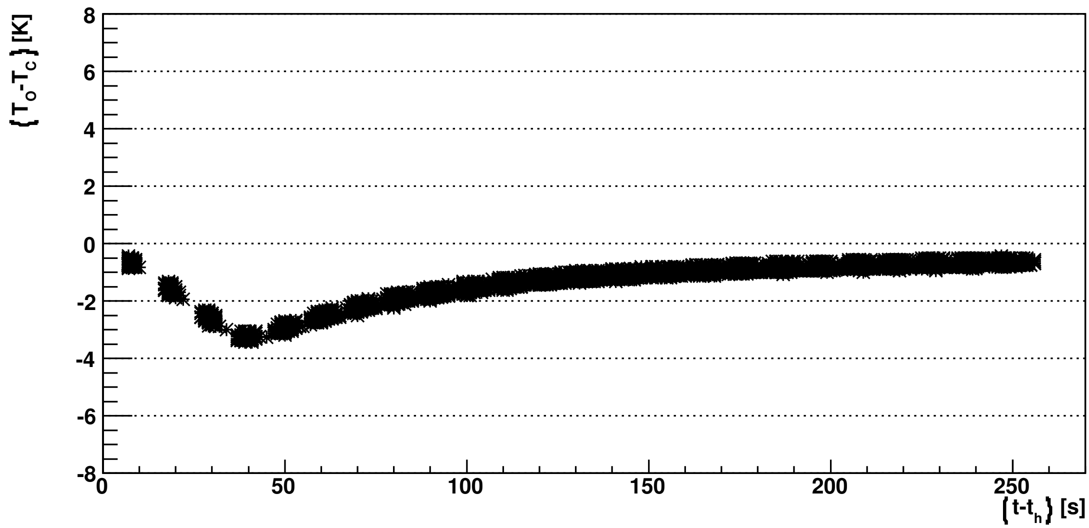

Valve 51: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

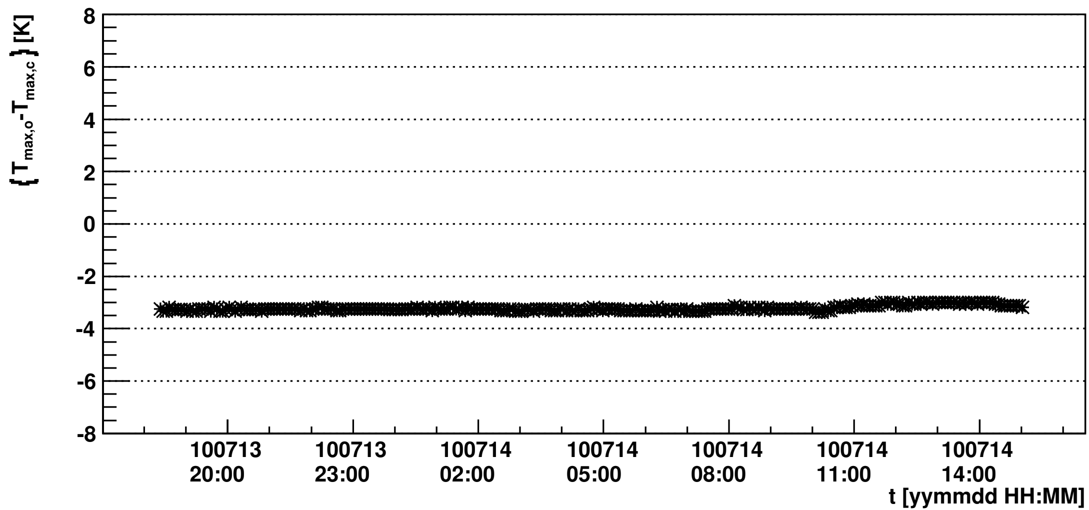

Valve 51: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -3.489
RMS 0.1017
#
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

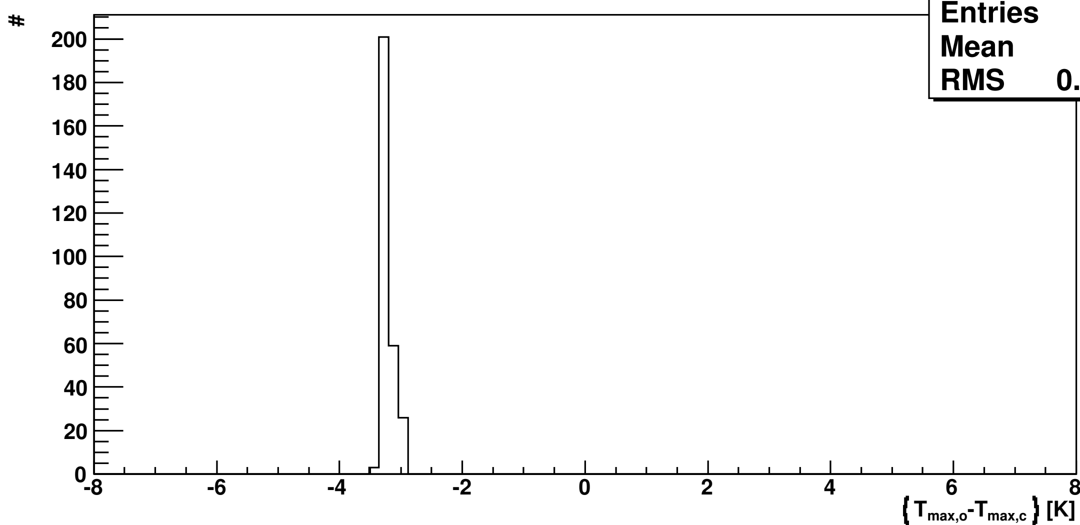

Valve 52: Opentemp - Closetemp
$\{T_o - T_C\}$ [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
$\{t - t_h\}$ [s]

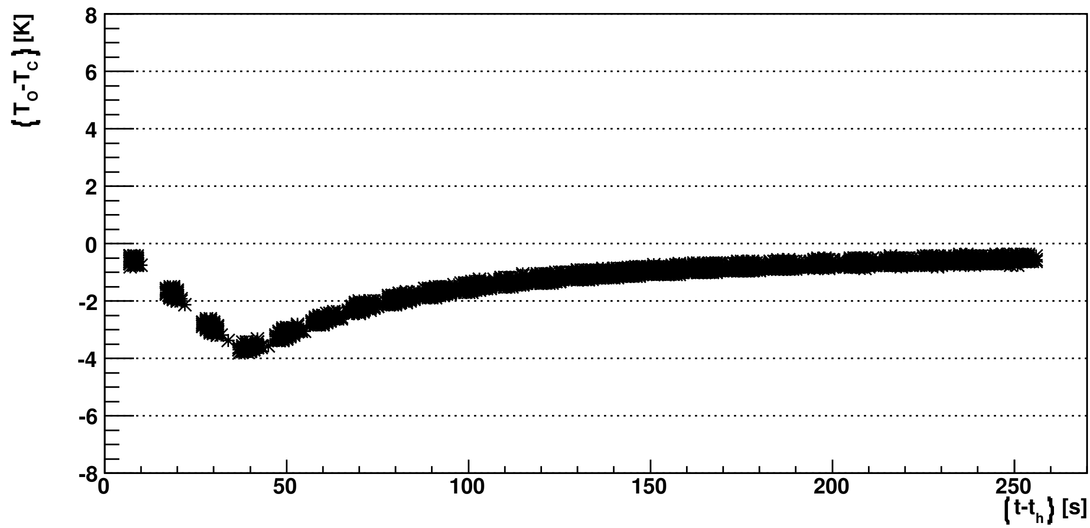

Valve 52: OpenMaxtemp - CloseMaxtemp
$\{T_{max,o} - T_{max,c}\}$ [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

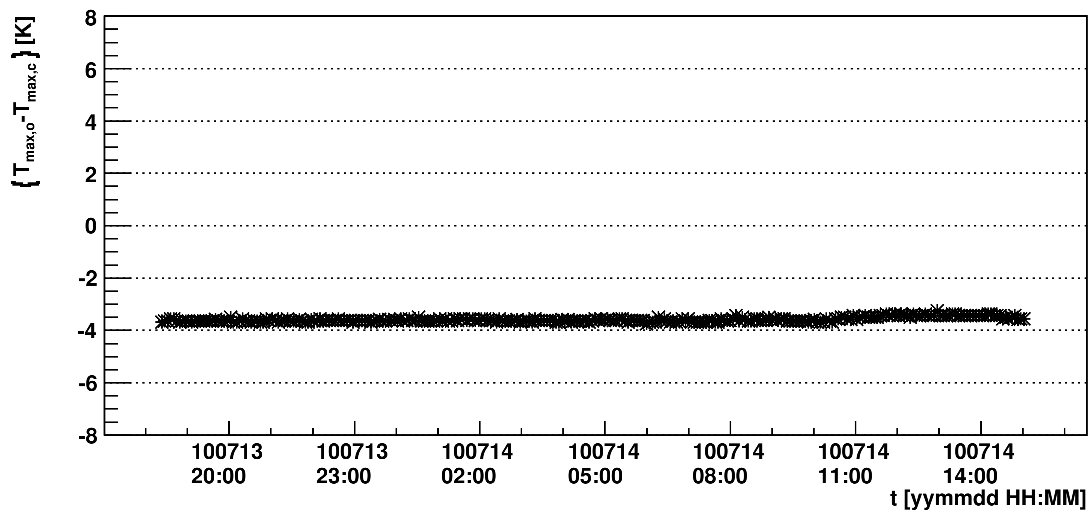

Valve 52: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -3.711
RMS 0.08603
#
220
200
180
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
$\{T_{max,o} - T_{max,c}\}$ [K]

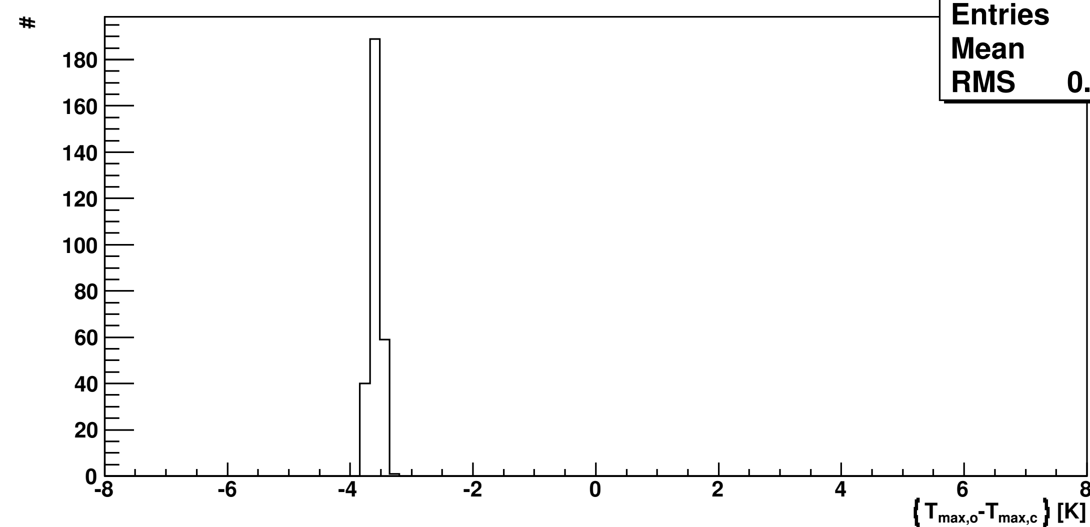

Valve 53: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

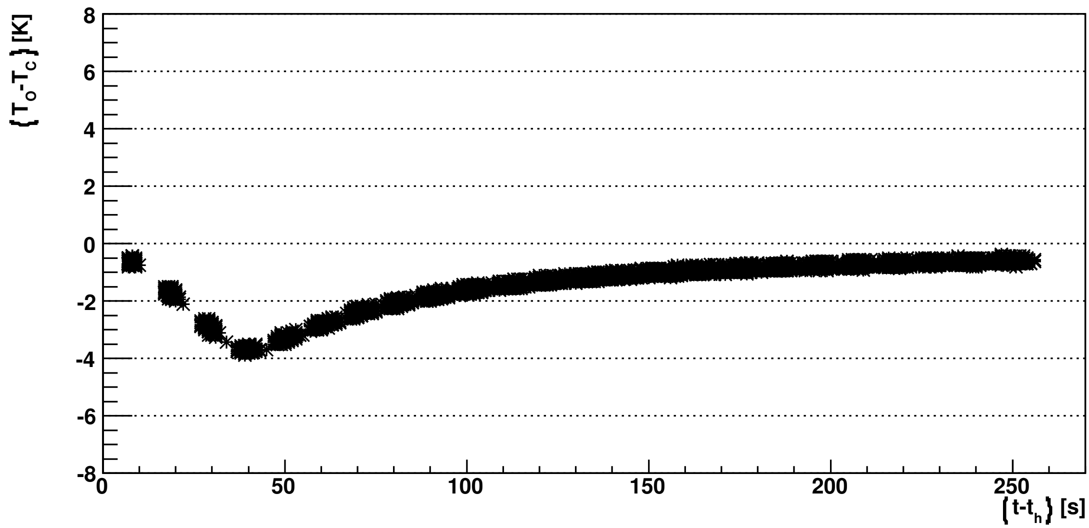

Valve 53: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

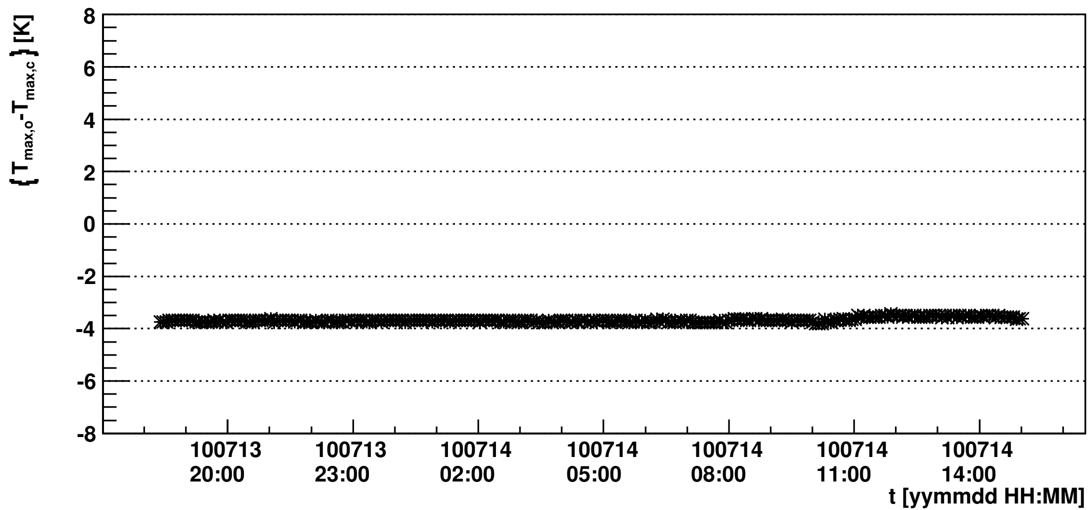

Valve 53: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -3.219
RMS 0.09365
#
200
180
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

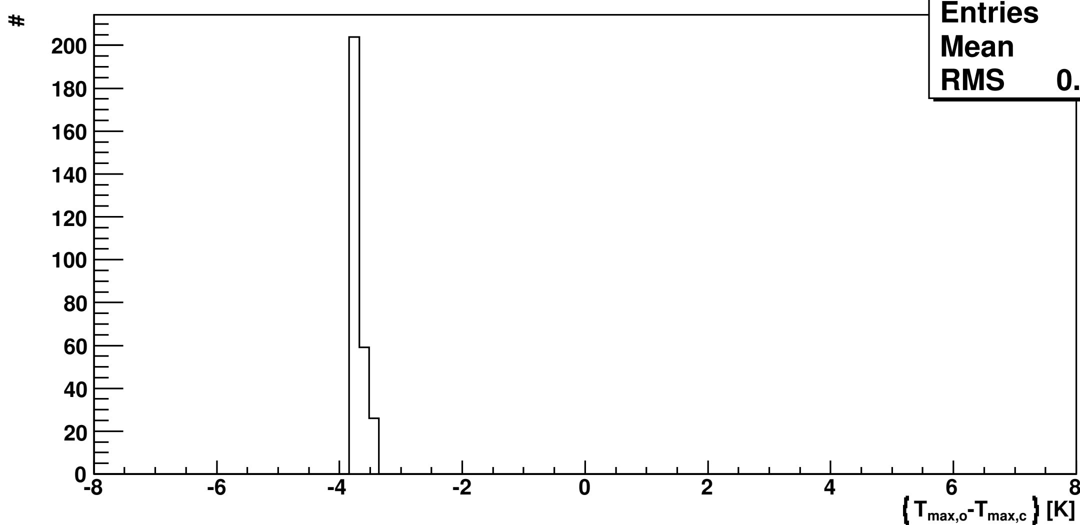

Valve 54: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

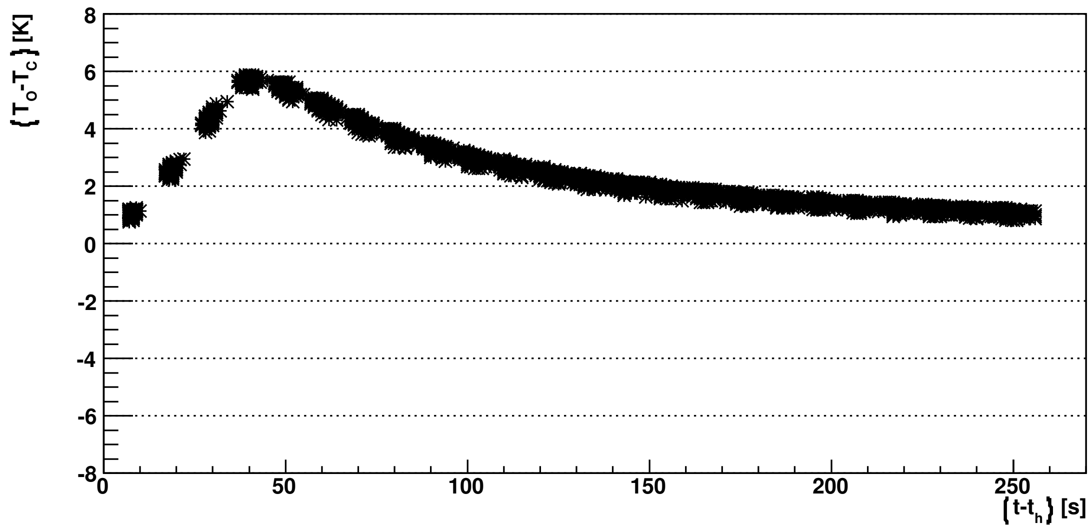

Valve 54: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

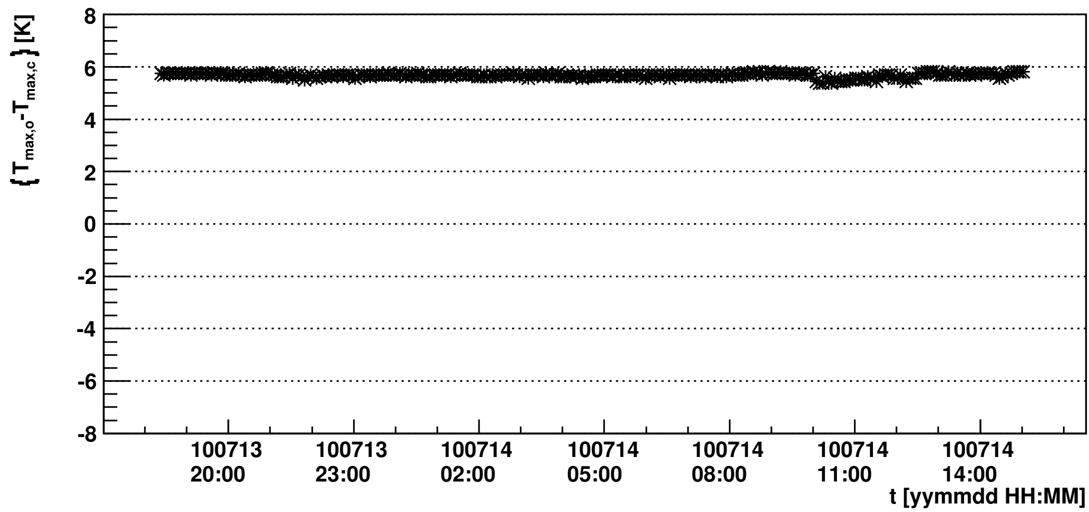

Valve 54: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -3.584
RMS 0.08448
#
180
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

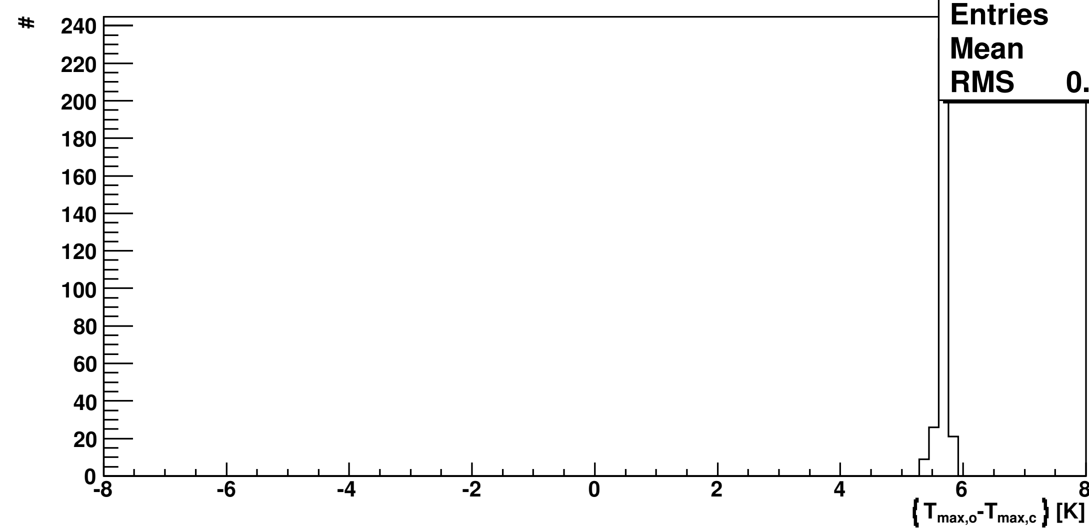

Valve 55: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

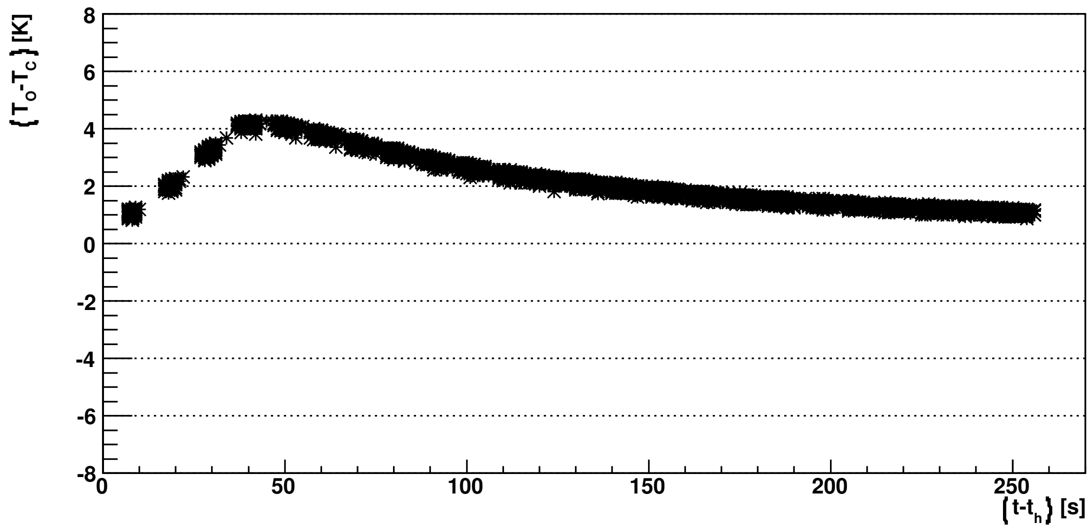

Valve 55: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

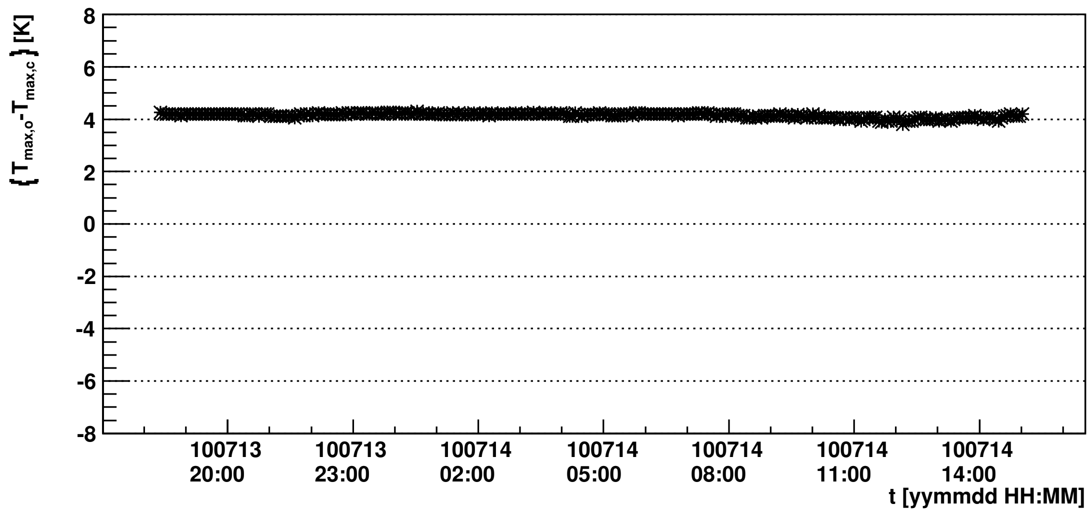

Valve 55: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -3.675
RMS 0.07874
#
200
180
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

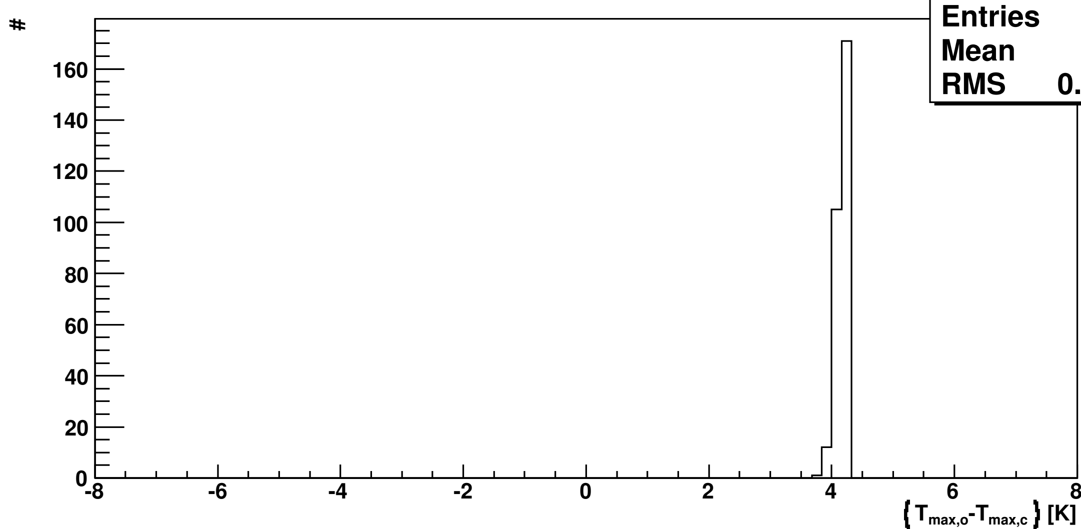

Valve 56: Opentemp - Closetemp
$\{T_O - T_C\}$ [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
$\{t - t_h\}$ [s]

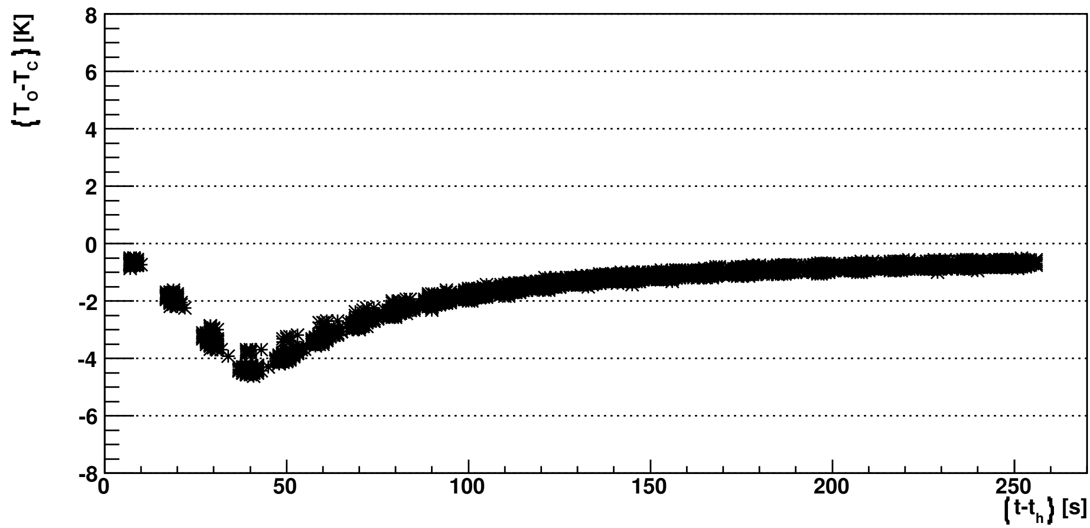

Valve 56: OpenMaxtemp - CloseMaxtemp
$\{T_{max,o} - T_{max,c}\}$ [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]


Valve 56: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean 5.676
RMS 0.07963
#
240
220
200
180
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
$\{T_{max,o} - T_{max,c}\}$ [K]

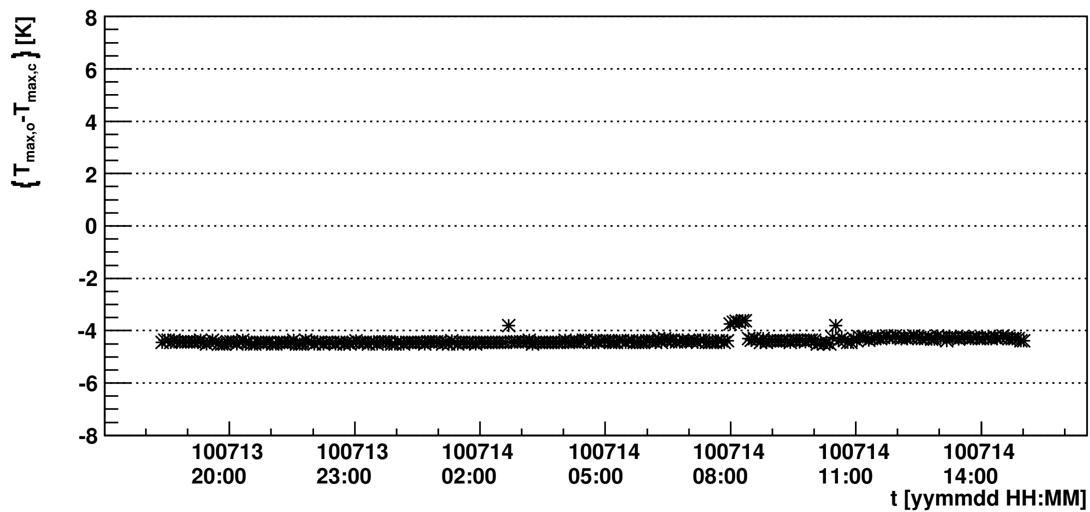

Valve 57: Opentemp - Closetemp
{T_O-T_C} [K]
{t-t_h} [s]


Valve 57: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
t [yymmdd HH:MM]

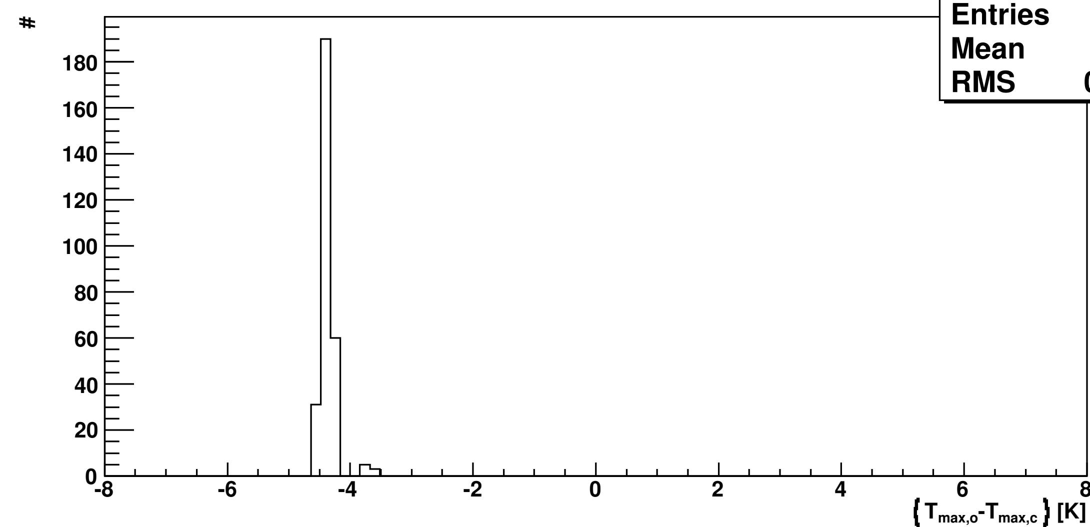

Valve 57: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean 4.146
RMS 0.08149
#
{T_max,o-T_max,c} [K]



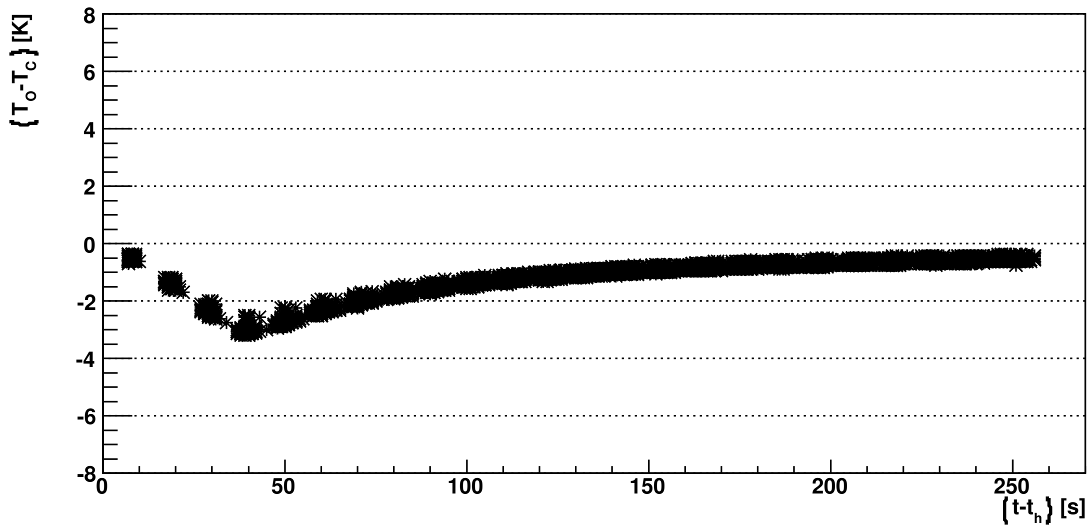
Valve 61: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

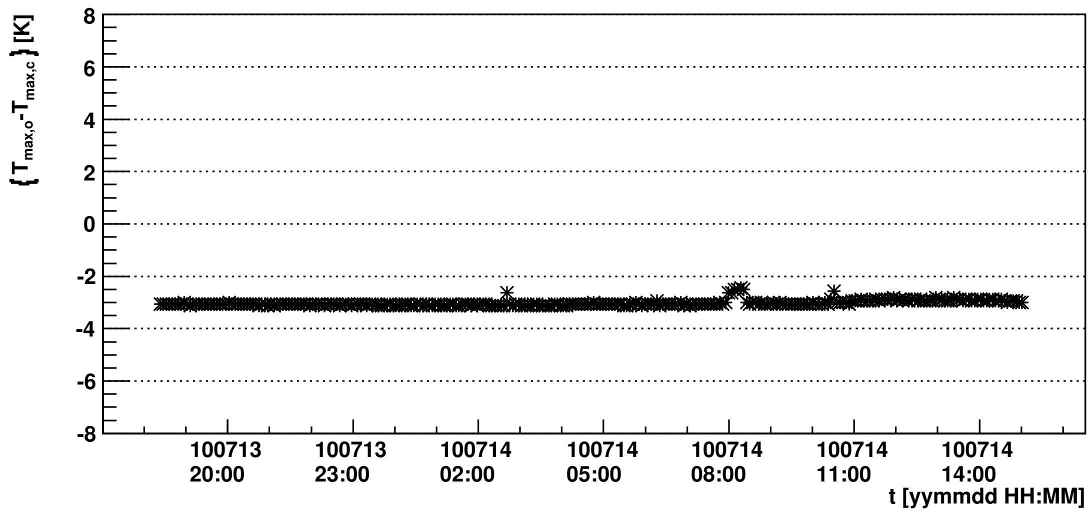


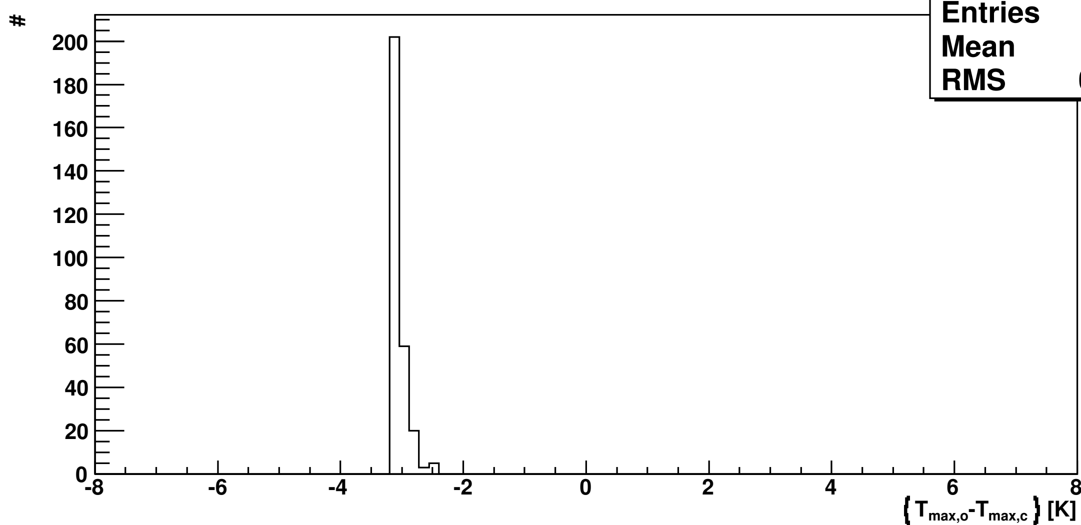
Valve 61: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

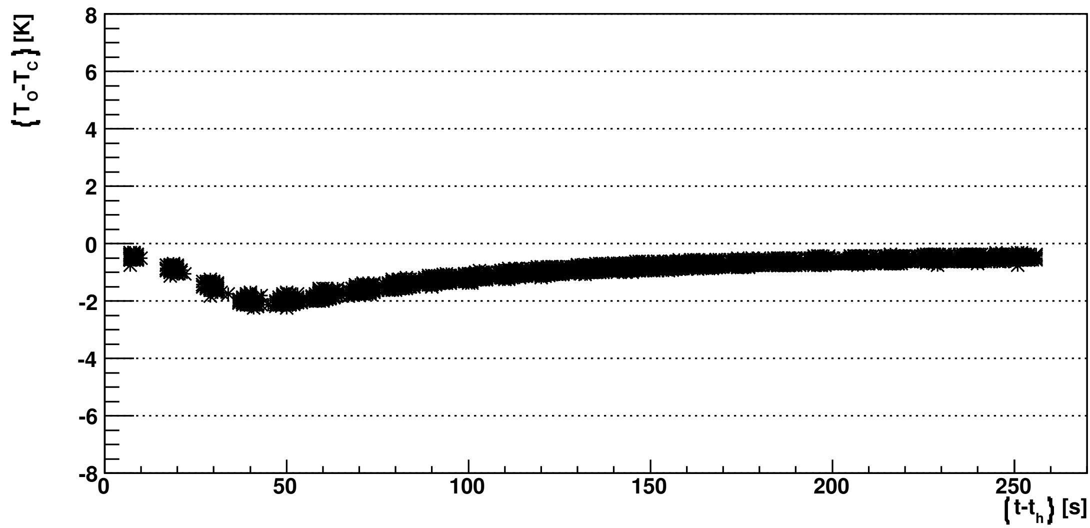


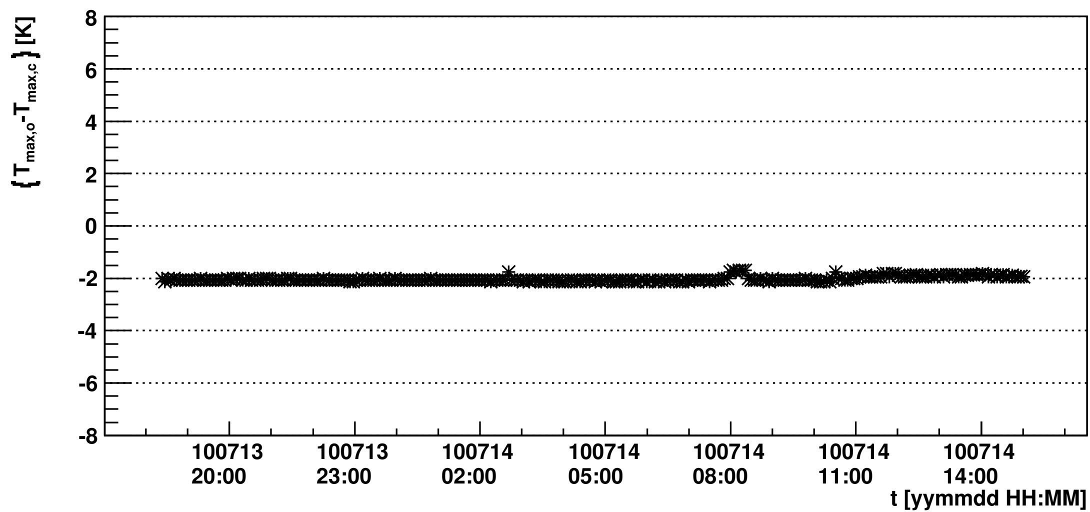
Valve 61: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -4.376
RMS 0.1365
#
180
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

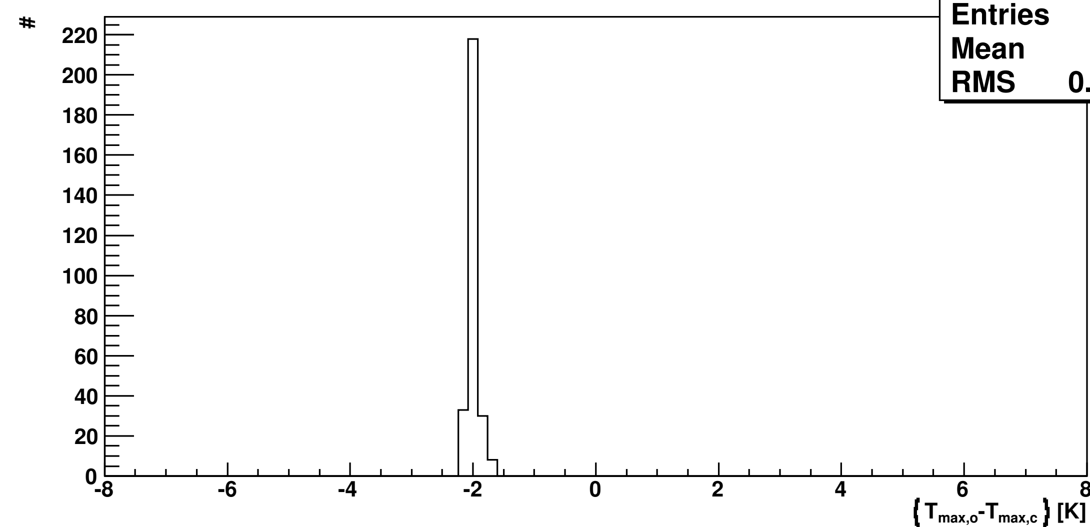

Valve 62: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

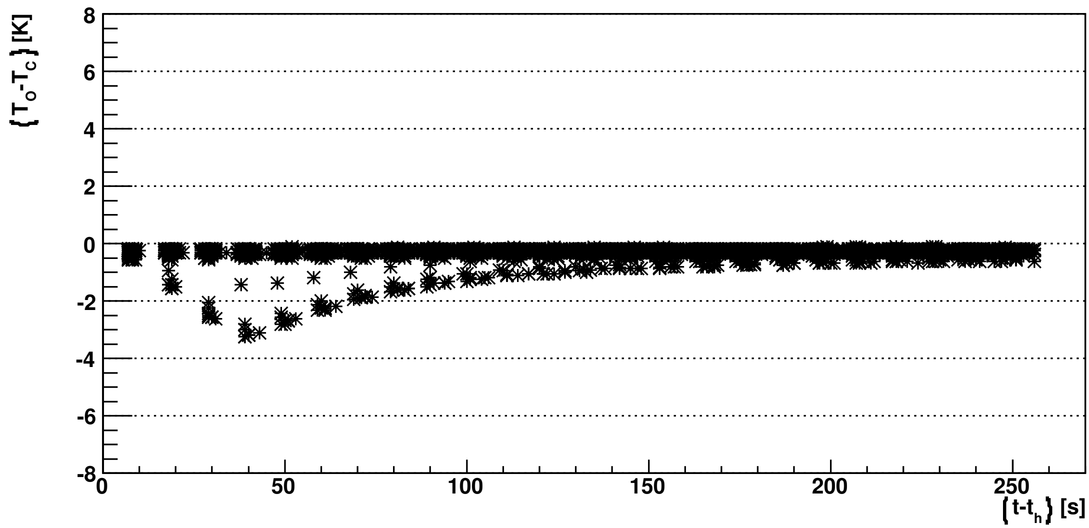

Valve 62: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

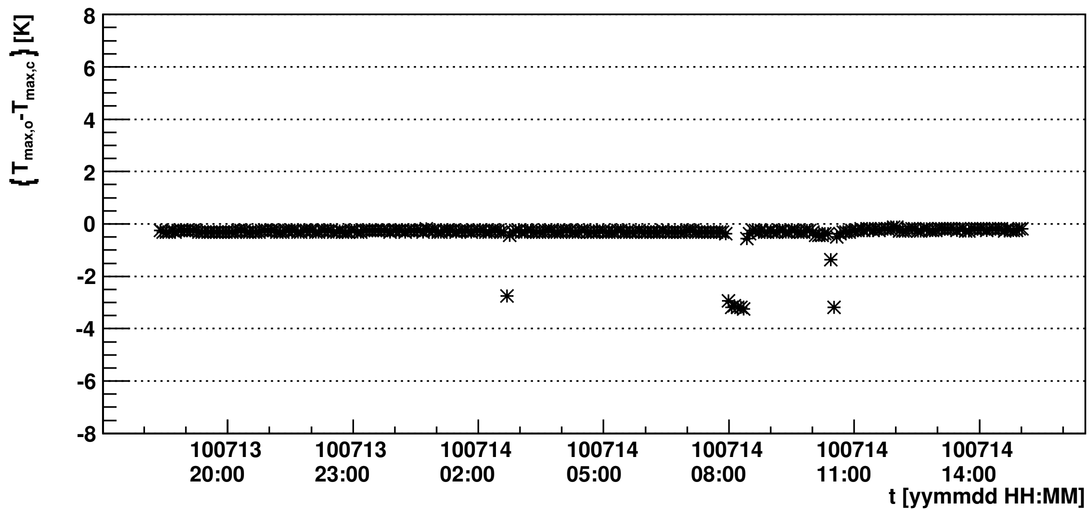

Valve 62: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -3.024
RMS 0.1101
#
200
180
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

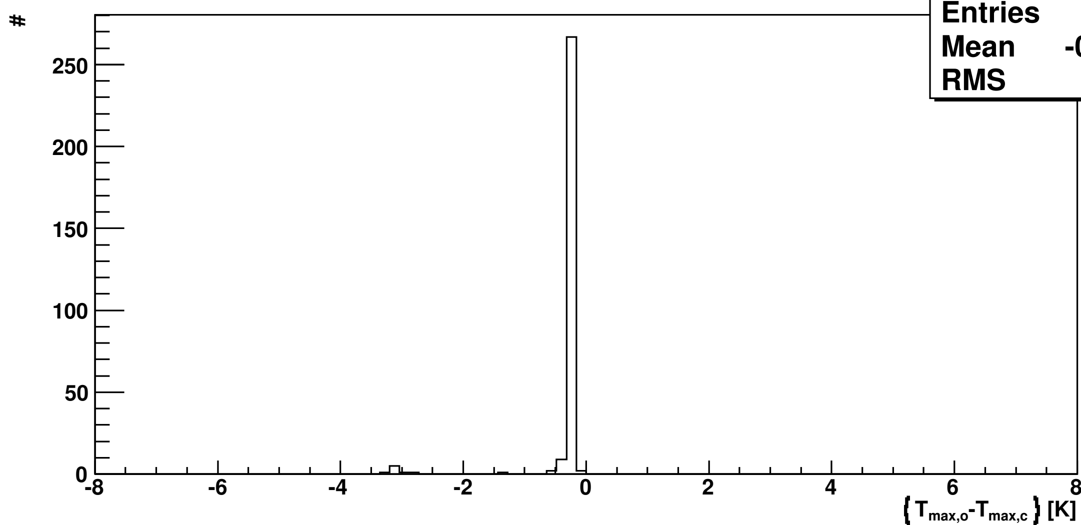

Valve 63: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

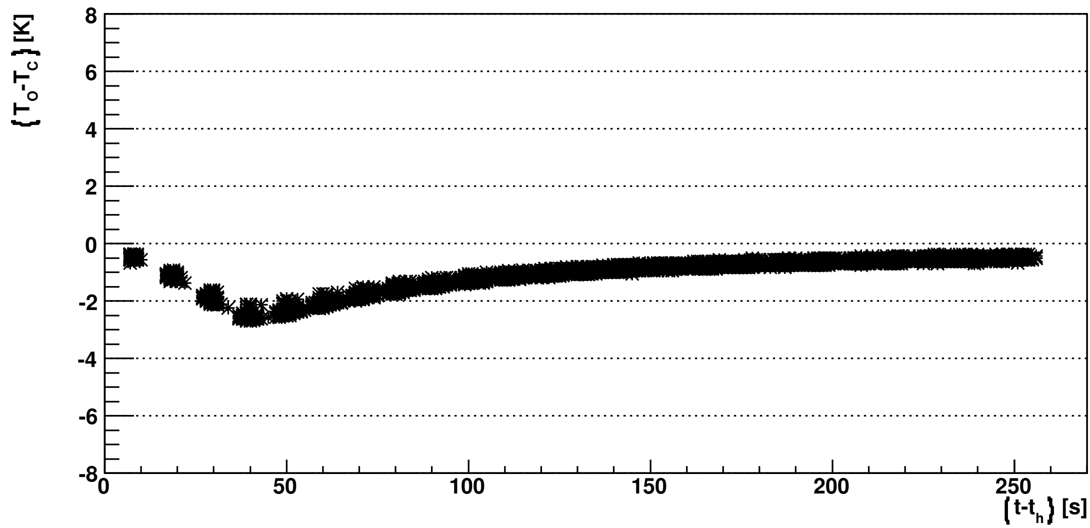

Valve 63: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

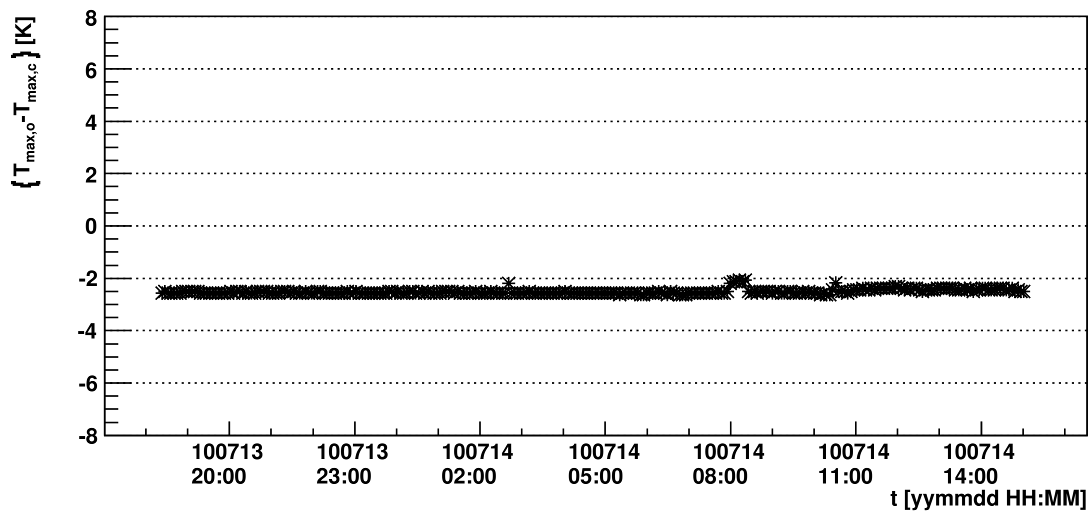

Valve 63: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -2.022
RMS 0.08932
#
220
200
180
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

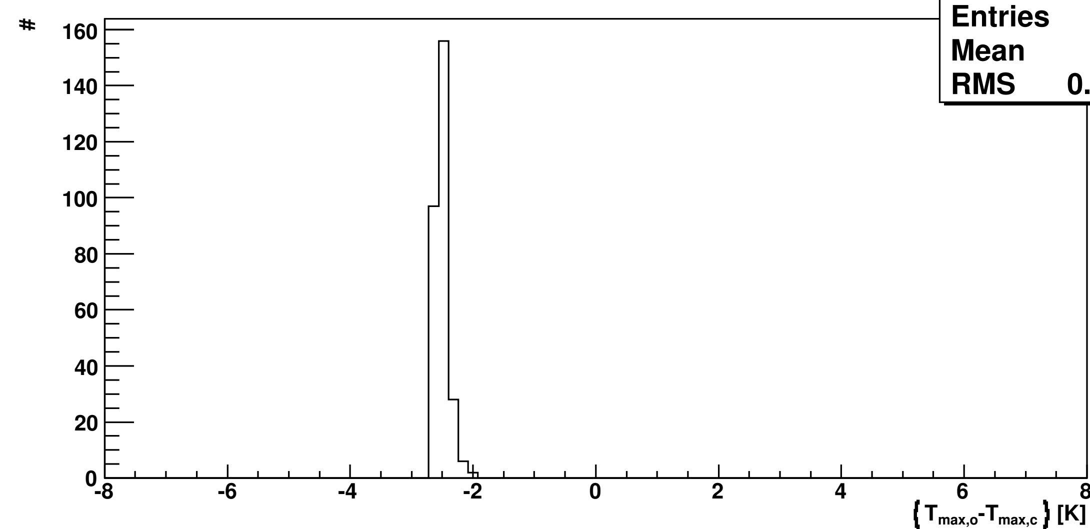

Valve 64: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

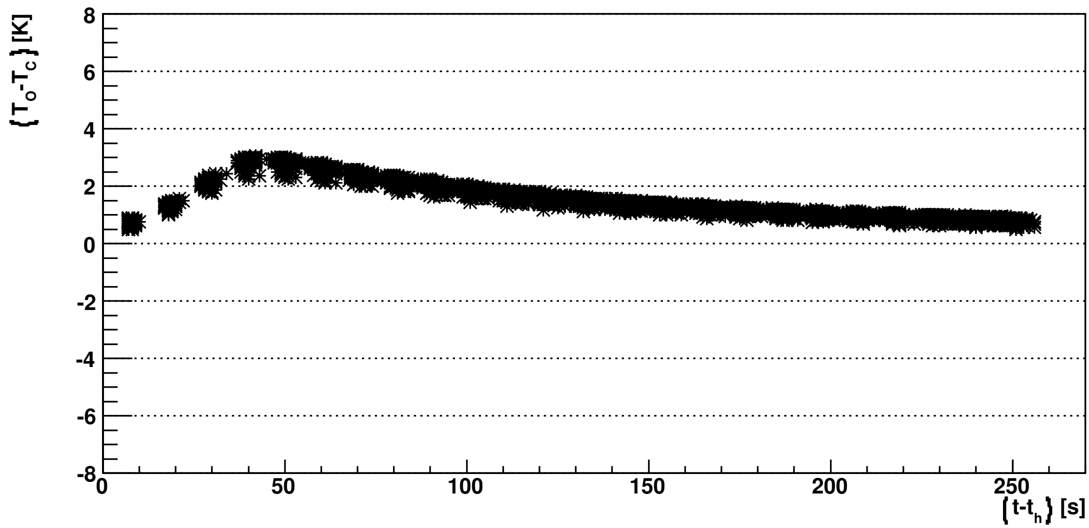

Valve 64: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

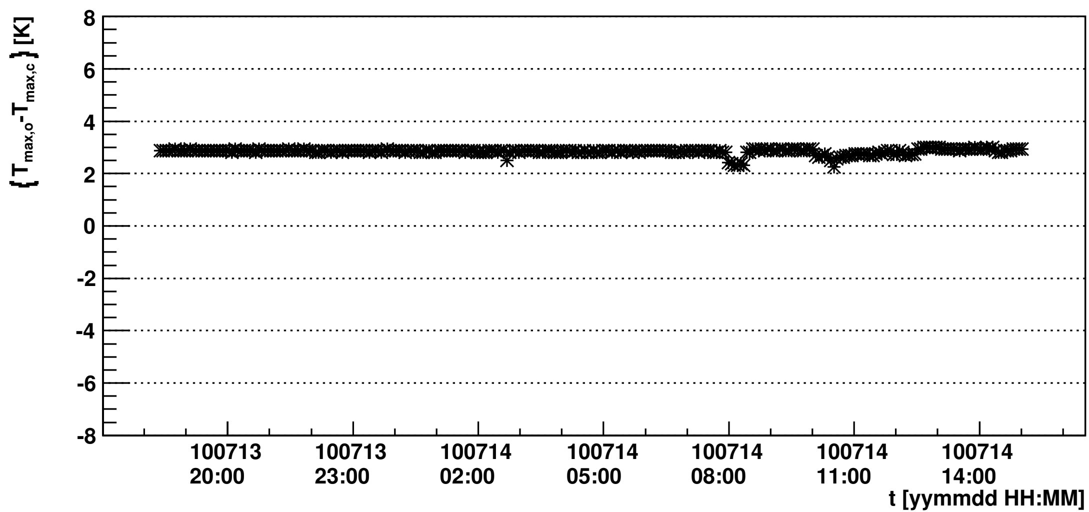

Valve 64: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -0.3548
RMS 0.471
#
250
200
150
100
50
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

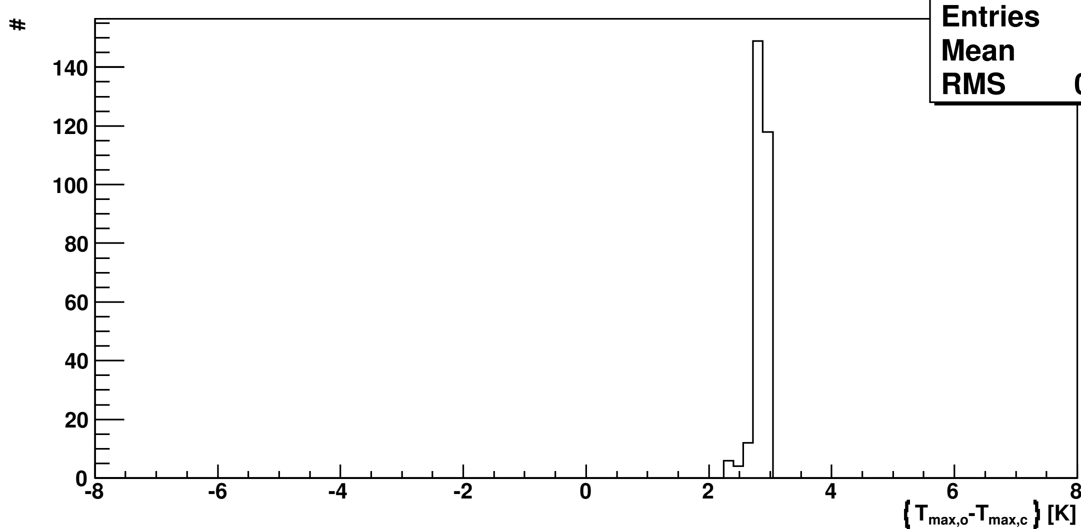

Valve 65: Opentemp - Closetemp
$\{T_o-T_C\}$ [K]
$\{t-t_h\}$ [s]

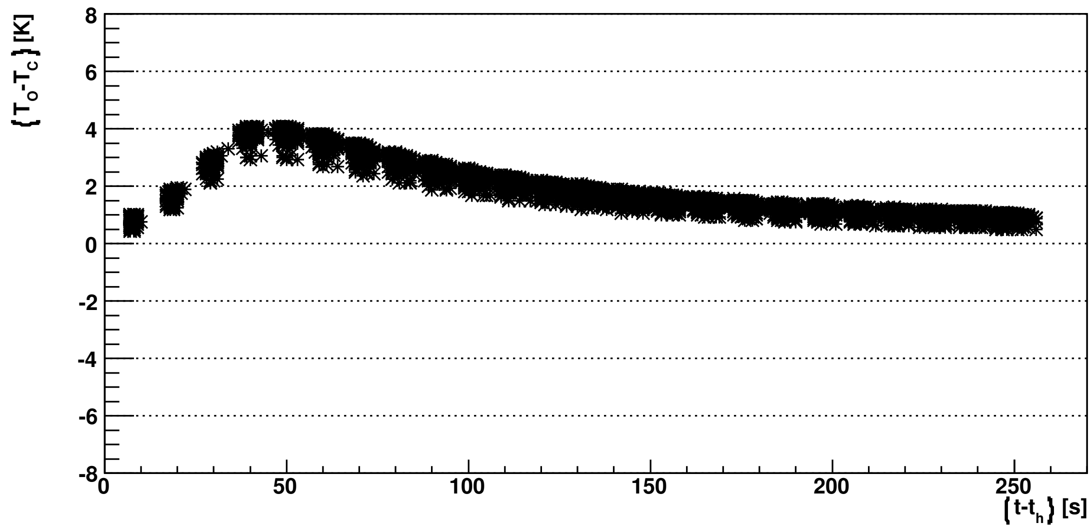

Valve 65: OpenMaxtemp - CloseMaxtemp
$\{T_{max,o}-T_{max,c}\}$ [K]
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

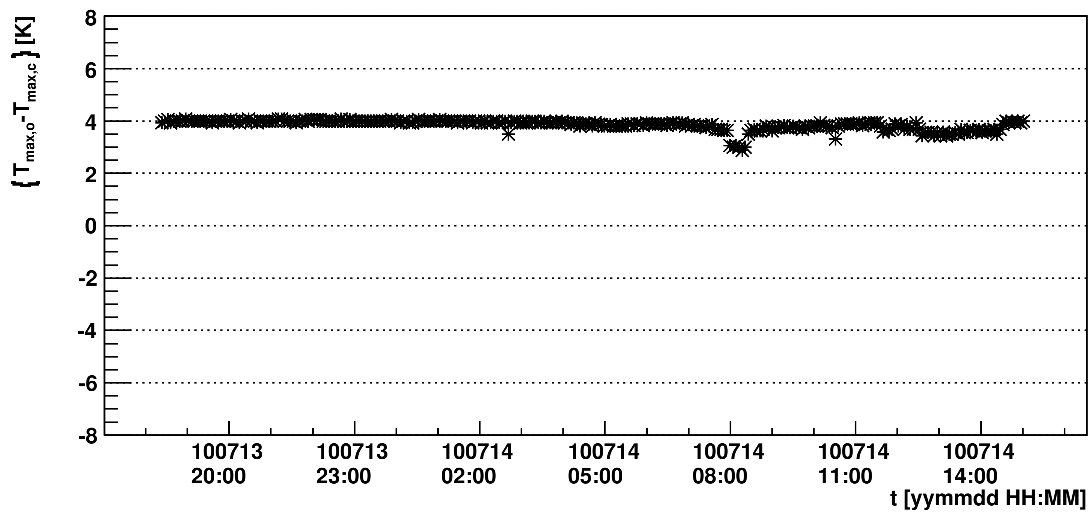

Valve 65: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -2.509
RMS 0.09205
#
$\{T_{max,o}-T_{max,c}\}$ [K]

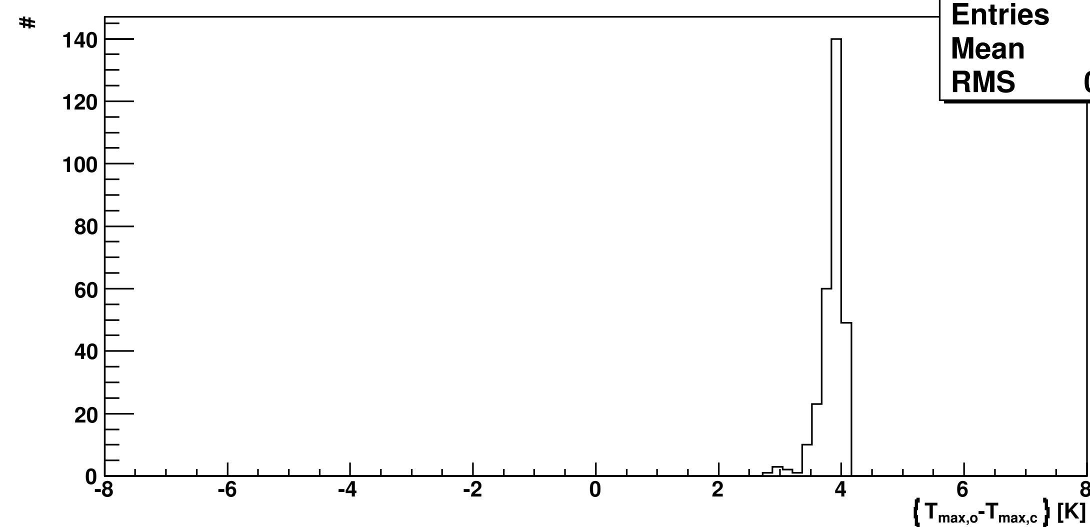

Valve 66: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

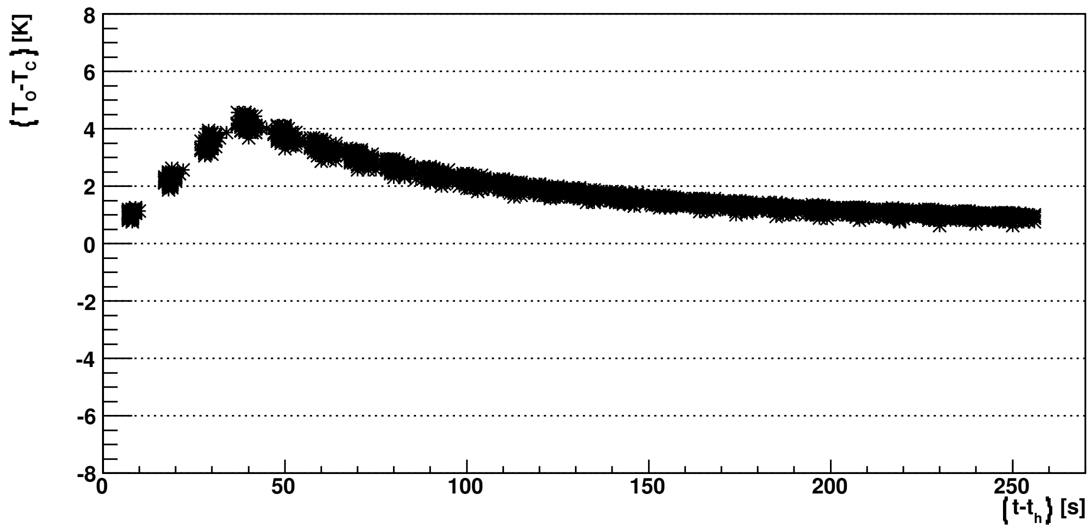

Valve 66: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

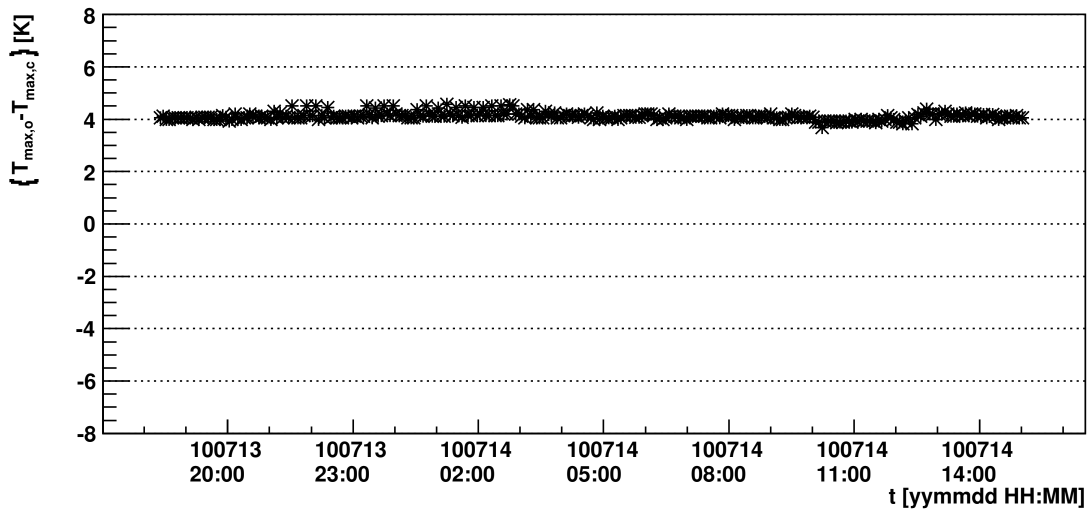

Valve 66: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean 2.845
RMS 0.1087
#
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

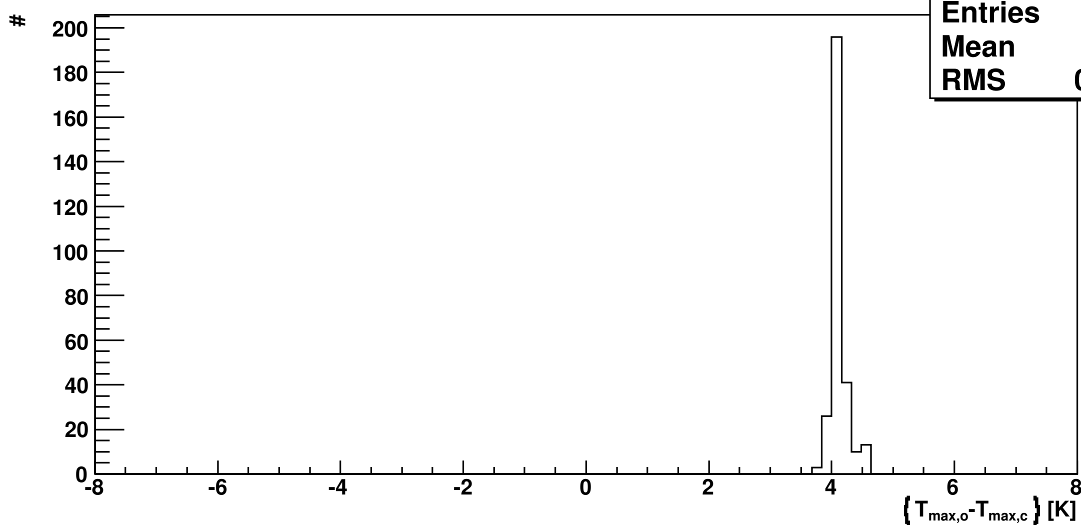

Valve 67: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]

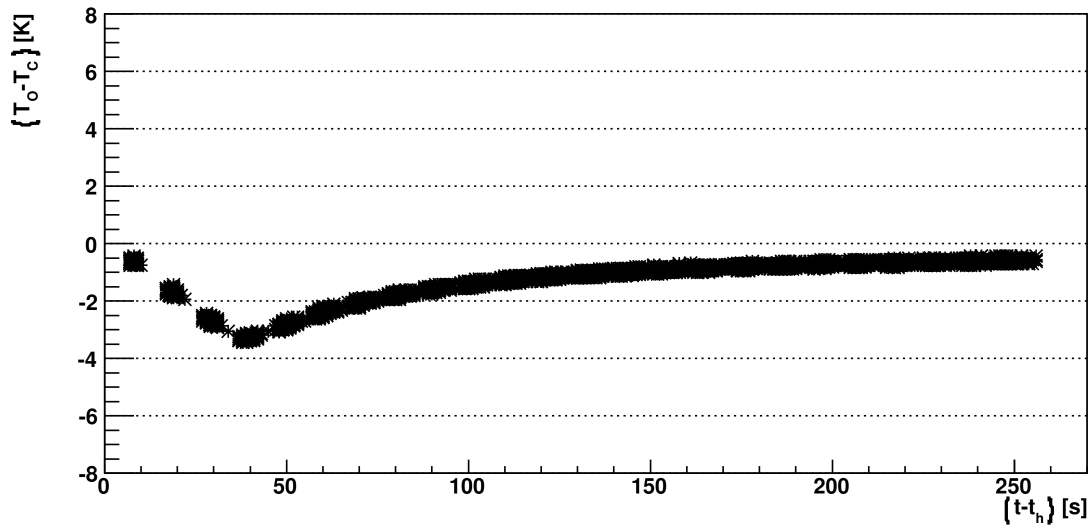

Valve 67: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]

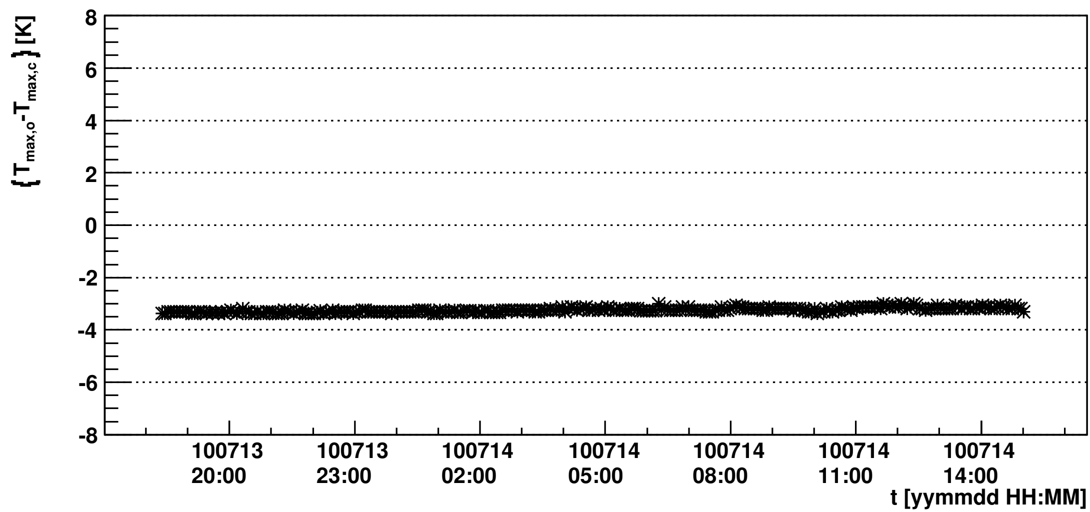

Valve 67: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean 3.858
RMS 0.1986
#
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{T_max,o-T_max,c} [K]

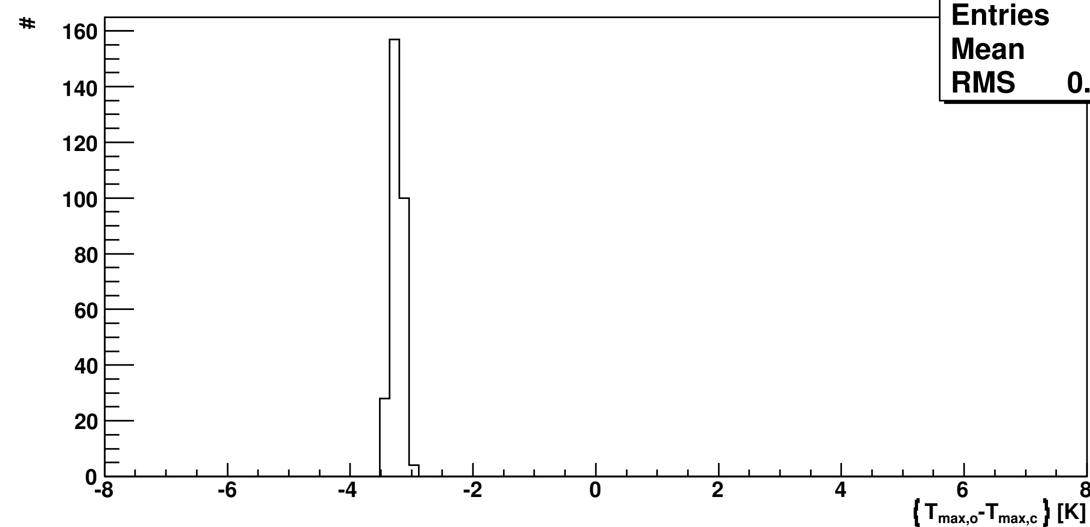

Valve 101: Opentemp - Closetemp
{T_O-T_C} [K]
{t-t_h} [s]


Valve 101: OpenMaxtemp - CloseMaxtemp
{T_max,o-T_max,c} [K]
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]


Valve 101: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean 4.113
RMS 0.1344
#
{T_max,o-T_max,c} [K]


Valve 102: Opentemp - Closetemp
{T_O-T_C} [K]
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
{t-t_h} [s]


Valve 102: OpenMaxtemp - CloseMaxtemp
{$T_{max,o}$-$T_{max,c}$} [K]
8
6
4
2
0
-2
-4
-6
-8
100713 20:00
100713 23:00
100714 02:00
100714 05:00
100714 08:00
100714 11:00
100714 14:00
t [yymmdd HH:MM]


Valve 102: OpenMaxtemp - CloseMaxtemp
hMaxDiff
Entries 289
Mean -3.243
RMS 0.08478
#
160
140
120
100
80
60
40
20
0
-8
-6
-4
-2
0
2
4
6
8
{$T_{max,o}$-$T_{max,c}$} [K]

B Quellcodes

B.1 MASTERD & MINID

Da bei MINID lediglich die Pfade zu den Dateien verschieden sind, wird hier nur der Quellcode von MASTERD angeführt.

masterd.h

```
#ifndef _masterd
#define _masterd

#define mdConfigFile "/home/messung/masterdaemon/masterdConfig.txt"
#define mdLogFile "/home/messung/masterdaemon/log/masterdLog.txt"
#define mdCheckCommand "/home/messung/masterdaemon/checkprocess"
#define mdScanForStopCommand "/home/messung/masterdaemon/scanforstop"

typedef struct{
    char name[375];
    char command[375];
    char config_file[375];
} process_t;

typedef struct process_list{
    process_t process;
    struct process_list *next;
} process_list_t;

#endif
```

masterd.c

```
#include <stdio.h>
#include <unistd.h>
#include "masterd.h"
#include "../../parser/source/parser.h"

int loop = 1; //main loop flag
int debug = 0; //debug level
#define DBG if(debug > 0)

process_list_t *plist = NULL;

void addp(char *name, char *cmd, char *cfg){
    process_list_t *newp;

    newp = malloc(sizeof(process_list_t));
    strncpy((char*)&(newp->process.name), name, 375);
    strncpy((char*)&(newp->process.command), cmd, 375);
    strncpy((char*)&(newp->process.config_file), cfg, 375);
    newp->next = plist;
    plist = newp;
}

void clearp(void){
    process_list_t *temp;

    while(plist != NULL){
        temp = plist->next;
```

```
28              free(plist);
29              plist = temp;
30          }
31  }
32
33  int read_cfg()
34  {
35      // Commands:
36      // $WATCHPROCESS, 3, processname, command, config file
37
38      int anz=0;
39      command_t* liste;
40
41      anz = anzahl_commands_in_file(mdConfigFile);
42      DBG printf("Commands in config file: %d\n", anz);
43      liste = (command_t*)malloc(sizeof(command_t) * anz);
44      read_cfg_file(mdConfigFile, liste);
45
46      int i;
47      clearp();
48      for(i = 0 ; i < anz ; i++){
49          switch (liste[i].cmd_nr){
50              case 1:
51                  loop = 0;
52                  break;
53              case 1001:
54                  DBG printf("Watching: %s # %s # %s\n", liste[i].
                        parameter[0].value, liste[i].parameter[1].value,
                        liste[i].parameter[2].value);
55                  addp(liste[i].parameter[0].value, liste[i].parameter[1].
                        value, liste[i].parameter[2].value);
56                  break;
57          }
58      }
59
60      free(liste);
61      return 1;
62  }
63
64  char lastline[1024] = "";
65
66  int checkprocess(process_t *pro){
67      char temp[1024];
68      FILE* check;
69
70      sprintf(temp, "%s \"%s\" \"%s\" 2>&1", mdCheckCommand, pro->name,
            pro->command);
71      DBG printf("Executing: %s\n", temp);
72
73      check = popen(temp, "r");
74      while(fgets(lastline, 1024, check)){
75          DBG printf(lastline);
76      };
77      return pclose(check) / 256;
78  //  return system(temp) / 256;
79  }
80
81  int scanforstop(char *fname){
```

```
82     char temp[1024];
83     sprintf(temp, "%s \"%s\"", mdScanForStopCommand, fname);
84     DBG printf("Executing: %s\n", temp);
85     return system(temp) / 256;
86 }
87
88 void watchprocesses(void){
89     char temp[1024];
90     process_list_t *current;
91
92     int ret;
93
94     current = plist;
95     while(current != NULL){
96         if(scanforstop((char*)&(current->process.config_file))){
97             DBG printf("Skipping process > %s <.\n", current->process.name);
98         }else{
99             ret = checkprocess(&(current->process));
100            switch(ret){
101                case 0:
102                    DBG printf("Process > %s < not running. Executed command > %s <.\n", current->process.name, current->process.command);
103                    sprintf(temp, "Process > %s < not running. Executed command > %s <.", current->process.name, current->process.command);
104                    write_log(mdLogFile, temp, 1);
105                    if(lastline[0]){
106                        int i;
107                        for(i=0; i<1024; i++)
108                            if(lastline[i] == '\n') break;
109                        lastline[i] = 0;
110                        sprintf(temp, "Last returned line: %s", lastline);
111                        write_log(mdLogFile, temp, 1);
112                        lastline[0] = 0;
113                    }
114                    break;
115                case 1:
116                    DBG printf("Process > %s < still running.\n", current->process.name);
117                    break;
118                case 255:
119                    sprintf(temp, "Error checking process > %s <!", current->process.name);
120                    write_log(mdLogFile, temp, 1);
121                    DBG fprintf(stderr, "Error checking process > %s <!\n", current->process.name);
122                    break;
123                default:
124                    DBG printf("Defaulted: %d\n", ret);
125            }
126        }
127        current = current->next;
128    }
129 }
130
```

```
131 int main (int argc, char *argv[])
132 {
133     int c;
134     while((c = getopt(argc, argv, "d:")) != -1){
135         switch(c){
136             case 'd':
137                 sscanf(optarg, "%d", &debug);
138                 break;
139             case '?':
140                 printf("Unknown argument or missing parameter.\n");
141                 return -1;
142         }
143     }
144 
145     if(debug==0) //daemonize when not debugging
146         daemon(0,0);
147 
148     write_log(mdLogFile, "Masterd started.", 1);
149     while(loop){ //main loop
150         read_cfg();
151         watchprocesses();
152         sleep(1);
153     }
154 
155     //clean up
156     clearp();
157 
158     write_log(mdLogFile, "Stopped.", 1);
159     return 0;
160 }
```

B.2 GASMATRIX

Da Teile von GASMATRIX von [5] übernommen wurden und der Quellcode insgesamt sehr lang ist, wird hier nur eine verkürzte Version mit den Routinen gezeigt, die modifiziert oder neu geschrieben wurden.

gasmatrix.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <ctype.h>
4 #include <unistd.h>
5 #if !defined(AIX) && !defined(SOLARIS) && !defined(FREEBSD) && !defined(
    DARWIN)
6 #include <getopt.h>
7 #endif /* !AIX and !SOLARIS and !FREEBSD and !DARWIN */
8 #include <sys/types.h>
9 #include <sys/stat.h>
10 #include <sys/time.h>
11 #include <time.h>
12 #include <math.h>
13 #include <string.h>
14 #include <fcntl.h>
15 #include <strings.h>
16 #include <stdint.h>
17 #include "ad26.h"
18 
```

```
19  // Include endian.h
20  #if DARWIN
21  #include <machine/endian.h>
22  #endif
23  #if FREEBSD
24  #include <sys/endian.h>
25  #endif
26  #if !defined(DARWIN) && !defined(FREEBSD)
27  #include <endian.h>
28  #endif
29
30  #ifdef LINUX
31  #ifndef OWUSB
32  #ifdef LOCKDEV
33  #include <lockdev.h>
34  #endif
35  #endif
36  #endif
37
38  #include "gasmatrix.h"
39  #include "device_name.h"
40  #include "ownet.h"
41  #include "owproto.h"
42  #include "../../parser/source/parser.h"
43
44  //Define Paths:
45  #define _GASMATRIXCONFIGFILE "/home/messung/gasmatrix/gasmatrixconfig.
        txt"
46  #define _GASCONFIGLOG "/home/vdc/data/gasmatrix/log/ConfigLog.txt"
47  #define _VALVELOG "/home/vdc/data/gasmatrix/log/ValveLog.txt"
48  #define _CURRENTSTATEVALVES "/home/messung/currentstateValves.txt"
49  #define _CURRENTSTATETEMP_tmp "/home/messung/gasmatrix/
        currentstateTemperatures.txt.tmp"
50  #define _CURRENTSTATETEMP "/home/messung/currentstateTemperatures.txt"
51  #define _TEMPWARNLOG "/home/vdc/data/temperatures/log/TempWarning.txt"
52  #define _TEMPLOG "/home/vdc/data/temperatures/log/TempLog.txt"
53  #define _TEMPTIMEDATALOG "/home/vdc/data/temperatures/log/Temp_Time_Data
        .txt"
54  #define _USBLOG "/home/vdc/data/gasmatrix/log/USBLog.txt"
55
56  /* Setup the correct getopt starting point */
57  #ifdef LINUX
58  #define GETOPTEOF -1
59  #define OPTINDSTART 0
60  #endif
61
62  #define uchar unsigned char
63
64  extern char     *optarg;
65  extern int   optind, opterr, optopt;
66
67  #if defined(FREEBSD) || defined(DARWIN)
68  extern int optreset;
69  #endif /* FREEBSD or DARWIN */
70  int opts = 0; // usbses wegen quiet mode
71
72  //--- Defines for our hardware config ---
73  #define ANZAHL_DS9490 2
```

```
74 #define MAXTEMP 120
75 #define MAXSINGLE 100
76 #define MAXSWITCH 10
77 #define INSTALLEDTEMP 96
78 #define INSTALLEDSWITCH 6   /* Hardware 4 , with 8 Channels , 6 Channels
       used*/
79 #define INSTALLEDSINGLE 100
80
81 /*--- Globale Variablen ---*/
82      int switchkanal_fuer_spalte[7]; // Kanal für jede Spalte(1-6)
83      char singleSensorPos[MAXSINGLE][20];
84      int high_limit[MAXSINGLE];
85      int low_limit [MAXSINGLE];
86      int anz_single_temp =0;
87      int cfg_position[48]; // config Ventilposition (0 zu , 1 auf)
88      int cur_position[48]; // current Ventilposition (0 zu , 1 auf)
89      float cur_opentemp[48];
90      float cur_closetemp[48];
91      float max_opentemp[48];
92      float max_closetemp[48];
93      float min_opentemp[48];
94      float min_closetemp[48];
95      float mean_opentemp[48];
96      float mean_closetemp[48];
97      float mean_opennum[48];
98      float mean_closenum[48];
99      int loop = 1; // used for STOP Command in CFG file
100     int debug = 0;
101     int verbose = 0;
102     int sleeptime = 6;
103     int runs = 240 / 10;
104     char logstring[1000];
105
106     int mindiff = 1; // 1°C Temp Diff subject to change ...
107     int heatingtime = 60; // Zeit in Sekunden die maximal geheizt wird
108
109     unsigned char sensor_list_roms[MAXTEMP*8];
110     unsigned char sensorsingle_list_roms[MAXSINGLE*8];
111     float mean_singletemp[MAXSINGLE];
112     float mean_singlenum[MAXSINGLE];
113     unsigned char switch_list_roms[MAXSWITCH*8];
114     int sw_port[6] = {0,0,0,0,0,0};
115     int temp_port[MAXTEMP] ;
116     int tempsingle_port[MAXSINGLE];
117     struct _roms sensor_list;
118     struct _roms sensorsingle_list;
119     struct _roms switch_list;
120
121     int valvenum[48] = {    11, 12, 13, 14, 15, 16, 17, 0,
122                 21, 22, 23, 24, 25, 26, 27, 0,
123                 31, 32, 33, 34, 35, 36, 37, 0,
124                 41, 42, 43, 44, 45, 46, 47, 101,
125                 51, 52, 53, 54, 55, 56, 57, 102,
126                 61, 62, 63, 64, 65, 66, 67, 0   };
127     float meantemp;
128     float tempRMS;
129     float tempnum;
130     float meantoptemp;
```

```
131     float temptopRMS;
132     float temptopnum;
133     float meanbottomtemp;
134     float tempbottomRMS;
135     float tempbottomnum;
136     float mean_meantemp;
137
138
139 /*--- Ende Globale Variablen ---*/
140
141 int heizung(int spalte ){
142 //int heizung(int spalte, struct _roms *switch_list){
143 // ausschalten / anschalten / ausschalten, so dass steigene und fallende
        Flanke vorhanden
144     int htc= 0 ;
145     if (debug > 0){printf("Start heizung : %d \n",spalte);}
146     for (htc = 0 ; htc < 10 ; htc++){
147         if(htc >= 1) msDelay(500);
148         if (write_switch( &switch_list, spalte-1, sw_port[spalte-1], 0
            xFF) == TRUE){break;}
149         if (htc == 9) { return FALSE;}
150     }
151
152
153     if (switchkanal_fuer_spalte[spalte-1] == 0){
154         for (htc = 0 ; htc < 10 ; htc++){
155             msDelay(500);
156             if (write_switch( &switch_list, spalte-1, sw_port[spalte-1],
                    0xFC + 0x02)== TRUE) {break;} // 10 , to turn channel A
                  on
157             if (htc == 9) { return FALSE;}
158         }
159     } else {
160         for (htc = 0 ; htc < 10 ; htc++){
161             msDelay(500);
162             if (write_switch( &switch_list, spalte-1, sw_port[spalte-1],
                    0xFC + 0x01)== TRUE) {break;} // 01 , to turn channel B
                  on
163             if (htc == 9) { return FALSE;}
164         }
165     }
166     for (htc = 0 ; htc < 10 ; htc++){
167         msDelay(500);
168         if (write_switch( &switch_list, spalte-1, sw_port[spalte-1], 0
            xFF)== TRUE ) { break;} // turn all off
169         if (htc == 9) { return FALSE;}
170     }
171     return TRUE;
172 }
173
174 int read_spalte (struct _roms *sensor_list, int spalte, int portnum){
175     float opentemp;
176     float closetemp;
177     int i,j,s,r ;
178     char help[512];
179     int max;
180     max = spalte*8 - 1;
181     if((spalte == 4) || (spalte == 5))
```

```
182             max++;
183
184 /*    for(i=0; i<3; i++){
185             if(start_convert_all_temp(0))
186                 break;
187       }*/
188
189       for ( i = (spalte-1)*8 ; i < max ; i++){
190             s = (i/8)+1;
191             r = (i%8)+1;
192             //read_temp(sensor_list, (2*i)   ,portnum, &closetemp);
193             for(j=0; j<10; j++){
194                 closetemp = -1;
195                 read_temp_no_convert(sensor_list, (2*i)   ,portnum, &
                        closetemp);
196                 if(closetemp > 0)
197                     break;
198             }
199             sprintf(help,"Sensorposition > Valve %2d (col %d ,row %d)
                    closetemp < Temperature is %3.2f [Celsius]",valvenum[i],s,r,
                    closetemp);
200             if(verbose)
201                 write_log(_TEMPLOG,help,1);
202             //read_temp(sensor_list, (2*i)+1,portnum, &opentemp);
203             for(j=0; j<10; j++){
204                 opentemp = -1;
205                 read_temp_no_convert(sensor_list, (2*i)+1,portnum, &opentemp
                        );
206                 if(opentemp > 0)
207                     break;
208             }
209             sprintf(help,"Sensorposition > Valve %2d (col %d ,row %d)
                    opentemp < Temperature is %3.2f [Celsius]",valvenum[i],s,r,
                    opentemp);
210             if(verbose)
211                 write_log(_TEMPLOG,help,1);
212
213             if(debug >= 1)
214                 printf("TempSensorNr(auf): %d TEMP: %2.3f C TempSensorNr(zu)
                        : %d TEMP: %2.3f C\n", 2*i, opentemp, 2*i+1, closetemp);
215             if(opentemp > 0){
216                 if(opentemp < min_opentemp[i])
217                     min_opentemp[i] = opentemp;
218                 if(opentemp > max_opentemp[i])
219                     max_opentemp[i] = opentemp;
220             mean_opentemp[i] += opentemp;
221             mean_opennum[i]++;
222             }
223             if(closetemp > 0){
224                 if(closetemp < min_closetemp[i])
225                     min_closetemp[i] = closetemp;
226                 if(closetemp > max_closetemp[i])
227                     max_closetemp[i] = closetemp;
228             mean_closetemp[i] += closetemp;
229             mean_closenum[i]++;
230             }
231             cur_opentemp[i] = opentemp;
232             cur_closetemp[i] = closetemp;
```

```
233         }
234         return 1;
235     }
236 
237     void reset_min_max(void)
238     {
239         int i;
240         for(i=0; i<48; i++){
241             min_opentemp[i] = 200;
242             min_closetemp[i] = 200;
243             max_opentemp[i] = 0;
244             max_closetemp[i] = 0;
245             mean_opentemp[i] = 0;
246             mean_closetemp[i] = 0;
247             mean_opennum[i] = 0;
248             mean_closenum[i] = 0;    }
249         for(i=0; i<MAXSINGLE; i++){
250             mean_singletemp[i] = 0;
251             mean_singlenum[i] = 0;
252         }
253         mean_meantemp = 0;
254     }
255 
256     int check_spalte (struct _roms *sensor_list, int spalte, int portnum){
257         int i,s,r ;
258         int max;
259         max = spalte*8 - 1;
260         if((spalte == 4) || (spalte == 5))
261             max++;
262 
263         for ( i = (spalte-1)*8 ; i < max ; i++){
264             s = (i/8)+1;
265             r = (i%8)+1;
266 
267             if ((max_closetemp[i] - max_opentemp[i]) > mindiff) {
268                 cur_position[i] = 0;
269             }else if ((max_closetemp[i] - max_opentemp[i]) < (-mindiff)){
270                 cur_position[i] = 1;
271             }else {
272                 cur_position[i] = -1;
273             }
274             if((max_closetemp[i] < 0) || (max_opentemp[i] <0)) //Sensors
                    disfunctional?
275                 cur_position[i] = -1;
276         }
277         return 1;
278     }
279 
280     int check_cfg_cur(){
281         char help[500];
282         int i;
283 
284         char valvepositions_log[] = _VALVELOG;
285         for ( i = 0 ; i < 48 ; i++){
286             int s,r ;
287             s = (i/8)+1;
288             r = (i%8)+1;
289             if (cfg_position[i] == cur_position[i]){
```

```
290                 // Nope, alles okay
291                 /*
292                 sprintf(help, "Valve %d (col %d ,row %d) is okay",i,s,r);
293                 if (debug >= 1){ printf("%s\n",help) };
294                 write_log(valvepositions_log,help,1);*/
295             } else if ((cfg_position[i] == 1) && (cur_position[i] == 0)){
296                 sprintf(help, "Valve %d (col %d ,row %d) is closed, but 
                        configured to be opened",valvenum[i],s,r);
297                 if (debug >= 1)
298                     printf("%s\n",help);
299                 write_log(valvepositions_log,help,1);
300             } else if ((cfg_position[i] == 0) && (cur_position[i] == 1)){
301                     sprintf(help, "Valve %d (col %d ,row %d) is opened, but 
                            configured to be closed",valvenum[i],s,r);
302                 if (debug >= 1)
303                     printf("%s\n",help);
304                 write_log(valvepositions_log,help,1);
305             } else if ((cfg_position[i] == 1) && (cur_position[i] == -1)){
306                 sprintf(help, "Valve %d (col %d ,row %d) is indifferent (o: 
                        %2.3f°C, c: %2.3f C) and configured to be open",valvenum[
                        i],s,r,max_opentemp[i],max_closetemp[i]);
307                 if (debug >= 1)
308                     printf("%s\n",help);
309                 write_log(valvepositions_log,help,1);
310             }else if ((cfg_position[i] == 0) && (cur_position[i] == -1)){
311                 sprintf(help, "Valve %d (col %d ,row %d) is indifferent (o: 
                        %2.3f°C, c: %2.3f C) and configured to be closed",
                        valvenum[i],s,r,max_opentemp[i],max_closetemp[i]);
312                 if (debug >= 1)
313                     printf("%s\n",help);
314                 write_log(valvepositions_log,help,1);
315             }
316         }
317 
318         char valvepositions_current[] = _CURRENTSTATEVALVES;
319         remove(valvepositions_current);
320         sprintf(help, "Current valve positions (Calculated from temperature 
                difference(max_opentemp - max_closetemp); 1 open; 0 closed; -1 
                undefined; must value in brackets):");
321         write_log(valvepositions_current,help,1);
322 
323         tempnum = 0;
324         int s,r;
325         for ( i = 0 ; i < 48 ; i++){
326             s = (i/8)+1;
327             r = (i%8)+1;
328             if(r != 8){
329                 sprintf(help, "Valve %3d (col %d ,row %d , opentemp %2.3f , 
                        closetemp %2.3f): %2d ( %d ) %s",
330                     valvenum[i],s,r,max_opentemp[i],max_closetemp[i],
                            cur_position[i],cfg_position[i],cfg_position[i]==
                            cur_position[i] ? "   OK   " : "WARNING!");
331                 if (debug >= 1)
332                     printf("%s\n",help);
333                 write_log(valvepositions_current,help,0);
334 
335             }
336         }
```

```
337         for(s = 4; s<=5; s++){
338             r = 8;
339             i = (s-1)*8+(r-1);
340
341             sprintf(help, "Valve %3d (col %d ,row %d ,opentemp %2.3f ,
                    closetemp %2.3f): %2d ( %d ) %s",
342                 valvenum[i],s,r,max_opentemp[i],max_closetemp[i],
                        cur_position[i],cfg_position[i],cfg_position[i]==
                        cur_position[i] ? "   OK   " : "WARNING!");
343             if (debug >= 1)
344                 printf("%s\n",help);
345             write_log(valvepositions_current,help,0);
346
347         }
348
349         return 1;
350 }
351
352 void calc_mean_cabinet_temperature( void )
353 {
354         float tempsum = 0;
355         float temptopsum = 0;
356         float tempbottomsum = 0;
357         tempnum = 0;
358         temptopnum = 0;
359         tempbottomnum = 0;
360         char help[500];
361         int i,s,r;
362         for ( i = 0 ; i < 48 ; i++){
363             s = (i/8)+1;
364             r = (i%8)+1;
365             if(r != 8){
366                 //Calculate mean cabinet temperature from unheated sensors
367                 switch(cur_position[i]){
368                     case 1:
369                         tempsum += cur_closetemp[i];
370                         tempnum++;
371                         if(s <= 3){
372                             temptopsum += cur_closetemp[i];
373                             temptopnum++;
374                         }else{
375                             tempbottomsum += cur_closetemp[i];
376                             tempbottomnum++;
377                         }
378                         break;
379                     case 0:
380                         tempsum += cur_opentemp[i];
381                         tempnum++;
382                         if(s <= 3){
383                             temptopsum += cur_opentemp[i];
384                             temptopnum++;
385                         }else{
386                             tempbottomsum += cur_opentemp[i];
387                             tempbottomnum++;
388                         }
389                         break;
390                 }
391             }
```

```
392         }
393         for(s = 4; s<=5; s++){
394             r = 8;
395             i = (s-1)*8+(r-1);
396 
397             //Calculate mean cabinet temperature from unheated sensors
398             switch(cur_position[i]){
399                 case 1:
400                     tempsum += cur_closetemp[i];
401                     tempnum++;
402                     if(s <= 3){
403                         temptopsum += cur_closetemp[i];
404                         temptopnum++;
405                     }else{
406                         tempbottomsum += cur_closetemp[i];
407                         tempbottomnum++;
408                     }
409                     break;
410                 case 0:
411                     tempsum += cur_opentemp[i];
412                     tempnum++;
413                     if(s <= 3){
414                         temptopsum += cur_opentemp[i];
415                         temptopnum++;
416                     }else{
417                         tempbottomsum += cur_opentemp[i];
418                         tempbottomnum++;
419                     }
420                     break;
421             }
422         }
423         meantemp = tempsum / tempnum;
424         meantoptemp = temptopsum / temptopnum;
425         meanbottomtemp = tempbottomsum / tempbottomnum;
426 
427         tempsum = 0;
428         tempnum = 0;
429         for ( i = 0 ; i < 48 ; i++){
430             s = (i/8)+1;
431             r = (i%8)+1;
432             if(r != 8){
433                 //Calculate mean cabinet temperature RMS from unheated
                        sensors
434                 switch(cur_position[i]){
435                     case 1:
436                         tempsum += pow((meantemp - cur_closetemp[i]), 2);
437                         tempnum++;
438                         break;
439                     case 0:
440                         tempsum += pow((meantemp - cur_opentemp[i]), 2);
441                         tempnum++;
442                         break;
443                 }
444 
445             }
446         }
447         for(s = 4; s<=5; s++){
448             r = 8;
```

```
449             i = (s-1)*8+(r-1);
450             //Calculate mean cabinet temperature RMS from unheated sensors
451             switch(cur_position[i]){
452                 case 1:
453                     tempsum += pow((meantemp - cur_closetemp[i]), 2);
454                     tempnum++;
455                     break;
456                 case 0:
457                     tempsum += pow((meantemp - cur_opentemp[i]), 2);
458                     tempnum++;
459                     break;
460             }
461 
462         }
463         tempRMS = tempsum / (tempnum - 1);
464         tempRMS = sqrt(tempRMS);
465 
466         sprintf(help,"Sensorposition > Mean cabinet temperature from %2.0f
                valve matrix sensors < Temperature is %3.2f [Celsius] RMS: %3.2f
                [Celsius]",
467             tempnum, meantemp, tempRMS);
468         if (debug >= 1)
469             printf("%s\n",help);
470         if(verbose)
471             write_log(_TEMPLOG,help,1);
472         sprintf(help,"Sensorposition > Mean top cabinet temperature from
                %2.0f valve matrix sensors < Temperature is %3.2f [Celsius]",
473             temptopnum, meantoptemp);
474         if (debug >= 1)
475             printf("%s\n",help);
476         if(verbose)
477             write_log(_TEMPLOG,help,1);
478         sprintf(help,"Sensorposition > Mean bottom cabinet temperature from
                %2.0f valve matrix sensors < Temperature is %3.2f [Celsius]",
479             tempbottomnum, meanbottomtemp);
480         if (debug >= 1)
481             printf("%s\n",help);
482         if(verbose)
483             write_log(_TEMPLOG,help,1);
484         mean_meantemp += meantemp;
485 }
486 
487 int check_single(int portnum){
488     int i,j ;
489     float ctemp;
490     char help[500];
491 
492     char temperatures_current_temp[] = _CURRENTSTATETEMP_tmp;
493     char temperatures_current[] = _CURRENTSTATETEMP;
494     remove(temperatures_current_temp);
495     sprintf(help, "Current temperatures:");
496     write_log(temperatures_current_temp,help,1);
497 
498 /*  for(i=0; i<3; i++){
499         if(start_convert_all_temp(0))
500             break;
501     }*/
502 
```

```
503         for (i = 0 ; i < sensorsingle_list.max ; i++){
504             if (debug >= 1)
505                 printf("Reading Single Sensor %u\n", i);
506             //read_temp(&sensorsingle_list, i, portnum, &ctemp);
507             for(j=0;j<10;j++){
508                 ctemp = -1;
509                 read_temp_no_convert(&sensorsingle_list, i, portnum, &ctemp)
                    ;
510                 if(ctemp > 0)
511                     break;
512             }
513             if ((ctemp < low_limit[i]) || (ctemp > high_limit[i])){ //Out of
                range
514                 sprintf(help,"Temperature Warning for Sensorposition >%s< 
                    Current temperature of %3.2f [Celsius] is out range (%d 
                    to %d )",singleSensorPos[i],ctemp,low_limit[i],high_limit
                    [i]);
515                 if (debug >= 1)
516                     printf("%s\n",help);
517                 write_log(_TEMPWARNLOG,help,1);
518                 sprintf(help,"Sensorposition >%s< Temperature is %3.2f [
                    Celsius] WARNING! Out of range (%d to %d )!",
                    singleSensorPos[i],ctemp,low_limit[i],high_limit[i]);
519             }else{//Normal Output
520                 sprintf(help,"Sensorposition >%s< Temperature is %3.2f [
                    Celsius]",singleSensorPos[i],ctemp);
521             }
522 
523             if (debug >= 1)
524                 printf("%s\n",help);
525             write_log(temperatures_current_temp,help,0);
526             if(verbose)
527                 write_log(_TEMPLOG,help,1);
528             if(ctemp > 0){
529                 mean_singletemp[i] += ctemp;
530                 mean_singlenum[i]++;
531             }
532         }
533 
534         sprintf(help,"Sensorposition >Mean cabinet temperature from %2.0f 
            valve matrix sensors < Temperature is %3.2f [Celsius] RMS: %3.2f 
            [Celsius]",
535             tempnum, meantemp, tempRMS);
536         write_log(temperatures_current_temp,help,0);
537 
538 
539         if (debug >= 1)
540             printf("Copying temperature status\n");
541 
542         sprintf(help, "cp -f \"%s\" \"%s\"", temperatures_current_temp,
            temperatures_current);
543         system(help);
544 
545         return 1;
546     }
547 
548     void log_mean_temps(void)
549     {
```

```
550         int i;
551         char help[500];
552 
553         for(i=0; i<48; i++){
554             mean_opentemp[i] /= mean_opennum[i];
555             mean_closetemp[i] /= mean_closenum[i];
556         }
557         for(i=0; i<MAXSINGLE; i++){
558             mean_singletemp[i] /= mean_singlenum[i];
559         }
560         mean_meantemp /= runs;
561 
562         sprintf(help,"Sensorposition > Mean cabinet temperature <
                Temperature is %3.2f [Celsius]", mean_meantemp);
563         write_log(_TEMPLOG,help,1);
564 
565         for (i = 0 ; i < sensorsingle_list.max ; i++){
566             if ((mean_singletemp[i] < low_limit[i]) || (mean_singletemp[i] >
                    high_limit[i])){ //Out of range
567                 sprintf(help,"Sensorposition > %s < Temperature is %3.2f [
                        Celsius] WARNING! Out of range ( %d to %d )!",
                        singleSensorPos[i],mean_singletemp[i],low_limit[i],
                        high_limit[i]);
568             }else{//Normal Output
569                 sprintf(help,"Sensorposition > %s < Temperature is %3.2f [
                        Celsius]",singleSensorPos[i],mean_singletemp[i]);
570             }
571 
572             write_log(_TEMPLOG,help,1);
573         }
574 
575 }
576 
577 int OpenUSBPort(char* serial_port, int portnum){
578         char temp[1024];
579 
580         if(debug>0)
581             printf("Acquiring USB Port\n");
582         if( !owAcquire( portnum, serial_port, temp ) ){
583             fprintf(stderr, "USB ERROR: %s\n", temp );
584             OWERROR_DUMP(stderr);
585             return 0;
586         }
587         return 1;
588 }
589 
590 int main (int argc, char **argv){
591         char serial_port[40];
592         char temp[1024];
593 
594         //Argumente verarbeiten
595         char c;
596         while ((c = getopt (argc, argv, "d:DT:v")) != -1){
597             switch (c){
598                 case 'd':
599                     sscanf(optarg, "%d", &debug);
600                     break;
601                 case 'T':
```

```
602                     sscanf(optarg, "%d", &sleeptime);
603                     if((sleeptime < 4) || (sleeptime > 10))
604                         sleeptime = 10;
605                     runs = 240 / sleeptime;
606                     sleeptime -= 4;
607                     break;
608                 case 'D':
609                     daemon(0,0);
610                     break;
611                 case 'v':
612                     verbose=1;
613                     break;
614                 case '?':
615                     if (optopt == 'd')
616                         fprintf (stderr, "Option -%c requires an argument.\n
                                ", optopt);
617                     else if (isprint (optopt))
618                         fprintf (stderr, "Unknown option `-%c'.\n", optopt);
619                     else
620                         fprintf (stderr, "Unknown option character `\\x%x'.\
                                n",optopt);
621                     return 1;
622                     break;
623                 default:
624                     break;
625             }
626         }
627
628
629         /* Formel für sensor in gasmatrix
630         sensor = (spalte-1)*14 + (Reihe-1)*2 + posaufzu
631         */
632
633         // clean up the struct variables
634         bzero( &sensor_list, sizeof( struct _roms ) );
635         bzero( &switch_list, sizeof( struct _roms ) );
636         bzero( &sensorsingle_list, sizeof( struct _roms ) );
637         // alloc mem for struct
638         if( ( sensor_list.roms = malloc( MAXTEMP * 8 ) ) == NULL ){
639                 fprintf( stderr, "Error reserving memory for %d sensors\n"
                        , MAXTEMP );
640                 return -1;
641         }
642         if( ( sensorsingle_list.roms = malloc( MAXSINGLE * 8 ) ) == NULL ){
643                 fprintf( stderr, "Error reserving memory for %d single 
                        sensors\n", MAXSINGLE );
644                 return -1;
645         }
646
647
648         if( ( switch_list.roms = malloc( MAXSWITCH * 8 ) ) == NULL ){
649                 fprintf( stderr, "Error reserving memory for %d switches\n
                        ", MAXSWITCH );
650                 return -1;
651         }
652
653         read_cfg();
654         // Ausgabe der Adressen wenn debug
```

```
655        if (debug > 0){
656            int ss, rr, j ;
657            for (ss = 1; ss <= 6; ss++){
658                printf("Spalte: %d\n",ss);
659                for ( rr = 1 ; rr <=7 ; rr++){
660
661                    for (j = 0 ; j <8 ; j++)
662                        printf("%02X",sensor_list.roms[8*((ss-1)*14 + (rr-1)
                               *2 +0) + j]);
663                    printf(", ");
664                }
665                printf("\n");
666                for ( rr = 1 ; rr <=7 ; rr++){
667                    for (j = 0 ; j <8 ; j++)
668                        printf("%02X",sensor_list.roms[8*((ss-1)*14 + (rr-1)
                               *2 +1) + j]);
669                    printf(", ");
670                }
671                printf("Spalte %d Ende\n\n ", ss);
672
673            }
674            printf("Ausgabe der Adressen fuer Valves komplett\n");
675            printf("Adressen Einzelsensoren , Anzahl %d\n",anz_single_temp )
               ;
676            for(ss = 0 ; ss < anz_single_temp ; ss++){
677                printf("Sensorposition: %20s Adr: ",singleSensorPos[ss]);
678                    for (j = 0 ; j <8 ; j++)
679                        printf("%02X",sensorsingle_list.roms[8*ss + j]);
680                printf(" Non-warning range : %d - %d ",low_limit[ss],
                       high_limit[ss]);
681                printf("\n");
682            }
683
684            for(ss = 0 ; ss < INSTALLEDSWITCH ; ss++){
685                printf("Switch %d ; Spalte %d ; Kanal %d ; Adr ",ss, ss+1,
                       switchkanal_fuer_spalte[ss] );
686                    for (j = 0 ; j <8 ; j++)
687                        printf("%02X",switch_list.roms[8*ss + j]);
688                printf("\n");
689
690            }
691        }
692        // Initializierung Addressen Ende
693
694        // Open USB Port
695
696        strcpy( serial_port, "USB" );
697        int portnum = 0;
698        OpenUSBPort(serial_port, portnum);
699
700    // --- USB Port(s) opened
701
702    // Hauptschleife für normale Überwachung
703        int main_loop = 1;
704        int run = 0;
705        reset_min_max(); //reset min and max temperatures
706        if (main_loop == 1){
707            while (loop ==1){
```

```
708             int spalte, i;
709
710             if(run == 0){ //first run in cycle => heat
711                 write_log(_TEMPLOG,"Starting heating cycle.",1);
712                 for ( spalte = 1; spalte <= 6 ;spalte++){
713                     if(debug >= 1) printf("Heating %u\n", spalte);
714                     heizung(spalte);
715                 }
716             }
717
718             //Convert all temperatures
719             if(debug >= 1) printf("Converting temperatures.\n");
720             for(i=0; i<10; i++){
721                 if(start_convert_all_temp(0)){
722                     if(verbose)
723                         write_log(_TEMPLOG,"Converted temperatures.",1);
724                     break;
725                 }
726             }
727             if(i==10) write_log(_TEMPLOG,"Failed to convert temperatures
                    .",1);
728
729             //Read out matrix temperatures
730             for ( spalte = 1; spalte <= 6 ;spalte++){
731                 read_spalte( &sensor_list, spalte, portnum );
732             }
733
734             calc_mean_cabinet_temperature();
735
736             //Check single sensors
737             check_single(0);
738
739             //Reset USB
740             owRelease(0, temp);
741             for(i=0; i<10; i++){
742                 if(OpenUSBPort(serial_port, 0))
743                     break;
744                 sprintf(temp, "Could not acquire USB port.");
745                 write_log(_USBLOG,temp,1);
746                 sprintf(temp, "Could not acquire USB port.\n");
747                 fprintf(stderr, temp);
748                 if(i==9){
749                     sprintf(temp, "Quitting.");
750                     write_log(_USBLOG,temp,1);
751                     sprintf(temp, "Quitting.\n");
752                     fprintf(stderr, temp);
753                     loop = 0;
754                 }else{
755                     msDelay(5*1000);
756                 }
757             }
758
759
760             if(++run >= runs){
761                 for ( spalte = 1; spalte <= 6 ;spalte++){
762                     check_spalte( &sensor_list, spalte, portnum );
763                 }
764
```

```
765                     // Compare Read back to config values !
766                     check_cfg_cur(); // Compare the 2 array with valve
                            positions
767
768                     //log mean temperatures if current temperatures are not
                            being logged
769                     if(verbose == 0)
770                         log_mean_temps();
771
772                     reset_min_max(); //reset min and max temperatures
773                     run = 0;
774                 }
775
776                 if(debug >= 1) printf("Next run in cycle: %u/%u\n", run+1,
                        runs);
777
778                 if(debug >= 1) printf("Sleep %u\n", sleeptime);
779                 sleep(sleeptime);
780
781                 read_cfg();
782             } // while loop ==1
783         }
784
785 // CleanUp
786         printf("Clean up sensor_list\n");
787         if( sensor_list.roms != NULL )
788             free( sensor_list.roms );
789         printf("Clean up switch_list\n");
790         if( switch_list.roms != NULL )
791             free( switch_list.roms );
792         printf("Clean up sensorsingle_list\n");
793         if( sensorsingle_list.roms != NULL )
794             free( sensorsingle_list.roms );
795
796         owRelease(0, temp );
797 //      owRelease(1, temp );
798         return 0;
799 }
```

B.3 FLOWBUS

flowbus.h

```
1  #ifndef _FLOWBUS
2  #define _FLOWBUS
3
4
5  #define fbConfigFile "/home/messung/flowbus/fbConfig.txt"
6  #define fbLogFile "/home/vdc/data/gas/log/flowbusLog.txt"
7  #define fbCurrentstate "/home/messung/currentstateFlowbus.txt"
8  #define fbTempstate "/home/messung/flowbus/currentstateFlowbus.txt"
9  #define SerialDevice "/dev/ttyS0"
10
11 typedef struct{
12     char pos[375];
13     int type;
14     int chan;
15     float min;
```

```
16      float max;
17  } entry_t;
18
19  typedef struct entry_list{
20      entry_t entry;
21      struct entry_list *next;
22  } entry_list_t;
23
24
25  #endif
```

flowbus.c

```
1   #include <stdio.h>
2   #include <fcntl.h>
3   #include <unistd.h>
4   #include <math.h>
5
6   #include "flowbus.h"
7   #include "../../parser/source/parser.h"
8
9   int loop = 1; //main loop flag
10  int debug = 0; //debug level
11  int Dflag = 0; //daemon flag
12  #define DBG if(debug > 0)
13  #define LOG(text) write_log(fbLogFile, text, 1)
14
15  struct{
16      float nominalflow;
17      float readnominalflow;
18      float actualflow;
19      float nominalpressure;
20      float readnominalpressure;
21      float actualpressure;
22  } Chamber[6];
23
24  #define flow2int(f) floor(((f) * 1600) + 0.5)
25  #define int2flow(i) ((i) / 1600.0)
26  #define pres2int(p) floor(((p) * 29.0909091) + 0.5)
27  #define int2pres(i) ((i) / 29.0909091)
28  #define maxdev(v) ((v) * 0.01)
29
30  //int cham2flownodeA[] = { 3, 4, 5, 6, 7, 8 };
31  //int cham2presnodeA[] = { 9, 10, 11, 12, 13, 14 };
32  #define cham2flownode(c) (2+c)
33  #define cham2presnode(c) (8+c)
34  #define isflownode(n) ((n >= 3) && (n <= 8))
35  #define ispresnode(n) ((n >= 9) && (n <= 14))
36  #define node2cham(n) (isflownode(n)?(n-2):(n-8))
37
38  FILE *input;
39  FILE *output;
40
41  int read_cfg()
42  {
43      // Commands:
44      // $SETFLOW,3,Chamber number,flow[l/h],pressure[mBar]
45
46      int anz=0;
```

```
47      command_t* liste;
48
49      anz = anzahl_commands_in_file(fbConfigFile);
50      DBG printf("Commands in config file: %d\n", anz);
51      liste = (command_t*)malloc(sizeof(command_t) * anz);
52      read_cfg_file(fbConfigFile, liste);
53
54      int i;
55      for(i = 0 ; i < anz ; i++){
56          int cham;
57          float flow, pres;
58          switch (liste[i].cmd_nr){
59              case 1:
60                  loop = 0;
61                  break;
62              case 601:
63                  sscanf(liste[i].parameter[0].value, "%u", &cham);
64                  sscanf(liste[i].parameter[1].value, "%f", &flow);
65                  flow = floor(flow * 100.0) / 100.0;
66                  sscanf(liste[i].parameter[2].value, "%f", &pres);
67                  pres = floor(pres);
68
69                  if(cham > 0 && cham < 7){
70                      DBG printf("Settings for Chamber %i:\tFlow=%1.2f\
                            tPressure=%4.2f\n", cham, flow, pres);
71                      Chamber[cham-1].nominalflow = flow;
72                      Chamber[cham-1].nominalpressure = pres;
73                  }else{
74                      fprintf(stderr, "Ignoring wrong chamber number: %i\n
                            ", cham);
75                  }
76                  break;
77          }
78      }
79
80      free(liste);
81      return 1;
82  }
83
84  int OpenSerialBus(void)
85  {
86      char cmd[512];
87
88      sprintf(cmd, "stty -F %s raw 38400", SerialDevice);
89      DBG printf("%s\n", cmd);
90      system(cmd);
91
92      DBG printf("Opening serial bus.\n");
93      input = fopen(SerialDevice, "r");
94      output = fopen(SerialDevice, "w");
95
96      if(input != NULL && output != NULL){
97          int fd = fileno(input);
98
99          int flags = fcntl(fd, F_GETFL, 0);
100         flags |= O_NONBLOCK;
101         fcntl(fd, F_SETFL, flags);
102
```

```
103         DBG printf("Done.\n");
104         return 1;
105     }
106     return 0;
107 }
108 
109 void CloseSerialBus(void)
110 {
111     DBG printf("Closing␣serial␣bus.\n");
112     if(input != NULL)
113         fclose(input);
114     if(output != NULL)
115         fclose(output);
116 }
117 
118 void GetReturnValues(int c, int f, int n)
119 {
120     int node=0, i=0xFF, val=0, cham=0, equals=0;
121     int read=0;
122     float fval;
123     char cmd[512];
124     char errstr[512];
125 
126     while((fgets(cmd, 512, input)) != NULL){
127         DBG printf("S<␣%s", cmd);
128         sscanf(cmd, ":06%02X02012%1X%04X", &node, &i, &val);
129         DBG printf("Node:␣%i\tIndex:␣%i\tValue:␣%i\n", node, i, val);
130 
131         if(node >= 3 && node <= 14){
132             cham = node2cham(node);
133 
134             if(c == cham && f == isflownode(node) && n == i-1)
135                 read = 1;
136 
137             float diff;
138 
139             if(isflownode(node)){
140                 fval = int2flow(val);
141 
142                 if(i == 0x01){ //actual value
143                     Chamber[cham-1].actualflow = fval;
144                     sprintf(cmd, "VDCp%u␣actual␣flow␣read␣as:␣%4.2f",
                            cham, fval);
145                     DBG printf("%s\n", cmd);
146                     LOG(cmd);
147                 }else if(i == 0x02){ //nominal value
148                     Chamber[cham-1].readnominalflow = fval;
149                     diff = fabs(fval - Chamber[cham-1].nominalflow);
150                     errstr[0] = 0;
151                     if(diff > maxdev(Chamber[cham-1].nominalflow)){
152                         sprintf(errstr, "WARNING!␣Config␣discrepancy:␣")
                                ;
153                     }
154                     sprintf(cmd, "%sVDCp%u␣nominal␣flow␣read␣as:␣%4.2f",
155                         errstr, cham, fval);
156                     DBG printf("%s\n", cmd);
157                     LOG(cmd);
158                 }
```

```
159                 }else{
160                     fval = int2pres(val);
161 
162                     if(i == 0x01){ //actual value
163                         Chamber[cham-1].actualpressure = fval;
164                         sprintf(cmd, "VDCp%u actual pressure read as: %4.2f"
                                , cham, fval);
165                         DBG printf("%s\n", cmd);
166                         LOG(cmd);
167                     }else if(i == 0x02){ //nominal value
168                         Chamber[cham-1].readnominalpressure = fval;
169                         diff = fabs(fval - Chamber[cham-1].nominalpressure);
170                         errstr[0] = 0;
171                         if(diff > maxdev(Chamber[cham-1].nominalpressure)){
172                             sprintf(errstr, "WARNING! Config discrepancy: ")
                                    ;
173                         }
174                         sprintf(cmd, "%sVDCp%u nominal pressure read as: 
                            %4.2f",
175                             errstr, cham, fval);
176                         DBG printf("%s\n", cmd);
177                         LOG(cmd);
178                     }
179                 }
180             }
181     }
182 
183     errstr[0] = 0;
184     if(read){
185         float diff;
186         if(n == 1){
187             fval = f?Chamber[c-1].readnominalflow:Chamber[c-1].
                    readnominalpressure;
188             diff = fabs(fval - (f?Chamber[c-1].nominalflow:Chamber[c-1].
                    nominalpressure));
189             if(diff > maxdev(f?Chamber[c-1].nominalflow:Chamber[c-1].
                    nominalpressure)){
190                 sprintf(errstr, " WARNING! Config discrepancy. Config: 
                    %4.2f",
191                     f?Chamber[c-1].nominalflow:Chamber[c-1].
                        nominalpressure);
192             }
193         }else{
194 
195             fval = f?Chamber[c-1].actualflow:Chamber[c-1].actualpressure
                    ;
196             diff = fabs(fval - (f?Chamber[c-1].nominalflow:Chamber[c-1].
                    nominalpressure));
197             if(diff > maxdev(f?Chamber[c-1].nominalflow:Chamber[c-1].
                    nominalpressure)){
198                 sprintf(errstr, " WARNING! Strong deviation from nominal
                     value.");
199             }
200         }
201         sprintf(cmd, "VDCp%u %s %s is %4.2f%s", c, n?"nominal":"actual",
                f?"flow":"pressure", fval,errstr);
202     }else{
```

```
203             sprintf(cmd, "WARNING! VDCp%u %s %s could not be read.", c, n?"
                    nominal":"actual", f?"flow":"pressure");
204         }
205         write_log(fbTempstate, cmd, 0);
206     }
207
208     void SetNominalValue(int node, int val){
209         char cmd[512];
210
211         sprintf(cmd, ":06%02X010121%04X\r\n", node, val);
212         DBG printf("S> %s", cmd);
213         fprintf(output,  cmd);
214
215         usleep(100000);
216         fgets(cmd, 512, input);
217         DBG printf("S< %s", cmd);
218
219     }
220
221     void GetNominalValue(int cham, int f){
222         char cmd[512];
223         int node;
224
225         node=f?cham2flownode(cham):cham2presnode(cham);
226
227         sprintf(cmd, ":06%02X0401220121\r\n", node);
228         DBG printf("S> %s", cmd);
229         fprintf(output,  cmd);
230
231         usleep(100000);
232         GetReturnValues(cham,f,1);
233
234     }
235
236     void GetActualValue(int cham, int f){
237         char cmd[512];
238         int node;
239
240         node=f?cham2flownode(cham):cham2presnode(cham);
241
242         sprintf(cmd, ":06%02X0401210120\r\n", node);
243         DBG printf("S> %s", cmd);
244         fprintf(output,  cmd);
245
246         usleep(100000);
247         GetReturnValues(cham,f,0);
248     }
249
250     void check_all(void)
251     {
252         int i;
253
254         CloseSerialBus();
255         for(i=0; i<10; i++){
256             if(OpenSerialBus())
257                 break;
258         }
259         for(i=1; i<=6; i++){
```

```
260             SetNominalValue(cham2flownode(i), flow2int(Chamber[i-1].
                    nominalflow));
261             SetNominalValue(cham2presnode(i), pres2int(Chamber[i-1].
                    nominalpressure));
262             GetNominalValue(i,1);
263             GetNominalValue(i,0);
264             GetActualValue(i,1);
265             GetActualValue(i,0);
266         }
267 }
268 
269 int main (int argc, char *argv[])
270 {
271     int c;
272     FILE *f;
273     char cmd[512];
274 
275     while((c = getopt(argc, argv, "d:D")) != -1){
276         switch(c){
277             case 'd':
278                 sscanf(optarg, "%d", &debug);
279                 break;
280             case 'D':
281                 Dflag = 1;
282                 break;
283             case '?':
284                 printf("Unknown argument or missing parameter.\n");
285                 return -1;
286         }
287     }
288 
289     if(Dflag == 1){
290         LOG("Starting as daemon.");
291         printf("Starting as daemon.\n");
292         daemon(0,0);
293     }
294 
295     read_cfg();
296 
297     printf("Starting main loop.\n");
298     LOG("Starting main loop.");
299     while(loop != 0){ //main loop
300         //prepare log file
301         f = fopen(fbTempstate, "w");
302         if(f)
303             fclose(f);
304         write_log(fbTempstate, "Current flowbus status", 1);
305 
306         check_all();
307 
308         sprintf(cmd,"cp -f \"%s\" \"%s\"", fbTempstate, fbCurrentstate);
309         DBG printf("%s\n", cmd);
310         system(cmd);
311 
312         DBG printf("Sleep 10\n");
313         sleep(10);
314         read_cfg();
315     }
```

```
316
317     //clean up
318     CloseSerialBus();
319
320     LOG("Stopped.");
321     printf("Stopped.\n");
322     return 0;
323 }
```

B.4 PBREADOUT

pbreadout.h

```
1  #ifndef _pbreadout
2  #define _pbreadout
3
4  #define pbrConfigFile "/home/messung/pbreadout/pbrConfig.txt"
5  #define pbrLogFile "/home/vdc/data/gas/log/pbreadoutLog.txt"
6  #define pbrCurrentstate "/home/messung/currentstatePressureBox.txt"
7  #define COMEDIDEVICE "/dev/comedi0"
8
9  typedef struct{
10     char pos[375];
11     int type;
12     int chan;
13     float min;
14     float max;
15 } entry_t;
16
17 typedef struct entry_list{
18     entry_t entry;
19     struct entry_list *next;
20 } entry_list_t;
21
22 #endif
```

pbreadout.c

```
1  #define _ISOC99_SOURCE //Needed for NAN
2
3  #include <stdio.h>
4  #include <unistd.h>
5  #include <math.h>
6  #include <comedilib.h>
7  #include "pbreadout.h"
8  #include "../../parser/source/parser.h"
9
10 int loop = 1; //main loop flag
11 int debug = 0; //debug level
12 int Dflag = 0; //daemon flag
13 #define DBG if(debug > 0)
14 #define LOG(text) write_log(pbrLogFile, text, 1)
15
16 comedi_t *device;
17
18 entry_list_t *elist = NULL;
19
20 void adde(char *pos, int type, int chan, float min, float max){
21     entry_list_t *newe;
```

```
22
23      newe = malloc(sizeof(entry_list_t));
24      strncpy((char*)&(newe->entry.pos), pos, 375);
25      newe->entry.type = type;
26      newe->entry.chan = chan;
27      newe->entry.min = min;
28      newe->entry.max = max;
29
30      newe->next = elist;
31      elist = newe;
32  }
33
34  void clearlist(void){
35      entry_list_t *temp;
36
37      while(elist != NULL){
38          temp = elist->next;
39          free(elist);
40          elist = temp;
41      }
42  }
43
44  int read_cfg()
45  {
46      // Commands:
47      // $PBSENSOR,5,Positionstring,Sensortype,Channelnumber,minval,maxval
48
49      int anz=0;
50      command_t* liste;
51
52      anz = anzahl_commands_in_file(pbrConfigFile);
53      DBG printf("Commands in config file: %d\n", anz);
54      liste = (command_t*)malloc(sizeof(command_t) * anz);
55      read_cfg_file(pbrConfigFile, liste);
56
57      int i;
58      clearlist();
59      for(i = 0 ; i < anz ; i++){
60          int type, chan;
61          float min, max;
62          switch (liste[i].cmd_nr){
63              case 1:
64                  loop = 0;
65                  break;
66              case 501:
67                  sscanf(liste[i].parameter[1].value, "%u", &type);
68                  sscanf(liste[i].parameter[2].value, "%u", &chan);
69                  sscanf(liste[i].parameter[3].value, "%f", &min);
70                  sscanf(liste[i].parameter[4].value, "%f", &max);
71                  DBG printf("Adding Sensor:\n\tPos: %s\n\tType: %u\n\
                        tChan: %u\n\tMin: %2.0f\n\tMax: %2.0f\n", liste[i].
                        parameter[0].filestring, type, chan, min, max);
72                  adde(liste[i].parameter[0].filestring, type, chan, min,
                        max);
73                  break;
74          }
75      }
76
```

```
77      free(liste);
78      return 1;
79  }
80
81  float read_voltage(entry_list_t *sensor)
82  {
83      lsampl_t data;
84      int ret, i;
85      comedi_range * range_info;
86      lsampl_t maxdata;
87      double physical_value, mean = 0, meansquare = 0;
88
89      device = comedi_open(COMEDIDEVICE);
90      if(!device){
91          comedi_perror(COMEDIDEVICE);
92          exit(-1);
93      }
94
95      DBG{
96          printf("measuring device=%s subdevice=%d channel=%d range=%d
                analog reference=%d\n",
97              COMEDIDEVICE, 0, sensor->entry.chan, 1, 0);
98      }
99
100     for(i=0; i<1000; i++){
101         ret = comedi_data_read(device, 0, sensor->entry.chan, 1, 0, &
                data);
102         if(ret < 0){
103             comedi_perror(COMEDIDEVICE);
104             exit(-1);
105         }
106
107         if(1) {
108             comedi_set_global_oor_behavior(COMEDI_OOR_NAN);
109             range_info = comedi_get_range(device, 0, sensor->entry.chan,
                    1);
110             maxdata = comedi_get_maxdata(device, 0, sensor->entry.chan);
111             DBG {
112                 printf("[0,%d] -> [%g,%g]\n", maxdata,
113                     range_info->min, range_info->max);
114             }
115             physical_value = comedi_to_phys(data, range_info, maxdata);
116             if(isnan(physical_value)) {
117                 DBG printf("Out of range [%g,%g]",
118                     range_info->min, range_info->max);
119             } else {
120                 DBG printf("%g", physical_value);
121                 switch(range_info->unit) {
122                     case UNIT_volt: DBG printf(" V"); break;
123                     case UNIT_mA: DBG printf(" mA"); break;
124                     case UNIT_none: break;
125                     default: DBG printf(" (unknown unit %d)",
126                         range_info->unit);
127                 }
128                 DBG {
129                     printf(" (%lu raw units)", (unsigned long)data);
130                 }
131             }
```

```
132         } else {
133             DBG printf("%lu", (unsigned long)data);
134         }
135         DBG printf("\n");
136 
137         mean += physical_value;
138         meansquare += physical_value * physical_value;
139     }
140 
141     comedi_close(device);
142 
143     mean /= 1000;
144     meansquare /= 1000;
145 
146     return mean;
147 }
148 
149 float U2P(int type, float U)
150 {
151     float P;
152 
153     switch(type){
154         case 0:
155             P = (U/5.0 - 0.04) / 0.09 * 10.0;
156             break;
157         case 1:
158             P = (U/5.0 - 0.04) / 0.018 * 10.0;
159             break;
160         case 2:
161             P = (U/5.0 + 0.095) / 0.009 * 10.0;
162             break;
163         default:
164             P=NAN;
165             break;
166     }
167     return P;
168 }
169 
170 void check_sensor(entry_list_t *sensor, FILE *csf)
171 {
172     float U, P;
173     char logstring[512];
174 
175     U = read_voltage(sensor);
176     P = U2P(sensor->entry.type, U);
177     if(isnan(P)){
178         sprintf(logstring, "Pressure at >%s< is NAN (Read voltage was %1.2f V.) WARNING!", sensor->entry.pos, U);
179     } else if(P < sensor->entry.min){
180         sprintf(logstring, "Pressure at >%s< is %2.1f mBar. WARNING! Out of Range (%2.1f to %2.1f mBar)!", sensor->entry.pos, P, sensor->entry.min, sensor->entry.max);
181     } else if(P > sensor->entry.max){
182         sprintf(logstring, "Pressure at >%s< is %2.1f mBar. WARNING! Out of Range (%2.1f to %2.1f mBar)!", sensor->entry.pos, P, sensor->entry.min, sensor->entry.max);
183     } else {
```

```
184         sprintf(logstring, "Pressure at > %s < is %2.1f mBar.", sensor->
               entry.pos, P);
185     }
186     DBG printf("%s\n", logstring);
187     write_data(pbrCurrentstate, logstring, 1);
188     LOG(logstring);
189 }
190
191 void check_all( void )
192 {
193     entry_list_t * cur;
194     FILE *f;
195
196     f = fopen(pbrCurrentstate, "w");
197     if(f != NULL) fclose(f);
198     write_log(pbrCurrentstate, "Current pressure values:", 1);
199
200     cur = elist;
201     while(cur != NULL){
202         check_sensor(cur,f);
203         cur = cur->next;
204     }
205 }
206
207 int main (int argc, char *argv[])
208 {
209     int c;
210     while((c = getopt(argc, argv, "d:D")) != -1){
211         switch(c){
212             case 'd':
213                 sscanf(optarg, "%d", &debug);
214                 break;
215             case 'D':
216                 Dflag = 1;
217                 break;
218             case '?':
219                 printf("Unknown argument or missing parameter.\n");
220                 return -1;
221         }
222     }
223
224     if(Dflag == 1){
225         LOG("Starting as daemon.");
226         printf("Starting as daemon.\n");
227         daemon(0,0);
228     }
229
230     LOG("Starting main loop.");
231     printf("Starting main loop.");
232     while(loop != 0){ //main loop
233         read_cfg();
234         check_all();
235         sleep(10);
236     }
237
238     //clean up
239     clearlist();
240
```

```
241     LOG("Stopped.");
242     printf("Stopped.");
243     return 0;
244 }
```