Fakultät für Physik und Astronomie

Ruprecht-Karls-Universität Heidelberg

Masterarbeit

Im Studiengang Physik

vorgelegt von

Daniel Hollain

geboren in Heidelberg

2014

Vermessung vom hochgeladenen Iridium mithilfe eines neuen Kontrollsystems

Die Masterarbeit wurde von Daniel Hollain

ausgeführt am

Max-Planck-Institut für Kernphysik

unter der Betreuung von

Priv.-Doz. Dr. José Ramón Crespo López-Urrutia

Vermessung vom hochgeladenen Iridium mithilfe eines neuen Kontrollsystems

Im Rahmen dieser Arbeit wurde ein Kontrollsystem für die Heidelberg-Elektronenstrahl-Ionenfalle auf der Grundlage von EPICS entworfen. Mit Hilfe dieses Kontrollsystems wurden Rekombinations- und Anregungsenergien von hochgeladen Iridiumionen (Ordnungszahl = 77) untersucht. Diese wurden in der Falle mittels energiereiche Elektronen sequentiell erzeugt. Simultan dabei rekombinieren freie Elektronen mit den Ionen sowohl resonant als auch nicht resonant. Die bei diesen Prozessen emittierten Photonen wurden für den Nachweis registriert, und ihre Energie bestimmt. Mit Hilfe der Röntgen-Absorptionskanten von Metallfolien wurden die radiative Rekombinationsenergien und Ionisationspotentiale ebenso bestimmt. Zusätzlich wurden von den dielektronischen KLL-Resonanzen der Helium- bis Sauerstoffartigen Iridiumionen die Photonen- und die jeweilige Anregungsenergie vermessen. Die Ergebnisse wurden mit theoretischen Vorhersagen verglichen. Die Photonenenergien liegen bei ungefähr 65 keV mit einer typischen Unsicherheit von 85 eV. Die Anregungsenergien um 45 keV zeigen dank eines hochpräzisen Spannungsteilers statistische Fehler von nur wenigen eV, aber eine systematische Verschiebung gegenüber der Theorie von etwa 20 bis 30 eV. Bei einigen Resonanzen sind asymmetrische Fano-Linienprofile erkennbar, die auf die Quanteninterferenz von radiativen und dielektronischen Prozessen zurückführbar sind.

Measurement of highly charged Iridium with a new control system

During the course of this thesis a control system based on EPICS was developed for the Heidelberg electron beam ion trap. This system was employed to investigate recombination and excitation processes in highly charged iridium (atomic number = 77) ions. Iridium atoms were ionized sequentially in the trap by high energy electrons. Free electrons simultaneously recombine with the ions via resonant and non-resonant channels. These recombination processes were observed by registering the emitted photons and measuring their energies. By exploiting the x-ray absorption edges of metal foils recombination energies and ionization potentials were obtained by observing the radiative recombination. Furthermore, dielectronic KLL resonances from helium-like to oxygen-like ions were observed and their photon and excitation energies determined. These energies were compared with theoretical predictions. The photon energies are in magnitude of 65 keV with a typical uncertainty of 85 eV. The excitation energy determination at ca. 45 keV has uncertainties of only a few eV due to a highly precise voltage divider, and a systematic shift with respect to theory of 20 to 30 eV. Some resonances show asymmetric Fano profiles originating from interferences of radiative and dielectronic processes.

Inhaltsverzeichnis

1 Einführung

Spektroskopie ist ein Hauptpfeiler der modernen Physik. Sie zielt darauf hin, charakteristische Energiemuster einer Probe zu untersuchen die Aufschlüsse über ihr Innenleben enthält. Mithilfe der Theorie lassen sich solche Spektren näherungsweise berechnen, jedoch tauchen mit steigender Genauigkeit und Präzision immer neue Effekte auf, wie Feinstrukturaufspaltung, Hyperfeinstrukturaufspaltung oder Lamb-Verschiebung. Die Wissenschaftler Kirchhoff und Bunsen haben in 19. Jahrhundert charakteristische Spektren über ein Prisma in einem Fernrohr mit einer Skala abgelesen, während im 21. Jahrhundert optische Uhren imstande sind, mit 17 Dezimalstellen Genauigkeit die zeitliche Stabilität der Naturkonstanten zu testen. Für ausreichende Präzision werden entsprechend lange Messzeiten sowie komplexe Abläufe benötigt. Zur Realisierung solcher Experimente benötigt man immer leistungsfähigere und flexiblere Kontrollsysteme. Die zur Verfügung stehende Zeit muss dabei ebenso berücksichtigt werden.

In dieser Arbeit werden Rekombinations- und Anregungsenergien hochgeladenes Iridium, welches eine Ordnungszahl von 77 hat, untersucht. Bei dieser Ordnungszahl beträgt die Feinstrukturaufspaltung zwischen den $2p_{1/2}$ und $2p_{3/2}$ Niveaus etwa 2 keV. Durch die vorliegende Messung wurden Ionisationspotentiale für Li- bis Ne-artige Ladungszustände bestimmt, sowie die Anregungsenergien der wichtigsten dielektronischen KLL Resonanzen. Durch die Größe dieser Effekte ist es möglich, mit einem Elektronenstrahl bekannter Energie einige neue Datenpunkte für diese bisher wenig untersuchte Ionenspezies zu gewinnen.

Zur Realisierung wird die Heidelberg-Elektronenstrahl-Ionenfalle des Max-Planck-Instituts für Kernphysik in Heidelberg verwendet, kurz HD-EBIT. Die theoretischen Grundlagen dazu werden in Kapitel 2 übermittelt. Kapitel 3 beschäftigt sich mit der HD-EBIT und wie sie das Energiespektrum der Ionen aufnimmt.

Die Softwareplattform EPICS (*Experimantal Physics and Industrial Control System*) hat eine breite Anwendung in Großforschungseinrichtungen gefunden, die sie die Möglichkeit bietet, Kontroll- und Messaufgaben hoher Komplexität zu integrieren. In dieser Arbeit wurde Software entwickelt, die auf EPICS basiert, um Messaufgaben in der HD-EBIT zu erleichtern und bessere Messungen zu ermöglichen. Ein großer Vorteil ist der, dass sich Aufgaben auf mehrere Computer verteilen lassen, welche über das lokale Netzwerk miteinander kommunizieren. Mehrere Programme auf unterschiedlichen Standorten können die HD-EBIT überwachen und sämtliche Parameter mitprotokollieren. Der Gefahr des Datenverlustes kann somit reduziert werden. Dass sich EPICS nicht nur auf eine Programmiersprache beschränkt, ist ebenfalls von Vorteil. In dieser Arbeit wird Python2 verwendet.

Für die Architektur des Kontrollsystems werden verschiedene Serverebenen ge-

nutzt, um Aufgaben auf diversen Abstraktionsniveaus angehen zu können. Dies erleichtert ebenfalls die Handhabung des Experiments, vor allem in Test- und Konfigurationsphasen.

Eine Messung von Rekombinationsenergien von hochgeladenen Iridium mit Auswertung steht in Kapitel 5, Kapitel 6 enthält noch eine kurze Zusammenfassung dieser Arbeit mit Ausblick. Im Anhang sind hauptsächlich Programmskripte zum Kontrollsystem zu finden.

2 Theoretische Grundlagen

2.1 Struktur von Atome und Ionen

Die experimentelle Grundlage der Atomphysik ist die Spektroskopie. Mit ihr lassen sich Atomstrukturen herleiten. Zuerst wurde Sie für chemische Elemente von Kirchhoff und Bunsen im Jahr 1859 angewandt. Die klassischen Theorien reichten mit zunehmender Präzision bald nicht mehr aus, um die beobachteten Phänomene zu beschreiben. Beim Bohrschen Atommodell ist die Kreisbahn der Elektronen um den Kern durch die De Broglie Wellenlänge beschränkt, in der Quantenmechanik sind die klassischen Observablen, wie Ort oder Impuls, durch Operatoren für sogenannte Wellenfunktionen, deren Betragsquadrat die Wahrscheinlichkeitsdichte des Systems entspricht, ersetzt.

Im folgenden wird das Wasserstoffatom beschrieben, da es das einfachste Atom, relativ leicht lösbar ist und der Rechenweg sich auf weitere Atome störungstheoretisch übertragen lässt. Ausgangspunkt ist die Schrödingergleichung [I. V. Hertel, C.-P. Schulz, 2008]:

$$\hat{H}\Psi(\vec{r},t) = i\hbar\partial_t\Psi(\vec{r},t) \tag{2.1}$$

mit dem Hamiltonian $\hat{H} = \frac{-\hat{p}^2}{2m} + V$, Impulsoperator $\hat{\vec{p}} = -i\hbar\nabla$, Elektronenmasse $m \approx 9.11 \times 10^{-31} kg$, Abstand vom Kern r, Ladung des baren Kerns $Z \times e$, Elementarladung $e \approx 1.6 \times 10^{-19} C$, Dielektrizitätskonstante $\varepsilon_0 \approx 8.85 \times 10^{-12} \frac{As}{Vm}$ und Coulombpotential des Kerns $V = -\frac{1}{4\pi\varepsilon_0}\frac{Ze^2}{r}$ wird daraus:

$$\hat{H} = \frac{-\hbar^2}{2m} - \frac{1}{4\pi\varepsilon_0}\frac{Ze^2}{r} \tag{2.2}$$

Mit den Separationsansatz $\Psi(\vec{r},t) = h(\vec{r})g(t)$ lässt sich die Gleichung in zeit- und ortsabhängigen Teil auflösen:

$$\frac{\hat{H}h(\vec{r})}{h(\vec{r})} = \frac{i\hbar\partial_t g(t)}{g(t)} = W \tag{2.3}$$

wobei W konstant, da Ort und Zeit jeweils auf der linken und rechten Seite der Gleichung stehen und unabhängig voneinander gewählt werden können, und die Energie des Systems ist. Daraus ergeben sich die Gleichungen:

$$\hat{H}h(\vec{r}) = Wh(\vec{r}) \tag{2.4}$$

und

$$i\hbar\partial_t g(t) = Wg(t) \tag{2.5}$$

Ihre Lösungen lauten:

$$g(t) = C \times exp(\frac{W}{i\hbar}t) \tag{2.6}$$

und

$$h_{nlm}(\vec{r}) = h_{nlm}(r, \theta, \varphi) = R_{nl}(r)Y_{lm}(\theta, \varphi) \tag{2.7}$$

mit der Radialfunktionen

$$R_{nl} = C_{nl}e^{-\rho/2}\rho^l L_{n+l}^{2l+1}(\rho) \tag{2.8}$$

$$C_{nl} = -(\frac{Z}{a_0})^{3/2}\frac{2}{n^2}\sqrt{\frac{(n-l-1)!}{((n+l)!)^3}} \tag{2.9}$$

und $\rho = \frac{2Z}{n}\frac{r}{a_0}$, $a_0 = \frac{\varepsilon_0 h^2}{e^2 m\pi} \approx 0.529 \times 10^{-10}m$ als Länge für atomare Einheiten (Bohrscher Radius) und $h \approx 6.61261 \times 10^{-34}Js$ als Planck'sches Wirkungsquantum. $L_{n+l}^{2l+1}(\rho)$ bezeichnet die assoziierten Laguerre-Polynome, $Y_{lm}(\theta, \varphi)$ die Kugelflächenfunktionen. Die Konstanten n, l und m sind die Quantenzahlen des Systems. $h_{nlm}(\vec{r})$ erfüllt die Eigenwertgleichung

$$\hat{H}h_{nlm}(\vec{r}) = W_{nlm}h_{nlm}(\vec{r}) \tag{2.10}$$

ist somit eine Eigenfunktion vom Hamiltonian mit diskreten Eigenenergiewerte

$$W = W_{nlm} = -\frac{Z^2}{2n^2}\frac{me^4}{4\varepsilon_0^2 h^2} \tag{2.11}$$

Wie beim Wasserstoffatom sind auch die Energiewerte der gefangenen Elektronen von anderen Atomen oder Ionen diskret. Zudem kommen noch weitere Effekte und Quantenzahlen, welche die Niveaus verschieben oder aufspalten. Zur Ionisation muss dem gebundenen Elektron mindestens seine Bindungsenergie hinzugefügt werden, damit es das das Atom verlassen kann. Andererseits gibt ein freies Elektron Energie ab, wenn es eingefangen wird. In dieser Arbeit betrachten wir die Photonen, die von Elektronen abgesendet werden, welche eingefangen werden.

2.2 Stoßprozesse

Atome und Ionen können Energie auf- bzw. abgeben. Dies geschieht durch Stoßprozesse mit anderen Teilchen, wie Photonen oder Elektronen. Bei der Anregung wird einem gebundenen Elektron durch einen Stoß Energie zugefügt, sodass es in ein höheres Niveau angehoben wird, aber immer noch gebunden bleibt. Fällt es danach in ein niedrigeres Niveau, sendet es dabei ein Photon ab oder regt ein anderes Elektron an. Bei der Ionisation hingegen bekommt das gestoßene Elektron mehr Energie als dessen Bindungsenergie ab, so dass es nicht mehr gebunden ist und frei wird. Die resultierende kinetische Energie entspricht der Differenz aus Stoßenergie und Bindungsenergie des Niveaus, aus den das Elektron gestoßen wurde.

Prozess	**Formel**
Photoionisation	$X^{q+} + \gamma \rightarrow X^{(q+1)+} + e$
Photo- oder Radiative Rekombination (RR)	$X^{(q+1)+} + e \rightarrow X^{q+} + \gamma$
Multielektronen Photoionisation	$X^{q+} + \gamma \rightarrow X^{(q+k)+} + ke, k \geq 1$
Bremsstrahlung	$X^{q+} + e \rightarrow X^{q+} + e + \gamma$
Dielektronische Rekombination (DR)	$X^{q+} + e \rightarrow X^{(q-1)+**} \rightarrow \begin{cases} X^{(q-1)+*} + \gamma \\ X^{q+} + e \end{cases}$

Tabelle 2.1: Diverse Stoßprozesse [H. F. Beyer, H.-J. Kluge, V. P. Shevelko, 1997] (X: Atomkern, q: Ladung, e: freies Elektron, γ: Photon)

Zur Ionisation gibt es noch reverse Prozesse, die Rekombination. Ein freies Elektron wird eingefangen, somit gebunden, und strahlt Energie in Form eines Photons ab oder regt andere Elektronen an. Bei der radiativen Rekombination (RR) fällt ein freies Elektron in ein Energieniveau des Ions und strahlt ein Photon ab, dessen Energie der kinetischen Energie plus der Bindungsenergie entspricht. Sie ist nicht resonant. Die dielektronische Rekombination (DR) hingegen beschreibt ein Prozess, bei der auch ein freies Elektron eingefangen wird, dessen abgegebene Energie jedoch ein schon gebundenes Elektron in ein höheres Niveau anregt. Dieses angeregte Elektron fällt kurz danach (lange bevor ein weiteres freies Elektron eintrifft [Crespo14]) in ein tieferes Niveau und strahlt ein Photon mit bestimmte Energie ab, sie entspricht der Differenz der Niveaus. Dieser Prozess ist resonant, Gleichung 2.12 zeigt die Resonanzbedingung, und der Wirkungsquerschnitt dafür steigt im dem Bereich gegenüber der RR an. Die Energie des abgestrahlten Photons entspricht der Bindungsenergie (positiv, Bindungspotential wäre vom gleichen Betrag jedoch negativ) vom Niveau, in der das freie Elektron rekombiniert ist plus der kinetischen Energie des freien Elektrons. Diese Resonanzpunkte sind, wie auch die RR-Linien, mit Breiten versehen, welche aus natürlicher und experimenteller Linienbreite stammen. Tabelle 2.1 listet einige Stoßprozesse auf.

$$kinetischeEnergie = Photonenenergie - Bindungsenergie \tag{2.12}$$

2.3 Radiative und dielektronische Rekombination

Die radiativen und dielektronischen Rekombinationen sind anhand von Linien bzw. Punkte in einem Photonenenergie-Elektronenstrahlenergie-Diagramm erkennbar. Die Steigung radiativer Linien sollte sich zu 1.0 ergeben, da für jedes keV kinetische Energie mehr das Photon ebenfalls 1keV mehr bekommt. Wird diese Gerade auf eine Elektronenstrahlenergie von 0 keV extrapoliert, bleibt die Bindungsenergie auf den Achsenabschnitt zurück. Da einige Linien mit gleichen Drehimpuls J nahe beieinander liegen uns sich durch ihre Breiten überlappen, entstehen je nach Drehimpuls des

Ions (=J) verschieden erkennbare radiative Rekombinationsbänder. In Abbildung 2.1 sind die Bänder J=1/2 (oberes) und J=3/2 (unteres) zu sehen. Die Bindungsenergie von J=1/2 ist größer, dieses Band eine höhere Photonenenergie als mit J=3/2.

Die dielektronischen Resonanzen sind vor allem bei der oben beschriebenen Resonanzbedingung aktiv. In Abbildung 2.1 sind diverse *Blobs*, engl. für Blasen, zu sehen. Hier werden nur KLL Resonanzen betrachtet. Die Bezeichnung hat drei Buchstaben, repräsentativ für Schalen. Das gebundene Elektron befindet sich anfangs in der ersten Schale (K-Schale) und wird in die zweite Schale (L-Schale) angeregt. Die dritte beschreibt die Schale (L-Schale), in der das freie Elektron rekombiniert [Bernitt, 2009]. Für KLL Resonanzen müssen somit zwei Löcher in der L-Schale , in der das freie Elektron fällt und das gebundene angeregt wird, und mindestens ein gebundenes Elektron in der K-Schale vorhanden sein. Die Ladungszustand der Ionen für KLL Prozesse reicht somit von He- bis O-artig. In der Abbildung sind drei KLL-Bereiche dargestellt ($KL_{1/2}L_{1/2}, KL_{1/2}L_3, KL_3L_3$). Sie folgen der Nomenklatur von [D. A. Knapp, P. Beiersdorfer, M. H. Chen, J. H. Scofield, and D. Schneider, 1995]. In $KL_{1/2}L_{1/2}$ geht alles von J=1/2 aus, und mit mindestens zwei Löcher sind daher nur He- bis Be-artige Resonanzen möglich. Das Element bezeichnet des Ausgangsladungszustand des Ions. Bei einer He-artigen Resonanz rekombiniert ein freies Elektron in ein He-artiges Ion zu einem angeregten Li-arigen Ion. Bei den anderen KLL-Bereichen vollzieht sich die Rekombination in das J=3/2 Niveau. Dabei gibt es zwei mögliche Abregungswege: ein gebundenes Elektron fällt entweder aus $L_{1/2}$ oder$L_{3/2}$ in die K-Schale. In Abbildung 2.1 sind in den letzten zwei KLL-Bereichen stets zwei horizontal übereinander liegende DR-Reihen zu sehen.

Unter einigen DRs sind *droplines*, engl. für Falllinien, zu sehen. Sie entstehen aus Zwei-Photonen Zerfälle [Martínez, 2005] und werden in dieser Arbeit nicht weiter behandelt.

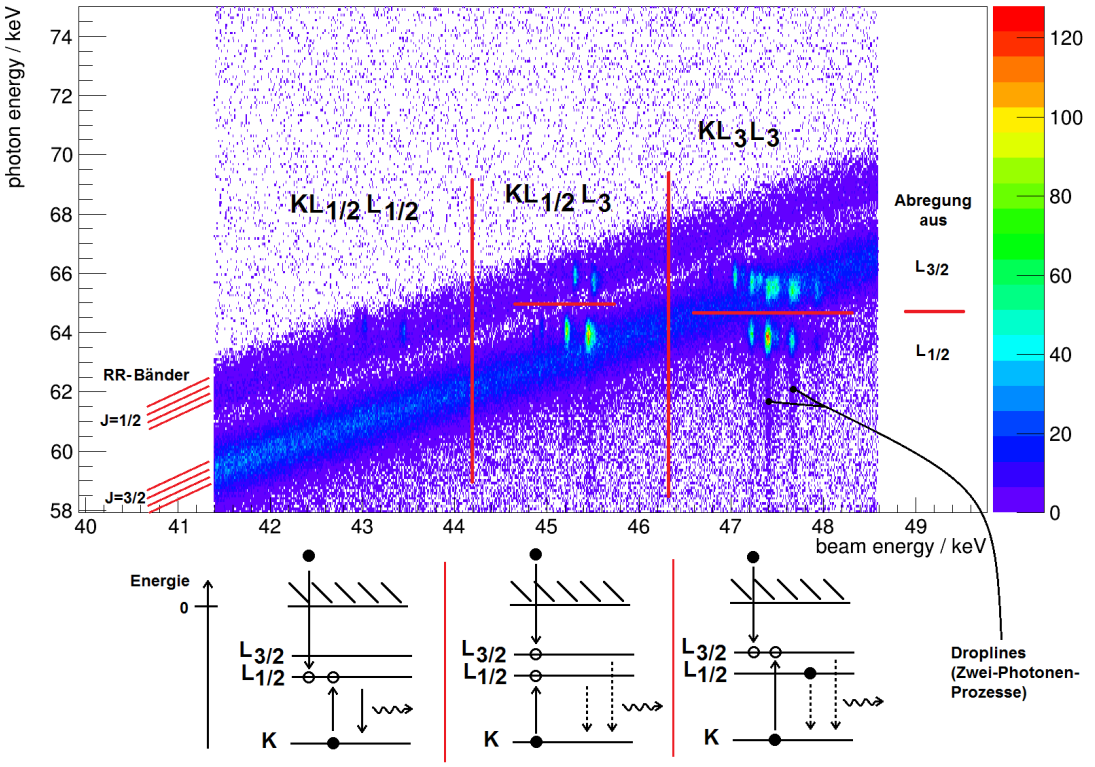

Abbildung 2.1: Radiative und dielektronische Rekombination - Diagramm. Die Elektronenstrahlenergie ist gegen die Photonenenergie aufgetragen. Die Anzahl der Ereignisse ist farbig entsprechend der rechten Skala markiert. Die steigenden Geraden entsprechen den radiativen Rekombinationsbändern und enthalten jeweils 4 einzelne radiativen Rekombinationslinien mit gleichen Drehimpuls J. Die Punkte, in denen auffällig viele Ereignisse zu sehen sind entsprechen dielektronischen Resonanzen. Sie sind in 3 Bereiche ($KL_{1/2}L_{1/2}$, $KL_{1/2}L_3$, KL_3L_3) nach der Nomenklatur von [D. A. Knapp, P. Beiersdorfer, M. H. Chen, J. H. Scofield, and D. Schneider, 1995] eingeteilt. Bei den letzten beiden Bereichen hat das angeregte Elektron zwei verschiedene Abregungswege (aus $L_{3/2}$ und $L_{1/2}$ Schale), daher sind zwei Reihen horizontaler Resonanzen übereinander. Dabei steht L für die Schale (Hauptquantenzahl=2) und der Indize (1/2 oder 3/2) für den Drehimpuls J. Unterhalb einiger starker Resonanzen sind Falllinien (Droplines) zu sehen, was auf Zwei-Photonen-Prozesse zurückzuführen ist. Die unteren Skizzen zeigen die Rekombinations- und Abregungswege der obigen Bereiche mit gleicher Reihenfolge, dabei sind unskaliert die Energieniveaus des Ions zu sehen. Von oben rekombiniert ein freies Elektron in eine L-Schale und regt ein gebundenes Elektron aus der K-Schale an. Der Zustand zerfällt nach kurzer Zeit (lange bevor ein weiteres freies Elektron eintrifft [Crespo14]) und setzt ein Photon frei, dessen Energie der Differenz der Niveaus entspricht

3 Experimenteller Aufbau

3.1 Die Elektronenstrahl-Ionenfalle

3.1.1 Grundlegendes zur Elektronenstrahl Ionenfalle

Eine Elektronenstrahl Ionenfalle, englisch *electron beam ion trap*, kurz EBIT, erzeugt und fängt Ionen. Das Kernstück ist der Elektronenstrahl, der die Falle durchquert. Er erzeugt Ionen, hält sie fest und regt sie an. Somit lässt sich Spektroskopie in der Falle betreiben.

Zuerst würde die Elektronenstrahl-Ionenquelle, englisch *electron beam ion source* (EBIS), um 1965 von *Donets* in Dubna entwickelt , um hochgeladene Ionen zu erzeugen. 1989 wurde die erste Elektronenstrahl-Ionenfalle von Levine *et al.* am Lawrence Livermore National Laboratory gebaut [Martínez, 2005].

3.1.2 Funktionsprintzip

Hochgeladene Ionen werden aus neutralen Atomen gewonnen. Dafür kann man Edelgase oder Molekülverbindungen nehmen. Ein hochenergetischer Elektronenstrahl dissoziiert Moleküle und ionisiert Atome mittels Stoßionisation (Kap. 2) zu Ionen, je nach Energie und Strom zu hochgeladenen Ionen. Freien Elektronen werden auch zur Rekombination genutzt. Es stellt sich nach bestimmter Zeit je nach Falleneinstellung eine zeitlich gemittelt konstante Ladungszustandsverteilung.

Die Ionen werden mit elektrischen Feldern in der Falle gehalten. Abbildung 3.1 zeigt das Funktionsprinzip der Elektronenstrahl Ionenfalle. Der Elektronenstrahl hat eine negative Raumladung und zieht die positiv geladenen Ionen zu ihn, sodass sie in radialer Richtung gefangen werden. Für die longitudinale Falle sorgen Drifröhren, auf denen elektrische Spannung gelegt wird. Geladene Teilchen in unterschiedlich geladenen Driftröhren haben somit auch unterschiedliches Potential. Aus diesem Prinzip lassen sich Barrieren erzeugen, die die Ionen, abgesehen vom Tunneleffekt, nicht überwinden können (siehe Abb. 3.2).

Die Temperatur der Ionenwolke lässt sich durch hauptsächlich durch den Elektronenstrahlstrom und -Energie sowie durch die Fallentiefe regulieren. In einer kühlen Ionenwolke sind die Ionen langsamer als in einer heißeren, somit ist der Dopplereffekt, der Linien verbreitert, geringer. Eine weitere Kühlungsmethode ist die evaporative Kühlung. Hierbei können schnellere Ionen der Falle entkommen und kühlen sie somit. Dazu muss die Fallentiefe entsprechend seicht gewählt werden. Ein Nachteil dieser Methode ist dass ein Teil der Ionen verschwindet und die Signalstärke der Ionenwolke sinkt, d.h. man bekommt weniger Statistik und muss z.B. länger messen.

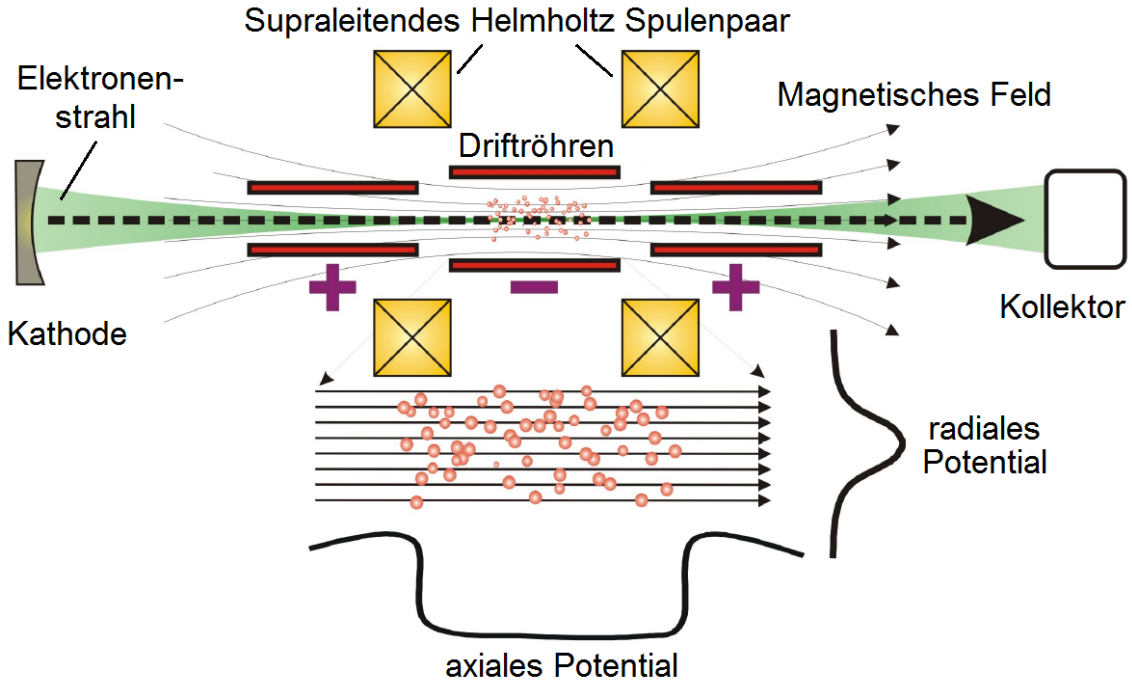

Abbildung 3.1: Funktionsprinzip der Elektronenstrahl Ionenfalle [Martínez, 2005]. Die Kathode sendet ein Elektronenstrahl aus, der die Falle durchquert und im Kollektor eintrifft. Im Fallenbereich wird er durch ein Magnetfeld, erzeugt von einem (hier supraleitenden) Helmholtz Spulenpaar, komprimiert, um möglichst hohe Stromdichten zu erreichen. Die negative Raumladung hält die positive geladenen Ionen in axialer Richtung fest. In longitudinaler Richtung werden sie durch Driftröhren festgehalten, auf denen elektrische Spannung gelegt wird und Potentialverlauf derart beeinflussen, dass ein Potentialtopf entsteht. Die Ionen stammen von neutralen Atomen, die in die Falle initiiert und sequentiell durch Stoßprozesse mit den energiereichen Elektronen des Strahls ionisiert werden.

Der Elektronenstrahl wird in einer Elektronenkanone erzeugt. Dafür wird eine spezielle Elektrode, die Kathode, mit einem Heizelement temperiert, sodass die thermische Elektronen austreten können. Die freien Elektronen werden in Richtung der Anode beschleunigt, auf der positive Spannung liegt und die Elektronen somit anzieht. Dazwischen befindet sich noch eine Fokussierelektrode, mit der man den Strom des Elektronenstrahles genauer kontrolliert, bzw. fokussiert (und nicht den Radius des Strahles). Zudem gibt es noch Bucking- und Trimmspule eingebaut; zwei Magnetspulen die äußere Magnetfelder aufheben sollen. Der Elektronenstrahl wird nach der Falle vom Kollektor aufgesammelt. Der Kollektor liegt auf dem selben Potential wie die Kanone. Die Extraktorelektrode wird auf negatives Potential höher als die Kathode gebracht, sodass die Elektronen sie nicht durchdringen können, Ionen aber schon. Die Elektronen werden somit abgebremst und treffen auf die Kollektorelektrode. Die Kollektormagnetspule schirmt auch hier äußere Magnetfelder ab. Das ganze Elektronenstrahlsystem kann noch auf Hochspannung gelegt werden um höhere kinetische Energien der Elektronen zu erreichen, ohne dass das Hochspannungsnetzteil den Elektronenstrahlstrom aufbringen muss.

Um den Radius des Elektornenstrahles im Fallenbereich bei den Drifröhren möglichst gering zu halten, befindet sich dieser im Zentrum eines Helmholtz-Spulenpaares. Dieser Magnet erzeugt in der Falle ein nahezu homogenes Magnetfeld entlang der Strahlenachse des Strahles. Die Elektronen führen eine Larmorpräzession aus. Je stärker das Magnetfeld, desto geringer ist ihr Präzisionsradius. Mit dem Herrmann-Radius lässt sich der Radius des Elektronenstrahlstromes errechnen, der 80% der Elektronen enthält.

3.1.3 Energie des Elektronenstrahl in der Falle

Die Energie des Elektronenstrahls im Fallenbereich wird hauptsächlich durch die Summe aller Beschleunigungsspannungen bestimmt (siehe Abb. 5.10). Sie ergibt sich in der Regel durch Addition der Einzelspannungen von Start- und Zielelektrode. Abbildung 3.2 illustriert den axialen Potentialverlauf von der Kanone bis zum Kollektor.

Die gesamte Beschleunigungsspannung der Elektronen im Fallenbereich ergibt sich aus der Summe von negativer Kathodenspannung, Driftröhrenplattform und Fallendriftröhre (DT9). Gleichung 3.1 zeigt die resultierende Beschleunigungsspannung U_{acc} für die Elektronen im Fallenbereich.

$$U_{acc} = U_{EGun} + U_C + U_{DTs} + U_{DT9} \tag{3.1}$$

Dabei stehen U_{EGun} für die Elektronenkanonenplattform, U_C für den Betrag der Kathodenspannung und U_{DT9} für die Spannung der Fallendriftröhre.

Der Elektronenstrahl und die Ionenwolke verursachen durch ihre Ladung noch Raumladungseffekte. In guter Näherung kann der Elektronenstrahl als zylinderförmig betrachtet werden. Nähere Beschreibungen des Elektronenstrahlradius finden sich in [Martínez, 2005]. Der Effekt der Ionenwolke wird zur Vereinfachung derart ange-

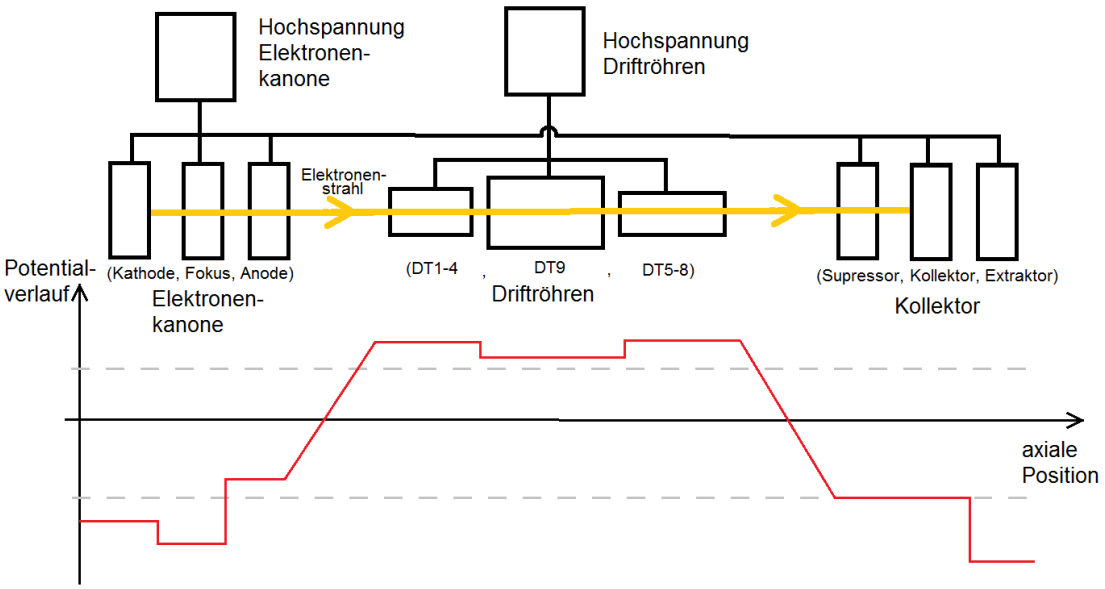

Abbildung 3.2: Axialer Potentialverlauf der Heidelberg-EBIT. Die obige Skizze zeigt die wichtigsten Konpotenten für den Elektronenstrahl. Sie erzeugen durch anlegen einer Spannung Potential, das sich mit der axialen Position ändert (unteres Diagtamm zeit Potentialverlauf rot). Suppressor- und Kollektorelektrode haben kein eigenes Netzteil. Die obige gestrichelte Linie zeigt das Potential der Elektronenkanonenplattform, die untere die der Driftröhren. Driftröhre 9 (DT9) entspricht der Driftröhre im Fallenbereich. Mit Hilfe der nebenstehenden Driftröhren (DT1-4 und DT5-8) kann ein Potentialtopf geformt werden, der die Ionen in axialer Richtung fängt. Der Extraktor hat ein etwas niedrigeres Potentialverlauf als die Kathode, somit lässt er keine Elektronen durch, Sie werden von der davorstehende Kollektorelektrode eingesammelt.

nommen, dass der Raumladungseffekt des Elektronenstrahles um einen bestimmten Faktor unterdrückt wird, da sie positiv und der Elektronenstrahl negativ geladen ist. Die Raumladung Φ des Elektronenstrahls ist vereinfacht zur seiner Elektronendichte ρ proportional, die wiederum propaortional zum Strom I und Antiproportional zur Geschwindigkeit v_e der Elektronen ist.

$$\Phi \propto \rho_e \propto \frac{I}{v_e} \tag{3.2}$$

$$v_e = c \times \sqrt{1 - (\frac{E_e}{m_e c^2} + 1)^{-2}} \tag{3.3}$$

Gleichung 3.3 gibt die relativistische Geschwindigkeit-Energie-Beziehung wieder, mit $c = 299792458\frac{m}{s}$ als Lichtgeschwindigkeit und E_e als Elektronenenergie. Die Ionenwolke schrumpft die effektive Raumladung Φ_{eff} um den Faktor f_W. Die Ruheenergie $m_e c^2$ des Elektrons beträgt ungefähr 511 keV.

$$\Phi_{eff} = \frac{\Phi}{f_W} \tag{3.4}$$

Die Elektronenstrahlenergie lässt sich somit schreiben als:

$$E_e = e \times U_{acc} - \Phi_{eff}. \tag{3.5}$$

Das negavite Vorzeichen vor Φ_{eff} in Gleichung 3.5 lässt sich dadurch erklären, dass die negative Ladung des Elektronenstrahls ankommende Elektronen abbremst, somit verringert sich ihre Energie im Fallenbereich. Die effektive Raumladung Φ_{eff} hängt jedoch von der sämtlichen EBIT Parametern ab und ist unbekannt. Daher wird sie am besten experimentell bestimmt, indem Resonanzschwerpunkte bei gleichen Beschleunigungsspannungen und unterschiedlichen Strömen gemessen wird. Dadurch ergibt sich ein annähernd linearer Zusammenhang:

$$\Phi_{eff} = S_{sp}(E_e) \times I. \tag{3.6}$$

Die Steigung S_{sp} kann somit ermittelt werden, die noch von der Elektronenstrahlenergie E_e abhängt. Wird die Steigung S_{sp} bei einer bestimmten Energie E_{sp} ermittelt, ergibt sich für die effektive Raumladung:

$$\Phi_{eff}(I, E_e) = S_{sp} \times \sqrt{\frac{1 - (\frac{E_{sp}}{m_e c^2} + 1)^{-2}}{1 - (\frac{E_e}{m_e c^2} + 1)^{-2}}} \times I. \tag{3.7}$$

Da der Raumladungseffekt klein gegenüber der Beschleunigungsspannung mal Elementarladung ist (siehe Abb. 5.10), kann im rechten Term von Gleichung 3.7 in guter Näherung E_e durch $e \times U_B$ ersetzt werden. Gleichung 3.7 vereinfacht sich somit zu:

$$\Phi_{eff}(I, U_{acc}) = S_{sp} \times \sqrt{\frac{1 - (\frac{E_{sp}}{m_e c^2} + 1)^{-2}}{1 - (\frac{U_{acc}}{m_e c^2} + 1)^{-2}}} \times I. \tag{3.8}$$

3.2 Die Heidelberg-Elektronenstrahl-Ionenfalle

3.2.1 Historisches zur Heidelberg-Elektronenstrahl-Ionenfalle

Die Heidelberg-Elektronenstrahl-Ionenfalle, abgekürzt HD-EBIT, war zuerst die Freiburger-EBIT, die dort 1999 in Betrieb genommen wurde. 2001 wurde sie nach Heidelberg zum Max-Planck-Institut für Kernphysik überführt und heißt seit dem Heiderberg-EBIT. Seit dem wurde an ihr zahlreiche Papers und Arbeiten verrichtet (siehe Tab. A.4). Eine Besonderheit der HD-EBIT ist die horizontale Lage des Ionenstrahls. Somit lassen sich relativ einfach, bzgl. vertikale Ionenstrahlen, weitere Experimente, wie das PENTATRAP-Experiment, anbauen.

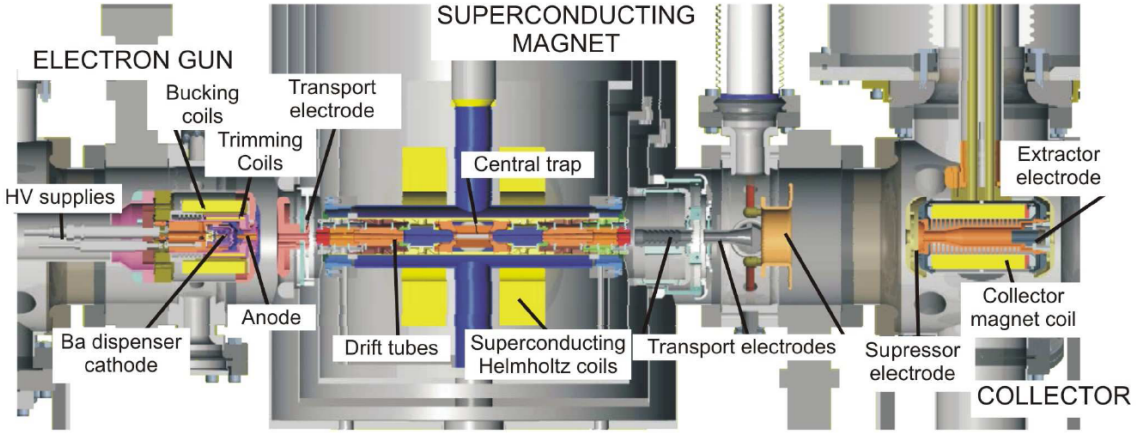

Abbildung 3.3: Schematischer Aufbau der Heidelberg-EBIT [Martínez, 2005]. Links befindet sich die Elektronenkanone, in der Mitte die Driftröhren (Fallenzentrum) und rechts der Kollektor.

3.2.2 Der Weg der Ionen

In dieser Arbeit werden Iridiumionen untersucht. Hierfür wird (Methylcyclopentadienyl) (1,5-cyclooctadiene) iridium(i), kurz $(C_6H_7)(C_8H_{12})$Ir, in die Falle durch Sublimation initiiert. Der Injektionsdruck kann mit einem Nadelventil reguliert werden. Die elektrisch neutralen Moleküle treten in die Falle ein und werden vom hochenergetischen Elektronenstrahl dissoziiert. Anschließend ionisieren die einzelnen Atome durch Stoßprozesse zu Ionen, je nach Elektronenstrahlenergie- bzw. Strom zu hochgeladenen Ionen, welche ebenso angeregt werden und charakteristische Photonen abstrahlen, die teilweise vom Röntgendetektor detektiert werden. Bei einem Dump oder Extraktion werden die Ionen aus der Falle zum Kollektor geführt, der den Elektronenstrahl einsammelt aber die Ionen durchlässt. Weiter geht es durch ein Linsensystem das den Ionenstrahl fokussiert. Danach ändert ein Ablenkmagnet seine Flugbahn um 90° und fächert ihn auf. Dabei werden Teilchen mit unterschiedlichen Masse- zu Ladungsverhältnis örtlich voneinander getrennt. Dieser Ionenstrahl trifft

Abbildung 3.4: Die Heidelberg-EBIT. Rechts ist der Faradaysche Käfig der Elektronenkanone zu sehen. Die gelb farbigen Bleiplatten umhüllen das Fallenzentrum der EBIT. Der beige Tank in der Mitte enthält flüssigen Stickstoff, der den Röntgendetektor, der zur Falle hin herausragt, kühlt.

entweder auf ein MPC (steht für engl. *micro channel plate* und ist ein Detektor für Teilchen, die beim Auftreffen eine Elektronenlawine auflösen und so ihre Position verraten), oder gehen zu dem PENTATRAP-Experiment weiter, das nebenan aufgebaut ist. Im PENTATRAP-Experiment werden hochpräzise Massenmessungen mit relativen Genauigkeiten unter 10^{-11} mit einer neuartigen Penningfalle durchgeführt.

3.2.3 Die Elektronenkanone

Die Elektronenkanone erzeugt einen Elektronenstrahl. Abbildung 3.6 zeigt den schematischen Aufbau. Sie besteht aus einem Heizelement, einer Kathode, Fokussierelektrode, Anode, Bucking- und Trimmspule. Dazu befindet sie sich noch auf einer vom Boden isolierte Plattform, die auf Hochspannungen bis zu 125 kV angelegt werden kann. Diese Spannung wird mit einem Spannungsteiler gemessen, welcher in der Dissertation von *F. A. J. González Martínez* [Martínez, 2005] genauestens beschrieben und getestet ist. Abbildung 3.7 zeigt die Plattform der Elektronenkanone inklusive Netzteile. Um einen Elektronenstrahl zu erzeugen, werden freie Elektronen benötigt. Dazu wird die Kathode mit einem Heizelement soweit erhitzt, dass thermische Elektronen austreten. Die Kathode besteht aus Barium und Wolfram, beides hitzebeständige Materialien. Die Austrittsarbeit der ELektronen betragt nur ca. 2 eV [Martínez, 2005]. Um die Elektronen zu beschleunigen, liegt zwischen der Kathode (negatives potential) und der Anode (positives Potential) ein elektrisches

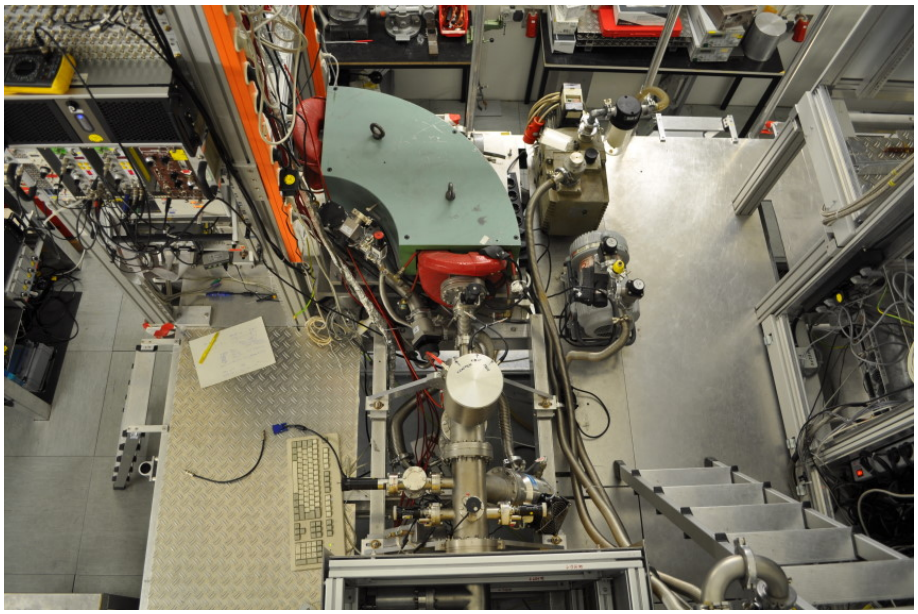

Abbildung 3.5: Extraktionssystem der Heidelberg-EBIT. Die extrahierten Ionen schießen vom unteren Bildrand zum oberen und werden durch den Ablenkmagneten (grünfarbig, nach links gebogen) abgelenkt und aufgefächert.

Feld an. Die dazwischen liegende Fokoselektrode erzeugt durch anlegen von Spannungen ebenfalls ein Potential, mit dem sich der Elektronenstrahlstrom regulieren lässt. Ist das Potential der Fokuselektrode niedrig genug (zur Erinnerung: Elektronen sind negativ geladen), können die Elektronen diese Barriere nicht mehr überwinden und der Strom verschwindet. Die Position und Lage der Elektronenkanone lässt sich manuell mit Justierschrauben verstellen. Dies hat sich vor allem beim Wechsel von Spannungsbereichen der Elektronenkanone oder längeren Wartezeiten als hilfreich erwiesen, damit die EBIT stabit läuft, d.h. dass nicht zu hohe Ströme bzw. Stromschwankungen auf der Anode, dem Suppressor oder dem Extraktor vorhanden sind. Tabelle 3.1 zeigt die für die Elektronenkanone verwendeten Geräte. Das dazugehörige Rack ist in Abbildung 3.7 zu sehen.

3.2.4 Der Fallenbereich

Der Fallenbereich befindet sich in den Driftröhren, siehe Abbildung 3.9. Sie befinden sich ebenfall auf einer isolierten Plattform, die der bis zu 30 kV angelegt werden kann. Zusätzlich kann auf jede einzelne Driftröhre eine bestimmten Spannung angelegt werden, um ein Fallenpotential zu erzeugen und die Ionen zu fangen, zu dumpen oder kontinuierlich zu extrahieren. Abbildung 3.8 zeigt ein Fallenpotential.

Die mittlere Driftröhre besitzt Beobachtungsschlitze, sodass sich daraus Photonen untersuchen lassen. Ein Teil dieser Photonen entwischt über ein Berylliumfenster zum Röntgendetektor, der ein Spektrum der Ionenwolke aufnimmt.

Bauteil	**Netzteil**
Heizelement	GW Laboratory DC Power Supply, Model GPS-1830
Kathode	fug MCA 750-1500
Fokus	Trek Model 609E-6, High Voltage Amplifier
Anode	fug HCN 35-12500
Kollektor Magnet	Sorensen DLM 8-75
Trimmspule A	Knürr-Heinzinger Economy Line LNG 32-3
Spannungsteiler Voltmeter	Keithley 2002 Multimeter
Hochspannung der Plattform	Glasman High Voltage, Inc., Series WR
Horizontaler Ablenkmagnet	Hewlett Packard 6268 DC Power Supply
Vertikaler Ablenkmagnet	Electro-Automatic GmbH & Co. KG ES-PS 3065-03B
Ventilatorversorgung Spannungsteiler	Laboratory Power Supply GW Model: GPR-1810HD
Extraktor	fug HCN 7E-6500
Supressor	geerdeter (bzgl. Plattform) Widerstand 1 kOhm zum Auslesen
Kollektor	geerdeter (bzgl. Plattform) Widerstand 10 Ohm zum Auslesen

Tabelle 3.1: Verwendete Geräte der Elektronenkanone

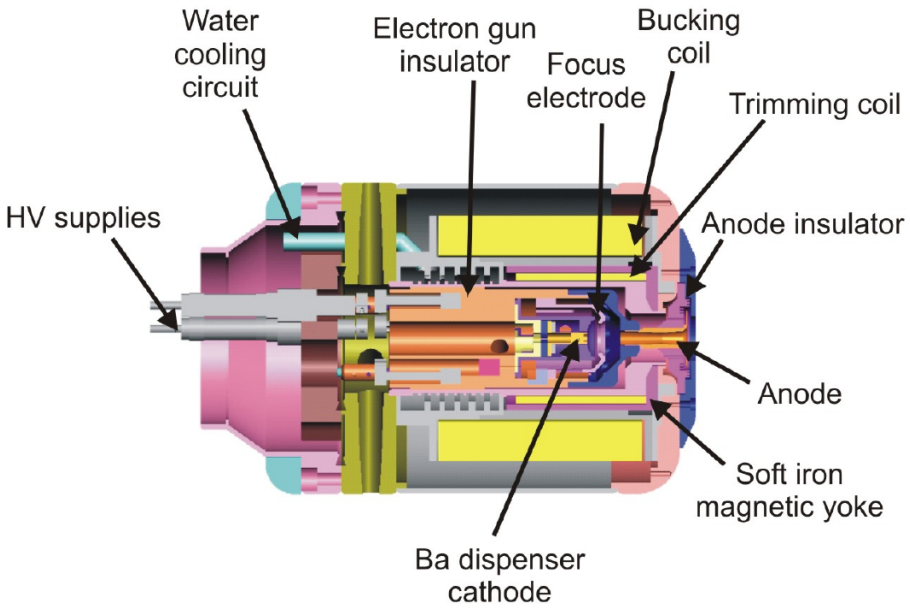

Abbildung 3.6: Schematischer Aufbau der Elektronenkanone [Martínez, 2005]. Die Kathode wird auf Temperatur gebracht, sodass thermische Elektronen austreten. Zwischen Kathode und Anode legt eine Spannung an die die Elektronen zur Anode beschleunigt. Mit der Fokuselektrode zwischen Anode und Kathode lässt sich durch anlegen bestimmter Spannungen der austretende Strom regulieren.

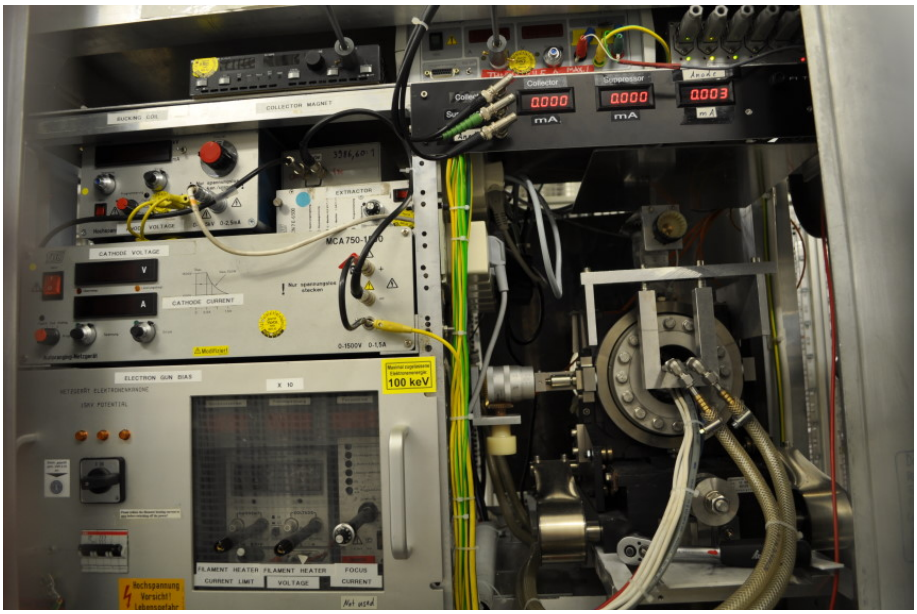

Abbildung 3.7: Rack der Elektronenkanone. Links sind Netzteile für Elektroden wie Anode, Extraktor, Kathode und um linken unteren Kasten das Netzteil für das Heizelement, das auf Kathodenspannung anliegt. Rechts ist die Halterung der Elektronenkanone.

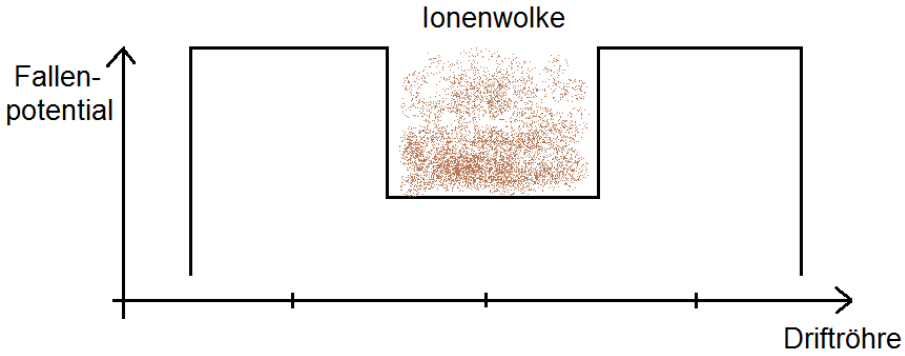

Abbildung 3.8: schematisches Fallenpotential für Driftröhren; Das Elektron kann sich in der Mitte frei bewegen (Potentialtopf in axialer Richtung). Die Ionenwolke ist braum dargestellt.

Im Fallenbereich befindet sich ein nahezu homogenes Magnetfeld, das von einem supraleitenden Helmholz-Spulenpaar erzeugt wird. Es erreicht eine magnetische Flussdichte von ca. 8 Tesla bei einem Strom von 76.24 A. Um überhaut Supraleitfähigkeit zu erreichen, dürfen Temperaturen von maximal 4.2 K herrschen. Dies lässt sich mit flüssengen Helium realisieren, dass einmal pro Woche nachgefüllt werden muss. Tabelle 3.2 listet die verwendeten Geräte für die Driftröhren auf.

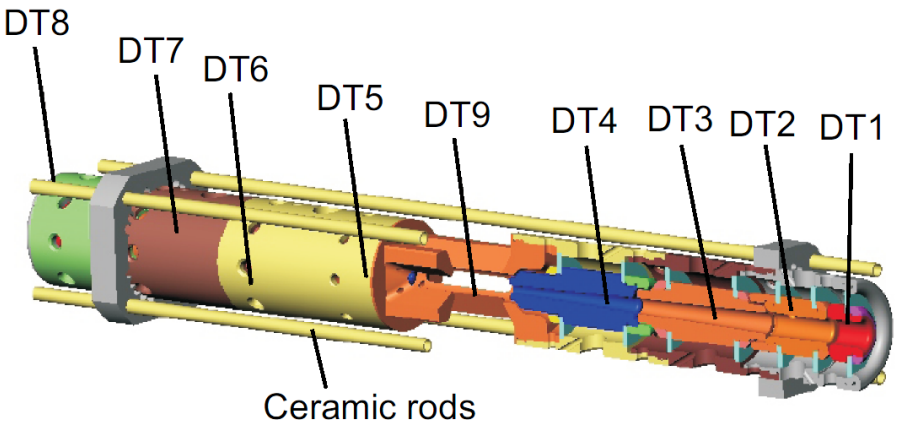

Abbildung 3.9: Schematischer Aufbau der Driftröhren [Martínez, 2005]. Der Elektronenstrahl geht in Driftröhre 1 rein und kommt bei Driftröhre 8 raus. Driftröhrer 9 ist die Fallenelektrode.

Bauteil	**Netzteil**
DT1	fug HCN 7E-3500
Trompete	fug HCN 7E-3500
DT2-DT8	Eigenbau
Hochspannung für DT-Plattform	Heinzinger HCN 30.000 - 5 pos.
Spannungsteiler Voltmeter	Keithley 2002 Multimeter

Tabelle 3.2: Verwendete Geräte der Driftröhren

3.2.5 Der Kollektor

Nachdem der Elektronenstrahl die Falle durchquert hat, muss er kontrolliert eingesammelt werden. Das hat zum einen den Grund, dass er von möglich extrahierten Ionen getrennt und nachfolgende Experimente nicht behindert soll, dass er nicht unkontrolliert auf sonstige Bauteile trifft und sie sogar beschädigen kann und den Hochspannungsbetrieb vereinfacht, da der Kollektor auf den selben Potential wie die Elektronenkanone liegt und das Hochspannungsnetzteil dafür deutlich weniger Strom bereitstellen muss (ca. 1mA statt bis zu 300 mA). Der Elektronenstrahlstrom ist also eine geschlossener Stromkreis auf der Hochspannungsplattform. Wenn

der Elektronenstrahl aus der Falle zum Kollektor fliegt, wird er somit abgebremst. Der Kollektor besteht aus einer Kollektorelektrode, einem Supressor, Extraktor und einem Kollektor-Magnetspule. Der Elektronenstrahl geht duch eine Öffnung der kreisförmige Supressorelektrode und wird in der Kollektorelektrode aufgefächert und eingesammelt. Die Extraktorelektrode verhindert durch ein positiv angelegtes Potential, dass der Elektronenstrahl mit den ionen durchwandert. Dabei muss das Potential höher als das der Kathode sein. In disem Experiment liegen auf der Suppressor- und Kollektorelektrode keine Spannung an. Ihre Ströme werden mit Hilfe von Spannungsteilern gemessen. Abbildung 3.10 zeigt den schematischen Aufbau des Kollektors.

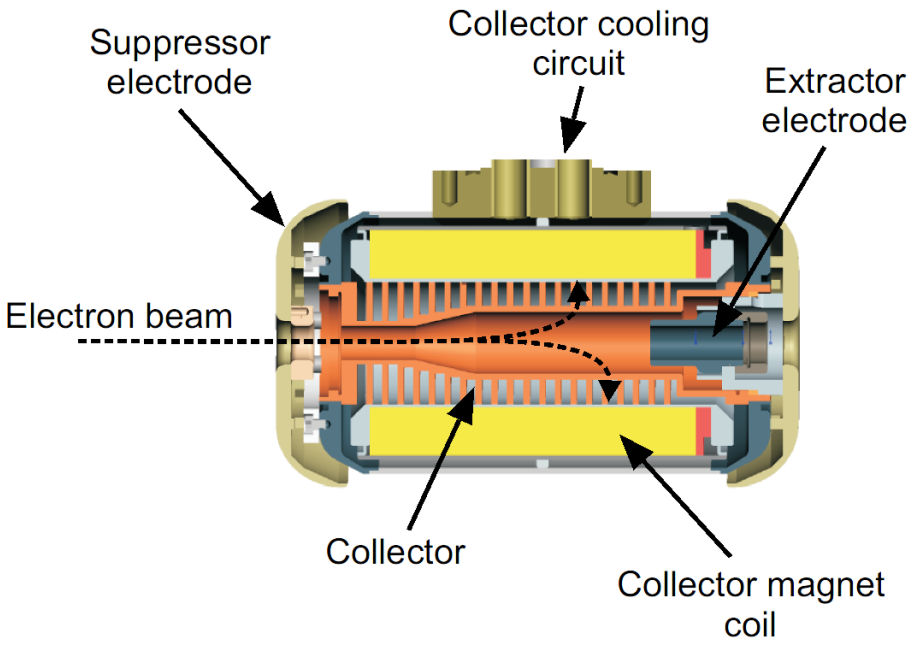

Abbildung 3.10: Schematischer Aufbau des Kollektors [Martínez, 2005]. Der Elektronenstrahl kommt von links. Auf dem Extraktor liegt eine Spannung, die eine axiale Potentialbarriere erzeugt, sodass die Elektronen nicht durchgelangen, abgebremst und von der Kollektorelektrode eingesammelt werden. In dieser Arbeit liegen an Suppressor- und Kollektorelektrode keine Spannung an, sondern sind über einen Messwiderstand, mit dem ihre Strom gemessen werden, mit der Elektronenkanonenplattform verbunden.

3.3 Der Röntgendetektor

Der Röntgendetektor nimmt Photonen, die aus der Falle austreten, auf. Er befindet sich in einem eigenen Vakuumbehälter, ist mit Blei und Messung abgeschirmt und wird mit flüssigen Stickstoff gekühlt (Siedepunkt 77.15K), dass sich in einem dazu

montierten Tank befindet. Abbildung 3.11 zeigt den Detektor. Er wird durch das Gewicht der Blei- und Messingabschirmung, siehe Abbildung 3.12, zusätzlich gestützt. Im Tank befindet sich flüssiger Stickstoff der ca. einmal in der Woche nachgefüllt werden muss. Im Detektor wird an einem dotierten Germaniumkristall, der als Diode fungiert, eine elektrische Spannung in Sperrrichtung angelegt. Durchquert ihn ein hochenergetisches Teilchen, löst es freie Ladungsträger heraus, die einen Stromimpuls auslösen. Die Pulshöhe gibt dabei Aufschluss über die Energie des Teilchens. Die hierfür verwendeten Geräte sind in Tabelle 3.3 zu finden.

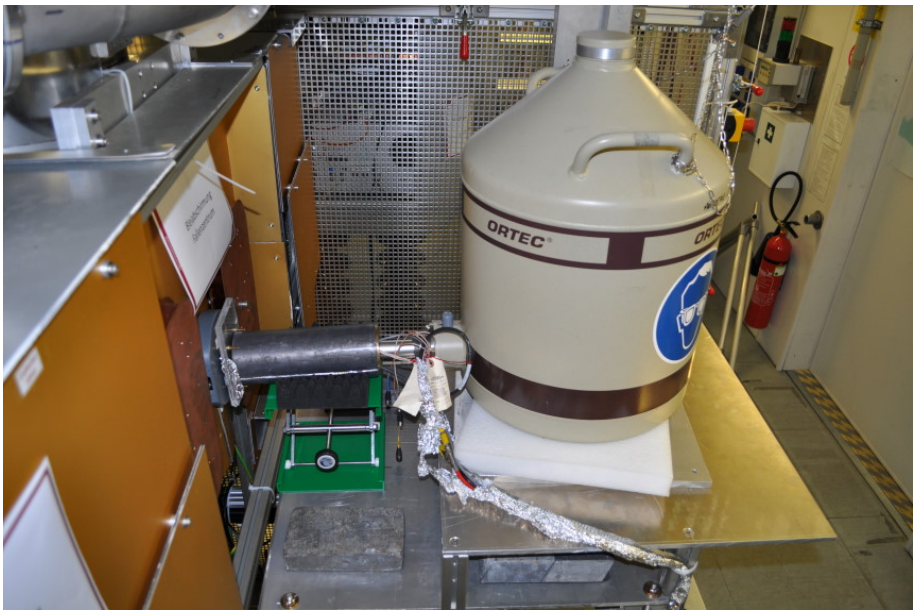

Abbildung 3.11: Röntgendetektor GLP Serie von *ORTEC*. Er ist direkt vor dem Berylliumfenster der HD-EBIT positioniert und mit Blei und Messing ummantelt. Der Tank enthält flüssigen Stickstoff, der den Detektor kühlt.

Sowohl die EBIT als auch der Detektor haben Berylliumfenster (siehe Abb. 3.13). Sie haben die Eigenschaft auch als dünne Folie große Druckdifferenzen Stand zu halten, sind jedoch giftig und können bei unvorsichtigen Wartungsarbeiten leicht zerstört werden.

Bauteil	**Netzteil**
Germanuim Detektor	ORTEC Model GLP-36360/13-P
Hochspannung Detektor	NHQ 215M N24
Signalwandler Detektor	Ortec Model 463 constant
Analog-Digital Wandler	Dual TADC 7072T

Tabelle 3.3: Verwendete Geräte für Detektor

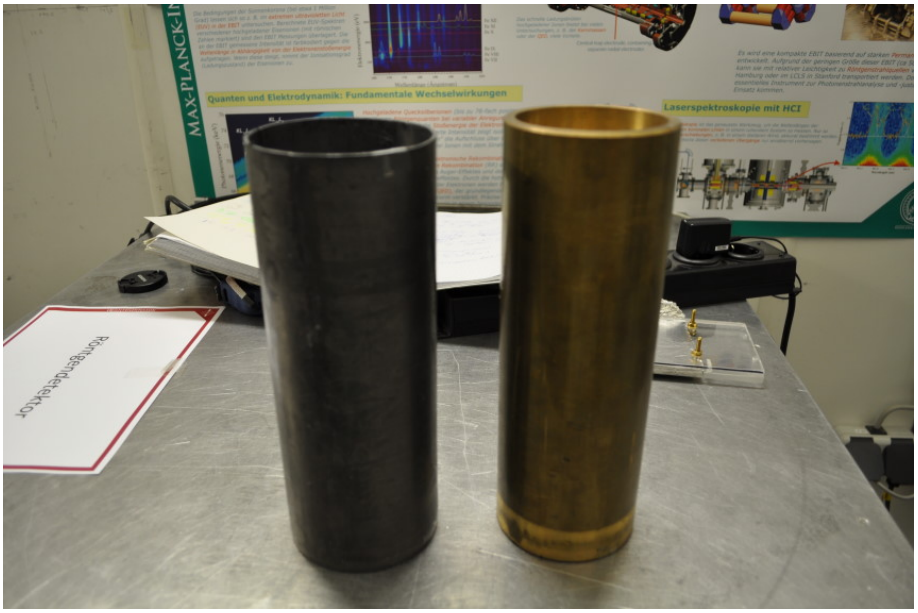

Abbildung 3.12: Abschirmungen des Röntgendetektors. Die Rohre bestehen aus Blei und Messing und werden beide über den Detektor gestülpt.

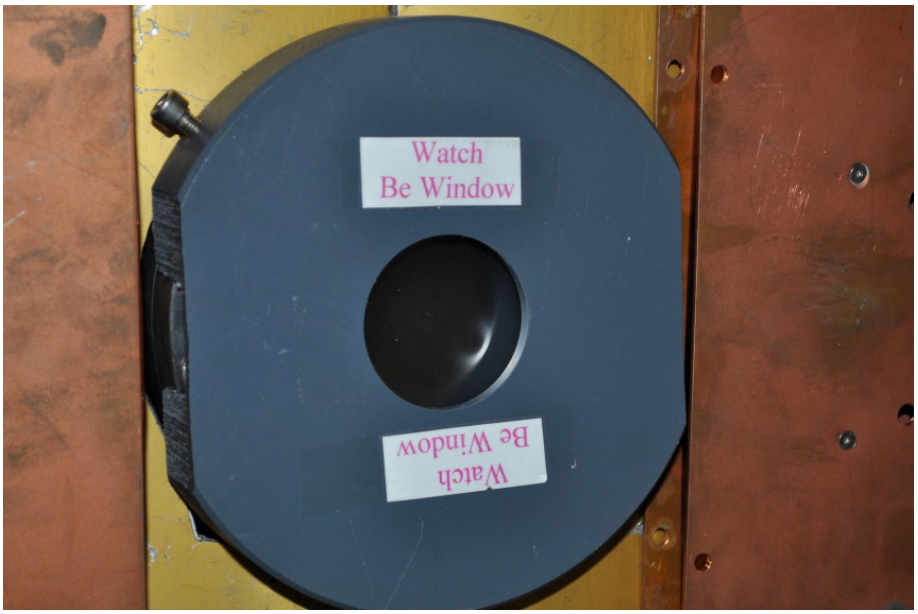

Abbildung 3.13: Berylliumfenster der HD-EBIT. Eine dünne Berylliumfolie trennt das Vakuum der Falle von der Atmosphäre, lässt aber Röntgenphotonen zu Beobachtungszwecke durch.

3.4 Das Datenaufnahmesystem

3.4.1 Datenaufnahme des Röntgensetektors

Die Signale aus dem Detektor gelangen mittels eines analog-digital-Wandlers und einem speziellem GPIB-Interface zu einem PC. Dieser besitzt ein MPA Datenaufnahmesystem der Firma *ComTec Communication Technology GmbH*, das die Signale speichert, verarbeitet und darstellt. Das Signal für die Elektronenstrahlenergie wird ähnlich verarbeitet, nur dass das Analogsignal nicht von einem Detektor kommt stattdessen von einer *National Instruments* Karte generiert wird. Diese wird vom Kontrollsystem der EBIT angesteuert, welches im nächsten Kapitel ausführlich besprochen wird. Das System löst eine Messgröße in 8192 Kanälen auf, die Kalibriert werden müssen. Das Singal vom Detektor, welches der Photonenenergie entspricht, ist mit einer Am241-Quelle kalibriert (Abb. 3.14). Diese wird für eine bestimmte Zeit vor dem Detektor positioniert und bestrahlt ihn, sodass eine charakteristisches Spektrum aufgenommen werden kann, mit dem die Kanäle kalibriert werden können.

Abbildung 3.14: Kalibrationsquelle des Röntgendetektors, der rote Pfeil zeigt darauf. Material: Am^{241}, Aktivität: 3.7 kBq am 6.2.64, physikalische Halbwärtszeit: 432 Jahre, Aktivität Juli'14 ca. 3.4 kBq

Neben der Photonenenergie wird noch die Elektronenstrahlenergie mitgemessen. Dazu gelangt ein analoges Signal, dass von einer *National Instruments* Karte generiert wird, in ein Signalwandler, der auch vom Detektorsignal getriggert wird. Das analoge Signal wird mit einem speziellen Server (*task_server_source_to_target_linear_02.py*) erstellt, der zum Kontrollsystem gehört (siehe Kapitel 4). Auch hier können charakteristische Schaubilder für die Kalibration erstellt werden.

3.4.2 Datenaufnahme vom EBIT Kontrollsystem

Das EBIT Kontrollsystem umfasst Loggingprogramme, welche sämtliche Daten der EBIT über das lokale Netzwerk aufnehmen und speichern. Während einige Programme einen Satz von Daten in regelmäßigen zeitlichen Abständen aufnehmen (zeitgesteuerte Logs), nehmen andere Programme vereinzelt Daten auf, falls sie sich ändern (ereignisgesteuerte Logs). Daneben gibt es noch graphische Benutzeroberflächen, die Daten nur temporär aufnehmen und graphisch darstellen. Ihr Zweck umfasst hauptsächlich die Überwachung.

4 Das Kontrollsystem

4.1 Motivation

Beim den bisherigen Kontrollsysteme der HD-EBIT konnte nur auf dem PC gesteuert werden, bei dem die Kontrollsoftware lief. Zugriffe über das Netzwerk war nur über *Remote Desktop Verbindung* möglich, was die Anzahl der Benutzer zur selben Zeit auf eins beschränkt. Ziel war es, von jedem PC aus im Netzwerk Zugriff und Kontrolle auf das Experiment zu bekommen. Zudem sollten möglichst einheitliche Programme verwendet werden, um Einarbeitungszeiten zu drosseln. Programme wurden zuvor mit LabView graphisch erstellt, anstatt Zeilen zu schreiben [NI]. Für das Kontrollsystem soll Python2 verwendet werden, eine Programmiersprache die zwar textbasiert ist, aber mit denen die meisten von unserem Team vertraut waren, und von ihnen als leicht erlernbar und übersichtlich betrachtet wird. Ein weiteres Ziel ist die Abstraktion des Experiments. Der Laborant soll während des Experiments möglichst mit physikalischen Größen konfrontiert werden anstatt mit Geräteparametern. Ein weiterer wichtiger Punkt sind Loggingprogramme, die auf diversen PCs laufen und unabhängig voneinander agieren können.

4.2 Architektur

Das Kontrollsystem wird von Computern verwaltet, die über das lokale Netzwerk miteinander verbunden sind und untereinander interagieren. Die Prinzip basiert auf den Client-Server-Modell; auf diversen Computer laufen Server Programme, die Datenbanken über die EBIT bereitstellen. Diese werden von anderen Server oder Clienten genutzt. Zur Realisierung benutzen wir EPICS [EPICS]. Es steht für *Experimental Physics and Industrial Control System* und ist nicht nur eine Softwaresammlung, sondern auch eine Kontrollsystem-Architektur und eine Kollaboration. Es nutzt das Channel Access Protokoll, das auf dem Internetprotokoll IP basiert und arbeitet mit sogenannten Prozessvariablen, eine Sammlung von Informationen wie z.B. Spannung, Zeitstempel und Einheit. Der Zugriff erfolgt durch den Namen der Prozessvariable. Somit brauch sich der Nutzer nicht um den Stadtort des Computers zu kümmern. Der Typ einer Prozessvariablen ist entweder Fließkommazahl, Ganzzahl, Zeichenkette oder Aufzählung (*enum*). Dabei können sie auch als Feld (engl. *array*, eine Ansammlung gleicher Typen) auftreten. Die hier beschriebene Architektur bezieht sich nicht auf die interne EPICS Verfahrensweise, sondern dessen Anwendung.

Der Name der Prozessvariablen sind in der Regel in Großbuchstaben und setzen sich aus verschiedenen Präfixen und dessen Zuordnung wie Gerät oder Physikali-

sche Größe, alle durch Doppelpunkt getrennt, zusammen. Der erste Prefix ist dabei immer *HDEBIT*, während der zweite durch den Server festgelegt ist, wie z.B. beim Elektronenkanonenserver: *HDEBIT:EGUN:CATHODE:VOLTAGE:IST.VAL*. Einige Prozessvariablen gleichen sich bis zu einem Punkt, danach folgen einheitliche Bezeichnungen wie *VAL* oder *INTERLOCK:HIGH*. Sie bilden ein sogenanntes Rekord, eine Ansammlung von Prozessvariablen für eine Bestimmte Aufgabe. Die Bezeichnung nach dem Punkt bildet das Attribut zum Rekord. Tabelle 4.1 zeigt einige Attribute und ihre Aufgabe. Die her verwendeten Rekords gleichen nicht denen von [EPICSRecord], jedoch ähneln sie sich.

Attribut	**Aufgabe**	**Beispiel**
VAL	Ist-Wert	5.34
SET	Soll-Wert	600.0
LOW	erreichbares Minimum	0.0
HIGH	erreichbares Maximum	1000.0
EGU	Einheit	'V'

Tabelle 4.1: Einige Attribute

Das Kontrollsystem der HD-EBIT ist derart gestaltet, dass bestimmte Server eine Laborbox verwalten, die am selben PC angeschlossen ist. Andere Server oder Clienten greifen ausschließlich auf dem Server zu, um die Laborbox anzusteuern. Laborboxen sind dabei Geräte wie Boxen, an denen BNC Anschlüsse vorhanden sind, die analoge oder digitale Ein- oder Ausgänge realisieren, oder Multimeter. Sie selbst werden mit speziellen PC Karten angesteuert. Tabelle 4.2 zeigt die verwendeten Karten und deren Laborbox.

Karte	**Laborbox**	**Spezifikationen**
NI PCI-6229	Box mit analoge oder digitale BNC Ein- oder Ausgänge	32 analoge I-, 4 analoge O-, 48 digitale I/O-Kanäle, Aufnahme mit 250 kS/s, 16 Bit
NI PCI-6703	Box mit digitale BNC Ein- oder Ausgänge, analoge BNC Ausgänge	16 analoge O-, 8 digitale I/O-Kanäle, 16 Bit
PCI-GPIB	Keithley 2002 Multimeter	1 analoger Eingang, Auflösung bei 20V DC Bereich: 10 nV

Tabelle 4.2: Karten und Laborboxen

In dem hier verwendeten Kontrollsystem gibt es mehrere Ebenen von Servern: Labboxserver, Deviceserver und Taskserver bzw. -Clienten (Abb. 4.1). Die Labboxserver laufen auf dem selben Computen, mit denen auch Laborboxen verbunden sind und steuern diese an bzw. lesen sie aus. Sie kümmern sich nicht was daran an den Laborboxen angeschlossen ist oder wer sie ansteuert. Die Deviceserver hingegen

verwalten die Geräte, die an den Laborboxen angeschlossen sind, wie Netzteile, Spannungsteiler oder Ventile. Sie stellen die physikalischen Größen (Spannung, Strom, ...) im Netzwerk zur Verfügung, mit denen man letztendlich arbeiten will. Dazu nutzen sie die Labboxserver und rechnen die elektrische Spannung an einer oder mehreren Anschlussbuchsen zur gewünschten Größe um. Zuletzt gibt es noch die Taskserver, welche die Deviceserver oder Labboxserver gebrauchen um bestimmte abstrakte Aufgaben zu erledigen, wie ein Dump auf den Driftröhren oder eine Stabilisierung des Elektronenstrahlstromes.

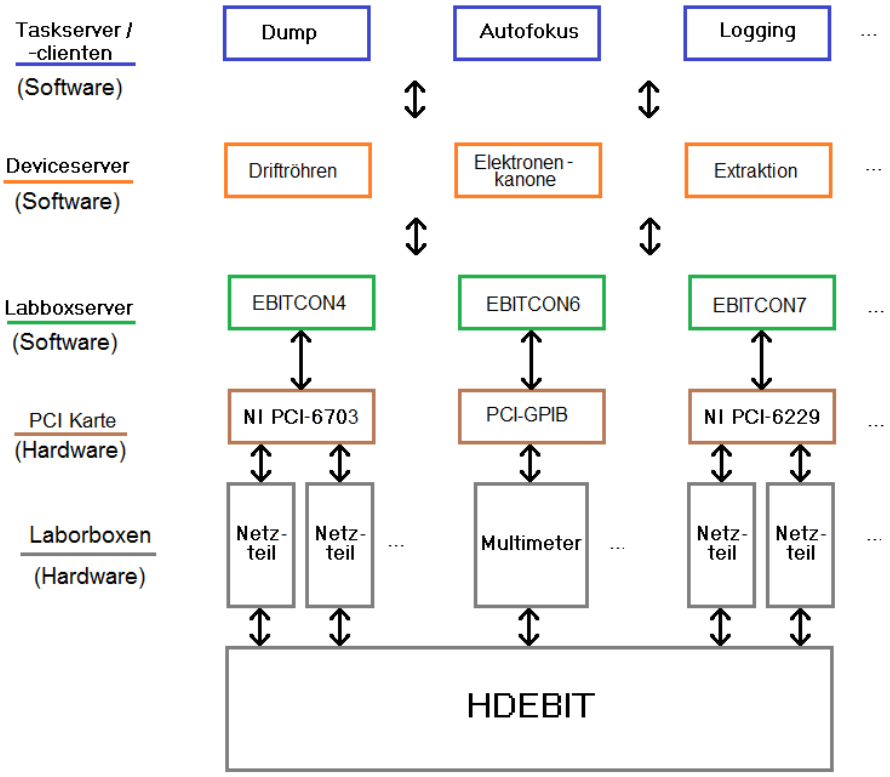

Abbildung 4.1: Verschiedene Serverebenen mit Beispielen

Neben den Servern gibt es auch Clienten. Sie können ebenfalls Prozess Variablen lesen oder beschreiben, stellen jedoch keine Datenbank zur Verfügung. Somit haben sie keine eigenen Prozess Variablen. Dennoch können sie auch gewisse Aufgaben erfüllen und sind i.d.R. schneller zu programmieren. Sie können auf alle Server-Ebenen zugreifen. Eine Hauptanwendungen sind graphische Benutzeroberflächen (*graphical user interface*, kurz *GUI*), die einen visuellen Einblick und Kontrolle des Systems erlauben. Ein weiteres wichtiges Beispiel für ein Client wäre ein Logging Programm, das den Verlauf von Prozessvariablen protokolliert. Mehrere Logging Programme auf diversen Computern verringern die Wahrscheinlichkeit eines Datenverlustes, falls einer von ihnen ausfällt (was schließlich die Grundidee des Internets war).

Die Trennung zwischen den einzelnen Server-Ebenen ist nicht strikt und durch die EPICS Software auch nicht garantiert. Es ist vielmehr eine Richtlinie für eine einfache Kontrollstruktur. Ein konfliktfreier Ablauf liegt letztendlich bei den Laborarten, die sich untereinander absprechen und Regeln festlegen sollten.

EPICS beschränkt sich nicht auf eine einzige Programmiersprache, sondern kann in diversen genutzt werden. Hier wird Python2 verwendet, eine Interpretersprache, die zur Ausführung ein Skript benötigt. Zudem muss nur der Interpreter und sonstige benötigte Software auf den Computern installiert werden um ein Skript, das ohne weiteres portiert werden kann, auszuführen. Um EPICS mit Python2 zu benutzen, braucht man zusätzlich noch PyEPICS. Diese Programmbibliothek dient als Schnittstelle zwischen Python und EPICS [PyEPICS]. Des Weiteren wird noch die EPICS-Erweiterung PCASPy, auch eine Programmbibliothek, mit der sich EPICS-Server relativ einfach aufbauen lassen (Module *SimpleServer* und *Driver*) [PCASPy]. Der Programmierstil ist objektorientiert, d. h. es stehen Basisklassen zur Verfügung, die man zur eigenen Klasse ableitet, welche die geforderten Aufgaben erledigen kann. In diesem Fall wird der Standartserver verwendet. Dieser braucht eine Treiberklasse, die abgeleitet wird. Die erstellten Programmskripte zu den Servern beinhalten hauptsächlich die Ableitung der Treiberklasse. Das Hauptprogramm startet lediglich den Server, bindet die Treiberklasse ein und lässt servereigenen Interpreter laufen. Durch diesen Interpreter, der in der Treiberklasse definiert und implementiert ist, kann der Server per Kommandoeingabe beeinflusst werden, wie das lesen oder beschreiben von Prozessvariablen, starten oder stoppen des Servers. Er dient lediglich als Hilfestellung, primär sollten die Server über das Netzwerk gesteuert werden.

4.3 Server und Clienten

Die Labbox- und Deviceserver bestehen grundsätzlich aus einem einheitlichen Skript mit einer Konfigurationsdatei, in der alle Konfigurationsparameter wie Kanalbenennungen drinn stehen, durch die sich die Server hauptsächlich unterscheiden. Die Labboxserver brauchen zusätzlich noch Skripte, um die Laborboxen korrekt anzusteuern. Dennoch sind einige Server-Skripte leicht verändert um ihre Aufgabe gerecht zu werden. Zum Beispiel der Deviceserver für die Elektronenkanone ändert den Umrechnungsfaktor des Spannungsteiler-Messgerätes jedes Mal wenn sich der Spannungsteiler auf einen anderen Wertebereich umschlägt (somit auch das Verhältnis der Widerstände zueinander). Im Deviceserver für die Extraktion sind zusätzliche Kanäle für Fokus, Astigmatismus, Horizontal und Vertikal, welche die Elektroden in der Silker-Linse mit linearen Superpositionen ansteuert. Die Taskserver erledigen spezielle Aufgaben, haben somit kein Einheitsprogramm oder Konfigurationsskript, jedoch wurden diverse Vorlagen geschrieben. Einige von ihnen haben ein internes Loggingprogramm, welches die wichtigsten Veränderungen mit Zeitstempel in eine externe Datei speichert.

Clienten können wie Server Prozessvariablen lesen, überwachen und beschreiben, haben aber keine eigene Datenbank. Sie haben aber den Vorteil, dass sie einfacher

und schneller zu schreiben sind, ein Minimalprogramm braucht nur wenige Codezeilen. Die hier verwendeten Clientprogramme haben keine Konfigurationsskript, alles wird in nur einem Skript, meistens in Großbuchstaben im Anfang, konfiguriert. Um die Fehlersuche zu erleichtern, haben sie in i.d.R. ein internes Loggingprogramm.

Die hier verwendeten Server haben grundsätzlich ein eigenes Rekord: *HDEBIT:(Serverprefix):SERVER.(Attribut)*. Tabelle 4.3 listet die einzelnen Atrribute auf.

Serverebene	**Attribut**	**Aufgabe**	**Beispiel**
Beide	RUNNING	Flag ob Server läuft	1
Beide	KEEPER	Zeichenkette für Verantwortlichen	'D.Hollain Tel490'
Beide	SHUTDOWN	Flag um Server in Ausgangszustand zu versetzen	0
Deviceserver	INTERLOCK:BREAK	Flag ob ein Interlock ausgelöst wurde	1
Deviceserver	TIME	Laufzeit in s des Servers	300.6
Deviceserver	TIME:TDELAY	Zeitschritt in s um TIME zu aktualisieren	0.1
Deviceserver	RAMP:ON	Flag für Rampenfunktion (an/aus)	0
Deviceserver	RMAP:TDELAY	Zeitschritt in s für Rampenfunktion	0.01
Deviceserver	RAMP:TRUN	Flag ob Thread für Rampenfunktion noch läuft	1
Deviceserver	RAMP:CHANGE	Flag ob Rampenfunktion gerade an- oder abgeschaltet wird	0

Tabelle 4.3: Servereigene Attribute

Die Deviceserver sind auf Ausfälle der Labboxserver vorbereitet. Falls ein Labboxserver ausfällt, können sämtliche Prozessvariablen mit dem Attribut *VAL*, die damit verknüpft sind, nicht mehr verändert werden. Die mit dem Attribut *SET* dagegen schon. Falls der Labboxserver wieder online ist, werden seine Kanäle entsprechen vom Deviceserver wieder auf die richtigen Werte geladen. Beim Start eines Deviceservers lädt er automatisch sämtliche Kanäle von den Labboxservern, die er benötigt, und beschreibt seine eigenen Prozessvariablen mit den umgerechneten Werten. Der Ausfall eines Deviceservers hingegen kümmert den Labboxserver nicht.

Schicht	Skriptname	Beschreibung
Labboxserver	labbox_server_08.py	Steuert NI6229 NI6703 Karten und GPIB Karte für Keithly 2002 Multimeter
	labbox_server_08_2.py	Steuert nur NI6229 und NI6703 Karten
Deviceserver	device_server_08_egun.py	Steuert Elektronenkanone
	device_server_08_dt.py	Steuert Driftröhren
	device_server_08_extraction.py	Steuert Ionenextraktion
Taskserver	task_server_02_dump_02.py	Steuert den Dump der Driftröhren
	task_server_source_to_target_linear_02.py	Steuert den analogen Eingang für den MPA1C Kanal des Datenaufnahmesystems

Tabelle 4.4: Server diverser Ebenen

Skriptname	Beschreibung
full_dump_01.py	Macht einen speziellen Dump auf den Driftröhren (alle Spannungen nach einender hoch, warten und zurück)
zeitmittelwert_pv_01.py	Zeigt einen bestimmten Mittelwert einer Prozessvariable an
egun_anode_special_interlock_01.py	Spezielles Interlock (löst Interlock aus falls Anodenstrom für bestimmte Zeit außerhalb des erlaubten Bereiches liegt)
egun_extractor_special_interlock_01.py	Spezielles Interlock (löst Interlock aus falls Extraktorstrom für bestimmte Zeit außerhalb des erlaubten Bereiches liegt)

Tabelle 4.5: Client-Skripte

4.4 Implementierung der Server

Dieser Abschnitt handelt von der Implementierung der Labbox- und Deviceserver. Die Programmiersprache ist wie oben erwähnt Python2.7. Auch hier wird nicht auf EPICS-interne Arbeitsweisen eingegangen, sondern deren Anwendung.

4.4.1 Labboxserver

Die Labboxserver benötigen zunächst Skripte, um die PCI-Karten korrekt anzusteuern. Diese Skripte (*labbox_ni_01.py* und *labbox_gpib_01.py*) enthalten Klassen mit Methoden, die für das Initialisieren, Auslesen oder Beschreiben von Kanälen zuständig sind. Bevor der Labboxserver startet, lädt er die Kanalnamen aus der Konfigurationsdatei und initialisiert sie mit Hilfe dieser Klassen. Jede PCI-Karte wird vom einem eigenen Python-Thread behandelt. Er durchläuft permanent eine Schleife wobei Eingänge ausgelesen und Ausgänge beschrieben werden. Lesen und Schreiben erfolgt somit nicht auf Anfrage.

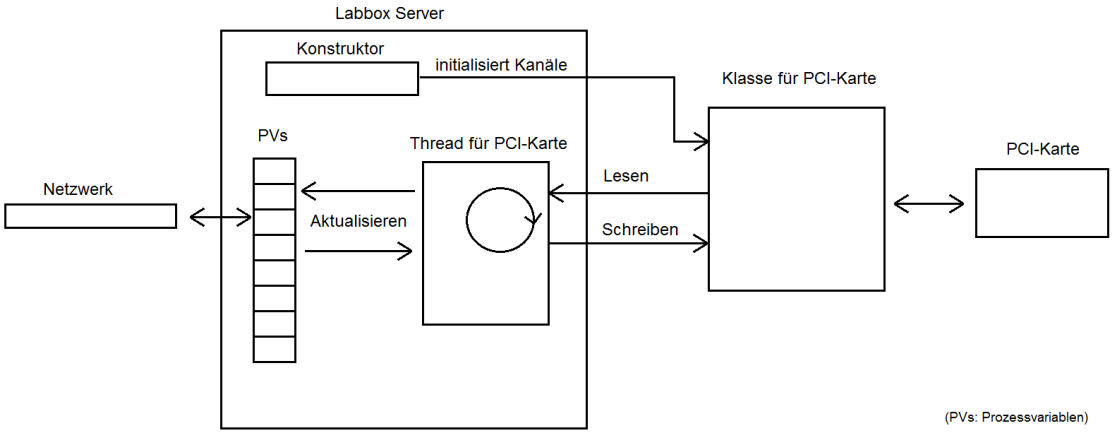

Abbildung 4.2: vereinfachte Darstellung vom Labboxserver

4.4.2 Deviceserver

Deviceserver verwenden *callbacks*, sogenannte Rückruffunktionen, um Labboxserver anzusteuern. Prozessvariablen für Lesekanäle werden ab dem Start des Deviceservers automatisch überwacht (mit der EPICS Funktion *camonitor*). Für jede Änderung am Wert wird die Rückruffunktion gestartet, die die entsprechende Prozessvarialbe im Deviceserver entsprechend umrechnet (üblicherweise durch Multiplikation mit einem Umrechungsfaktor) aktualisiert. Umgekehrt funktioniert es mit Prozessvariablen für Schreibkanäle. Hier werden sie vom Deviceserver überwacht und die Rückruffunktion aktualisiert den Labboxserver. Als Sicherung gegen falsche Handhabung werden auch die Prozessvariablen für Schreibkanäle vom Deviceserver aus überwacht und zurückgeändert, falls sie von jemand anders falsch beschrieben werden.

Zudem werden die eigenen Prozessvariablen noch aus Sicherheitsgründen überwacht. Falls der Wert einer Prozessvariablen außerhalb des geeigneten Interlock-Intervalls gerät, wird ein Interlock ausgelöst, d.h. der Server beschreibt seine Prozessvariablen mit ihren Notwerten, blockiert das Beschreiben und schickt eine E-Mail mit entsprechenden Informationen. Ob und bei welcher Prozessvariablen eine derartige Überwachung, Schreibblockade, Reset ausgelöst werden soll oder an wen die EMail geschickt wird, ist der Konfigurationsdatei entnommen. Der Deviceserver besitzt außerdem noch eine interne Rampenfunktion, welche sich vor allem Kalibrations- und Testphasen als hilfreich erwiesen hat. Ist sie aktiv, fährt der VAL-Wert nicht sprunghaft gegen den SET-Wert eines Rekords, sondern mit einer bestimmten Rate.

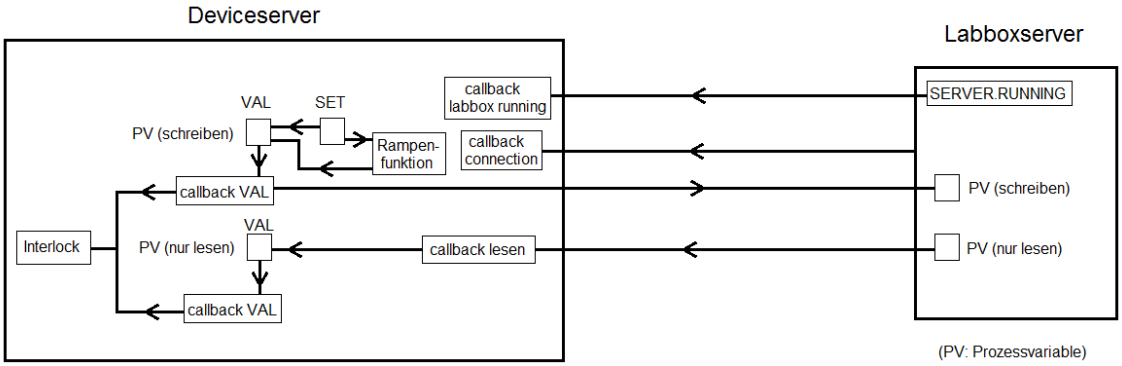

Abbildung 4.3: vereinfachte Darstellung vom Deviceserver

4.5 Graphische Benutzeroberflächen

Graphische Benutzeroberflächen (englisch *graphical user interface* oder GUI) erlauben eine graphische Kontrolle. Sie wurden mit dem Programm caQtDM erstellt, das einen einfaches Desigin per Drag&Drop, anbietet. Tabelle 4.6 zeigt die verwendeten GUIs. Für jedes wurde noch ein weiteres mit der Endung „_watchonly“erstellt. Hierfür wurde das Original so umgeändert, dass keine Eingaben mehr möglich sind. Das reduziert die Gefahr unabsichtlich falschliche Eingaben weiterzuleiten.

Dateiname	**Beschreibung**
caqtdm_egun_control_05.ui	GUI für Elektronenkanone
caqtdm_hv_control_03.ui	GUI für Hochspannungskontrolle (Bias für Elektronenkanone und Driftröhren)
caqtdm_dt_control_07.ui	GUI für Driftröhren
caqtdm_extraction_control_03.ui	GUI für Ionenextraktion

Tabelle 4.6: Graphische Benutzeroberflächen, Dateiname (1. Spalte) mit Beschreibung (2. Spalte)

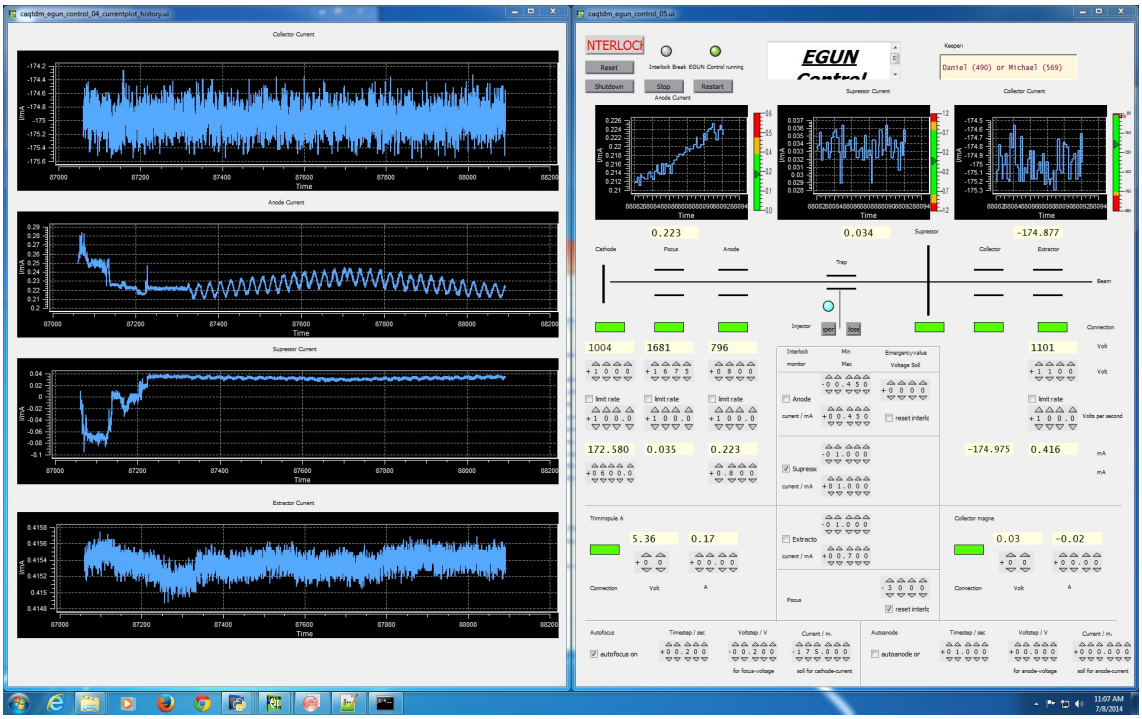

Abbildung 4.4: Screenshot Graphischer Benutzeroberflächen für Historie diverser Ströme für etwa 20 min (links) und Elektronenkanonensteuerung (rechts)

5 Messung

5.1 Vermessung Radiativer und Dielektronischer Rekombination von Iridium

In dieser Arbeit wurden radiative und dielektronische Rekombinationsenergien von hochgeladenen Iridiumionen vermessen. In der HD-EBIT wird eine Wolke von hochgeladenen Iridiumionen erzeugt. In ihr finden diverse Stoßprozesse (siehe Kap. 2) wie radiative und dielektronische Rekombination statt. Die so ausgestrahlten Photonen gelangen zum Röntgendetektor, der ihre Energie misst und sie zum Datenaufnahmesystem weiterleitet. Während den Messungen wurde die Elektronenstrahlenergie variiert. Ihr Signal wird zusammen mit dem Detektorsignal getriggert und zum Datenaufnahmesystem weitergeleitet (Abb. 5.1). Der Elektronenstrahlstrom wurde während einer Messung konstant gehalten.

Die Ereignisse sind in einem Photonenenergie-Elektronenstrahlenergie-Diagramm zusammengefasst. Durch die radiative Rekombination ergeben sich Streifenmuster, ideal mit einer Steigung von eins, da ihre Photonenenergie die Summe von Ioinisationsenergie (bleibt konstant) und kinetische Energie (wird variiert) des Elektrons entspricht. Die dielektronische Rekombination hingegen sorgt für ein punktartiges Muster, da sie resonanter Natur ist. Photonen- und Elektronenstrahlenergie sind hierbei fest. Diese Prozesse sind noch mit Breiten behaftet. Diese stammen von der natürlichen Linienbreite und der experimentellen Apparatebreite, welche Dopplereffekt und Messungenauigkeit zusammenfasst. Es wird angenommen, dass die Apparatebreite für alle Resonanzen einer Messung gleich bleibt. Zuletzt sind die Messwerte noch mit Rauschen (Streulicht und Dunkelstrom und andere Störeffekte) behaftet.

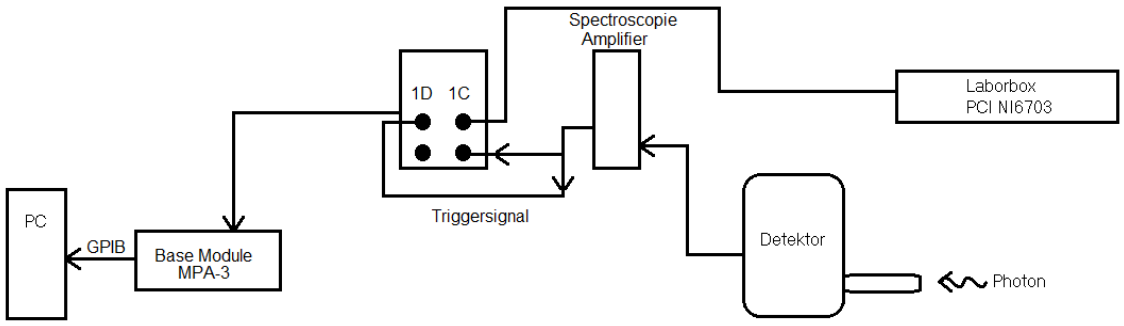

Abbildung 5.1: Schematische Darstellung der Datenaufnahme

5.2 Messparameter

Alle Messungen wurden an der HD-EBIT zwischen 26.06.2014 und 08.07.2014 durchgeführt. Vorherige- und Testmessungen werden bei dieser Arbeit nicht ausgewertet. Die Tabellen 5.1 und 5.2 zeigt alle für die Auswertung relevanten Messungen inklusive sämtliche Parameter.

5.3 Kalibration

5.3.1 Kalibration der Photonenenergie

Der Datenaufnahmesystem klassifiziert Photonen- und Elektronenstrahlenergie in Kanälen, jeweils 8192 Stück. Um sie Energien zuzuordnen, müssen Kalibrationsmessungen vorgenommen werden. Die Photonenenergiekanäle sind mit dem radioaktiven Element ^{241}Am kalibriert. Dazu wurde die Quelle vor dem Detektor positioniert und dieser hat ein Spektrum aufgenommen, welches charakteristische Linien aufweist, die mit Gaußprofilen gefittet wurden (wie in Abb. 5.4). Für einfache Peaks wurde die Fitfunktion GF(x) aus Gleichung 5.1 genommen.

$$GF(x) = \frac{A}{\sqrt{2\pi}\sigma} \times exp(-\frac{(x-\mu)^2}{2\sigma^2}) + y_0 \tag{5.1}$$

A entspriche dabei der gesamten Fläche zwischen Graph und Abszisse, μ dem Peakschwerpunkt, σ der Standartabweichung und y_0 dem Offset, also einem konstanten Hintergrund.

Einige Resonanzen liegen so nahe beieinander, dass die Fitfunktion einem Doppelpeak entspricht (siehe Abb. 5.2). Dazu wird zu der Fitfunktion einfach ein weiteres Gaußprofil dazu addiert. Gleichung 5.2 zeigt diese Funktion (DGF(x)).

$$DGF(x) = \frac{A1}{\sqrt{2\pi}\sigma 1} \times exp(-\frac{(x-\mu 1)^2}{2\sigma 1^2}) + \frac{A2}{\sqrt{2\pi}\sigma 2} \times exp(-\frac{(x-\mu 2)^2}{2\sigma 2^2}) + y_0 \tag{5.2}$$

Den Peakschwerpunkten wurden Linien aus dem bekannten ^{241}Am-Spektrum (Abb. 5.3) zugeordnet.

Die ersten drei Einzelmessungen (D1-D3) sowie die letzten zwei (D4, D5) sind aufaddiert. Die zugeordneten Wertepaare (Peakschwerpunkt vom aufgenommenen Spektrum, Photonenenergie) sind in Abbildung 5.5 gegeneinander aufgetragen. Ihr Verlauf zeigt ein lineares Verhalten, welhalb auch eine Lineare Funktion angefittet wurde. Die Residuen weisen keinen Trend auf, somit wurde die lineare Kalibration beibehalten. Tabelle 5.5 listet die Kalibrationsfunktionen auf.

5.3.2 Kalibration der Elektronenstrahlenergie

Die Kalibration der Elektronenstrahlenergie ist dagegen aufwändiger. Dielektronische Resonanzen würden sich analog zum ^{241}Am-Spektrum dazu eignen, jedoch wer-

Nr.	Beginn	Ende	t (min)	N	Mess-ziel	Abbruch	Folie	X-Kali	Y-Kali
1	26.06.2014 17:28	27.06.2014 00:43	437	8	DR	Programm abbruch	Al	D1,2,3	C7
2	27.06.2014 00:55	27.06.2014 08:41	466	10	DR	Manuell	Al	D1,2,3	C7
3	27.06.2014 18:06	27.06.2014 22:15	249	6	DR	Manuell	Al	D1,2,3	C7
4	27.06.2014 22:33	28.06.2014 06:05	452	9	DR	Icoll instabil	Al	D1,2,3	C7
5	28.06.2014 10:21	28.06.2014 18:22	480	10	DR	Manuell	Al	D1,2,3	C7
6	28.06.2014 20:46	29.06.2014 17:49	1263	51	Kante, W	Interlock	W	D1,2,3	C8
7	29.06.2014 21:31	29.06.2014 23:34	123	6	Kante, W	Manuell	W	D1,2,3	C8
8	02.07.2014 17:09	02.06.2014 23:30	381	18	Kante, Ta	Interlock	Ta	D4,5	C9
9	03.07.2014 16:12	04.06.2014 08:53	1001	22	DR	Manuell	Al	D4,5	C7
10	04.07.2014 17:41	05.06.2014 14:27	1246	21	DR	Manuell	Al	D4,5	C2
11	05.07.2014 16:10	05.06.2014 17:14	64	3	Kante, Al	Manuell	Al	D4,5	C4
12	06.07.2014 00:22	06.06.2014 06:56	394	17	Kante, Ta	Interlock	Ta	D4,5	C4
13	06.07.2014 14:39	07.06.2014 09:26	1127	49	Kante, Ta	Manuell	Ta	D4,5	C4
14	07.07.2014 18:19	07.06.2014 22:38	295	6	DR, Strom	Manuell	Al	D4,5	C5
15	07.07.2014 22:43	08.06.2014 05:18	395	8	DR, Strom	Manuell	Al	D4,5	C5
16	08.07.2014 05:25	08.06.2014 10:48	323	7	DR, Strom	Manuell	Al	D4,5	C5
17	08.07.2014 10:54	08.06.2014 13:18	144	3	DR, Strom	Manuell	Al	D4,5	C5

Tabelle 5.1: Parameter der Messungen. Nr: allgemeine Mess-Nummer, Beginn/Ende: Datum und Uhrzeit vom Start/Stop der Messung, t: Gesamtmesszeit, N: Anzahl der Einzelmessungen, die aufaddiert werden, Messziel: DR: für DR Messung, Kante: für RR Messungen, Strom: für Raumladungsmessung (Verschiebung des Peakschwerpunktes gegen den Strom), Abbruch: Art des Abbruches der Messung: Programmabbruch: Messprogramm hat selbst abgebrochen (Absturz), Manuell: Abbruch durch Laborant, Interlock: Interlock wurde ausgelöst, nachfolgende Messungen bleiben unbeachtet, bis Problem behoben wurde, Folie: Forlie vor dem Detektor: Aluminium, Wolfram oder Tantal, X-Kali: Nummer der Kalibration für Beschleunigungsspannung, Y.Kali: Nummer der Kalibration für Phtotonenenergie

Nr.	$\boldsymbol{I_{coll}}$ **(mA)**	$\boldsymbol{U_{eb}}$ **(kV)**	$\boldsymbol{U_{ee}}$ **(kV)**	$\boldsymbol{U_c}$ **(kV)**	$\boldsymbol{U_a}$ **(kV)**	$\boldsymbol{U_{ex}}$ **(kV)**	**SMIN (kV)**	**SMAX (kV)**
1	250	33	40	1,5	1	2,4	41	49
2	250	33	40	1,5	1	2,4	41	49
3	200	33	40	1,5	1,1	2,4	41	49
4	250	33	40	1,5	1,1	2,4	41	49
5	150	33	40	1,5	1,1	2,4	41	49
6	230	38	41,5	1,5	1,1	2,4	46,5	50,5
7	230	38	41,5	1,5	1,1	2,4	46,5	50,5
8	230	39	42	1,5	1,2	2,4	47	51
9	230	33	40	1,5	1,2	2,4	41	49
10	100	33	40,5	1,5	1,2	2,4	41	49,5
11	230	39	42	1,5	1,2	2,4	47,5	51
12	150	39	42	1	1	1,1	47,5	51
13	150	39	42	1	0,8	1,1	47,5	51
14	80	38	41,5	1	0,8	1,1	46,1	50
15	50	38	41,5	1	0,8	1,1	46,1	50
16	125	38	41,5	1	0,8	1,1	46,1	50
17	175	38	41,5	1	0,8	1,1	46,1	50

Tabelle 5.2: EGUN Parameter der Messungen. Nr: allgemeine Mess-Nummer, Icoll: Kollektorstrom, ($U_{eb,ee}$: minimale/maximale Spannung der Elektronenkanonenplattform, U_c: Kathodenspannung, U_a: Anodenspannung, (U_{ex}: Extraktorspannung (der Elektronenkanone), SMIN/SMAX: minimale/-maximale Beschleunigungsspannung für Datenaufnahmesystem

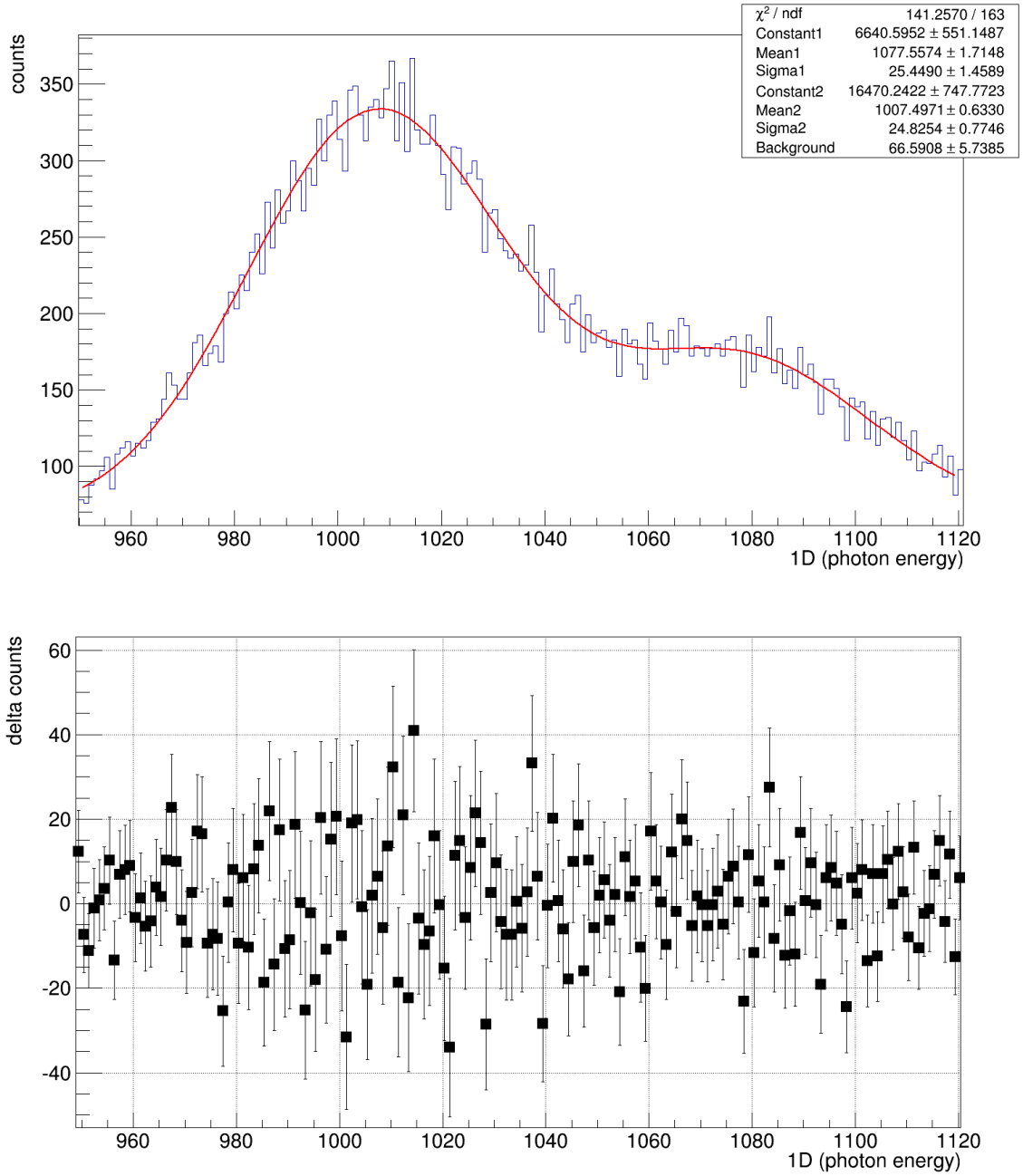

Abbildung 5.2: Doppelfit zweier naheliegender Peaks aus Photonenenergie-Kalibrationsmessung D1,D2,D3 (alle aufaddiert). Die Fitergebnisse sind rechts oben zu sehen: Chiqs/NDF ist das Chi-Quadrat / Anzahl der Freiheitsgrade, Constant(i) die totale Peakfläche, mu(i) der Peakschwerpunkt, Sigma(i) die Standartabweichung , Background der konstante Hintergrund und (i) die Peaknummer (1 für linken, 2 für rechten Peak)

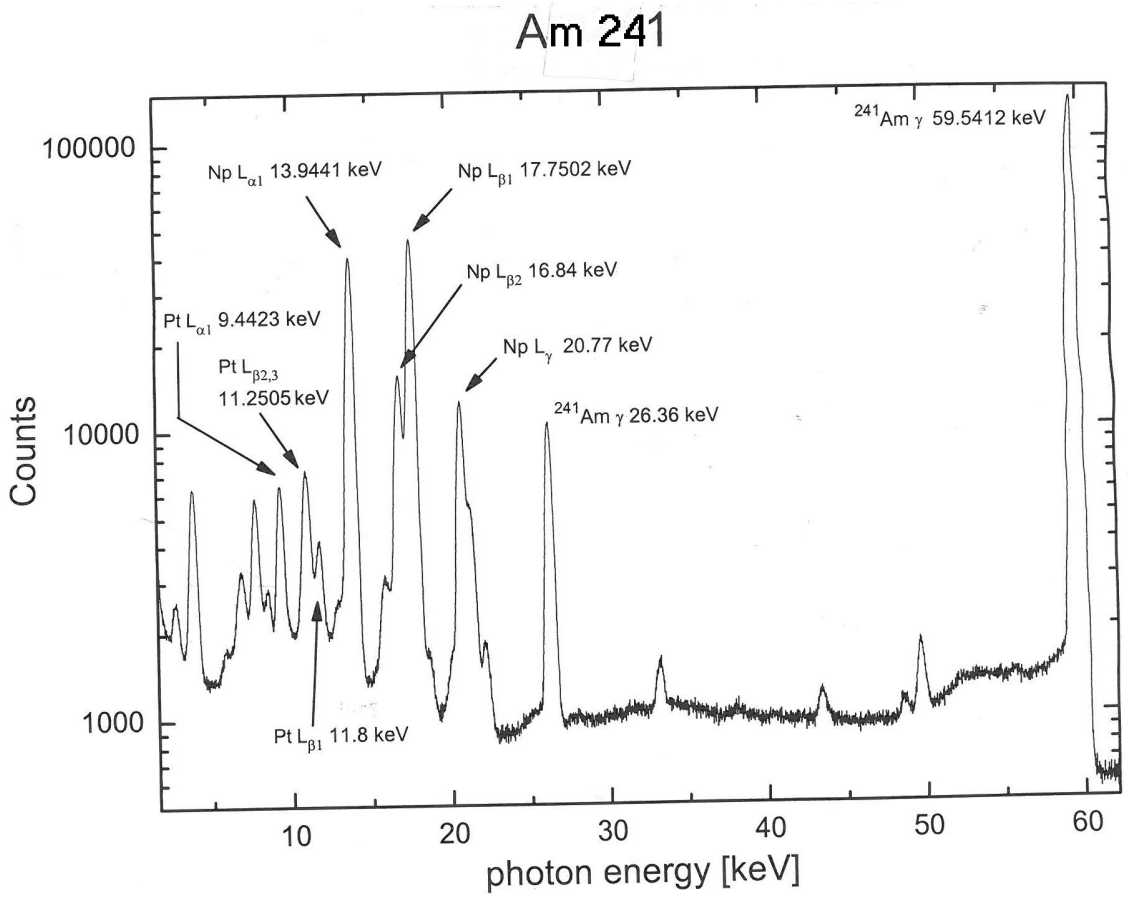

Abbildung 5.3: Bekanntes Spektrum von Am241

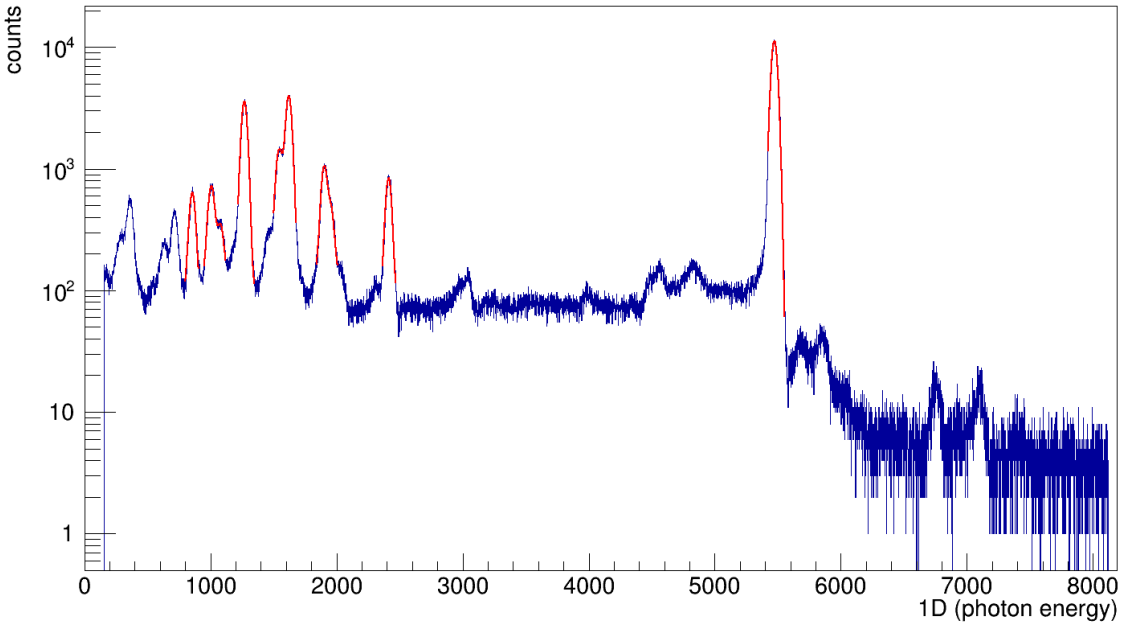

Abbildung 5.4: aufgenommesen Am-Spektrum von Y-Kalibration D1-D3 (blau) mit Gaußfits (rot), Energie hat logarithmische Skala

den sie in dieser Arbeit vermessen. Stattdessen wurden spezielle Kalibrationsmessungen vollzogen. Zuerst wurde die Beschleunigungsspannung der Elektronen im Fallenbereich kalibriert, danach eine Raumladungskorrektur durchgeführt.

Kalibration der Beschleunigungsspannung

Die Beschleunigungsspannung des Elektronenstrahls im Fallenbereich wurde ähnlich wie die Photonenenergie kalibriert, nur dass das charakteristische Spektrum der Spannung selbst erstellt wurde. Auf dem Kanaleingäng des Datenaufnahmesystems wurde eine analoge Spannung gesetzt, die mit der Beschleunigungsspannung der Elektronen im Fallenbereich linear zusammenhängt (Abb. 5.1).

$$U_{Kanaleingang} = m \times U_{acc} + b \tag{5.3}$$

U_{acc} ist die Beschleunigungsspannung nach Gleichung 3.1. Die Spannungen der Fallen-Driftröhre (DT9) und Driftröhrenplattform wurde von Spannungsteiler der Driftröhren zusammen gemessen, die der Elektronenkanonenplattform von einem anderen Spannungsteiler und die Kathodenspannung vom Netzteil dafür selber. Diese drei Geräte werden vom Kontrollsystem (siehe Kap. 4) ausgelesen. Die Spannung am Kanaleingang wurde von einem Server generiert. Die Parameter m und b für den Kanaleingang sind so gewählt, dass im Beschleunigungsspannungsbereich, die

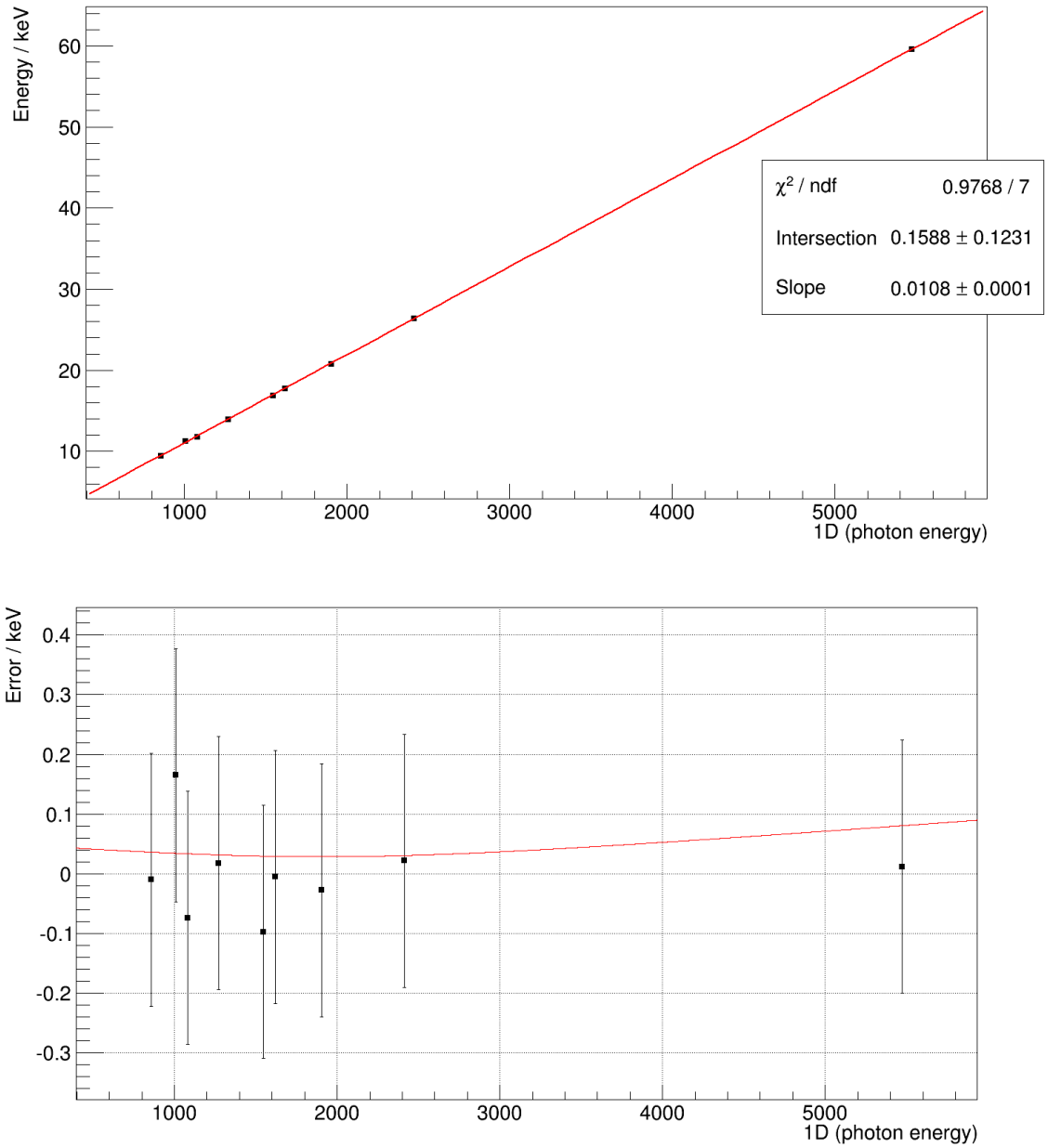

Abbildung 5.5: Y-Kalibration von D1-D3, Kanäle gegen Energie aufgetragen, links: Wertepaare (schwarz) mit Fitfunktion (rot), rechts: Residuen (schwarz) mit 68% Vertrauensband (rot)

Grenzen in 5.2 als SMIN und SMAX bezeichnet sind und der Kanaleingang von 1 V bis 9 V variiert.

$$m = (9V - 1V)/(SMAX - SMIN) \tag{5.4}$$

$$b = 1V - m \times SMIN \tag{5.5}$$

Um eine Kalibrationskurve zu erstellen, wurde Beschleunigungsspannung sprunghaft verändert und von einem Loggingprogramm mitgemessen, sodass sich Spektren mit einer bestimmten Anzahl an Peaks ergebenen. Das Spektrum vom Datenaufnahmesystem weist ein ähnliches Spektrum mit den selben charakteristischen Peaks auf. Abbildung 5.6 zeigt ein jeweils ein Spektrum vom Datenaufnahmesystem und vom Loggingprogramm. Die charakteristischen Peaks wurden mit Gaußfunktionen wie in der Photonenenergiekalibration gefittet.

Auch hier wurden die Wertepaare gegeneinander aufgetragen und mit einer linearen Funktion gefittet (siehe Abb. 5.7). Tabelle 5.7 enthält alle Kalibrationsfunktionen.

Diese Kalibration gibt nicht gänzlich der Elektronenstrahlenergie wieder, da noch Raumladungseffekte vom Elektronenstrahl hinzugezogen werden müssen.

Raumladungskorrektur

Die Raumladung wird durch den Elektronenstrahl selbst hervorgerufen. Er ist negativ geladen und bremst die ankommenden Elektronen ab. Dieser Effekt wird durch die positiv geladene Ionenwolke teilweise kompensiert. Man nimmt an, dass vom Raumladungseffekt die Anteile der Elektronen und der Ionenwolke unabhängig vom Strom sind, der gesamte Effekt allerdings proportional zum Strom und antiproportional zur Geschwindigkeit der Elektronen (siehe Kap. 3). Zur Berechnung der raumladungskorrigierten Elektronenstrahlenergie wird Gleichung 3.8 verwendet.

Hierfür werden bei verschiedenen Strömen Elektronenstrahlenergie-Peakschwerpunkte dreier Peaks vermessen (per Gaußfits) und gegen den Strom aufgetragen (siehe Abb. 5.9). Es ergibt sich nahezu eine Gerade, die zu einem Strom von 0 mA extrapoliert wird. Die Steigung und die korrigierte Energie von den einzelnen Peaks wurde gemittelt. Abbildung 5.8 zeigt diese genutzten Peaks bei Messung Nr. 9 (Strom=230mA)

$$Raumladungssteigung = (3.734 \pm 0.024) \times 10^{-4} \frac{keV}{mA} =: S_{sp} \tag{5.6}$$

$$korrigierteEnergie = (47.4320 \pm 0.0004) keV =: E_{sp} \tag{5.7}$$

Die Raumladungskorrektur beträgt für Messung Nr. 9 (siehe DR Messung) ungefähr 90 eV. Bei ca. 45 keV Gesamtenergie entspricht das einen relativen Anteil von nur 0,2 %. Somit kann die Näherung in Gleichung 3.8 angewandt werden. Abbildung 5.10 zeigt die Differenz von der unkorrigierten zur raumladungskorrigierten Elektronenstrahlenergiekalibrierung in Abhängigkeit der Elektronenstrahlenergie.

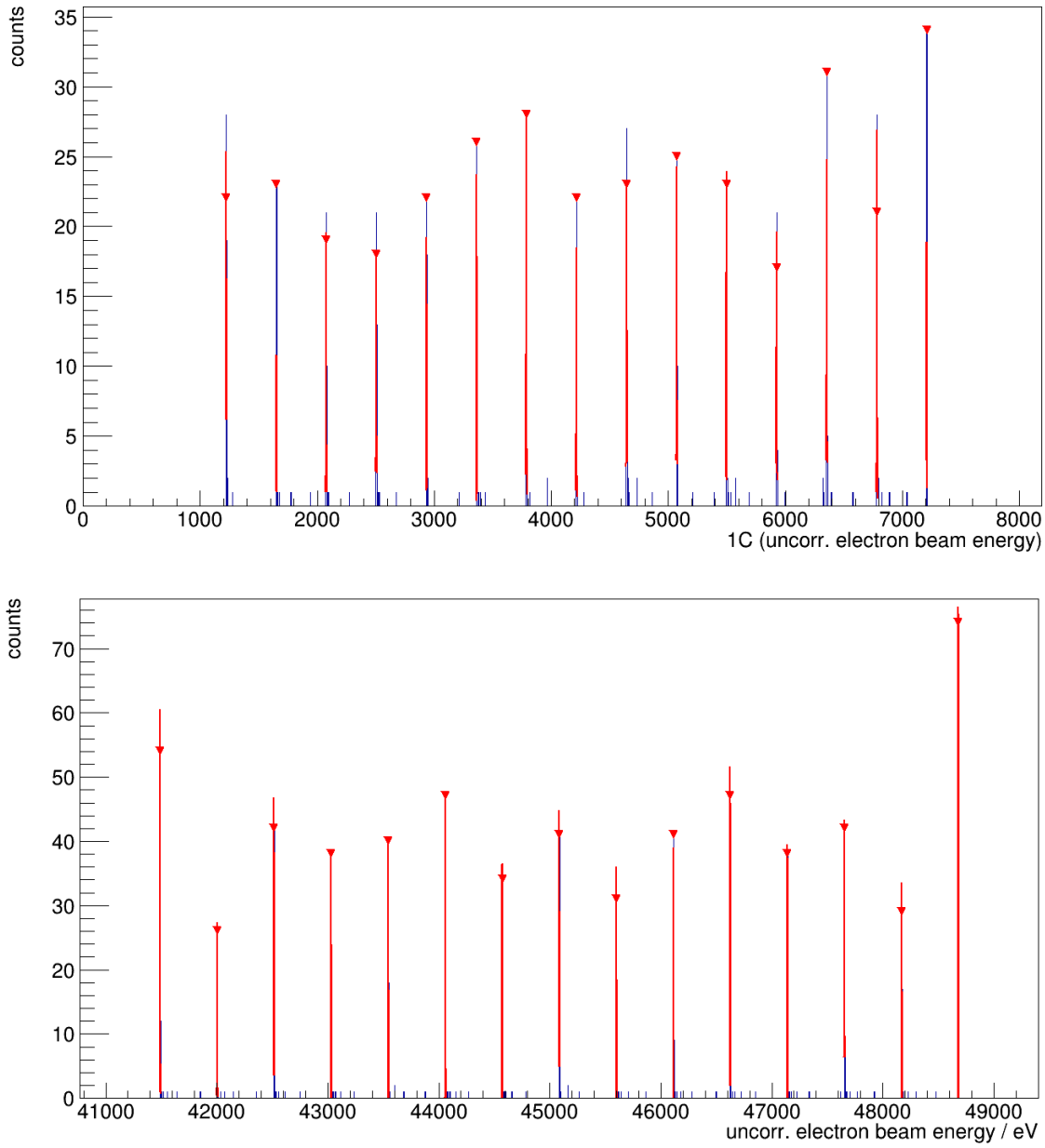

Abbildung 5.6: X-Kalibration Spektren von Nr. 7 vom Datenaufnahmesystem (links) und vom Loggingprogramm (rechts), rote Linie: Gauß-fitfunktion, blau: Rohdaten, roter Pfeil: gefundener Peak (vom Auswertungsprogramm)

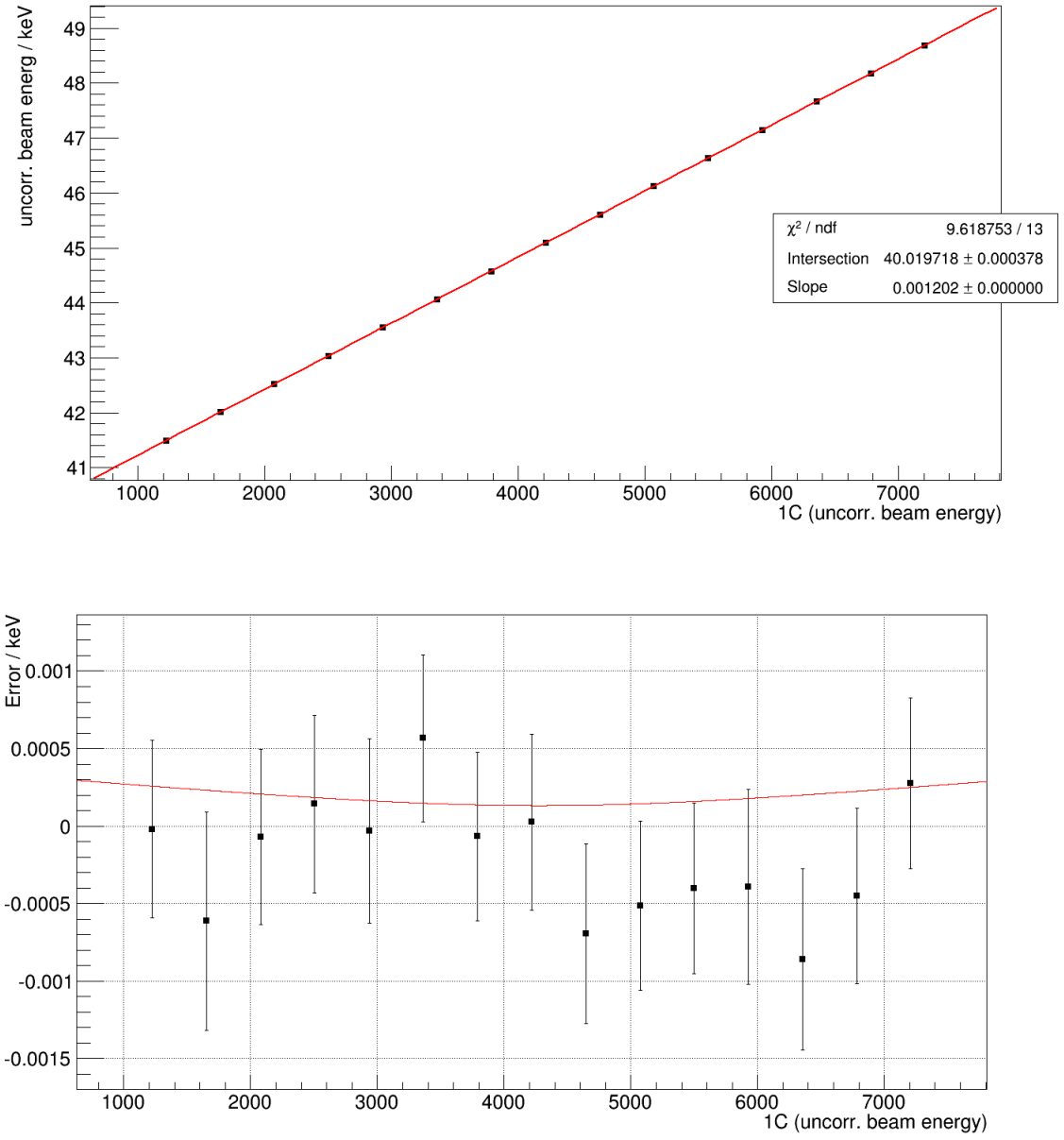

Abbildung 5.7: X-Kalibration von C7, Kanäle gegen Energie (noch nicht Raumladungskorrigiert, entspricht somit Beschleunigungsspannung) aufgetragen, links: Wertepaare (schwarz) mit Fitfunktion (rot), rechts: Residuen (schwarz) mit 68% Vertrauensband (rot)

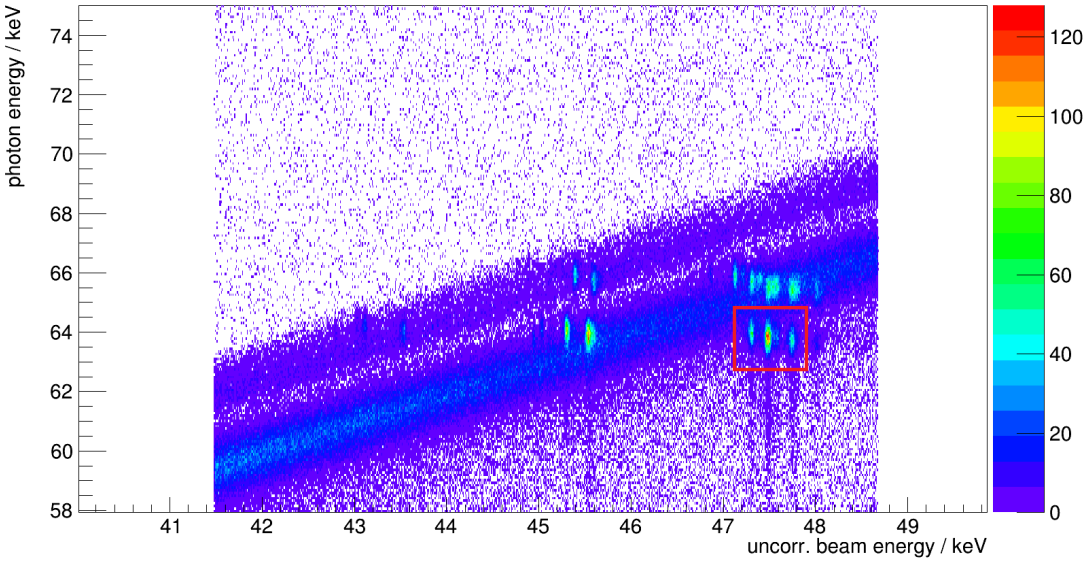

Abbildung 5.8: Messung Nr. 9 für DR Vermessung bei einem Elektronenstrahlstrom von 230 mA. Die Peaks für die Raumladungsermittlung sind rot eingerahmt.

Fehlerbehandlung

Ein Ziel der Auswertung ist es, die gemessenen Energiewerte mit systematischen und statistischen Fehler zu versehen. Ein relativ geringer statistischer Fehler zeigt, dass die Messung genügend Statistik hat, jedoch eine verbesserungswürdige Systematik. Im umgekehrten Fall ist die Systematik in Ordnung, jedoch wurden zu wenig Daten gesammelt. Durch solche Informationen lassen sich künftige Messungen optimieren. Generell gilt, dass sich statistische Fehler eben durch mehr Daten verringern lassen, z.B. durch längere Messzeiten.

Schon in die Kalibration der Photonenenergie gehen systematischer und statistischer Fehler mit ein. Die Fehlerangaben der Kalibrationslinien sind systematischer Natur. Zu dem eigentlichen Fehler wird noch die Detektorauflösung von ca. 500 eV quadratisch hinzuaddiert. Vom aufgenommenen Spektrum sind Peaks gefittet. Ihre Peakschwerpunkte bilden das Pendant zu den Linien, welche mit statistischen Fitfehlern behaftet sind. Aus dem Fit der Kalibrationslinie geht auch das 68%-Vertrauensband hervor. Es beschreibt den Fehler der Kalibrationslinie, der sowohl systematischen als auch statistischen Fehler enthält. Um den systematischen Fehler daraus Abzuschätzen, wird der systematische Anteil der Rohdaten genutzt. Der systematische Anteil f_{syst} ([Hollain, 2012]) ist hier definiert als:

$$f_{syst} := \frac{E_{syst}^2}{E_{syst}^2 + E_{stat}^2}. \tag{5.8}$$

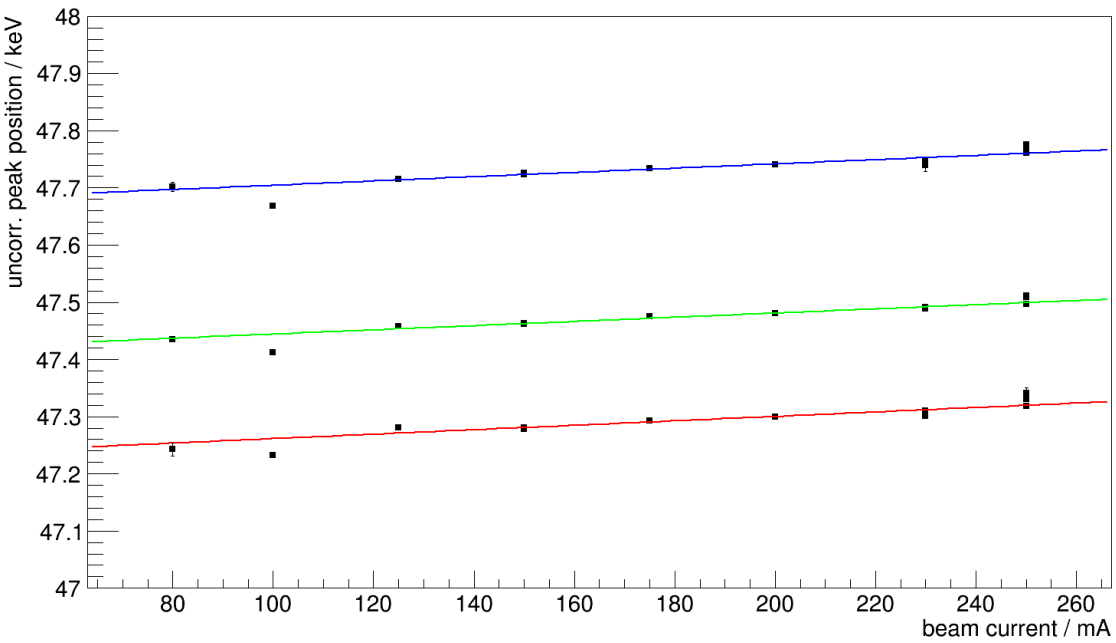

Abbildung 5.9: Peakschwerpunkte bei verschiedenen Strömen. Einige Messpunkte liegen nahe beieinander. Die Geraden sind lineare Fits dazu. Blau: Peak Nr. 32, Grün: Peak Nr. 28, Rot: Peak Nr. 23. Steigung und Achsenabschnitt mitteln sich zu $3.734(24) \times 10^{-4}$ keV/mA bzw. 47.4320(4) keV.

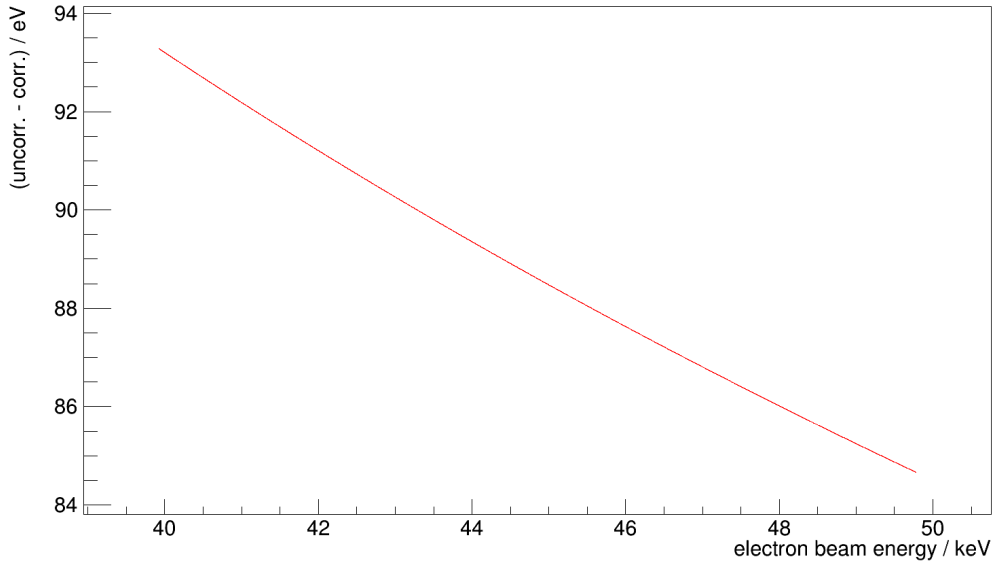

Abbildung 5.10: Differenz der Elektronenstrahlkalibrierung: unkorrigiert zu raumladungskorrigiert in Abhängigkeit der Elektronenstrahlenergie.

E_{syst} ist dabei der statistische Fehler, E_{stat} der statistische Fehler und E_V der Vertrauensbandfehler. Für Anteile zwischen den Messpunkten der Rohdaten wird linear interpoliert, bzw. außerhalb davon wird das genutzt was am nächsten ist. Die Fehler lassen sich aus dem Vertrauensband dementsprechend zerlegen:

$$E_{syst} = \sqrt{E_V^2 \times f_{syst}} \tag{5.9}$$

$$E_{stat} = \sqrt{E_V^2 \times (1 - f_{syst})} \tag{5.10}$$

Die quadratische Addition der Fehler ergibt dann wieder den Vertrauensbandfehler:

$$E_{syst}^2 + E_{stat}^2 = E_V^2 \tag{5.11}$$

Gleiches gilt für die Kalibration der Elektronenstrahlenergie.

Die Raumladungskorrektur ergibt sich aus der Steigung des Peakschwerpunkt-Strom-Geraden. Der Steigungsfehler entspricht dem Fitfehler. Der systematische Anteil wird mit Hilfe aller Fitpunkte abgeschätzt: er entspricht dem Verhältnis der quadratischen Summen aller systematischen Fehler zu dem Gesamtfehlern. Die entgültige Steigung mit Fehler entsteht aus der Mittlung aller drei Einzelsteigungen. Der statistische Fehler übersteigt hier den systematischen Fehler um mehrere Größenordnungen, was auf die begrenzten Anzahl an Messpunkten zurückzuführen ist. Letztendlich wird nur der statistische Fehler angegeben.

Die Ionisationspotentiale aus der radiativen Rekombinationsmessung ergeben sich aus einer nicht linearen Fitfunktion. Dennoch könnte Sie auch bei unkalibrierten Daten eingesetzt werden. Der entscheidende Bereich ist der bei dem sich die RR-Linie und die Absorptionskante schneiden. Daher wird der Kalibrationsfehler aus diesem Schnittpunkt quadratisch zum Fitfehler hinzu addiert.

Die Fehler der dielektronischen Rekombinationsresonanzen ergibt sich wie oben beschrieben aus den quadratischen Additionen von dem Vertrauensbandfehler ihren jeweiligen Positionen auf den Energieachsen und aus dem Fitfehler für Peakschwerpunkt (statistisch).

5.4 Radiative Rekombination

Die radiative Rekombination erzeugt ein Streifenmuster im Photonenenergie - Elektronenstrahlenergie - Diagramm. Einzelne RR Linien sind so nahe beieinander, dass die nicht mehr aufgelöst werden können. Der Drehimpuls J des Ions ist bei solchen sog. Bändern gleich. In dieser dieser Arbeit sind 2 Bänder mit jeweils J=1/2 (He-, Li-, Be-, B-arig) und J=3/2 (C-, N, O-, F-artig) erkennbar. Um sie dennoch aufzulösen wurden Wolfram- oder Tantalfolien vor dem Detektor positioniert. Ihre Absorptionskurve weist einen sprunghaften Anstieg im Röntgenbereich auf (siehe Abb. 5.11), dessen Breite kleiner die Anstände der einzelnen RR Linien eines Bandes ist.

Durch geeigneten Elektronenstrahlenergiebereich erhält man einen „Schnitt“durch ein Band. In diesem Schnittbereich werden die einzelnen RR Linien unterschiedlich

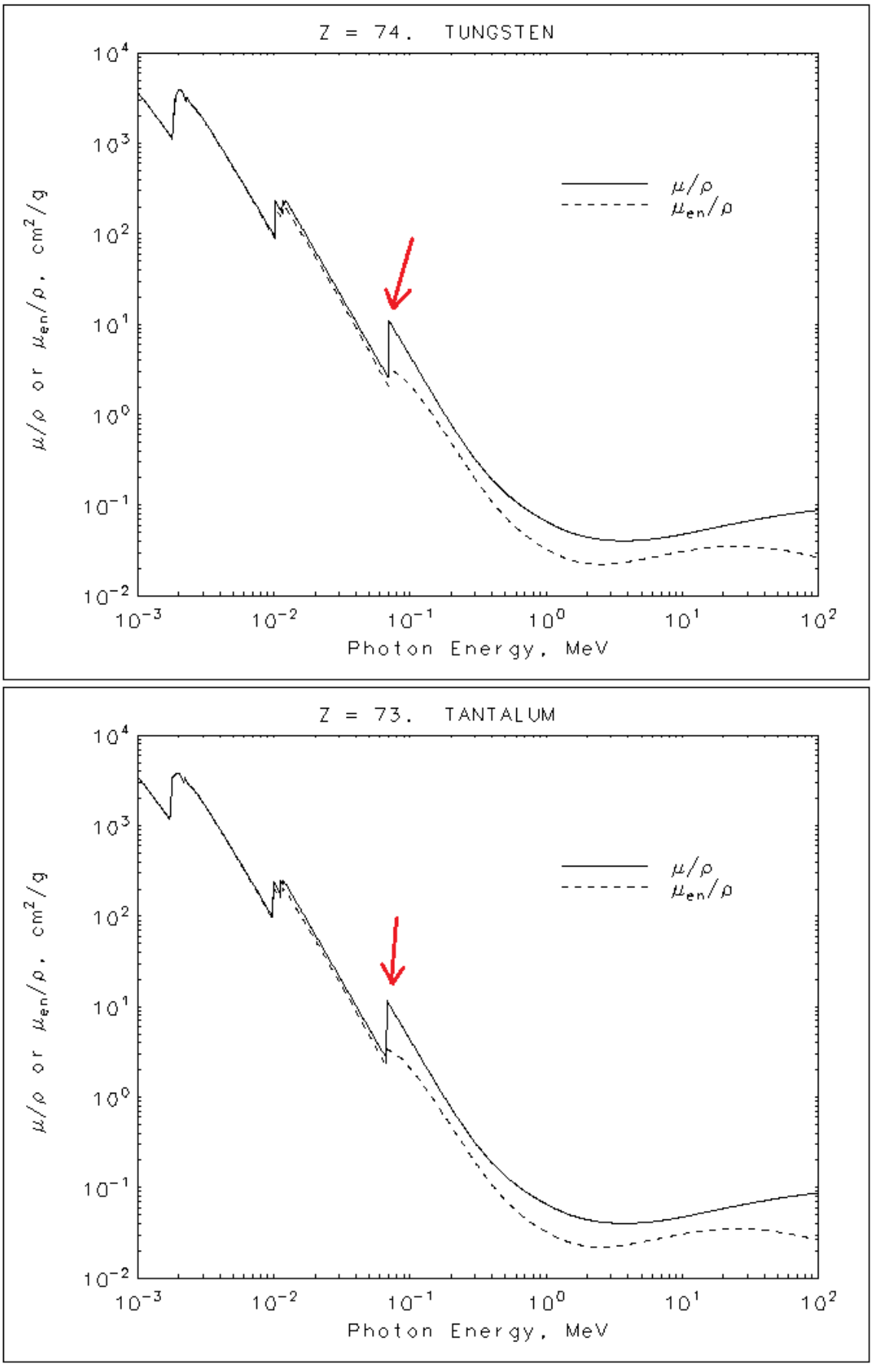

Abbildung 5.11: Absorbtionskurven für Wolfram und Tantal [NIST] in doppellogarithmischer Darstellung. Dabei ist die Photonenenergie gegen den Absorbtionskoeffitienten/Dichte aufgetragen. Die Absorptionskanten sind mit roten Pfeilen markiert.

stark absorbiert, wodurch Abstufungen im Intensitätsverlauf erkennbar werden. Die Positionen dieser Abstufungen lassen Rückschlüsse auf die Ionisationsenergie der einzelnen RR Linien zu. Der Bereich, wo sich Absorptionskante und Band schneidet, wird ausgeschnitten und auf die Elektronenstrahlenergieachse projiziert. Abbildung 5.12 zeigt schematisch die Verfahrensweise. Das resultierende Histogramm weist auch Abstufungen auf. An diesem wird eine entsprechende Funktion gefittet, die die Ionisationspotentiale als freie Parameter enthält.

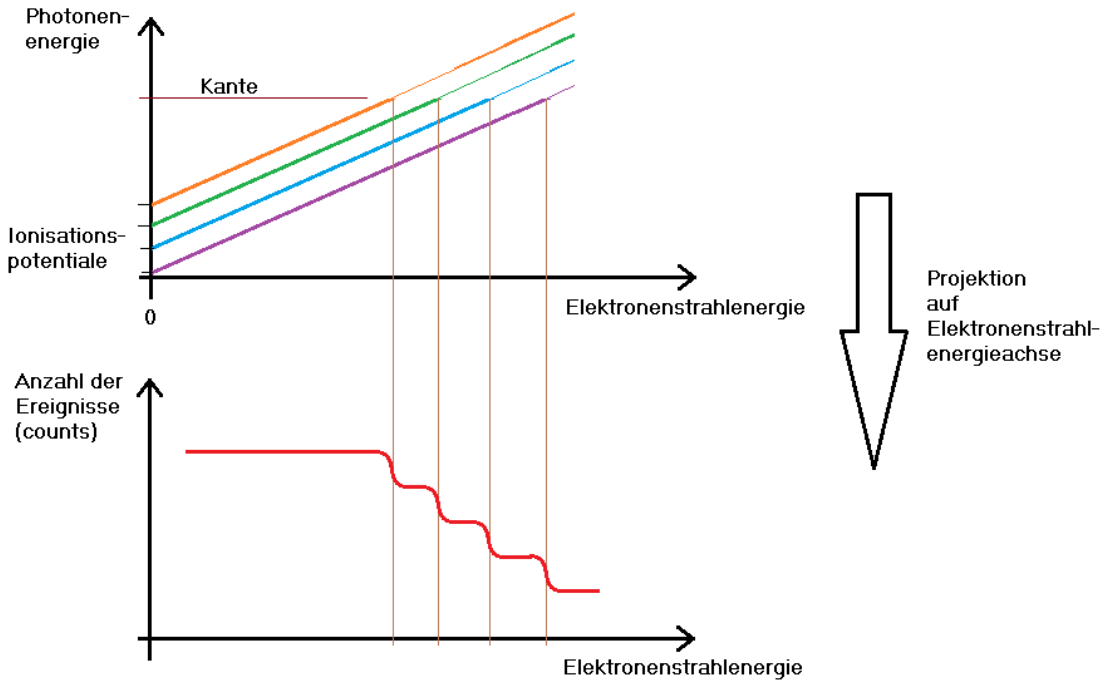

Abbildung 5.12: Vermessung einzelner RR Linien: oberes Diagramm: Photonenenergie-Elektronenstrahlenergie-Diagramm. Zu sehen sind RR Linien als steigende Geraden. Die Absorptionskante (horizontale Linie) schwächt abrupt die Linien ab. Der vom Diagramm gezeigten Teil ausgeschnitten und auf die Elektronenstrahlenergieachse projiziert (unteres Diagramm). Abstufungen sind erkennbar und ein entsprechender Fit wird vollzogen.

Für die Fitfunktion $F_{RR}(x)$ werden folgende vereinfachte Annahmen gemacht. Eine einzelne RR-Linie besteht aus einer Geraden mit Steigung 1.0, wie auch in der Theorie, und streut gaußartig in der Photonenenergie. Die Streuung in der Elektronenstrahlenergie wird vernachlässigt. Die Absorptionskante schneidet sprunghaft die Linie mit Streuung ab. Zusätzlich gibt es noch ein gleichmäßiger Hintergrund, die von der Absorptionskante nicht beeinflusst wird. Die Breite aller einzelnen RR-Linine ist gleich, zumindest sollten sie von nahezu gleicher Größe sein [Crespo14]. Für eine einzelne RR-Linie wird somit eine Gausfunktion von $-\infty$ zur Absorptionskante integriert.

$$Gerade(x) = Ipot + 1.0 \times x \tag{5.12}$$

$$\Delta y(x) := K - Gerade(x) = K - Ipot - x \tag{5.13}$$

$$Ig(x, Ipot, A) = \int_{-\infty}^{\Delta y(x)} \frac{A}{\sqrt{2\pi}\sigma} exp(\frac{-x^2}{2\sigma^2})dx = \frac{A}{2}(1 + erf(\frac{K-Ipot-x}{\sqrt{2}\sigma})) \tag{5.14}$$

$$F_{RR}(x) = H + \sum_i Ig(x, Ipot(i), A(i)) \tag{5.15}$$

Mit A als Linienstärke, K als Absorbtionskante, i als Indize einer einzelnen RR-Linie, σ als Linienbreite, H als Hintergrund, Ipot als Ionisationspotential und erf() als die Fehlerfunktion $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-k^2} dk$. Die Absorptionskante ist als Parameter fixiert und ihr Wert stammt von [R. D. Deslattes et al., 2003].

Nr.	**Folie**	$\boldsymbol{I_{coll}}$ **(mA)**	**Bez.**	**K (kV)**	**Nrs**
1	Wolfram	230	K edge (c)	69.5249(25)	6, 7
2	Tantal	230	K edge (c)	67.41124(79)	8
3	Tantal	150	K edge (c)	67.41124(79)	12, 13

Tabelle 5.3: Parameter der Kantenmessungen. Nr.: Nummer die Kantenmessung (1-3), Icoll: Elektornenstrahlstrom, Bez.: Bezeichnung der Absorptionskante in [R. D. Deslattes et al., 2003], K: Absorptionskante Wert mit Fehler in runde Klammer, Nrs: aufaddierte Messungen (entspricht einer Kantenmessung)

Die Ionisationspotentiale sind mit denen aus der Theorie ([NIST], [Scofield03]) verglichen, siehe Tabelle 5.8. Kantenmessung Nr. 2 hat vergleichsweise nur wenig Statistik, daher wurde die Breite der Linen aus Kantenmessung Nr. 3 entnommen und als Parameter der Fitfunktion fest fixiert. Kantenmessung Nr. 3 hat zwar mehr Statistik, jedoch konnte das energieniedrigste Ionisationspotential (F-artig) nicht mehr gefittet werden, da das Signal dafür zu schwach ist.

5.5 Dielektronische Rekombination

Die Dielektronische Rekombinatione ist ein resonanter Prozess und macht sich als Punkt im Diagramm bemerkbar. Zur Vermessung werden aus dem Diagramm sogenannte ROIs, *regions of interest* zu Deutsch interessante Bereiche, die diese Art von Resonanzen enthalten herausgeschnitten und auf die Elektronenstrahlenergie- bzw. Photonenenergieachse projiziert. Der Peakschwerpunkt der Elektronenstrahlenergie entspricht der Anregungsenergie. Die Resonanzen für die Photonenenergie werden nur mit Gausprofilen gefittet. Um bei den Projektionen die Variation der Zählrate entlang der Strahlenenergie durch die RR-Linien zu vermeiden, haben die ROIs die selbe Steigung (eins) wie der RR Linien. Hier wird die Messung Nr. 9 (Kollektorstrom = 230 mA) verwendet. Intensitätsreiche Resonanzen lassen sich problemlos erkennen, um jedoch auch die Positionen schwacher Resonanzen zu finden, werden

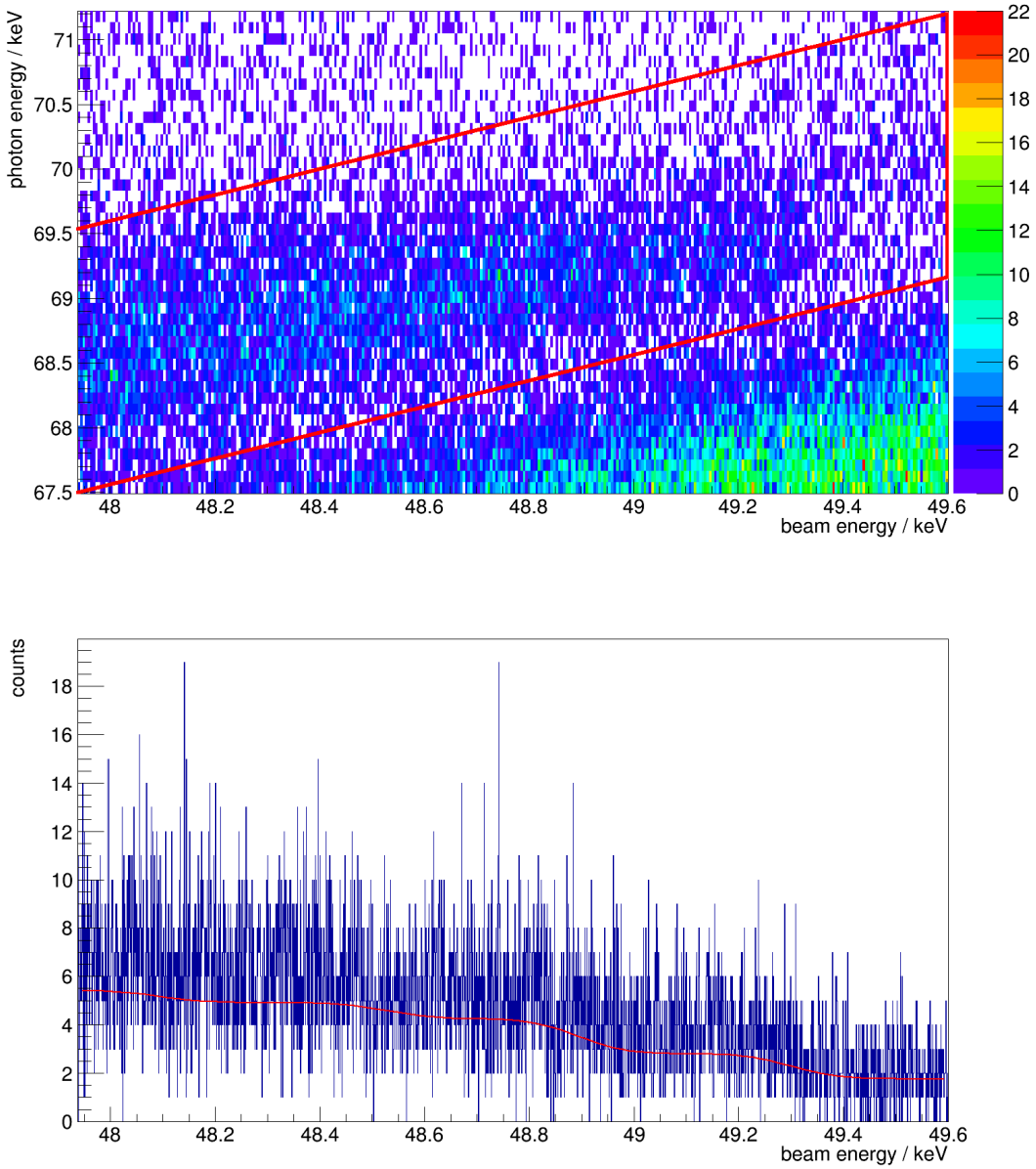

Abbildung 5.13: Radiative Rekombination vom J=1/2 - Band, abgeblendet durch eine Wolframfolie (Messungen von Nr. 6 und 7 aufaddiert). Im oberen Diagramm ist Elektronenstrahlenergie gegen Photonenenergie gebinnt (mit Faktor 8) aufgetragen und ein Ausschnittsrahmen (rot) zu sehen. Im unteren Diagramm ist die Projektion der Ereignisse (ungebinnt) im Rahmen auf die Elektronenstrahlenergie zu sehen (blau) mit Fitfunktion (rot).Der Fit wurde an den ungebinnten Daten vollzogen.

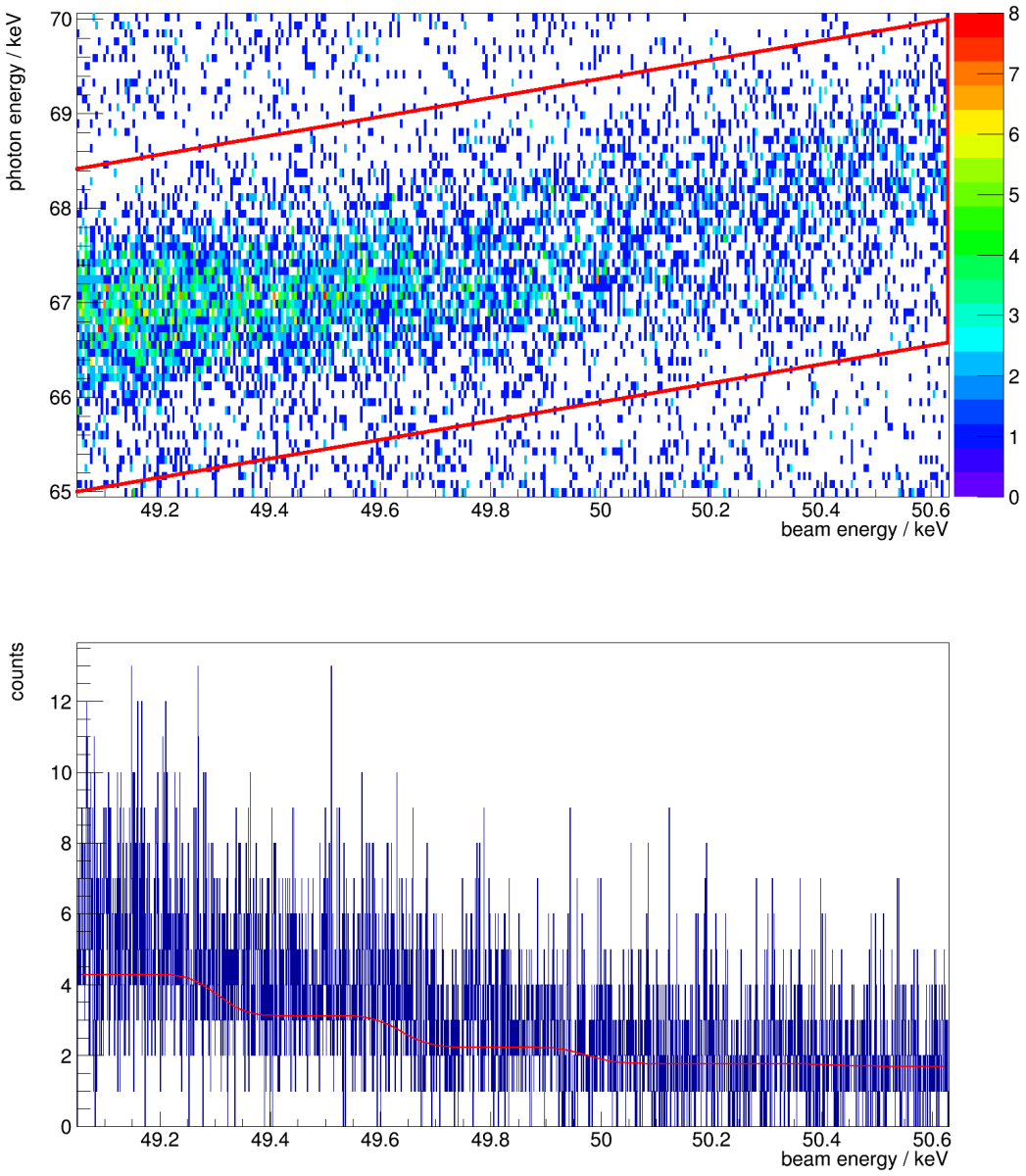

Abbildung 5.14: Radiative Rekombination vom J=3/2 - Band, abgeblendet durch eine Tantalfolie (Messung Nr. 8). Im oberen Diagramm ist Elektronenstrahlenergie gegen Photonenenergie gebinnt (mit Faktor 8) aufgetragen und ein Ausschnittsrahmen (rot) zu sehen. Im unteren Diagramm ist die Projektion der Ereignisse (ungebinnt) im Rahmen auf die Elektronenstrahlenergie zu sehen (blau) mit Fitfunktion (rot).Der Fit wurde an den ungebinnten Daten vollzogen.

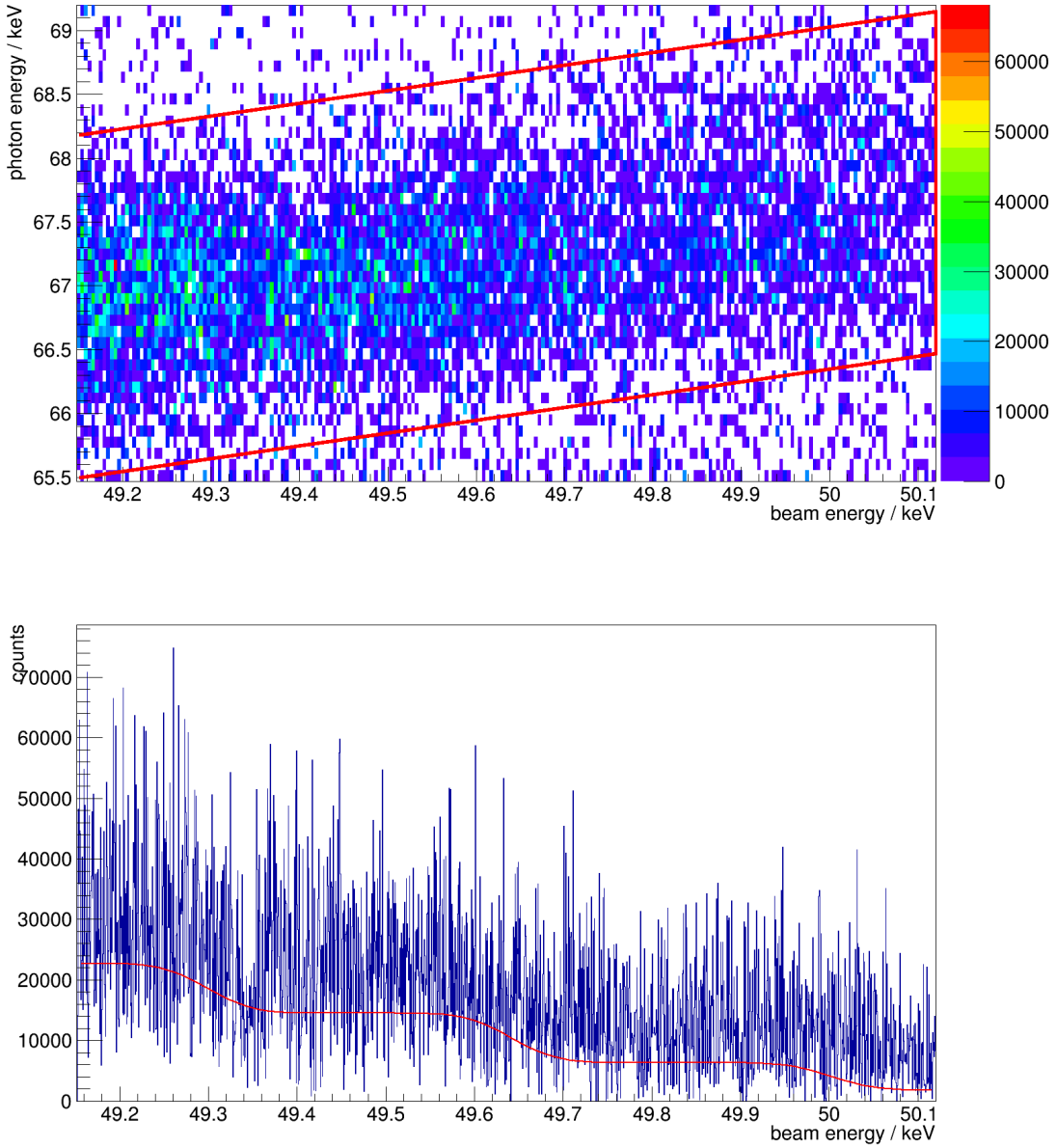

Abbildung 5.15: Radiative Rekombination vom J=3/2 - Band, abgeblendet durch eine Tantalfolie (Messungen von Nr. 12 und 13 aufaddiert). Im oberen Diagramm ist Elektronenstrahlenergie gegen Photonenenergie gebinnt (mit Faktor 8) aufgetragen und ein Ausschnittsrahmen (rot) zu sehen. Im unteren Diagramm ist die Projektion der Ereignisse (ungebinnt) im Rahmen auf die Elektronenstrahlenergie zu sehen (blau) mit Fitfunktion (rot).Der Fit wurde an den ungebinnten Daten vollzogen.

zuerst ROIs, die sich über die ganze Elektronenstrahlenergieachse ziehen qualitativ untersucht. Unter Zunahme der theoretischen Anregungsenergien können somit auch Zuordnungen getroffen werden. Primäres Ziel ist die Vermessung der Resonanzen, die theoretischen Werte dienen als Vergleich. Die Breiten der Linien bestehen aus der natürlichen Linienbreite, gefaltet mit dem Apparateprofil, hierfür wird ein Gaußprofil angenommen, des Experiments. Um die Breite des Apparateprofils zu erhalten, wird eine möglichst schmale Resonanz benötigt, in dieser Messung wird die schmalste He-artige Resonanz genommen (Nr. 0 der von den DR-Resonanzen, sie die theoretisch und experimentell schmalste). Sie wird mit einem Gausfit gefittet und von der gemessenen Breite wird die theoretische quadratisch abgezogen. Das Ergebnis wird für die Apparatebreite genommen und entspricht 18.83 eV. Die übrigen Resonanzen werden mit einer Faltung aus Fano- (F(x)) und Gaußprofil (G(x)) (für asymmetrische Peaks), in Gleichung 5.19 als FGF(x) bezeichnet, oder Lorenzs- (L(x)) und Gaußprofil, entspricht Voigtprofil (für symmetrische Peaks) gefittet.

$$G(x) = \frac{\sqrt{8ln(2)}}{\sqrt{2\pi}FWHM_{exp}} \times exp(-\frac{(x-\mu)^2 8ln(2)}{2FWHM_{exp}^2}) \tag{5.16}$$

$$L(x) = \frac{\Gamma}{2\pi}\frac{1}{x^2 + (\frac{\Gamma}{2})^2} \tag{5.17}$$

$$F(x) = \frac{2}{Q^2\Gamma\pi}(\frac{(\epsilon+Q)^2}{\epsilon^2+1} - 1), \epsilon := \frac{2(x-\mu)}{\Gamma} \tag{5.18}$$

$$FGF(x) = \frac{2A}{FWHM_{exp}}\sqrt{\frac{ln(2)}{\pi}}((1-\frac{1}{Q^2})Re(\omega(z)) - \frac{2}{Q}Im(\omega(z))) + y_0 \tag{5.19}$$

$$z := \frac{\sqrt{ln(2)}}{FWHM_{exp}}(2(\mu - x) + i\Gamma) \tag{5.20}$$

A ist wieder die totale Peakfläche, $FWHM_{exp}$ die Apparatebreite, Γ die natürliche Linienbreite, μ der Peakschwerpunkt, Q der Fano-Faktor und y_0 der Offset. $\omega()$ ist die komplexe Fehlerfunktion (Fehlerfunktion mit komplexen Argument). Fano und Fano-Gauß-Faltung richten sich an denen von [Martínez, 2005]. Dabei wird die Apparatebreite $FWHM_{exp}$ fixiert und die natürliche Linienbreite Γ als freier Parameter eingesetzt. Die Breiten entsprechen den vollen Halbwärtsbreiten. Die Ergebnisse werden mit theoretischen Vorhersagen verglichen. Abbildung 5.16 zeigt ein Fit eines asymmetrischen Peaks mit einem Fanoprofil mit den Residuen und ein Vergleichbarer Gaußfit. Man erkennt dass der Fano-Gauß-Faltungs-Fit die Daten besser beschreibt, was auch von einem reduzierten Chi-Quadrat von 0.96 untermauert wird gegenüber einem reduzierten Chi-Quadrat von 1.42 vom Gaußfit.

Für Messungen dielektronischer Resonanzen wurde eine Aluminiumfolie vor dem Detektor positioniert. Sie hat vor allem die Aufgabe, niederenergetische Photonen (<10 keV), welche unnötige Ereignisse hervorrufen, zu absorbieren. Ihre Absorbtionskante liegt bei ca. 1.56 keV, was weit unter dem hier interessanten Bereich liegt. Abbildung 5.17 zeigt die Absorbtion von Aluminium.

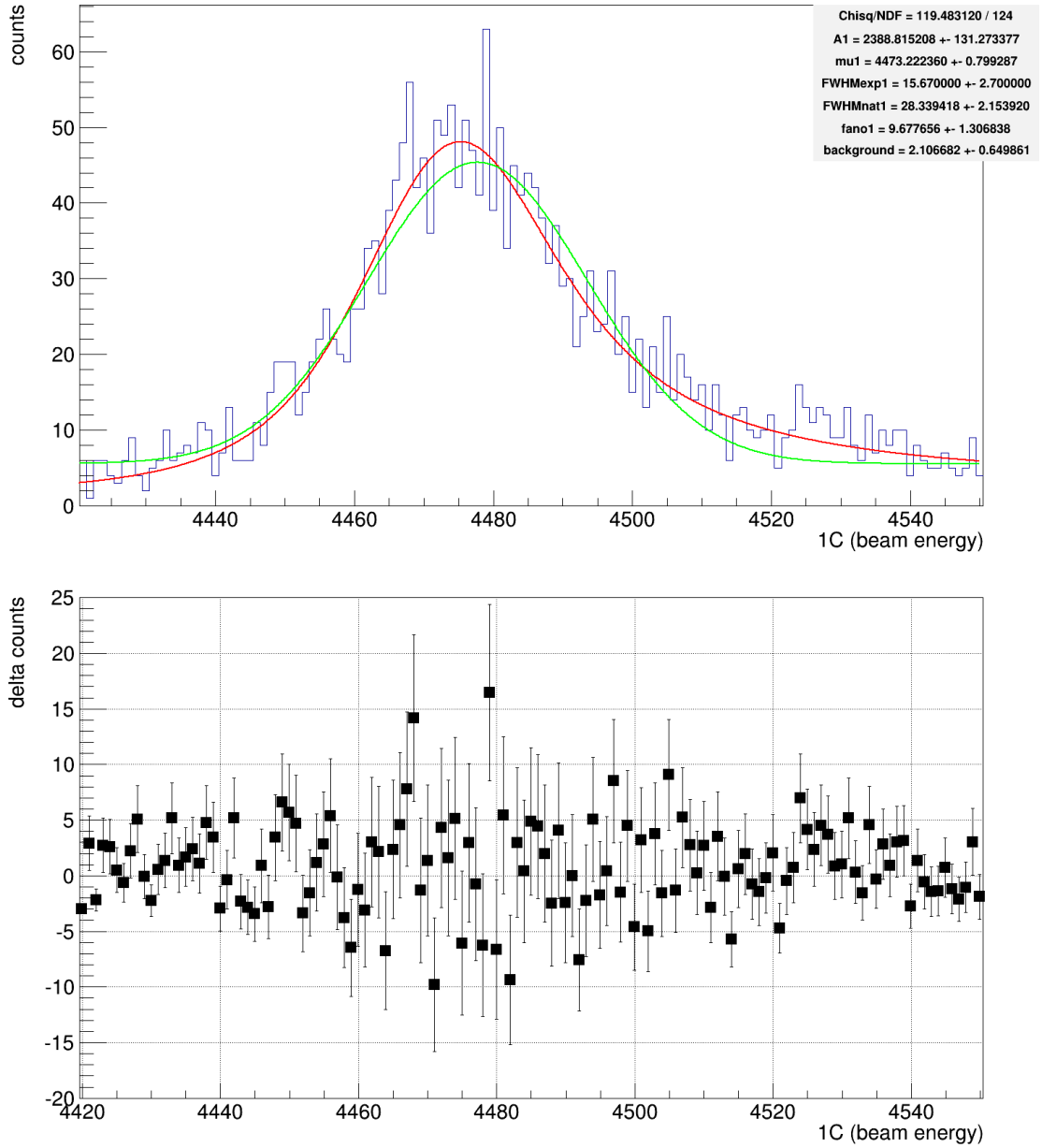

Abbildung 5.16: Oberes Diagramm: Vergleich Fano-Gauß-Faltungs-Fit (rot) von DR-Resonanz Nr. 14, projiziert auf die Elektronenstrahlachse, mit dazugehörigen Residuen (unteres Diagramm) zu Gaußfit (grün) mit Fitergebnissen (rechts oben) zum Fano-Gauß-Faltungs-Fit: Chiqs/NDF ist das Chi-Quadrat / Anzahl der Freiheitsgrade, A1 die totale Peakfläche, mu1 der Peakschwerpunkt, FWHM(exp,nat)1 die volle Halbwärtsbreite (exp: Apparatebreite, nat: natürliche Linienbreite), background der konstante Hintergrund und fano1 der Fano-Parameter.

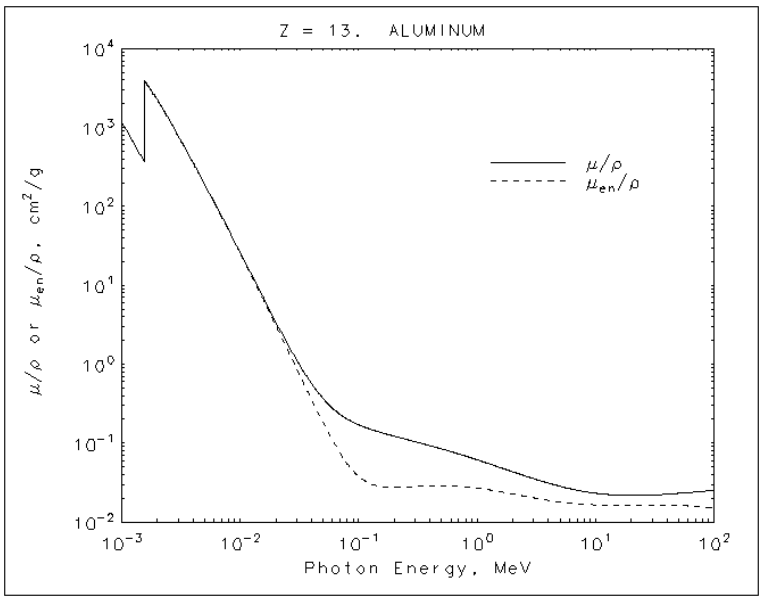

Abbildung 5.17: Absorbtionskurve für Aluminium [NIST] in doppellogarithmischer Darstellung. Dabei ist die Photonenenergie gegen den Absorbtionskoeffitienten/Dichte aufgetragen.

5.5.1 Qualitative Untersuchung

Zur qualitativen Untersuchung werden sieben ROIs gewählt, die über die ganze Elektronenstrahlenergie reichen und jeweils 1 keV entlang der Photonenenergie breit sind, wie es in Abbildung 5.18 zu sehen ist. Die ROIs werden aus dem kalibrierten Spektrum herausgeschnitten. Die Projektionen sind in Abbildung 5.19 dargestellt.

Es wurden 35 Resonanzen gefunden. Einige konnten zu theoretischen zugeordnet werden. Die Abbildungen 5.20, 5.21 und 5.22 zeigen jeweils die Nummerierung für die Bereiche $KL_{1/2}L_{1/2}$, $KL_{1/2}L_3$ und KL_3L_3.

5.5.2 Vermessung der Resonanzen

Die gefundenen Resonanzen sollten auch Vermessen werden. Wie oben beschrieben wird dazu ein ROI herausgeschnitten und auf die Achsen projiziert. Es wird dabei mit dem unkalibrierten Messergebnis gerbeitet. Bei der Photonenenergie wird ein Rechteck herausgeschnitten, bei der Elektronenstrahlenergie ein Parallelogramm, um die selbe Steigung wie die RR Linien zu haben. Abbildung 5.23 zeigt ein ROI mit gefitteten Projektionen und zur verbesserten auch Anschauung die gebinnte (alle 8 Bins werden zu einem zusammengefasst) Version. Die Fits werden an den ungebinnten Histogrammen vollzogen.

Die Peakschwerpunkte sowie die Halbwärtsbreiten werden noch mit den Kalibrationsfunktionen umgerechnet und mit den Theorien verglichen. Die Tabellen 5.9, 5.11 und 5.13 mit 5.15 listen die Anregungsenergien der $KL_{1/2}L_{1/2}$, $KL_{1/2}L_3$ und

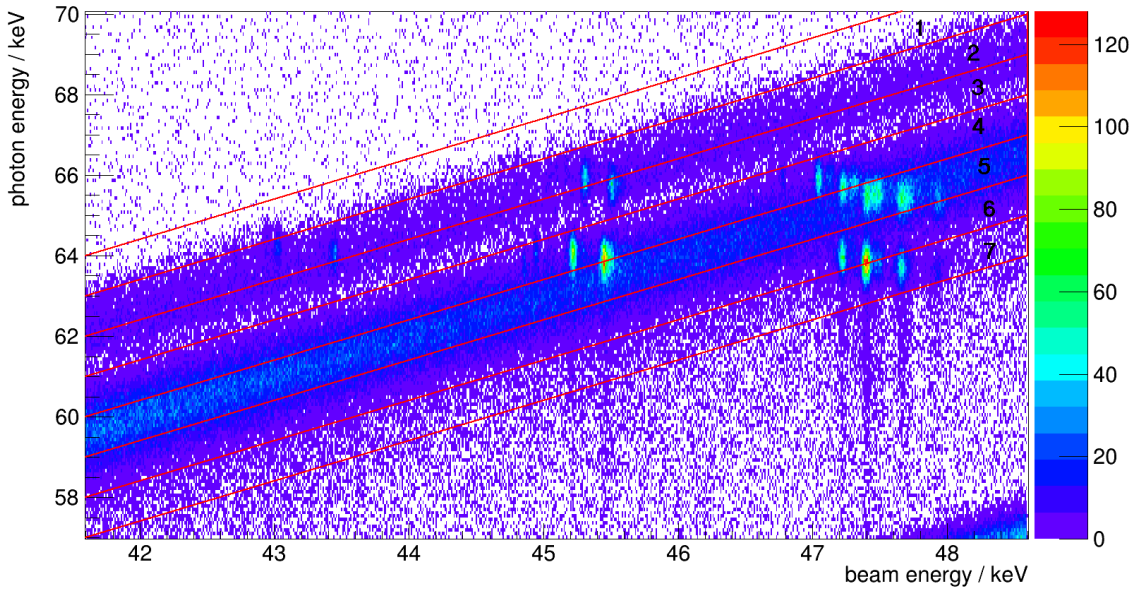

Abbildung 5.18: Kalibriertes Histogramm, Elektronenstrahlenergie ist gegen Photonenenergie gebinnt (Faktor 8) aufgetragen, mit ROIs (rot umrahmt) von DR-Messung Nr. 9

KL_3L_3 Bereiche auf. Entsprechend enthalten die Tabellen 5.10, 5.12 und 5.14 mit 5.16 die Photonenenergien. Abbildung 5.24 zeigt den Vergleich mit der Theorie.

Alle theoretischen Werte sind kleiner als erwartet. Das könnte daran liegen dass ein systematischer Einfluss unbeachtet blieb. Absolutmessungen, wie hier die Elektronenstrahlenergie, sind für Fehler dieser Art anfällig [Crespo14]. Die He-artigen Resonanzen liegen ca. 20 eV unterhalb. Da He-artige Resonanzen in der Regel genaue theoretische Vorhersagen wiedergeben, könnte man sagen, dass eine systematische Fehlerquelle die Elektronenstrahlenergie um +20 eV verschoben hat.

Nr.	**Beginn**	**Ende**	**t (s)**
D1	24.06.2014 14:50	24.06.2014 15:24	2008
D2	24.06.2014 15:40	24.06.2014 16:06	1552
D3	24.06.2014 16:06	24.06.2014 17:02	3341
D4	02.07.2014 16:28	02.07.2014 17:45	1040
D5	08.07.2014 14:14	08.07.2014 14:45	1800

Tabelle 5.4: Parameter Photonenenergie-Kalibration. Nr.: Nummer der Photonenenergie-Kalibration, Beginn/Ende: Datum und Uhrzeit vom Start/Stop der Messung, t: Messzeit

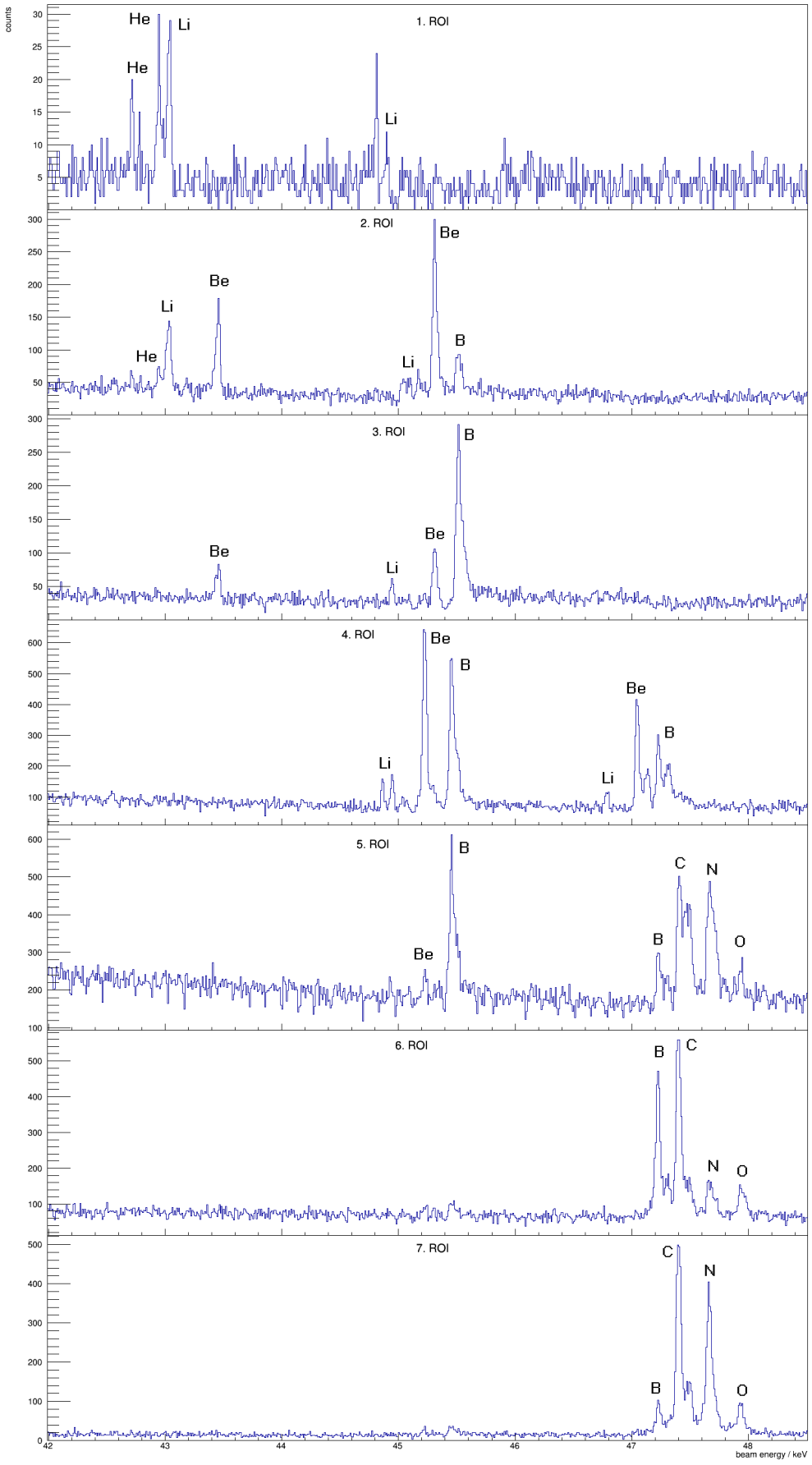

Abbildung 5.19: Projektionen (gebinnt, Faktor 8) der ROIs von DR-Messung Nr. 9 auf die Elektronenstrahlachse. Einige Peaks sind mit entsprechenden Ladungszuständen markiert (Element-artig).

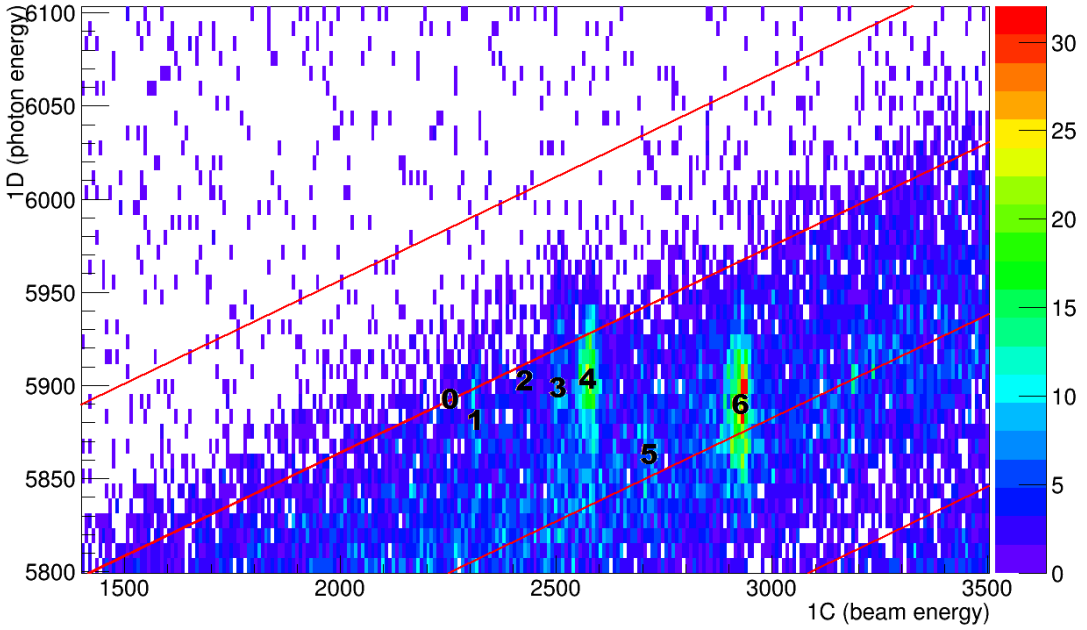

Abbildung 5.20: DR Nummerierung der Resonanzen für$KL_{1/2}L_{1/2}$ Bereich von Messung Nr. 9 (gebinnt, Faktor 8) mit ROI Rahmen (rot)

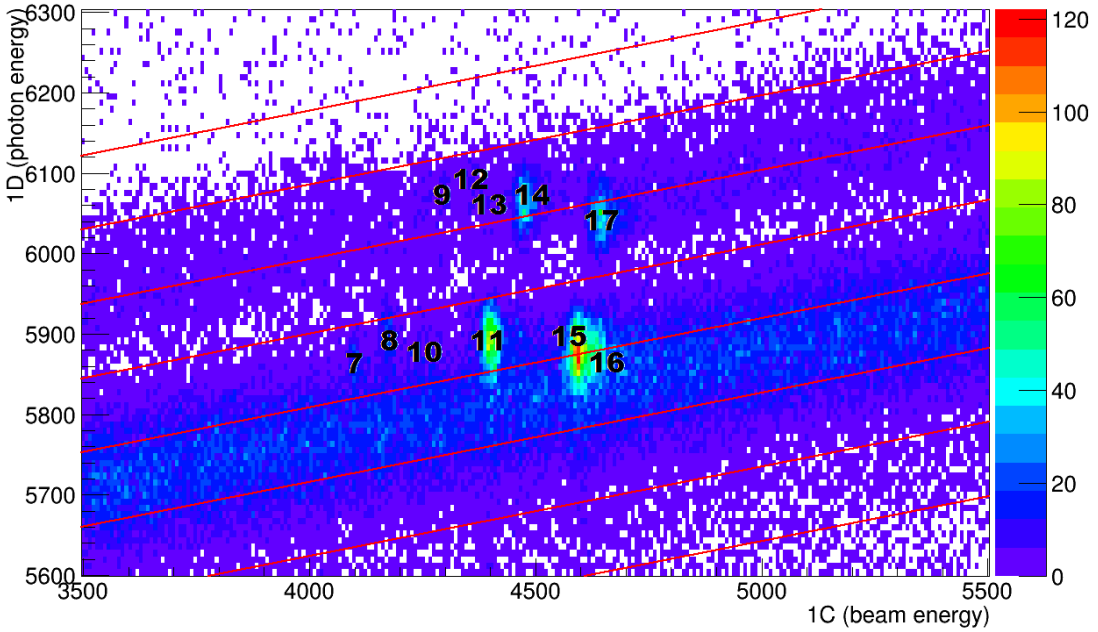

Abbildung 5.21: DR Nummerierung der Resonanzen für $KL_{1/2}L_3$ Bereich von Messung Nr. 9 (gebinnt, Faktor 8) mit ROI Rahmen (rot)

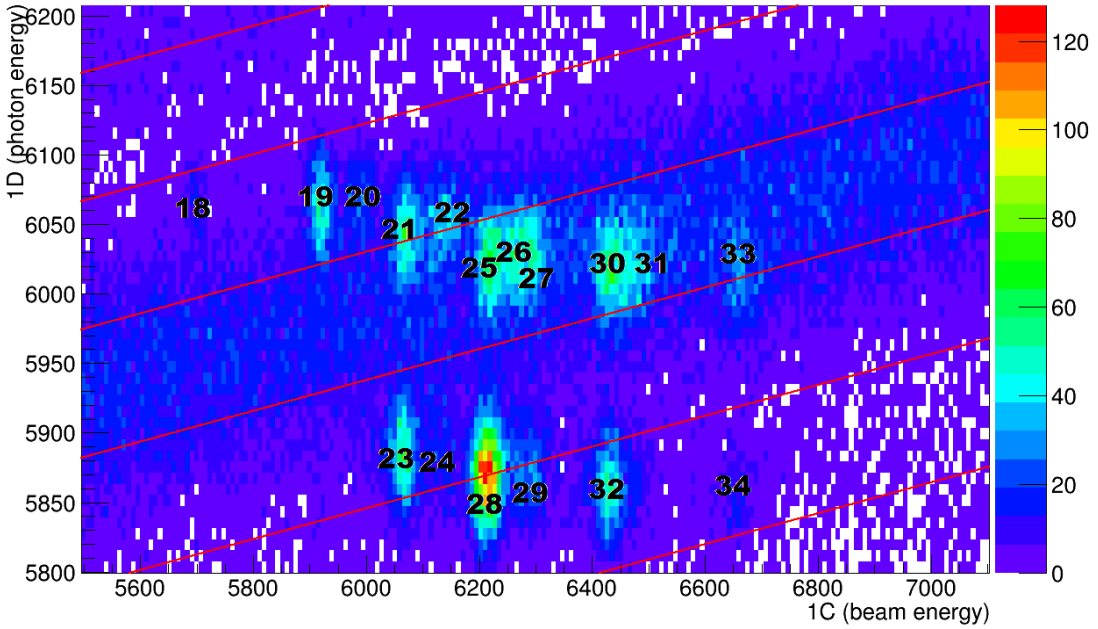

Abbildung 5.22: DR Nummerierung der Resonanzen für KL_3L_3 Bereich von Messung Nr. 9 (gebinnt, Faktor 8) mit ROI Rahmen (rot)

Nr.	**Funktion (kV)**
D1,2,3	E(D) = 0.0108470(2) × D + 0.1713(7)
D4,5	E(D) = 0.0108427(2) × D + 0.175(1)

Tabelle 5.5: Photonenenergie-Kalibration Funktionen. Nr.: Nummer der Photonenenergie-Kalibration, Funktion: Formel für die Umrechnung von der Kanalnummer zum Energiewert.

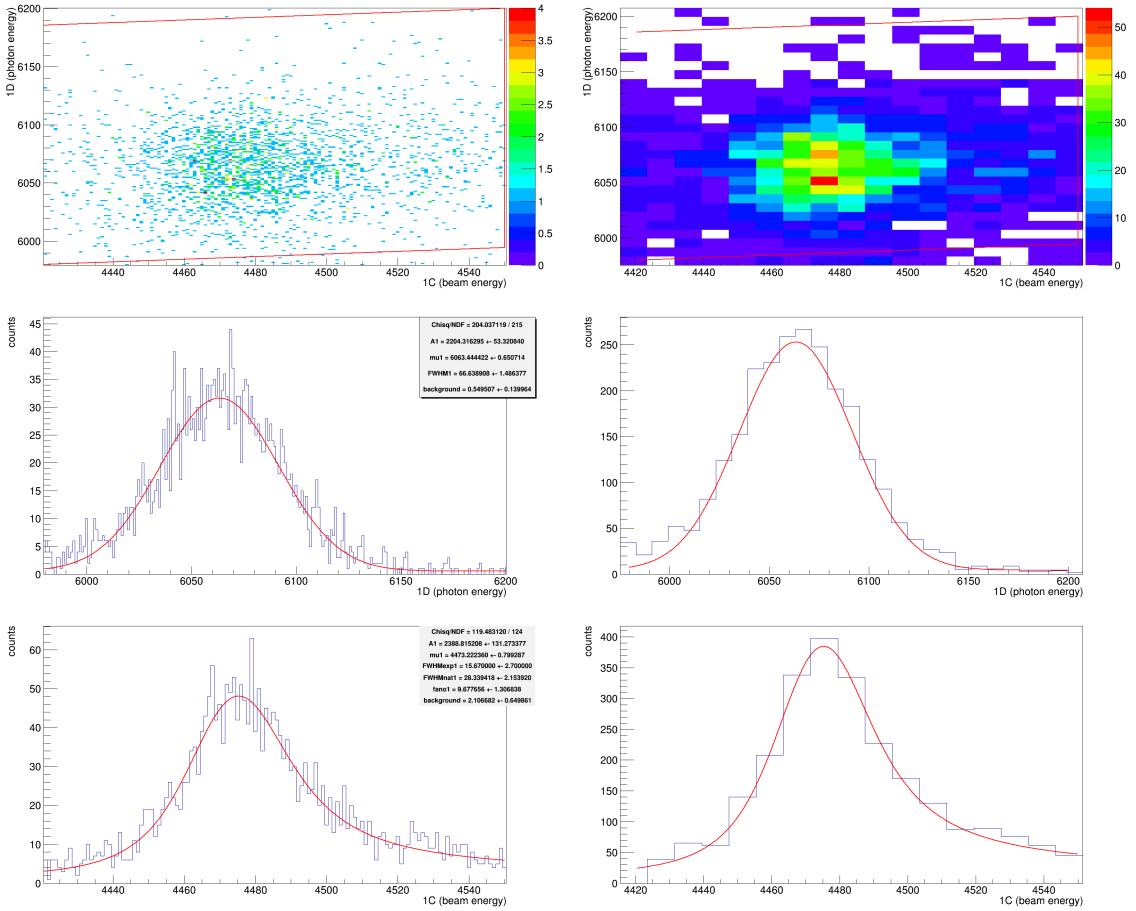

Abbildung 5.23: ROI, Projektionen und Fit einer DR Resonanz, ungebinnt (links) und gebinnt (Faktor 8, rechts) von Resonanz Nr. 14 aus Messung Nr. 9. In den obigen Diagrammen ist Elektronenstrahlenergie gegen Photonenenergie mit den Ausschnittsgrenzen (roter Rahmen) aufgetragen, in den mittleren die Projektion der kompletten Ausschnitte gegen die Photonenenergie (blau) mit Gaußfit (rot) und in den unteren die Projektion der rot umrahmten Ausschnitts gegen die Elektronenstrahlenergie mit Fitfunktion (Faltung aus Fano- und Gaußprofil plus Konstante). Der Fit wurde immer am ungebinnten Diagramm vollzogen, wo auch die Fitergebnisse zu sehen sind (rechtes oberes Kästchen). Chiqs/NDF ist das Chi-Quadrat / Anzahl der Freiheitsgrade, A1 die totale Peakfläche, mu1 der Peakschwerpunkt, FWHM(exp,nat)1 die volle Halbwärtsbreite (exp: Apparatebreite, nat: natürliche Linienbreite), background der konstante Hintergrund und fano1 der Fano-Parameter.

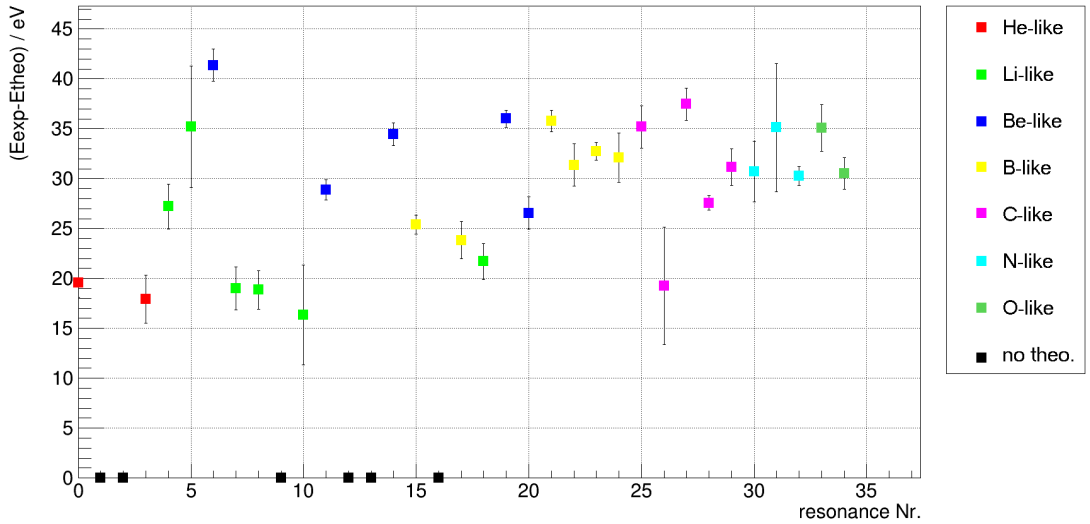

Abbildung 5.24: Differenz der experimentellen zu den theoretischen Resonanzen der Elektronenstrahlenergie mit Fehlerbalken. Die zugeordneten Resonanzen sind farbig entsprechend den Ladungszuständen (rot, grün, blau, gelb, rosa, hellblau, dunkelgrün entsprechen He-, Li-, Be, B-, C-, N-, O-artigen Resonanzen, schwarz: keine Zuordnung) markiert. Zu sehen ist dass alle theoretischen Werte kleiner als erwartet sind.

Nr.	**Beginn**	**Ende**	**t (s)**	$\boldsymbol{U_{eb}}$ **(kV)**	$\boldsymbol{U_{ee}}$ **(kV)**	$\boldsymbol{\delta U}$ **(kV)**	**dt (s)**	$\boldsymbol{U_c}$ **(kV)**	**SMIN (kV)**	**SMAX (kV)**
C1	03.07.2014 09:23	03.07.2014 9:28	300	39	41	1	60	1.5	47	51
C2	05.07.2014 14:32	05.07.2014 14:43	660	40.5	33	0.5	120, 120, 30, ...	1.5	41	49.5
C3	05.07.2014 15:45	05.07.2014 15:49	210	42	39	0.5	30	1.5	47	51
C4	05.07.2014 15:55	05.07.2014 15:59	210	39	42	0.5	30	1.5	47.5	51
C5	07.07.2014 18:10	07.07.2014 18:14	240	38	41.5	0.5	30	1	46.1	50
C6	08.07.2014 13:22	08.07.2014 13:26	240	38	41.5	0.5	30	1	46.1	50
C7	08.07.2014 13:31	08.07.2014 13:38	450	40	33	0.5	30	1.5	41	49
C8	08.07.2014 13:46	08.07.2014 13:50	240	38	41.5	0.5	30	1.5	46.5	50.5
C9	08.07.2014 13:58	08.07.2014 14:01	210	39	42	0.5	30	1.5	47	51

Tabelle 5.6: Parameter Beschleunigungsspannung-Kalibration. Nr.: Nummer der Beschleunigungsspannung-Kalibration, Beginn/Ende: Datum und Uhrzeit vom Start/Stop der Messung,t: Messzeit, $U_{eb,ee}$: minimale/maximale Spannung der Elektronenkanonenplattform, δU: Abstände der Kalibationsspannungen, U_c: Kathodenspannung, SMIN/SMAX: minimale/-Maximale Beschleunigungsspannung für Datenaufnahmesystem

Nr.	Funktion (kV)
C1	E(C) = 6.00895(16)E-04 × C + 46.50980(8)
C2	E(C) = 1.27688(2)E-03 × C + 39.95800(10)
C3	E(C) = 6.0089(2)E-04 × C + 46.50970(12)
C4	E(C) = 5.25792(15)E-04 × C + 47.07090(4)
C5	E(C) = 5.8581(2)E-04 × C + 45.62210(10)
C6	E(C) = 5.8586(2)E-04 × C + 45.62210(12)
C7	E(C) = 1.20167(5)E-03 × C + 40.0198(2)
C8	E(C) = 6.0093(3)E-04 × C + 46.00950(15)
C9	E(C) = 6.0086(5)E-04 × C + 46.5103(2)

Tabelle 5.7: Elektronenstrahl-Kalibration Funktionen (ohne Raumladungskorrektur). Nr.: Nummer der Elektronenstrahl-Kalibration, Funktion: Formel für die Umrechnung von der Kanalnummer zum Energiewert. Energiewert entspricht Beschleunigungsspannung mal Elementarladung.

Nr.	ChSt	I_p selbst gemessen (kV)	I_p von [NIST] (kV)	I_p von [Scofield03] (kV)
1	He	21.432(3)(58)	21.55660(24)	21.5568
	Li	20.997(3)(98)	21.200(90)	21.2121
	Be	20.632(3)(27)	20.650(70)	20.6417
	B	20.224(3)(12)	(20.210(60)	20.2474
2	C	18.107(1)(20)	18.120(60)	18.1092
	N	17.774(1)(19)	17.770(50)	17.761
	O	(17.437(1)(33)	17.390(50)	17.4142
	F	17.001(1)(115)	17.040(40)	17.0474
3	C	18.112(1)	18.120(60)	18.1092
	N	17.770(1)	17.770(50)	17.7612
	O	17.409(1)	17.390(50)	17.414)

Tabelle 5.8: Ionisationspotentiale der Kantenmessungen. Nr.: Nummer der Kantenmessung: 1: mit Wolframfolie, 2 und 3 mit Tantalfolie, ChSt: Ladungszustand des Ions vor der Rekombination: Element-artig, Ip: Ionisationspotential: 3. Spalte: von dieser Arbeit mit runde Klammern: (statistischer) oder (systematischer)(statistischer) Fehler, die systematische Verschiebung gegenüber drn theoretischen He-artigen Resonanzen wurde dabei nicht mit einbezogen, 4. Spalte: Werte von [NIST], Fehler in Runden Klammern, 5. Spalte: Werte von [Scofield03]

Peak-Nr.	**theo / exp**	**ChSt J P**	**μ (keV)**	**Γ (eV)**	**Q**
0	theo	He 1/2 +	42.6901	0.991	
	exp		42.7097(2)(1)	2(6)	
1	exp		42.783(2)	42(8)	
2	exp		42.849(6)	4(16)	
3	theo	He 1/2 -	42.9255	6.19	
	exp		42.943(2)	13(10)	
4	theo	Li 1/2 -	43.0033	9.65	
	exp		43.031(2)	32(4)	-10(4)
5	theo	Li 1 +	43.1570	13.7	
	exp		43.192(6)	65(40)	
6	theo	Be 1/2 +	43.4148	13.7	
	exp		43.456(10)(8)	26(10)(8)	-6(1)

Tabelle 5.9: Anregungsenergie der DRs vom $KL_{1/2}L_{1/2}$ Bereich. Peak-Nr.: Bezeichnungs-Nummer der Resonanz in dieser Arbeit, 2. Spalte: theo: theoretische Elektronenstrahlenergie, exp: gemessene Elektronenstrahlenergie, 3. Spalte:ChSt J P: Ladungszustand (Element-artig) mit Drehimpuls J und Parität P, μ: Peakschwerpunkt, Γ: natürliche Halbwärtsbreite, Q: Fanoparameter, Fehler in runden Klammern: (systematischer)(statistischer) oder nur (statistischer)

Peak-Nr.	**μ (keV)**	**FWHM (eV)**
0	63.90(8)(3)	580(80)(90)
1	63.4(4)	1600(500)
2	63.40(8)(5)	900(80)(200)
3	64.19(8)(4)	610(90)(60)
4	64.19(8)(4)	610(90)(60)
5	63.67(8)(2)	1130(80)(90)
6	63.980(80)(9)	870(80)(30)

Tabelle 5.10: Photonenenergie der DRs vom $KL_{1/2}L_{1/2}$ Bereich. Peak-Nr.: Bezeichnungs-Nummer der Resonanz in dieser Arbeit, μ: gemessener Peakschwerpunkt, FWHM: gemessene volle Halbwärtsbreite, Fehler in runden Klammern: (systematischer)(statistischer) oder nur (statistischer)

Peak-Nr.	**theo / exp**	**ChSt J P**	**μ (keV)**	**Γ (eV)**	**Q**
7	theo	Li 2 -	44.8391	0.506	
	theo	He 1/2 +	44.8335	25.8	
	exp		44.858(2)	7(4)	16(19)
8	theo	Li 3 +	44.9274	9.32	
	exp		44.946(2)	13(3)	-22(23)
9	exp		45.057(4)	131(12)	
10	theo	Li 2 +	45.0226	17.7	
	exp		45.039(5)	115(30)	
11	theo	Be 5/2 +	45.1905	9.45	
	theo	Be 3/2 +	45.1919	8.78	
	exp		45.220(9)(6)	23(2)	25(10)
12	exp		43.0299(109)(80)	32.3(10.9)(8.7)	
	exp		45.052(4)	96(11)	
13	exp		43.0299(109)(80)	32.3(10.9)(8.7)	
	exp		45.198(4)	95(13)	
14	theo	Be 3/2 +	45.2724	16.1	
	exp		45.3069(1)(11)	34(3)	9.7(1.3)
15	theo	B 2 -	45.4240	13.9	
	exp		45.449(3)	28(2)	7.1(8)
16	exp		45.505(2)	12(8)	
17	theo	B 1 -	45.4793	29.4	
	exp		45.503(2)	46(4)	4.4(5)

Tabelle 5.11: Anregungsenergie der DRs vom $KL_{1/2}L_3$ Bereich. Peak-Nr.: Bezeichnungs-Nummer der Resonanz in dieser Arbeit, 2. Spalte: theo: theoretische Elektronenstrahlenergie, exp: gemessene Elektronenstrahlenergie, 3. Spalte:ChSt J P: Ladungszustand (Element-artig) mit Drehimpuls J und Parität P, μ: Peakschwerpunkt, Γ: natürliche Halbwärtsbreite, Q: Fanoparameter, Fehler in runden Klammern: (systematischer)(statistischer) oder nur (statistischer)

Peak-Nr.	**μ (keV)**	**FWHM (eV)**
7	63.73(8)(8)	550(80)(90)
8	64.00(8)(11)	670(80)(140)
9	66.06(9)(2)	460(90)(60)
10	64.32(9)(11)	340(90)(150)
11	64.10(8)(1)	630(90)(30)
12	66.02(9)(3)	680(90)(80)
13	66.02(9)(3)	680(90)(80)
14	65.904(90)(7)	722(90)(16)
15	63.92(8)(1)	720(80)(20)
16	63.92(8)(1)	720(80)(20)
17	65.750(90)(7)	770(90)(20)

Tabelle 5.12: Photonenenergie der DRs vom $KL_{1/2}L_3$ Bereich. Peak-Nr.: Bezeichnungs-Nummer der Resonanz in dieser Arbeit, μ: gemessener Peakschwerpunkt, FWHM: gemessene volle Halbwärtsbreite, Fehler in runden Klammern: (systematischer)(statistischer)

5.6 Diskussion der Ergebnisse

Die Messergebnisse zeigen wie erwartet charakteristische Muster bezüglich den verschiedenen Rekombinationsprozessen: Streifenmuster für die radiative und Punktmuster für die dielektronische Rekombination. Ähnlich wie bei anderen Arbeiten mit schweren, hochgeladenen Ionen machten sich bei zwei RR-Bänder und drei KLL-Bereiche (vergl. [Martínez, 2005], [Mäckel, 2006]) im vermessenen Bereich bemerkbar (siehe Abb. 2.1).

Die Fehler liegen für die Photonenergie bei ca. 85 eV und die der Elektronenstrahlenergie bei wenigen keV. Der große Unterschied stammt von den verschiedenen Messmethoden der jeweiligen Energie und ihre Präzision. Die elektrische Spannung konnte mit höheren relativen Genauigkeiten gemessen werden als die Röntgenphotonen. Die Photonenenergieauflösung konnte durch den Einsatz von Metallfolien und ihren Absorptionseigenschaften für Röntgenstrahlen derart gestiegen werden, dass einzelne RR-Linien eines RR-Bandes vermessen werden konnten.

Die errechneten Ionisationspotentiale von der Kantenmessung zeigen stark unterschiedliche statistische Fehler. Der Grund dafür liegt zu einen in der Statistik, d.h. wie lange bzw. wie viele Ereignisse gemessen wurden. Das wird vor alle bei Kantenmessung Nr. 3 und 2 bemerkbar. Kantenmessung Nr. 3 hat eine längere Messzeit als Nr. 2, somit auch einen gerineren statistischen Fehler. Aber auch die Deutlichkeit der Abstufungen der RR-Linien an der Absorptionskante beeinflusst den Fehler maßgebend.

Bei den Anregungsenergien der dielektronischen Resonanzen ist eine Verschiebung bezüglich der theoretischen Werte erkennbar. Die gemessenen Anregungsenergien

Peak-Nr.	**theo / exp**	**ChSt J P**	**μ (keV)**	**Γ (eV)**	**Q**
18	theo theo exp	Li 3 + He 1/2 +	46.7571 46.7656 46.781(8)	4.43 12.1 20(4)	
19	theo exp	Be 5/2 +	47.0103 47.0463(2)(8)	4.39 26(2)	
20	theo exp	Be 1/2 +	47.1052 47.132(2)	12.0 40(5)	
21	theo exp	B 3 -	47.1912 47.227(1)	13.3 29(5)	
22	theo ex	B 1 -	47.2758 47.307(2)	20.9 65(10)	
23	theo exp	B 3 -	47.1912 47.2240(2)(9)	13.3 28(4)	
24	theo exp	B 1 -	47.2758 47.307(2)	20.9 37(13)	
25	theo exp	C 5/2 +	47.3734 47.409(2)	17.6 43(10)	
26	theo exp	C 3/2 +	47.4408 47.460(6)	36.2 44(14)	 4(3)
27	theo exp	C 1/2 +	47.4660 47.503(2)	25.0 7(8)	
28	theo exp	C 5/2 +	47.3734 47.4010(2)(7)	17.6 38(1)	
29	theo exp	C 1/2 +	47.4660 47.497(2)	25.0 68(6)	

Tabelle 5.13: Anregungsenergie der DRs vom KL_3L_3 Bereich. Peak-Nr.: Bezeichnungs-Nummer der Resonanz in dieser Arbeit, 2. Spalte: theo: theoretische Elektronenstrahlenergie, exp: gemessene Elektronenstrahlenergie, 3. Spalte:ChSt J P: Ladungszustand (Element-artig) mit Drehimpuls J und Parität P, μ: Peakschwerpunkt, Γ: natürliche Halbwärtsbreite, Q: Fanoparameter, Fehler in runden Klammern: (systematischer)(statistischer) oder nur (statistischer)

Peak-Nr.	**μ (keV)**	**FWHM (eV)**
18	66.08(9)(3)	250(90)(80)
19	65.95(9)(3)	650(90)(50)
20	65.95(9)(3)	650(90)(50)
21	65.76(9)(1)	690(90)(40)
22	65.76(9)(1)	690(90)(40)
23	63.97(8)(2)	760(80)(40)
24	63.97(8)(2)	760(80)(40)
25	65.494(90)(5)	850(90)(20)
26	65.494(90)(5)	850(90)(20)
27	65.494(90)(5)	850(90)(20)
28	63.838(80)(4)	647(80)(15)
29	63.838(80)(4)	647(80)(15)

Tabelle 5.14: Photonenenergie der DRs vom KL_3L_3 Bereich. Peak-Nr.: Bezeichnungs-Nummer der Resonanz in dieser Arbeit, μ: gemessener Peakschwerpunkt, FWHM: gemessene volle Halbwärtsbreite, Fehler in runden Klammern: (systematischer)(statistischer)

Peak-Nr.	**theo / exp**	**ChSt J P**	**μ (keV)**	**Γ (eV)**
30	theo exp	N 2 -	47.6325 47.663(3)	25.0 58(12)
31	theo exp	N 1 -	47.6853 47.720(6)	39.4 64(13)
32	theo exp	N 2 -	47.6325 47.6628(2)(9)	25.0 41(3)
33	theo exp	O 1/2 +	47.9019 47.937(2)	35.5 37(12)
34	theo exp	O 1/2 +	47.9019 47.932(2)	35.5 58(13)

Tabelle 5.15: Anregungsenergie der DRs vom KL_3L_3 Bereich. Peak-Nr.: Bezeichnungs-Nummer der Resonanz in dieser Arbeit, 2. Spalte: theo: theoretische Elektronenstrahlenergie, exp: gemessene Elektronenstrahlenergie, 3. Spalte:ChSt J P: Ladungszustand (Element-artig) mit Drehimpuls J und Parität P, μ: Peakschwerpunkt, Γ: natürliche Halbwärtsbreite, Fehler in runden Klammern: (systematischer)(statistischer) oder nur (statistischer)

Peak-Nr.	**μ (keV)**	**FWHM (eV)**
30	65.491(90)(6)	870(90)(25)
31	65.491(90)(6)	870(90)(25)
32	63.735(80)(8)	585(80)(20)
33	65.39(9)(5)	770(90)(75)
34	63.65(8)(1)	545(80)(40)

Tabelle 5.16: Photonenenergie der DRs vom KL_3L_3 Bereich. Peak-Nr.: Bezeichnungs-Nummer der Resonanz in dieser Arbeit, μ: gemessener Peakschwerpunkt, FWHM: gemessene volle Halbwärtsbreite, Fehler in runden Klammern: (systematischer)(statistischer)

He-artiger Resonanzen liegt ca. 20 eV höher als theoretisch erwartet, die bei den anderen Zuständen ca. 30 eV. Je weniger Elektronen, desto zuverlässiger ist das theoretische Ergebnis. Die 20 eV Verschiebung zu den He-artigen Resonanzenergien könnte von unbeachteten systematischen Fehlerquellen herrühren, da Absolutmessungen, wie hier bei der Elektronenstrahlenergie, für solche Fehler anfällig sind. Die Messung Photonenenergie wurde mittels dem charakteristischen Spektrum des radioaktiven Elements ^{241}Am kalibriert und zählt daher nicht zu den Absolutmessungen. Eine systematische Fehlerquelle würde lediglich die Kalibrationsfunktion verzerren. Aber selbst wenn man die Verschiebung der Energieachse bezüglich der He-artigen Resonanzen berücksichtigt, bleiben für die übrigen Resonanzen immer noch ca. 10 eV Verschiebung übrig. Dies ist bei schweren, hochgeladenen Ionen kein Einzelfall (siehe [Mäckel, 2006]). Die theoretischen Berechnungen solcher Energien gestaltet sich durch QED-Beiträge und Kerngrößeneffekte als schwierig. Präzisere Messungen könnten zum hierbei zum besseren Verständnis führen.

Die Ergebnisse dieser Arbeit zeigen, Abweichungen von Theorie und Experiment bei Rekombinationsenergien von schweren, hochgeladenen Ionen nichts Außergewöhnliches ist. Theoretische Werte für Ionen mit wenigen Elektronen gelten als verlässlich und könnten sogar zur Kalibration genutzt werden. Für Ionen mit vielen Elektronen können die experimentellen Werte genutzt werden um Theorien zu überarbeiten und zu testen.

6 Zusammenfassung / Ausblick

In dieser Arbeit wurden Ionisationspotentiale und dielektronische Resonanzen von hochgeladenen Iridiumionen vermessen. Die Iridiumionen wurden in der HD-EBIT erzeugt und gefangen. Die aus den Rekombinationsprozessen stammenden Photonen wurden detektiert und ihre Energie zusammen mit der Elektronenstrahlenergie und Strom vermessen. Charakteristische Merkmale spiegelten sich in den Messergebnissen wieder, die Aufschluss über die gesuchten Energien gaben.

Im Allgemeinen konnten eine gute Übereinstimmung mit aktuellen Strukturberechnungen festgestellt werden. Künftige Experimente könnten mithilfe der Effekte von Absorptionskanten noch genauere Resultate erzielen, wenn entsprechend längere und systematischere Messkampagnen durchgeführt würden, was im Rahmen der vorliegenden Arbeit zeitlich nicht möglich war. Stärkere, genauestens vermessene DR-Resonanzen, würden dann für weitere Messungen schwächerer Resonanzen als Kalibration dienen, was die Messungen nicht nur vereinfachen sondern auch systematische Fehlerquellen vorbeugen würde.

In Zukunft könnte man auch die Automatisierung von Experimenten weiter ausbauen und für Nutzer leichter und intuitiver gestalten. Dadurch ließe sich nicht nur leichter mehr Statistik sammeln, sondern auch vermehrt abstrakte und komplexe Aufgaben dem Kontrollsystem überlassen, wodurch man Zeit für andere wissenschaftliche Aufgaben einsparen könnte. Zudem lassen sich mit der Software auch gefahrlos Tests und Simulationen durchführen, bevor dem Experimenten Neuerungen hinzugefügt werden. Ein weiterer wichtiger Punkt ist die Flexibilität, da Laborumbauten an der Tagesordnung sind. Die Labboxserver müssen nur entsprechend ihrer Anschlüsse konfiguriert werden, die Deviceserver entsprechend den Verkabelungen der Laborboxen. Die Server oder Clienten, die ausschließlich auf die Deviceserver zugreifen (i.d.R. Taskserver oder GUIs) müssen in der Regel nicht verändert werden. Alle diese Vorteile sprechen für eine vermehrte Verwendung der in dieser Arbeit entwickelten, auf EPICS basierenden Kontrollsoftware in den EBIT-Experimenten.

Teil I

Anhang

A sonstiges Material

Bauteil	**Netzteil**
Versorgung Helmholtz-Spulen	LakeShore Superconducting Magnet Power Supply Model 622
Temperaturmesser Helmholtz-Spulen	Cyromagnetics, Inc., Model TM-600
Helium-Füllmesser	AMI Liquid Helium Level Monitor Model 135
Drucksonden	Varian Multi-Gauge
Turbopumpen	Varian TV 3000 (oder 70,550,301) contoller

Tabelle A.1: Sonstige verwendete Geräte der HD-EBIT

Linie	**Energie / keV**	**Fehler / keV**
$^{241}Am_{\gamma}$	59.5412	0.0002
$^{241}Am_{\gamma}$	26.36	0.005
NpL_{γ}	20.77	0.005
$NpL_{\beta 1}$	17.7502	0.00005
$NpL_{\beta 2}$	16.84	0.005
$NpL_{\alpha 1}$	13.9441	0.00005
$PtL_{\beta 1}$	11.8	0.005
$PtL_{\beta 2,3}$	11.2505	0.00005
$PtL_{\alpha 1}$	9.4423	0.00005

Tabelle A.2: Kalibrationslinien der Am241-Quelle (siehe z.B. [Nuc14])

Jahr	**Arbeit**
2005	F. A. J. González Martínez, Quantum interference in the dielectronic recombination of heavy highly charged ions, Dissertation, Ruperto-Carola University of Heidelberg, Germany
2005	R. S. Orts, Isotopic effect in B-like and Be-like argon ions, Dissertation, Johann Wolfgang Goethe - Universität in Frankfurt am Main, Deutschland
2006	V. Mäckel, Spektroskopische Untersuchung dielektronischer Resonanzen von hochgeladenen Ionen, Diplomarbeit, Ruprecht-Karls-Universität Heidelberg, Deutschland
2007	G. Brenner, Quantenelektrodynamische Einflüsse auf die Lebensdauer metastabiler Zustände, Dissertation, Ruprecht-Karls-Universität Heidelberg, Deutschland

Tabelle A.3: Abschlussarbeiten mittels der HD-EBIT bis 2007

Jahr	**Arbeit**
2008	A. Krauß, Vollständig differentielle Experimente zum Elektroneneinfang im langsamen Ion-Atom Stoß, Diplomarbeit, Ruprecht-Karls-Universität Heidelberg, Deutschland
2008	C. Beilmann, Beobachtung resonanter Elektroneneinfangprozesse höherer Ordnung in hochgeladenem Krypton, Diplomarbeit, Ruprecht-Karls-Universität Heidelberg, Deutschland
2008	M. Binder, Spektroskopie an hoch geladenen Ionen aus der EBIT im Sichtbaren und im nahen Ultraviolett, Diplomarbeit, Max-Planck-Insitut für Kernphysik, Heidelberg, Deutschland
2009	R. Klawitter, Resonant laser spectroscopy of a visible magnetic dipole transition in Ar13+, Diplomarbeit, Ruprecht-Karls-Universität Heidelberg, Deutschland
2009	S. Higgins, Implementation of a Deceleration Beamline for the Investigation of Charge Exchange Processes using Highly Charged Ions, MSci Thesis, Blackett Laboratory, Imperial College London, United Kingdom
2009	S. Bernitt, Optimierung der Ladungszustandsverteilung in einer EBIT durch resonante Photorekombination, Diplomarbeit, Ruprecht-Karls-Universität Heidelberg, Deutschland
2010	V. Mäckel, Laserspektroskopie hochgeladener Ionen an der Heidelberger Elektronenstrahl-Ionenfalle, Dissertation, Ruprecht-Karls-Universität Heidelberg, Deutschland
2010	P. Mrowczynski, Konstruktion und Inbetriebnahme eines Flugzeit-Spektrometers für Ionen in Ladungsaustausch-Reaktionen an der Heidelberg-EBIT, Diplomarbeit, Ruprecht-Karls-Universität Heidelberg, Deutschland
2010	J. Link, Hochgenaue Laborbestimmung der Wellenlänge des verbotenen Übergangs der FeXIV-Linie, Bachelorarbeit, Ruprecht-Karls-Universität Heidelberg, Deutschland
2010	K. Kubicek, Absolut- und Relativbestimmungen der Energien von 2p-1s-Übergängen in wasserstoff-, helium- sowie lithiumartigen Schwefel-, Argon- und Eisenionen, Dissertation, Ruprecht-Karls-Universität Heidelberg, Deutschland
2010	R. Ginzel, Wechselwirkung niederenergetischer hochgeladener Ionen mit Materie, Dissertation, Ruprecht-Karls-Universität Heidelberg, Deutschland
2011	K. Schnorr, Laserspektroskopie hochgeladener Ionen an einer EBIT, Diplomarbeit, Justus-Liebig-Universität Gießen, Deutschland
2011	T. Ballance, Commissioning of a cryogenic Paul trap for highly charged ions, MSci thesis, Blackett Laboratory, Imperial College London, United Kingdom
2014	D. Hollain, Vermessung vom hochgeladenen Iridium mithilfe eines neuen Kontrollsystems, Masterarbeit, Ruprecht-Karls-Universität Heidelberg, Deutschland

Tabelle A.4: Abschlussarbeiten mittels der HD-EBIT von 2008 bis 2014

B Programmskripte

B.1 Python-Skript für NI-PCI Ansteuerung

```
1 # -*- coding: cp1252 -*-
2
3 # labbox class for labbox-server
4 # file: labbox_ni_01.py
5 # Only for NI Box (NI-PCI 6229 or NI-PCI 6703)
6 # no single in/outputs, they are listet in the tasks
7 # according to: http://pythonhosted.org/PyDAQmx/usage.html
8 # written in Python 2.7
9
10 # 13./17./27. 02. / 08./28./29. 04. / 07.05. 2014
11 # Daniel Hollain (master student)
12
13 import PyDAQmx
14 import PyDAQmx.DAQmxTypes
15 import time
16 import numpy
17
18 SAMPLESPERCHANNEL = 2
19 SAMPLESPERCHANNELHALF = int(SAMPLESPERCHANNEL/2.0)
20
21 # class
22 class labbox:
23     def __init__(self):
24         self.good = True
25
26         self.ais_list = []
27         self.ais_task = None
28         self.data_ais = None
29         self.ais_nr = 0
30         self.ai_arraysizeinsamps = 0
31         self.act_ais = []
32         self.last_ais = []
33         self.read = PyDAQmx.DAQmxTypes.int32()
34
35         self.aos_list = []
```

```
36          self.aos_task = None
37          self.data_aos = None
38          self.aos_nr = 0
39          self.last_aos = []
40
41          self.dis_list = []
42          self.dis_task = None
43          self.data_dis = None
44          self.dis_nr = 0
45          self.act_dis = []
46          self.last_dis = []
47          self.bytesPerSamp = PyDAQmx.DAQmxTypes.int32()
48
49          self.dos_list = []
50          self.dos_task = None
51          self.data_dos = None
52          self.dos_nr = 0
53          self.last_dos = []
54          return
55
56      # ckeck error
57      def chkerr(self,err):
58          if err:
59              raise
60          return None
61
62      def __del__(self):
63          if self.ais_task!=None:
64              self.ais_task.ClearTask()
65          if self.aos_task!=None:
66              self.aos_task.ClearTask()
67          if self.dis_task!=None:
68              self.dis_task.ClearTask()
69          if self.dos_task!=None:
70              self.dos_task.ClearTask()
71
72      def create_ais(self,channels):
73          try:
74              for i in channels.split(","):
75                  self.ais_list.append(i)
76              self.ais_nr = len(self.ais_list)
77              self.ai_arraysizeinsamps = SAMPLESPERCHANNEL*
    self.ais_nr
78
```

```
79              self.data_ais = []
80              for i in range(self.ai_arraysizeinsamps):
81                  self.data_ais.append(0.0)
82
83              self.data_ais = numpy.zeros(self.
    ai_arraysizeinsamps, dtype=numpy.float64)
84              self.ais_task = PyDAQmx.Task()
85              self.chkerr(self.ais_task.CreateAIVoltageChan(
    channels,"",PyDAQmx.DAQmx_Val_Cfg_Default,-10.0,10.0,
    PyDAQmx.DAQmx_Val_Volts,None))
86              self.chkerr(self.ais_task.CfgSampClkTiming(""
    ,10000.0,PyDAQmx.DAQmx_Val_Rising,PyDAQmx.
    DAQmx_Val_FiniteSamps,SAMPLESPERCHANNEL))
87
88              for i in self.ais_list:
89                  self.last_ais.append(0.0)
90                  self.act_ais.append(0.0)
91
92          except:
93              print "Error in create_ais():", channels
94              self.good = False
95          return None
96
97
98      def create_aos(self,channels):
99          try:
100             for i in channels.split(","):
101                 self.aos_list.append(i)
102             self.aos_nr = len(self.aos_list)
103
104             self.data_aos = []
105             for i in range(self.aos_nr):
106                 self.data_aos.append(0.0)
107
108             self.data_aos = numpy.zeros(self.aos_nr, dtype=
    numpy.float64)
109             self.aos_task = PyDAQmx.Task()
110             self.chkerr(self.aos_task.CreateAOVoltageChan(
    channels,"",-10.0,10.0,PyDAQmx.DAQmx_Val_Volts,""))
111
112             for i in self.aos_list:
113                 self.last_aos.append(0.0)
114         except:
115             print "Error in create_aos():", channels
```

```
116             self.good = False
117         return None
118 
119 
120     def create_dis(self,channels): # bit inputs
121         try:
122             for i in channels.split(","):
123                 self.dis_list.append(i)
124             self.dis_nr = len(self.dis_list)
125 
126             self.data_dis = []
127             for i in range(self.dis_nr):
128                 self.data_dis.append(False)
129 
130             self.data_dis = numpy.zeros(self.dis_nr, dtype=
    numpy.uint8)
131             self.dis_task = PyDAQmx.Task()
132             self.dis_task.CreateDIChan(channels,"",PyDAQmx.
    DAQmx_Val_ChanForAllLines)
133 
134             for i in self.dis_list:
135                 self.last_dis.append(0.0)
136                 self.act_dis.append(0.0)
137         except:
138             print "Error in create_dis():", channels
139             self.good = False
140         return None
141 
142 
143     def create_dos(self,channels):
144         try:
145             for i in channels.split(","):
146                 self.dos_list.append(i)
147             self.dos_nr = len(self.dos_list)
148 
149             self.data_dos = []
150             for i in range(self.dos_nr):
151                 self.data_dos.append(False)
152 
153             self.data_dos = numpy.zeros(self.dos_nr, dtype=
    numpy.uint8)
154             self.dos_task = PyDAQmx.Task()
155             self.chkerr(self.dos_task.CreateDOChan(channels,
    "",PyDAQmx.DAQmx_Val_ChanForAllLines))
```

```
156
157             for i in self.dos_list:
158                 self.last_dos.append(0.0)
159         except:
160             print "Error in create_dos():", channels
161             self.good = False
162         return None
163
164     def get_ais(self):
165         try:
166             self.chkerr(self.ais_task.StartTask())
167             self.chkerr(self.ais_task.ReadAnalogF64(
    SAMPLESPERCHANNEL,10.0,PyDAQmx.DAQmx_Val_GroupByChannel,
    self.data_ais,self.ai_arraysizeinsamps,PyDAQmx.byref(self
    .read),None))
168             self.chkerr(self.ais_task.StopTask())
169             for i in range(self.ais_nr):
170                 self.act_ais[i] = self.last_ais[i] = float(
    self.data_ais[SAMPLESPERCHANNEL*i+SAMPLESPERCHANNELHALF])
171             return self.act_ais
172         except:
173             print "Error in get_ais()"
174             return self.last_ais
175         return None
176
177     def set_ais(self,values):
178         return None
179
180
181
182     def set_aos(self,values):
183         try:
184             self.data_aos = numpy.array(values, dtype=numpy.
    float64)
185             self.chkerr(self.aos_task.StartTask())
186             self.chkerr(self.aos_task.WriteAnalogF64
    (1,1,10.0,PyDAQmx.DAQmx_Val_GroupByChannel,self.data_aos,
    None,None))
187             self.chkerr(self.aos_task.StopTask())
188             for i in range(self.aos_nr):
189                 self.last_aos[i] = values[i]
190             return True
191         except:
192             print "Error in set_aos():",values
```

```
193             return False
194         return None
195 
196     def get_aos(self):
197         return self.last_aos
198 
199 
200 
201     def get_dis(self):
202         try:
203             self.chkerr(self.dis_task.StartTask())
204             self.chkerr(self.dis_task.ReadDigitalLines(
    PyDAQmx.DAQmx_Val_Auto,10.0,PyDAQmx.
    DAQmx_Val_GroupByChannel,self.data_dis,self.dis_nr,
    PyDAQmx.byref(self.read),PyDAQmx.byref(self.bytesPerSamp)
    ,None))
205             self.chkerr(self.dis_task.StopTask())
206             for i in range(self.dis_nr):
207                 self.act_dis[i] = self.last_dis[i] = bool(
    self.data_dis[i])
208             return self.act_dis
209         except:
210             print "Error in get_dis()"
211             return self.last_dis
212         return None
213 
214     def set_dis(self,values):
215         return None
216 
217 
218 
219     def set_dos(self,values):
220         try:
221             self.data_dos = numpy.array(values, dtype=numpy.
    uint8)
222             self.chkerr(self.dos_task.StartTask())
223             self.chkerr(self.dos_task.WriteDigitalLines
    (1,1,10.0,PyDAQmx.DAQmx_Val_GroupByChannel,self.data_dos,
    None,None))
224             self.chkerr(self.dos_task.StopTask())
225 
226             for i in range(self.dos_nr):
227                 self.last_dos[i] = values[i]
228             return True
```

```
229             except:
230                 print "Error in set_dos():",values
231                 return False
232             return None
233 
234         def get_dos(self):
235             return self.last_dos
```

B.2 Python-Skript für NI-GPIB Ansteuerung (für Keithley 2002 Multimeter)

```
1 # -*- coding: cp1252 -*-
2
3 # labbox class for labbox-server
4 # file: labbox_gpib_01.py
5 # only for Keithley 2002 Multimeter GPIB devices
6 # according to: http://pyvisa.readthedocs.org/en/latest/
    tutorial.html
7 # written in Python 2.7
8
9 # 13./17./27. 02. / 08./28./29. 04. / 07.05. 2014
10 # Daniel Hollain (master student)
11
12
13 import time
14
15 # for GPIB
16 import visa
17 import struct
18 rm = visa.ResourceManager()
19
20 # class
21 class labbox:
22     def __init__(self):
23         self.good = True
24
25         self.gpibdevices = {}
26         self.act_val = {}
27         self.last_val = {}
28
29
30     def __del__(self):
31         for i in self.gpibdevices:
32             time.sleep(0.2)
33             self.gpibdevices[i].write("*rst")
34             time.sleep(0.2)
35             self.gpibdevices[i].write("*cls")
36             time.sleep(0.2)
37             self.gpibdevices[i].write(':init:cont on')
38         time.sleep(0.2)
39
```

```
40      # create channels from config-file
41      def create_ai(self,channel): # analog in
42          try:
43              self.gpibdevices[channel] = rm.get_instrument(
    channel)
44              self.gpibdevices[channel].write("*rst")
45              time.sleep(0.2)
46              self.gpibdevices[channel].write("status:preset")
47              time.sleep(0.2)
48              self.gpibdevices[channel].write("*cls")
49              time.sleep(0.2)
50              self.gpibdevices[channel].write("format:data
    sreal")
51              self.gpibdevices[channel].value_format = float
52
53             # non-SCPI commands implemented by Model 2001 (
    Keithley Multimeter)
54              self.gpibdevices[channel].write(":sens:volt:dc:
    range:auto 0") # autorange off
55              self.gpibdevices[channel].write(":sens:volt:dc:
    range:upp 10") #  upper range 10V
56              self.gpibdevices[channel].write(":sens:volt:dc:
    nplc 4") # highest accuracy (number of power line cycles)
57              self.gpibdevices[channel].write(":sens:volt:dc:
    aver:tcon rep") # repeating filter (fill stack, average,
    flush stack)
58              self.gpibdevices[channel].write(":sens:volt:dc:
    aver:coun 100") # average over 100 measurements
59
60              self.gpibdevices[channel].write(":init:cont on")
61
62              self.act_val[channel] = 0.0
63              self.last_val[channel] = 0.0
64          except:
65              print "Error in create_ai():",channel
66              self.good = False
67
68
69
70      def get_ai(self,channel):
71          try:
72              try:
73                  self.act_val[channel] = struct.unpack('<f',
    self.gpibdevices[channel].ask(":fetch?"))[0]
```

```
74              except:
75                  pass
76              self.last_val[channel] = self.act_val[channel]
77              return self.act_val[channel]
78          except:
79              return self.last_val[channel]
80
81      def set_ai(self,channel,val):
82          return None
```

B.3 Python-Basisskript Labboxserver

```
1 # -*- coding: cp1252 -*-
2
3 # Basis labbox-server
4 # for HDEBIT control system
5 # file: labbox_server_08.py
6 # written in Python 2.7
7
8 # needs labbox_ni_01.py for NI6229, NI6703
9 # needs labbox_gpib_01.py for IEEE488
10
11 # 10./12./13./17./28. 02. / 01./11./14./15./17./29. 04. /
     05./06./07./12. 05. 2014
12 # Daniel Hollain (master student)
13
14 # safety stop
15 safetystop = "not OK"
16 while safetystop != "OK":
17     safetystop = raw_input('labbox_server_08, safety stop\
18         \n close window to exit\n or \
19         \ntype \'OK\' and press enter to continue !  ')
20     print ""
21 print ""
22
23 # change name of configure-file is neccessary
24 CONFIGUREFILE = 'configure_labboxserver'
25
26 # imports
27 import time
28 from pcaspy import SimpleServer,Driver
29 import epics
30 from pcaspy.tools import ServerThread
31 import thread
32 import labbox_ni_01
33 import labbox_gpib_01
34
35 exec("import "+CONFIGUREFILE)
36 exec("cht = "+CONFIGUREFILE+".cht")
37 exec("prefix = "+CONFIGUREFILE+".prefix")
38 exec("bt = "+CONFIGUREFILE+".bt")
39
40 # card-table
```

```
41 exec("ct = "+CONFIGUREFILE+".ct")
42
43 for i in ct:
44     if ct[i]['desc']=='NI6229' or ct[i]['desc']=='NI6703':
45         ct[i]['string_ais'] = ""
46         ct[i]['string_aos'] = ""
47         ct[i]['string_dis'] = ""
48         ct[i]['string_dos'] = ""
49
50         ct[i]['pvlist_ais'] = []
51         ct[i]['pvlist_aos'] = []
52         ct[i]['pvlist_dis'] = []
53         ct[i]['pvlist_dos'] = []
54
55         ct[i]['set_aos'] = []
56         ct[i]['set_dos'] = []
57         ct[i]['ind_aos'] = {}
58         ct[i]['ind_dos'] = {}
59
60         ct[i]['ais'] = False
61         ct[i]['aos'] = False
62         ct[i]['dis'] = False
63         ct[i]['dos'] = False
64
65     elif ct[i]['desc']=='IEEE488':
66         ct[i]['chlist'] = []
67         ct[i]['pvlist_ais'] = []
68         ct[i]['ais'] = False
69
70
71 for channel in cht:
72     # label
73     for card in ct:
74         if card in channel:
75             if ct[card]['desc']=='NI6229' or ct[card]['desc'
    ]=='NI6703':
76                 string_for_ch = ""
77                 if cht[channel]['channel_type'] == 'ai':
78                     string_for_ch = 'string_ais'
79                     ct[card]['ais'] = True
80                     ct[card]['pvlist_ais'].append(channel+'.
    VAL')
81                 elif cht[channel]['channel_type'] == 'ao':
82                     string_for_ch = 'string_aos'
```

```
83                      ct[card]['aos'] = True
84                      ct[card]['pvlist_aos'].append(channel+'.
    VAL')
85                      ct[card]['set_aos'].append(0.0)
86                      ct[card]['ind_aos'][channel+'.VAL'] =
    len(ct[card]['set_aos'])-1
87                  elif cht[channel]['channel_type'] == 'di':
88                      string_for_ch = 'string_dis'
89                      ct[card]['dis'] = True
90                      ct[card]['pvlist_dis'].append(channel+'.
    VAL')
91                  elif cht[channel]['channel_type'] == 'do':
92                      string_for_ch = 'string_dos'
93                      ct[card]['dos'] = True
94                      ct[card]['pvlist_dos'].append(channel+'.
    VAL')
95                      ct[card]['set_dos'].append(False)
96                      ct[card]['ind_dos'][channel+'.VAL'] =
    len(ct[card]['set_dos'])-1
97
98                  if ct[card][string_for_ch]:
99                      ct[card][string_for_ch] += ','
100                 ct[card][string_for_ch] += channel
101
102
103
104             elif ct[card]['desc']=='IEEE488':
105                 if cht[channel]['channel_type'] == 'ai':
106                     ct[card]['ais'] = True
107                     ct[card]['chlist'].append(channel)
108                     ct[card]['pvlist_ais'].append(channel+'.
    VAL')
109
110 channel_list = []
111 for i in cht:
112     channel_list.append(i+'.VAL')
113
114 # process variables
115 pvdb = {'SERVER.COMMAND':{'type':'string','value':''},
116         'SERVER.RUNNING':{'type':'int','value':0},
117         'SERVER.SHUTDOWN':{'type':'int','value':0},
118         }
119 exec("pvdb['SERVER.KEEPER'] = {'type':'string','value':"+
    CONFIGUREFILE+".keeper}")
```

```
120
121 for card in ct:
122     pvdb[card+'.DELAY'] = {'type':'float', 'prec':4}
123
124 for i in cht:
125     # get type of channel
126     if cht[i]['channel_type'] == 'ao':
127         pvdb[i+'.VAL'] = {}
128         pvdb[i+'.VAL']['prec'] = 4
129         pvdb[i+'.VAL']['unit'] = cht[i]['unit']
130         pvdb[i+'.VAL']['lolo'] = cht[i]['limit_low']
131         pvdb[i+'.VAL']['low'] = cht[i]['limit_low']
132         pvdb[i+'.VAL']['high'] = cht[i]['limit_high']
133         pvdb[i+'.VAL']['hihi'] = cht[i]['limit_high']
134         pvdb[i+'.VAL']['lolim'] = cht[i]['limit_low']
135         pvdb[i+'.VAL']['hilim'] = cht[i]['limit_high']
136         pvdb[i+'.EGU'] = {'type':'string','value':cht[i]['
    unit']}
137         pvdb[i+'.NAME'] = {'type':'string','value':prefix+i}
138         pvdb[i+'.DESC'] = {'type':'string','value':'analog
    output'}
139
140     elif cht[i]['channel_type'] == 'ai':
141         pvdb[i+'.VAL'] = {}
142         pvdb[i+'.VAL']['prec'] = 4
143         pvdb[i+'.VAL']['unit'] = cht[i]['unit']
144         pvdb[i+'.VAL']['lolo'] = cht[i]['limit_low']
145         pvdb[i+'.VAL']['low'] = cht[i]['limit_low']
146         pvdb[i+'.VAL']['high'] = cht[i]['limit_high']
147         pvdb[i+'.VAL']['hihi'] = cht[i]['limit_high']
148         pvdb[i+'.VAL']['lolim'] = cht[i]['limit_low']
149         pvdb[i+'.VAL']['hilim'] = cht[i]['limit_high']
150         pvdb[i+'.EGU'] = {'type':'string','value':cht[i]['
    unit']}
151         pvdb[i+'.NAME'] = {'type':'string','value':prefix+i}
152         pvdb[i+'.DESC'] = {'type':'string','value':'analog
    input'}
153
154     elif cht[i]['channel_type'] == 'bi':
155         pvdb[i+'.VAL'] = {}
156         pvdb[i+'.VAL']['type'] = 'int'
157         pvdb[i+'.VAL']['unit'] = cht[i]['unit']
158         pvdb[i+'.VAL']['lolo'] = cht[i]['limit_low']
159         pvdb[i+'.VAL']['low'] = cht[i]['limit_low']
```

```
160             pvdb[i+'.VAL']['high'] = cht[i]['limit_high']
161             pvdb[i+'.VAL']['hihi'] = cht[i]['limit_high']
162             pvdb[i+'.VAL']['lolim'] = cht[i]['limit_low']
163             pvdb[i+'.VAL']['hilim'] = cht[i]['limit_high']
164             pvdb[i+'.EGU'] = {'type':'string','value':cht[i]['
        unit']}
165             pvdb[i+'.NAME'] = {'type':'string','value':prefix+i}
166             pvdb[i+'.DESC'] = {'type':'string','value':'digital
        input'}
167
168         elif cht[i]['channel_type'] == 'bo':
169             pvdb[i+'.VAL'] = {}
170             pvdb[i+'.VAL']['type'] = 'int'
171             pvdb[i+'.VAL']['unit'] = cht[i]['unit']
172             pvdb[i+'.VAL']['lolo'] = cht[i]['limit_low']
173             pvdb[i+'.VAL']['low'] = cht[i]['limit_low']
174             pvdb[i+'.VAL']['high'] = cht[i]['limit_high']
175             pvdb[i+'.VAL']['hihi'] = cht[i]['limit_high']
176             pvdb[i+'.VAL']['lolim'] = cht[i]['limit_low']
177             pvdb[i+'.VAL']['hilim'] = cht[i]['limit_high']
178             pvdb[i+'.EGU'] = {'type':'string','value':cht[i]['
        unit']}
179             pvdb[i+'.NAME'] = {'type':'string','value':prefix+i}
180             pvdb[i+'.DESC'] = {'type':'string','value':'digital
        output'}
181
182
183 # driver for server
184 class LDriver(Driver):
185
186     def __init__(self):
187         super(LDriver,self).__init__()
188         self.box = {}
189         self.setParam('SERVER.RUNNING',0)
190         self.updatePVs()
191         self.lock_thread_running = thread.allocate_lock()
192         self.restart_ready = True
193
194         for card in ct:
195             if ct[card]['desc']=='NI6229' or ct[card]['desc'
        ]=='NI6703':
196                 self.box[card] = labbox_ni_01.labbox()
197                 if ct[card]['ais']:
198                     self.box[card].create_ais(ct[card]['
```

```
    string_ais'])
199                 if ct[card]['aos']:
200                     self.box[card].create_aos(ct[card]['
    string_aos'])
201                 if ct[card]['dis']:
202                     self.box[card].create_dis(ct[card]['
    string_dis'])
203                 if ct[card]['dos']:
204                     self.box[card].create_dos(ct[card]['
    string_dos'])
205
206             elif ct[card]['desc']=='IEEE488':
207                 self.box[card] = labbox_gpib_01.labbox()
208                 if ct[card]['ais']:
209                     for ch in ct[card]['chlist']:
210                         self.box[card].create_ai(ch)
211
212         self.thread_box_on = {}
213         self.thread_box_running = {}
214
215         for i in ct:
216             self.thread_box_on[i] = False
217             self.thread_box_running[i] = False
218             self.setParam(i+'.DELAY',0.01)
219
220         return None
221
222     def post_constructor(self):
223         self.lock_thread_running.acquire()
224         self.start_running()
225         self.shutdown()
226         self.lock_thread_running.release()
227         return None
228
229     def start_running(self): # or restart
230         if self.getParam('SERVER.RUNNING'):
231             self.stop_running()
232
233         self.setParam('SERVER.RUNNING',0)
234         self.updatePVs()
235
236         for card in ct:
237             self.thread_box_on[card] = True
238             if ct[card]['desc']=='NI6229' or ct[card]['desc'
```

```
]=='NI6703':
239                 thread.start_new_thread(self.thread_box_ni,
(card,))
240             elif ct[card]['desc']=='IEEE488':
241                 thread.start_new_thread(self.thread_box_gpib
, (card,))
242             else:
243                 self.thread_box_on[card] = False
244
245         self.setParam('SERVER.RUNNING',1)
246         self.updatePVs()
247         return True
248
249     def stop_running(self): # stop server
250         if not self.getParam('SERVER.RUNNING'):
251             return False
252
253         # stop threads
254         for i in self.thread_box_on:
255             self.thread_box_on[i] = False
256
257         # wait for end
258         timestart = time.time()
259         timeout = 0.0
260         check = True
261         while check and timeout < 60.0:
262             check = False
263             for i in bt:
264                 check += self.thread_box_running[i]
265             timeout = time.time() - timestart
266
267         self.setParam('SERVER.RUNNING',0)
268         self.updatePVs()
269         return True
270
271
272     def __del__(self):
273         super(LDriver,self).__del__()
274
275     def pre_destructor(self):
276         self.lock_thread_running.acquire()
277         self.stop_running()
278         self.lock_thread_running.release()
279
```

```
280     def thread_box_ni(self,card):
281         self.thread_box_running[card] = True
282         while self.thread_box_on[card]:
283             if ct[card]['ais']:
284                 values = self.box[card].get_ais()
285                 ind=0
286                 for pv in ct[card]['pvlist_ais']:
287                     self.setParam(pv,values[ind])
288                     ind += 1
289                 self.updatePVs()
290 
291             if ct[card]['aos']:
292                 self.box[card].set_aos(ct[card]['set_aos'])
293 
294             if ct[card]['dis']:
295                 values = self.box[card].get_dis()
296                 ind=0
297                 for pv in ct[card]['pvlist_dis']:
298                     self.setParam(pv,values[ind])
299                     ind += 1
300                 self.updatePVs()
301 
302             if ct[card]['dos']:
303                 self.box[card].set_dos(ct[card]['set_dos'])
304 
305             if not self.thread_box_on[card]:
306                 self.thread_box_running[card] = False
307                 return None
308 
309             time.sleep(self.getParam(card+'.DELAY'))
310 
311         self.thread_box_running[card] = False
312         return None
313 
314 
315 
316     def thread_box_gpib(self,card):
317         self.thread_box_running[card] = True
318         while self.thread_box_on[card]:
319             if ct[card]['ais']:
320                 for i in range(len(ct[card]['chlist'])):
321                     value = self.box[card].get_ai(ct[card]['
    chlist'][i])
322                     self.setParam(ct[card]['pvlist_ais'][i],
```

```
    value)
323                     self.updatePVs()
324
325             if not self.thread_box_on[card]:
326                 self.thread_box_running[card] = False
327                 return None
328
329             time.sleep(self.getParam(card+'.DELAY'))
330         self.thread_box_running[card] = False
331         return None
332
333     # set Set-Values to zero
334     def shutdown(self):
335         # stop special programmed threads
336         # ...
337         for i in cht:
338             if cht[i]['channel_type'] == 'ai' or cht[i]['
    channel_type'] == 'bi':
339                 continue
340             if cht[i]['limit_low'] > 0.0:
341                 self.write(i+'.VAL',cht[i]['limit_low'])
342             else:
343                 self.write(i+'.VAL',0)
344         self.updatePVs()
345         return None
346
347     def thread_restart(self):
348         self.restart_ready = False
349         timestart = time.time()
350         timeout = 0.0
351         check = True
352         while check and timeout < 60.0:
353             check = False
354             for i in bt:
355                 check += self.thread_box_running[i]
356             timeout = time.time() - timestart
357         self.restart_ready = True
358
359
360     def read(self,pv):
361         return self.getParam(pv)
362
363
364     def write(self,pv,value):
```

```
365         if pv == 'SERVER.RUNNING':
366             self.lock_thread_running.acquire()
367             if value: # if true: (re)start all threads
368                 thread.start_new_thread(self.start_running,
    ())
369             else: # stop server
370                 thread.start_new_thread(self.stop_running,
    ())
371             self.lock_thread_running.release()
372             return True
373 
374         elif pv=='SERVER.KEEPER':
375             self.setParam(pv,value)
376             self.updatePVs()
377             return True
378 
379         if not self.getParam('SERVER.RUNNING'):
380             return False
381 
382         if pv in channel_list:
383             pv_red = pv[:-4]
384             if cht[pv_red]['channel_type'] == 'ao' or cht[
    pv_red]['channel_type'] == 'do':
385                 if cht[pv_red]['limit_low'] <= value <= cht[
    pv_red]['limit_high']:
386                     self.setParam(pv,value)
387                     self.updatePVs()
388                     if cht[pv_red]['channel_type'] == 'ao':
389                         ct[cht[pv_red]['box']]['set_aos'][
    ct[cht[pv_red]['box']]['ind_aos'][pv] ] = value
390                     elif cht[pv_red]['channel_type'] == 'do'
    :
391                         ct[cht[pv_red]['box']]['set_dos'][
    ct[cht[pv_red]['box']]['ind_dos'][pv] ] = value
392                 else:
393                     return False
394             else:
395                 return False
396 
397         elif pv == 'SERVER.SHUTDOWN':
398             if value==1:
399                 thread.start_new_thread(self.shutdown,())
400                 self.setParam(pv,0)
401             else:
```

```
402                 self.setParam(pv,value)
403             self.updatePVs()
404             return True
405 
406         elif pv.endswith('.DELAY'):
407             try:
408                 if value >= 0.0:
409                     self.setParam(pv,value)
410                     self.updatePVs()
411                     return True
412                 else:
413                     return False
414             except:
415                 return False
416 
417         else:
418             try:
419                 self.setParam(pv,value)
420                 self.updatePVs()
421                 return True
422             except:
423                 return False
424 
425         return False
426 
427 
428         self.updatePVs()
429         return True
430 
431     # for server-interpreter
432     def eval_command(self,command):
433         if command=='exit': # exit
434             return True
435 
436         elif command=='help': # show commands
437             print ""
438             print "Commands:"
439             print "{0:<25}show commands and their
    descriptions".format('help')
440             print "{0:<25}exit programm".format('exit')
441             print "{0:<25}stop server".format('stop')
442             print "{0:<25}restart server".format('restart')
443             print "{0:<25}set all SET-values to zero".format
    ('shutdown')
```

```
444             print "{0:<25}read channel".format('get <channel
    >')
445             print "{0:<25}put value (float!) to channel".
    format('put <channel> <value>')
446             print "{0:<25}return loop-delay of thread of <
    box>".format('delay <box>')
447             print "{0:<25}set loop-delay of thread of <box>
    to <new delay>".format('delay <box> <new delay>')
448             print "{0:<25}show status of server".format('
    status')
449             print "{0:<25}show the channels".format('
    showchannels')
450             print "{0:<25}is channel in server".format('is <
    channel>')
451             print ""
452
453         elif command=='stop':
454             self.lock_thread_running.acquire()
455             self.stop_running()
456             self.lock_thread_running.release()
457
458         elif command=='restart':
459             self.lock_thread_running.acquire()
460             self.start_running()
461             self.lock_thread_running.release()
462
463         elif command=='shutdown':
464             thread.start_new_thread(self.shutdown,())
465
466         elif command.startswith('get'):
467             channel = command[4:]
468             print ""
469             if channel in pvdb:
470                 print channel, self.getParam(channel)
471             else:
472                 print "Not there"
473             print ""
474
475         elif command.startswith('put'):
476             try:
477                 both = command[4:].split()
478                 if len(both) == 2:
479                     self.write(both[0],float(both[1]))
480                 else:
```

```
481                     raise
482             except:
483                 print ""
484                 print "wrong command"
485                 print ""
486
487         elif command.startswith('delay'):
488             try:
489                 cmds = command[6:].split()
490                 if len(cmds) == 1:
491                     print ""
492                     print self.getParam(cmds[0]+'.DELAY')
493                     print ""
494                 elif len(cmds) == 2:
495                     self.write(cmds[0]+'.DELAY',float(cmds
    [1]))
496                 else:
497                     raise
498             except:
499                 print ""
500                 print "wrong command"
501                 print ""
502
503         elif command=='status':
504             print ""
505             if self.getParam('SERVER.RUNNING'):
506                 print "RUNNING"
507             else:
508                 print "OFF"
509             print ""
510
511         elif command=='showchannels':
512             print ""
513             for i in cht:
514                 print i
515             print ""
516
517         elif command.startswith('is'):
518             channel = command[3:]
519             print ""
520             if channel in pvdb:
521                 print "Yes"
522             else:
523                 print "No"
```

```
524             print ""
525
526         elif command == '':
527             pass
528         else:
529             print ""
530             print "unknown command:",command
531             print ""
532
533         return False
534
535
536
537 # main programm
538 if __name__=='__main__':
539     server = SimpleServer()
540     server.createPV(prefix,pvdb)
541     driver = LDriver()
542     server.setDebugLevel(0)
543     server_thread = ServerThread(server)
544     server_thread.start()
545     driver.post_constructor()
546
547     print prefix+"Server is running, type \'exit\' to kill
      it ... \n"
548     while True:
549         cmd = raw_input(prefix+"Server> ")
550         stop = driver.eval_command(cmd)
551         if stop:
552             break
553
554     driver.pre_destructor()
555     server_thread.stop()
556     print prefix+"Server has stopped\n"
```

B.4 Python-Basisskript Konfiguration für Labboxserver

```
1 # -*- coding: cp1252 -*-
2
3 # configure-file for labbox_server_08_basis.py
4 # for HDEBIT control system
5 # file: configure_labboxserver_basis.py
6 # written in  Python 2.7
7
8 # configure according to the examples
9 # (without #)
10
11 # 11.04.2014
12 # Daniel Hollain (master student)
13
14 ################################
15
16 # keeper
17 # example:
18 # keeper = 'Daniel (Tel. 490)'
19
20 keeper = ''
21
22 ################################
23
24 # prefix (PCNAME:)
25 # example:
26 # prefix = 'EBITCON4:'
27
28 prefix = ''
29
30 ################################
31
32 # box-table ('device':'type')
33 # example:
34 #bt = {
35 #     'Dev1':'NI',
36 #     'Dev2':'NI',
37 #     'GPIB':'GPIB',
38 #     }
39
40 # NI for National Instruments
```

```
41 # GPIB for GPIB (e.g. Keithley Multimeter)
42 bt = {
43     }
44
45 ################################
46
47 # channel-Table
48 # example:
49 #cht = {
50 #     'Dev1/ao0':{'channel_type':'ao',
51 #         'box':'Dev1',
52 #         'limit_low':-10.0,
53 #         'limit_high':10.0,
54 #         'unit':'V'
55 #         },
56 #     'Dev2/ai3':{'channel_type':'ai',
57 #         'box':'Dev2',
58 #         'limit_low':-10.0,
59 #         'limit_high':10.0,
60 #         'unit':'V'
61 #         },
62 #     'GPIB::16':{'channel_type':'ai',
63 #         'box':'GPIB',
64 #         'limit_low':-30.0,
65 #         'limit_high':30.0,
66 #         'unit':'V'
67 #         },
68 #      }
69
70 cht = {
71     }
72
73 ################################
74
75 # card-tree
76 # example:
77 #ct = {
78 #     'Dev1':{
79 #         'desc': 'NI6703',
80 #         },
81 #
82 #     'Dev2':{
83 #         'desc': 'NI6229',
84 #         },
```

```
85 #
86 #     'GPIB':{
87 #         'desc': 'IEEE488',
88 #         },
89 #     }
90
91 ct = {
92     }
93
94 ################################
95
96 # you are finish, now save this document
97 # and start labbox_server_08_basis.py with doubleklick
```

```
1 # -*- coding: cp1252 -*-
2
3 # Basis device-server
4 # for HDEBIT control system
5 # file: device_server_08_basis.py
6 # written in Python 2.7
7
8 # 09./10./13./14. 02. / 01./08./14./15./17./18. 04. /
    05./06./26./27. 05. 2014
9 # Daniel Hollain (master student)
10
11 ### stop while writing un update
12 ##print "Server not ready"
13 ##print "write an update"
14 ##exit(0)
15
16 # safety stop
17 safetystop = "not OK"
18 while safetystop != "OK":
19     safetystop = raw_input('device_server_08, safety stop\
20         \n close window to exit\n or \
21         \ntype \'OK\' and press enter to continue !  ')
22
23 # change name of configure-file is neccessary
24 CONFIGUREFILE = "configure_deviceserver"
25
26 # imports
27 import time
28 import copy
29 from pcaspy import SimpleServer,Driver
30 from epics import caget, caput, camonitor, camonitor_clear,
    PV
31 from pcaspy.tools import ServerThread
32 import thread
33 exec("import "+CONFIGUREFILE)
34 import smtplib
35
36 # channel Table
37 exec("cht = "+CONFIGUREFILE+".cht")
38 exec("prefix = "+CONFIGUREFILE+".prefix")
39 exec("labbox_server = "+CONFIGUREFILE+".labbox_server")
40
41 exec("absender = "+CONFIGUREFILE+".absender")
42 exec("adressat = "+CONFIGUREFILE+".adressat")
```

```
43 exec("betreff = "+CONFIGUREFILE+".betreff")
44 exec("server_adresse = "+CONFIGUREFILE+".server_adresse")
45
46 # factor_up = 1.0/factor_down
47 for i in cht:
48     if cht[i]['factor_down'] != 0.0:
49         cht[i]['factor_up'] = 1.0/cht[i]['factor_down']
50     else:
51         cht[i]['factor_up'] = 0.0
52
53 # process variables
54 # put extra-PVs there
55 pvdb = {
56          'SERVER.RUNNING':{'type':'int','value':0},
57          'SERVER.INTERLOCK:BREAK':{'type':'int','value':0},
58          'SERVER.SHUTDOWN':{'type':'int','value':0},
59
60          'SERVER.TIME':{'type':'float','value':0.0},
61          'SERVER.TIME:TDELAY':{'type':'float','value':0.1},
62
63          'SERVER.RAMP:TDELAY':{'type':'float','value':0.01},
64          'SERVER.RAMP:ON':{'type':'int','value':0},
65          'SERVER.RAMP:TRUN':{'type':'int','value':0},
66          'SERVER.RAMP:CHANGE':{'type':'int','value':0},
67          }
68
69 exec("pvdb['SERVER.KEEPER'] = {'type':'string','value':"+
     CONFIGUREFILE+".keeper}")
70
71 # check channel table for i, put i in if i is not there
72 def chk_cht(i, desc, postfix, default=0.0, Typ='float'):
73     if desc in cht[i]:
74         pvdb[i+postfix] = {'type':Typ,'value':cht[i][desc]}
75     else:
76         cht[i][desc] = default
77         pvdb[i+postfix] = {'type':Typ,'value':default}
78     return
79
80
81 # translate channel table to process variables data base
82 for i in cht:
83     if 'limit_low' not in cht[i]:
84         cht[i]['limit_low'] = -1.0
85     if 'limit_high' not in cht[i]:
```

```
86          cht[i]['limit_high'] = 1.0
87      if 'unit' not in cht[i]:
88          cht[i]['unit'] = ""
89
90      pvdb[i+'.VAL'] = {'type':'float','value':0.0,'prec':0}
91      if not cht[i]['readonly']:
92          pvdb[i+'.SET'] = {'type':'float','value':0.0}
93          cht[i]['writeflag'] = True
94          chk_cht(i,'ramp_on','.RAMP:ON',0,'int')
95          chk_cht(i,'ramp_limit','.RAMP:LIMIT',100.0,'float')
96      else:
97          cht[i]['writeflag'] = False
98
99      if not cht[i]['readonly']:
100         chk_cht(i,'emergency_value','.INTERLOCK:EMERGENCYVAL
    ',0.0,'float')
101         chk_cht(i,'interlock_on','.INTERLOCK:ON',0,'int')
102     chk_cht(i,'interlock_low','.INTERLOCK:LOW',cht[i]['
    limit_low'],'float')
103     chk_cht(i,'interlock_high','.INTERLOCK:HIGH',cht[i]['
    limit_high'],'float')
104     chk_cht(i,'interlock_monitor','.INTERLOCK:MON',0,'int')
105
106     pvdb[i+'.VAL']['low'] = cht[i]['limit_low']
107     pvdb[i+'.VAL']['high'] = cht[i]['limit_high']
108     pvdb[i+'.VAL']['lolo'] = cht[i]['limit_low']
109     pvdb[i+'.VAL']['hihi'] = cht[i]['limit_high']
110     pvdb[i+'.VAL']['lolim'] = cht[i]['limit_low']
111     pvdb[i+'.VAL']['hilim'] = cht[i]['limit_high']
112     pvdb[i+'.VAL']['unit'] = cht[i]['unit']
113     if not cht[i]['readonly']:
114         pvdb[i+'.SET']['unit'] = cht[i]['unit']
115         pvdb[i+'.SET']['lolim'] = cht[i]['limit_low']
116         pvdb[i+'.SET']['hilim'] = cht[i]['limit_high']
117         pvdb[i+'.SET']['low'] = cht[i]['limit_low']
118         pvdb[i+'.SET']['high'] = cht[i]['limit_high']
119         pvdb[i+'.SET']['lolo'] = cht[i]['limit_low']
120         pvdb[i+'.SET']['hihi'] = cht[i]['limit_high']
121         pvdb[i+'.INTERLOCK:EMERGENCYVAL']['unit'] = cht[i]['
    unit']
122         pvdb[i+'.RAMP:LIMIT']['unit'] = cht[i]['unit']+'/s'
123
124     pvdb[i+'.HOPR'] = {'type':'float','value':cht[i]['
    limit_high']}
```

```
125     pvdb[i+'.LOPR'] = {'type':'float','value':cht[i]['
        limit_low']}
126     pvdb[i+'.HIGH'] = {'type':'float','value':cht[i]['
        limit_high']}
127     pvdb[i+'.LOW']  = {'type':'float','value':cht[i]['
        limit_low']}
128     pvdb[i+'.HIHI'] = {'type':'float','value':cht[i]['
        limit_high']}
129     pvdb[i+'.LOLO'] = {'type':'float','value':cht[i]['
        limit_low']}
130     pvdb[i+'.PREC'] = {'type':'int','value':0}
131     pvdb[i+'.EGU'] = {'type':'string','value':cht[i]['unit'
        ]}
132     pvdb[i+'.NAME'] = {'type':'string','value':prefix+i}
133
134     if cht[i]['readonly']:
135         pvdb[i+'.DESC'] = {'type':'string','value':'analog
        input for '+i}
136     else:
137         pvdb[i+'.DESC'] = {'type':'string','value':'analog
        output for '+i}
138
139     pvdb[i+'.CONN'] = {'type':'int','value':0}
140
141 # channel table translate (like the inverse)
142 cht_trans = {}
143 for i in cht:
144     cht_trans[cht[i]['channel']+'.VAL'] = i
145     cht_trans[prefix+i+'.VAL'] = i
146     cht_trans[prefix+i+'.SET'] = i
147
148 # driver for server
149 class BasisDriver(Driver):
150
151     # monitor input from labbox-server
152     def callback_input(self, pvname=None, value=None, **kwds
        ):
153         if pvname==None:
154             return None
155         self.setParam(cht_trans[pvname]+'.VAL',value*cht[
        cht_trans[pvname]]['factor_up'])
156         self.updatePVs()
157         return None
158
```

```
159     def callback_output(self, pvname=None, value=None, **
        kwds):
160         if pvname==None:
161             return None
162         val = self.getParam(cht_trans[pvname]+'.VAL')*cht[
        cht_trans[pvname]]['factor_down']
163         if val != value:
164             thread.start_new_thread(self.caput_thread, (
        pvname,val))
165         return None
166 
167     def caput_thread(self,pv,value):
168         try:
169             caput(pv,value,timeout=60.0)
170         except:
171             pass
172         return None
173 
174     def camonitor_thread(self,pv,cb):
175         try:
176             camonitor(pv,callback=cb)
177         except:
178             pass
179         return None
180 
181     # monitor connection status of labbox RUNNING
182     def conn_callback_labbox(self, pvname=None, conn=None,
        **kwds):
183         if pvname==None:
184             return None
185         pv_labbox = pvname[:-15]
186         if conn: # connected
187             self.labbox_connect[pv_labbox] = True
188             for i in cht:
189                 if cht[i]['labbox_server']==pv_labbox:
190                     self.setParam(i+'.CONN',1)
191                     if (not cht[i]['readonly']) and self.
        getParam('SERVER.RUNNING'):
192                         if (self.getParam('SERVER.INTERLOCK:
        BREAK') and self.getParam(i+'.INTERLOCK:ON')) or self.
        getParam(i+'.RAMP:ON'):
193                             cht[i]['writeflag'] = False
194                         else:
195                             cht[i]['writeflag'] = True
```

```
196                             thread.start_new_thread( \
197                                 self.caput_thread, \
198                                 (cht[i]['channel']+'.VAL',self.
    getParam(i+'.VAL')*cht[i]['factor_down']) \
199                             )
200                             if cht[i]['writeflag']:
201                                 self.setParam(i+'.VAL',self.
    getParam(i+'.SET'))
202         else: # not connected
203             self.labbox_connect[pv_labbox] = False
204             for i in cht:
205                 if cht[i]['labbox_server']==pv_labbox:
206                     self.setParam(i+'.CONN',0)
207                     cht[i]['writeflag'] = False
208         self.updatePVs()
209         return None
210
211
212     # monitor labbox RUNNING
213     def callback_labbox_running(self, pvname=None, value=
    None, **kwds):
214         if pvname==None:
215             return None
216         pv_labbox = pvname[:-15]
217         if value: # connected
218             self.labbox_connect[pv_labbox] = True
219             for i in cht:
220                 if cht[i]['labbox_server']==pv_labbox:
221                     self.setParam(i+'.CONN',1)
222                     if (not cht[i]['readonly']) and self.
    getParam('SERVER.RUNNING'):
223                         if (self.getParam('SERVER.INTERLOCK:
    BREAK') and self.getParam(i+'.INTERLOCK:ON')) or self.
    getParam(i+'.RAMP:ON'):
224                             cht[i]['writeflag'] = False
225                         else:
226                             cht[i]['writeflag'] = True
227                         thread.start_new_thread( \
228                             self.caput_thread, \
229                             (cht[i]['channel']+'.VAL',self.
    getParam(i+'.VAL')*cht[i]['factor_down']) \
230                         )
231                         if cht[i]['writeflag']:
232                             self.setParam(i+'.VAL',self.
```

```
    getParam(i+'.SET'))
233         else: # not connected
234             self.labbox_connect[pv_labbox] = False
235             for i in cht:
236                 if cht[i]['labbox_server']==pv_labbox:
237                     self.setParam(i+'.CONN',0)
238                     cht[i]['writeflag'] = False
239         self.updatePVs()
240         return None
241 
242     # monitor VAL and trigger interlock if VAL is out of
    range
243     # and change labbox-server-channel to VAL
244     def callback_VAL(self, pvname=None, value=None, **kwds):
245         if pvname==None:
246             return None
247         if self.getParam(cht_trans[pvname]+'.INTERLOCK:MON')
    :
248             ch = cht_trans[pvname]+'.INTERLOCK'
249             if value > self.getParam(ch+':HIGH') or value <
    self.getParam(ch+':LOW'):
250                 self.lock_thread_running.acquire()
251                 thread.start_new_thread(self.email_thread,(
    pvname,value))
252                 self.reset_interlock()
253                 self.setParam('SERVER.INTERLOCK:BREAK',1)
254                 self.updatePVs()
255                 self.lock_thread_running.release()
256             else:
257                 pv = cht_trans[pvname]
258                 if self.labbox_connect[cht[pv]['
    labbox_server']]:
259                     thread.start_new_thread(self.
    caput_thread,(cht[pv]['channel']+'.VAL',value*cht[pv]['
    factor_down']))
260         else:
261             pv = cht_trans[pvname]
262             if self.labbox_connect[cht[pv]['labbox_server'
    ]]:
263                 thread.start_new_thread(self.caput_thread,(
    cht[pv]['channel']+'.VAL',value*cht[pv]['factor_down']))
264         return None
265 
266     def email_thread(self,pv,value):
```

```
267         self.lock_thread_mail.acquire()
268         if (not self.free_mailing) or self.getParam('SERVER.
    INTERLOCK:BREAK'):
269             self.lock_thread_mail.release()
270             return None
271         self.free_mailing = False
272         self.lock_thread_mail.release()
273 
274         try:
275             zeit = time.ctime(time.time())
276             text = 'From: '+absender+'\n'
277             text += 'To: '+adressat+'\n'
278             text += 'Date: '+zeit+'\n'
279             text += 'Subject: '+betreff+'\n'
280             text += "Interlock\nPV: "+pv+", value: "+str(
    value)
281             server = smtplib.SMTP(server_adresse)
282             server.sendmail(absender,adressat,text)
283             server.quit()
284         except:
285             try:
286                 print ""
287                 print "warning interlock: no mail can be
    send"
288                 print "Interlock\nPV: "+pv+", value: "+str(
    value)
289                 print ""
290                 fil = open('mailerror_device_server_08.txt',
    'a')
291                 fil.write("Interlock\nPV: "+pv+", value: "+
    str(value)+'\n')
292                 fil.close()
293             except:
294                 print ""
295                 print "warning interlock: neither mail can
    be send or file can be written"
296                 print "Interlock\nPV: "+pv+", value: "+str(
    value)
297                 print ""
298         self.free_mailing = True
299         return None
300 
301     def __init__(self):
302         super(BasisDriver,self).__init__()
```

```
303         self.setParam('SERVER.INTERLOCK:BREAK',0)
304         self.setParam('SERVER.RUNNING'        ,0)
305 
306         self.server_time_running = True
307         thread.start_new_thread(self.server_time,())
308 
309         self.lock_thread_ramp      = thread.allocate_lock()
310         self.lock_thread_running   = thread.allocate_lock()
311         self.lock_thread_autoscale = thread.allocate_lock()
312 
313         self.free_mailing = True
314         self.lock_thread_mail      = thread.allocate_lock()
315 
316 
317         self.labbox_pvs = {}
318         self.labbox_connect = {}
319         for i in labbox_server:
320             self.labbox_pvs[i] = PV(i+':SERVER.RUNNING')
321             try:
322                 lrun = caget(i+':SERVER.RUNNING')
323                 if lrun:
324                     self.labbox_connect[i] = True
325                 else:
326                     self.labbox_connect[i] = False
327             except:
328                 self.labbox_connect[i] = False
329 
330         for i in cht:
331             for j in labbox_server:
332                 if cht[i]['labbox_server']==j:
333                     if self.labbox_connect[j]:
334                         self.setParam(i+'.CONN',1)
335                     else:
336                         self.setParam(i+'.CONN',0)
337 
338         for i in cht:
339             try:
340                 val = caget(cht[i]['channel']+'.VAL',timeout
    =60.0)*cht[i]['factor_up']
341             except:
342                 val = 0.0
343             self.setParam(i+'.VAL',val)
344             if not cht[i]['readonly']:
345                 #self.setParam(i+'.RAMP:ON',cht[i]['ramp_on
```

```
        ']) # already done
346                 #self.setParam(i+'.RAMP:LIMIT',cht[i]['
    ramp_limit']) # already done
347                 self.setParam(i+'.SET',val)
348                 if cht[i]['ramp_on'] or (not self.getParam(i
    +'.CONN')):
349                     cht[i]['writeflag'] = False
350         self.updatePVs()
351         return None
352
353     def post_constructor(self):
354         self.lock_thread_running.acquire()
355         self.start_running()
356         for i in labbox_server:
357             self.labbox_pvs[i].connection_callbacks.append(
    self.conn_callback_labbox)
358             self.labbox_pvs[i].add_callback(self.
    callback_labbox_running)
359         self.lock_thread_running.release()
360
361     def pre_destructor(self):
362         self.server_time_running = False
363         self.lock_thread_running.acquire()
364         self.stop_running()
365         for i in labbox_server:
366             self.labbox_pvs[i].disconnect()
367         self.lock_thread_running.release()
368
369     def server_time(self):
370         timestart = time.time()
371         self.setParam('SERVER.TIME',0.0)
372         self.updatePVs()
373         while self.server_time_running:
374             timenow = time.time()-timestart
375             self.setParam('SERVER.TIME',timenow)
376             self.updatePVs()
377             time.sleep(self.getParam('SERVER.TIME:TDELAY'))
378         return None
379
380     def start_ramp_thread(self):
381         self.lock_thread_ramp.acquire()
382         if self.getParam('SERVER.RAMP:CHANGE'):
383             self.lock_thread_ramp.release()
384             return False
```

```
385             self.setParam('SERVER.RAMP:CHANGE',1)
386             self.updatePVs()
387             self.lock_thread_ramp.release()
388 
389             thread.start_new_thread(self.ramp_thread, ())
390             while not self.getParam('SERVER.RAMP:TRUN'):
391                 time.sleep(0.01)
392             self.setParam('SERVER.RAMP:CHANGE',0)
393             self.updatePVs()
394             return True
395 
396         def stop_ramp_thread(self):
397             self.lock_thread_ramp.acquire()
398             if self.getParam('SERVER.RAMP:CHANGE'):
399                 self.lock_thread_ramp.release()
400                 return False
401             self.setParam('SERVER.RAMP:CHANGE',1)
402             self.updatePVs()
403             self.lock_thread_ramp.release()
404 
405             self.setParam('SERVER.RAMP:ON',0)
406             self.updatePVs()
407             while self.getParam('SERVER.RAMP:TRUN'):
408                 time.sleep(0.001)
409             self.setParam('SERVER.RAMP:CHANGE',0)
410             self.updatePVs()
411             return True
412 
413         def start_running(self): # or restart
414             if self.getParam('SERVER.RUNNING'):
415                 self.stop_running()
416 
417             self.setParam('SERVER.RUNNING',0)
418             self.updatePVs()
419             self.start_ramp_thread()
420 
421             for i in cht:
422                 try:
423                     if cht[i]['readonly']:
424                         camonitor(cht[i]['channel']+'.VAL',
    callback=self.callback_input)
425                         camonitor(prefix+i+'.VAL',callback=self.
    callback_VAL)
426                 else:
```

```
427                     if (not self.getParam(i+'.CONN')) or
    self.getParam(i+'.RAMP:ON'):
428                         cht[i]['writeflag'] = False
429                     else:
430                         cht[i]['writeflag'] = True
431
432                     if cht[i]['writeflag']:
433                         self.setParam(i+'.VAL',self.getParam
    (i+'.SET'))
434
435                     camonitor(prefix+i+'.VAL',callback=self.
    callback_VAL)
436                     camonitor(cht[i]['channel']+'.VAL',
    callback=self.callback_output)
437             except:
438                 if not cht[i]['readonly']:
439                     if cht[i]['writeflag']:
440                         self.setParam(i+'.VAL',self.getParam
    (i+'.SET'))
441
442         self.setParam('SERVER.RUNNING',1)
443         self.setParam('SERVER.INTERLOCK:BREAK',0)
444
445         self.updatePVs()
446         return True
447
448     def stop_running(self): # stop server
449         if not self.getParam('SERVER.RUNNING'):
450             return False
451         self.stop_ramp_thread()
452
453         for i in cht:
454             try:
455                 if cht[i]['readonly']:
456                     camonitor_clear(cht[i]['channel']+'.VAL'
    )
457                 else:
458                     cht[i]['writeflag'] = False
459                     camonitor_clear(cht[i]['channel']+'.VAL'
    )
460                     camonitor_clear(prefix+i+'.VAL')
461                     caput(cht[i]['channel']+'.VAL',self.
    getParam(i+'.VAL')*cht[i]['factor_down'])
462             except:
```

```
463                 pass
464
465         self.setParam('SERVER.RUNNING',0)
466         self.updatePVs()
467         return True
468
469
470     def __del__(self):
471         super(BasisDriver,self).__del__()
472
473
474     def ramp_thread(self):
475         self.setParam('SERVER.RAMP:ON',1)
476         timefifo = [time.time()]
477         time.sleep(self.getParam('SERVER.RAMP:TDELAY'))
478         factor = 1.0
479         self.setParam('SERVER.RAMP:TRUN',1)
480         self.updatePVs()
481         while self.getParam('SERVER.RAMP:ON'):
482             for i in cht:
483                 if (not cht[i]['readonly']) and self.
    getParam(i+'.RAMP:ON'):
484                     act_val = self.getParam(i+'.VAL')
485                     end_val = self.getParam(i+'.SET')
486
487                     if act_val<end_val: # ramp up
488                         act_val += self.getParam(i+'.RAMP:
    LIMIT') * factor
489                         if act_val>end_val:
490                             act_val=end_val
491                     elif act_val>end_val: # ramp down
492                         act_val -= self.getParam(i+'.RAMP:
    LIMIT') * factor
493                         if act_val<end_val:
494                             act_val=end_val
495
496                     if (not self.getParam('SERVER.RAMP:ON'))
     or (not self.labbox_connect[cht[i]['labbox_server']]) or
     (self.getParam(i+".INTERLOCK:ON") and self.getParam('
    SERVER.INTERLOCK:BREAK')):
497                         pass
498                     else:
499                         self.setParam(i+'.VAL',act_val)
500             self.updatePVs()
```

```
501             timefifo.append(time.time())
502             timefifo = timefifo[-50:]
503             factor = (timefifo[-1]-timefifo[0])/(len(
    timefifo)-1)# + self.ramp_thread_delay - self.
    ramp_thread_delay
504             time.sleep(self.getParam('SERVER.RAMP:TDELAY'))
505         self.setParam('SERVER.RAMP:TRUN',0)
506         self.updatePVs()
507         return None
508
509
510     # set Set-Values to zero
511     def shutdown(self):
512         # stop special programmed threads
513         # ...
514         for i in cht:
515             if cht[i]['readonly']:
516                 continue
517             self.write(i+'.SET',cht[i]['emergency_value'])
518             # writeflag for emergency_value not necessary
519         self.updatePVs()
520         return None
521
522
523     # reset everything except inputs
524     def reset(self):
525         # stop special programmed threads
526         # ...
527         for i in cht:
528             if cht[i]['readonly']:
529                 continue
530             self.setParam(i+'.RAMP:ON',cht[i]['ramp_on'])
531             if (not self.getParam(i+'.CONN')) or (self.
    getParam('SERVER.INTERLOCK:BREAK') and self.getParam(i+'.
    INTERLOCK:ON')) or cht[i]['ramp_on']:
532                 cht[i]['writeflag'] = False
533             else:
534                 cht[i]['writeflag'] = True
535             self.setParam(i+'.RAMP:LIMIT',cht[i]['ramp_limit
    '])
536             if cht[i]['limit_low'] > 0.0:
537                 self.write(i+'.SET',cht[i]['limit_low'])
538             else:
539                 self.write(i+'.SET',0.0)
```

```
540             self.updatePVs()
541             return None
542 
543 
544         # reset everything to zero immediately
545         def reset_interlock(self):
546             # stop special programmed threads
547             # ...
548             for i in cht:
549                 if not cht[i]['readonly']:
550                     if self.getParam(i+'.INTERLOCK:ON'):
551                         self.setParam(i+'.RAMP:ON',0)
552                         cht[i]['writeflag'] = False
553                         self.setParam(i+'.SET',self.getParam(i+'
    .INTERLOCK:EMERGENCYVAL'))
554                         self.setParam(i+'.VAL',self.getParam(i+'
    .INTERLOCK:EMERGENCYVAL'))
555                         if not self.getParam('SERVER.RUNNING'):
556                             try:
557                                 if not cht[i]['readonly']:
558                                     caput(cht[i]['channel']+'.
    VAL',self.getParam(i+'.INTERLOCK:EMERGENCYVAL')*cht[i]['
    factor_down'])
559                             except:
560                                 print "error in reset_interlock
    ():",cht[i],"is not ready"
561             self.updatePVs()
562             return None
563 
564 
565         def read(self,pv):
566             return self.getParam(pv)
567 
568         def write(self,pv,value):
569             if pv.endswith('.VAL'):
570                 pv_red = pv[:-4]
571                 if not cht[pv_red]['writeflag']:
572                     return False
573                 else:
574                     if cht[pv_red]['limit_low'] <= value <= cht[
    pv_red]['limit_high']:
575                         self.setParam(pv_red+'.SET',value)
576                         self.setParam(pv,value)
577                         self.updatePVs()
```

```
578                     return True
579
580         elif pv.endswith('.SET'):
581             pv_red = pv[:-4]
582             if cht[pv_red]['limit_low'] <= value <= cht[
    pv_red]['limit_high']:
583                 self.setParam(pv,value)
584                 if cht[pv_red]['writeflag']:
585                     self.setParam(pv_red+'.VAL',value)
586                 self.updatePVs()
587                 return True
588             else:
589                 return False
590
591         elif pv == 'SERVER.SHUTDOWN':
592             if value==1:
593                 thread.start_new_thread(self.shutdown,())
594                 self.setParam(pv,0)
595             elif value==2:
596                 thread.start_new_thread(self.reset,())
597                 self.setParam(pv,0)
598             elif value==3:
599                 thread.start_new_thread(self.email_thread,('
    manueller interlock',0))
600                 thread.start_new_thread(self.reset_interlock
    ,())
601                 self.setParam(pv,0)
602             else:
603                 self.setParam(pv,value)
604             self.updatePVs()
605             return True
606
607         elif pv == 'SERVER.INTERLOCK:BREAK':
608             self.lock_thread_running.acquire()
609             erg = True
610             if self.getParam(pv):
611                 if value: # reset
612                     thread.start_new_thread(self.
    email_thread,('manueller interlock',0))
613                     self.reset_interlock()
614                 if not value:
615                     erg = False
616             else:
617                 if value: # stop server
```

```
618                     thread.start_new_thread(self.
    email_thread,('manueller interlock',0))
619                     self.reset_interlock()
620                     self.setParam('SERVER.INTERLOCK:BREAK'
    ,1)
621                     self.updatePVs()
622
623             self.lock_thread_running.release()
624             return erg
625
626
627         elif pv.endswith('.INTERLOCK:LOW'):
628             pv0 = pv[:-14] + '.VAL'
629             pvdb[pv0]['lolo'] = value
630             self.setParam(pv,value)
631             self.updatePVs()
632             return True
633
634         elif pv.endswith('.INTERLOCK:HIGH'):
635             pv0 = pv[:-15] + '.VAL'
636             pvdb[pv0]['hihi'] = value
637             self.setParam(pv,value)
638             self.updatePVs()
639             return True
640
641         elif pv == 'SERVER.RUNNING':
642             self.lock_thread_running.acquire()
643             if value: # if true: (re)start all threads
644                 thread.start_new_thread(self.start_running,
    ())
645             else: # stop server
646                 thread.start_new_thread(self.stop_running,
    ())
647             self.lock_thread_running.release()
648             return True
649
650         elif pv.endswith('.RAMP:ON'):
651             if value == self.getParam(pv):
652                 return True
653             i = pv[:-8]
654             pv_set = prefix + i + '.SET'
655             if value: # on
656                 cht[i]['writeflag'] = False
657             elif self.getParam('SERVER.RUNNING'): # off and
```

```
        Server runs
658                 if (not self.getParam(i+'.CONN')) or (self.
    getParam('SERVER.INTERLOCK:BREAK') and self.getParam(i+'.
    INTERLOCK:ON')):
659                     cht[i]['writeflag'] = False
660                 else:
661                     cht[i]['writeflag'] = True
662                 if cht[i]['writeflag']:
663                     self.setParam(i+'.VAL',self.getParam(i+'
    .SET')))
664             self.setParam(pv,value)
665             self.updatePVs()
666             return True
667
668         elif pv.endswith('.RAMP:LIMIT'):
669             if value < 0.0:
670                 return False
671             self.setParam(pv,value)
672             self.updatePVs()
673             return True
674
675         elif pv == 'SERVER.RAMP:ON':
676             if value: # on
677                 thread.start_new_thread(self.
    start_ramp_thread,())
678             else: # off
679                 thread.start_new_thread(self.
    stop_ramp_thread,())
680             return True
681
682         # read-only PVs
683         elif pv=='SERVER.RAMP:TRUN' or pv=='SERVER.RAMP:
    CHANGE' or \
684              pv=='SERVER.TIME' or pv.endswith('.CONN'):
685             return False
686
687         # elif pv == special channels:
688             # ...
689
690         else:
691             try:
692                 self.setParam(pv,value)
693                 self.updatePVs()
694                 return True
```

```
695             except:
696                 return False
697 
698         return False
699 
700     # for server-interpreter
701     def eval_command(self,command):
702         if command=='exit': # exit
703             return True
704 
705         elif command=='help': # show commands
706             print ""
707             print "Commands:"
708             print "{0:<25}show commands and their
    descriptions".format('help')
709             print "{0:<25}exit programm".format('exit')
710             print "{0:<25}stop server".format('stop')
711             print "{0:<25}restart server".format('restart')
712             print "{0:<25}set all SET-values to zero".format
    ('shutdown')
713             print "{0:<25}reset everything".format('reset')
714             print "{0:<25}make an interlock".format('
    interlock')
715             print "{0:<25}make an interlockt".format('itl')
716             print "{0:<25}make an interlock".format('il')
717             print "{0:<25}read channel".format('get <channel
    >')
718             print "{0:<25}put value (float!) to channel".
    format('put <channel> <value>')
719             print "{0:<25}show status of server".format('
    status')
720             print "{0:<25}show the channels".format('
    showchannels')
721             print "{0:<25}is channel in server".format('is <
    channel>')
722             print ""
723 
724         elif command=='stop':
725             self.lock_thread_running.acquire()
726             self.stop_running()
727             self.lock_thread_running.release()
728 
729         elif command=='restart':
730             self.lock_thread_running.acquire()
```

```
731             self.start_running()
732             self.lock_thread_running.release()
733 
734         elif command=='shutdown':
735             thread.start_new_thread(self.shutdown,())
736 
737         elif command=='reset':
738             thread.start_new_thread(self.reset,())
739 
740         elif command=='interlock' or command=='itl' or
    command=='il':
741             self.lock_thread_running.acquire()
742             thread.start_new_thread(self.email_thread,('
    manueller interlock',0))
743             self.reset_interlock()
744             self.setParam('SERVER.INTERLOCK:BREAK',1)
745             self.updatePVs()
746             self.lock_thread_running.release()
747 
748         elif command.startswith('get'):
749             channel = command[4:]
750             print ""
751             if channel in pvdb:
752                 print channel, self.getParam(channel)
753             else:
754                 print "Not there"
755             print ""
756 
757         elif command.startswith('put'):
758             both = command[4:].split()
759             if len(both) != 2:
760                 print ""
761                 print "wrong command"
762                 print ""
763             else:
764                 self.write(both[0],float(both[1]))
765 
766         elif command=='status':
767             print ""
768             if self.getParam('SERVER.RUNNING'):
769                 print "RUNNING"
770             else:
771                 if self.getParam('SERVER.INTERLOCK:BREAK'):
772                     print "OFF INTERLOCK"
```

```
773                 else:
774                     print "OFF"
775             print ""
776 
777         elif command=='showchannels':
778             print ""
779             for i in cht:
780                 print i
781             print ""
782 
783         elif command.startswith('is'):
784             channel = command[3:]
785             print ""
786             if channel in pvdb:
787                 print "Yes"
788             else:
789                 print "No"
790             print ""
791 
792         elif command == '':
793             pass
794         else:
795             print ""
796             print "unknown command:",command
797             print ""
798 
799         return False
800 
801 # main programm
802 if __name__=='__main__':
803     server = SimpleServer()
804     server.createPV(prefix,pvdb)
805     driver = BasisDriver()
806     server.setDebugLevel(0)
807 
808     server_thread = ServerThread(server)
809     server_thread.start()
810     time.sleep(3.0)
811     driver.post_constructor()
812     print prefix+"Server is running, type \'exit\' to kill
      it ... \n"
813     while True:
814         cmd = raw_input(prefix+"Server> ")
815         stop = driver.eval_command(cmd)
```

```
816             if stop:
817                 break
818         driver.pre_destructor()
819         time.sleep(3.0)
820         server_thread.stop()
821         print prefix+"Server has stopped\n"
```

B.5 Python-Basisskript Konfiguration für Deviceserver

```
1 # -*- coding: cp1252 -*-
2
3 # configure-file for device_server_08_basis.py
4 # for HDEBIT control system
5 # file: configure_device_server_basis.py
6 # written in  Python 2.7
7
8 # configure according to the examples
9 # (without #)
10
11 # 12.04.2014
12 # Daniel Hollain (master student)
13
14 ################################
15
16 # keeper
17 # example:
18 #keeper = 'Daniel (Tel. 490)'
19
20 keeper = ''
21
22 ################################
23
24 # prefix (HDEBIT:DEVICE:)
25 # example:
26 # prefix = 'HDEBIT:EGUN:'
27
28 prefix = ''
29
30 ################################
31
32 # mail data
33
34 # example:
35 #absender = 'hollain@mpi-hd.mpg.de'
36 #adressat = 'hollain@mpi-hd.mpg.de'
37 #betreff = 'alarm interlock deviceserver'
38 #server_adresse = 'mail.mpi-hd.mpg.de'
39
40 absender = ''
```

```
41 adressat = ''
42 betreff = ''
43 server_adresse = ''
44
45 ################################
46
47 # labbox-server ('PCNAME' of labbox-server)
48 # list of used labbox-server
49 # example:
50 #labbox_server = [
51 #     'EBITCON4',
52 #     'EBITCON6',
53 #     ]
54
55 labbox_server = [
56     ]
57
58 ################################
59
60 # channel-Table
61 # example:
62 #cht = {
63 #     'DT1':{'channel':'EBITCON4:Dev1/ao1',
64 #         'labbox_server':'EBITCON4',
65 #         'factor_down':10.0/3500.0,
66 #         'limit_low':0.0,
67 #         'limit_high':500.0,
68 #         'emergency_value':0.0,
69 #         'readonly':False,
70 #         'unit':'V',
71 #         },
72 #     'DT2':{'channel':'EBITCON4:Dev1/ao2',
73 #         'labbox_server':'EBITCON4',
74 #         'factor_down':0.002,
75 #         'limit_low':0.0,
76 #         'limit_high':500.0,
77 #         'emergency_value':0.0,
78 #         'readonly':False,
79 #         'unit':'V',
80 #         },
81 #     'ANODE:CURRENT:IST':{'channel':'EBITCON6:Dev1/ai2',
82 #         'labbox_server':'EBITCON6',
83 #         'factor_down':10.0/2.5,
84 #         'limit_low':0.0,
```

```
85 #         'limit_high':2.5,
86 #         'readonly':True,
87 #         'unit':'mA',
88 #         'interlock_monitor':1,
89 #         'interlock_low':-0.45,
90 #         'interlock_high':0.45,
91 #         },
92 #      'ANODE:VOLTAGE:IST':{'channel':'EBITCON6:Dev1/ai3',
93 #         'labbox_server':'EBITCON6',
94 #         'factor_down':10.0/12500.0,
95 #         'limit_low':0.0,
96 #         'limit_high':12500.0,
97 #         'readonly':True,
98 #         'unit':'V',
99 #         },
100 #       }
101
102 cht = {
103       }
104
105 ################################
106
107 # you are finish, now save this document
108 # and start device_server_08_<servername>.py with
      doubleklick
```

C Liste

C.1 Abbildungsverzeichnis

C.2 Tabellenverzeichnis

D Literaturverzeichnis

S. Bernitt. Optimierung der Ladungszustandsverteilung in einer EBIT durch resonante Photorekombination. Master's thesis, Ruprecht-Karls-Universität Heidelberg, Deutschland, 2009.

Crespo14. persönliches Gepsräch mit J.R. Crespo López-Urrutia, 2014.

D. A. Knapp, P. Beiersdorfer, M. H. Chen, J. H. Scofield, and D. Schneider. Observation of Interference between Dielectronic Recombination and Radiative Recombination in Highly Charged Uranium Ions. *PHYSICAL REVIEW LETTERS, 74, 54*, 1995.

EPICS. URL http://www.aps.anl.gov/epics/.

EPICSRecord. URL https://wiki-ext.aps.anl.gov/epics/index.php/RRM_3-14.

H. F. Beyer, H.-J. Kluge, V. P. Shevelko. *X-Ray Radiation of Highly Chargd Ions*. Springer-Verlag Berlin Heidelberg, 1997.

D. Hollain. Reproduzierbarkeitsstudie zur Bestimmung der Wellenlänge der 2S1/2 2P1/2 , 3/2 Übergänge des Li-artigen Eisenions, 2012.

I. V. Hertel, C.-P. Schulz. *Atome, Moleküle und optische Physik 1*. Springer-Verlag Berlin Heidelberg, 2008.

A. J. González Martínez. *Quantum interference in the dielectronic recombination of heavy highly charged ions*. PhD thesis, Ruperto-Carola University of Heidelberg, Germany, 2005.

V. Mäckel. Spektroskopische Untersuchung dielektronischer Resonanzen von hochgeladenen Ionen. Master's thesis, Ruprecht-Karls-Universität Heidelberg, Deutschland, 2006.

NI. URL http://germany.ni.com/.

NIST. URL http://www.nist.gov/.

Nuc14, 2014. URL http://nucleardata.nuclear.lu.se/.

PCASPy. URL https://code.google.com/p/pcaspy/.

PyEPICS. URL http://cars9.uchicago.edu/software/python/pyepics3/.

PyVISA. URL http://pyvisa.readthedocs.org/en/1.5-docs/.

R. D. Deslattes et al. X-ray transition energies: new approach to a comprehensive evaluation. *Reviews of modern physics, Vol. 75*, januray 2003.

ROOT. URL http://root.cern.ch/drupal/.

Scofield03. J. H. Scofield, private Kommunikation, 2003.

Erklärung:

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 25.09.2014 .....