Weiterentwicklung der Online-Datenanalyse sowie der Ladungs- und Stromauslese am LINTOTT-Spektrometer

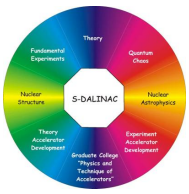
Improvement of the online data analysis, including charge and current readout at the LINTOTT spectrometer

Bachelor-Thesis von Sergej Bassauer

August 2012

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Physik
Institut für Kernphysik



Gefördert durch die DFG im Rahmen des SFB 634.

Weiterentwicklung der Online-Datenanalyse sowie der Ladungs- und Stromauslese am LINTOTT-Spektrometer
Improvement of the online data analysis, including charge and current readout at the LINTOTT spectrometer

Vorgelegte Bachelor-Thesis von Sergej Bassauer

1. Gutachten: Prof. Dr. P. von Neumann-Cosel
2. Gutachten: Andreas Krugmann, M. Sc.

Tag der Einreichung:

Zusammenfassung

Ziel dieser Bachelorarbeit war es, die komplizierte Steuerung des LINTOTT-Spektrometers sowie die Auswertung der gemessenen Spektren zu vereinfachen. Hierzu wurde ein bereits existierendes Programm namens HDTV verwendet, in das dann zusätzliche Funktionen implementiert wurden, bzw. die Initialisierung des Programms auf das Spektrometer und die gemessenen Spektren optimiert wurde. Ein weiteres Ziel war es die Stromauslese und die damit verbundene Ladungsmessung zu automatisieren und in den Ausleseprozess des Spektrometers zu integrieren. Dies wurde mit der QM07-Elektronik, die hier an der TU Darmstadt am Institut für Kernphysik entwickelt wurde, realisiert. In einer Teststrahlzeit im Dezember 2011 wurde das hier beschriebene Auslesesystem erfolgreich getestet und ist seitdem in Betrieb.

Inhaltsverzeichnis

1 Einleitung

Am supraleitenden Darmstädter Elektronenlinearbeschleuniger (S-DALINAC) am Institut für Kernphysik der TU Darmstadt werden Untersuchungen zur Struktur von Kernen mit Hilfe von inelastischer Elektronenstreuung gemacht. Seit 1980 konzipiert und seit 1991 betrieben wird der S-DALINAC im Rahmen von Bachelor-, Master- und Doktorarbeiten ständig weiter entwickelt. In Abb. 1.1 ist der Beschleuniger mit den verschiedenen Experimentierplätzen abgebildet.

Zur Erzeugung der Elektronen wird eine thermionische Elektronenkanone verwendet. Die Elektronen werden dann elektrostatisch auf eine Energie von 250 keV beschleunigt [2]. Alternativ dazu wurde vor kurzem eine Elektronenquelle entwickelt, die spinpolarisierte Elektronen mit Hilfe eines Titan-Saphir-Lasersystems erzeugt [3]. In der nachfolgenden Chopper-Prebuncher-Sektion wird dem Elektronenstrahl eine 3 GHz Zeitstruktur aufgeprägt und der Strahl kann im sogenannten Injektor-Beschleuniger mit Hilfe von 20-zelligen Niobstrukturen auf 10 MeV beschleunigt werden. Am Injektormessplatz (①) können nun Bremsstrahlungsexperimente mit Elektronenströmen von bis zu 60 μA durchgeführt werden. Der Elektronenstrahl kann aber auch in den Hauptbeschleuniger umgelenkt werden, wo er einen Energiegewinn von bis zu 40 MeV pro Rezirkulation erhält. Der nun hochenergetische Elektronenstrahl mit einer Energie von bis zu 130 MeV kann nun an einem der beiden Elektronenstreuspektrometer (④, ⑤) für Kernstrukturexperimente verwendet werden [2]. Am LINTOTT-Spektrometer können (e, e')-Experimente durchgeführt werden. Die Besonderheit des Spektrometers ist hierbei die sehr gute Energieauflösung im sogenannten Energieverlustmodus. Leider deckt das Spektrometer nur einen kleinen Raumwinkel ab. Ein weiterer Nachteil ist die geringe Impulsakzeptanz. Das QCLAM-Spektrometer ermöglicht es Koinzidenzexperimente und Experimente unter Streuwinkeln von 180° durchzuführen. Auch hat das Spektrometer größere Winkel- und Impulsakzeptanz. Da im Rahmen dieser Bachelorarbeit nur am LINTOTT-Spektrometer gearbeitet wurde, wird auf das QCLAM nicht weiter eingegangen.

1.1 Das LINTOTT-Spektrometer

Das Spektrometer ist ein Dipolmagnetspektrometer der Firma LINTOTT. Die Elektronen werden um einen Winkel von 169° in die Fokalebene abgelenkt, da dieser Winkel besonders gute Abbildungseigenschaften besitzt [4]. Messungen lassen sich bei Winkeln zwischen 33° und 165° in Schritten von 12° durchführen, somit kann man bei einer Elektronenenergie, Messungen unter verschiedenen Impulsüberträgen durchführen. Der Fokalebenendetektor besteht aus 4 Modulen mit je 96 Siliziumstreifenzählern (2 cm × 650 μm × 500 μm) und einem Tscherenkowdetektor als Triggerdetektor [5]. Es ergibt sich somit eine effektive Detektorlänge von 24,96 cm. Bedingt

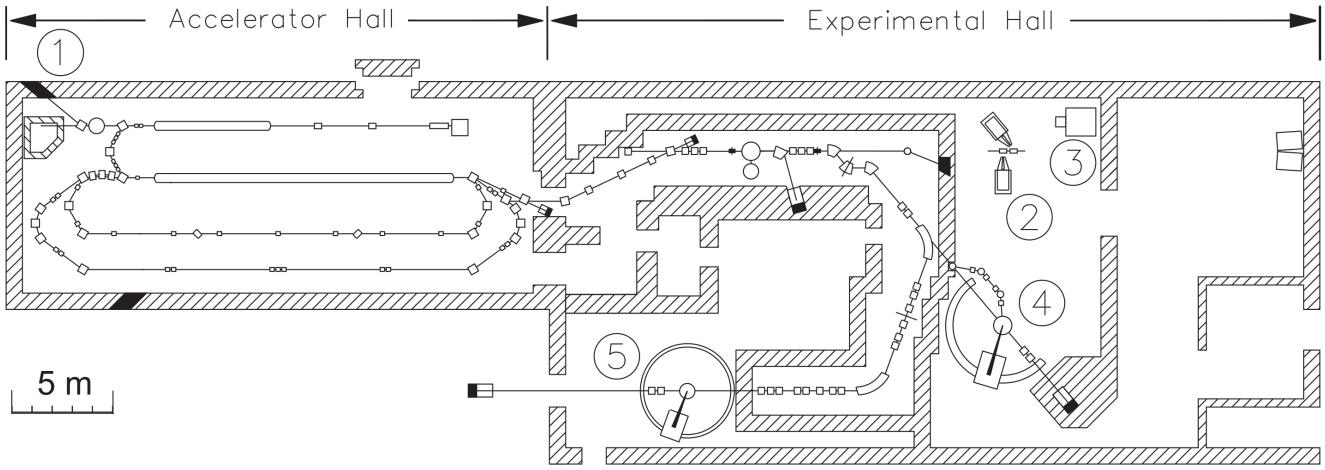


Abbildung 1.1: Der S-DALINAC mit seinen Experimentierplätzen. ① Bremsstrahlungsexperimente. ② Polarisierbarkeit des Nukleons. ③ $(\gamma,\gamma'x)$-Experimente ④ Koinzidenzexperimente am QCLAM-Spektrometer. ⑤ (e,e')-Experimente am LINTOTT-Spektrometer. [1]

Tabelle 1.1: Technische Parameter des Spektrometers [6].

Elektronenenergiebereich	(20-120) MeV
Streuwinkelbereich	33°-165° (12° Schritte)
Radius der zentralen Trajektorie	1 m
Neigung der Fokalebene	35°
Dispersion	3,76 cm/1%
Impulsakzeptanz	±2,1%
Raumwinkelakzeptanz	6 msr
Auflösung (FWHM)	0,015%
Magnetfeldstärke	(0,6-4,0) kG

durch die modulare Bauweise des Fokalebenendetektors gibt es drei Lücken zwischen den Modulen mit einer Breite von 7 mm bzw. 10,5 Streifen. In diesen Bereichen können keine Teilchen detektiert werden. Des Weiteren lässt sich das Spektrometer auf zwei verschiedene Weisen betreiben, im dispersiven Modus oder im Energieverlustmodus [5, 6]. Die wichtigsten technischen Parameter sind in Tabelle 1.1 zusammengefasst.

1.1.1 Dispersiver Modus

Im dispersiven Modus werden die gestreuten Elektronen, je nach deren Impuls, im Magnetfeld auf unterschiedlichen Bahnen abgelenkt und auf die Fokalebene des Spektrometers projiziert. Diese Situation ist in Abb. 1.2 auf der linken Seite dargestellt. Das Magnetspektrometer wandelt die Impulsinformation in eine Ortsinformation um, welche sich viel einfacher mit hoher Genauigkeit bestimmen lässt. Aufgrund unterschiedlich langer Flugbahnen werden Elektronen

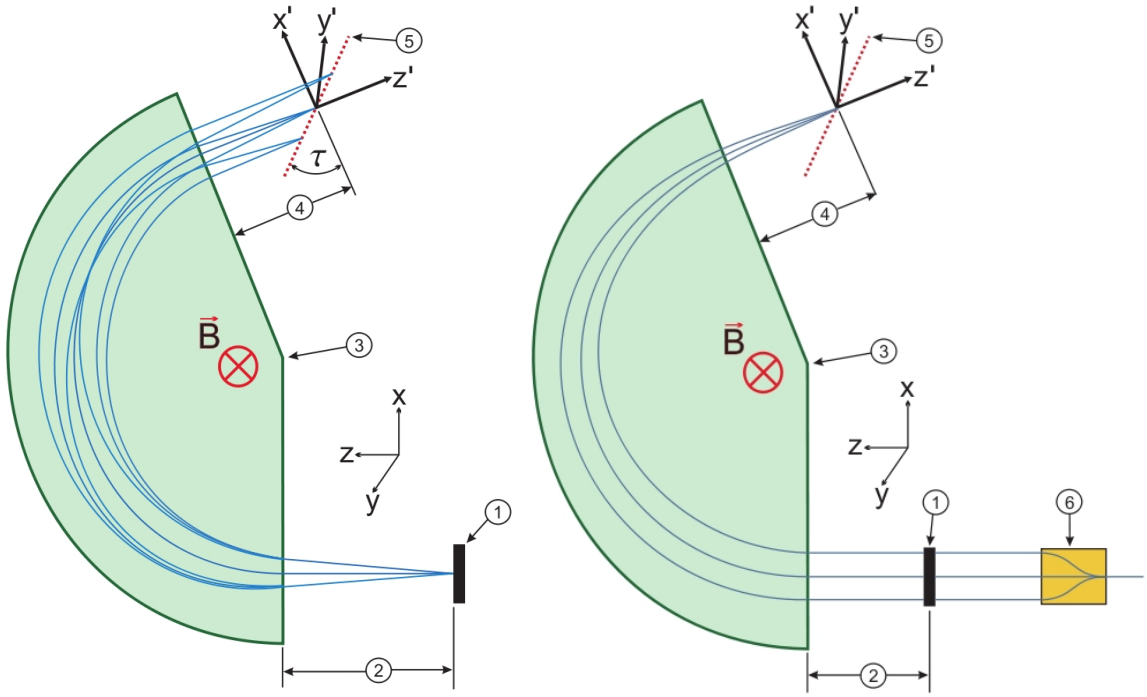


Abbildung 1.2: Links: Dispersiver Modus. Die gestreuten Elektronen werden impulsselektiv abgelenkt.① Target. ② Gegenstandsweite. ③ Dipolmagnet. ④ Bildweite. ⑤ Fokalebene [5].
Rechts: Energieverlustmodus. Der Auftreffort der Elektronen in der Fokalebene hängt nur vom Energieverlust ab, den die Elektronen im Target erfahren.① Target. ② Gegenstandsweite. ③ Dipolmagnet. ④ Bildweite. ⑤ Fokalebene. ⑥ Strahlführungssystem [5].

mit gleichem Impuls und unterschiedlichen Eintrittswinkeln in der Fokalebene auf den selben Punkt projiziert, sodass das Spektrometer radial fokussierend wirkt [5].

1.1.2 Energieverlustmodus

Im Energieverlustmodus wird das Spektrometer so betrieben, dass das gesamte System, bestehend aus Strahlführung und Spektrometer, dispersionsfrei wirkt. Die Strahlführung bildet hierbei den Elektronenstrahl dispersiv als schmalen, vertikalen Streifen auf das Target ab. Dies erreicht man durch die Einstellung der Dispersion der Strahlführung so, dass die feste Dispersion des Spektrometers gerade ausgeglichen wird. Somit werden Elektronen mit einem speziellen Energieverlust $\triangle E$ mit einem entsprechenden Versatz $\triangle x$ von der Sollbahn auf die Fokalebene projiziert. Elektronen ohne Energieverlust hingegen, werden in einem einzigen Punkt auf der Fokalebene abgebildet. Dadurch sind Messungen von der Energieunschärfe des Elektronenstrahls unabhängig, was zu einer sehr guten Auflösung führt [5]. In Abb. 1.2 auf der rechten Seite wird das Prinzip veranschaulicht.

2 Grundlagen zur Online-Datenanalyse von Elektronenstreuexperimenten

Da am LINTOTT-Spektrometer (e, e')-Experimente durchgeführt werden, sollen hier einige der wichtigsten Formeln für diese Art der Experimente zusammengefasst werden. Bei der Elektronenstreuung werden Elektronen mit der Anfangsenergie E_i auf einen ruhenden Targetkern geschossen. An diesem werden sie dann elastisch oder inelastisch gestreut. Vernachlässigt man die Rückstoßenergie, beträgt die Anregungsenergie des Kerns $E_a = E_i - E_f$, mit der Anfangsenergie E_i und der Endenergie E_f des Elektrons. Der Impulsübertrag ist gegeben durch

$$q = \frac{1}{\hbar c}\sqrt{\frac{4E_iE_f\sin^2\frac{\theta}{2} + E_a^2}{1 + \frac{2E_i}{Mc^2}\sin^2\frac{\theta}{2}}}, \tag{2.1}$$

wobei M die Massenzahl des Targetkerns, c die Lichtgeschwindigkeit, $\hbar$ das reduzierte Plancksche Wirkungsquantum und θ der Streuwinkel ist. Unter Berücksichtigung des Rückstoßes beträgt die Energie der gestreuten Elektronen

$$E_f = \frac{E_i - E_a(1 + \frac{E_a}{2Mc^2})}{1 + \frac{2E_i}{Mc^2}\sin^2\frac{\theta}{2}}. \tag{2.2}$$

Die Rückstoßenergie kann durch

$$E_R = \sqrt{(Mc^2 + E_a)^2 + (q\hbar c)^2} - (Mc^2 + E_a) \tag{2.3}$$

berechnet werden bzw. durch

$$E_R = \frac{(q\hbar c)^2}{2(Mc^2 + E_a)}, \tag{2.4}$$

falls $q\hbar c << Mc^2$ [6].

Der experimentelle differentielle Wirkungsquerschnitt ist gegeben durch

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega} = A_{exp} \cdot \frac{1}{\Delta\Omega} \cdot \frac{e}{Q} \cdot \frac{M}{t_{eff}N_A}. \tag{2.5}$$

Hierbei ist A_{exp} die Fläche unter dem Peak, $\Delta\Omega$ der Raumwinkel des Spektrometers, Q die gesammelte Ladung, M die molare Masse des Isotops, N_A die Avogadrokonstante, t_{eff} die effektive

Flächenbelegung des Targets in [g/cm^2] [1]. Stellt man diese Formel etwas um, erhält man das Verhältnis der Fläche unter dem Peak zur gesammelten Ladung, welches konstant ist.

$$\frac{A_{exp}}{Q} = \frac{\mathrm{d}\sigma}{\mathrm{d}\Omega} \cdot \frac{\Delta\Omega}{e} \cdot \frac{t_{eff} N_A}{M} = \text{konst.} \tag{2.6}$$

In der Online-Datenalyse wurde jedoch nicht die Fläche unter dem Peak genommen, sondern die Gesamtfläche des Spektrums, da diese sehr viel einfacher zu bestimmen ist. Hierbei muss jedoch die elastische Linie sehr viel dominanter gegenüber den restlichen Peaks bzw. dem Untergrund sein. Die Formel (2.6) ist dann nur noch näherungsweise konstant, stellt jedoch weiterhin eine gute und schnelle Möglichkeit die Korrektheit der Stromauslese zu überwachen. Diese Näherung gilt allerdings nur bei Anwesenheit der elastischen Linie im Spektrum und gilt nicht für extreme Rückstreuwinkel.

2.1 Analyse eines Peaks

Bei der Elektronenstreuung an Kernen entsteht ein kontinuierliches Energieverlustspektrum, die Bremsstrahlung. Aufgrund der Bremsstrahlung sind die Peaks im Streuspektrum nicht mehr Gaußförmig, sondern auf der einen Seite asymmetrisch. Die asymmetrische Seite wird approximativ durch eine an die Gaußfunktion angeschlossene Hyperbelfunktion beschrieben. Die gesamte Funktion sieht dann folgendermaßen aus [7]

$$f(x) = y_0 \begin{cases} \exp(-\ln 2 \cdot (x - x_0)^2 / \Delta x_1^2) & x < x_0 \\ \exp(-\ln 2 \cdot (x - x_0)^2 / \Delta x_2^2) & x_0 < x \leq x_0 + \eta \Delta x_2 \\ A/(B + x - x_0)^\gamma & x > x_0 + \eta \Delta x_2 \end{cases} \tag{2.7}$$

Mit den folgenden Parametern [6]:

x_0	Energie im Peakmaximum
y_0	Zählrate im Peakmaximum
$\Delta x_{1,2}$	HWHM für $(x - x_0) < x_0$ bzw. $(x - x_0) > x_0$
η	Startpunkt des asymmetrischen Teils in Einheiten von Δx_2
γ	Exponent der Hyperbelfunktion

Die Parameter A und B lassen sich aus der Bedingung der stetigen Differenzierbarkeit im Anschlusspunkt $x_0 + \eta \Delta x_2$ bestimmen. Für die beiden Parameter erhält man dann

$$A = 2^{-\eta^2} (B + \Delta x_2 \eta)^\gamma \tag{2.8}$$

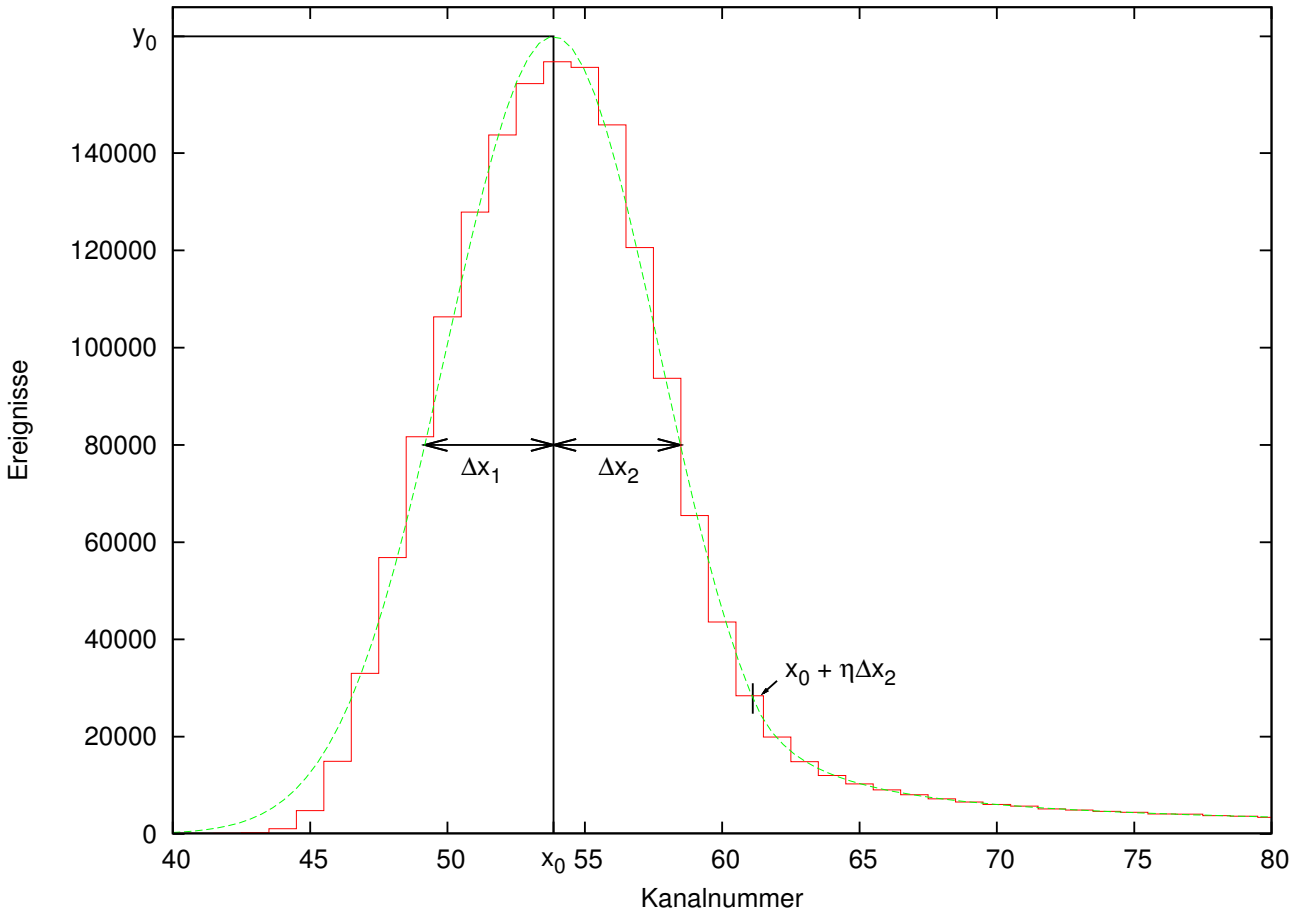


Abbildung 2.1: Ein Elektronenstreupeak mit der angepassten Funktion (2.7).

$$B = \frac{\Delta x_2 \gamma - 2\Delta x_2 \eta^2 \log 2}{2\eta \log 2} \tag{2.9}$$

Anschaulich ist die Anpassungsfunktion in Abb. 2.1 gezeigt.

3 Weiterentwicklung der Online-Datenanalyse

Seit der Inbetriebnahme der FPGA (Field Programmable Gate Arrays) basierten Auslesehardware [5] ist die Online-Datenanalyse noch in keinem ausgereiften Zustand. Das aktuelle Spektrum kann lediglich, wie in Abb. 3.1 als kleine Bilddatei in einem Internetbrowser angezeigt werden, eine genauere Online-Analyse der Streuspektren ist damit unmöglich.

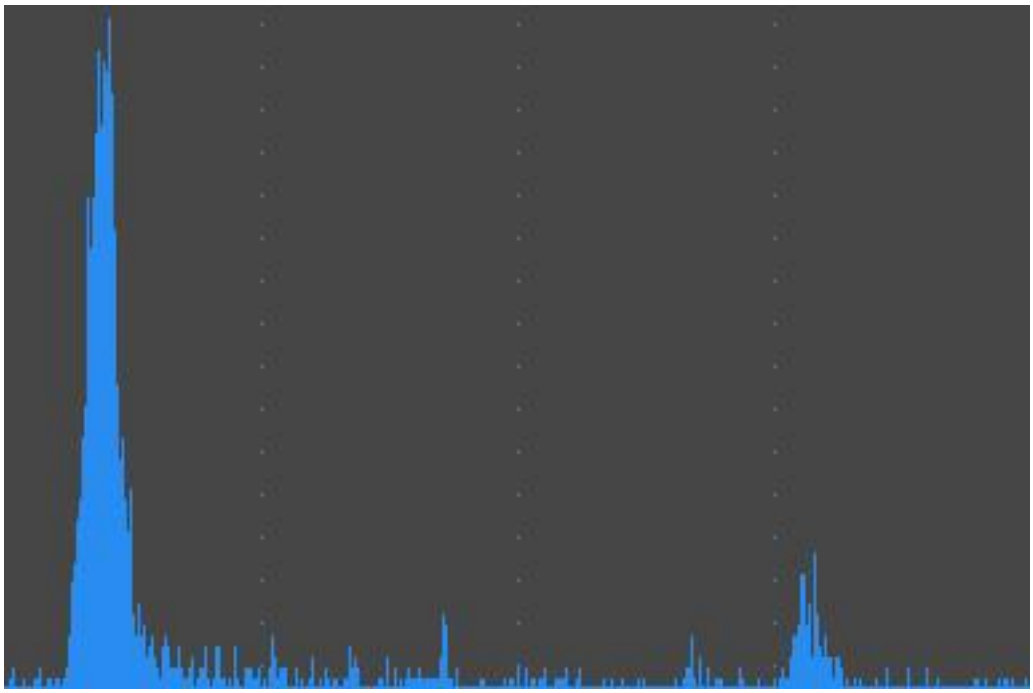


Abbildung 3.1: Bild eines Elektronenstreuspektrums in der alten Online-Datenanalyse.

3.1 Anforderungen

An die Weiterentwicklung der Online-Datenanalyse werden somit einige Anforderungen gestellt um die Auslese und Anzeige des Spektrums komfortabler zu gestalten. Das Spektrum sollte nach Möglichkeit sofort und ohne größere Umstände angezeigt werden. Auch sollte man das Spektrum vergrößern, bzw. verkleinern können sowie die Darstellungsart von linear zu logarithmisch wechseln können. Nützlich wären auch wichtige Manipulationsmöglichkeiten des Spektrums, wie z.B. Anpassungen von Funktionen an das Spektrum, addieren oder subtrahieren von mehreren Spektren und ähnliches. Das Spektrum sollte man auch jederzeit in verschiedenen Formaten abspeichern und ausdrucken können.

Ein weiterer wichtiger Aspekt ist die Stromauslese. Wichtige Informationen sind hierbei die geflossene Ladung während einer Messung sowie das Stromverhalten über die Zeit, welches zur

Strahldiagnose beitragen könnte. Hierzu gibt es bereits eine Stromauslese (Current-Digitiser). Diese ist jedoch sehr ungenau, da man beim Starten des Spektrometers am Computer gleichzeitig den Current-Digitiser im Messraum manuell starten muss. Dass beide Prozesse dabei tatsächlich gleichzeitig starten, ist nicht immer gewährleistet. Des Weiteren ist das Gerät bereits sehr alt, so dass nicht klar ist ob es noch ordnungsgemäß funktioniert. Aus diesen Gründen sollte die Stromauslese modernisiert und in den Ausleseprozess eingebunden werden.

Die regelmäßige Sicherung der Messungen ist ein weiterer Punkt, der z.B. durch automatisches Speichern des Spektrums nach bestimmten Zeitabständen erfüllt werden sollte.

3.2 Konzept

Um die im letzten Abschnitt genannten Anforderungen zu erfüllen, wurde beschlossen ein bereits vorhandenes Programm namens HDTV zu verwenden und entsprechend den gestellten Anforderungen zu erweitern. Das Programm erlaubt es das Spektrum auf verschiedenste Weise zu manipulieren, so dass dieses ein sehr guter Ausgangspunkt für die Weiterentwicklung der Online-Datenanalyse darstellt. Um in das Programm so wenig wie möglich einzugreifen wurde entschieden ein Plug-in zu schreiben, mit dem die Steuerung des Spektrometers direkt aus dem Programm möglich sein sollte. Für die Stromauslese bietet es sich an die QM07-Elektronik zu verwenden, die eine digitale Auslese ermöglicht und am Institut für Kernphysik bereits in anderen Bereichen erfolgreich verwendet wird.

4 Stromauslese und Ladungsmessung

Um die am Spektrometer gemessenen Spektren auswerten zu können, ist die Kenntnis der während des Experiments geflossenen Ladung sehr wichtig. Hierzu muss man den Strom am Faraday-Cup messen, damit die Spektren auf die gesammelte Ladung normiert werden können. Für die Strommessung wurde die QM07-Elektronik benutzt, die im Rahmen einer Dissertation am Institut für Kernphysik entwickelt wurde [8]. Der Grundaufbau besteht aus einem Einbaurahmen mit mehreren Einschubplätzen und einer QM07-Hauptplatine auf der Rückseite des Einbaurahmens. In Abb. 4.1 ist ein derartiger Aufbau zu sehen.

4.1 QM07-Einschübe

Auf der äußersten linken Seite in Abb. 4.1 befindet sich die Kontrolleinheit. Über diese Einheit wird der ganze Messaufbau eingeschaltet. Die Einheit muss mit 24 V versorgt werden. Alle anderen Einschübe werden über diese Einheit mit 5 V versorgt. Die Einheit besitzt zwei CAN-Ausgänge (**C**ontroller **A**rea **N**etwork), über die eine Verbindung zum Computer hergestellt werden kann [9].

Neben dem CAN-Interface befindet sich die IADC-Einheit. Diese Einheit ermöglicht es Ströme im Bereich von ±100 µA zu messen. Zur Auslese besitzt jede IADC-Einheit zwei Kanäle mit jeweils einer Koaxialkabelbuchse [9].

Weiter rechts befinden sich zwei DS-Einheiten. Diese können benutzt werden um Widerstandsverhältnisse zu messen. Sie besitzen je zwei Netzwerkausgänge [9]. Diese Einheiten fanden im Rahmen dieser Bachelorarbeit jedoch keine Verwendung.

Als nächstes sieht man die PU-Einheit. Die CAN-Packete werden von der Kontrolleinheit an die Prozessoreinheit über die Hauptplatine geschickt. Die PU übernimmt die digitale Signalver-

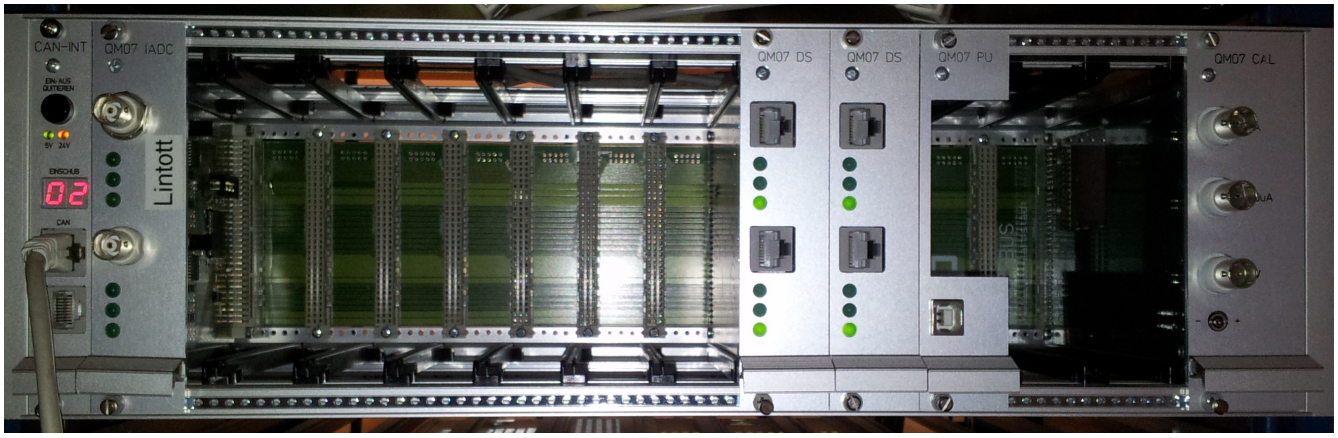


Abbildung 4.1: Einbaurahmen mit QM07-Einschüben.

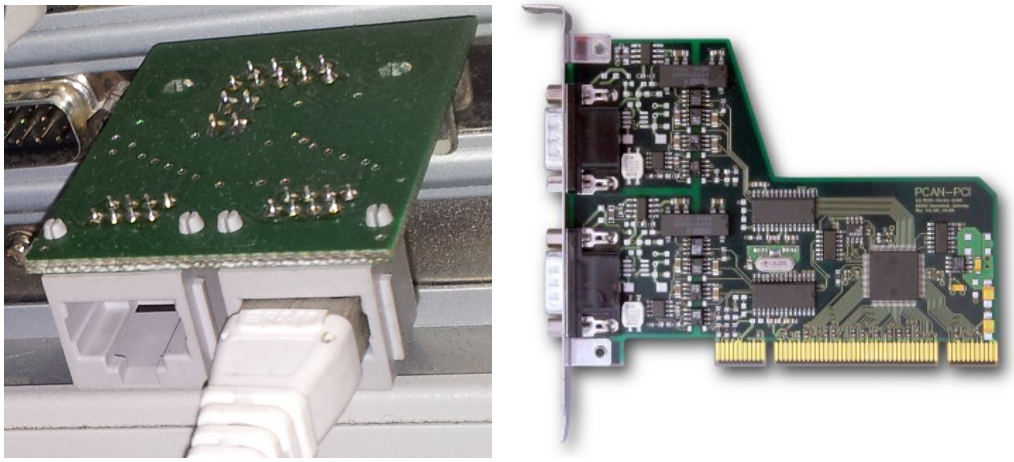


Abbildung 4.2: Links: Serielle CAN-Adapter. Rechts: PCAN-PCI-Karte [11].

arbeitung innerhalb des QM07. Über diese Einheit kann man, alternativ zu den CAN-Ausgängen der Kontrolleinheit, per USB-Kabel mit dem Computer eine Verbindung herstellen [8, 9]. Diese Möglichkeit wurde im Rahmen dieser Bachelorarbeit nur zu Testzwecken verwendet.

Ganz auf der rechten Seite befindet sich schließlich noch eine Kalibrierungseinheit. Diese ermöglicht es konstante Ströme von $\pm 1\,\mu A$ bzw. $\pm 100\,\mu A$ sowie Spannungen von $\pm 10\,V$ zu erzeugen und über Koaxialkabelbuchsen zur Verfügung zu stellen [9].

4.2 CAN-Bus und PEAK-Adapterkarte

Um die Daten aus dem QM07 auslesen zu können, wurde eine PCAN-PCI-Karte mit seriellen Eingängen der Firma PEAK-System Technik GmbH angeschafft. Mit Hilfe eines Adapters, der in der hauseigenen Elektronik-Werkstatt hergestellt wurde [10], mit einem seriellen und zwei CAN-Ausgängen kann eine Verbindung zwischen dem QM07 und dem Computer hergestellt werden. Die PCAN-PCI-Karte und der serielle CAN-Adapter sind in Abb. 4.2 abgebildet. Um die Verbindung zwischen dem QM07 und dem Computer nutzen zu können, wurden der CAN-Treiber sowie weitere Software installiert. Eine genaue Installationsanleitung findet sich im Anhang A.

4.3 Datenauslese

Nach der Installation der nötigen Software sind einige neue Befehle verfügbar über die man mit dem QM07 interagieren kann. Für die Datenauslese sind insbesondere zwei Befehle wichtig: *candump* und *cansend*. Beide werden im Terminal am Messrechner eingegeben.

Der Befehl *candump* erhält als Argument die CAN-Einheit, als Standard *can0* oder *can1*. Nach Eingabe des Befehls werden alle Daten, die gesendet oder empfangen werden, auf dem Bildschirm ausgegeben.

Mit dem Befehl *cansend* kann man selbst Daten senden oder anfordern. Auch hier muss als Argument die CAN-Einheit übergeben werden. Zusätzlich wird eine acht-stellige Hexadezimalzahl übergeben, der gefolgt von einer Raute eine max. acht-stellige Zahl angehängt werden kann. Für die Auslese des aktuellen Stroms sieht der Befehl dann folgendermaßen aus: *cansend can1 0608410F#00*. Hierbei haben die einzelnen Ziffern folgende Bedeutung:

- Die ersten beiden Ziffern geben an, ob der Befehl gesendet (06) oder empfangen (07) wird.
- Die nächsten drei Ziffern ergeben sich aus der Gerüstnummer und der Messkartenadresse.
- Die letzten drei Ziffern vor der Raute bilden schließlich den eigentlichen Befehl. Nach der Raute wird falls nötig die Kanalnummer des anzusprechenden Einschubs angegeben oder weitere Spezifikationen des Befehls. Alle Befehle sind in der Datei *cps_can.h* am Rechner *linux21* im Verzeicnis */home/experiment/Desktop/hdtvtest* beschrieben. Die wichtigsten Befehle werden im folgenden angegeben.
  - 10F#00 Abfragen des aktuellen Stroms.
  - 10F#FD Abfragen der Anzahl interner Messzyklen seit dem Einschalten des Gerätes.
  - 10F#FC Abfragen der vergangenen Zeit seit dem Einschalten des Gerätes.
  - 111#00 Abfragen des aufaddierten Stroms seit dem Einschalten des Gerätes.

Nach dem Senden des Befehls kann man die Antwort im *candump*-Fenster einsehen. Diese ist jedoch wieder hexadezimal kodiert [9].

4.4 Strommessung und Berechnung der gesammelten Ladung

Wie am Anfang des Kapitels erwähnt, kann man die gesammelte Ladung aus der Strommessung am Faraday-Cup berechnen. Man erhält allerdings alle Daten kodiert in Hexadezimalzahlen, weshalb diese erst in Stromwerte umgerechnet werden müssen. Den aktuellen Strom erhält man als eine 32-Bit Hexadezimalzahl, die als vorzeichenbehafteter Integer kodiert ist. Hierbei entsprechen 29 Bit dem Vollausschlag, der einem Strom von 101,6 µA entspricht [9]. Somit lässt sich der positive Strom mit

$$I = I_{dez} \cdot \frac{101,6 \times 10^{-6}\,\mathrm{A}}{2^{28} - 1} \tag{4.1}$$

berechnen [12], wobei I_{dez} die ausgelesene Zahl ist, die aus dem Hexadezimalen ins Dezimale umgewandelt wurde. Für den negativen Strom erhält man entsprechend

$$I = (2^{32} - I_{dez}) \cdot \frac{-101,6 \times 10^{-6}\,\mathrm{A}}{2^{28}}. \tag{4.2}$$

Zur Ladungsmessung wurde die Funktion zur Auslese des aufaddierten Stroms ausgenutzt. Diese liefert allerdings den Strom als eine 56-Bit Hexadezimalzahl, ebenfalls kodiert als vorzeichenbehafteter Integer. Die Umrechnung in Strom erhält man durch

$$I = I_{dez} \cdot 256 \cdot \frac{101,6 \times 10^{-6}\,\mathrm{A}}{2^{28}}. \tag{4.3}$$

Die Multiplikation mit 256 ist nötig um die Zahl auf 64 Bit aufzuwerten, so dass die positiven Zahlen 32-Bit besetzen und entsprechend auch die negativen. Durch diese künstliche Verschiebung der Bits ergibt sich allerdings das Problem, dass das letzte Byte unbestimmt ist. Hier können Werte zwischen *00* und *FF* auftreten. Umgerechnet in Strom folgt daraus eine maximale Unsicherheit von $\Delta I = 0,096515\,\mathrm{nA}$. Da am Spektrometer Ströme von einigen hundert nA bis einige µA gemessen werden, erscheint diese Unsicherheit akzeptabel. Um nun die gesammelte Ladung zu erhalten, muss man aus den aufaddierten Strömen am Anfang der Messung und am Ende der Messung die Differenz bilden und anschließend mit der Messdauer zwischen zwei Stromwerten multiplizieren. Es ergibt sich die folgende Formel

$$Q = (I_{Ende} - I_{Start}) \cdot \frac{T_{Ende} - T_{Start}}{N_{Ende} - N_{Start}}, \tag{4.4}$$

wobei die Ströme, wie gezeigt, mit Gleichung (4.3) umgerechnet werden. Die Messdauer zwischen zwei Stromwerten ergibt sich aus dem Verhältnis der Gesamtmessdauer T und der Anzahl der internen Messzyklen N.

Der mittlere Strom einer Messung berechnet sich einfach aus

$$I_{Mittel} = \frac{I_{Ende} - I_{Start}}{N_{Ende} - N_{Start}} \tag{4.5}$$

4.5 Messungenauigkeit bei der Ladungsmessung

Um sicher zu gehen, dass die Schwankungen bei der Stromauslese klein sind, wurden Messungen über eine Stunde lang bei 1 µA und 100 µA vorgenommen. Hierbei ergab sich eine mittlere Abweichung von 0,023 nA (0,023‰) bei 1 µA und 0,107 nA (0,001‰) bei 100 µA. Die beiden Messungen sind in Abb. 4.3 zu sehen. Um eine absolute Unsicherheit anzugeben wären weitere Messungen unter Variation des Stroms nötig, da diese offensichtlich vom Strom abhängt. Der Elektronenstrom am Beschleuniger variiert jedoch, wie schon erwähnt, zwischen einigen hundert nA und einigen µA. Nimmt man die im letzten Abschnitt berechnete Unsicher-

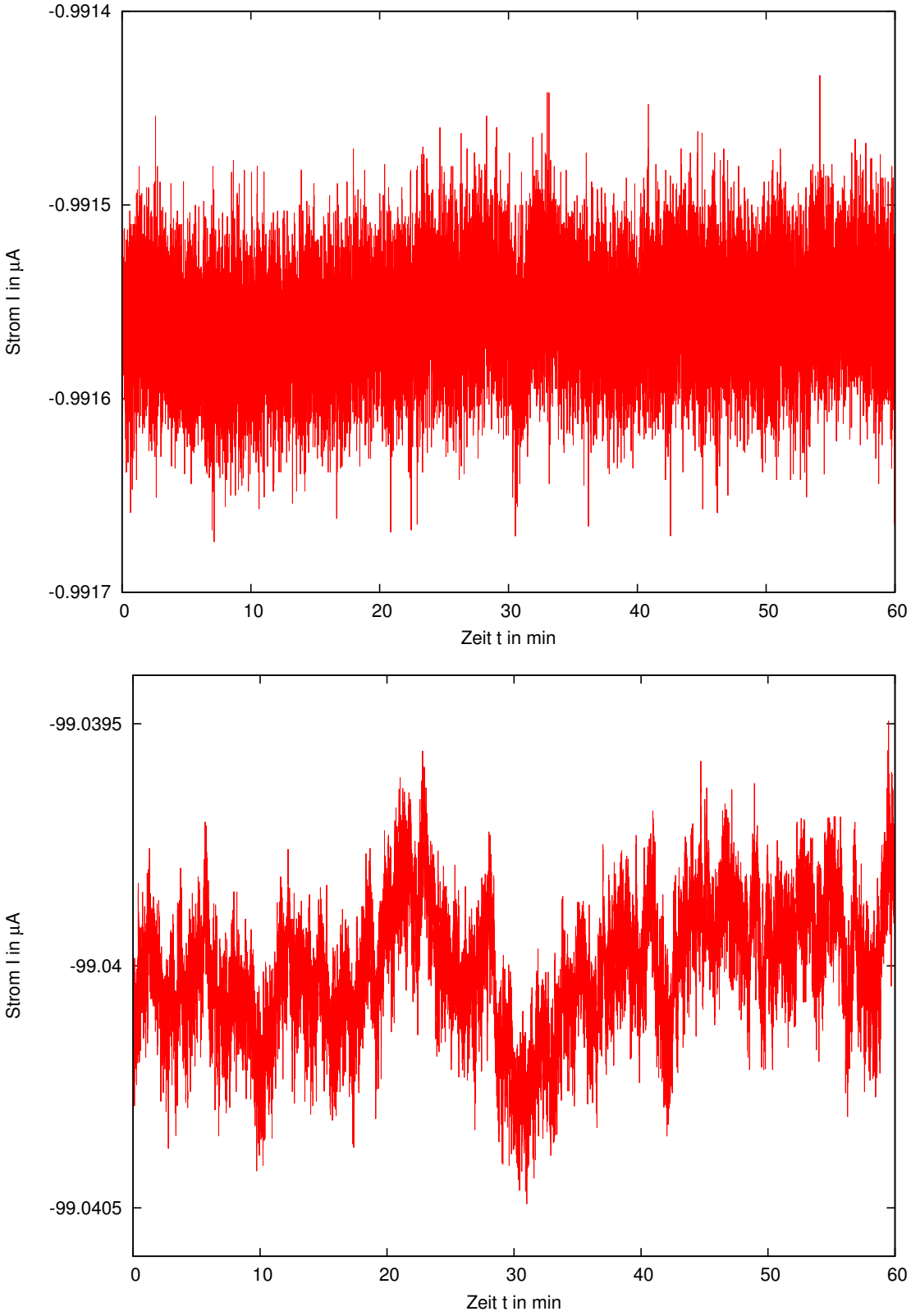


Abbildung 4.3: Oben: Messung über 60 min bei 1 µA. Unten: Messung über 60 min bei 100 µA. Die Abweichung der Absolutwerte von den Sollwerten folgt daraus, dass die Stromquelle nicht genau auf 1 µA bzw. 100 µA kalibriert ist.

heit, die aus der Umrechnung des Stroms folgte hinzu, ergibt sich eine Gesamtunsicherheit von $\Delta I = 0,096515\,\text{nA} + 0,023\,\text{nA} \approx 0,12\,\text{nA}$.

Eine weitere fehlerbehaftete Größe ist die Zeit. Hier liegt die Unsicherheit, bedingt durch die Auslese, bei 1 ms. Insgesamt gibt es somit zwei fehlerbehaftete Größen. Die Gesamtunsicherheit kann nun mittels Gaußscher Fehlerfortpflanzung bestimmt werden

$$\Delta Q = \sqrt{2 \cdot (\frac{I_{Ende} - I_{Start}}{N_{Ende} - N_{Start}} \cdot \Delta T)^2 + 2 \cdot (\frac{T_{Ende} - T_{Start}}{N_{Ende} - N_{Start}} \cdot \Delta I)^2} \tag{4.6}$$

$$\Delta I_{Mittel} = \sqrt{2 \cdot (\frac{\Delta I}{N_{Ende} - N_{Start}})^2}. \tag{4.7}$$

Nun kann man sich eine Abschätzung dieser Unsicherheiten anschauen. Im ersten Term der Ladungsunsicherheit taucht der Term für den mittleren Strom auf. Im zweiten Term taucht der Kehrwert der internen Messzyklusrate auf, die annähernd konstant bei etwa 7 Hz liegt. Bei einem Strom von $1\,\mu A$ folgt somit für die gesammelte Ladung eine Unsicherheit von $\Delta Q = \sqrt{2 \cdot (1\,\mu\text{A} \cdot 1 \times 10^{-3}\,\text{s})^2 + 2 \cdot (\frac{0,12 \times 10^{-3}\,\mu\text{A}}{7\,\text{Hz}})^2} \approx 0,0014\,\mu\text{C} = 1,4\,\text{nC}$. Allerdings sollte beachtet werden, dass der Strom am Beschleuniger üblicherweise nicht konstant ist.

Die Unsicherheit des mittleren Stroms hängt nur noch von der Anzahl der internen Messzyklen ab. Selbst nach nur einem Messwert, was einer Messdauer von einem Siebtel einer Sekunde entspricht, liegt der Fehler bereits bei $\Delta I_{Mittel} = \sqrt{2 \cdot (0,12\,\text{nA})^2} \approx 0,17\,\text{nA}$. Bei längeren Messdauern wird der Fehler somit verschwindend gering.

5 Das Programm HDTV

Das Programm HDTV wurde von Norbert Braun, Tanja Kotthaus und Ralf Schulze von der Universität Köln basierend auf dem älteren Programm TV [13], welches am CERN entwickelt wurde, geschrieben und wird ständig verbessert und erweitert. Es wurde in Python [14] und teilweise in C++ [15] geschrieben. Da das Programm größtenteils in Python geschrieben ist, welches eine breite Anwendung als Skriptsprache findet, kann man das Programm sehr einfach erweitern bzw. modifizieren.

Die Entscheidung dieses Programm als Basis für das Spektrometer zu benutzen, fiel aufgrund der bereits in das Programm implementierten Funktion (2.7), mit der Anpassungen für Elektronenstreuspektren vorgenommen werden können. Eine ausführliche Beschreibung aller Funktionen des Programms sowie die aktuellste Version findet sich auf der Internetseite der Entwickler [13].

5.1 Erweiterte Funktionen

Das Programm wurde um mehrere Funktionen zur direkten Steuerung des Spektrometers erweitert. Des Weiteren wurden einige nützliche Funktionen eingebaut, die im folgenden erläutert werden. Alle neuen Befehle werden mit dem Schlüsselwort *lintott* eingeleitet, gefolgt von dem eigentlichen Befehl, evtl. mit weiteren Argumenten.

1. *lintott start* [Auslesezeit]: Dieser Befehl startet das Spektrometer. Als optionales Argument kann man die Auslesezeit festlegen. Diese hat einen Standardwert von 2 s. Dies ist vor allem bei starker Netzwerkbelastung nützlich, da es in solchen Fällen zu Störungen des Startvorgangs kommen kann. Nach der Eingabe des Befehls werden einige Meldungen zum Ausführungsstatus des Befehls im Terminal ausgegeben (siehe Abb. 5.1). Schließlich wird der Benutzer aufgefordert einen Dateinamen für das Spektrum einzugeben. Auch wird mit diesem Befehl gleichzeitig die Strommessung gestartet. Zusätzlich wird alle 5 Minuten automatisch eine Sicherungskopie des Spektrums erstellt. Hierbei werden die folgenden Daten abgespeichert: Das Rohspektrum, das Spektrum mit Kanalnummer und der entsprechenden Ereignisanzahl in dem jeweiligen Kanal, eine Datei mit diversen Informationen zum Spektrum (Totzeit, Messzeit, mittlerer Strom etc.), Rohdaten mit Strominformationen beim Starten bzw. beim Beenden der Messung sowie ein Graph des Spektrums.

2. *lintott update* [Auslesezeit]: Mit diesem Befehl kann man das aktuelle Spektrum auslesen ohne den Messvorgang zu beenden. Auch hier gibt es die Möglichkeit die Auslesezeit festzulegen. Hier liegt der Standardwert für die Auslesezeit bei 0,5 s, da diese Funktion öfter

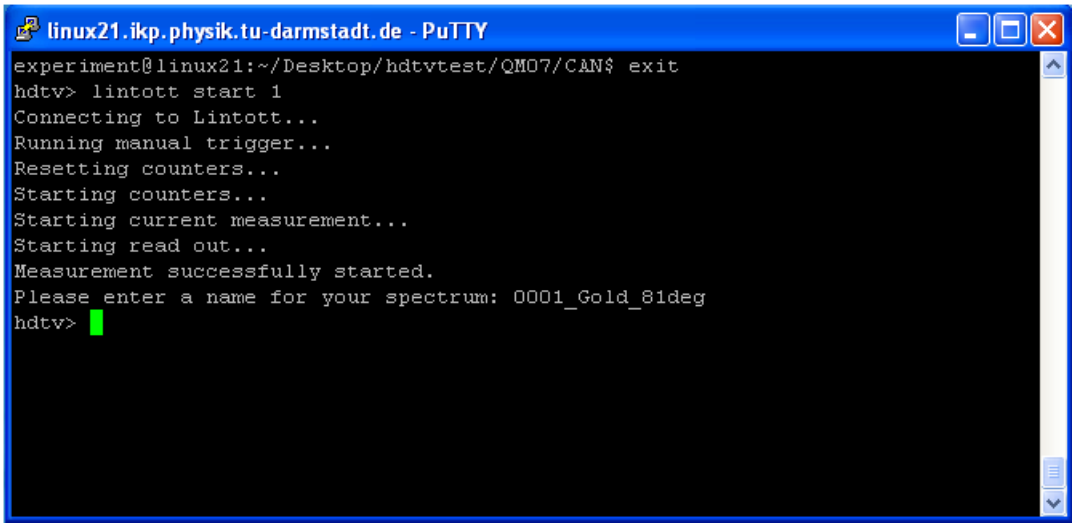


Abbildung 5.1: Diverse Meldungen zum Ausführungsstatus des Befehls *lintott start*. Die Auslesezeit wurde hier auf 1 s gesetzt.

benutzt wird und eine zu niedrige Auslesezeit nicht zu einem Absturz des Auslesemoduls des Spektrometers führen würde wie dies bei *lintott start* oder *lintott stop* der Fall wäre. Nach der Eingabe des Befehls werden auch hier einige Meldungen zum Ausführungsstatus im Terminal ausgegeben. Zusätzlich werden im Terminal einige wichtige Informationen zum Spektrum ausgegeben. Die wichtigsten sind Zählrate, Totzeit und Messdauer. Des Weiteren werden Informationen des QM07 angezeigt, nämlich der mittlere Strom, die gesammelte Ladung, die interne QM07-Zeit sowie das Verhältnis der Fläche des elastischen Peaks zur gesammelten Ladung (siehe hierzu Kapitel 2, Formel (2.6)). In einem separaten Fenster wird zudem das Spektrum selbst angezeigt. In Abb. 5.2 ist ein Beispiel nach Eingabe des Befehls gezeigt.

3. *lintott stop* [Auslesezeit]: Dieser Befehl beendet die Messung und speichert diese. Der Standardwert für die Auslesezeit beträgt hier wieder 2 s. Zusätzlich zu den gespeicherten Daten, die bei *lintott start* beschrieben wurden, werden zwei weitere Dateien abgespeichert, nämlich der Stromverlauf in Hexadezimalzahlen und ein Graph desselben. Im Terminal erhält man ebenfalls diverse Informationen, wie bei *lintott update*. Auch hier wird das Spektrum schließlich grafisch dargestellt.

4. *lintott status*: Mit diesem Befehl kann abgefragt werden, ob momentan eine Messung läuft oder nicht.

5. *lintott load* [Datei1, Datei2, ...]: Dieser Befehl erlaubt es, ein oder mehrere Spektren zu laden. Die Besonderheit gegenüber dem allgemeinen HDTV-Befehl zum Laden eines Spektrums liegt darin, dass hiermit automatisch die Anpassungsfunktion für Elektronenstreuspektren gewählt wird. Noch wichtiger ist jedoch, dass mit dem allgemeinen HDTV-Befehl die LINTOTT-Spektren fehlerhaft eingelesen werden. Wie in Abschnitt 1.1 beschrieben

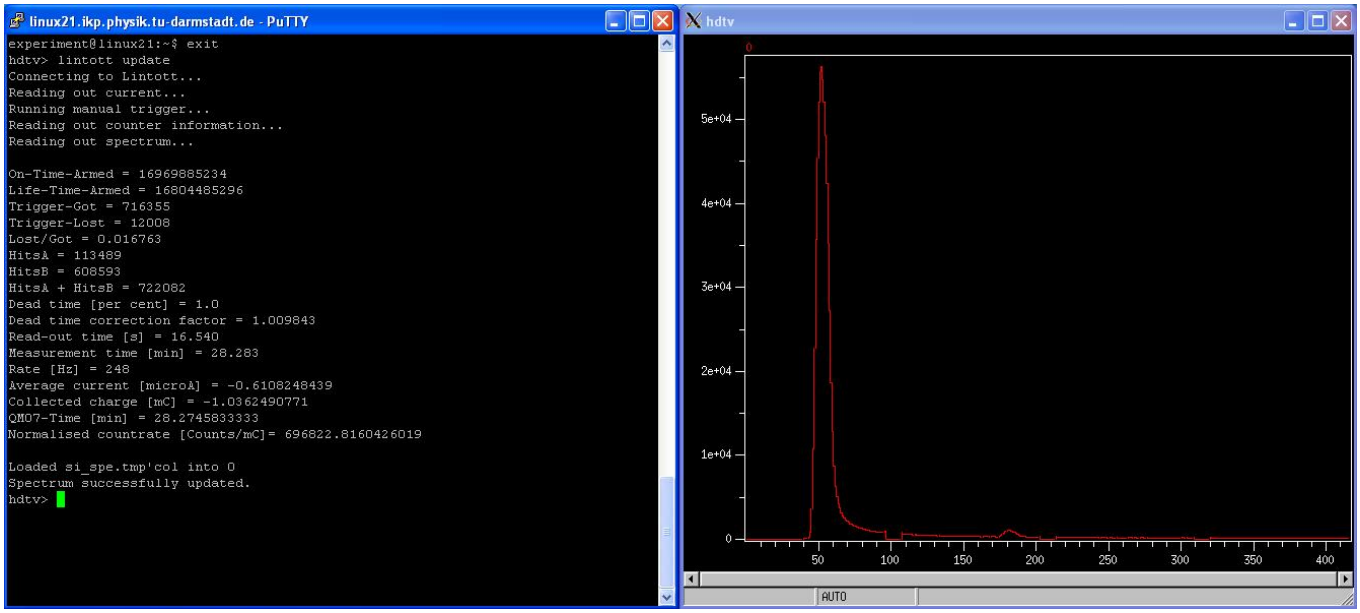


Abbildung 5.2: Diverse Meldungen zum Ausführungsstatus des Befehls *lintott update* sowie Informationen zum Spektrum und das Spektrum selbst in einem separaten Fenster.

besteht der Halbleiterdetektor des Spektrometers aus vier einzelnen, an einander angereihten Modulen, wodurch im Spektrum drei Lücken entstehen. Das Programm fehlinterpretiert diese Lücken, so dass es nach jeder Lücke zur Verdopplung der Kanalbreite des nach der Lücke folgenden Kanals kommt. Dieses Problem wird mit dem Befehl *lintott load* berücksichtigt und behoben, so dass ausschließlich dieser Befehl zum Einlesen der LINTOTT-Spektren benutzt werden sollte.

6. *lintott graph* [Datei1, Datei2, ...]: Mit diesem Befehl ist es möglich einen Graphen aus den angegebenen Spektren zu erstellen. Hierzu hat man drei verschiedene Formate zur Auswahl: eps, svg, png. Nach der Eingabe des Befehls und der gewünschten Spektren wird der Benutzer nach dem gewünschten Format gefragt. Nach der Auswahl des Formats wird schließlich noch der Dateiname abgefragt, unter dem der Graph abgespeichert werden soll. Werden als Argument mehrere Spektren übergeben, so werden diese in einem Graphen dargestellt. Des Weiteren werden die Graphen mit einer Anpassungsfunktion ausgegeben, falls eine Anpassung vorher durchgeführt wurde.

Die gesamte Programmerweiterung mit allen Befehlen ist im Anhang B mit Dokumentation dargestellt. Bei der Neuinstallation sollte beachtet werden, dass im Hauptprogramm von HDTV das Erweiterungsskript eingebunden werden muss. Hierzu muss im Quellcode des Hauptprogramms in der Datei *hdtv* die Zeile *import hdtv.plugins.lintott* hinzugefügt werden. Zusätzlich muss das Skript *lintott.py* und der dazugehörige Ordner *Lintott* in das Verzeichnis des Programms */hdtv/plugins/* kopiert werden.

6 Inbetriebnahme

6.1 Erste Experimente

Nach der Fertigstellung der Programmierarbeiten und der Einbindung des QM07 in den Auslesprozess wurden erste Experimente mit einer Batterie und einem Widerstand durchgeführt. Ziel dieser Experimente war es, die QM07-Elektronik sowie das Programm auf die richtige Auslese und Umwandlung der Daten in Stromwerte zu testen. Auch wurden hiermit die Zuleitungen vom Faraday-Cup in den Messraum getestet. Der Messaufbau ist in Abb. 6.1 zu sehen. Die Messungen wurden über ein Wochenende aufgenommen. Da das QM07 nur Ströme bis $\pm$ 100 µA messen kann, wurden entsprechend angepasste Widerstände verwendet. Bei der Batterie handelte es sich um eine handelsübliche 1,5 V-Batterie. Der Widerstandswert betrug 120 kΩ. Hieraus folgt ein Strom von $I = \frac{U}{R} = \frac{1{,}5\,\mathrm{V}}{120\,\mathrm{k\Omega}} = 12{,}5\,\mu\mathrm{A}$. Die Messung ist in Abb. 6.2 dargestellt. Die Abweichung vom theoretischen Stromwert lässt sich damit erklären, dass es sich um eine bereits gebrauchte Batterie handelte. Aus der Messung folgt eine ordnungsgemäße Funktionsfähigkeit der QM07-Elektronik.

6.2 Experimente am Spektrometer

Nach den erfolgreichen Experimenten mit der Batterie wurden schließlich während der Teststrahlzeit im Dezember 2011 Experimente am LINTOTT-Spektrometer durchgeführt. Hierbei hat sich die ordnungsgemäße Funktionalität des Gesamtsystems, bis auf kleinere Programmierschwachstellen, die beseitigt wurden, wieder bestätigt.

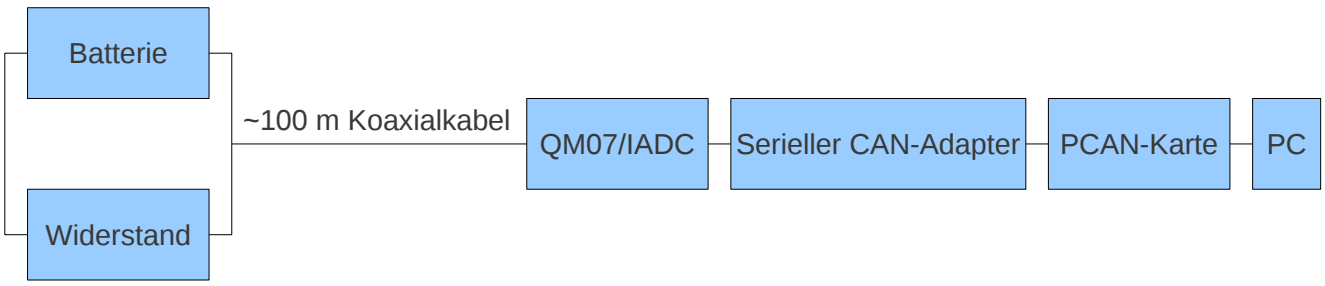


Abbildung 6.1: Schematischer Messaufbau zum Test der QM07-Elektronik.

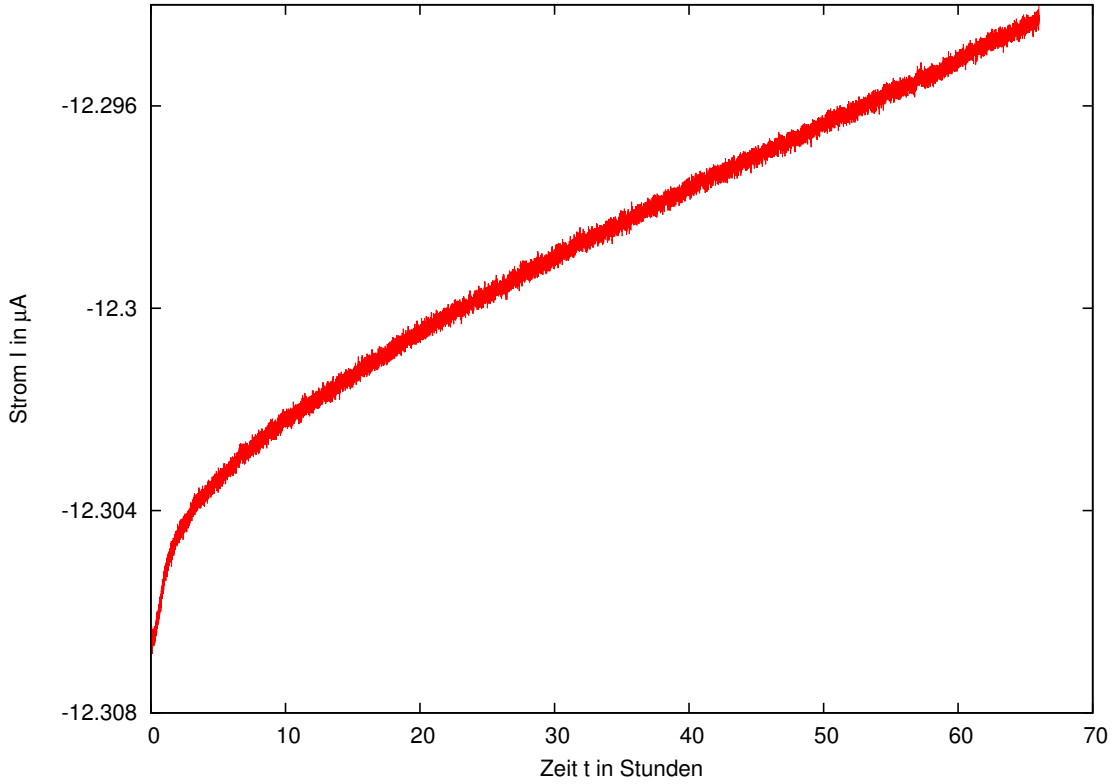


Abbildung 6.2: Messung einer Batterieentladung über das Wochenende.

6.3 EPICS System

Das EPICS (Experimental Physics and Industrial Control System) ist eine Open-Source-Software um Kontrollsysteme für Beschleuniger zu entwickeln [16]. Das System ermöglicht die Kommunikation zwischen verschiedenen Computern und kann somit Informationen von verschiedenen Messinstrumenten sammeln und auf einem Computer zusammenfassen und wiedergeben. So werden beim S-DALINAC bereits die digitale HF-Regelung, die thermionische Kanone sowie eine Reihe weiterer Instrumente über EPICS gesteuert [17]. Auch das QM07 stellt die Strominformationen über EPICS im Kontrollraum bereit, so dass der Stromfluss am LINTOTT-Spektrometer in Echtzeit wiedergegeben werden kann. Dies ist vor allem zur Strahldiagnose sehr wertvoll. Abb. 6.3 zeigt ein Bildschirmfoto des Stromverlaufs, welcher mit Hilfe des EPICS Systems direkt im Kontrollraum in Echtzeit angezeigt wird.

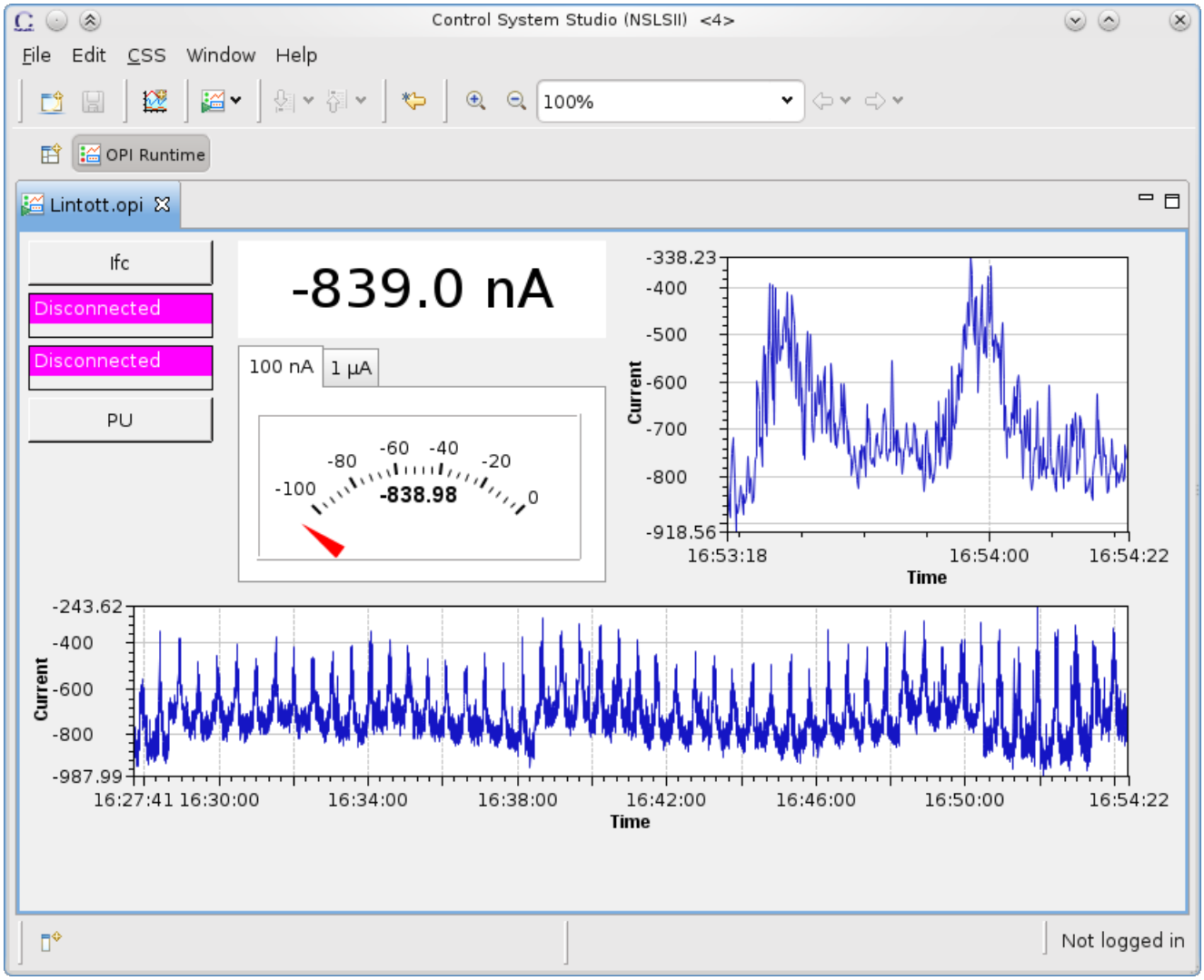


Abbildung 6.3: Ein Bildschirmfoto des Stromverlaufs.

7 Zusammenfassung und Ausblick

Um die Steuerung des LINTOTT-Spektromenters zu optimieren, wurden einige neue Befehle in das HDTV-Programm eingefügt. Die wichtigsten Befehle für die Steuerung sind *lintott start*, *lintott update* und *lintott stop*. Mit diesen Befehlen kann das Spektrometer gestartet bzw. gestoppt werden sowie mit dem Befehl *lintott load* Spektren in das Programm eingelesen werden. Spektren können im Programm nach belieben bearbeitet und in verschiedenen graphischen Formaten abgespeichert werden. Des Weiteren wurde die QM07-Elektronik erfolgreich in das Auslesesystem des Spektrometers integriert. Hieraus erhält man Informationen über den mittleren Strom und die gesammelte Ladung. Zusätzlich können diese Informationen direkt aus dem Kontrollraum abgerufen und zur Strahldiagnose benutzt werden.

Wie bei jedem Programm, gibt es immer viele Dinge, die verbessert oder erweitert werden können sowie neue Ideen, die man in das Programm implementieren könnte. Zum Beispiel könnte man für alle Werte, die das Programm ausgibt, noch die jeweiligen Unsicherheiten berechnen und ausgeben. Wie man in Kapitel 4 gesehen hat, sind die theoretischen Grundlagen bereits vorhanden. Eine Weitere Idee wäre der direkte Vergleich zwischen den Verhältnissen der Fläche der elastischen Linie zur gesammelten Ladung zweier verschiedener Spektren, was eine bessere und direktere Aussage über den ordnungsgemäßen Ablauf des Messprozesses liefern würde. Auch müsste noch eine Sicherung der Stromdaten in einer zentralen Datenbank realisiert werden. Des Weiteren könnte man eine ähnliche Modernisierung der Stromauslese beim QCLAM-Spektrometer durchführen, da dort zur Stromauslese ebenfalls ein Current-Digitiser benutzt wird, wie er bisher am LINTOTT im Einsatz war.

A Installationsanleitung

A.1 Installation des CAN-Treibers

Der Treiber muss auf dem Messrechner installiert werden, im Moment *linux21* im Messraum. Man muss alle Befehle als *root* oder unter Ubuntu mit *sudo* ausführen.

Als erstes muss man den Treiber herunterladen. Diesen findet man auf der Internetseite des Herstellers [18]. Als nächstes muss man den Treiber mit *tar -xzf peak-linuxdriver-7.4.tar.yz* entpacken und das Verzeichnis betreten. Aktuell ist die Version 7.4 installiert. Vor der Installation sollte noch darauf geachtet werden, dass die Packete *libpopt* und *libstdc++* installiert sind. Außerdem darf die Kernelversion von Linux nicht älter als 2.6.x sein. Sind alle Voraussetzungen erfüllt, startet man die Installation aus dem Verzeichnis des Treibers mit den Befehlen *make* und danach *make install*. Wurde der Treiber erfolgreich installiert muss man noch die Datenrate setzen. Dazu wechselt man in das Verzeichnis */etc/modeprobe.d/*. Hier muss man nun die Datei *pcan.conf* editieren (z.B. mit dem Editor *nano*). In dieser Datei muss die folgende Zeile hinzufügt werden *„options pcan bitrate = 0x0014“*. Außerdem muss im Verzeichnis */etc/* die Datei *modules* editiert werden. Dort werden die Zeilen *„lp“*, *„rtc“* und *„can-raw“* hinzugefügt. Nun gibt man den Befehl *modprobe can-raw* in das Terminal ein und startet den Computer neu. Mit diesem Befehl wird das Modul eingebunden. Nach dem Neustart führt man den Befehl *ifconfig can1 up* aus. Dieser muss nach jedem Neustart ausgeführt werden. Um das Modul beim Neustart automatisch zu laden kann man die Datei *45-pcan.rules* im Verzeichnis */etc/udev/rules.d/* editieren. Dort muss die Zeile *„RUN+=”/sbin/ifconfig %k up”“* hinzugefügt werden. Nun sollte der Treiber installiert sein. Um Daten an das Gerät zu senden oder vom Gerät zu empfangen muss jedoch zusätzliche Software, wie im nächsten Abschnitt beschrieben, installiert werden.

A.2 Installation der QM07-Software

Für die Installation der QM07-Software muss das Packet *svn* installiert sein. Zum herunterladen der Software gibt man im Terminal den Befehl *svn checkout svn://svn.berlios.de/socketcan/trunk* ein. Nach dem Herunterladen muss das Verzeichnis *trunk/can-utils/* betreten werden. Dort führt man die Befehle *make* und *make install* aus. Nun sollte auch die QM07-Software installiert sein. Jetzt sollten die Befehle *candump* und *cansend* vorhanden sein mit denen man nun mit dem QM07 interagieren kann.

B Programmcode

```
# -*- coding: utf-8 -*-

#Diese Datei ist eine Erweiterung des HDTV-Programms. Hier sind einige neue Funktionen zur Bedienung des Lintott-
     Spektrometers, sowie Funktionen zur Auswertung der Lintott-Spektra. Die Datei wurde von Sergej Bassauer im Rahmen
     einer Bachelorarbeit geschrieben. Letzte Änderung: 13.03.2012

#Importierung im Programm benötigter Pakete.
import hdtv.cmdline
import os
import time
import telnetlib
import sys
import socket
import threading

#Globale Variablendeklaration.
global firstVisit
global autoSaveStatus

#Setzen der Variablenanfangswerte.
firstVisit = False

#Funktion zum Einstellen der richtigen Fitfunktion für Elektronenstreuexperimente.
def Lintott(args):
    #Ausführen des HDTV-Befehls für das Setzten der Fitfunktion.
    hdtv.cmdline.ExecCommand("fit function ee")
    #Falls nur das Schlüsselwort "lintott" eingegeben wird, wird ein Hinweis angezeigt.
    if args == []:
        print '\nFit function set for ee-fitting.\n'

#Funktion zum Starten eines Messvorgangs am LINTOTT-Spektrometer.
def Start(args):
    #Setzen der Anfangswerte lokaler Variablen.
    answer = 'n'
    #Abfrage des aktuellen Verzeichnisses.
    olddir = os.getcwd()
    #Setzen des Pfades zum Ordner "Lintottdata".
    directory = os.path.dirname(__file__) + '/Lintottdata/'
    #Wechsel des Verzeichnisses zum Ordner "Lintottdata".
    os.chdir(directory)
    #Überprüfung ob das Spektrometer bereits läuft, falls ja wird der Start abgebrochen.
    if open('LintottRunning.dat','r').readline() == 'True':
        print '\nLINTOTT is already online. Measurement is in process.\n'
        os.chdir(olddir)
        return
    #Überprüfung, ob auf LINTOTT zugegriffen wird, falls ja wird der Start abgebrochen.
    if open('LintottAccess.dat','r').read() == 'True':
        print 'LINTOTT is busy. Please wait a few seconds and try again.'
        os.chdir(olddir)
        return
    #Setzten des Zugriffsstatus auf Wahr.
    open('LintottAccess.dat','w').write('True')
    #Setzten der Auslesezeit.
    try:
        readouttime = float(args[0])
    except:
        readouttime = 2
    #Öffnen des candump-Programms in dem die Stromdaten empfangen werden und Umleiten dieser in eine Datei. Die Daten
         werden hierbei so gefiltert, dass nur die Stromsumme, die Anzahl der internen Messzyklen und die Zeit
         abgespeichert werden.
```

```
57      os.system('candump can1 | grep -e "111  \[8\] 00" -e "10F  \[8\] FD" -e "10F  \[8\] FC" > currentdatastart.dat &')
58      #Senden des Befehls zum Abrufen der Stromsumme.
59      os.system('cansend can1 06084111#00')
60      #Senden des Befehls zum Abrufen der Anzahl interner Messzyklen.
61      os.system('cansend can1 0608410f#fd')
62      #Senden des Befehls zum Abrufen der Zeit.
63      os.system('cansend can1 0608410f#fc')
64      #Schließen des candump-Programms.
65      os.system('killall candump')
66      #Falls keine Stromdaten erhalten wurden, wird der Benutzer nach dem weiteren Vorgehen gefragt.
67      if HexToDec('currentdatastart.dat') == [0,0,0]:
68          answer = raw_input('\nQM07 (current measurement) seems to be off. Start measurement anyway? (y/n)\n\n')
69          if answer != 'y':
70              print '\nStarting measurement aborted.\n'
71              open('LintottAccess.dat','w').write('False')
72              os.chdir(olddir)
73              return
74      #Herstellen einer Verbindug zum LINTOTT via Telnet und Setzen der nötigen Parameter.
75      print 'Connecting to LINTOTT...'
76      try:
77          tn = telnetlib.Telnet('lintott_r')
78      #Falls die Verbindung nicht möglich ist, wird noch ein Versuch nach 5 Sekunden unternommen.
79      except EnvironmentError:
80          print 'Connection failed. Trying to reconnect...'
81          time.sleep(5)
82          try:
83              tn = telnetlib.Telnet('lintott_r')
84          #Scheitert auch der zweite Versuch, wird der Vorgang abgebrochen.
85          except EnvironmentError:
86              print 'Connecting to LINTOTT was not possible.'
87              open('LintottAccess.dat','w').write('False')
88              os.chdir(olddir)
89              return
90      #Setzen der LINTOTT-Parameter
91      tn.read_until('Username: ')
92      tn.write('burda\n')
93      tn.read_until('Password: ')
94      tn.write('lintott\n')
95      tn.write('enexttrg(0);\n')
96      time.sleep(0.2)
97      tn.write('setset(0x0410);\n')
98      time.sleep(0.2)
99      tn.write('setreqstat(3);\n')
100     time.sleep(0.2)
101     print 'Running manual trigger...'
102     tn.write('mantrg();\n')
103     time.sleep(0.2)
104     print 'Resetting counters...'
105     tn.write('rstallcnts();\n')
106     time.sleep(0.2)
107     print 'Starting counters...'
108     tn.write('setset(0);\n')
109     time.sleep(0.2)
110     print 'Starting current measurement...'
111     os.system('candump can1 | grep -e "111  \[8\] 00" -e "10F  \[8\] FD" -e "10F  \[8\] FC" > currentdatastart.dat &')
112     #Abspeichern der Prozess-ID.
113     os.system('pidof candump > pid.dat')
114     os.system('cansend can1 06084111#00')
115     os.system('cansend can1 0608410f#fd')
116     os.system('cansend can1 0608410f#fc')
117     #Starten eines weiteren candump-Programms, das den Stromverlauf misst und alle 10 s abspeichert.
118     os.system('candump can1 | grep "708410F  \[8\] 00" | sed -ne "1~63p" > currenthex.dat &')
119     print 'Starting read out...'
120     tn.write('enexttrg(1);\n')
121     time.sleep(readouttime)
122     tn.write('exit\n')
```

```
123     time.sleep(0.1)
124     data = tn.read_all()
125     #Abspeichern der Prozess-ID der Stromverlaufmessung.
126     os.system('pidof -o "%s" candump > pidcurrent.dat' % open('pid.dat','r').read())
127     #Beenden des candump-Programms, das die Stromsumme, Anzahl der interner Messzyklen und die Zeit auslas.
128     os.system('kill %s' % open('pid.dat','r').read())
129     #Schreiben der gesendeten Befehle und der erhaltenen Antworten in eine Logdatei.
130     open('StartLogFile.dat','w').write(data)
131     #Neustart der Messung, falls Stromauslese Fehlerhaft wahr.
132     if (HexToDec('currentdatastart.dat') == [0,0,0]) and (answer != 'y'):
133         print '\n!WARNING! - Current read out failed - !WARNING!\n'
134         print 'I will try to restart the measurement...\n'
135         time.sleep(5)
136         os.system('killall candump')
137         open('LintottAccess.dat','w').write('False')
138         os.chdir(olddir)
139         #Neustart.
140         hdtv.cmdline.ExecCommand("lintott start")
141         return
142     #Lintottstatus wird auf Wahr gesetzt (D.h. LINTOTT ist nun am Messen).
143     open('LintottRunning.dat','w').write('True')
144     print 'Measurement successfully started.'
145     #Abfragen des gewünschten Namens für das Spektrum.
146     specName = raw_input("Please enter a name for your spectrum: ")
147     #Abspeichern des Namens in einer Datei.
148     open('SpecName.dat','w').write(specName)
149     #Schließen der Telnetverbindung.
150     tn.close()
151     #Setzen des Zugriffsstatus auf Falsch.
152     open('LintottAccess.dat','w').write('False')
153     os.chdir(olddir)
154 
155 #Funktion zum Auslesen des Spekrums ohne die Messung zu beenden.
156 def Update(args):
157     global firstVisit
158     olddir = os.getcwd()
159     directory = os.path.dirname(__file__) + '/Lintottdata/'
160     os.chdir(directory)
161     #Überprüfung ob das Spektrometer läuft, falls nicht wird das Spektrum nicht ausgelesen.
162     if open('LintottRunning.dat','r').readline() == 'False':
163         print '\nLINTOTT is offline. No measurement is in process.\n'
164         os.chdir(olddir)
165         return
166     if open('LintottAccess.dat','r').read() == 'True':
167         print 'LINTOTT is busy. Please wait a few seconds and try again.'
168         os.chdir(olddir)
169         return
170     open('LintottAccess.dat','w').write('True')
171     print 'Connecting to LINTOTT...'
172     try:
173         tn = telnetlib.Telnet('lintott_r')
174     except EnvironmentError:
175         print 'Connection failed. Trying to reconnect...'
176         time.sleep(5)
177         try:
178             tn = telnetlib.Telnet('lintott_r')
179         except EnvironmentError:
180             print 'Connecting to LINTOTT was not possible.'
181             open('LintottAccess.dat','w').write('False')
182             os.chdir(olddir)
183             return
184     try:
185         readouttime = float(args[0])
186     except:
187         readouttime = 0.5
188     tn.read_until('Username: ')
```

```
189     tn.write('burda\n')
190     tn.read_until('Password: ')
191     tn.write('lintott\n')
192     print 'Reading out current...'
193     os.system('candump can1 | grep -e "111  \[8\] 00" -e "10F  \[8\] FD" -e "10F  \[8\] FC" > currentdataend.dat &')
194     os.system('cansend can1 06084111#00')
195     os.system('cansend can1 0608410f#fd')
196     os.system('cansend can1 0608410f#fc')
197     tn.write('setreqstat(3);\n')
198     time.sleep(0.2)
199     print 'Running manual trigger...'
200     tn.write('mantrg();\n')
201     time.sleep(0.2)
202     print 'Reading out counter information...'
203     tn.write('getallcnts(0);\n')
204     time.sleep(0.2)
205     print 'Reading out spectrum...'
206     tn.write('getspecrams(1);\n')
207     time.sleep(readouttime)
208     tn.write('exit\n')
209     time.sleep(0.1)
210     data = tn.read_all()
211     open('UpdateLogFile.dat','w').write(data)
212     os.system('pidof -o "%s" candump > pid.dat' % open('pidcurrent.dat','r').read())
213     os.system('kill %s' % open('pid.dat','r').read())
214     #Suchen und Abspeichern der Messwerte in einer separaten Datei (Hexadezimal).
215     open('si_counter.dat','w').write(data[data.find('burda >>  Spectra RAMs:') + 23:-56])
216     #Ausführen eines Skriptes, welches die Messwerte aus dem Hexadezimalen ins Dezimale konvertiert.
217     os.system('./si_conv')
218     #Suchen und Abspeichern der Messinformationen (Totzeit, Messzeit, etc.) in einer separaten Datei (Hexadezimal).
219     rateraw = data[data.find('On-Time-Armed:'):data.find('burda >>  Spectra RAMs:')]
220     rateraw = rateraw.replace('.',':').split(':')
221     text = ''
222     for number in rateraw:
223         if number.find('x') != -1:
224             text = text + number.lstrip() + '\n'
225     open('rate_raw.dat','w').write(text)
226     #Ausführen eines Skriptes, welches die Messinformationen aus dem Hexadezimalen ins Dezimale konvertiert.
227     os.system('./rate_conv')
228     #Aufrufen der Funktion zum Berechnen der Stromdaten.
229     currentdata = CurrentAndCharge()
230     #Ausgabe der Messinformationen, sowie der Stromdaten auf dem Bildschirm.
231     print '\n' + open('rate_conv_out.dat','r').read()
232     #Löschen des Spektrums im HDTV-Programm, falls vorhanden.
233     if firstVisit != False:
234         hdtv.cmdline.ExecCommand("spectrum delete all")
235     #Laden des Spektrums in HDTV.
236     GetSpec(['si_spe.dat'])
237     #Setzen der Variable (Spektrum geladen) auf Wahr.
238     firstVisit = True
239     #Ausgabe einer Warnung, falls die Stromdaten nicht richtig eingelesen werden konnten.
240     if currentdata == (0,0,0):
241         print '\n!WARNING! - Current read out failed - !WARNING!\n'
242         print 'Spectrum updated.'
243     else:
244         print 'Spectrum successfully updated.'
245     tn.close()
246     open('LintottAccess.dat','w').write('False')
247     os.chdir(olddir)
248 
249 #Funktion zum Beenden und Abspeichern der Messung.
250 def Stop(args):
251     global firstVisit
252     olddir = os.getcwd()
253     directory = os.path.dirname(__file__) + '/Lintottdata/'
254     os.chdir(directory)
```

```
    if open('LintottRunning.dat','r').readline() == 'False':
        print '\nLINTOTT is offline. No measurement is in process.\n'
        os.chdir(olddir)
        return
    if open('LintottAccess.dat','r').read() == 'True':
        print 'LINTOTT is busy. Please wait a few seconds and try again.'
        os.chdir(olddir)
        return
    open('LintottAccess.dat','w').write('True')
    print 'Connecting to LINTOTT...'
    try:
        tn = telnetlib.Telnet('lintott_r')
    except EnvironmentError:
        print 'Connection failed. Trying to reconnect...'
        time.sleep(5)
        try:
            tn = telnetlib.Telnet('lintott_r')
        except EnvironmentError:
            print 'Connecting to LINTOTT was not possible.'
            open('LintottAccess.dat','w').write('False')
            os.chdir(olddir)
            return
    try:
        readouttime = float(args[0])
    except:
        readouttime = 2
    tn.read_until('Username: ')
    tn.write('burda\n')
    tn.read_until('Password: ')
    tn.write('lintott\n')
    print 'Reading out current...'
    os.system('candump can1 | grep -e "111  \[8\] 00" -e "10F  \[8\] FD" -e "10F  \[8\] FC" > currentdataend.dat &')
    os.system('cansend can1 06084111#00')
    os.system('cansend can1 0608410f#fd')
    os.system('cansend can1 0608410f#fc')
    #Beenden aller candump-Programme, d.h. der Strominfos und des Stromverlaufs.
    os.system('killall candump')
    print 'Stopping read out...'
    tn.write('enexttrg(0);\n')
    time.sleep(0.2)
    tn.write('setreqstat(3);\n')
    time.sleep(0.2)
    print 'Running manual trigger...'
    tn.write('mantrg();\n')
    time.sleep(0.2)
    print 'Reading out counter information...'
    tn.write('getallcnts(0);\n')
    time.sleep(0.2)
    print 'Reading out spectrum...'
    tn.write('getspecrams(1);\n')
    time.sleep(readouttime)
    tn.write('exit\n')
    time.sleep(0.1)
    data = tn.read_all()
    open('StopLogFile.dat','w').write(data)
    open('si_counter.dat','w').write(data[data.find('burda >>  Spectra RAMs:') + 23:-56])
    os.system('./si_conv')
    rateraw = data[data.find('On-Time-Armed:'):data.find('burda >>  Spectra RAMs:')]
    rateraw = rateraw.replace('.',':').split(':')
    text = ''
    for number in rateraw:
        if number.find('x') != -1:
            text = text + number.lstrip() + '\n'
    open('rate_raw.dat','w').write(text)
    os.system('./rate_conv')
    currentdata = CurrentAndCharge()
```

```
321     print '\n' + open('rate_conv_out.dat','r').read()
322     if firstVisit != False:
323         hdtv.cmdline.ExecCommand("spectrum delete all")
324     GetSpec(['si_spe.dat'])
325     firstVisit = True
326     open('LintottRunning.dat','w').write('False')
327     SaveSpectrum()
328     tn.close()
329     if currentdata == (0,0,0):
330         print '\n!WARNING! - Current read out failed - !WARNING!\n'
331         print 'Measurement stopped.'
332     else:
333         print 'Measurement successfully stopped.'
334     open('LintottAccess.dat','w').write('False')
335     open('LintottRunning.dat','w').write('False')
336     os.chdir(olddir)
337 
338 #Funktion zum Abspeichern des Spektrums.
339 def SaveSpectrum():
340     #Abspeichern der auktuellen Zeitinformationen im Format: (Jahr, Monat, Tag, Stunde, Minute, Sekunde, Wochentag,
          Jahrtag, dls-flag).
341     t = time.localtime()
342     #Einlesen des vom Benutzer eingegebenen Dateinamens.
343     try:
344         filename = open('SpecName.dat','r').read()
345     except IOError:
346         filename = raw_input("No filename was found. Please enter a filename to save the spectrum: ")
347         open('SpecName.dat','w').write(filename)
348     #Nach dem Dateinamen werden die ersten 6 Zeitinfos angehängt.
349     filename = (filename + '_%04i-%02i-%02i_%02i-%02i-%02i') % (t[0], t[1], t[2], t[3], t[4], t[5])
350     #Erstellen eines Ordners mit dem vom Benutzer angegebenen Dateinamen und den Zeitinfos.
351     os.system('mkdir /home/experiment/Work/Lintott/Lintott_Control/lintottdata/' + filename)
352     #Falls die Messung gestoppt wurde, werden einige zusätzliche Dateien abgespeichert.
353     if open('LintottRunning.dat','r').readline() == 'False':
354         #Erstellen eines gnuplot-Graphen des Stromverlaufs.
355         HexToCurrent(filename)
356         #Kopieren des Stromverlaufs (in Hexadezimal) in den oben genannten Ordner.
357         os.system('cp currenthex.dat /home/experiment/Work/Lintott/Lintott_Control/lintottdata/' + filename + '/' +
              filename + '_currenthex.dat')
358         #Kopieren des Stromverlaufs (in Mikroampere) in den oben genannten Ordner.
359         os.system('cp currentmicroampere.dat /home/experiment/Work/Lintott/Lintott_Control/lintottdata/' + filename + '/
              ' + filename + '_currentmicroampere.dat')
360         #Löschen des Namens.
361         os.system('rm SpecName.dat')
362         #Löschen der Prozess-ID.
363         os.system('rm pid.dat')
364         #Löschen der Prozess-ID des Stromverlaufs.
365         os.system('rm pidcurrent.dat')
366     #Kopieren des Spektrums in den oben genannten Ordner.
367     os.system('cp si_spe.dat /home/experiment/Work/Lintott/Lintott_Control/lintottdata/' + filename + '/' + filename + '
          .dat')
368     #Kopieren des Rohspektrums in den oben genannten Ordner.
369     os.system('cp si_counter.dat /home/experiment/Work/Lintott/Lintott_Control/lintottdata/' + filename + '/' + filename
           + '_raw.dat')
370     #Kopieren der Messinformationen in den oben genannten Ordner.
371     os.system('cp rate_conv_out.dat /home/experiment/Work/Lintott/Lintott_Control/lintottdata/' + filename + '/' +
          filename + '_rate.dat')
372     #Kopieren der Strominformationen (in Hexadezimal) beim Start in den oben genannten Ordner.
373     os.system('cp currentdatastart.dat /home/experiment/Work/Lintott/Lintott_Control/lintottdata/' + filename + '/' +
          filename + '_currenthexstart.dat')
374     #Kopieren der Strominformationen (in Hexadezimal) beim Update bzw. Stop in den oben genannten Ordner.
375     os.system('cp currentdataend.dat /home/experiment/Work/Lintott/Lintott_Control/lintottdata/' + filename + '/' +
          filename + '_currenthexend.dat')
376     script = open('spectrum.tmp','w')
377     script.write(
378     #Gnuplot-Code.
```

```
379     "unset key\n\
380     set samples 10000,10000\n\
381     set xlabel 'channel'\n\
382     set ylabel 'counts'\n\
383     set terminal postscript color solid\n\
384     set out '/home/experiment/Work/Lintott/Lintott_Control/lintottdata/%s/%s.eps'\n\
385     plot '%s' with histeps"\
386     % (filename, filename, ConvertFile('si_spe.dat')))
387     script.close()
388     #Erstellen des Graphen.
389     os.system('gnuplot spectrum.tmp')
390     #Löschen des Gnuplot-Skriptes.
391     os.remove('spectrum.tmp')
392     #Löschen der temporären Spektrumsdatei.
393     os.remove('si_spe.tmp')
394     if open('LintottRunning.dat','r').readline() == 'False':
395         print 'The spectrum was saved in /home/experiment/Work/Lintott/Lintott_Control/lintottdata/' + filename + '/'
396
397 #Funktion, die das Spektrum alle 5 min abspeichert.
398 def AutoSave():
399     while True:
400         #Wiederholung alle 5 min.
401         time.sleep(300)
402         olddir = os.getcwd()
403         directory = os.path.dirname(__file__) + '/Lintottdata/'
404         os.chdir(directory)
405         #Überprüfung, ob eine Messung läuft und ob kein anderer Befehl im Moment ausgeführt wird.
406         if open('LintottRunning.dat','r').readline() == 'True' and open('LintottAccess.dat','r').readline() == 'False':
407             open('LintottAccess.dat','w').write('True')
408             print 'Autosaving spectrum',
409             #Sofortige Ausgabe des print-Befehls.
410             sys.stdout.flush()
411             try:
412                 tn = telnetlib.Telnet('lintott_r')
413             except EnvironmentError:
414                 time.sleep(10)
415                 try:
416                     tn = telnetlib.Telnet('lintott_r')
417                 except EnvironmentError:
418                     print 'Autosaving failed. Connecting to LINTOTT was not possible. Try to restart HDTV.'
419                     open('LintottAccess.dat','w').write('False')
420                     open('AutoSaveStatus.dat','w').write('False')
421                     os.chdir(olddir)
422                     return
423             tn.read_until('Username: ')
424             tn.write('burda\n')
425             print '.',
426             sys.stdout.flush()
427             tn.read_until('Password: ')
428             tn.write('lintott\n')
429             os.system('candump can1 | grep -e "111  \[8\] 00" -e "10F  \[8\] FD" -e "10F  \[8\] FC" > currentdataend.dat
                    &')
430             os.system('cansend can1 06084111#00')
431             os.system('cansend can1 0608410f#fd')
432             os.system('cansend can1 0608410f#fc')
433             print '.',
434             sys.stdout.flush()
435             tn.write('setreqstat(3);\n')
436             time.sleep(0.2)
437             print '.',
438             sys.stdout.flush()
439             tn.write('mantrg();\n')
440             time.sleep(0.2)
441             print '.',
442             sys.stdout.flush()
443             tn.write('getallcnts(0);\n')
```

```
            time.sleep(0.2)
            print '.',
            sys.stdout.flush()
            tn.write('getspecrams(1);\n')
            time.sleep(2)
            print '.',
            sys.stdout.flush()
            tn.write('exit\n')
            time.sleep(0.1)
            print '.',
            sys.stdout.flush()
            data = tn.read_all()
            open('UpdateLogFile.dat','w').write(data)
            os.system('pidof -o "%s" candump > pid.dat' % open('pidcurrent.dat','r').read())
            os.system('kill %s' % open('pid.dat','r').read())
            open('si_counter.dat','w').write(data[data.find('burda >>  Spectra RAMs:') + 23:-56])
            os.system('./si_conv')
            rateraw = data[data.find('On-Time-Armed:'):data.find('burda >>  Spectra RAMs:')]
            rateraw = rateraw.replace('.',':').split(':')
            text = ''
            for number in rateraw:
                if number.find('x') != -1:
                    text = text + number.lstrip() + '\n'
            open('rate_raw.dat','w').write(text)
            os.system('./rate_conv')
            currentdata = CurrentAndCharge()
            print 'Done!'
            if currentdata == (0,0,0):
                print '\n!WARNING! - Current read out failed - !WARNING!\n'
            tn.close()
            sys.stdout.flush()
            SaveSpectrum()
            open('LintottAccess.dat','w').write('False')
            os.chdir(olddir)
        else:
            os.chdir(olddir)

#Funktion zur Berechnung des Stroms aus dem Hexadezimalen.
def HexToCurrent(filename):
    #Öffnen der nötigen Dateien.
    datafile = open('currenthex.dat','r')
    current = open('currentmicroampere.dat','w')
    script = open('currentgnuplot.tmp','w')
    #Heraussuchen der nötigen Stromdaten.
    for line in datafile:
        hexnumber = []
        line = line[line.find('[8]') + 7:-9]
        line = line.split()
        line = line[-1::-1]
        hexnumber.append(''.join(line))
        #Umwandeln aus dem Hexadezimalen ins Binäre. Dies ist nötig um das Vorzeichen herauszufinden.
        binnumber = bin(int(hexnumber[0], 16))
        binnumber = binnumber[2:]
        #Falls die Länge kleiner als 32 ist (Volle Länge bei 32 Bit), dann handelt es sich um eine Positive Zahl.
        if len(binnumber) < 32:
            #Umwandlung ins Dezimale.
            number = float(int(hexnumber[0], 16))
            #Umwandlung in Mikroampere.
            curr = number*101.6/((2**(28))-1)
            #Abspeichern des Wertes in einer Datei.
            current.write('%3.10f\n' % curr)
        else:
            #Ähnliche Umrechnung und Abspeichern der negativen Zahl.
            number = -1*((2**(32))-float(int(hexnumber[0], 16)))
            curr = number*101.6/(2**(28))
            current.write('%3.10f\n' % curr)
```

```
510     script.write(
511     #Gnuplot-Code.
512     "unset key\n\
513     set terminal postscript color enhanced solid\n\
514     set samples 10000,10000\n\
515     set xlabel 'time in 10 s'\n\
516     set ylabel 'current in {/Symbol m}A'\n\
517     set out '/home/experiment/Work/Lintott/Lintott_Control/lintottdata/%s/%s_current.eps'\n\
518     plot '%s' with histeps"\
519     % (filename, filename, 'currentmicroampere.dat'))
520     datafile.close()
521     current.close()
522     script.close()
523     #Ausführen des Gnuplot-Skriptes.
524     os.system('gnuplot currentgnuplot.tmp')
525     #Löschen des Gnuplot-Skriptes.
526     os.remove('currentgnuplot.tmp')
527 
528 #Funktion zum Einlesen der Stromdaten aus dem QM07, sowie zum Umwandeln der Daten in Dezimalzahlen.
529 def HexToDec(filename):
530     datafile = open(filename,'r')
531     data = []
532     #Heraussuchen der nötigen Stromdaten.
533     for line in datafile:
534         line = line[line.find('[8]') + 7:]
535         line = line.split()
536         line = line[-1::-1]
537         data.append(''.join(line))
538     datafile.close()
539     #Falls die Stromauslese Fehlerhaft war, wird die Funktion nicht weiter ausgeführt.
540     if len(data) != 3:
541         return [0,0,0]
542     #Umwandeln der Stromdaten in Dezimalzahlen.
543     data[0] = int(data[0], 16)
544     data[1] = int(data[1][6:], 16)
545     data[2] = int(data[2][6:], 16)
546     return data
547 
548 #Funktion zum Berechnen des Mittleren Stroms, der gesammelten Ladung,der Zeit amn QM07, sowie des Elastische Linie zuu
        Ladung Verhältnisses.
549 def CurrentAndCharge():
550     #Umwandeln der Startwerte der Stromdaten ins Dezimale.
551     startvalues = HexToDec('currentdatastart.dat')
552     #Falls nicht möglich, wird die Funktion nicht weiter ausgeführt.
553     if startvalues == [0,0,0]:
554         return (0,0,0)
555     #Umwandeln der Endwerte der Stromdaten ins Dezimale.
556     endvalues = HexToDec('currentdataend.dat')
557     #Falls nicht möglich, wird die Funktion nicht weiter ausgeführt.
558     if endvalues == [0,0,0]:
559         return (0,0,0)
560     currentsum = endvalues[0]-startvalues[0]
561     cycle = endvalues[1]-startvalues[1]
562     timediff = endvalues[2]-startvalues[2]
563     #Berechnen der QM07-Zeit.
564     time = timediff/(1000*60.)
565     #Berechnen des Mittleren Stroms.
566     averagecurrent = (currentsum*256./cycle)*(101.6/(2**28))
567     #Berechnen der gesammelten Ladung.
568     charge = currentsum*256*((101.6*10**-6)/(2**28))*(time*60./cycle)*1000
569     #Berechnen des Verhältnisses zwischen elastischer Linie und gesammelter Ladung (normalized elastic countrate).
570     counts = open('rate_conv_out.dat','r').read()
571     ratio = abs(float(counts[counts.find('HitsA + HitsB =') + 15:counts.find('\n', counts.find('HitsA + HitsB ='))])/
            charge)
572     #Abspeichern der berechneten Daten.
```

```
573     open('rate_conv_out.dat','a').write('\nAverage current [microA] = %3.10f\nCollected charge [mC] = %3.10f\nQM07-Time
            [min] = %3.10f\nNormalised countrate [Counts/mC]= %3.10f\n' % (averagecurrent, charge, time, ratio))
574     #Rückgabe der berechnenten Daten.
575     return averagecurrent, charge, time
576 
577 #Funktion zur Überprüfung, ob das Spektrometer läuft.
578 def OnlineStatus():
579     if open(os.path.dirname(__file__) + '/Lintottdata/LintottAccess.dat','r').readline() == 'True':
580         return
581     open(os.path.dirname(__file__) + '/Lintottdata/LintottAccess.dat','w').write('True')
582     try:
583         tn = telnetlib.Telnet('lintott_r')
584     except EnvironmentError:
585         time.sleep(5)
586         tn = telnetlib.Telnet('lintott_r')
587         open(os.path.dirname(__file__) + '/Lintottdata/LintottAccess.dat','w').write('False')
588         return
589     tn.read_until('Username: ')
590     tn.write('burda\n')
591     tn.read_until('Password: ')
592     tn.write('lintott\n')
593     tn.write('getallcnts(0);\n')
594     time.sleep(0.2)
595     tn.write('getallcnts(0);\n')
596     time.sleep(0.2)
597     tn.write('exit\n')
598     time.sleep(0.1)
599     data = tn.read_very_eager()
600     open(os.path.dirname(__file__) + '/Lintottdata/StatusLogFile.dat','w').write(data)
601     #Heraussuchen der ersten Zeit.
602     time1 = data[data.find('On-Time-Armed:') + 17:data.find('.', data.find('On-Time-Armed:'))]
603     #Heraussuchen der zweiten Zeit.
604     time2 = data[data.find('On-Time-Armed:', data.find('On-Time-Armed:') + 1) + 17:data.find('.', data.find('On-Time-
            Armed:', data.find('On-Time-Armed:')+1))]
605     tn.close()
606     open(os.path.dirname(__file__) + '/Lintottdata/LintottAccess.dat','w').write('False')
607     #Falls die Zeiten nicht übereinstimmen, läuft das Spektrometer.
608     if time1 != time2:
609         open(os.path.dirname(__file__) + '/Lintottdata/LintottRunning.dat','w').write('True')
610         return True
611     else:
612         open(os.path.dirname(__file__) + '/Lintottdata/LintottRunning.dat','w').write('False')
613         return False
614 
615 #Funktion zur Überprüfung des Lintottstatus.
616 def Status(args):
617     status = OnlineStatus()
618     if status == True:
619         print '\nLINTOTT is online. Measurement is in process.\n'
620     elif status == False:
621         print '\nLINTOTT is offline. No measurement is in process.\n'
622     else:
623         print 'Connecting to LINTOTT was not possible. Please try again in a few seconds.'
624 
625 #Funktion zum Laden eines oder mehrerer Spektren, sowie Setzen der richtigen Fitfunktion.
626 def GetSpec(filename):
627     try:
628         files = []
629         #Setzen der Fitfunktion.
630         Lintott('')
631         i = 0
632         #Konvertieren der Dateien, falls nötig (Siehe Funktion "ConvertFile()") und laden des Spektrums bzw. der
                Spektren.
633         while i < len(filename):
634             files.append(ConvertFile(filename[i]))
635             hdtv.cmdline.ExecCommand("spectrum get " + files[i] + "'col")
```

```
636             i += 1
637         #Löschen der entstandenen temporären Dateien.
638         DeleteTempFiles(files)
639     except IOError:
640         print 'There is no such file.'
641 
642 #Funktion zur Ausgabe eines Graphen im eps-, svg- oder png-Format.
643 def FormatGraph(filename):
644     files = []
645     try:
646         #Abfragen des gewünschten Formats.
647         format = raw_input("Please choose the format in which you want to save the graph.\nPossible formats are: 1-
                postscript, 2-svg, 3-png.\n")
648         #Setzen des gewünschten Formats und der entsprechenden Dateierweiterung.
649         if format in '2-svg' and format != '':
650             format = 'svg'
651             extension ='.svg'
652         elif format in '3-png' and format != '':
653             format = 'png enhanced'
654             extension = '.png'
655         elif format in '1-postscript' and format != '':
656             format = 'postscript color solid'
657             extension = '.eps'
658         else:
659             #Falls das Format nicht erkannt wird, wird 'postscript' ausgewählt.
660             print "Format not recognised. Set format automatically to 'postscript'.\n"
661             format = 'postscript color solid'
662             extension = '.eps'
663         #Abfrage des gewünschten Dateinamens.
664         newname = raw_input("Please enter a filename to save the graph.\n")
665         #Falls kein Name eingegeben wird, wird ein Standartname mit Zeitinfos generiert.
666         if newname == '':
667             t = time.localtime()
668             newname = '%04i-%02i-%02i_%02i-%02i-%02i_graph' % (t[0], t[1], t[2], t[3], t[4], t[5])
669         print 'Writing...'
670         #Konvertieren der Datei, falls nötig.
671         specfiles = ConvertFile(filename[0])
672         files.append(specfiles)
673         i = 1
674         #Falls mehrere Dateien eingegeben wurden, werden diese ebenfalls konvertiert.
675         while i < len(filename):
676             specfiles = specfiles + "' with histeps, '" + ConvertFile(filename[i])
677             files.append(ConvertFile(filename[i]))
678             i += 1
679         #Überprüfung, ob eine Dateierweiterung eingegeben wurde.
680         if newname.find('.', -5) != -1:
681             extension = ''
682         script = open('plot.tmp','w')
683         script.write(
684         #Gnuplot-Code.
685         "unset key\n\
686         set samples 10000,10000\n\
687         set xlabel 'channel'\n\
688         set ylabel 'counts'\n\
689         set terminal %s\n\
690         set out '%s%s'\n\
691         plot '%s' with histeps %s"\
692         % (format, newname, extension, specfiles, FitReadOut(newname)))
693         script.close()
694         #Erstellen des Graphen.
695         os.system('gnuplot plot.tmp')
696         os.remove('plot.tmp')
697         DeleteTempFiles(files)
698         print "Wrote graph to %s%s" % (newname, extension)
699     except EOFError:
700         return
```

```
701
702 #Funktion zum Auslesen der Fitparameter und Ausgabe der Fitformel im gnuplot-Format.
703 def FitReadOut(filename):
704     #Verankern des Fits.
705     hdtv.cmdline.ExecCommand("fit store")
706     #Speichern der Parameter in einer Datei.
707     hdtv.cmdline.ExecCommand("fit write %s.xml" % filename)
708     #Definition der Fitfunktion für Elektronenstreuung und Setzen der Variablen.
709     fitformula = '((x<%(x0)s) ? %(y0)s*exp(-log(2)*(x-%(x0)s)**2/%(sigma1)s**2) : (x>%(x0)s & x<=(%(x0)s+%(eta)s*%(
            sigma2)s)) ? %(y0)s*exp(-log(2)*(x-%(x0)s)**2/%(sigma2)s**2) : %(y0)s*(2**(-%(eta)s*%(eta)s)*(((%(sigma2)s*%(
            gamma)s-2*%(sigma2)s*%(eta)s**2*log(2))/(2*%(eta)s*log(2)))+%(sigma2)s*%(eta)s)**%(gamma)s)/(((%(sigma2)s*%(
            gamma)s-2*%(sigma2)s*%(eta)s**2*log(2))/(2*%(eta)s*log(2)))+x-%(x0)s)**%(gamma)s)'
710     #Setzen der Variablenanfangswerte.
711     plotformula = ''
712     parameters = []
713     peakpass = False
714     uncalpass = False
715     #Heraussuchen der Fitparameter.
716     try:
717         for line in open(filename + '.xml','r'):
718             #Entfernen der Leerzeichen.
719             line = line.strip()
720             #Suche des Peaks.
721             if line.find('<peak>') != -1:
722                 peakpass = True
723             #Suche der nicht kalibrierten Fitparameter.
724             if line.find('<uncal>') != -1 and peakpass == True:
725                 uncalpass = True
726             #Falls die Stelle gefunden wird, werden die Parameter ausgelesen.
727             if peakpass == True and uncalpass == True:
728                 if line.find('<value>') != -1:
729                     parameters.append(line[line.find('>') + 1:line.find('<', 1)])
730             #Abschluss des ersten Peaks.
731             if line.find('</peak>') != -1:
732                 peakpass = False
733                 #Einsetzten der Fitparameter in die Fitfunktion.
734                 plotformula = plotformula + fitformula % {'x0':parameters[1], 'y0':parameters[3], 'sigma1':parameters
                        [4], 'sigma2':parameters[6], 'eta':parameters[2], 'gamma':parameters[5]} + ' + '
735                 parameters = []
736             #Abschluss der nicht kalibrierten Parameter.
737             if line.find('</uncal>') != -1 and peakpass == True:
738                 uncalpass = False
739         #Rückgabe der Fitfunktion.
740         return ', ' + plotformula[:-3] + ' lt 3'
741     except IOError:
742         #Falls nicht gefittet wurde, wird keine Fitfunktion zurückgegeben.
743         return ''
744
745 #Funktion zum Löschen temporärer Dateien.
746 def DeleteTempFiles(files):
747     i = 0
748     #Dursuchen der Dateiliste nach Dateien mit .tmp Dateierweiterung und Löschen dieser bei Fund.
749     while i < len(files):
750         if files[i].find('.tmp') != -1:
751             os.remove(files[i])
752         i += 1
753
754 #Funktion zum Konvertieren der Spektren in ein Format, welches HDTV richtig einliest.
755 def ConvertFile(filename):
756     file = open(filename, 'r+')
757     tfile = file.read()
758     file.close()
759     #Suche der ersten Lücke (von drei).
760     firstgapstart = tfile.find('97.00')
761     firstgapend = tfile.find('107.50')
762     #Falls keine Lücke gefunden wird, muss die Datei nicht konvertiert werden.
```

```
763     if firstgapstart == -1:
764         return filename
765     else:
766         #Ersetzten der ersten Lücke durch HDTV-konforme Lücke.
767         tfile = tfile.replace(tfile[firstgapstart:firstgapend-1], '101.75\t0.00')
768         secondgapstart = tfile.find('203.50')
769         secondgapend = tfile.find('214.00')
770         tfile = tfile.replace(tfile[secondgapstart:secondgapend-1], '208.25\t0.00')
771         thirdgapstart = tfile.find('310.00')
772         thirdgapend = tfile.find('320.50')
773         tfile = tfile.replace(tfile[thirdgapstart:thirdgapend-1], '314.75\t0.00')
774         #Suche der Dateierweiterung.
775         extension = filename.find('.',-5)
776         #Falls keine gefunden wird, wird .tmp angehängt.
777         if  extension == -1:
778             newname = filename + ".tmp"
779         #Ansonsten wird die Dateierweiterung durch .tmp ersetzt.
780         else:
781             newname = filename.replace(filename[extension:],'.tmp')
782         open(newname,'w').write(tfile)
783         return newname
784 
785 #Definition einer neuen Funktion für das Beenden des Programms.
786 def Exit(args):
787     #Falls die Autosavefunktion aktiv ist wird diese nach dem Beenden von HDTV deaktiviert und der neue Status in einer
          Datei abgespeichert.
788     if autoSaveStatus == True:
789         open(os.path.dirname(__file__) + '/Lintottdata/AutoSaveStatus.dat','w').write('False')
790     #Beenden von HDTV
791     hdtv.cmdline.ExecCommand("quit")
792 
793 #Funktion zur Ausgabe eines Hinweises an Benutzer, die das Programm nicht als "experiment" oder "root" gestartet haben.
794 def Message(args):
795     print '\nPlease start hdtv as "experiment" or "root" to use LINTOTT commands.\n'
796 
797 print "loaded LINTOTT plugin"
798 
799 
800 #Überprüfung des Benutzers und des LINTOTT-Status.
801 if os.getlogin() == 'experiment' or os.getlogin() == 'root':
802     try:
803         OnlineStatus()
804         #Überprüfen, ob die Autosavefunktion schon in einem anderen HDTV-Programm läuft. Falls ja, wird sie nicht
              nochmal gestartet.
805         if open(os.path.dirname(__file__) + '/Lintottdata/AutoSaveStatus.dat','r').readline() == 'False':
806             #Prozess zum automatischen Abspeichern.
807             thread = threading.Thread(target=AutoSave)
808             #Setzen des Prozesses als Nebenprozess. Dies bewirkt, dass der Prozess beim Beenden von HDTV ebenfalls
                  beendet wird.
809             thread.setDaemon(True)
810             #Starten des Prozesses zur automatischen Abspeicherung.
811             thread.start()
812             autoSaveStatus = True
813             open(os.path.dirname(__file__) + '/Lintottdata/AutoSaveStatus.dat','w').write('True')
814         else:
815             autoSaveStatus = False
816         #Hinzufügen  der neuen Kommandos.
817         hdtv.cmdline.AddCommand("lintott", Lintott, nargs=0)
818         hdtv.cmdline.AddCommand("lintott load", GetSpec, minargs=1, fileargs=True)
819         hdtv.cmdline.AddCommand("lintott graph", FormatGraph, minargs=1, fileargs=True)
820         hdtv.cmdline.AddCommand("lintott start", Start, minargs=0)
821         hdtv.cmdline.AddCommand("lintott update", Update, minargs=0)
822         hdtv.cmdline.AddCommand("lintott stop", Stop, minargs=0)
823         hdtv.cmdline.AddCommand("lintott status", Status, nargs=0)
824         #Neudefinition des Kommandos zum Beenden des Programms.
825         hdtv.cmdline.RemoveCommand("exit")
```

```
826         hdtv.cmdline.AddCommand("exit", Exit, nargs=0)
827     except socket.error:
828         print "\n*****NETWORK ERROR*****NETWORK ERROR*****NETWORK ERROR*****NETWORK ERROR****\n"
829         print "Connection to LINTOTT can't be established. Try to reset the readout module."
830         print "\n*****NETWORK ERROR*****NETWORK ERROR*****NETWORK ERROR*****NETWORK ERROR****\n"
831 
832 else:
833     hdtv.cmdline.AddCommand("lintott", Message)
```

Literaturverzeichnis

[1] Oleksiy Burda. *Nature of Mixed-Symmetry 2^+ States in ^{94}Mo from High-Resolution Electron and Proton Scattering and Line Shape of the First Excited $1/2^+$ State in ^{9}Be*. Dissertation, TU Darmstadt, 2007. D 17.

[2] S-DALINAC, Aufruf am: 08.03.2012. `http://www.ikp.tu-darmstadt.de/sdalinac_ikp/index.de.jsp`.

[3] Yuliya Fritzsche. *Aufbau und Inbetriebnahme einer Quelle polarisierter Elektronen am supraleitenden Darmstädter Elektronenlinearbeschleuniger S-DALINAC*. Dissertation, TU Darmstadt, 2011. D 17.

[4] Th. Walcher et al. Nuclear instruments and methods **153** (1978) 17.

[5] Alexander Wilhelm Lenhardt. *Entwicklung eines Si-Mikrostreifendetektors für das 169°-Spektrometer am S-DALINAC*. Dissertation, TU Darmstadt, 2004. D 17.

[6] Oleksiy Burda und Andreas Krugmann. *Performing Electron Scattering Experiments at the LINTOTT Spectrometer at the S-DALINAC*, 2009.

[7] F. Hofmann et al. Physical Review **C 65** (2002) 024311.

[8] Asim Araz. *Aufbau und Erprobung einer digitalen HF-Regelung und Aufbau eines modularen Messsystems zur Energiestabilisierung für den S-DALINAC*. Dissertation, TU Darmstadt, 2009. D 17.

[9] QM07-Elektronik, Aufruf am: 13.12.2011. `http://ikpweb.ikp.physik.tu-darmstadt.de/mediawiki/index.php/QM07-ELEKTRONIK`.

[10] Uwe Bonnes. Serielle CAN-Adapter, 2011. Mündliche Mitteilung.

[11] CAN PCI Interfaces, Aufruf am: 25.02.2012. `http://www.computer-solutions.co.uk/gendev/can-pci.htm`.

[12] Ulrich Tietze und Christoph Schenk. *Halbleiter-Schaltungstechnik*. Springer, 2002.

[13] HDTV-Programm, Aufruf am: 04.01.2012. `https://www.ikp.uni-koeln.de/projects/hdtv/trac`.

[14] Python, Aufruf am: 03.12.2011. `http://www.python.org`.

[15] C++, Aufruf am: 05.12.2011. `http://www.cplusplus.com`.

[16] Experimental Physics and Industrial Control System, Aufruf am: 09.03.2012. http://www.aps.anl.gov/epics.

[17] EPICS System, Aufruf am: 09.03.2012. http://ikpweb.ikp.physik.tu-darmstadt.de/mediawiki/index.php/EPICS.

[18] CAN-Treiber, Aufruf am: 14.12.2011. http://www.peak-system.com/fileadmin/media/linux/index.htm.

Danksagung

Zuallererst möchte ich mich bei Herrn Prof. Dr. Peter von Neumann-Cosel bedanken, der mir die Gelegenheit gab an einem so interessanten Thema zu arbeiten. Ich bedanke mich bei meinem Betreuer Andreas Krugmann, M. Sc., der mich in allen Angelegenheiten tatkräftig unterstützt hat. Weiterhin bedanke ich mich bei Christoph Burandt, M. Sc., der mich in Sachen QM07-Software ausführlich beraten hat. Insbesondere möchte ich mich auch bei Herrn Dipl.-Ing. U. Bonnes bedanken, der mir bei der QM07-Elektronik sehr weiter geholfen hat. Ich möchte mich auch bei meinen Bürokollegen bedanken, die mich sowohl unterstützt, als auch zu einem angenehmen Arbeitsklima beigetragen haben. Schließlich möchte ich mich bei meiner Familie bedanken, ohne deren Unterstützung diese Arbeit nicht möglich gewesen wäre.

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 29. August 2012

(Sergej Bassauer)