TECHNISCHE UNIVERSITÄT WIEN

ATOMINSTITUT

DIPLOMARBEIT

Planung und Aufbau einer neuen Neutronenradiographieanlage am TRIGA Mark II Forschungsreaktor der TU Wien

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Master-Studiums

Physikalische Energie- und Messtechnik

eingereicht von

Clemens Trunner
Matrikelnummer 01226220

ausgeführt am **Atominstitut**
der Fakultät für **Physik**
der **Technischen Universität Wien**

unter der Anleitung von
Privatdoz. Dipl.-Ing. Dr.techn. Stephan SPONAR
Mitwirkung: Ass.Prof. Dipl.-Ing. Dr.techn. Michael ZAWISKY

Wien, 07.02.2024

(Unterschrift Verfasser/in)

(Unterschrift Betreuer/in)

Für meinen Papa

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Arbeit unterstützt und begleitet haben.

Zuerst möchte ich mich bei Stephan Sponar bedanken, der meine Diplomarbeit betreut und begutachtet hat. Des weiteren möchte ich Michael Zawisky für seine fachliche Unterstützung zum Thema Neutronenradiographie danken. Außerdem gilt mein Dank Andreas Musilek, der sich unter anderem um die Finanzierung der neuen Radiographieanlage gekümmert hat. Für die Idee, die Pläne sowie den regen fachlichen Austausch möchte ich mich bei Burkhard Schillinger und seinem Team von der Forschungsneutronenquelle Heinz Maier-Leibnitz (FRMII) bedanken. Aber der größte Dank gebührt mit Sicherheit meiner Frau Katharina, die mich immer motiviert hat, weiterzumachen, und die mit der allergrößten Aufgabe betraut ist, meine schriftlichen Ergüsse zu lektorieren.

Ich möchte mich auch bei meinen Kolleginnen und Kollegen von der TU Wien bedanken, die mich immer tatkräftig unterstützt haben.

- Vom TRIGA Center Atominsitut: Mario Villa, Robert Bergmann, Thomas Stummer, Dieter Hainz und Monika Veit-Öller
- Von der EDV: Helmut Richter und Christian Völker
- Von der Mechanischen Werkstatt: Rudi Gergen, Heinz Matusch, Roman Flasch, Fabian Quurk und Walter Klikovich

Abschließend danke ich auch allen anderen Kolleginnen und Kollegen, die zum Gelingen dieser Arbeit beigetragen haben, und natürlich meiner Familie, die stets für mich da ist.

Zusammenfassung

Das Neutron als elektrisch neutrales Teilchen besitzt Eigenschaften, die sich von denen der Elektronen, Protonen oder Photonen unterscheiden. Freie Neutronen sind instabil und wechselwirken nicht über die Coulombkraft. Da Neutronen aber Bausteine der Atomkerne sind, unterliegen sie der Kernkraft. Neben den genannten Eigenschaften gibt es noch zahlreiche weitere, die dem Neutron in der Physik einen besonderen Stellenwert geben. Wesentlich ist jedoch nicht alleine die Erforschung des Neutrons und seiner Eigenschaften, sondern auch sein Spezifikum, selbst zum Werkzeug der Physik zu werden. Das ist beispielsweise bei der Neutronenradiographie der Fall. Anders als bei Röntgenaufnahmen werden dabei Neutronen genutzt, um die innere Struktur von Objekten abzubilden. Die Besonderheit besteht darin, dass Neutronen nicht mit der Atomhülle, sondern mit dem Atomkern wechselwirken. Dadurch können gänzlich andere Informationen über ein Objekt gewonnen werden. Am TRIGA Mark II Forschungsreaktor der TU Wien wird genau das gemacht.

Im Rahmen dieser Arbeit wurde die bisherige Neutronenradiographieanlage, die auf einer mit Flüssigstickstoff gekühlten CCD-Kamera basiert hat, ersetzt. Ein neues, modernes Experiment ist dabei entstanden. Besonderes Augenmerk bei der Planung und dem anschließenden Aufbau der neuen Radiographiestation wurde auf die Funktion der TU Wien als Forschungseinrichtung gelegt. Anstatt des Ankaufs einer kommerziellen Anlage wurde ein eigenes Design nach Plänen der TU München entworfen, die bereits eine ähnliche Anlage betreiben. Die neue Neutronenradiographiestation ist ein kostengünstiges Instrument, das dem aktuellen Stand der Entwicklung voll entspricht. Das ist durch die Nutzung eines 3D-Druckers gelungen, mit dem ein Großteil der Bestandteile selbst designt und gefertigt wurde. Auch die verbaute CMOS-Astrokamera mit Peltierkühlung ist eine deutliche Verbesserung gegenüber dem alten System. Mitsamt der neuen Messsoftware und der eigens entwickelten Steuerung ist eine komplette und leistungsfähige Anlage entstanden. An der TU Wien steht nun ein modernes Instrument für Lehre und Forschung zur Verfügung, um Radiographie und Tomographie mit Neutronen zu betreiben.

Abstract

The neutron as an electrically neutral particle has properties different to those of electrons, protons or photons. Free neutrons are unstable and have no interaction with the coulomb force. Neutrons as part of the atomic nucleus subject to the nuclear force. In addition to the named properties, multiple others are existent, which gives the neutron a particular status in physics. It is not only the research on neutrons and their properties that is essential, but also its feature of becoming a tool in physics itself. Unlike X-ray imaging, neutrons are used to indicate the internal structure of objects. The special feature is that neutrons do not interact with the atomic shell, but with the nucleus. Due to this fact, completely different information about an object can be obtained. This is being done at the TRIGA Mark II research reactor at the TU Wien.

As part of this thesis, the previous neutron radiography system based on a liquid nitrogen cooled CCD-camera was replaced. Therefore a new, modern unit was created. During the designing and successive construction of the new radiography station, special attention was paid to the function of the TU Wien as a research facility. Instead of purchasing a commercial system, a special design was created based on plans from the Technical University of Munich, where a similar system is already in use. The new neutron radiography station is a cost-effective instrument that fully corresponds to the state of art technology. This was achieved by using a 3D printer to manufactur most of the parts in-house. The built-in CMOS astro camera with peltier cooling is also a significant improvement over the former system. With the self-developed measuring software and the specifically built controller, a complete and powerful facility was created. By that, a modern instrument for teaching and research is on hand at the TU Wien to perform radiography and tomography with neutrons.

Inhaltsverzeichnis

1 Einführung

1.1 Historischer Abriss

Die Geschichte der Radiographie begann in der Antike, wo bereits bekannt war, wie sich Objekte mit Licht abbilden lassen. Aussgehend von dem einfachen Prinzip der Lochkamera entwickelte sich im 19. Jahrhundert die Fotografie und das dauerhafte Abbilden von Objekten wurde möglich. Auch die von Wilhelm Conrad Röntgen entdeckte Röntgenstrahlung wurde bald darauf eingesetzt, um damit Objekte abzubilden. Mit Entdeckung der Quantenphysik im beginnenden 20. Jahrhundert kamm es erstmals zur Beschreibung des Wellen-Teilchen-Dualismus. In Analogie zur gut erforschten Lichtoptik gelang das erste Experiment mit Materiewellen [4] im Jahr 1927. Die Entdeckung des Neutrons durch James Chadwick [3] im Jahr 1932 makierte den Beginn der Physik des Neutrons und dessen Erforschung. Mit Neutronen von einem kleinen Neutronengenerator begannen die beiden deutschen Wissenschafter Hartmut Kallmann und Ernst Kuhn mit ersten Experimenten zur Neutronenradiographie. Die erste Publikation von Otto Peter [15] zum Thema Neutronenradiographie erschien im Jahr 1946. Kurz darauf publizierte auch Hartmut Kallmann [12] zum Thema Neutronenradiographie. Der Grundstein der Neutronenradiographie ist damit gelegt. Die besonderen Eigenschaften des Neutrons und die Art seiner Wechselwirkungen mit Materie ermöglichen eine gänzlich andere Art der Abbildung im Vergleich zu Photonen. Die Neutronenradiographie sowie die Neutronentomographie finden heute Anwendung in der zerstörungsfreien Analyse von Proben und sind nach wie vor Gegenstand aktueller Forschung.

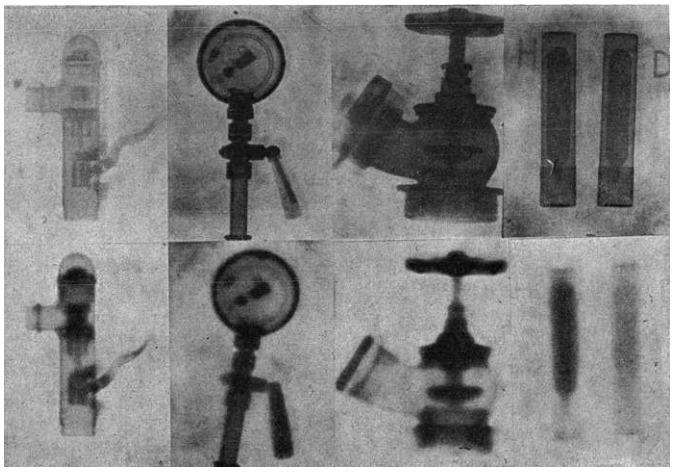


Abbildung 1.1: Durchleuchtungsaufnahmen mit langsamen Neutronen (unten) und Vergleichsbilder mit γ-Strahlen (oben) von Otto Peter aus dem Jahr 1944 [15]

1.2 Zur vorliegenden Arbeit

Am Forschungsreaktor der TU Wien wird eine Neutronenradiographie- und -tomographieanlage betrieben. Die bestehende Anlage rund um die stickstoffgekühlte CCD-Kamera entspricht nicht mehr dem aktuellen Stand der Technik. Daher wurde im Zuge dieser Arbeit das alte System ersetzt und ein neues und modernes Instrument aufgebaut. Der Fokus des Aufbaus lag primär darauf, den Anforderungen der TU Wien als Forschungs- und Bildungseinrichtung zu genügen.
Zu Beginn der Arbeit wird der theoretische Hintergrund zur Neutronenradiographie beleuchtet. Das umfasst sowohl die Grundlagen der Neutronenphysik als auch die Grundlagen der Radio- und Tomographie mit Neutronen. Im darauf folgenden Kapitel werden der detaillierte Aufbau am Forschungsreaktor beschrieben und daran anschließend die zur Überprüfung der Funktion durchgeführten Messungen. Im letzten Kapitel finden sich abschließende Überlegungen und mögliche Weiterentwicklungen. Der Anhang beinhaltet die gesammelten Datenblätter der verbauten Komponenten sowie die im Zuge dieser Arbeit entstandenen Schaltpläne, Konstruktionszeichnungen und Quellcodes.

2 Neutronenphysik

2.1 Neutronen

2.1.1 Entdeckung des Neutrons

Neutronen sind Teilchen, welche keine elektrische Ladung besitzen und annähernd die Masse eines Protons haben. Gemeinsam mit den Protonen bilden sie die Bausteine der Atomkerne. Der Umstand, dass Neutronen keine Ladung besitzen und mit den ähnlich schweren Protonen im Kern gebunden sind, erschwerte ihre Entdeckung. Die ersten Hinweise auf das Neutron lieferten Untersuchungen zum rutherfordschen Atommodell Anfang des 20. Jahrhunderts. Rutherford selbst postulierte 1920, dass der Atomkern aus positiv geladenen Protonen und aus von ihm *neutral doublet* genannten neutralen Teilchen mit ähnlicher Masse aufgebaut ist.[17]
Den experimentellen Beweis für die Existenz dieser neutralen Teilchen lieferte Chadwick im Jahr 1932. Er konnte bereits über die Masse von Deuteron und Proton sowie die Bindungsenergie des Deuterons die Masse des Neutrons berechnen.[3] Bis heute ist das Modell des Atomkerns, welcher aus Z Protonen und $A-Z$ Neutronen besteht, gültig. In der Teilchenphysik spielt die Bestimmung der Eigenschaften des Neutrons, wie zum Beispiel die exakte Lebenszeit oder ein potenziell endliches elektrisches Dipolmoment, nach wie vor eine zentrale Rolle.

2.1.2 Freie Neutronen

Als Kernbausteine sind Neutronen im Atomkern stabil gebunden. Ist das Neutron jedoch nicht in einem Kern gebunden und somit frei, ist es instabil mit einer mittleren Lebensdauer von $878{,}4 \pm 0{,}5\,\mathrm{s}$ [7]. Beim Zerfall des freien Neutrons entstehten ein Proton, ein Elektron und ein Elektronen-Antineutrino.

$$n \rightarrow p^+ + e^- + \bar{\nu}_e + 0{,}78\,\mathrm{MeV} \tag{2.1}$$

Mit einer Masse von $1{,}674\,93 \cdot 10^{-27}\,\mathrm{kg}$ [7] besitzen sie auch eine potentielle Energie im Schwerefeld der Erde. Neben der potentiellen Energie im Gravitationsfeld haben Neutronen auch kinetische Energie. Über die Neutronenmasse resultieren daraus auch eine Geschwindigkeit v und eine de-Broglie-Wellenlänge λ.

$$E = \frac{m_n \cdot v^2}{2} = \frac{h^2}{2 \cdot m_n \cdot \lambda^2} = k_B \cdot T \tag{2.2}$$

Zudem kann einem Neutron über seine kinetische Energie eine Temperatur mittels der Boltzmannkonstante zugeordnet werden. In der Praxis werden die Neutronen in Energiebereiche eingeteilt und wie aus Tabelle 2.1 ersichtlich entsprechend der zugehörigen

Energie	Temperatur	Bezeichnung
<300 neV	<4 mK	ultrakalte Neutronen
0,3 - 100 µeV	0,004 - 1 K	sehr kalte Neutronen
0,1 - 10 meV	1 - 120 K	kalte Neutronen
5 - 100 meV	60 - 1200 K	thermische Neutronen
100 - 500 meV	1200 - 6000 K	schnelle Neutronen
>500 meV	>6000 K	epithermische Neutronen

Tabelle 2.1: Neutronenbezeichnung nach Energie und Temperatur

Masse	$A_r(n) = 1{,}008\,664\,916\,00 \pm 0{,}000\,000\,000\,59$ u
mittlere Lebensdauer	$\tau_n = 878{,}4 \pm 0{,}5$ s
magnetisches Moment	$\mu_n = 1{,}913\,042\,7 \pm 0{,}000\,000\,5\,\mu_N$
elektrische Ladung	$q_n = 0$
Spinquantenzahl	$s_n = 1/2$

Tabelle 2.2: Fundamentale Eigenschaften des freien Neutrons [7]

Temperatur bezeichnet. Freie Neutronen bilden ihrerseits eigene Kerne (${}^{0}_{1}\mathrm{n}$). Sie können mit anderen Kernen wechselwirken und nehmen damit an Kernreaktionen teil. Neutronen können an Atomkernen elastisch oder inelastisch gestreut, aber auch absobiert werden.

2.1.3 Neutronenquellen

Es gibt verschiedene Methoden, um Neutronen freizusetzen. Chadwick benutzte für den ersten Nachweis die Reaktion von α-Teilchen mit Berilliumkernen.

$$\alpha + {}^{9}\mathrm{Be} \longrightarrow {}^{12}\mathrm{C} + n + 5{,}704\,\mathrm{MeV}$$

Ein α-Teilchen wird vom Berillium absorbiert, dadurch entsteht ein instabiler Kohlenstoffkern. Dieser Zwischenkern zerfällt unter Abgabe eines Neutrons und Energie im MeV-Bereich zu ${}^{12}\mathrm{C}$. Es kann auch ganz ohne äußeren Einfluss von anderen Teilchen zur spontanen Spaltung kommen und damit zum Freiwerden von Neutronen. Dies tritt bei schweren Kernen wie zum Beispiel bei ${}^{252}\mathrm{Cf}$ ein. In der Neutronenphysik und speziell in der Neutronenoptik liefern diese Reaktionen in den meisten Fällen eine zu geringe Intensität. Um höhere Neutronenflussdichten zu erhalten, gibt es im wesentlichen drei Möglichkeiten.

(i) Kernreaktionen hervorgerufen von beschleunigten Teilchen. Lässt man zum Beispiel beschleunigte Deuteriumkerne auf ein Tritiumtarget treffen, entsteht dabei ein Heliumkern sowie 10^{-4} Neutronen pro Deuteron.

$$D + T \longrightarrow {}^{4}\mathrm{He} + n + 17{,}59\,\mathrm{MeV}$$

Diese als Kernfusion bezeichneten Reaktionen können in Abhängigkeit der Projektilgeschwindigkeit und des Aufprallwinkels einen nahezu monochromatischen Neutronenstrahl erzeugen.

(ii) Kernspaltung benutzt spaltbare Isotope wie ^{235}U in Kernreaktoren. Die Spaltung wird von thermischen Neutronen ausgelöst, im Schnitt werden 2,4 Neutronen pro Kernspaltung freigesetzt.

$$n + {}^{235}\mathrm{U} \longrightarrow A_1 + A_2 + \nu_{fi} \cdot n + 200\,\mathrm{GeV}$$

A_1 und A_2 stellen die verschiedenen Spaltprodukte dar. Ein großer Vorteil von Neutronen aus dem Reaktor ist der kontinuierliche Neutronenfluss, der durch die selbsterhaltende Kettenreaktion im Reaktor entsteht.

(iii) Spallation-Reaktionen verwenden hochenergetische Protonen (1 MeV) aus Teilchenbeschleunigern. Mit diesen werden schwere Kerne beschossen und unter Zerstörung des Target-Kerns Neutronen freigesetzt.

$$n + {}^{238}\mathrm{U} \longrightarrow A_1 + A_2 + \ldots + A_n + \nu_{sp} \cdot n + 2{,}5\,\mathrm{GeV}$$

Bei Spallation-Reaktionen werden bis zu 50 Neutronen pro Proton erzeugt.

Bei den drei beschriebenen Vorgängen werden zumeist schnelle Neutronen mit Energie im MeV-Bereich frei. In einem Moderator, bestehend aus Kernen mit niedriger Ordnungszahl, werden die entstandenen Neutronen durch elastische Stöße abgebremst. Die so erhaltenen thermischen Neutronen stehen dann für neutronenoptische Experimente zur Verfügung.[10]

2.2 Wechselwirkung mit Materie

2.2.1 Neutronen als Materiewelle

Materiellen Teilchen können neben den bekannten Teilcheneigenschaften wie der Energie E und dem Impuls $\vec{p}$ auch Eigenschaften einer Welle wie die Kreisfrequenz ω und der Wellenvektor $\vec{k}$ zugeordnet werden. Die daraus folgende quantenmechanische Beschreibung des Neutrons geschieht mittels der Wellenfunktion $\psi(\vec{r},t)$, welche die gesamte Information des Teilchens enthält. Für die Wellenfunktion $\psi(\vec{r},t)$ des Neutrons mit der Masse m gilt unter dem Einfluss eines äußeren Potentials $V(\vec{r},t)$ die Schrödingergleichung.

$$-\frac{\hbar^2}{2m}\Delta\psi(\vec{r},t) + V(\vec{r})\psi(\vec{r},t) = i\hbar\frac{\partial}{\partial t}\psi(\vec{r},t) \tag{2.3}$$

Die zeitabhängige Schrödingergleichung ist eine partielle Differenzialgleichung der zweiten Ordnung. Sie ist linear und homogen in ψ. Die Schrödingergleichung besitzt auch eine stationäre Form, in der es keine Abhängigkeit von der Zeit gibt.

$$-\frac{\hbar^2}{2m}\Delta\psi(\vec{r}) + V(\vec{r})\psi(\vec{r}) = E\psi(\vec{r}) \tag{2.4}$$

Für ein freies Neutron ($V(\vec{r},t) = 0$) folgt aus der Schrödingergleichung (2.3) die Lösung der Wellenfunktion in Form einer ebenen Welle.

$$\psi(\vec{r},t) = A \cdot e^{i(\vec{k}\vec{r}-\omega_{\vec{k}}t)} \tag{2.5}$$

Das freie Neutron besitzt mit dieser Lösung den Impuls

$$\vec{p} = \hbar\vec{k} \tag{2.6}$$

sowie die Energie

$$E = \frac{\hbar^2\vec{k}^2}{2m} = \hbar\omega_{\vec{k}} \quad . \tag{2.7}$$

Die Ausbreitungsgeschwindigkeit des Neutrons entspricht der Gruppengeschwindigkeit der Materiewelle.

$$\vec{v}_{gr} = \frac{\mathrm{d}\omega}{\mathrm{d}t} = \frac{\hbar\vec{k}}{m} = \frac{\vec{p}}{m} := \vec{v} \tag{2.8}$$

Neben der Gruppengeschwindigkeit kann auch eine Phasengeschwindigkeit angegeben werde. Sie ist jene Geschwindigkeit, mit der sich ein Punkt konstanter Amplitude auf der Welle bewegt.

$$\vec{v}_{ph} = \frac{\omega}{\vec{k}} = \frac{\hbar\vec{k}}{2m} = \frac{1}{2}\vec{v} \tag{2.9}$$

Bei der Beschreibung von Materiewellen durch ebene Wellen entspricht die Phasengeschwindigkeit der halben Teilchengeschwindigkeit. Das ist dem Umstand geschuldet, dass die Kreisfrequenz ω eine Funktion des Wellenvektors $\vec{k}$ ist und von diesem quadratisch abhängt. Dieses Phänomen wird als Dispersion bezeichnet. Der mathematische Zusammenhang von Kreisfrequenz und Wellenvektor folgt direkt aus Gleichung (2.7) und wird als Dispersionsrelation bezeichnet.

$$\omega_{\vec{k}} = \frac{\hbar\vec{k}^2}{2m_n} \tag{2.10}$$

Die Lösung der Wellenfunktion in Gleichung (2.5) ist ident zu jener von ebenen Wellen in der Optik, nur die Beziehung von ω zu $\vec{k}$ unterscheidet sich. Im Gegensatz zur linearen Dispersionsrelation elektromagnetischer Wellen $\omega = ck$ weisen Neutronen eine quadratische Dispersionsrelation auf. Diese Beobachtungen lassen darauf schließen, dass in der Neutronenoptik ähnliche Phänomene wie in der Lichtoptik auftreten.[5]

2.2.2 Absorbtionswirkungsquerschnitt

Ausgehend von der zeitabhängigen Schrödingergleichung (2.3) können unter Verwendung des komplexen optischen Potentials

$$V(\vec{r}) = V'(\vec{r}) - iV''(\vec{r}) \tag{2.11}$$

der Streu- und Absorptionswirkungsquerschnitt hergeleitet werden.
Durch Einführen der Beziehungen

$$\begin{aligned} \rho &= |\psi|^2 \\ \vec{J} &= \mathrm{Re}\{\psi^*\vec{v}\psi\} \\ s &= \frac{2}{\hbar}\rho V'' \end{aligned} \tag{2.12}$$

wird die Schrödingergleichung (2.3) in die Form

$$\dot{\rho} + \operatorname{div} \vec{J} = -s \tag{2.13}$$

übergeführt. Diese Schreibweise wird als Kontinuitätsgleichung aufgefasst. Dabei ist ρ die mittlere Teilchendichte der Neutronen, $\vec{J}$ die mittlere Neutronenflussdichte mit der Neutronengeschwindigkeit $\vec{v}$ und s die Dichte der Neutronensenken. Die Gesamtzahl der Neutronen in einem beliebigen Volumen V ist gegeben durch

$$N = \int_V \rho \, \mathrm{d}\vec{r} \quad . \tag{2.14}$$

Die zeitliche Veränderung der Neutronen kann unter Verwendung der Gleichung (2.13) wie folgt geschrieben werden.

$$\dot{N} = -\int_V (\operatorname{div} \vec{J} + s) \, \mathrm{d}\vec{r} \tag{2.15}$$

Nach dem gaußschen Theorem kann $\dot{N}$ umgeschrieben werden zu

$$\dot{N} = \int_s \vec{J}(-\vec{n}) \, \mathrm{d}S - \int_V s \, \mathrm{d}\vec{r} \quad , \tag{2.16}$$

wobei S die Fläche um das Volumen V ist und dS ein Flächenelement von S. Des weiteren ist $\vec{n}$ ein Vektor normal zur Oberfläche S, der aus dem Volumen V zeigt.
Gleichung (2.16) veranschaulicht, dass die zeitliche Änderung der Neutronen gleich der in das Volumen V eintretenden Neutronen minus der im Volumen absorbierten Neutronen ist. Betrachtet man in dieser Gleichung den gesamten Raum, verschwindet der erste Teil und es bleibt

$$\dot{N} = -\int_V s \, \mathrm{d}\vec{r} \quad . \tag{2.17}$$

Wenn das optische Potential reellwertig ist, also V'' verschwindet, wird $s = 0$ und damit auch $\dot{N}$. Dadurch kann die Normierung $N = 1$ gewählt werden, deren Konsequenz $\int \rho d\vec{r} = 1$ ist. Mit dieser Normierung erhält ρ die übliche Interpretation als Wahrscheinlichkeitsdichte. Ist das optische Potential wie anfangs angenommen jedoch komplex, bleibt die Interpretation für ρ als Neutronenteilchendichte bestehen, da die Zahl der Neutronen aufgrund von Absorption nicht erhalten bleibt.
In den weiteren Betrachtungen wird von einem stationären Zustand bei Streuung ausgegangen. Nun kann der sogenannte differenzielle Streuwirkungsquerschnitt $\mathrm{d}\Sigma$ angegeben werden. Er ist definiert als die mittlere Anzahl der einfallenden Neutronen, die in den Raumwinkel $\mathrm{d}\Omega$ pro Zeiteinheit und einfallenden Fluss gestreut werden.

$$\frac{\mathrm{d}\Sigma}{\mathrm{d}\Omega} = |f(\theta)|^2 \tag{2.18}$$

Die Größe $f(\theta)$ wird als Streuamplitude bezeichnet und ist die Amplitude der an einem Target gestreuten Welle. Betrachtet man alle Raumwinkel, so wird $\mathrm{d}\Sigma$ zum totalen Streuwirkungsquerschnitt.

$$\Sigma = \int \mathrm{d}\Sigma = \int_{4\pi} |f(\theta)|^2 \, \mathrm{d}\Omega \tag{2.19}$$

Mit dem totalen Streuwirkungsquerschnitt Σ können Streuprozesse von Neutronen beschrieben werden.
Zur Beschreibung von Absorptionsprozessen ist der Absorptionswirkungsquerschnitt Σ_a die relevante Größe. Er ist definiert als die mittlere Zahl der einfallenden Neutronen, die pro Einheit des einfallenden Flusses und pro Einheit der Zeit absorbiert werden.

$$\Sigma_a = \frac{1}{J} \int_V s \, \mathrm{d}\vec{r} \tag{2.20}$$

Die Größe V ist in diesem Fall ein beliebiges Volumen, das den Atomkern für den Absorptionsprozess enthält. Unter Verwendung der Beziehung (2.16) für den stationären Fall ($\dot{N} = 0$) folgt

$$\Sigma_a = \frac{1}{J} \int_S \vec{J}' \cdot (-\hat{n}) \, \mathrm{d}S \tag{2.21}$$

mit der totalen Neutronenflussdichte $\vec{J}'$. Diese enthält sowohl einfallende als auch auslaufende (gestreute) Neutronen. Für die weitere Berechnung von Σ_a wird das Volumen V als Kugel mit dem Radius r angenommen. Durch Bilden des Grenzwerts $r \to \infty$ ergibt sich

$$\Sigma_a = \frac{r^2}{J} \int_{4\pi} \vec{J}' \cdot (\hat{r}) \, \mathrm{d}\Omega \quad . \tag{2.22}$$

Unter Zuhilfenahme der Partialwellenzerlegung für freie Neutronen und des Formalismus der Streulänge kann der Absorptionswirkungsquerschnitt als

$$\Sigma_a = \frac{\pi}{k^2} \sum_{l=0}^{\infty} (2l+1)[1 - |e^{2i\delta_l}|^2] \tag{2.23}$$

geschrieben werden. Die Größe δ_l wird als Phasenschub bezeichnet und resultiert aus dem optischen Potential $V(\vec{r})$. Die Koeffizienten l benennen die Partialwellen ($l = 0, 1, 2, 3, \ldots$), die meistens mit s, p, d, f, ... bezeichnet werden.
Da das optische Potential im Allgemeinen eine komplexe Größe ist, ist auch der Phasenschub als komplex zu betrachten.

$$\delta_l = \delta_l' + \delta_l'' \tag{2.24}$$

Somit lässt sich Σ_a unter den zuvor getroffenen Annahmen über den Imaginärteil des Phasenschubs ausdrücken.

$$\Sigma_a = \frac{\pi}{k^2} \sum_{l=0}^{\infty} (2l+1)[1 - e^{-4\delta_l''}] \tag{2.25}$$

Ist das optische Potential reellwertig, so ist auch der Phasenschub reell und es folgt $\delta_l'' = 0$ mit Gleichung (2.25) auch $\Sigma_a = 0$. Das bedeutet, sowohl Streuung, als auch Absorption resutieren aus der selben physikalischen Größe, dem komplexen optischen Potential. Dabei können der Realteil des Potentials mit Streuung und der Imaginärteil mit Absorption in Verbindung gebracht werden.
Die Summe aus totalem Streuwirkungsquerschnitt Σ_s und Absorptionswirkungsquerschnitt Σ_a ergibt den totalen Wirkungsquerschnitt Σ_t.

$$\Sigma_t = \Sigma_s + \Sigma_a \tag{2.26}$$

Der totale Wirkungsquerschnitt beschreibt die mittlere Anzahl einfallender Neutronen, die pro Zeiteinheit entweder gestreut oder absorbiert werden. Der Wirkungsquerschnitt stellt also die Wahrscheinlichkeit dar, dass ein Neutron mit einem Kern in Wechselwirkung tritt und ist dabei von der Energie des Neutrons, der Stabilität des Kerns und der Art der Wechselwirkung abhängig.[20]

2.2.3 Absorbtion thermischer Neutronen

Die gesonderte Betrachtung der Absorption von thermischen Neutronen hat mehrere Gründe. An Neutronenquellen stehen zumeist thermische Neutronen zur Verfügung, außerdem ist es für thermische Neutronen ausreichend, nur den Term $l = 0$ der Partialwellenzerlegung zu berücksichtigen. Die Streuamplitude kann daher in Form von

$$f(\theta) = \frac{1}{k \cot(\delta_0 - ik)} \tag{2.27}$$

angegeben werden. Die Streuamplitude kann auch als Reihenentwicklung geschrieben werden.

$$f(\theta) = -a + ika^2 + \mathcal{O}(k^2) \tag{2.28}$$

Hierbei wird die Streulänge a als Eigenschaft des optischen Potentials eingeführt. In der Praxis stellt die Streulänge eine phänomenologische Konstante dar, die im Experiment bestimmt wird. Geht man von einer komplexen Streulänge

$$a = a' - ia'' \tag{2.29}$$

aus, so lässt sich mit Hilfe der Formel (2.28) der differenzielle Streuwirkungsquerschnitt wie folgt angeben

$$\frac{\mathrm{d}\Sigma}{\mathrm{d}\Omega} = |a|^2[1 - 2ka'' + \mathcal{O}(k^2)] \tag{2.30}$$

und mit der Gleichung (2.19) ergibt sich der totale Streuwirkungsquerschnitt für thermische Neutronen.

$$\Sigma_s = 4\pi|a|^2[1 - 2ka'' + \mathcal{O}(k^2)] \tag{2.31}$$

Mit dem optischen Theorem

$$\Sigma_t = \frac{4\pi}{k}\,\mathrm{Im}[f(0)] \tag{2.32}$$

sowie der Beziehung (2.26) ergibt dies für den Absorptionswirkungsquerschnitt

$$\Sigma_a = \frac{4\pi}{k}a''[1 - 2ka'' + \mathcal{O}(k^2)] \quad . \tag{2.33}$$

Für thermische Neutronen kann angenommen werden $|a| \simeq 5\,\mathrm{fm}$ und $k \simeq 10^{-4}\,\mathrm{Å}^{-1}$. Für die meisten Nuklide in der Neutronenphysik gilt $\Sigma_a \leq \Sigma_s$ und folglich

$$\frac{a''}{a'} \lesssim ka' \simeq 10^{-4} \qquad \text{und} \qquad ka'' \lesssim (ka')^2 \simeq 10^{-8} \quad . \tag{2.34}$$

Unter diesen Einschränkung können in (2.31) und (2.33) die k-Terme vernachlässigt werden und es ergibt sich der einfache Zusammenhang

$$\Sigma_s = 4\pi|a|^2 \tag{2.35}$$

und

$$\Sigma_a = \frac{4\pi}{k}a'' \quad . \tag{2.36}$$

Selbst für stark absorbierende Kerne sind das gute Näherungen, denn ist $\Sigma_a \simeq 10^4\Sigma_s$ wird $a'' \simeq a'$ und folglich $ka'' \simeq 10^{-4} \leq 1$. Daher sind die Beziehungen (2.35) und (2.36) nach wie vor gute Approximationen für die Wirkungsquerschnitte. Für thermische Neutronen ist nach Gleichung (2.35) der Streuwirkungsquerschnitt unabhängig vom Wellenvektor k. Im Gegensatz dazu ist der Absorptionswirkungsquerschnitt in Gleichung (2.36) indirekt proportional zu k und hat damit eine $1/v$ Abhängigkeit von der Neutronengeschwindigkeit.[20]

2.3 Nachweis von Neutronen

Geladene Teilchen wie Elektronen oder Ionen wechselwirken mit Materie hauptsächlich über die Coulombkraft und können so direkt nachgewiesen werden. Neutronen besitzen keine elektrische Ladung und können daher nicht über das Coulombfeld wechselwirken. Daher ist es kaum möglich, Neutronen direkt nachzuweisen. Neutronen treten mit Atomen über deren Kern in Wechselwirkung und werden entweder von diesem absorbiert oder gestreut. Dies wird zur Detektion von Neutronen genutzt. Bei der Absorption von Neutronen im Kern kommt es zu Kernreaktionen, in deren Folge Teilchen emittiert werden. Die Streuung am Atomkern führt zu Rückstoßkernen, die ihre Bewegungsrichtung und Energie signifikant ändern. Basierend auf einem dieser Konversionsprozesse werden in Neutronendetektoren geladene Sekundärteilchen erzeugt, welche dann direkt nachgewiesen werden können.

$$\text{Neutron} + \text{Targetkern} \rightarrow \begin{cases} \text{Rückstoßkerne} \\ \text{Protonen} \\ \alpha\text{-Teilchen} \\ \text{Spaltfragmente} \end{cases}$$

Die Wahrscheinlichkeit der Wechselwirkung ist stark abhängig von der Neutronenenergie. Thermische Neutronen besitzen eine geringe kinetische Energie, daher sind nur Kernreaktionen und keine Stöße für ihren Nachweis geeignet. Ein wichtiges Kriterium der Kernreaktionen beim Detektieren von Neutronen ist der Q-Wert der Reaktion. Er gibt an, wie viel Energie bei der neutroneninduzierten Reaktion frei wird. Es bietet sich an, Kernreaktionen mit einem hohen Q-Wert zu verwenden. Dies hat zur Folge, dass die Reaktionsprodukte eine hohe kinetische Energie erhalten und sich daher gut nachweisen lassen. Vor allem ist dadurch eine Unterscheidung zu Störsignalen einfacher möglich. Bevorzugte Reaktionen sind (n, α)-, (n, p)- und $(n, Spaltung)$-Reaktionen, da diese geladene Teilchen als Reaktionsprodukte aufweisen. Im Folgenden werden nur Detektoren für langsame Neutronen unterhalb des *cadmium cutoff* von ca. 500 meV beschrieben. In Abbildung 2.2 sind

die Wirkungsquerschnitte der folgenden Reaktionen dargestellt. Die $1/v$-Abhängigkeit, wie sie in Abschnitt 2.2.3 für thermische Neutronen beschrieben wurde, ist sehr deutlich erkennbar.[13]

^{10}B(n, α)-Reaktion

Die am Häufigsten verwendete Reaktion zum Nachweis langsamer Neutronen ist die $^{10}\mathrm{B}(n,\alpha)$-Reaktion. Dabei entsteht ^{7}Li und ein direkt nachweisebares α-Teilchen.

$$ {}^{1}_{n}\mathrm{n} + {}^{10}_{5}\mathrm{B} \rightarrow \underbrace{\begin{cases} {}^{7}_{3}\mathrm{Li} + {}^{4}_{2}\alpha & 2{,}792\,\mathrm{MeV} \\ {}^{7}_{3}\mathrm{Li}^{*} + {}^{4}_{2}\alpha & 2{,}310\,\mathrm{MeV} \end{cases}}_{Q\text{-Wert}} \tag{2.37} $$

Das Lithium kommt nach der Reaktion zu 94% im angeregten Zustand und zu 6% im Grundzustand vor. Die Energie des thermischen Neutrons am Beginn der Reaktion ist im meV-Bereich, daher kann diese im Vergleich zum Q-Wert der Reaktion vernachlässigt werden. Ebenso ist der Impuls des Neutrons sehr klein, dies führt dazu, dass die beiden Reaktionsprodukte in entgegengesetzte Richtung emittiert werden, da der Gesamtimpuls vor der Reaktion quasi Null ist. Die jeweiligen Energien vom Lithium- und α-Teilchen lassen sich über Energie- und Impulserhaltung berechnen.

$$ \begin{aligned} \sqrt{2m_{Li}E_{Li}} &= \sqrt{2m_{\alpha}E_{\alpha}} \\ m_{Li}v_{Li} &= m_{\alpha}v_{\alpha} \end{aligned} \tag{2.38} $$

$$ E_{Li^*} + E_{\alpha} = Q = 2{,}31\,\mathrm{MeV} \tag{2.39} $$

Die Lösung aus Gleichung (2.38) und (2.39) ergibt

$$ E_{Li^*} = 0{,}84\,\mathrm{MeV} \qquad \text{und} \qquad E_{\alpha} = 1{,}47\,\mathrm{MeV} \tag{2.40} $$

für den angeregten Zustand von ^{7}Li. Die häufige Verwendung dieser Reaktion zum Nachweis von Neutronen ist zum einen dem großen Wirkungsquerschnitt von 3840 Barn für thermische Neutronen und zum anderen dem Anteil von ^{10}B mit 19,8 % in natürlichem Bor geschuldtet. Abbildung 2.1 zeigt ein typisches Pulshöhenspektrum eins Detektors mit Bortrifluorid.[13]

^{6}Li(n, α)-Reaktion

Die (n,α)-Reaktion in Lithium ist eine weitere häufig verwendete Kernreaktion zum Nachweis von langsamen Neutronen.

$$ {}^{1}_{n}\mathrm{n} + {}^{6}_{3}\mathrm{Li} \rightarrow \underbrace{{}^{3}_{1}\mathrm{H} + {}^{4}_{2}\alpha \quad 4{,}78\,\mathrm{MeV}}_{Q\text{-Wert}} \tag{2.41} $$

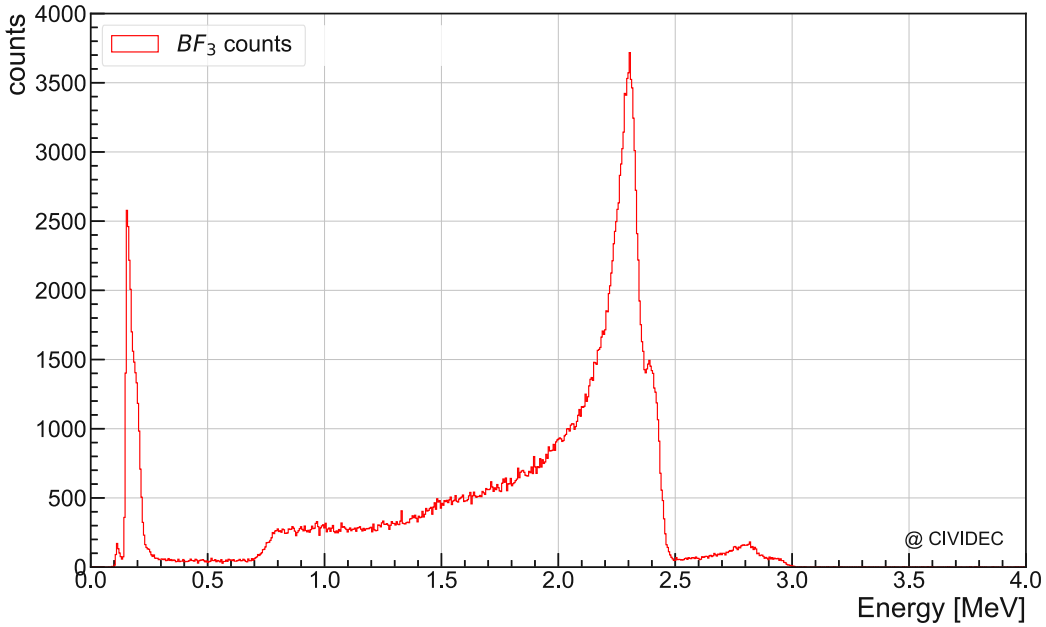


Abbildung 2.1: Pulshöhenspektrum eins BF_3-Zählrohrs aufgenommen am TRIGA Mark-II Reaktor der TU Wien

Dabei kommt das Tritium nach der Reaktion nur im Grundzustand vor. Die Energie nach der Reaktion lässt sich wiederum unter Vernachlässigung der Neutronenenergie berechnen.

$$E_H = 2{,}73\,\text{MeV} \qquad \text{und} \qquad E_\alpha = 2{,}05\,\text{MeV} \tag{2.42}$$

Der Wirkungsquerschnitt für thermische Neutronen beträgt für ^{6}Li 940 Barn, die natürliche Häufigkeit in Lithium liegt bei 7,4 %.[13]

^{3}He(n, p)-Reaktion

Das gasförmige ^{3}He findet ebenso Anwendung bei der Detektion von langsamen Neutronen.

$$^1_n\text{n} + {}^3_2\text{He} \rightarrow {}^3_1\text{H} + {}^1_1\text{p} \quad \underbrace{0{,}764\,\text{MeV}}_{Q\text{-Wert}} \tag{2.43}$$

Bei dieser Reaktion handelt es sich um eine (n, p)-Reaktion, neben dem Tritium entsteht auch ein Proton, welches in weiterer Folge nachgewiesen werden kann. Wird die Energie der langsamen Neutronen erneut vernachlässigt, so kann die Verteilung der Energie auf die Reaktionsprodukte wieder über Energie- und Impulserhaltung berechnet werden.

$$E_H = 0{,}191\,\text{MeV} \qquad \text{und} \qquad E_p = 0{,}573\,\text{MeV} \tag{2.44}$$

Für thermische Neutronen beträgt der Wirkungsquerschnitt dieser Reaktion 5330 Barn, was signifikant höher ist als für die anderen Reaktionen. ^{3}He ist in der Natur sehr selten, dennoch wird es industriell gewonnen und für Neutronendetektoren eingesetzt. Das

knappe Vorkommen führt jedoch zu hohen Preisen und ist für viele Anwendungen ein entscheidender Faktor.[13]

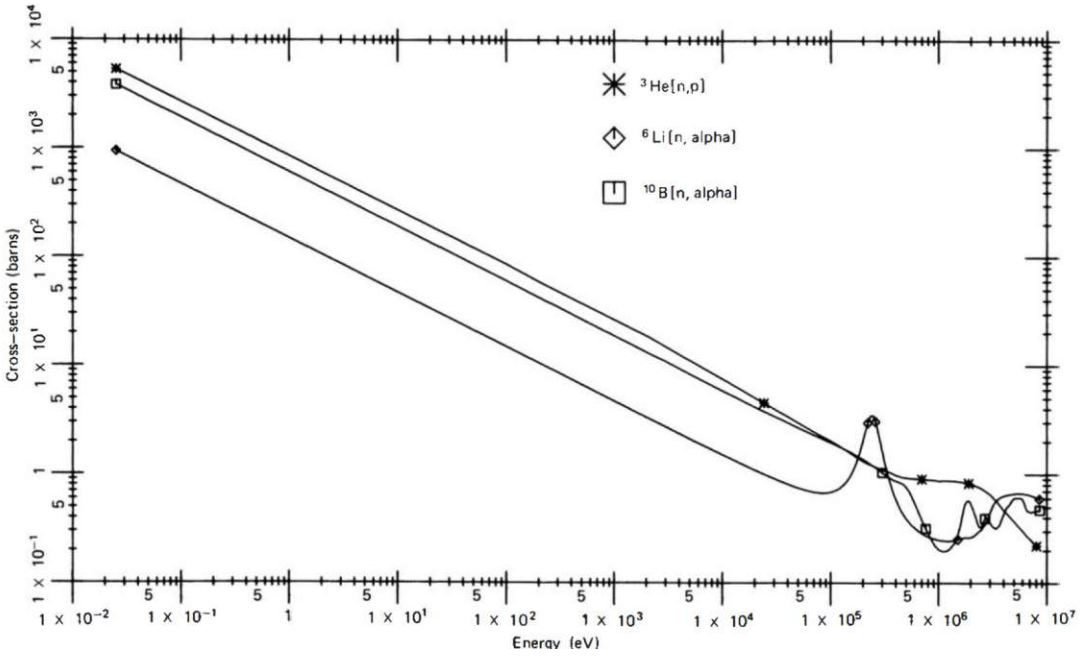


Abbildung 2.2: Absorptionswirkungsquerschnitte der Isotope ^{3}He, ^{6}Li und ^{10}B in Abhängigkeit der Neutronenenergie mit deutlicher $1/v$-Abhängigkeit im thermischen Spektrum [13]

3 Bildgebung mit Neutronen

3.1 Einführung in die Abbildungen mit Neutronen

Im vorigen Kapitel wurden grundlegende Eigenschaften von Neutronen diskutiert. Besonderes Augenmerk lag dabei auf der Wechselwirkung von Neutronen und Materie. Diese Wechselwirkung kann genutzt werden, um Objekte mittels Neutronen abzubilden. Wird ein Neutronenstrahl auf ein Objekt gerichtet, so treten die Neutronen in Abhängigkeit des Wirkungsquerschnitts mit den Atomkernen des Objekts in Wechselwirkung. Der transmittierte Strahl hat nach dem Durchtritt durch das Probenobjekt eine Intensitätsverteilung, welche der Verteilung der Wirkungsquerschnitte, also der Kerne im Objekt und somit der inneren Struktur der Probe entspricht. Die Abbildung mit Neutronen stellt ein zerstörungsfreies Verfahren dar, ähnlich derer mit Röntgenstrahlen. Trotz der Ähnlichkeit zu Abbildungen mit elektromagnetischen Wellen gibt es einen bedeutenden Unterschied. Die Absorption von Neutronen ist eine Eigenschaft des Atomkerns. Selbst Isotope des selben Elements absorbieren Neutronen verschieden stark. Im Gegenatz dazu ist die Wechselwirkung elektromagnetischer Wellen fast ausschließlich auf die Atomhülle beschränkt. Abbildung 3.1 vergleicht die Wirkungsquerschnitte einiger Elemente bzw. Isotope für Röntgenstrahlen und Neutronen. Sehr deutlich zu erkennen ist, dass Röntgenstrahlen mit Wasserstoff und Deuterium gleichermaßen wechselwirken. Beide tragen die selbe Ladung in der Atomhülle. Im Gegensatz dazu ist der Wirkungsquerschnitt im Fall der Neutronen für Wasserstoff und Deuterium verschieden, da Deuterium ein zusätzliches Neutron im Kern trägt. Trotz dieser fundamentalen Unterschiede bei der Wechselwirkung gibt es viele Gemeinsamkeiten, die sowohl den Aufbau von Instrumenten, als auch die Auswertung der Daten betreffen.

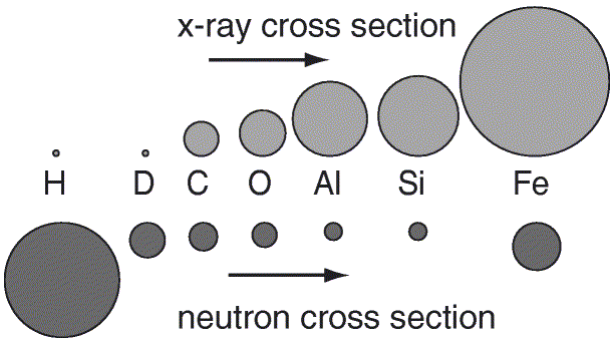


Abbildung 3.1: Vergleich der Absorbtionswirkungsquerschnitte von Röntgenstrahlen mit denen von Neutronen für verschiedene Elemente bzw. Isotope [18]

3.2 Neutronenradiographie

3.2.1 Prinzip einer Radiographieanlage

Radiographische Experimente basieren auf dem einfachen Prinzip der Lochkamera und bestehen aus einer Strahlungsquelle, einem parallelen Strahlbündel, dem zu observierenden Objekt und einem Detektor. Im Falle einer Radiographieanlage mit Neutronen wird der Neutronenstrahl aus der Quelle mittels Kollimator zu einem parallelen Strahlbündel umgeformt. Der so kollimierte Neutronenstrahl trifft auf die Probe und wird durch diese entsprechend verändert. Danach trifft der Strahl auf den Detektor, wo die Neutronen nachgewiesen werden. In modernen Radiographieanlagen kommt ein Szintillationsdetektor zur Anwendung. Dieser besteht aus einem Szintillator, der die Neutronen in Lichtimpulse umwandelt, welche dann von einer Kamera detektiert werden. Abbildung 3.2 zeigt den schematischen Aufbau eines neutronenradiographischen Experiments.

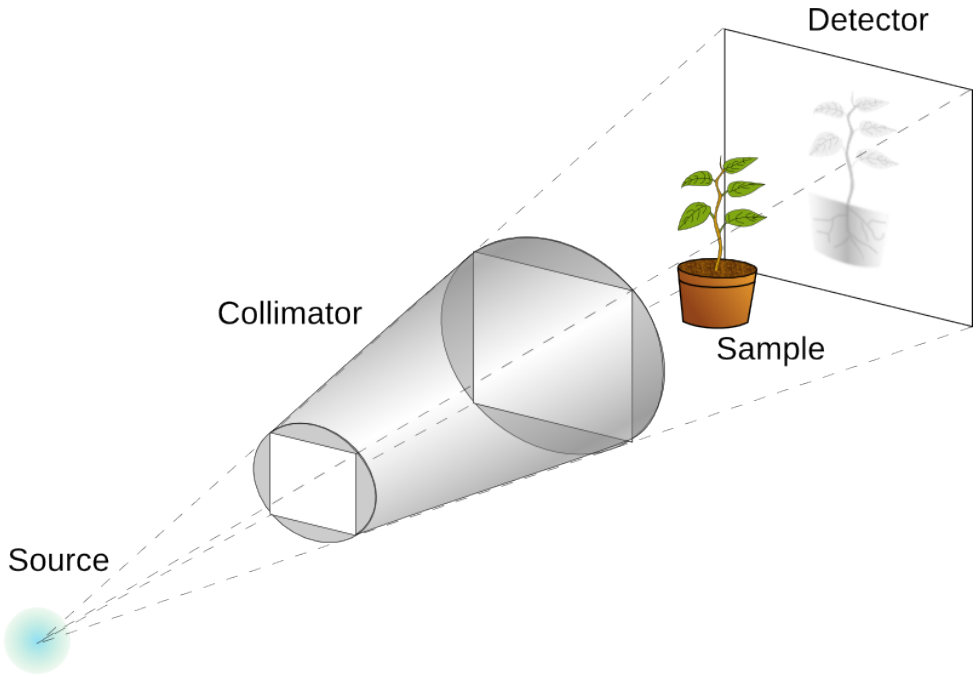


Abbildung 3.2: Schematische Darstellung eines Neutronenradiographieexperiments bestehend aus einer Quelle, einem Kollimator, der Probe und dem Detektor [8]

3.2.2 Kollimator

Die aus einer der Quellen wie in Abschnitt 2.1.3 beschrieben kommenden thermischen Neutronen müssen zu einem nutzbaren Strahl geformt werden. Neutronen werden in der Quelle emittiert und dann im Moderator in zufällige Richtungen gestreut. Neutronen, welche sich in die gewünschte Richtung bewegen, können durch Einführen eines Strahlrohrs in den Moderator selektiert werden. Das führt zu einem definierten Neutronenfluss durch das Strahlrohr in Richtung des Experiments. Für die Neutronenradiographie wird ein möglichst paralleler Neutronenstrahl benötigt. Dies kann erreicht werden, wenn das Strahlrohr

als Kollimator ausgeführt wird. Dazu wird die Wand des Strahlrohrs mit Neutronen absorbierenden Materialien versehen, welche einen großen Absorbtionswirkungsquerschnitt aufweisen (z.B. Bor, Gadolinium oder Cadmium). So kann verhindert werden, dass gestreute Neutronen von außen eindringen. Zudem wird die Kleinwinkelstreuung innerhalb des Kollimators unterdrückt. Die am meisten verbreiteten Kollimatoren sind divergente Kollimatoren bestehend aus einer kleinen Eintrittsöffnung, die zur Quelle gerichtet ist, und einem größeren Austritt, der zu Probe und Detektor zeigt. Diese Geometrie maximiert den Neutronenfluss im Hinblick auf einen ausreichend großen Strahlquerschnitt in der Bildebene. In Abbildung 3.3 ist ein solcher divergenter Kollimator dargestellt. Die Winkelverteilung des kollimierten Neutronenstrahls ist abhängig vom Verhältnis der Länge der Kollimatorröhre (L) und dem Durchmesser der Eintrittsblende (D). Über das sogenannte L/D-Verhältnis wird der divergente Kollimator charaktrisiert. Ziel ist ein möglichst großes L/D-Verhältnis, da es zu einem Strahl mit kleiner Winkelverteilung führt. Um dennoch einen möglicht großen Neutronenfluss am Ende der Apparatur zu erreichen und ein hohes L/D-Verhältnis garantieren zu können, müssen der Eintritt möglichst groß und der Kollimator somit relativ lang sein. Typische Werte für das L/D-Verhältnis sind 150 bis 500.[1]

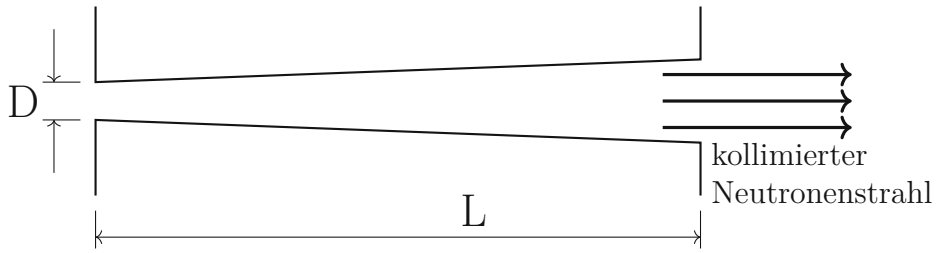


Abbildung 3.3: Darstellung des divergenten Kollimators für Neutronenstrahlen mit seinem charakteristischen L/D-Verhältnis [1]

3.2.3 Szintillationsdetektor

Szintillatoren sind Materialien, welche ionisierende Strahlung absorbieren und Photonen im niedrigen Energiespektrum emittieren. Ein Szintillator für thermische Neutronen benötigt zusätzlich einen Konverter, wie in Abschnitt 2.3 diskutiert, welcher Neutronen einfängt und geladene Teilchen emittiert, die dann ihrerseits im Szintillatormaterial Photonen erzeugen. In der Anwendung werden der Konverter (^{6}Li, ^{10}B oder Gd) sowie das Szintillatormaterial als Pulver gemischt und auf eine Trägerplatte aufgebracht. Zu beachten ist, dass für die Detektion die Szintillatordicke entscheidend ist. Diese muss groß genug sein, um eine entsprechende Konvertierung und Lichtausbeute zu erreichen, mit der Limitierung, dass der Schirm dünn genug ist, um das emittierte Licht nicht selber abzuschirmen. Gerade für die Radiographie und die Abbildung mit Neutronen ist eine räumliche Auflösung des Detektors von Bedeutung, daher gibt es immer einen Kompromiss zwischen der Szintillatordicke und der Detektor-Efficiency. Die am meisten genutzten Szintillatoren, die diese Bedingungen erfüllen, sind ^{6}LiF – ZnS(Ag) und Gd_2O_2S Szintillatoren.

Beim ^{6}LiF – ZnS(Ag) Szintillator werden die entstandenen Landungsträger in Licht mit einer Wellenlänge von 400 bis 550 nm umgewandelt. Es entstehen ca. 10^4 Photonen pro im Szintillator absorbierten Neutron. Durch Verwendung von silberdotiertem Zinksulfid als Konvertermaterial kann eine hohe Lichtausbeute erreicht werden. Die Dotierung mit einem Aktivator wie Silber reduziert den Abstand von Valenz- und Leitungsband und verbessert so die Lichtausbeute.
In den Anfängen der Neutronenradiographie wurden Filme und Analogkameras verwendet, um die Photonen vom Szintillatorschirm zu detektieren. Moderne Setups zeichnen das Licht digital mit CCD- oder CMOS-Kameras auf. Abbildung 3.4 zeigt einen solchen Aufbau. Dabei ist anzumerken, dass die Kamera das Licht vom Szintillator über einen Spiegel aufnimmt, da die Kamera sonst im Strahl positioniert wäre und Neutronen sowie γ-Strahlung von der Quelle den Fotosensor der Kamera zerstören würden.[1]

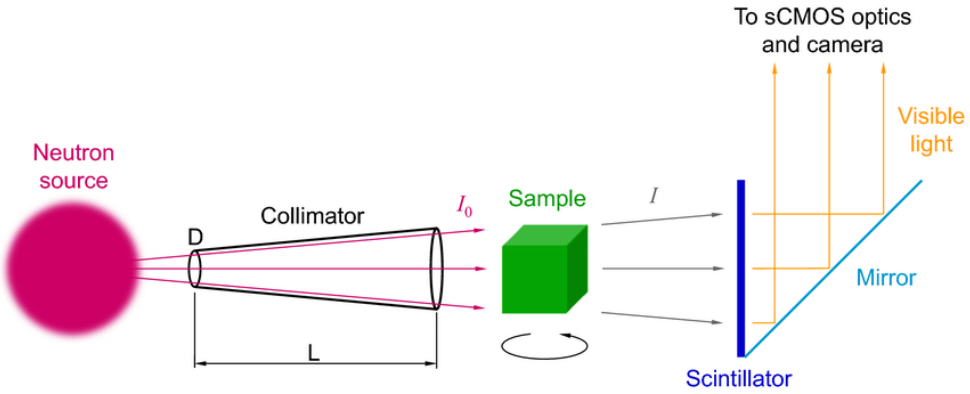


Abbildung 3.4: Neutronenradiographie-Setup bestehend aus Strahlungsquelle, Kollimator, Szintillator und der Kamera, welche die Photonen vom Szintillator über einen Spiegel detektiert [6]

3.3 Mathematische Grundlagen der Radio- und Tomographie

3.3.1 Radiographie

Durchquert Strahlung ein Objekt, so kommt es zur Absorption in der Materie und dadurch zu einem Verlust der Intensität. Der Strahl wird im Medium abgeschwächt. Dies kann durch das Lambert-Beer'sche Gesetz

$$I(x) = I_0 e^{-\mu x} \tag{3.1}$$

beschrieben werden. Dabei ist I_0 die Intensität des einfallenden Strahls, $I(x)$ die Intensität des transmittierten Strahls, μ der Schwächungskoeffizient und x die Dicke des Mediums. Für inhomogene Medien, welche einen vom Ort abhängigen Schwächungskoeffizienten aufweisen, wird der Exponent von Gleichung (3.1) durch ein Integral über den Weg im Medium ersetzt.

$$I(x) = I_0 e^{-\int \mu(x)\,\mathrm{d}x} \tag{3.2}$$

Das Lambert-Beer'sche Gesetz für inhomogene Medien bildet die Grundlage für bildgebende Verfahren, welche Abbildungen durch Transmission erzeugen.
Die Wechselwirkung von Neutronen mit den Atomkernen eines Objekts wird beschrieben durch den Wirkungsquerschnitt σ, welcher die Wahrscheinlichkeit einer Wechselwirkung zwischen Neutron und Kern darstellt. Unter Verwendung der Erkenntnisse aus Abschnitt 2.2 ergibt sich das Lambert-Beer'sche Gesetz für einen Neutronenstrahl durch ein Target-Material der Dicke x.

$$I(x) = I_0 e^{-\Sigma_t x} \tag{3.3}$$

Der totale makroskopische Wirkungsquerschnitt Σ_t ist definiert als

$$\Sigma_t = N\sigma_t = \sigma_t \rho N_A / A \quad , \tag{3.4}$$

wobei N die Anzahl der Atome im Target ist, σ_t der mikroskopische Wirkungsquerschnitt, ρ die Dichte des Targets, N_A die Avogadro-Konstante und A das Atomgewicht. Unter der Annahme eines Objekts bestehend aus verschiedenen Atomen bzw. Isotopen kann das Lambert-Beer'sche Gesetz für Neutronen im inhomogenen Fall angeschrieben werden.

$$I(x) = I_0 e^{-\int \Sigma_t(x)\,\mathrm{d}x} = I_0 e^{-\int N\sigma_t(x)\,\mathrm{d}x} \tag{3.5}$$

Gleichung (3.5) gibt das Prinzip der Neutronenradiographie wieder. Der einfallende Strahl (I_0) wird durch die Probe (Σ_t) verändert und der transmittierte Strahl $I(x)$ wird an einem Konverterschirm sichtbar gemacht.[16]

3.3.2 Tomographie

In Gleichung (3.1) für die Intensität steht auf der rechten Seite ein Integral im Exponenten. Dieses Linienintegral kann als Integral des linearen Schwächungskoeffizienten des Objekts entlang einer Linie aufgefasst werden. Im Kontext der Neutronenradiographie wird das Linienintegral als Schwächung des Neutronenstrahls verstanden, der sich geradlinig durch das Objekt bewegt. Das Objekt wie in Abbildung 3.5 ersichtlich, kann durch die Funktion $f(x,y)$ beschrieben werden und weiter in das Koordinatensystem

$$\begin{aligned} s &= x\cos\theta + y\sin\theta \\ t &= -x\sin\theta + y\cos\theta \end{aligned} \tag{3.6}$$

übergeführt werden. Weiters wird damit das Linienintegral

$$P_\theta(s) = \int_{(s,\theta)line} f(x,y)\,\mathrm{d}t \tag{3.7}$$

definiert. Die Linie (s,θ) ist der Pfad, über den das Integral ausgewertet wird bzw. der Weg des Neutronenstrahls durch das Objekt. Unter Anwendung einer Deltafunktion kann Gleichung (3.7) umgeschrieben werden als

$$P_\theta(s) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\delta(x\cos\theta + y\sin\theta - s)\,\mathrm{d}x\,\mathrm{d}y \quad . \tag{3.8}$$

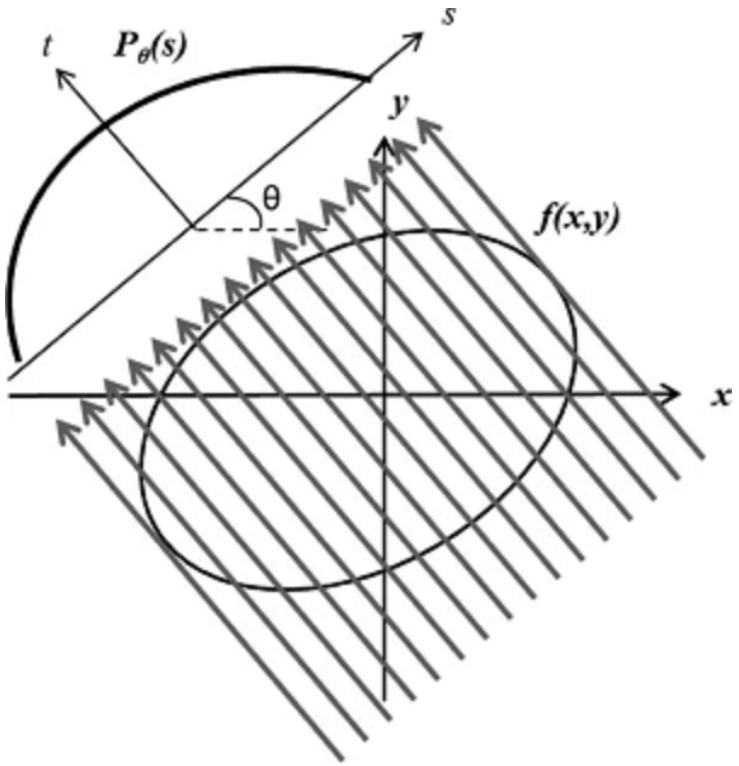


Abbildung 3.5: Veranschaulichung des Linienintegrals $P_\theta(s)$ als Projektion eines Objekts bei paralleler Strahlgeometrie [16]

Die Menge der Linienintegrale $P_\theta(s)$ ergibt zusammen die Projektion des Objekts. Die einfachste Form der Projektion ist eine mit parallelem Strahl, dies wird durch die Menge der Integrale für einen konstanten Winkel θ widergespiegelt. Wird der Winkel des Objekts zum Strahl verändert, lassen sich verschiedene Projektionen erstellen. Dazu kann entweder die Quelle um das Objekt rotiert werden oder, wie für Neutronen üblich, das Objekt im Neutronenstrahl. Sind Projektionen unter verschiedenen Einfallswinkeln vorhanden, kann aus diesen der innere Aufbau des Objekts rekonstruiert werden. Diese Technik wird Tomographie genannt und bei Verwendung eines Neutronenstrahls Neutronentomographie. Um die Querschnittsinformationen aus den Radiographieaufnahmen unter verschiedenen Winkeln zu erhalten, gibt es unterschiedliche mathematische Rekonstruktionsmethoden. Die Grundlage dieser Methoden bildet die Radon-Transformation.
Bezeichnet $\mathbb{R}^2$ den zweidimensionalen euklidischen Raum, so hat dieser eine Punktdarstellung $\bar{x} = (x, y)$ in kartesischen Koordinaten. Aus Abbildung 3.6 kann die Koordinatentransformation zu (s,t) abgeleitet werden, wobei die Achsen parallel zu den Vektoren $\hat{\theta}(\theta)$ und $\hat{\theta}^{\perp}(\theta)$ stehen.

$$\begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.9}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} s \\ t \end{pmatrix} \tag{3.10}$$

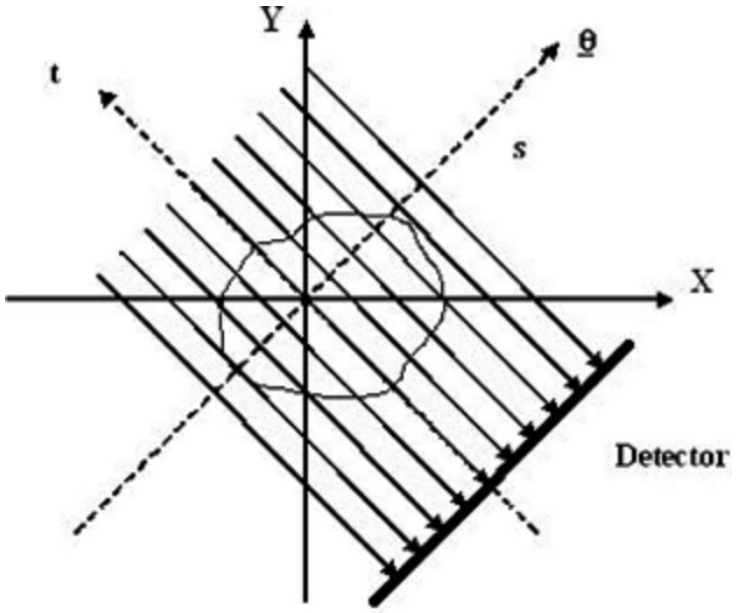


Abbildung 3.6: Koordinatensystem bei Projektion mit paralleler Strahlgeometrie [16]

Eine Funktion $f(x,y)$ im $\mathbb{R}^2$ ist im rotierten Koordinatensystem (s,t), welches gegen den Uhrzeigersinn zu (x,y) um θ verdreht, beschrieben durch

$$f(s,t) = f(x\cos\theta + y\sin\theta, -x\sin\theta + y\cos\theta) \quad . \tag{3.11}$$

Ein Integral einer zweidimensionalen Funktion $f(s,t)$ für alle möglichen geradlinigen Wege wird als zweidimensionale Radon-Transformation von $f(s,t)$ bezeichnet.

$$g(s,\theta) = \int_{\infty}^{\infty} f(s,t)\,\mathrm{d}t \tag{3.12}$$

Die Radon-Transformation in Form von Gleichung (3.8) hat deutliche Ähnlichkeit zur zweidimensionalen Fourier-Transformation.

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-i2\pi(xu+yv)}\,\mathrm{d}x\,\mathrm{d}y \tag{3.13}$$

Der Unterschied der beiden Transformationen besteht darin, dass die Radon-Transformation ein δ-Funktion statt einer Exponentialfunktion beinhaltet. Damit ist die Transformation nach Radon besonders für geradenhafte Strukturen wie Abbildungen mit parallelen Strahlen geeignet. Im Gegensatz dazu hat die Fourier-Transformation einen periodischen Kern und ist daher insbesondere für periodische Strukturen passend. Die Ähnlichkeit dieser beiden Transformationen lässt sich im Zentralschnitt-Theorem nutzen.

Theorem (Zentralschnitt-Theorem)

Die eindimensionale Fourier-Transformation einer Projektion der Funktion f(x,y) ist mit einem zentralen Schnitt durch die zweidimensionale Fourier-Transformierte F(u,v) der Funktion f(x,y) verknüpft.

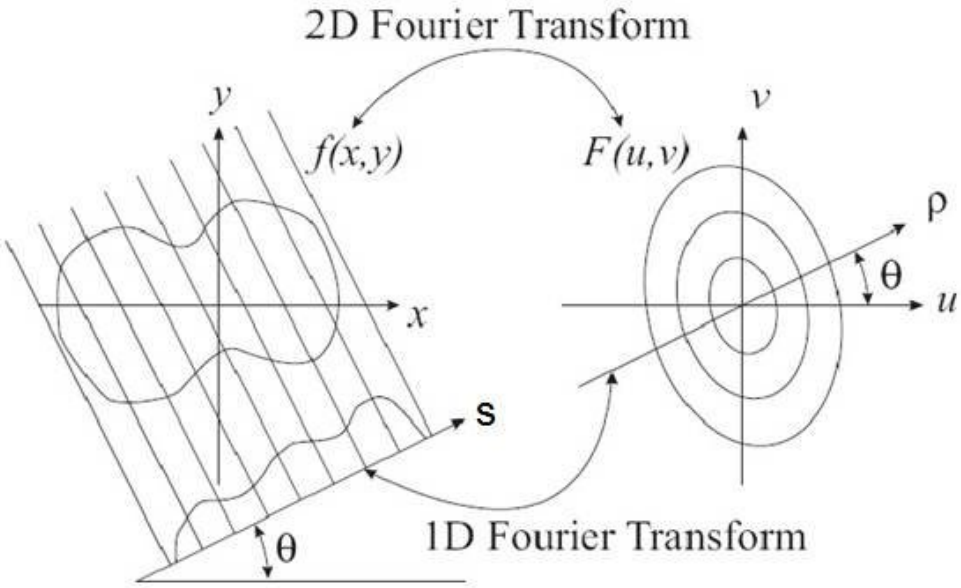


Abbildung 3.7: Schematische Darstellung des Zentralschnitt-Theorems [2]

In Abbildung 3.7 wird das Zentralschnitt-Theorem veranschaulicht. Es wird in Bezug auf tomographische Aufnahmen dazu genutzt, die Radon-Transformation in effizienter Weise unter Verwendung der Fourier-Transformation zu lösen.
Entsprechend dem Zentralschnitt-Theorem bildet sich die eindimensionale Fourier-Transformation von $g(s,\theta)$ aus Gleichung (3.12) zu

$$G(R,\theta) = \int_{-\infty}^{\infty} g(s,\theta) e^{-i2\pi Rs}\,\mathrm{d}s \quad . \tag{3.14}$$

Mit Gleichung (3.8) kann $g(s,\theta)$ ersetzt werden.

$$G(R,\theta) = \int_{-\infty}^{\infty} \left\{ \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y)\delta(x\cos\theta + y\sin\theta - s)dxdy \right\} e^{-i2\pi Rs)}\,\mathrm{d}s \tag{3.15}$$

Ordnet man die Integrale um, ergibt sich daraus

$$G(R,\theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) \left\{ \int_{-\infty}^{\infty} \delta(x\cos\theta + y\sin\theta - s) e^{-i2\pi Rs}\,\mathrm{d}s \right\} \mathrm{d}x\,\mathrm{d}y \quad . \tag{3.16}$$

Wird für das innere Integral nun die Beziehung (3.9) angewandt, so ergibt sich die Fourier-Transformierte von $g(s,\theta)$ zu

$$G(R,\theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) e^{-i2\pi(x\cos\theta + y\sin\theta)R}\,\mathrm{d}x\,\mathrm{d}y \quad . \tag{3.17}$$

Durch die Substitution

$$u = R\cos\theta$$
$$v = R\sin\theta$$

wird aus Gleichung (3.17)

$$G(R,\theta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-i2\pi(xu+yv))}\,\mathrm{d}x\,\mathrm{d}y = F(u,v) \quad . \tag{3.18}$$

Die Beziehung in (3.18) stellt das Zentralschnitt-Theorem dar und zeigt, dass die eindimensionale Fourier-Transformierte $G(R,\theta)$ der Projektion von $f(x,y)$ gleich der zweidimensionalen Fourier-Transformierten $F(u,v)$ von $f(x,y)$ ist.
Wird das Zentralschnitt-Theorem auf die Tomographie angewandt, so folgt $F(u,v)$ direkt aus der Fourier-Transformierten $G(R,\theta)$ der mittels Radiographie aufgenommen Projektionen $g(s,\theta)$. Die Funktion $f(x,y)$, also das Objekt, kann dann durch die inverse Fourier-Transformation aus $F(u,v)$ berechnet werden. Diese direkte Fourier-Rekonstruktion, *Backprojection*, funktioniert aber nur, wenn eine unendliche Zahl an Projektionen vorhanden ist und dadurch alle Punkte in der (u,v)-Ebene bekannt sind. Dies ist der Tatsache geschuldet, dass die Datenpunkte aus dem Zentralschnitt-Theorem auf radialen Linien liegen. Der Fourier-Raum des Objekts ist aber rechtwinkelig. Wie Abbildung 3.8 veranschaulicht, ist eine Interpolation der Daten zur Berechnung notwendig, wenn nur endlich viele Projektionen vorhanden sind. Des Weiteren ist in Abbildung 3.8 ersichtlich, dass die Dichte der radialen Datenpunkte nach außen hin abnimmt. Die hohen Frequenzen im Fourier-Raum sind unterrepräsentiert. Um dem entgegenzuwirken, wird ein Hochpassfilter vor der Rückprojektion angewandt. Das ist die Grundlage der *Filtered Backprojection (FBP)*, sie stellt die meist verbreitetste Methode für tomographische Rekonstruktionen dar. Der Hochpassfilter, wie in Abbildung 3.9 gezeigt, ist nichts anderes als ein Rampenfilter im Fourier-Raum, also eine Gewichtung der Datenpunkte proportional dem Abstand zum Ursprung. Der Algorithmus der *Filtered Backprojection* lässt sich durch die inverse Fourier-Transformation aus Gleichung (3.18) ableiten. Die Objektfunktion $f(x,y)$ lässt sich daher schreiben als

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)e^{i2\pi(ux+vy)}\,\mathrm{d}u\,\mathrm{d}v \quad . \tag{3.19}$$

Unter Verwendung der Substitution

$$\begin{aligned} u &= R\cos\theta \\ v &= R\sin\theta \end{aligned}$$

kann Gleichung (3.19) in Polarkoordinaten übergeführt werden.

$$tf(x,y) = \int_{0}^{2\pi}\int_{0}^{\infty} F(R,\theta)e^{i2\pi R(x\cos\theta+y\sin\theta)}R\,\mathrm{d}R\,\mathrm{d}\theta \tag{3.20}$$

Nun ist $F_\theta(R,\theta)$ die Fourier-Transformierte in Polarkoordinaten. Das Integral über den Winkel kann aufgeteilt werden und es folgt für Gleichung (3.20)

$$\begin{aligned} f(x,y) = &\int_{0}^{\pi}\int_{0}^{\infty} F_\theta(R,\theta)e^{i2\pi R(x\cos\theta+y\sin\theta)}RdRd\theta \\ &+ \int_{0}^{\pi}\int_{0}^{\infty} F_\theta(R,\theta+\pi)e^{i2\pi R(x\cos(\theta+\pi)+y\sin(\theta+\pi))}R\,\mathrm{d}R\,\mathrm{d}\theta \quad . \end{aligned} \tag{3.21}$$

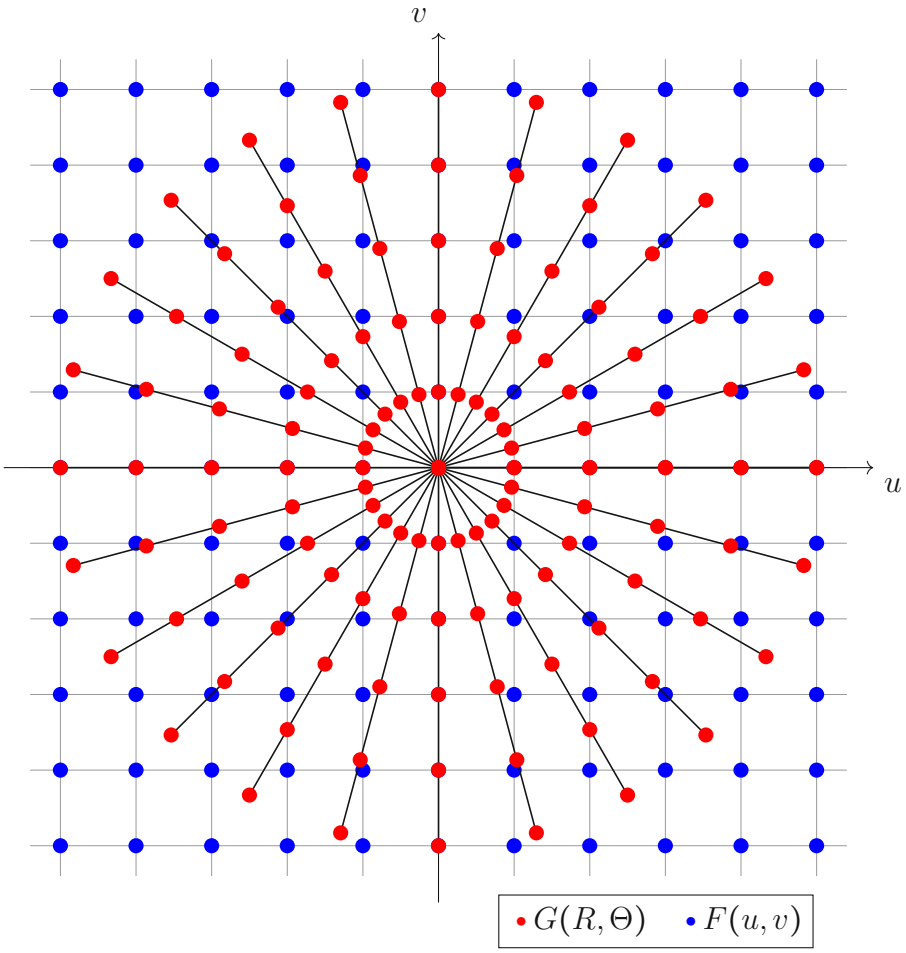


Abbildung 3.8: Datenpunkte von $G(R,\theta)$ und $F(u,v)$ im rechtwinkeligen Koordinatensystem des Fourier-Raums von $f(x,y)$

F_θ ist eine 2π-periodische Funktion und daher gilt

$$F_\theta(R,\theta+\pi) = F_\theta(-R,\theta) \quad .$$

Verwendet man dies und betrachtet das Intervall $0 \le \theta < \pi$ für $-\infty < R < \infty$, wird Gleichung (3.21) zu

$$f(x,y) = \int_0^\pi \left[\int_{-\infty}^{\infty} F_\theta(R,\theta)|R| e^{i2\pi R(x\cos\theta + y\sin\theta)}\,\mathrm{d}R \right] \mathrm{d}\theta \quad . \tag{3.22}$$

Wendet man das Zentralschnitt-Theorems an, ist $F_\theta(R,\theta)$ gleich der eindimensionalen Fourier-Transformierten der Projektion unter dem Winkel θ. Daraus ergibt sich

$$\begin{aligned} f(x,y) &= \int_0^\pi \left[\int_{-\infty}^{\infty} G_\theta(R,\theta)|R| e^{i2\pi R(x\cos\theta + y\sin\theta)}\,\mathrm{d}R \right] \mathrm{d}\theta \\ &= \int_0^\pi \left[\int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} g(s,\theta) e^{-i2\pi R}\,\mathrm{d}s \right] |R| e^{i2\pi R(x\cos\theta + y\sin\theta)}\,\mathrm{d}R \right] \mathrm{d}\theta \quad . \end{aligned} \tag{3.23}$$

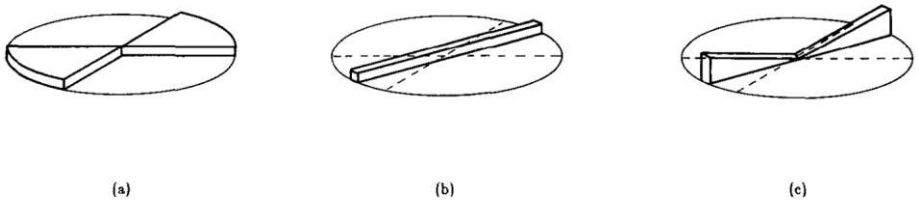


Abbildung 3.9: Die Abbildung zeigt die Daten der Projektion im Fourier-Raum. *(a)* zeigt die ideale Situation für eine perfekte Rekonstruktion. *(b)* ist die tatsächliche Situation ausgehend vom Zentralschnitt-Theorem. *(c)* zeigt die Daten nach dem Rampenfilter, idealerweise sollte das Volumen gleich mit *(a)* sein [11]

Gleichung (3.23) beschreibt das volle Schema der *Filtered Backprojection* mit einem Hochpassfilter. Die Projektionsfunktion $g(s,\theta)$ wird fouriertransformiert ($\int_{-\infty}^{\infty} g(s,\theta)e^{i2\pi Rs}\,\mathrm{d}s$), gefiltert mit dem Filter $|R|$, invers fouriertransformiert ($\int_{-\infty}^{\infty}[\ldots]\ldots e^{i2\pi R(x\cos\theta+y\sin\theta)}\,\mathrm{d}R$) und schließlich rückprojiziert ($\int_0^{\pi}[\ldots]\,\mathrm{d}\theta$).
Ganz allgemein lässt sich das Werkzeug der *Filtered Backprojection* als Faltung der Projektion mit dem Filter verstehen.

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\big[F_\theta(u,v)\times Filter\big]e^{i2\pi(ux+vy)}\,\mathrm{d}u\,\mathrm{d}v \tag{3.24}$$

Neben der *FBP* gibt es noch weitere Verfahren, um Objekte über ihre Projektionen zu rekonstruieren, jedoch stellt die *Filtered Backprojection* die anschaulichste Methode dar.[16]

4 Neutronenradiographie an der TU Wien

4.1 TRIGA Forschungsreaktor

Der TRIGA Mark-II Kernreaktor der TU Wien ist am Atominstitut im 2. Wiener Gemeindebezirk nahe dem Prater angesiedelt. Wie der Name TRIGA (Training Research Isotope Production General Atomics) schon sagt, handelt es sich dabei um einen Forschungsreaktor. Der Reaktor ist ein Swimmingpoolreaktor mit einer maximalen thermischen Dauerleistung von 250 kW_{th}. Das Besondere am TRIGA-Design ist der Brennstoff, dieser besteht aus einer Uran-Zirkon-Hydrid-Mischung. Der Wasserstoff im Zirkon-Hydrid fungiert als Moderator, daher besitzt der Reaktorkern ein sehr starkes negatives Temperatur-Feedback. Der Brennstoff übernimmt bis zu 80% des Moderationsprozesses, lediglich 20% der Moderation finden im Wasser um die Brennelemente statt. Steigt die Leistung im Reaktor an, so erwärmt sich der Brennstoff und somit auch das Zirkon-Hydrid. Dadurch verliert der Wasserstoff seine moderierende Eigenschaft und die Reaktorleistung sinkt, bis die Brennstofftemperatur auf einen Wert fällt, der die moderierende Wirkung des Wasserstoffs wieder einsetzen lässt. Dank diesem starken negativen Temperaturkoeffizienten ist auch ein Impulsbetrieb möglich, bei dem für 40 ms eine Reaktorleistung von 250 MW_{th} erreicht wird. Der Reaktorkern besteht aus ca. 80 Brennelementen und drei Steuerstäben, um die Leistung zu regulieren. Neben der Möglichkeit der Bestrahlung von Proben am oder im Kern gibt es auch noch sechs Experimentierpositionen außerhalb des Reaktors. Vier sind Strahlrohre für Neutronenexperimente und die restlichen zwei stehen für Neutronenradiographie zur Verfügung, namentlich der Trockenbestrahlungsraum und die thermische Säule.

4.2 Thermische Säule und Radiographie

Abbildung 4.2 zeigt die beiden Experimentierpositionen für Neutronenradiographie. Auf der linken Seite im Bild (NRI) ist der Trockenbestrahlungsraum mit dem konischen Kollimator und dem Probenlift dargestellt. Der Kollimator hat ein L/D-Verhältnis von 50, leider ist dieses Verhältnis unzureichend, um scharfe Radiographieaufnahmen mit moderner Technik zu machen. Daher liegt der Fokus auf der zweiten Radiographiestation (NRII), der thermischen Säule, rechts in Abbildung 4.2. Auch hier ist ein konischer Kollimator verbaut, gefolgt von einem zylindrischen. Der Kollimator an der thermischen Säule besitzt ein L/D-Verhältnis von 130 und ist daher besser für radiographische Aufnahmen geeignet. Vor dem Kollimator ist ein Bismutfilter platziert, der die Gammastrahlung vom

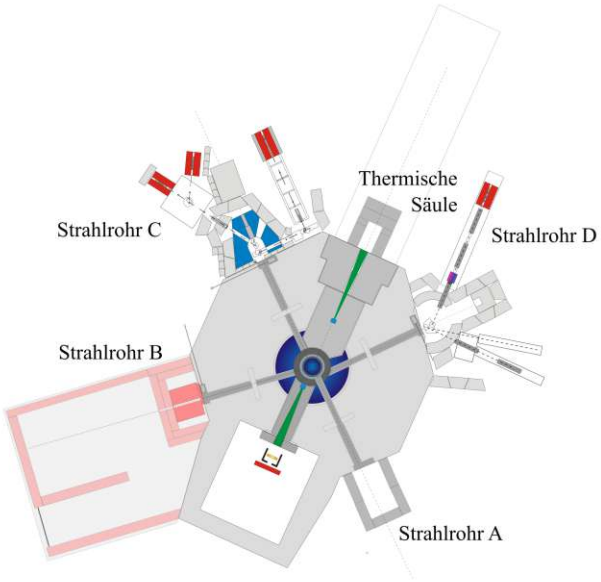


Abbildung 4.1: Die Neutronenstrahlpositionen rund um den TRIGA Mark-II Forschungsreaktor

Reaktorkern abschirmt. Der konische Teil des Kollimators befindet sich in der Schwerbetonabschirmung des Reaktors, der zylindrische Kollimator befindet sich im thermischen Tor, einer auf Schienen geführten beweglichen Schwerbetonabschirmung. Am Ende der Kollimatorstrecke ist ein Shutter angebracht, der zur definierten Belichtung der Probe

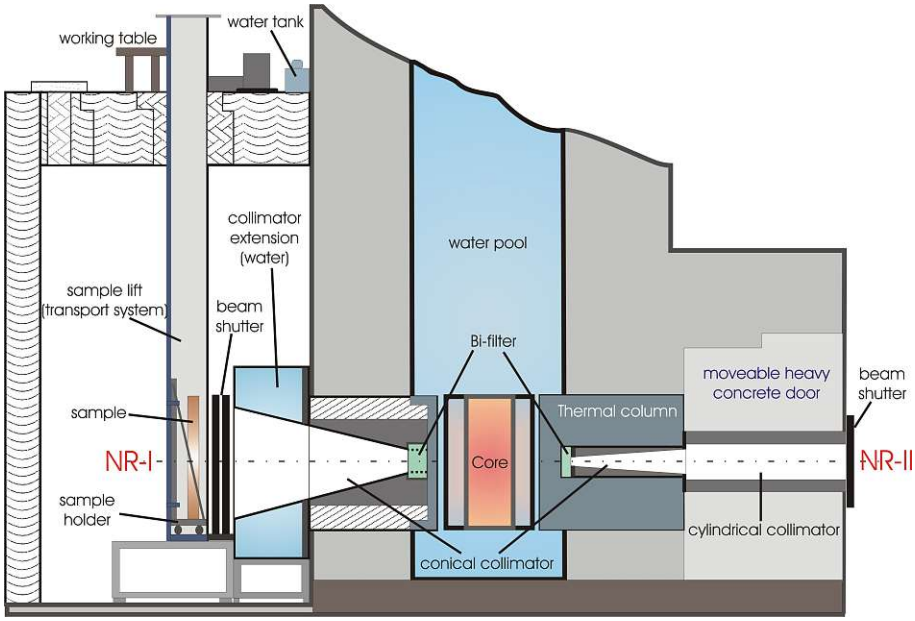


Abbildung 4.2: Details der Neutronenradiographiestationen am TRIGA Mark-II Forschungsreaktor, links der Trockenbestrahlungsraum und rechts die thermische Säule

dient. Der Shutter wird über einen Pneumatikzylinder bewegt und mit einem Elektroventil angesteuert. Bei geöffnetem Shutter beträgt die Flussdichte der thermischen Neutronen an der Probe 10^5 n/cm^{2}s für eine Reaktorleistung von 250 kW. Da der konische Kollimator im Graphitreflektor sitzt, sind kaum epithermische oder schnelle Neutronen vorhanden.

4.3 Aufbau der Neutronenradiographieanlage

Die neue Neutronenradiographie- und -tomographieanlage an der thermischen Säule gliedert sich in vier Hauptkomponenten. Der Probentisch dient dazu, die Probe im Strahl zu platzieren und für Tomographieaufnahmen zu rotieren. Die Steuereinheit stellt die Schnittstelle zwischen dem Computer mit der Messsoftware und dem Probentisch dar. Die Detektoreinheit zum Abbilden der Neutronen besteht aus der Halterung für den Szintillator, dem L-förmigen Gehäuse mit einem Spiegel zur Führung der Photonen und der Kamera. Der vierte Bestandteil ist die Messsoftware. Sie dient als Knotenpunkt zwischen den einzelnen Komponenten. Die Software sorgt dafür, dass Messungen mit der neuen Anlage einfach und unkompliziert möglich sind.

4.4 Probentisch

Der Probentisch hat die Aufgabe, das Probenobjekt im Neutronenstrahl zu bewegen und zu platzieren. Dabei sind Bewegungen über zwei Achsen möglich. Die erste Achse ist horizontal im rechten Winkel zum Neutronenstrahl ausgerichtet. Mit einem Lineartisch ist es daher möglich, die Probe aus dem Strahl zu fahren, um Referenzaufnahmen ohne Objekt zu machen. Entlang der linearen Achse sind zwei Positionen definiert, im Neutronenstrahl und außerhalb des Strahls. Über zwei Endschalter wird die jeweilige Lage definiert.

Abbildung 4.3: Probentisch der neuen Neutronenradiographiestation am TRIGA Mark-II Forschungsreaktor

Bewegt wird der Schlitten des 400 mm langen Lineartisches über eine Spindel, die von einem Schrittmotor angetrieben wird. Der Motor in der Baugröße NEMA-17 ist mit der Spindel über eine Faltenbalgkupplung verbunden. Angesteuert wird der Schrittmotor von der Steuereinheit (siehe Abschnitt 4.5), als Schnittstelle dient ein neunpoliger D-Sub-Stecker (DE-9). Die Steckverbindung führt die vier Pole des Schrittmotors sowie die Si-

gnale der Endschalter, die Versorgungsspannung und die Masse. Platziert ist die Steckverbindung direkt neben dem Schrittmotor, um die Kabellängen kurz zu halten. Abbildung 4.4 zeigt das mit dem 3D-Drucker gefertigte Gehäuse, in dem der D-Sub-Stecker am Probentisch verbaut ist. Mechanisch ist der Schrittmotor über eine Aluminiumkonstruktion mit dem Lineartisch verbunden. Die Konstruktion dient auch als Befestigung der Schleppkette und des Steckergehäuses. Zudem ist am Schlitten eine Adapterplatte für den Drehtisch montiert. Der Adapter dient auch als Kontaktgeber für die Endschalter. In Anhang C.1 sind die Konstruktionszeichnungen der einzelnen Teile aufgelistet. Weil alle Teile so konstruiert sind, dass sie symmetrisch zur Bewegungsachse des Schlittens sind, können die Schleppkette und die Endschalter sowohl links als auch rechts des Lineartisches montiert werden.

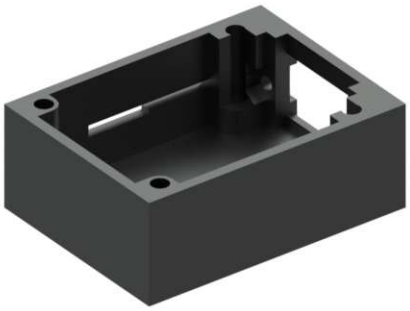


Abbildung 4.4: Gehäuse der D-Sub DE-9 Steckverbindung am Probentisch

Die zweite Bewegungsachse ist die Rotationsachse vertikal durch die Probe. Sie ist notwendig, um Radiographieaufnahmen unter verschiedenen Winkeln für die Tomographie zu machen. Realisiert ist das mit einem Drehtisch, der über die Adapterplatte mit dem Schlitten verbunden ist. Der Drehtisch, Standa 8MR174-11-28, besitzt einen eingebauten Schrittmotor, der über ein Kabel mit einem 15-poligen D-Sub-Stecker (DE-15) angesteuert wird. Das Kabel wird mittels Schleppkette vom beweglichen Schlitten zum nicht beweglichen Teil des Probentisches geführt. Die rotierende Scheibe des Standa 8MR174-11-28 besitzt konzentrische Bohrungen, um die Probe bzw. einen Probenhalter darauf zu befestigen. Die genauen Abmessungen können der Zeichnung in Anhang A.4.3 entnommen werden. In Abbildung A.1 sind alle Einzelteile des Probentisches angeführt, soweit möglich auch mit dem jeweiligen Lieferanten. Für jene Komponenten, die im Haus gefertigt wurden, finden sich die Konstruktionspläne in Anhang C.1.

4.5 Steuereinheit

4.5.1 Konzept

Die Steuerung dient als Schnittstelle zwischen dem Mess-PC und dem Probentisch. Die vom Computer kommenden Befehle werden in der Steuerung verarbeitet und die Schrittmotoren sowie der Shutter entsprechend der Kommandos angesteuert. Realisiert ist dies

mit einem Arduino NANO Every. Dem Arduino werden definierte Befehle über die USB-Schnittstelle und das serielle Protokoll übergeben. Der im Arduino verbaute Mikrocontroller verarbeitet diese Befehle und steuert dann die Schrittmotoren, den Motortreiber oder den Shutter an. Zudem werden die Signale der Endschalter vom Probentisch verarbeitet und dem Messcomputer übergeben.

Abbildung 4.5: Steuerung für den Probentisch der neuen Neutronenradiographiestation

4.5.2 Hardware

Die Hardware der Steuerung besteht aus einem Netzgerät, der Steuerplatine, einer Relaisplatine und diversen anderen Komponenten wie Steckverbindungen, LEDs und Tastern. Das Herzstück der Hardware bildet die Steuerplatine mit dem Arduino und den Motortreibern. Die Steuerplatine ist, wie in Abbildung 4.6 dargestellt, in vier Bereiche gegliedert.

1. Die Eingangsstufe dient zur Entprellung der Signale des Reset-Tasters und jener der Endschalter am Probentisch. Dies ist durch je ein invertierendes Gatter mit Schmitt-Trigger und vorgeschaltetem Tiefpassfilter realisiert. Zusätzlich sind ein Pullup-Widerstand (Reset) bzw. Pulldown-Widerstände (Endschalter) vor dem Filter platziert, um keinen undefinierten Betriebszustand zu riskieren. Die beiden Endschalter sind als Öffner gegen 5 V ausgeführt. Der Reset-Taster ist ein Schließerkontakt gegen GND. Die entprellten Signale der Endschalter werden an die digitalen

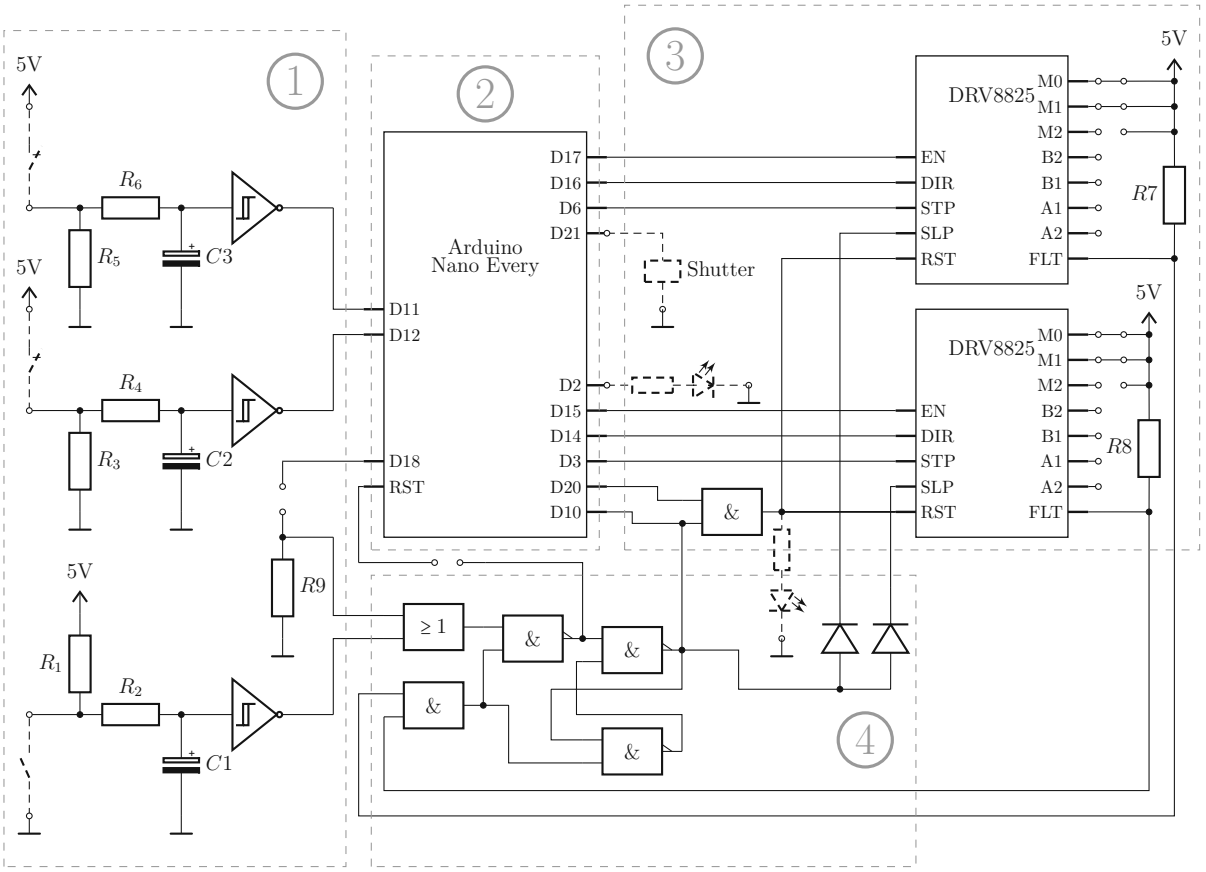


Abbildung 4.6: Gliederung der Steuerplatine in vier Bereiche: 1. Eingangsstufe zur Entprellung, 2. Arduino Nano Every mit Mikrocontroller ATmega4809, 3. Ausgangsstufe mit Motortreiber und externem Relais, 4. Logikeinheit zur Fehlerspeicherung

Eingänge des Arduino weitergegeben. Das Signal des Reset-Tasters geht zum Fehlerspeicher.

2. Das Zentrum der Steuerplatine bildet das Mikrocontroller-Board Arduino Nano Every. Der Arduino ist um einen ATmega4809 Mikrocontroller aufgebaut. Er besitzt eine USB-Schnittstelle zur Energieversorgung und Kommunikation sowie einen Reset-Taster und eine LED. Die am Arduino zur Verfügung stehenden Ports des ATmega4809 und weitere Pegel wie die 5 V der USB-Schnittstelle oder das GND Potential sind auf Lötaugen aufgelegt. Tabelle 4.1 gibt einen Überblick über die Signalbelegung, welche an den Pins für die Steuerung verwendet werden.

3. Die Ausgangsstufe dient zur Ansteuerung der Schrittmotoren und des Shutters. Die Motortreiber vom Typ DRV8825 sind fertige Module, die über Stiftleisten mit der Steuerplatine verbunden sind. Für jeden der beiden Schrittmotoren (Lineartisch und Drehtisch) ist ein Treiberbaustein vorhanden. Diese erhalten vom Arduino je ein individuelles Taktsignal (step) für Geschwindigkeit und Weg so wie je ein Signal für Drehrichtung (direction) und Freigabe (enable). Über Dip-Schalter kann

zusätzlich der Schrittmodus eingestellt werden. Zudem gibt es einen Reset- sowie einen Sleep-Eingang an den DRV8825. Die Sleep-Eingänge der beiden Motortreiber werden über die Fehlerspeicherlogik mittels Dioden bedient. Das Signal am Reset-Eingang kommt auch vom Fehlerspeicher, wird aber mit einem Freigabesignal vom Arduino verknüpft. Weitere Deatils zu den DRV8825 finden sich in Anhang A.4.5. Der Motortreiber ist mit dem Lineartisch über einen D-Sub DE-9 Stecker verbunden. Zur Verbindung zum Drehtisch dient ein D-Sub DE-15 Stecker. Das Relais für den Shutter befindet sich auf einer eigenen Platine und wird über Jumper-Kabel mit der Steuerplatine verbunden.

4. Um die Schrittmotoren und die Motortreiber zu schützen, ist ein Fehlerspeicher vorhanden. Die DRV8825 besitzen Fehlerausgänge, welche logisch 0 ausgeben, sobald es zu Übertemperatur oder Überstrom im Treiber kommt. Die Fehlersignale der beiden Motortreiber werden über ein AND-Gatter zusammengeführt und dann an den SET-Eingang eines NAND-Flip-Flops gelegt. Der invertierende Ausgang des Flip-Flops ist über je eine Diode mit dem Sleep-Eingang der Motortreiber verbunden. Sobald einer der beiden Motortreiber einen Fehler registriert, werden beide DRV8825 über das Flip-Flop gesperrt. Das Rücksetzen des Flip-Flops kann über den entprellten Reset-Taster oder optional über den Arduino erfolgen. Dazu ist der entsprechende Jumper auf der Platine zu setzen. Nach der Oder-Verknüpfung folgt noch ein NAND-Gatter zur Reset-Freigabe, um sicherzustellen, dass kein Rücksetzen des Flip-Flops erfolgt, so lange ein Fehlersignal anliegt.

Die Steuerplatine bietet einige Einstellmöglichkeiten für einen individuellen Gebrauch. Es können zwei Jumper gesetzt werden. Einer, um mit dem Reset-Taster nicht nur den Fehlerspeicher rückzusetzen, sondern auch den Arduino. Der andere Jumper dient dazu, den Fehlerspeicher vom Arduino aus und in weiterer Folge durch die Software zurückzusetzen. Neben den Jumpern können über DIP-Schalter die Microstepping-Modi der Motortreiber gewählt werden. In Zusammenhang mit den Schrittmotoren und der nachfolgend beschriebenen Software sollten die beiden Jumper nicht gesetzt und die Microstep-Auflösung nach Tabelle A.7 mit 1/8 Schritten gewählt werden. In Abbildung B.1 ist die interne Verdrahtung der Steuerung mit allen Komponenten und den Verbindungen dargestellt. Die auf der Steuerplatine zur Verfügung stehenden Anschlüsse sind in Tabelle B.2 aufgelistet. Um einen fehlerlosen Betrieb zu gewährleisten und den Fehlerspeicher und die Motortreiber korrekt nutzen zu können, muss die Strombegrenzung an den DRV8825 richtig eingestellt werden. Dazu wird die Referenzspannung am Potenziometer gemessen und entsprechend der Beziehung

$$U_{ref} = \frac{I_{limit}}{2}$$

variiert.
Der Shutter wird über ein Relais-Modul gesteuert, das sein Signal und die Versorgung von der Steuerplatine erhält. Die Spannung des Arbeitsstromkreises beträgt 230 V. Am Relais wird der Schließer des Wechslerkontakts zum Schalten des Shutters verwendet.

4.5.3 Software

Um die Hardware der Steuerung richtig nutzen zu können, benötigt der Arduino Nano Every eine Software. Dieses Programm, beim Arduino *Sketch* genannt, dient dazu, Befehle vom Messcomputer zu empfangen, entsprechend zu verarbeiten und in weiterer Folge Motortreiber, Shutter und LEDs mit Signalen zu versorgen. Der Arduino Sketch wird in der Programmiersprache C++ mittels der Open-Source-Software Arduino IDE entwickelt und dann auf den Prozessor geladen. Die Arduino IDE bietet zur C++-Bibliothek noch einige Erweiterungen und vereinfachte Funktionen.
Der Sketch für die Steuerung beginnt damit, konstante Werte im Header-Bereich zu definieren. Diese werden beim Kompilieren des Programms direkt an die Stelle der Verweise gesetzt und benötigen damit keinen weiteren Speicherplatz. Im vorliegenden Programm werden so die Ein- und Ausgangspins definiert.

```
#define READY_PIN 20  //controller ready LED signal, Pin 20 of the Arduino
    ↳ Nano Every or pin PD4 on ATmega4809
#define LIN_STEP_PIN 6    //square wave signal for linear movement, Pin 6 of
    ↳ the Arduino Nano Every or pin PF4 on ATmega4809
#define LIN_DIR_PIN  16   //direction signal for linear movement, Pin 16 of
    ↳ the Arduino Nano Every or pin PD1 on ATmega4809
#define LIN_ENBL_PIN  17  //enable signal for linear movement, Pin 17 of the
    ↳ Arduino Nano Every or pin PD0 on ATmega4809
#define ROT_STEP_PIN  3   //square wave signal for rotational movement, Pin
    ↳ 3 of the Arduino Nano Every or pin PF5 on ATmega4809
#define ROT_DIR_PIN  14   //direction signal for rotational movement, Pin 14
    ↳ of the Arduino Nano Every or pin PD3 on ATmega4809
#define ROT_ENBL_PIN  15  //enable signal for rotational movement, Pin 4 of
    ↳ the Arduino Nano Every or pin PD2 on ATmega4809
#define STATUS_LED_PIN 2  //status motor driver LED signal, Pin 2 of the
    ↳ Arduino Nano Every or pin PA0 on ATmega4809
#define SHUTTER_RELAY_PIN 21  //shutter signal, Pin 21 of the Arduino Nano
    ↳ Every or pin PD5 on ATmega4809
#define DRIVER_RESET_PIN 18 //reset driver error memory, Pin 18 of the
    ↳ Arduino Nano Every or pin PA2 on ATmega4809
#define DRIVER_ERROR_PIN 10 //error from driver error memory, Pin 2 of the
    ↳ Arduino Nano Every or pin PB1 on ATmega4809
#define LIMIT_OUT_PIN 11  //signal from limit switch out position, Pin 11 of
    ↳ the Arduino Nano Every or pin PE0 on ATmega4809
#define LIMIT_IN_PIN  12  //signal from limit switch in position, Pin 12 of
    ↳ the Arduino Nano Every or pin PE1 on ATmega4809
```

Quelltext 4.1: Definition der Ein- und Ausgangspins

In Tabelle 4.1 ist die Pinbelegung angeführt. Dabei ist zu beachten dass es verschiedene Bezeichnungen für die jeweiligen Pins gibt. Es wird unterschieden zwischen den Arduino-Pins, den Pins des ATmega4809 und den physischen Anschlusspins der Platine. Innerhalb

Arduino	ATmega	Board	Name	Beschreibung
D20	PD4	10	`READY_PIN`	Bereitschaftsanzeige
D6	PF4	24	`LIN_STEP_PIN`	Taktsignal Lineartisch
D16	PD1	6	`LIN_DIR_PIN`	Richtung Motor Linearbewegung
D17	PD0	7	`LIN_ENBL_PIN`	Freigabe Motor Lineartisch
D3	PF5	21	`ROT_STEP_PIN`	Taktsignal Drehtisch
D14	PD3	4	`ROT_DIR_PIN`	Drehrichtung der Probe
D15	PD2	5	`ROT_ENBL_PIN`	Freigabe Motor Drehtisch
D2	PA0	20	`STATUS_LED_PIN`	Statusanzeige der Treiber
D21	PD5	11	`SHUTTER_RELAY_PIN`	Signal Shutter Relais
D18	PA2	8	`DRIVER_RESET_PIN`	Rücksetzen Fehlerspeicher
D10	PB1	28	`DRIVER_ERROR_PIN`	Fehler Motortreiber
D11	PE0	29	`LIMIT_OUT_PIN`	Endposition Außen
D12	PE1	30	`LIMIT_IN_PIN`	Endposition Innen
RESET	-	18	-	Rücksetzen Arduino

Tabelle 4.1: Pinbelegung Arduino Nano Every und ATmega4809

des Sketches kommt es immer wieder zu unterschiedlicher Benennung des selben Pins. Um Klarheit zu schaffen, wird im jeweiligen Kommentar angemerkt, um welchen Pin es sich handelt.
Anschließend an die Definition der Konstanten werden die globalen Variablen deklariert. Diese stehen für alle Funktionen global zur Verfügung. Die Variable `rotStepCount` zählt die Takte der Rotationsbewegung und `rotStepSetpoint` ist der zugehörige Sollwert, der die Anzahl der benötigten Takte vorgibt.

```
volatile int rot_step_count = 0;  //step counter variable to count the steps ↙
    ↳ of the clock signal
volatile int rot_step_setpoint = 0; //set point variable for the total ↙
    ↳ number of steps that are required
```

Quelltext 4.2: Deklarieren der globalen Variablen

Bei der Programmierung von Arduino-Boards gibt es eine weitere Besonderheit, die beiden verpflichtenden Funktionen `setup` und `loop`. Die `setup`-Funktion wird bei jedem Programmstart einmal aufgerufen und erledigt Initialisierungsaufgaben. Im Fall der Steuerungssoftware werden in der `setup`-Funktion weitere Funktionen aufgerufen. Die Ein- und Ausgangspins werden initialisiert, die serielle Verbindung geöffnet und die Timer und Interrupts eingerichtet. Anschließend werden eine Nachricht ausgegeben, dass die Steuerung bereit ist, sowie die beiden LED-Ausgänge für den Status der Treiber und die Betriebsanzeige der Steuerung gesetzt.

```
void setup()
{
  pinSetup(); //call function to set up I/O pins
  serialSetup();  //call function to set up serial connection
  timerInterruptSetup();  //call function to set up timers and interrupts
  Serial.println("controller_ready"); //report "controller_ready" via serial ↙
    ↳ port
  digitalWrite(READY_PIN, HIGH); //set controller ready LED signal high
  digitalWrite(STATUS_LED_PIN, HIGH);  //set status motor driver LED signal ↙
    ↳ high
}
```

Quelltext 4.3: `setup`-Funktion

Bei der Initialisierung der Pins werden zuerst die Ausgangspins definiert und dann die kritischen Pins auf einen definierten Wert gesetzt. Anschließend findet die Initialisierung der Eingangspins statt. Bei diesen wird der interne Pullup-Widerstand aktiviert.

```
void pinSetup()
{
  /* declare pins as output */
  pinMode(LIN_STEP_PIN, OUTPUT);
  pinMode(LIN_DIR_PIN, OUTPUT);
  pinMode(LIN_ENBL_PIN, OUTPUT);
  pinMode(ROT_STEP_PIN, OUTPUT);
  pinMode(ROT_DIR_PIN, OUTPUT);
  pinMode(ROT_ENBL_PIN, OUTPUT);
  pinMode(STATUS_LED_PIN, OUTPUT);
  pinMode(SHUTTER_RELAY_PIN, OUTPUT);
  pinMode(READY_PIN, OUTPUT);
  /* set initial value to output pins */
  digitalWrite(READY_PIN, LOW);
  digitalWrite(LIN_ENBL_PIN, HIGH);
  digitalWrite(ROT_ENBL_PIN, HIGH);
  digitalWrite(STATUS_LED_PIN, LOW);
  digitalWrite(SHUTTER_RELAY_PIN, HIGH);
  /* declare pins as input and enable internal pullup resistor*/
  pinMode(LIMIT_OUT_PIN, INPUT_PULLUP);
  pinMode(LIMIT_IN_PIN, INPUT_PULLUP);
  pinMode(DRIVER_ERROR_PIN, INPUT_PULLUP);
}
```

Quelltext 4.4: Initialisieren der Ein- und Ausgangspins

Nach dem Initialisieren der Ein- und Ausgangspins wird die serielle Verbindung gestartet,

um mit dem Messcomputer über die USB-Schnittstelle kommunizieren zu können. Dabei wird eine Datenrate von 9600 Bd verwendet.

```
void serialSetup()
{
  Serial.begin(9600);
}
```

Quelltext 4.5: Starten der seriellen Verbindung

Ist die Datenübertragung gestartet, werden die Timer zur Erzeugung der Taktsignale sowie die Hardwareinterrupts initialisiert. Begonnen wird damit, das *clear interrupt global enable flag bit* (`cli()`) zu setzen. Damit werden alle Interrupts deaktiviert und können verändert werden, ohne Gefahr zu laufen, während der Änderung aufgerufen zu werden. Jetzt können die einzelnen Timer und Hardwareinterrupts initialisiert werden. Nach dem Initialisieren wird das *set interrupt global enable flag bit* (`sei()`) gesetzt und die Interrupts dadurch wieder aktiv.

```
void timerInterruptSetup()
{
  cli();  //disable global interrupt
  initTCA0(); //call function to initialize timer/counter A0
  initTCB0(); //call function to initialize timer/counter B0
  initTCB1(); //call function to initialize timer/counter B1
  initRTC();  //call function to initialize Real Timer Counter
  initPORTEINT(); //call function to initialize interrupt on Port E
  initPORTBINT(); //call function to initialize interrupt on Port B
  sei();  //enable global interrupt
}
```

Quelltext 4.6: Einrichten der Timer und Interrupts

Ist die `setup`-Funktion fertig abgearbeitet, kommt die zweite Arduino-spezifische Funktion ins Spiel, die `loop`-Funktion. Diese arbeitet als Endlosschleife und wird nach jedem Durchlauf erneut aufgerufen. Die `loop`-Funktion kann dadurch auf Änderungen am Arduino-Board reagieren und diese verarbeiten. Im Zusammenhang mit der Steuerung wird mit der `loop`-Funktion fortlaufend abgefragt, ob Daten vom Messcomputer am seriellen Port zur Verfügung stehen. Ist das der Fall, werden die Daten bis zur Escape-Sequenz, dem Zeilenvorschub `\n`, gelesen.

```
void loop()
{
  /* check if data is available on serial port */
  if (Serial.available() > 0)
```

```
  {
    handleInputString(Serial.readStringUntil('\n'));  //call function to ↙
    ↳ handle input string with string from serial port as argument
  }
}
```

Quelltext 4.7: `loop`-Funktion

Nach dem erfolgreichen Einlesen der Daten wird der Eingabe-String direkt an den Eingabehandler weitergegeben. Die übergebene Stringvariable wird im Parameter `command` abgelegt. Nach dem Deklarieren der zwei lokalen Variablen `index` und `steps` wird der String nach einem +-Symbol durchsucht und gegebenenfalls bei diesem getrennt. Der erste Teil vor dem + bleibt dann im Parameter `command`, der zweite Teil wird an die Variable `steps` übergeben. Dadurch wird der Befehl für die Rotationsbewegung vorbereitet. Die Befehle, welche von der Steuerung über die serielle Verbindung verarbeitet werden können, sind in Tabelle 4.2 aufgelistet. Darin wird auch klar, warum eine Trennung bei vorhandenem +-Symbol notwendig ist. Sobald die Eingabedaten aufbereitet sind, werden diese mit den vordefinierten Schlüsselwörtern aus Tabelle 4.2 verglichen und die zugehörige Funktion samt Parameter aufgerufen.

```
void handleInputString(String input_command)
{
  int delimiter_index;  //position of the delimiter
  int requested_steps;  //number of steps passed
  input_command.trim(); //removing all white spaces from input string
  delimiter_index = input_command.indexOf("+"); //find position of delimiter
  /* check if delimiter is in input string */
  if(delimiter_index>0)
  {
    requested_steps = input_command.substring(delimiter_index+1).toInt(); ↙
    ↳ //separate steps from input string and convert to integer
    input_command = input_command.substring(0,delimiter_index); //keep ↙
    ↳ remaining input string after separating steps information
  }
  /* check if input string equals keyword*/
  if(input_command.equals("connect"))
  {
    Serial.println("connected");  //report "connected" via serial port
  }
  else if(input_command.equals("move_in"))
  {
    startLin(LOW);  //call function to start moving in
  }
  else if(input_command.equals("move_out"))
  {
    startLin(HIGH); //call function to start moving out
```

```
  }
    if(input_command.equals("stop_lin"))
    {
    stopLin();  //call function to stop linear motion
  }
  else if(input_command.equals("rot_cw"))
  {
    startRot(HIGH, requested_steps);  //call function to start rotation ↙
    ↳ clockwise
    }
  else if(input_command.equals("rot_ccw"))
  {
    startRot(LOW, requested_steps); //call function to start rotation ↙
    ↳ counterclockwise
  }
  else if(input_command.equals("stop_rot"))
  {
    stopRot();  //call function to stop rotational motion
  }
  else if(input_command.equals("open_shutter"))
  {
    startShutter(LOW); //call function to open shutter
  }
  else if(input_command.equals("close_shutter"))
  {
    startShutter(HIGH);  //call function to close shutter
  }
  else if(input_command.equals("stop_all"))
  {
    stopAll();  //call function to stop all motion
  }
}
```

Quelltext 4.8: Handler zum Verarbeiten des Eingabe-Strings

Zur Steuerung der Bewegungen an Probentisch und Shutter sind eine Vielzahl an Funktionen definiert, die vom Eingabehandler aufgerufen werden. Die erste Gruppe dieser Funktionen bildet die Ansteuerung des Lineartisches. Wird der Lineartisch angesteuert und seine Bewegung gestartet, so muss zuerst kontrolliert werden, ob sich der Probentisch nicht schon in der Endposition befindet. Ist dies nicht der Fall, wird eine Nachricht an den seriellen Port geschrieben und mit der Ansteuerung des Motortreibers begonnen. Dazu wird die Bewegungsrichtung entsprechend dem übergebenen Funktionsparameter gesetzt und der Motortreiber freigegeben. Durch Starten der beiden Timer `TCB0` und `TCA0` werden die Taktsignale für den Treiber und die Statusanzeige erzeugt.

Befehle	Beschreibung
`connect`	Überprüft, ob die Verbindung vorhanden ist
`move_in`	Bewegt die Probe in die In-Position
`move_out`	Bewegt die Probe in die Out-Position
`stop_lin`	Stoppt die Linearbewegung
`rot_cw+XXXX`	Rotiert die Probe um XXXX Schritte im Uhrzeigersinn
`rot_ccw+XXXX`	Rotiert die Probe um XXXX Schritte gegen den Uhrzeigersinn
`stop_rot`	Stoppt die Rotationsbewegung
`open_shutter`	Öffnet den Shutter
`close_shutter`	Schließt den Shutter
`stop_all`	Stoppt alle Bewegungen

Tabelle 4.2: Eingabebefehle für die Steuerungssoftware

```
void startLin(bool direction)
{
  /* check which direction is requested and if it is possible */
  if(!checkPosition(LIMIT_OUT_PIN) && direction){
    Serial.println("moving_out"); //report "moving_out" via serial port
  }
  else if(!checkPosition(LIMIT_IN_PIN) && !direction){
    Serial.println("moving_in"); //report "moving_in" via serial port
  }
  else{
    return;
  }
  digitalWrite(LIN_DIR_PIN, direction); //set direction of driver
  digitalWrite(LIN_ENBL_PIN, LOW); //enable driver
  startTCB0();  //call function to start step signal
  startTCA0();  //call function to start LED blinking
}
```

Quelltext 4.9: Start der Linearbewegung

Die Abfrage, ob sich der Schlitten des Lineartisches in einer der Endpositionen befindet, erfolgt über eine eigene Funktion. Dieser wird der Pin des Endschalters übergeben, um die entsprechende Information zu erhalten. Ist einer der Endschalter betätigt, so wird eine Benachrichtigung am seriellen Port ausgegeben und der Wert `true` zurückgegeben. Ist der Endschalter der abgefragten Position nicht gedrückt, so ist der Rückgabewert `false`.

```
bool checkPosition(int pin)
{
  /* check if requested limitation switch is pressed */
  if(digitalRead(pin)){
```

```
    /* check if outer position is requested*/
    if(pin == LIMIT_OUT_PIN){
      Serial.println("pos_out"); //report "pos_out" via serial port
    }
    /* check if inner position is requested*/
    if(pin == LIMIT_IN_PIN){
      Serial.println("pos_in"); //report "pos_in" via serial port
    }
    return true;  //if limitation switch is pressed, return true
  }else{
      return false; //if limitation switch is not pressed, return false
  }
}
```

Quelltext 4.10: Überprüfen der Position des Lineartisches

Der sich in Bewegung befindliche Schlitten kann über eine eigene Stopp-Funktion angehalten werden. Dazu werden der Motortreiber deaktiviert, die Timer für die Taktsignale gestoppt und die Meldung über den erfolgten Stopp an die serielle Schnittstelle übergeben.

```
void stopLin()
{
  digitalWrite(LIN_ENBL_PIN, HIGH); //disable driver
  stopTCB0(); //call function to stop step signal
  stopTCA0(); //call function to stop LED blinking
  Serial.println("lin_stopped");
}
```

Quelltext 4.11: Stopp der Linearbewegung

Der Stopp kann entweder über einen Befehl vom seriellen Port und dem Eingabehandler erfolgen, oder über das Betätigen der Endschalter. Wird einer der Endschalter gedrückt, werden über ein Interrupt die Funktion `posReached()` aufgerufen, die Bewegung gestoppt und die Position abgefragt.

```
void posReached()
{
  stopLin();  //call function to stop linear motion
  checkPosition(LIMIT_IN_PIN);  //call function to check if inner position ↙
    ↳ was reached
  checkPosition(LIMIT_OUT_PIN); //call function to check if outer position ↙
    ↳ was reached
}
```

Quelltext 4.12: Endposition erreicht

Die Bewegung des Rotationstisches wird über eine ähnliche Gruppe an Funktionen gesteuert wie der Lineartisch. So wird die Rotation über eine Funktion mit Richtungsparameter gestartet. Zusätzlich gibt es aber auch einen Sollwertparameter für die Motorschritte. Die Rotationsbewegung wird nicht durch eine Endposition begrenzt, sondern durch einen geforderten Drehwinkel. Der Winkel, um den gedreht wird, steht im direkten Zusammenhang mit der Anzahl der Schritte, um die sich der Motor weiter bewegt. Die Start-Funktion der Rotation beginnt damit, die beiden global definierten Variablen `rotStepCount` und `rotStepSetpoint` zu überschreiben, um das Zählen der Schritte vorzubereiten. Nach der Ausgabe der Nachricht zur bevorstehenden Drehung werden die Richtung entsprechend des Richtungsparameters gesetzt und der Motortreiber freigegeben. Mit dem Start der Timer `TCB1` und `TCA0` werden die beiden Taktsignale für den Treiber und die Statusanzeige erzeugt.

```
void startRot(bool direction, int set_point)
{
  rot_step_setpoint = set_point; //set global setpoint variable to required ↙
    ↳ value
  rot_step_count=0; //set global count variable to 0
  Serial.println((String)"start_rot_"+rot_step_setpoint); //report ↙
    ↳ "start_rot_XXXX" via serial port
   (ROT_DIR_PIN, direction); //set direction of driver
  digitalWrite(ROT_ENBL_PIN, LOW); //enable driver
  startTCB1();  //call function to start step signal
  startTCA0(); //call function to start LED blinking
}
```

Quelltext 4.13: Start der Rotationsbewegung

Bei jedem Takt des Schrittsignals wird über ein Interrupt eine Zählfunktion aufgerufen. Dieser Aufruf führt zum Inkrementieren der globalen Variable `rotStepCount`. Erreicht die Zählvariable den Sollwert, wird die Rotationsbewegung gestoppt.

```
void stepCounter()
{
  rot_step_count++; //increment count by 1
  /* check if counter has reached setpoint  */
  if(rot_step_count>=rot_step_setpoint){
    stopRot(); //stop rotation when setpoint variable is reached
  }
}
```

Quelltext 4.14: Zählen der Takte des Schrittsignals

Neben dem Erreichen des Sollwerts kann die Rotation auch über einen Befehl am seriellen

Port gestoppt werden. In Analogie zum Beenden der Linearbewegung werden auch hier der Motortreiber deaktiviert, die Timer für die Taktsignale gestoppt und die Meldung über den erfolgten Stopp an die serielle Schnittstelle übergeben.

```
void stopRot()
{
  digitalWrite(ROT_ENBL_PIN, HIGH);  //disable driver
  stopTCB1(); //call function to stop step signal
  stopTCA0();  //call function to stop LED blinking
  Serial.println("rot_stopped"); //report "rot_stopped" via serial port
}
```

Quelltext 4.15: Stopp der Rotationsbewegung

Der Shutter wird ebenso über ein Set an Funktionen angesprochen. So wird die Bewegung wieder über eine Funktion mit Richtungsparameter gestartet. Dabei werden richtungsspezifisch eine Nachricht an den seriellen Port geschrieben und dann der Ausgabepin des Shutters mit dem Richtungsparameter gesetzt. Darauf folgt der Start der Timer für die Shutterbewegung und das Taktsignal für die Statusanzeige.

```
void startShutter(bool direction)
{
  /* check which direction is requested */
  if (direction){
    Serial.println("start_close_shutter"); //report "start_close_shutter" ↙
    ↳ via serial port
  }
  else{
    Serial.println("start_open_shutter"); //report "start_open_shutter" via ↙
    ↳ serial port
  }
  digitalWrite(SHUTTER_RELAY_PIN, direction); //set the shutter output
  startRTC(); //call function to start shutter timer
  startTCA0(); //call function to start LED blinking
}
```

Quelltext 4.16: Öffnen oder Schließen des Shutters

Da der Shutter keine Rückmeldung für die Lage besitzt, wird nach Ablauf der im Shuttertimer definierten Zeit über ein Interrupt die Stopp-Funktion aufgerufen. Nach dem Aufruf wird der Timer für die Statusanzeige beendet und in Abhängigkeit vom Signal am Pin die jeweilige Benachrichtigung an die serielle Schnittstelle übergeben.

```
void stopShutter()
{
  stopTCA0();  //call function to stop LED blinking
  /* check which position was required */
  if(digitalRead(SHUTTER_RELAY_PIN)){
    Serial.println("shutter_closed"); //report "shutter_closed" via serial port
  }
  else{
    Serial.println("shutter_opened"); //report "shutter_opened" via serial port
  }
}
```

Quelltext 4.17: Shutter in Endposition

Für den Fall, dass es notwendig ist, alle Bewegungen zu stoppen, gibt es eine eigene Funktion, welche die Linear-Rotationsbewegung anhält und den Shutter schließt. Über die serielle Verbindung wird anschließend benachrichtigt, dass alle Bewegungen gestoppt wurden.

```
void stopAll()
{
  stopLin();  //call function to stop linear motion
  stopRot();  //call function to stop rotational motion
  startShutter(HIGH);  //call function to close shutter
  Serial.println("all_stopped"); //report "all_stopped" via serial port
}
```

Quelltext 4.18: Stoppen aller Bewegungen

Sollte ein Fehler der Motortreiber auftreten, wird über `DRIVER_ERROR_PIN` und den damit verbundenen Interrupt die `driverError()`-Funktion aufgerufen. Es werden alle Bewegungen gestoppt und eine Benachrichtigung über den aufgetretenen Fehler weitergegeben.

```
void driverError()
{
  stopAll(); //call function to stop all motion
  Serial.println("driver_error"); //report "driver_error" via serial port
}
```

Quelltext 4.19: Störungsbehandlung bei Motortreiberfehler

Einige der oben angeführten Funktionen geben Nachrichten über die serielle Schnittstelle an den Messcomputer weiter. In Tabelle 4.3 sind alle Rückgabemeldungen der Steuerungssoftware aufgelistet. Diese Meldungen können in Kombination mit den Befehlen aus

Rückgabemeldung	Beschreibung
`controller_ready`	Steuerung ist bereit
`connected`	Verbindung ist vorhanden
`moving_out`	Beginne Linearbewegung in Richtung Out-Position
`moving_in`	Beginne Linearbewegung in Richtung In-Position
`lin_stopped`	Linearbewegung gestoppt
`pos_out`	Out-Position erreicht
`pos_in`	In-Position erreicht
`start_rot_XXXX`	Beginne Rotationsbewegung um XXXX Schritte
`rot_stopped`	Rotationsbewegung gestoppt
`start_close_shutter`	Beginne Shutter zu schließen
`start_open_shutter`	Beginne Shutter zu öffnen
`shutter_closed`	Shutter geschlossen
`shutter_opened`	Shutter geöffnet
`driver_error`	Motortreiberfehler

Tabelle 4.3: Rückgabemeldungen der Steuerungssoftware

Tabelle 4.2 verwendet werden, um eine sequenzielle Ansteuerung mithilfe des Messcomputers zu implementieren.
Die bereits oben erwähnten Taktsignale für die Motortreiber, den Shuttter und die Statusanzeige werden über Timer erzeugt. Bei diesen Timern handelt es sich um die Hardwaretimer des ATmega4809. Sie werden direkt durch Bitmanipulation an den jeweiligen Registern im Mikroprozessor gesteuert. Im vorliegenden Sketch werden vier verschiedene Timer verwendet. Timer/Counter A0 für die Statusanzeige, Timer/Counter B0 für die Linearbewegung, Timer/Counter B1 für die Rotation und der Real-Time-Counter für die Shutter-Verschlusszeit.
Der Timer/Counter A0, kurz TCA0, ist ein 16 bit-Timer des ATmega4809. Er wird verwendet, um das Taktsignal für die Statusanzeige zu erzeugen. Um den TCA0 nutzen zu können, muss er initialisiert werden. Das geschieht, wie oben beschrieben, durch Aufrufe in der `setup`-Funktion. In einem ersten Schritt wird über den Portmultiplexer der Port A des ATmega4809 als Ausgabe gewählt. Als nächstes werden der Teiler für den Zählvorgang auf ein 1024stel des Prozessortakts festgelegt und der *frequency mode* des Timers gewählt. Zuletzt wird der Vergleichswert des Timers auf den Hexadezimalwert 0x0A00 gesetzt, das entspricht einem Dezimalwert von 2560. Wird der Timer nun gestartet, beginnt er bei jedem 1024sten Prozessortakt nach oben zu zählen. Erreicht der Zähler den Vergleichswert, wird der Ausgang invertiert und wechselt von einem Pegel zum anderen. Dadurch entsteht ein Rechtecksignal, dessen Frequenz sich durch den Teiler und den Vergleichswert definieren lässt. Im Fall des TCA0 ist die Frequenz des Signals auf ca. 3 Hz festgelegt.

```
void initTCA0()
{
  PORTMUX.TCAROUTEA = PORTMUX_TCA0_PORTA_gc;  //select port A as output
  TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV1024_gc; //set prescaler to 1024
```

```
  TCA0.SINGLE.CTRLB = TCA_SINGLE_WGMODE_FRQ_gc; //select frequency mode
  TCA0.SINGLE.CMP0  = 0x0A00; //set compare value at 2560 to get 3Hz signal
}
```

Quelltext 4.20: Initialisieren des Timer/Counter A0

Der Start des TCA0 erfolgt durch Übergeben des Taktsignals an den Ausgang und durch Setzen des `enable`-Bit im Steuerregister A.

```
void startTCA0()
{
  TCA0.SINGLE.CTRLB |= TCA_SINGLE_CMPOEN_bm;  //enable waveform on output pin
  TCA0.SINGLE.CTRLA |= TCA_SINGLE_ENABLE_bm;  //enable TCA0
}
```

Quelltext 4.21: Start des Timer/Counter A0

Soll der TCA0 gestoppt werden, müssen lediglich die beiden Bits gelöscht werden, die beim Start gesetzt wurden.

```
void stopTCA0()
{
  TCA0.SINGLE.CTRLB &= ~(TCA_SINGLE_CMPOEN_bm); //disable waveform on output ↙
    ↳ pin
  TCA0.SINGLE.CTRLA &= ~(TCA_SINGLE_ENABLE_bm); //disable TCA0
}
```

Quelltext 4.22: Stopp des Timer/Counter A0

Das Taktsignal für die Linearbewegung wird mit dem Timer/Counter B0 erzeugt. Der TCB0 ist ein 8 bit-Timer und wird sehr ähnlich dem TCA0 initialisiert. Zuerst wird mit dem Portmultiplexer Port F des ATmega4809 festgelegt. Als Taktquelle wird beim TCB0 nicht der geteilte Prozessortakt verwendet, sondern der TCA0 gesetzt. Als Modus wird der *8-Bit PWM Mode* gewählt. Der *8-Bit PWM Mode* besitzt zwei Vergleichswerte, der CCMPL-Wert steuert die Frequenz bzw. Periode, der CCMPH-Wert den Duty-Cycle. Der Zähler zählt kontinuierlich vom Wert 0 bis zum Wert CCMPL. Zu beginn des Zählvorgangs wird der Ausgang logisch 1 gesetzt. Sobald der Zähler den Wert CCMPH erreicht, wird der Ausgang auf logisch 0 gesetzt. Der Zähler zählt dann weiter bis zum Wert CCMPL. Wird dieser erreicht, beginnt der Zählvorgang von vorne und der Ausgang wird wiederum logisch 1 gesetzt. Im vorliegenden Programm ist der Wert für CCMPL 0x18, das entspricht einem Dezimalwert von 24 und ergibt eine Frequenz von ca. 1 kHz. Der Wert für CCMPH ist mit 0x0C auf die Hälfte von CCMPL festgelegt und führt zu einem symmetrischen Rechtecksignal mit Duty-Cycle 50 % am Ausgang.

```
void initTCB0()
{
  PORTMUX.TCBROUTEA |= PORTMUX_TCB0_bm; //select Port F as output
  TCB0.CTRLA = TCB_CLKSEL_CLKTCA_gc;  //use TCA0 as clocksource
  TCB0.CTRLB = TCB_CNTMODE_PWM8_gc;  //select 8-Bit PWM mode and enable ↙
    ↳ waveform output
  TCB0.CCMPL = 0x18;  //set CCMPL value at 24 to get 1kHz signal
  TCB0.CCMPH = 0x0C;  //set CCMPH value at 12 to get 50% duty cycle
}
```

Quelltext 4.23: Initialisieren des Timer/Counter B0

Der Start des TCB0 erfolgt durch Übergeben des Taktsignals an den Ausgang und durch Setzten des `enable`-Bit im Steuerregister A.

```
void startTCB0()
{
  TCB0.CTRLB |= TCB_CCMPEN_bm;  //enable waveform on output pin
  TCB0.CTRLA |= TCB_ENABLE_bm;  //enable TCB0
}
```

Quelltext 4.24: Start des Timer/Counter B0

Soll der TCB0 gestoppt werden, müssen lediglich die beiden Bits gelöscht werden, die beim Start gesetzt wurden.

```
void stopTCB0()
{
  TCB0.CTRLB |= TCB_CCMPEN_bm;  //enable waveform on output pin
  TCB0.CTRLA  &= ~(TCB_ENABLE_bm);  //disable TCB0
}
```

Quelltext 4.25: Stopp des Timer/Counter B0

Der Motortreiber der Rotationsbewegung erhält das Taktsignal vom Timer/Counter B1. Der TCB1 ist genau wie der TCB0 ein 8 bit-Timer und wird im Grunde gleich konfiguriert. Ein Unterschied sind die verschiedenen Vergleichswerte CCMPL und CCMPH. Der Wert CCMPL ist beim TCA1 0xFF, was einer Frequenz von 60 Hz entspricht. Für den Duty-Cycle ist wieder 50 % gewählt, was mit einem CCMPH Wert von 0x80 erreicht wird. Der wichtigste Unterschied bei der Konfiguration ist jedoch, dass beim TCB1 der Interrupt aktiviert wird. Somit wird jedes Mal, wenn der Zähler den Wert CCMPL erreicht, die *Interrupt Service Routine* mit Interruptvektor `TCB1_INT_vect` aufgerufen.

```
void initTCB1()
{
  PORTMUX.TCBROUTEA |= PORTMUX_TCB1_bm; //select Port F as output
  TCB1.CTRLA = TCB_CLKSEL_CLKTCA_gc;  //use TCA0 as clocksource
  TCB1.CTRLB = TCB_CNTMODE_PWM8_gc; //select 8-Bit PWM mode and make
    ↳ waveform output available
  TCB1.CCMPL = 0xFF;  //set CCMPL value at 255 to get 60Hz signal
  TCB1.CCMPH = 0x80;  //set CCMPH value at 128 to get 50% duty cycle
  TCB1.INTCTRL |= TCB_CAPT_bm;  //enable interrupt
}
```

Quelltext 4.26: Initialisieren des Timer/Counter B1

Die *Interrupt Service Routine* ist eine Funktion, die ausgeführt wird, sobald das Interrupt-Flag gesetzt ist und dabei den Programmablauf unterbricht. Im Fall von TCB1 wird dann die Zählfunktion der Rotationsbewegung aufgerufen. Am Ende einer `ISR` wird das Interrupt-Flag wieder gelöscht, um für den nächsten Aufruf bereit zu sein.

```
ISR(TCB1_INT_vect)
{
  stepCounter();  //call function to count steps and stop movement if necessary
  TCB1.INTFLAGS |= TCB_CAPT_bm; //clear the interrupt flag
}
```

Quelltext 4.27: *Interrupt Service Routine* von TCB1

Der Start des TCB1 erfolgt durch Übergeben des Taktsignals an den Ausgang und durch Setzen des `enable`-Bit im Steuerregister A.

```
void startTCB1()
{
  TCB1.CTRLB |= TCB_CCMPEN_bm;  //enable waveform on output pin
  TCB1.CTRLA |= TCB_ENABLE_bm;  //enable TCB1
}
```

Quelltext 4.28: Start des Timer/Counter B1

Soll der TCB0 gestoppt werden, müssen lediglich die beiden Bits gelöscht werden, die beim Start gesetzt wurden.

```
void stopTCB1()
{
  TCB1.CTRLB &= ~(TCB_CCMPEN_bm); //disable waveform on output pin
```

```
  TCB1.CTRLA &= ~(TCB_ENABLE_bm); //disable TCB1
}
```

Quelltext 4.29: Stopp des Timer/Counter B1

Wie schon weiter oben angemerkt hat der Shutter keine Rückmeldung über seine Position. Daher wird die endliche Bewegungsgeschwindigkeit des Shutters über den Real-Time-Counter simuliert. Der RTC als 16 bit-Zähler wird ähnlich den anderen Timern konfiguriert. Als Quelle für den Zählertakt wird ein interner 1024 Hz-Taktgeber verwendet. Dieser wird ohne Teiler gezählt. Der Vergleichswert, welcher die Dauer der simulierten Shutterbewegung vorgibt, wird auf 0x0600 gesetzt und entspricht 1,5 s. Zudem wird auch der Überlauf-Interrupt aktiviert, um bei Erreichen des Vergleichswerts die *Interrupt Service Routine* mit Interruptvektor `RTC_CNT_vect` aufzurufen.

```
void initRTC()
{
  RTC.CTRLA = RTC_PRESCALER_DIV1_gc; //set prescaler
  RTC.CLKSEL = RTC_CLKSEL_INT1K_gc; //use 1024Hz clocksource
  RTC.PER = 0x0600;                 //set overflow value at 1536 to wait 1,5s
  RTC.INTCTRL |= RTC_OVF_bm;        //enable overflow interrupt
}
```

Quelltext 4.30: Initialisieren des Real-Time-Counter

Die `ISR` des Real-Time-Counter ruft ihrerseits die Stopp-Funktion des Shutters auf. Dann wird der RTC durch Löschen des `enable`-Bit im Steuerregister A deaktiviert, da der einmalige Durchlauf für die Dauer der Bewegung ausreichend ist. Abschließend wird das Interrupt-Flag gelöscht.

```
ISR(RTC_CNT_vect)
{
  stopShutter(); //call function to indicate shutter stopped moving
  RTC.CTRLA &= ~(RTC_RTCEN_bm); //disable RTC
  RTC.INTFLAGS |= RTC_OVF_bm; //clear the interrupt flag in the interrupt ↙
    ↳ flags register to enable the interrupt again
}
```

Quelltext 4.31: *Interrupt Service Routine* des RTC

Gestartet wird der RTC durch Setzen des `enable`-Bit im Steuerregister A.

```
void startRTC()
{
  RTC.CTRLA |= RTC_RTCEN_bm; //enable RTC
}
```

Quelltext 4.32: Start des RTC

Die Eingangssignale der Steuerung wie die Endschalter oder ein Motortreiberfehler werden über die externen Interrupts des ATmega4809 verarbeitet. Wie schon bei den Interrupts der Timer wird beim Aufruf der *Interrupt Service Routine* durch ein Ereignis am Eingangspin der Programmablauf unterbrochen und die `ISR` ausgeführt. Die externen Interrupts werden im Zuge der `setup`-Funktion konfiguriert, um dann jederzeit auf ein Signal am Eingang reagieren zu können. Die beiden Endschalter der Linearbewegung verwenden die Interrupts an Port E. Der Pin 0 des Ports dient für den Endschalter in der äußeren Position und der Pin 1 für die innere Position. Bei beiden Pins wird die `ISR` mit dem Interruptvektor `PORTE_PORT_vect` bei steigender Flanke am Signal aufgerufen.

```
void initPORTEINT()
{
  PORTE.INTFLAGS |= PIN0_bm;  //Pin 0 as interrupt source (LIMIT_OUT_PIN)
  PORTE.INTFLAGS |= PIN1_bm;  //Pin 1 as interrupt source (LIMIT_IN_PIN)
  PORTE.PIN0CTRL = PORT_ISC_RISING_gc;  //raise interrupt of Pin 0 on rising ↙
    ↳ edge
  PORTE.PIN1CTRL = PORT_ISC_RISING_gc;  //raise interrupt of Pin 1 on rising ↙
    ↳ edge
}
```

Quelltext 4.33: Initialisieren der externen Interrupts an Port E

Innerhalb der *Interrupt Service Routine* wird durch einen Funktionsaufruf überprüft, welcher Endschalter betätigt wurde und welche Position somit erreicht ist. Anschließend werden beide Interrupt-Flags gelöscht, um für das nächste Ereignis bereit zu sein.

```
ISR(PORTE_PORT_vect)
{
  posReached();
  PORTE.INTFLAGS |= PIN0_bm;  //clear the interrupt flag of Pin 0
  PORTE.INTFLAGS |= PIN1_bm;  //clear the interrupt flag of Pin 1
}
```

Quelltext 4.34: *Interrupt Service Routine* von Port E

Für das Anzeigen einer Störung der Motortreiber wird der externe Interrupt an Port B

verwendet. Die Quelle des Interrupts bildet Pin 1 an Port B. Die *Interrupt Service Routine* mit Interruptvektor `PORTB_PORT_vect` wird wiederum bei einer steigenden Flanke des anliegenden Signals aufgerufen.

```
void initPORTBINT()
{
  PORTB.INTFLAGS |= PIN1_bm;  //Pin 1 as interrupt source (DRIVER_ERROR_PIN)
  PORTB.PIN1CTRL = PORT_ISC_RISING_gc; //raise interrupt on rising edge
}
```

Quelltext 4.35: Initialisieren der externen Interrupts an Port B

Durch den Funktionsaufruf in der `ISR` kann die Störung des Motortreibers wie weiter oben beschrieben behandelt werden. Nach der Behandlung wird das Interrupt-Flag gelöscht.

```
ISR(PORTB_PORT_vect)
{
 driverError(); // call driver error
 PORTB.INTFLAGS = PIN1_bm;  //clear the interrupt flag
}
```

Quelltext 4.36: *Interrupt Service Routine* von Port B

Die Beschreibung des Arduinosketches macht die Vielzahl an Aufgaben deutlich, die softwaremäßig erledigt werden. Die Kommunikation mit dem Messcomputer wird über das serielle Protokoll ermöglicht. Die Motoren und der Shutter werden durch ein Set an Funktionen in Bewegung versetzt. Über die externen Interrupts wird eine Echtzeitverarbeitung der Eingangssignale ermöglicht. Das Verständnis des Programms spielt in der Anwendung jedoch eine untergeordnete Rolle. Wichtig für die Verwendung der Steuerung sind die Befehle in Tabelle 4.2, die Rückmeldungen in Tabelle 4.3 sowie die Arduino Pinbelegung aus Tabelle 4.1 und die verschiedenen Anschlüsse der Steuerplatine in Tabelle B.2. Zum Zweck der vollständigen Dokumention ist der gesamte Quelltext in Anhang D.1 angefügt.

4.6 Detektoreinheit

Die Detektoreinheit erfüllt den Zweck, Neutronen, die nicht in der Probe absorbiert wurden, in Photonen umzuwandeln und anschließend aufzuzeichnen. Das alles muss unter Beachtung der Exposition im Strahlungsfeld der Neutronenquelle passieren. Aus diesem Grund ist ein L-förmiges Design entstanden, welches das vom Szintillator erzeugte Licht am Eingang des Detektors über einen Spiegel aus dem Neutronenstrahl auslenkt und zur Kamera leitet. Dadurch können die Kamera neben dem Strahl positioniert und Störungen durch die Gammastrahlung weitgehendst vermieden werden.
Der im Weiteren beschriebene Aufbau der Detektoreinheit geht auf die Technische Uni-

versität München als Kooperationspartner am Heinz Maier-Leibnitz Zentrum zurück. Dr. rer. nat. Burkhard Schillinger und sein Team von der Tomographieanlage ANTARES an der Forschungsneutronenquelle Heinz Maier-Leibnitz (FRMII) haben ihre Vision einer günstigen Radiographieanlage, welche großteils im Eigenbau errichtet werden kann, umgesetzt. Dankenswerterweise haben sie ihr Wissen und auch die Pläne geteilt, um an der TU Wien eine vergleichbare Anlage aufbauen zu können.[19]

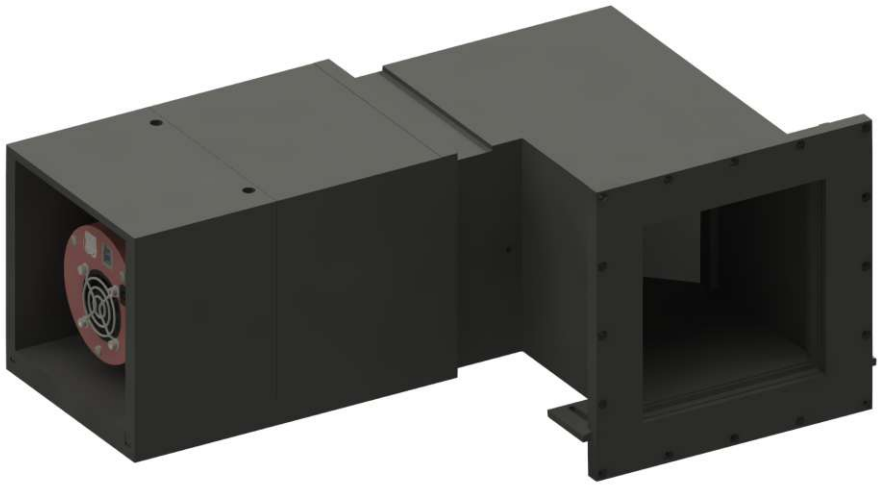


Abbildung 4.7: L-förmiges Detektorgehäuse mit Kamera

Das L-förmige Detektorgehäuse besteht aus sieben Einzelteilen, welche mit dem FDM-3D-Drucker gefertigt wurden. Der Grundteil, der sich in Strahlrichtung befindet, besitzt am hinteren Ende eine Nut, die den Spiegel aufnimmt. Der Spiegel ist ein handelsüblicher Badezimmerspiegel, der auf die passende Größe zugeschnitten wurde. Der zum Strahl um 90° versetzte Gehäuseteil enthält den eigentlichen Detektor, die Kamera. Vor der Kamerahalterung befindet sich ein Hohlraum, welcher mit einer speziellen Bleiabschirmung gefüllt ist. Die Abschirmung ist so gefertigt, dass sie über das Objektiv der Kamera passt, um diese vor gestreuter Gamma- und Neutronenstrahlung zu schützen. Das Detektorgehäuse ist so konstruiert und gefertigt, dass es möglichst lichtdicht ist, um Störungen bei der Detektion des Lichts vom Szintillator zu minimieren. Der Strahleingang des Detektorgehäuses an der dem Reaktor zugewandten Seite hat eine Öffnung von 110 mm × 110 mm. An dieser Position ist auch der Szintillator befestigt. Es handelt sich dabei um einen ^{6}LiF – ZnS(Ag)-Szintillator mit einer Dicke von 100 µm. Der Szintillator erreicht eine räumliche Auflösung von 150 µm und ist mit einem Rahmen am Detektorgehäuse montiert, der am Gehäuseeingang aufgesteckt ist. Das vom Szintillator erzeugte Licht wird über den Spiegel von einer lichtempfindlichen Astronomiekamera detektiert. Verwendet wird dazu eine Astronomiekamera des Herstellers ZWO. In der ASI294MM Pro von ZWO sind ein Sony IMX492 CMOS-Sensor und eine thermoelektrische Kühlung verbaut. Sie erreicht eine Auflösung von 11,7 Megapixel bei einem Seitenverhältnis von 4144 × 2822 Pixel. Mit dem monochromen Sensor sind Aufnahmen nur in schwarzweiß möglich. Das erlaubt eine höhere Empfindlichkeit und ist aufgrund des begrenzten

Abbildung 4.8: Astronomiekamera ZWO ASI294MM Pro

Wellenlängenbereichs am Szintillator mehr als ausreichend. Die Kamera verfügt über ein T2-Gewinde (M42 × 0,75) zur Montage eines Objektivs. Da der vorliegende Aufbau ein Objektiv mit einem C-Mount-Gewinde (1″ - 32) verwendet, ist ein Adapterring notwendig. Das über den Adapterring an der Kamera angeschlossene Objektiv besitzt eine Brennweite von 35 mm und eine Blendenzahl von $f/1{,}7$. Weitere Details und Kennwerte der Kamera und des verwendeten Objektivs sind Anhang A.4.6 bzw. Anhang A.4.7 zu entnehmen.
Wie schon weiter oben ausgeführt, wird zum Schutz der Kamera vor der gestreuten Gamma- und Neutronenstrahlung eine spezielle Abschirmung benötigt. Diese Bleiabschirmung ist direkt vor der Kamera im Detektorgehäuse platziert und umschließt das Objektiv. Sie ist so gefertigt, dass schneller Zugang zum Objektiv gewährt ist. Da es sich um eine Eigenfertigung handelt, wurden als Ausgangsmaterial bereits vorhandene Bleiplatten verwendet. Diesem Umstand ist geschuldet, dass die Abschirmung aus zwei aneinandergereihten Platten besteht. Beide Platten weisen in der Mitte eine Bohrung für das Objektiv und den Photonenstrahl auf. Jene Platte, die direkt an die Kamera grenzt, ist horizontal

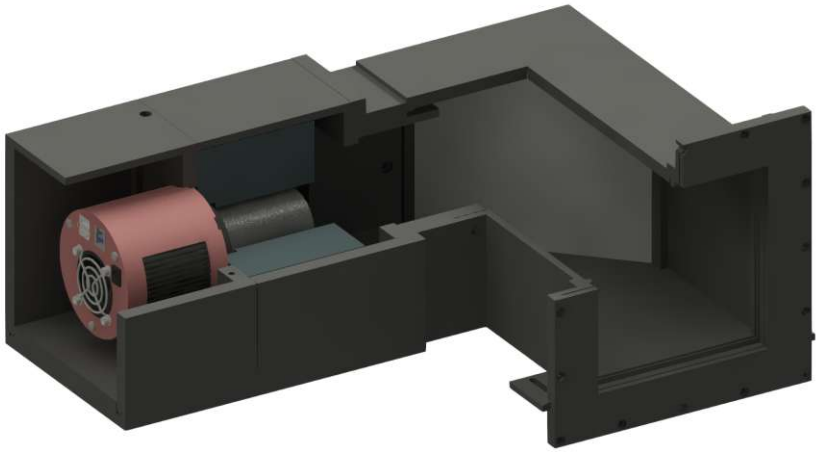


Abbildung 4.9: Innenansicht des Detektorgehäuses mit Bleiabschirmung, Kamera, Objektiv und Spiegel

geteilt, um das Objektiv zugänglich zu machen. Die Teilungsebene ist in ihrem Verlauf abgestuft, um zu verhindern, dass die Strahlung durch einen möglichen Spalt die Kamera erreicht. Ein genaueres Verständnis über die Kameraabschirmung liefern die Konstruktionszeichnungen in Anhang C.3.
Das Detektorgehäuse hat eine Breite von 378 mm, eine Tiefe von 235 mm und eine Höhe von 136 mm. Die Gegenstandsweite vom Szintillator bis zur Hauptebene der ersten Linse im Objektiv beträgt ca. 310 mm. Die Pläne und Modelle zum Gehäuse der Detektoreinheit sind in Anhang C.4 angefügt. Diese sollen einen guten Überblick über die Konstruktion geben, erfüllen aber nicht den Anspruch, vollständig zu sein, da die Fertigung der Teile mit den von der TU München erstellten 3D-Modellen erfolgte.

4.7 Messsoftware

4.7.1 Schnittstellen

Die Messsoftware dient der neuen Neutronenradiographiestation als Bedienoberfläche. Damit eine ausreichende Kommunikation der einzelnen Komponenten über den Computer möglich ist, werden passende Schnittstellen benötigt. Für die in der Detektorbox verbaute Kamera erfolgt die softwaremäßige Anbindung über ein Software-Development-Kit (SDK), welches vom Hersteller der Kamera zur Verfügung gestellt wird. Für die selbst entwickelte Steuerung wurde die Benutzerschnittstelle eigens erstellt. Das so entstandene Python-Paket `imgctrl` dient zur Bedienung der Steuerung über den Computer. Es stellt das Modul `ctrlcmd` zur Verfügung, welches die Schlüsselwörter aus Tabelle 4.2 mit aufrufbaren Methoden einer Klasse verknüpft. Diese Methoden bieten zusätzlich die Möglichkeit, die dem Befehl zugehörigen Meldungen aus Tabelle 4.3 abzuwarten und bei einer Zeitüberschreitung einen Fehler zurückzugeben.
Grundlage des Pakets `imgctrl` ist das Modul `ctrlcom`, dieses führt die Kommunikation zwischen Computer und Steuerung aus. In einem ersten Schritt importiert das Modul alle notwendigen Bibliotheken, welche für die serielle Kommunikation, die Zeitgebung sowie neue Ausführungsstränge, genannt Threads, benötigt werden. Des Weiteren wird die Klasse `Observer` definiert, um das *Observer Design Pattern* zu nutzen. Dabei können sich beobachtete Objekte (Observer) bei einem zu beobachtenden Objekt (Subject) anhängen und von diesem über Änderungen informiert werden.

```
import serial
from serial.tools import list_ports
import time
import threading


class Observer():

    def __init__(self):
        self._methods = []
```

```
    def attach(self, method):
        if method not in self._methods:
            self._methods.append(method)

    def detach(self, method):
        if method in self._methods:
            self._methods.remove(method)

    def call(self, *args, **kwargs):
        for method in self._methods:successful deutsch
            method(*args, **kwargs)
```

Quelltext 4.37: Einbinden der Bibliotheken im Modul `ctrlcom` und Definieren der Klasse *Observer*

Anschließend folgt die Definition der internen Klasse `_SerialCommunication`, diese hat die Aufgabe, die Kommunikation über den seriellen Port abzuwickeln. Beim Erzeugen einer Instanz der Klasse wird automatisch die Konstruktor-Methode `__init__` aufgerufen. In dieser wird ein Instanzobjekt der Klasse `Observer` angelegt, um Beobachterobjekte über neu eingegangene Daten zu informieren. Dann werden ein leeres Stringattribut für die zu empfangenden Meldungen, ein Listenattribut für die möglichen seriellen Kommunikationsports, ein Stoppattribut für den Ausführungsstrang zum Empfangen der Daten sowie der Ausführungsstrang selbst erstellt. Anschließend durchsucht der Konstruktor die Kommunikationsports des Computers nach den Schlüsselwörtern `ACM` und `USB`. Wird ein entsprechendes Gerät zur seriellen Kommunikation gefunden, wird es der zuvor erstellten Liste hinzugefügt.

```
class _SerialCommunication():

    def __init__(self):
        self._receive_observer = Observer()
        self._message = ""
        self._ports = []
        self.stop_receive_data_thread = False
        self.receive_thread = threading.Thread(target = self.__receive)
        # Search for qualified ports in all listed ports
        for port in serial.tools.list_ports.comports():
            # If name of port contains 'ACM' or 'USB', append it to ports
            if ('ACM' or 'USB') in port.device:
                self._ports.append(port.device)
```

Quelltext 4.38: Definition und Konstruktor der Klasse `_SerialCommunication`

Die Klasse `_SerialCommunication` enthält eine Vielzahl an Methoden, die zur Kommuni-

kation benötigt werden. Die intern am Häufigsten verwendete Methode überprüft, ob eine Verbindung über den seriellen Port besteht, und gibt den entsprechenden Wahrheitswert zurück.

```
def _connected(self):
    try:
        return self.serial.is_open
    except:
        return False
```

Quelltext 4.39: Überprüfen der seriellen Verbindung

Beim Aufbau der seriellen Verbindung wird mit der oben beschriebenen Methode zuerst überprüft, ob bereits eine Verbindung besteht. Ist das nicht der Fall, beginnt der Verbindungsaufbau. Sobald die Verbindung mit dem als Parameter übergebenen Port hergestellt ist, wird mit dem Empfangen von Daten begonnen. Kann keine Verbindung hergestellt werden oder ist bereits eine Verbindung vorhanden, wird der Wert `False` zurückgegeben.

```
def _connect(self, port):
    try:
        # Open exclusive serial port with baudrate 9600
        self.serial = serial.Serial(port, 9600, timeout = 0.5, ↙
↪ write_timeout = 1, exclusive = True)
    except:
        return False
    else:
        return self.__start_receive()
```

Quelltext 4.40: Aufbau der seriellen Kommunikation

Nach dem Aufbau der Verbindung wird über den im Konstruktor definierten Ausführungsstrang mit dem Empfangen von Daten begonnen. Vor dem Start des Threads wird noch der Wert `False` in das Stoppattribut geschrieben, um diesen gegebenenfalls auch beenden zu können.

```
def __start_receive(self):
    self.stop_receive_data_thread = False
    self.receive_thread.start()
    return self.receive_thread.is_alive()
```

Quelltext 4.41: Start des Empfangens von Daten über den seriellen Port

Die im neuen Ausführungsstrang aufgerufene Methode enthält eine Schleife, die ausgeführt

wird, solange eine Verbindung besteht. In der Schleife werden zeilenweise Daten vom seriellen Port eingelesen. Nach Dekodierung der empfangenen Daten wird die Escape-Sequenz entfernt. Anschließend werden alle Observer der zu Beginn im Konstruktor erstellten Instanz informiert, dass neue Daten vorliegen. Vor dem Verarbeiten der Eingangsdaten gibt es noch eine Abfrage, ob der Ausführungsstrang beendet werden soll.

```
def __receive(self):
    while self._connected():
        input = self.serial.readline()
        if self.stop_receive_data_thread:
            break
        if input:
            # Decode input data with utf-8 encoding and remove escape ↙
↳ sequence
            self._message = input.decode('utf-8', ↙
↳ 'ignore').replace('\r\n', '')
            self._receive_observer.call(self._message)
```

Quelltext 4.42: Empfangen von Daten über den seriellen Port

Das Beenden des Ausführungsstrangs, falls notwendig, erfolgt über eine eigene Methode. In dieser wird das Stoppattribut mit dem Wert `True` versehen, um den Ausführungsstrang zu beenden. Zusätzlich wird das Lesen der Daten vom seriellen Port abgebrochen. Nach Abwarten der Beendigung des Ausführungsstrangs wird dessen Status invertiert zurückgegeben. Somit bedeutet der Rückgabewert `True`, das Beenden war erfolgreich. Hingegen steht der Wert `False` für eine Zeitüberschreitung beim Beenden.

```
def __stop_receive(self):
    self.stop_receive_data_thread = True
    self.serial.cancel_read()
    self.receive_thread.join(timeout = 2)
    return not self.receive_thread.is_alive()
```

Quelltext 4.43: Beenden des Empfangens von Daten über den seriellen Port

Das Senden von Daten über den seriellen Port erfolgt über eine Methode mit drei Parametern: die zu sendenden Daten, die zu erwartende Rückmeldung und die Zeit in Sekunden, bis eine Zeitüberschreitung vorliegt. Sollte einer der letzten beiden Parameter nicht übergeben worden sein oder den Wert `None` aufweisen, so werden die Daten direkt über die Schreibmethode an den seriellen Port geschrieben. Soll jedoch eine Meldung abgewartet werden und ist auch eine Zeit übergeben worden, werden die Daten unter Zuhilfenahme der Schreibmethode mit der Sendemethode übergeben. War das Schreiben erfolgreich, wird eine Wartemethode aufgerufen und auf die Rückmeldung gewartet. Als Rückgabewert dient im ersten Fall die Sendemethode und im zweiten die Wartemethode oder, sofern

das Senden nicht bereits erfolgreich war, der Wert `False`.

```
    def _send(self, data, response = None, timeout = None):
        # If no timeout or response is passed, just write data
        if timeout is None or response is None:
            return self.__write(data)
        # If timeout is type integer and response is type string, write data
    ↪ and wait for response
        elif isinstance(timeout, int) and isinstance(response, str):
            if self.__write(data):
                    # If writing was successful, wait for response
                    return self.__wait(response, timeout)
            else:
                    return False
        else:
            return False
```

Quelltext 4.44: Senden von Befehlen

Der Schreibvorgang beginnt mit Aufruf der Schreibmethode und den Daten als Argument. Nach dem Überprüfen der bestehenden Verbindung werden die Daten mit einer Escape-Sequenz versehen. Anschließend beginnt der eigentliche Sendevorgang mit dem Schreiben der Daten als *utf-8* kodierte Bytes über den seriellen Port. Ist der Vorgang abgeschlossen, folgt die Rückgabe des Werts `True`. Sollte die Verbindung nicht vorhanden sein, wird `False` zurückgegeben.

```
    def __write(self, data):
        if self._connected():
            try:
                # Add escape sequence to data
                data = data + "\n"
                # Write utf-8 encoded data to serial port
                self.serial.write(bytes(data.encode('utf-8', 'ignore')))
            except serial.SerialTimeoutException:
                return False
            else:
                return True
        else:
            return False
```

Quelltext 4.45: Schreiben von Daten über seriellen Port

Soll auf die Rückmeldung gewartet werden, kommt nach dem Senden der Daten die Wartemethode zur Anwendung. Mit dem Parameter für die Zeitüberschreitung und der aktuellen Zeit wird der Zeitpunkt der Überschreitung berechnet. In einer Schleife wird gewar-

tet, bis die Rückmeldung im übergebenen Parameter den empfangenen Daten entspricht. Das Überschreiten des vorher berechneten Zeitpunkts dient als Abbruchbedingung für die Schleife und als Indikator, dass die Verarbeitung der Daten in der Steuerung nicht fehlerfrei funktioniert hat.

```
    def __wait(self, response, timeout):
        # Build expire time from timeout
        exittime=time.time()+timeout
        while self._message != response:
            # Break loop and stop waiting when time exceeds expire time
            if time.time() > exittime:
                return False
        return True
```

Quelltext 4.46: Warten auf Rückmeldung am seriellen Port

Als Pendant zur Methode des Verbindungsaufbaus gibt es auch eine Methode zum Trennen der Verbindung. Dabei wird überprüft, ob die Verbindung überhaupt besteht. Anschließend wird der Ausführungsstrang zum Einlesen der Daten gestoppt und die serielle Verbindung beendet. Bei einem erfolgreichen Trennen ist der Rückgabewert `True`, in allen anderen Fällen wird `False` zurückgegeben.

```
    def _disconnect(self):
        if self._connected():
            if self.__stop_receive():
                try:
                    self.serial.close()
                except:
                    return False
                else:
                    return True
            else:
                return False
        else:
            return True
```

Quelltext 4.47: Beenden der seriellen Kommunikation

Das Paket `imgctrl` ermöglicht mit einem weiteren Modul die Kommunikation mit der Steuerung über aufrufbare Objekte. Das Modul `ctrlcmd` stellt mit den öffentlichen Methoden der Klasse `Commander` für jeden Befehl in Tabelle 4.2 eine aufrufbare Methode zur Verfügung. Wiederum aufbauend auf der Klasse zur seriellen Kommunikation importiert die neue Klasse von `_SerialCommunication` Attribute und Methoden. Der Konstruktor der Kindklasse ruft zuerst den Konstruktor der Elternklasse auf, um diese zu initiali-

sieren. Dann wird eine neue Instanz der `Observer`-Klasse erstellt, um Beobachter über gesendete Daten zu informieren. Anschließend werden die nicht öffentlichen Attribute der Elternklasse durch Neudefinition öffentlich gemacht.

```
import ctrlcom

class Commander(_SerialCommunication):

    def __init__(self):
        _SerialCommunication.__init__(self)
        self.send_observer = Observer()
        self.receive_observer = self._receive_observer
        self.ports = self._ports
```

Quelltext 4.48: Definition und Konstruktor der Klasse `Commander`

Die in der Elternklasse bereits vorhandene Methode zum Senden der Daten wird überschrieben. Sie wird so abgeändert, dass vor dem Aufruf der Sendemethode der Elternklasse die Sendebeobachter über das Senden und die dabei geschriebenen Daten informiert werden.

```
    def _send(self, instruction, *args, **kwargs):
        self.send_observer.call(instruction)
        return _SerialCommunication._send(self, instruction, *args, **kwargs)
```

Quelltext 4.49: Erweiterte Sendemethode

Die Methoden zum Aufbau, Überprüfen und Trennen der seriellen Verbindung der Elternklasse werden mit Verweisen in neu definierten Methoden über die Kindklasse öffentlich gemacht.

```
    def connect(self, *args, **kwargs):
        return _SerialCommunication._connect( *args, **kwargs)

    def disconnect(self):
        return self._disconnect()

    def connected(self):
        return self._connected()
```

Quelltext 4.50: Öffentlichmachen der Methoden zum Aufbau, Überprüfen und Trennen der seriellen Verbindung

Anschließend folgt für jeden Befehl aus Tabelle 4.2 und der zugehörigen Meldungen aus Tabelle 4.3 eine eigene aufrufbare Methode. Jede Befehlsmethode ruft ihrerseits die Sendemethode mit Befehlen und der erwarteten Rückmeldung als Argument auf. Zusätzlich kann der Befehlsmethode ein Wert für die Zeitüberschreitung als Argument mit dem optionalen Schlüsselwort `timeout` übergeben werden. Die beiden Methoden zur Rotationsbewegung benötigen als Argument mit Schlüsselwort `step` noch zusätzlich die Anzahl der Schritte, um die gedreht werden soll.

```
    def linear_out(self, *args, **kwargs):
         return self._send("move_out", "pos_out", *args, **kwargs)

    def linear_in(self, *args, **kwargs):
        return self._send("move_in", "pos_in",*args, **kwargs)

    def rotate_clockwise(self, *args, step = None, **kwargs):
        # Check if step is given and has correct type
        if type(step) is int:
            return self._send("rot_cw+" + str(step), "rot_stopped", *args, ↙
↳ **kwargs)
        else:
            return False

    def rotate_counter_clockwise(self, *args, **kwargs):
        # Check if step is given and has correct type
        if type(step) is int:
            return self._send("rot_ccw+" + str(step),"rot_stopped", *args, ↙
↳ **kwargs)
        else:
            return False

    def open(self, *args, **kwargs):
        return self._send("open_shutter", "shutter_opened", *args, **kwargs)

    def close(self, *args, **kwargs):
        return self._send("close_shutter", "shutter_closed", *args, **kwargs)

    def stop_all(self, *args, **kwargs):
        return self._send("stop_all", "all_stopped", *args, **kwargs)

    def stop_rotate(self, *args, **kwargs):
        return self._send("stop_lin", "lin_stopped", *args, **kwargs)

    def stop_linear(self, *args, **kwargs):
        return self._send("stop_rot", "rot_stopped", *args, **kwargs)
```

Quelltext 4.51: Öffentliche Befehlsmethoden mit den Anweisungen für die Steuerung

4.7.2 Messapplikation

Die Bedienung der neuen Neutronenradiographiestation erfolgt über die Messapplikation mit grafischer Benutzeroberfläche. Diese ist der Knotenpunkt zwischen den einzelnen Komponenten. Sie bietet eine gemeinsame Bedienung für alle mit dem Computer verbundenen Teile wie die Kamera in der Detektoreinheit oder der Arduino in der Steuerung, welcher die Befehle an den Probentisch oder den Shutter weitergibt. Die Messapplikation ist in der Programmiersprache Python verfasst und nutzt das Python GUI-Toolkit Tk zur Erstellung der grafischen Benutzeroberfläche.
Die Messapplikation besteht aus vier verschiedenen Anwendungen.

- Die Anwendung **Controller** dient der Bedienung der Steuerung.
- Die Kamera wird über die Anwendung **Camera** gesteuert.
- Mit dem Anwenderprogramm **Radiography** werden die beiden vorher genannten kombiniert und um einige Funktionen ergänzt.
- Das Anwenderprogramm **Tomography** entwickelt die Radiography-Anwendung weiter und ergänzt diese um eine Ablaufsteuerung.

Wird die Messapplikation über das Pythonskript `neuimg.py` gestartet, so öffnet sich das in Abbildung 4.10 dargestellte Dialogfenster. Im *AppManager* kann ausgewählt werden, welche der vier oben angeführten Anwendungen ausgeführt werden soll.

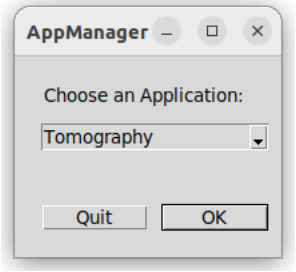


Abbildung 4.10: Dialogfenster AppManager

4.7.3 Controller

Die *Controller*-Anwendung bietet eine grafische Oberfläche zur Kommunikation mit der Steuerung. Die serielle Verbindung zum Arduino wird beim Start der Applikation automatisch hergestellt. Soll eine Verbindung zu einem anderen Gerät hergestellt werden, so muss der entsprechende serielle Port im Dropdown-Menü angewählt werden. Daraufhin wird die bestehende Verbindung getrennt und die neue Verbindung aufgebaut. Das unter dem Dropdown-Menü positionierte Ausgabefeld dient dazu, die gesamte Kommunikation mit der Steuerung abzubilden. Meldungen vom Arduino werden mit vorangestelltem >> markiert. Die Steuerung der einzelnen Komponenten erfolgt über die Schaltflächen des Bedienfensters. So kann die Probe über die Schaltflächen *out*, *stop* und *in* aus der oder in

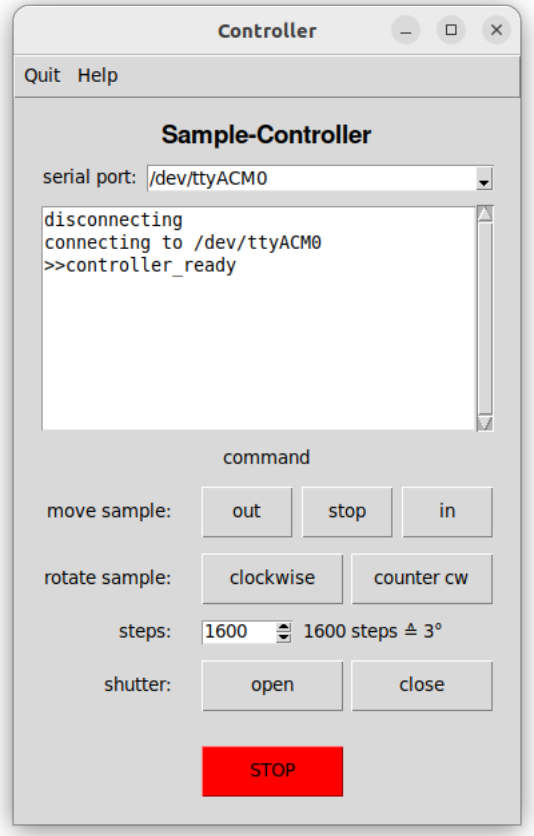


Abbildung 4.11: Dialogfenster Steuerung

die Bestrahlungsposition bewegt oder auch gestoppt werden. Durch Eingabe der Schritte im Feld *steps* und anschließendem Drücken der Schaltflächen *clockwise* oder *counterclockwise* wird die Probe im oder gegen den Uhrzeigersinn gedreht. Der Drehwinkel pro Schritt errechnet sich aus der Auflösung des Drehtisches bei vollen Schritten (Tabelle A.3) und der gewählten Microstep-Auflösung (Tabelle A.7) des Motortreibers.

$$\text{resolution at full step} * \text{microstep resolution} = 0{,}015\,°/\text{step} * \frac{1}{8} = 0{,}001\,875\,°/\text{step}$$

Da der Arduino und in weiterer Folge auch der Motortreiber und der Schrittmotor nur ganzzahlige Schritte verarbeiten können und der Wert aus obiger Gleichung daher unpraktisch ist, ergibt dieser multipliziert mit 1600 eine weitaus bessere Beziehung. Dann entsprechen 1600 Schritte einem Winkel von 3°. Über die beiden Schaltflächen *open* und *close* wird der Shutter geöffnet oder geschlossen. Der rote Stopp-Button erfüllt die Aufgabe eines Not-Aus, alle Bewegungen werden gestoppt und der Shutter schließt. Beim Schließen des Bedienfensters über den Menüeintrag *Quit* oder das obligatorische ⊗ wird zuerst die Funktion des Stopp-Buttons aufgerufen, dann die serielle Verbindung getrennt und das Fenster in Abbildung 4.11 geschlossen. Als Grundlage für die Kommunikation der Messsoftware mit der Steuerung dient das oben beschriebene und vollständig abgebildete Paket `imgctrl`, welches die Funktionen zur Verfügung stellt, die direkt mit den Schaltflächen der Bedienoberfläche verknüpft sind.

4.7.4 Camera

Die Kamera in der Detektoreinheit wird über die *Camera*-Applikation bedient. Wird die Kamera beim Start der Anwendung bereits erkannt, so wird automatisch eine Verbindung zu dieser hergestellt. Ist mehr als eine Kamera vorhanden, kann im Dropdown-Menü *camera* die Kamera gewechselt werden. Über das Eingabefeld *file path* oder die Schaltfläche *search* wird der Pfad zum Ordner festgelegt, in dem die Aufnahmen abgelegt werden. Das darunter befindliche Eingabefeld legt den Namen der Aufnahmen fest. Der Name kann auch Zeit- oder Datumswerte enthalten, die dem `strftime()` Formatcodes entsprechen und im C-Standard aus 1989 definiert sind. Im Dropdown-Menü *image type* kann der Bildtyp und im Eingabefeld *bandwidth* die USB-Bandbreite in fps gewählt werden. Das Menü *resolution* legt die Auflösung der Aufnahme fest. Es gibt einige Standard-Auflösungen oder die Möglichkeit der benutzerdefinierten Auflösung. Diese wird durch die vier Eingabefelder *width*, *height*, *x_0* und *y_0* angegeben. Die vordefinierte Auflösung ist 4800×4800 Pixel,

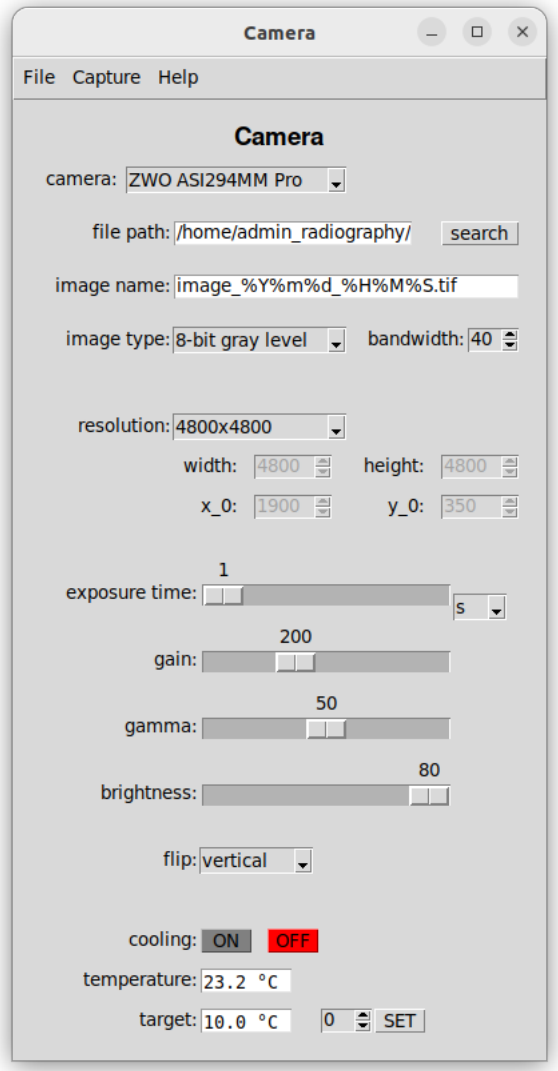


Abbildung 4.12: Dialogfenster Kamera

diese ist in Kombination mit der Detektoreinheit am besten geeignet, mehr dazu in Abschnitt 5.1.1. Die Parameter für die Aufnahme können über Schieberegler gesetzt werden. Die Aufnahmezeit kann von 3,2 µs bis 2000 s variiert werden. Die Verstärkung kann in einem Bereich von 0 bis 570 eingestellt werden, wobei zu beachten ist, dass eine hohe Verstärkung zu mehr Rauschen führt, jedoch die Aufnahmezeit verkürzt werden kann. Zusätzlich gibt es die beiden Schieberegler *gamma* für die Gammakorrektur und *brightness*, der einen Offset zu den Pixelwerten addiert, um negative Werte am Analog-Digital-Konverter zu vermeiden. Über das Menü *flip* kann die Aufnahme horizontal oder vertikal gespiegelt werden. Die Kühlung der Kamera wird über die beiden Schaltflächen *ON* und *OFF* kontrolliert. Es werden sowohl die Ist-, als auch die Solltemperatur angezeigt. Die Solltemperatur kann mit der Schaltfläche *SET* verändert werden. Der Menüeintrag *File* bietet die Möglichkeit, erstellte Konfigurationen abzuspeichern oder bereits gespeicherte zu laden. Außerdem befindet sich das Bedienelement *Quit* zum Beenden der Anwendung und Schließen des Dialogfensters in Abbildung 4.12 ebenfalls im Menüeintrag *File*. Das Erstellen von Aufnahmen erfolgt über den Menüeintrag *Capture*. Dabei werden die Parameter gemäß der gewählten Einstellungen an die Kamera übergeben und eine Aufnahme gestartet. Nach dem Ablauf der Aufnahmezeit und dem Beenden der Aufnahme wird die Bilddatei im vorab ausgewählten Ordner abgelegt. Zu dieser Bilddatei wird zudem eine Textdatei mit der Konfiguration gespeichert. Diese dient zur Dokumentation und kann über den Menüeintrag *File* zu einem späteren Zeitpunkt geladen werden. Die Kamera wird über ein Softwarepaket des Herstellers in die Anwendung eingebunden.

4.7.5 Radiography

Die Anwendung *Radiography* setzt sich im Wesentlichen aus den beiden zuvor beschriebenen Appliktionen zusammen. Die zwei Dialogfenster der Anwendungen *Controller* und *Camera* bilden, wie in Abbildung 4.13 dargestellt, die Grundlage der Bedienoberfläche. Zusätzlich zu den bereits oben beschriebenen Einstell- und Bedienmöglichkeiten wird in der *Radiography*-Anwendung noch ein Fenster *Settings* mit ergänzenden Einstellungen angefügt. In dem Dialogfenster besteht die Möglichkeit, über ein Häkchen die Shutterautomatik zu aktivieren. Ist sie aktiv, wird der Shutter vor dem Start der Aufnahme automatisch geöffnet und nach dem Aufnahmevorgang wieder geschlossen.

4.7.6 Tomography

Die *Tomography*-Applikation erweitert die zuvor beschriebene *Radiography*-Anwendung um zusätzliche Funktionen. Wie in Abschnitt 3.3.2 beschrieben, werden für eine Neutronentomographieaufnahme Projektionen aus verschiedenen Einfallswinkeln benötigt. Das erreicht man durch Drehen der Probe im Neutronenstrahl. Um das in der Praxis umzusetzen, muss die Probe nach jeder Aufnahme vom Drehtisch weitergedreht werden, um dann eine weitere Aufnahme zu starten. Essenziell für die Umsetzung ist daher das Verarbeiten der Rückmeldungen der Steuerung, sodass die Aufnahme erst erfolgt, wenn die Drehung abgeschlossen ist. Zur Rekonstruktion des Probenobjekts aus den Aufnahmen ist es notwendig, vor der eigentlichen Messung noch den Dunkelstrom und den leeren Strahl zu messen. Für die Dunkelstrommessung wird ohne Belichtung, also bei geschlossenem

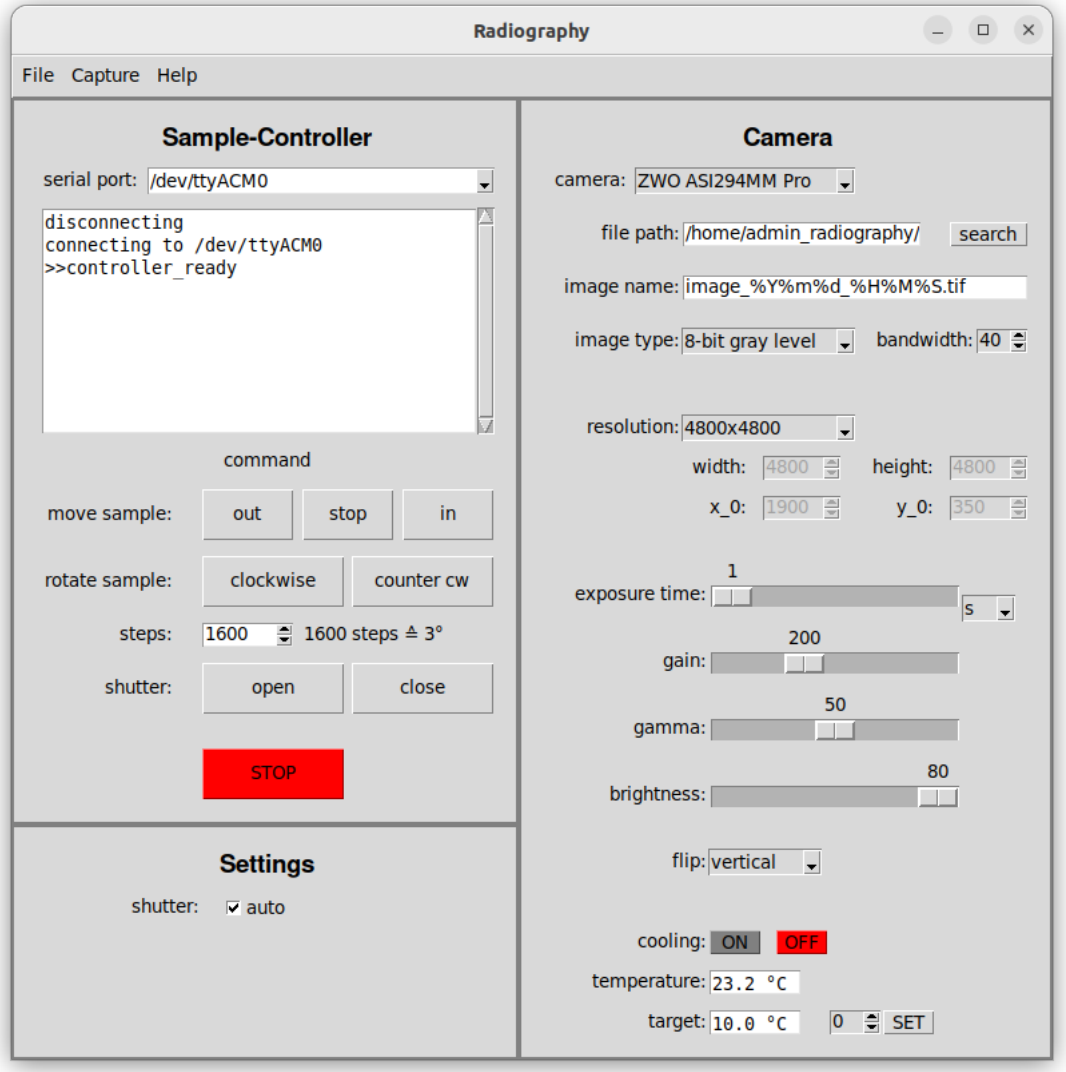


Abbildung 4.13: Dialogfenster Radiographie

Shutter, gemessen. Die Messung des leeren Strahls erfolgt, indem die Probe mit dem Linearantisch aus dem Strahl gefahren wird. Bei der eigentlichen Messung befindet sich die Probe im Neutronenstrahl und der Shutter ist geöffnet. Nach jeder Aufnahme wird von 0° beginnend in gleichgroßen Winkelschritten bis 180° gedreht. Dann erfolgt eine Drehung um einen halben Winkelschritt, um zu verhindern, dass die selben Projektionen ein zweites Mal von der anderen Seite aufgenommen werden. Danach wird weiter um den vollen Winkelschritt bis zur Ausgangsposition gedreht. Zusätzlich werden zu jedem Set von Aufnahmen eine Textdatei mit der Konfiguration und eine Textdatei mit den Winkelinformationen jeder Projektion im selben Ordner abgelegt. Die für Tomographieaufnahmen notwendigen Erweiterungen der *Radiography*-Applikation befinden sich im Fenster *Settings*. So kann über ein Dropdown-Menü die Anzahl der Aufnahmen gewählt werden. Wird der Haken bei *reference shot* gesetzt, so werden die Dunkelstrommessung und die Messung des offenen Strahls vor dem Aufnehmen der Projektionen automatisch erstellt.

Im Eingabefeld *darkcurrent shots* kann die Anzahl der Dunkelstromaufnahmen gewählt werden. Zusätzlich kann die Zeit zwischen den Aufnahmen angegeben werden. Auch zur Messung des offenen Neutronenstrahls kann die Anzahl der Aufnahmen vorgegeben werden, sowie auch die Zeit zwischen den einzelnen Aufnahmen. Der Start der gesamten Messung erfolgt im Menüeintrag *Measure* mit der Option *Start*. Soll die Messung abgebrochen werden, kann dies über das Feld *Stop* im Menüeintrag erfolgen. Zusätzlich bietet das Menü *Measure* analog zur Kameraanwendung auch die Möglichkeit von Einzelaufnahmen mit der Schaltfläche *Capture*. Im Menüeintrag *File* kann die Konfiguration geladen oder gespeichert sowie das Programm beendet werden.

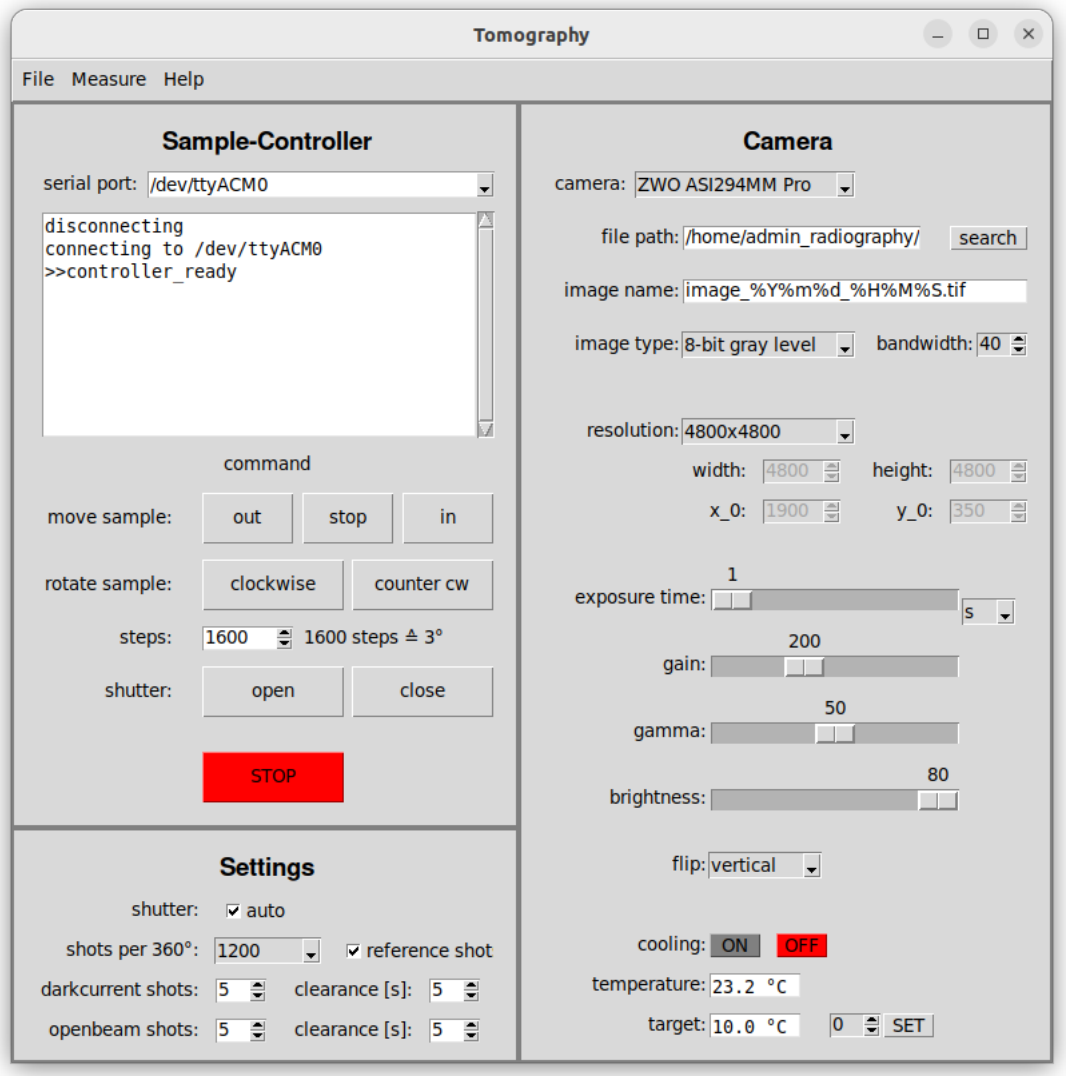


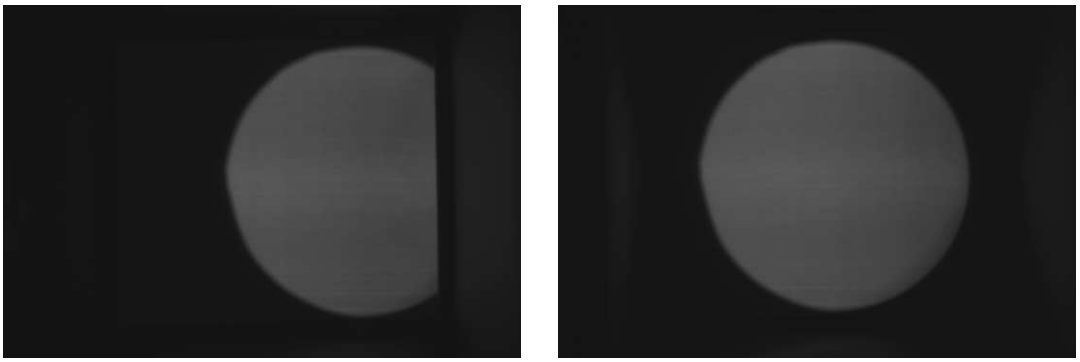
Abbildung 4.14: Dialogfenster Tomographie

5 Aufnahmen mit der neuen Neutronenradiographieanlage

5.1 Radiographische Messungen

5.1.1 Offener Neutronenstrahl

Die erste Messung mit der neu installierten Neutronenradiographie- und -tomographieanlage an der thermischen Säule des TRIGA Mark-II diente der Überprüfung der Funktion und dem Ausrichten der Anlage im Strahl. Dazu wurden ohne Probe mehrere Aufnahmen vom offenen Strahl gemacht. In Abbildung 5.1a ist deutlich zu erkennen, dass die vertikale Ausrichtung bereits sehr gut ist, die horizontale jedoch noch verbessert werden muss. Der Neutronenstrahl trifft die Szintillatorfläche nur teilweise. Die zweite Aufnahme, Abbildung 5.1b, stellt eine deutliche Verbesserung der Ausrichtung dar, der gesamte Strahl trifft auf den Szintillator. Diese beiden ersten Aufnahmen mit der neuen Neutronenradiographieanlage zeigen aber noch die gesamte Öffnungsfläche der Detektoreinheit. Durch Wahl der in Abschnitt 4.7 beschriebenen Auflösung von 4800×4800 Pixel der Kamera kann der Aufnahmebereich begrenzt werden. Abbildung 5.2 zeigt den Strahl bei optimaler Ausrichtung der Detektoreinheit und Begrenzung des Aufnahmebereichs. In Tabelle 5.1 ist die Konfiguration der Messsoftware für die drei Aufnahmen des offenen Neutronenstrahls angeführt.



(a) Strahl verfehlt Szintillator (b) Strahl trifft Szintillator

Abbildung 5.1: Aufnahmen des offenen Neutronenstrahls zur Bestimmung der optimalen Position der Detektoreinheit

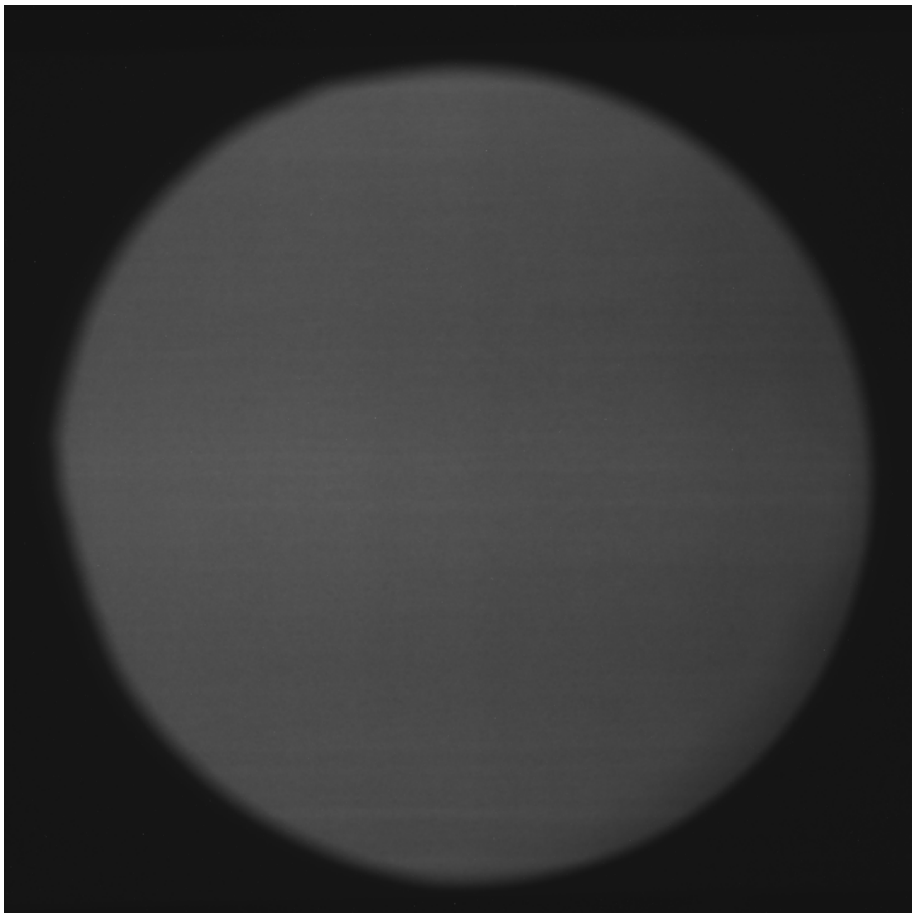


Abbildung 5.2: Optimale Aufnahme des offenen Neutronenstrahls nach Positionierung der Detektoreinheit und Begrenzung des Aufnahmebereichs

Parameter	Wert
image type:	8-bit gray level
resolution:	4800x4800
x_0:	2000
y_0:	600
width:	4800
height:	4800
bandwidth:	40
exposure time:	300 s
gain:	200
gamma:	50
brightness:	80
flip:	vertical
target:	−20,0 °C
temperature:	−19,0 °C

Tabelle 5.1: Konfiguration der Messsoftware zu den Aufnahmen des offenen Strahls

5.1.2 Borstahlblech

Die ersten Messungen des Neutronenstrahls bestätigten die Funktion der neuen Anlage und dienten dazu, den Aufnahmebereich zu optimieren. Die nächste Messung sollte überprüfen, ob eine vorliegende Intensitätsverteilung im Neutronenstrahl auch tatsächlich abgebildet wird. Dazu wurde ein Borstahlblech im Strahl positioniert. Wie in Abschnitt 2.3 beschrieben, ist Bor ein sehr guter Neutronenabsorber und besitzt einen großen Wirkungsquerschnitt für thermische Neutronen. Durch Einbringen des Borstahlblechs in den Neutronenstrahl kann ein großer Unterschied in der Strahlintensität zwischen der vom Blech bedeckten Fläche und der unbedeckten Fläche erzeugt werden. Abbildung 5.3 zeigt das Ergebnis der Messung mit der in Tabelle 5.2 angeführten Konfiguration der Messsoftware. Dabei ist sehr gut zu erkennen, dass vom Bor nahezu alle Neutronen absorbiert werden, da sich der Bereich, in dem sich das Blech befindet, als ähnlich dunkel wie die unbelichteten Ränder darstellt. Der so zu erwartende hohe Kontrast bei der Messung mit Bor kann als quantitativer Nachweis der guten Funktionalität der neuen Anlage angesehen werden.

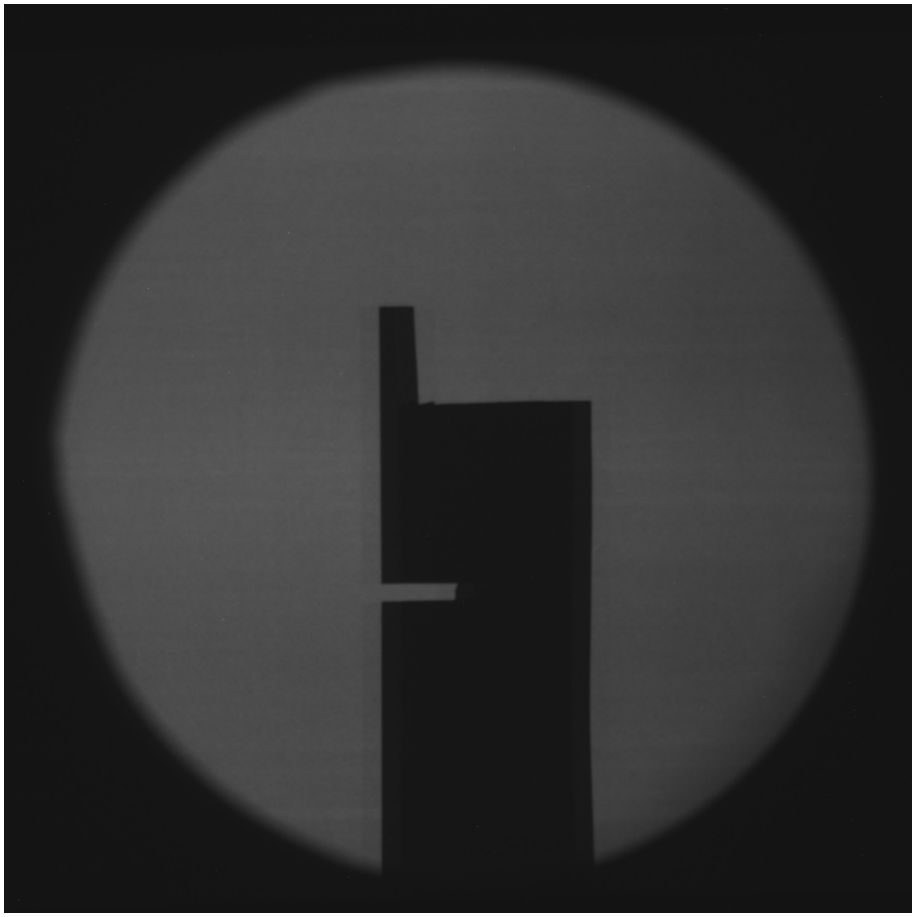


Abbildung 5.3: Abbildung eines Borstahlblechs zur Überprüfung der Empfindlichkeit gegenüber verschiedener Intensitäten des Neutronenstrahls

Parameter	Wert
image type:	8-bit gray level
resolution:	4800x4800
x_0:	2000
y_0:	600
width:	4800
height:	4800
bandwidth:	40
exposure time:	300 s
gain:	200
gamma:	50
brightness:	80
flip:	vertical
target:	−20,0 °C
temperature:	−19,0 °C

Tabelle 5.2: Konfiguration der Messsoftware zur Aufnahme des Borstahlblechs

5.1.3 Wasserwaage

Die neu installierte Neutronenradiographieanlage bietet die Möglichkeit zur zerstörungsfreien Analyse von Proben. Da die Neutronen in Abhängigkeit der Kerneigenschaften der Atome mit der Probe interagieren, bietet sich die Abbildung mit Neutronen vor allem für Proben an, deren Bestandteile stark unterschiedliche Wirkungsquerschnitte für die Absorption von Neutronen besitzen. Zum Beispiel lassen sich organische Verbindungen, welche zumeist aus Wasserstoff und Kohlenstoff bestehen, sehr gut abbilden. Der Wasserstoff besitzt im Vergleich zu den in der Technik häufig verwendeten Metallen wie Eisen oder Aluminium einen relativ großen Absorbtionswirkungsquerschnitt. Dadurch heben sich Kunststoffteile deutlich von metallischen Komponenten einer Probe ab. Sehr anschaulich lässt sich das durch Abbilden der Libelle einer Wasserwaage zeigen. Eine Wasserwaage besteht aus einem Aluminiumprofil, welches ein mit Flüssigkeit gefülltes Glasröhrchen trägt, die sogenannte Libelle. In dem Röhrchen befindet sich eine Luftblase, um die horizontale Ausrichtung anzuzeigen. In Abbildung 5.4 ist die Aufnahme einer Wasserwaagenlibelle mit der neuen Radiographieanlage abgebildet. Nachdem der Kontrast mittels Bildbearbeitung verbessert wurde, sind die Kunststoffteile der Halterung und das flüssigkeitsgefüllte Glasröhrchen in dunkel deutlich zu erkennen. Das Aluminiumprofil ist nur als leichter Schatten am Rand sichtbar. Die aus Eisen bestehende Schraube zum Fixieren der Libelle im Profil ist auch sehr gut zu erkennen. Bei genauer Betrachtung des Röhrchens kann in der Mitte ein etwas hellerer Fleck identifiziert werden. Dabei handelt es sich um die Luftblase in der Libelle, welche die Flüssigkeit verdrängt und dadurch schemenhaft sichtbar wird. Die Aufnahme der Wasserwaage ist eine besonders anschauliche Möglichkeit, die Funktionalität der neuen Radiographieanlage zu überprüfen. Außerdem ist sie ein gutes Anwendungsbeispiel für die zerstörungsfreie Abbildung von komplexen Proben, die aus verschiedenen Materialien zusammengesetzt sind. Nebenbei wird auch das

gute Auflösungsvermögen der Anlage sichtbar. So ist das Gewinde der Schraube deutlich zu erkennen, bei genauer Betrachtung sind auch die Gewindegänge im Bereich der Mutter sichtbar. Diese erste detailreiche Aufnahme ist sehr vielversprechend und zeigt zahlreiche Möglichkeiten für die zukünftige Anwendung der neuen Neutronenradiographieanlage auf.

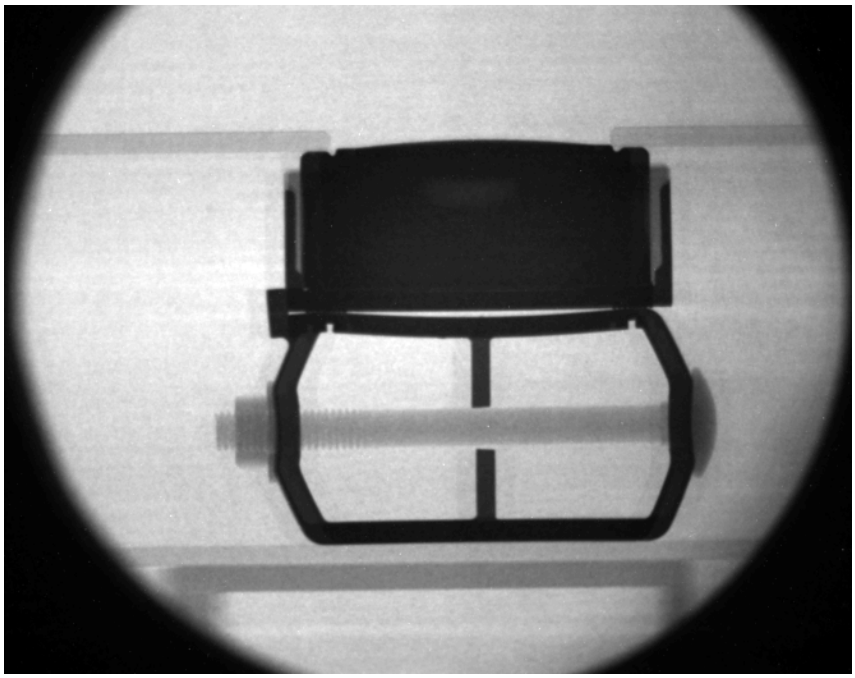


Abbildung 5.4: Abbildung der Libelle einer Wasserwaage mit dem Ziel, die Wechselwirkung verschiedener Materialien in der Probe sichtbar zu machen

Parameter	Wert
image type:	16-bit gray level
resolution:	4800x4800
x_0:	2000
y_0:	600
width:	4800
height:	4800
bandwidth:	40
exposure time:	300 s
gain:	200
gamma:	50
brightness:	80
flip:	vertical
target:	−20,0 °C
temperature:	−19,8 °C

Tabelle 5.3: Konfiguration der Messsoftware zur Aufnahme der Wasserwaage

5.1.4 Stoppuhr

Die Auswertung der vorangegangenen Messungen zeigen die sehr guten Ergebnisse der Aufnahmen mit der neuen Radiographiestation. Um noch mehr Informationen über die Qualität der Messungen zu bekommen, ist in Abbildung 5.5 die Aufnahme einer mechanischen Stoppuhr dargestellt. Die mit der Konfiguration aus Tabelle 5.4 erstellte Abbildung zeigt die Qualität der Aufnahmen deutlich. Nachdem die Aufnahme bearbeitet und der Kontrast verbessert wurde, sind die Lagersteine der Uhr als dunkle Punkte um die Zahnräder deutlich zu erkennen. Auch die Feder in der Krone ist sehr gut sichtbar. Bei genauer Betrachtung der Aufnahme kann sogar die Unruh mit Schenkel und Ring rechts unten identifiziert werden. Auch bei dieser Abbildung ist ersichtlich, dass Teile aus Kunststoff, wie der oben liegende Haltebügel, Neutronen stark absobieren und daher dunkel am Bild erscheinen. In der Regel werden Radiographieabbildungen nachbearbeitet, um jene Regionen besser darzustellen, die von Interesse sind. Dabei ist zu beachten, dass bei jeder Bearbeitung eines Bilds Informationen verloren gehen. Es ist daher stets abzuschätzen, ob dieser Verlust akzeptiert werden kann oder nicht. In den meisten Fällen trifft der Informationsverlust jedoch Teilbereiche der Abbildung, die ohnehin nicht von Relevanz sind.

Parameter	Wert
image type:	16-bit gray level
resolution:	4800x4800
x_0:	2000
y_0:	600
width:	4800
height:	4800
bandwidth:	40
exposure time:	360 s
gain:	200
gamma:	50
brightness:	80
flip:	vertical
target:	−25,0 °C
temperature:	−20,6 °C

Tabelle 5.4: Konfiguration der Messsoftware zur Aufnahme der Stoppuhr

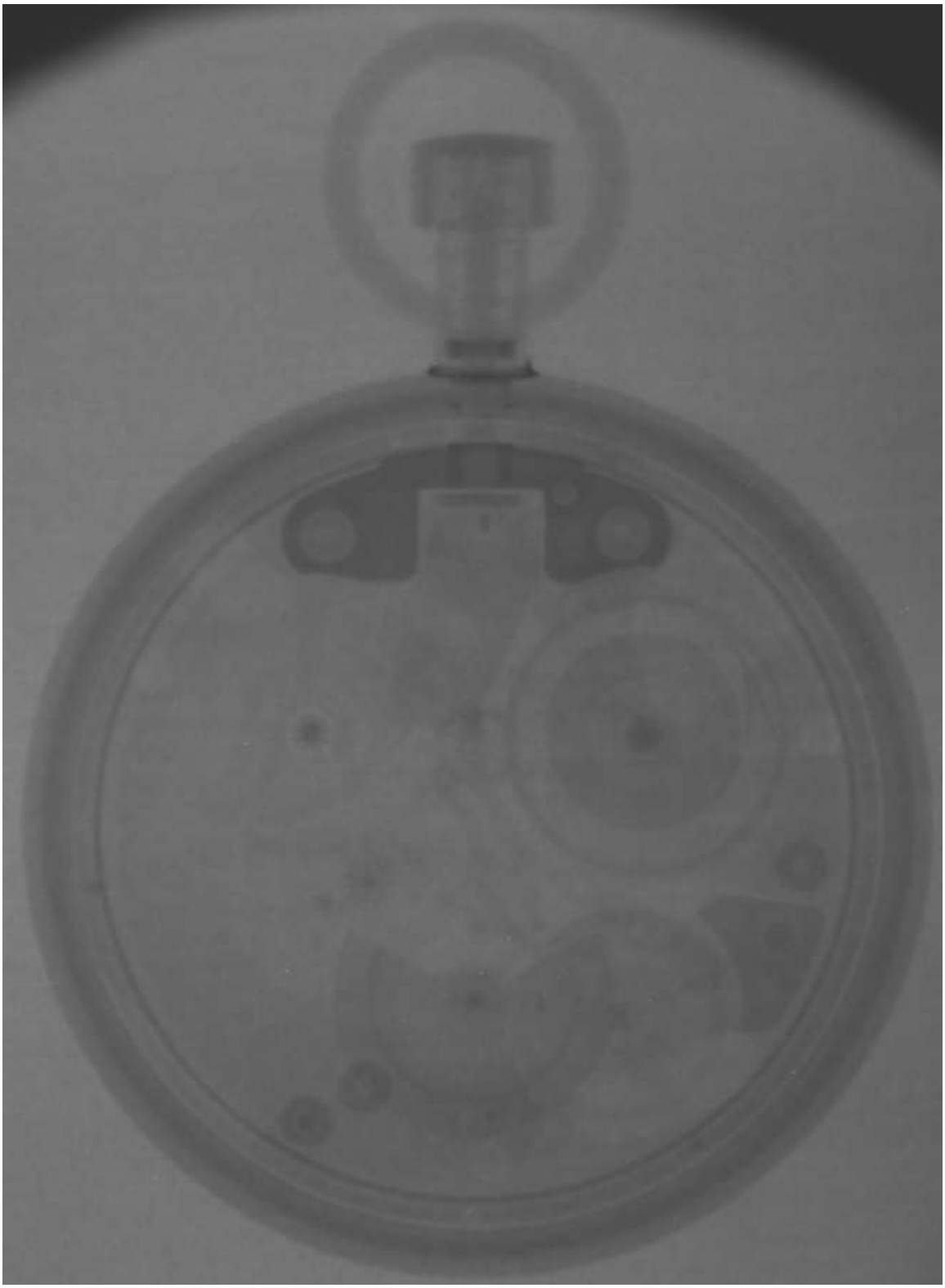


Abbildung 5.5: Abbildung einer Stoppuhr zur Veranschaulichung des Auflösungsvermögens der neuen Neutronenradiographieanlage

5.1.5 Wasserstoffisotope

Die Neutronenradiographie bietet nicht nur den Vorteil der zerstörungsfreien Analyse von Proben und der damit verbundenen Abbildung der entsprechenden Kerneigenschaften des jeweiligen Elements in der Probe, sondern auch die Möglichkeit, Isotope eines Elements zu unterscheiden. Da die Neutronen mit dem Kern interagieren und die Kerneigenschaften sich nicht nur für jedes Element unterscheiden, sondern auch für jedes Isotop eines Elements, sind isotopenselektive Aufnahmen möglich. Sehr gut veranschaulicht wird das in Abbildung 5.6. Die Aufnahme zeigt zwei Kapseln, welche mit Wasser gefüllt sind. Dabei unterscheiden sich die beiden darin, dass die linke Kapsel mit leichtem und die rechte Kapsel mit schwerem Wasser gefüllt ist. Bei leichtem Wasser besteht der Wasserstoffkern aus einem Proton, bei schwerem Wasser aus einem Proton und einem Neutron. Der Absorbtionswirkungsquerschnitt für thermische Neutronen von leichtem Wasserstoff ist mit 332 mBarn [14] um mehr als Faktor 1000 größer als jener von schwerem Wasserstoff mit 500 µBarn [14]. Daher ist in Abbildung 5.6 die linke, mit leichtem Wasser gefüllte Kapsel deutlich dunkler als die rechte mit schwerem Wasser. Die Tatsache, dass nicht nur verschiedene Elemente, sondern auch Isotope unterschieden werden können, macht die Neutronenradiographie zu einem einzigartigen Werkzeug für die Abbildung von Objekten und Proben.

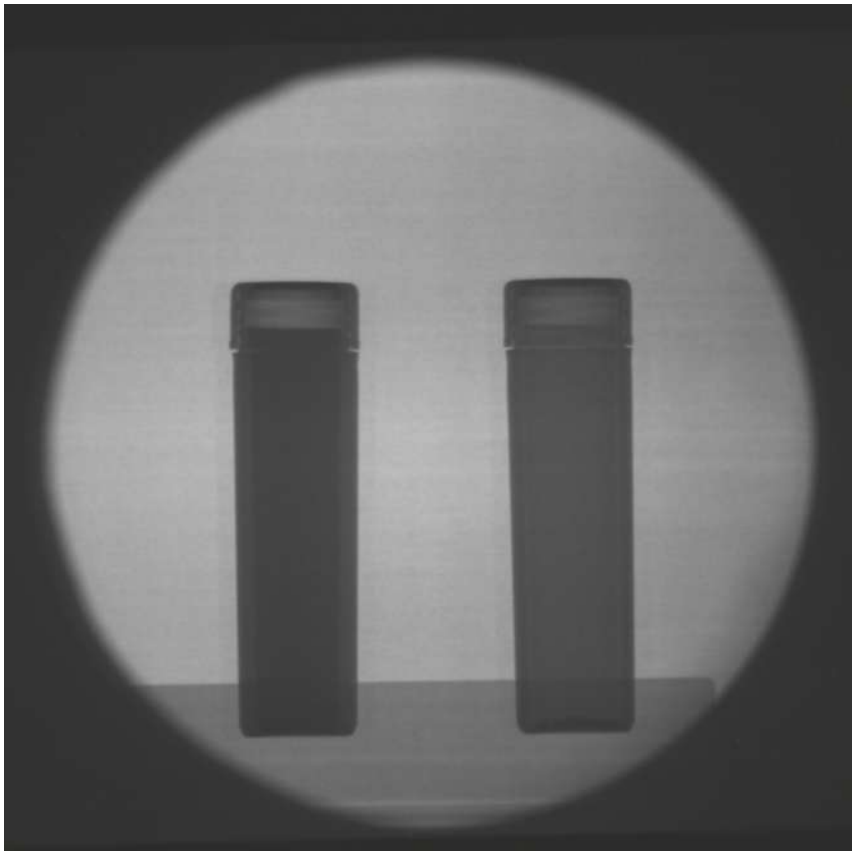


Abbildung 5.6: Abbildung einer H_2O- und einer D_2O-gefüllten Kapsel, um die Selektivität für Isotope bei der Abbildung mit Neutronen zu veranschaulichen

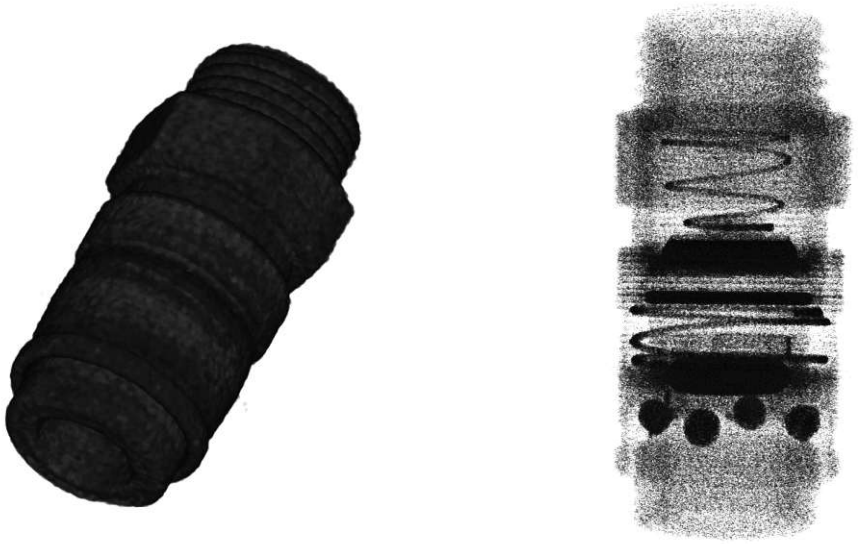
Parameter	Wert
image type:	16-bit gray level
resolution:	4800x4800
x_0:	2000
y_0:	600
width:	4800
height:	4800
bandwidth:	40
exposure time:	600 s
gain:	200
gamma:	50
brightness:	80
flip:	vertical
target:	−20,0 °C
temperature:	−19,8 °C

Tabelle 5.5: Konfiguration der Messsoftware zur Aufnahme der H_2O- und D_2O-Kapseln

5.2 Tomographische Messungen

5.2.1 Druckluftkuppelbuchse

Die neue Anlage ermöglicht auch Tomographieaufnahmen als mögliche Messapplikation. Die Probe wird dazu im Strahl mit Hilfe des Drehtisches schrittweise rotiert. Bei jedem Schritt wird eine Aufnahme gemacht. So entstehen Projektionen der Probe unter verschiedenen Winkeln. Im konkreten Fall war der erste Versuch die Erstellung einer Tomographieaufnahme einer Druckluftkuppelbuchse. Für diese entstanden 300 Projektionen der Buchse mit einer Winkelschrittweite von 1,2°. Mit einer Aufnahmezeit von 90 s konnten alle Projektionen an einem Reaktorbetriebstag erstellt werden. Nach Aufnahme der Projektionen wurde der Kontrast der Bilder mit Hilfe einer Software optimiert. Anschließend wurde die Druckluftkuppelbuchse mit dem Programm *MuhRec* [9] rekonstruiert. *MuhRec* ist ein Softwarepaket von Anders P. Kaestner am Paul Scherrer Institut in der Schweiz. Die Software ermöglicht die Berechnung von tomographischen Rekonstruktionen in wenigen Schritten. Dieser Umstand und die Tatsache, dass es sich um Freeware handelt, machen *MuhRec* zum perfekten Instrument im universitären Umfeld. Nach der Rekonstruktion liegen die Daten der Probe nun als Schichtbilder vor und können weiterverarbeitet werden. Dabei werden das Rauschen entfernt und die Konturen geglättet. Für diesen Zweck kommt die Anwendung *ImageJ* oder *Fiji* zum Einsatz. Nachdem dort die Bilder als *Image Sequence* geladen sind, können verschiedene Filter und Funktionen angewendet werden, um die Daten zu verbessern. Für die Rekonstruktion der Druckluftkuppelbuchse kamen die Funktionen *Remove Outliers* und *Despecle* sowie der Filter *3D Median* zum Einsatz. *ImageJ* bzw. *Fiji* bieten auch mit dem Plugin *3D_Viewer* eine komfortable Möglichkeit, die Schichtbilder als 3D-Modell darzustellen. Abbildung 5.7a zeigt das 3D-Modell der Druckluftkuppelbuchse. Im *3D_Viewer* kann auch die Transparenz des



(a) 3D Modell (b) Innere struktur

Abbildung 5.7: Rekonstruktion der Druckluftkuppelbuchse

Modells verändert und so die innere Struktur der Probe dargestellt werden. Diese ist für die Druckluftkuppelbuchse in Abbildung 5.7b ersichtlich und besteht aus zwei Kunststoffdichtungen, Spiralfedern und den Verschlusskugeln. Um die Qualität der aufgenommenen Projektionen und der Rekonstruktion zu verdeutlichen, sind in Abbildung 5.8 ein Foto der Druckluftkuppelbuchse und das entstandene 3D-Modell gegenübergestellt.

Abbildung 5.8: Gegenüberstellung Foto mit 3D-Modell der Druckluftkuppelbuchse

6 Ausblick und Entwicklungspotenzial

Die neue Neutronenradiographieanlage am Forschungsreaktor der TU Wien wurde im Rahmen dieser Arbeit geplant und aufgebaut. Nach der Inbetriebnahme und den ersten Tests kann die neue Anlage als voll funktionsfähig angesehen werden. Die Auswertungen der ersten Messungen zeigen sehr gute Ergebnisse bezüglich Auflösung und Qualität der Abbildungen. Dennoch gibt es nach diesen ersten Erfahrungen im Umgang mit der neuen Anlage bereits erste Ideen zur Adaptierung und Verbesserung. Auch der einfache Zugang und das Öffentlichmachen der Quelltexte und Zeichnungen in digitaler Form ist geplant. Somit zeigen sich bereits bei der abschließenden Evaluierung des Projekts konkrete Maßnahmen, um die neu entstandene Radiographieanlage weiterzuentwickeln.

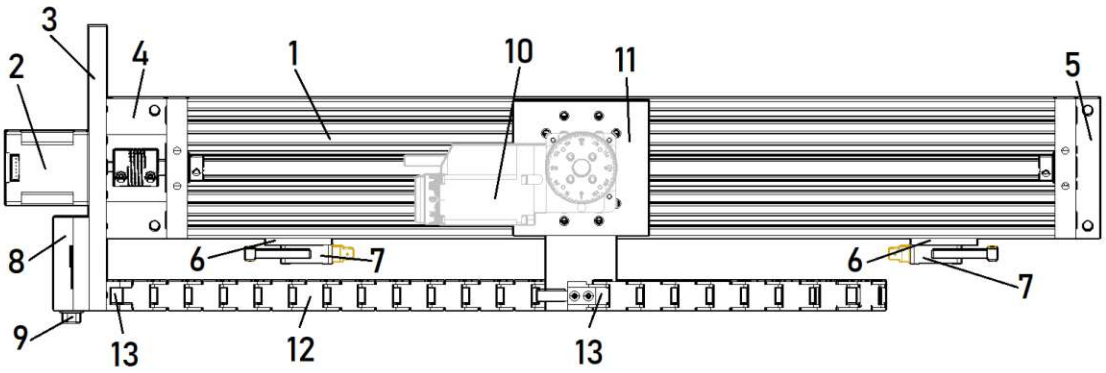
- Vermessung und Charakterisierung des Neutronenstrahls zur optimierten Ausrichtung der Anlage im Strahl
- Bestimmung der räumlichen Auflösung und Empfindlichkeit des vorhandenen Szintillators
- Testen verschiedener Szintillatoren zur Verbesserung von Auflösung und Empfindlichkeit
- Optimierung der Kamerakühlung duch eine externe Luft- oder Wasserkühlung
- Adaptieren der Steuerung, insbesondere der Steuerplatine, um alle Relais des Relais-Moduls nutzen zu können
- Verbessern der Entprellung der Endschalter, um kürzere Ansprechzeiten zu erhalten
- Erstellen einer englischsprachigen Dokumentation für die Steuerung inklusive Arduino Sketch
- Ausführliches Testen der Messsoftware und Weiterentwicklung der Funktionen
- Verbesserung der grafischen Oberfläche der Messsoftware
- Erstellen einer englischsprachigen Dokumentation für imgctrl-Paket und Messsoftware
- Anlegen einer Versionsverwaltung für Arduino Sketch, imgctrl-Paket und Messsoftware
- Hosten der Python-Pakete im Python Package Index

- Weitere Auseinandersetzung mit softwareunterstützter Bildnachbearbeitung und Rekonstruktion

Die gelisteten Maßnahmen sind sicher nur ein Teil des erreichbaren Entwicklungspotenzials, dennoch geben sie einen guten Einblick, in welche Richtungen noch Optimierung möglich ist.
Abschließend lässt sich über die vorliegende Arbeit *Planung und Aufbau einer neuen Neutronenradiographieanlage am TRIGA Mark II Forschungsreaktor der TU Wien* sagen, dass es ein durchwegs erfolgreiches Projekt ist. Die bisherige Neutronenradiographieanlage basierend auf einer flüssigstickstoffgekühlten CCD-Kamera konnte ersetzt werden. Die einzelnen Komponenten der neuen Radiographiestation wurden zum Großteil am 3D-Drucker gefertigt, auch die Elektronik und die Software wurden selbst entwickelt. Die neue CMOS-Astrokamera mit Peltierkühlung ermöglicht deutlich kürzere Experimentierzeiten bei sehr hoher Qualität der Aufnahmen. Die im Laufe dieser Arbeit entstandene Neutronenradiographieanlage konnte bei den ersten Messungen bereits überzeugen. Damit steht der Forschung und Lehre an der TU Wien ein modernes Instrument zur Verfügung. Die neue Neutronenradiographieanlage befindet sich auf dem aktuellen Stand der Technik und ermöglicht eine einfache Bedienung über das benutzerfreundliche Interface.

A Komponenten

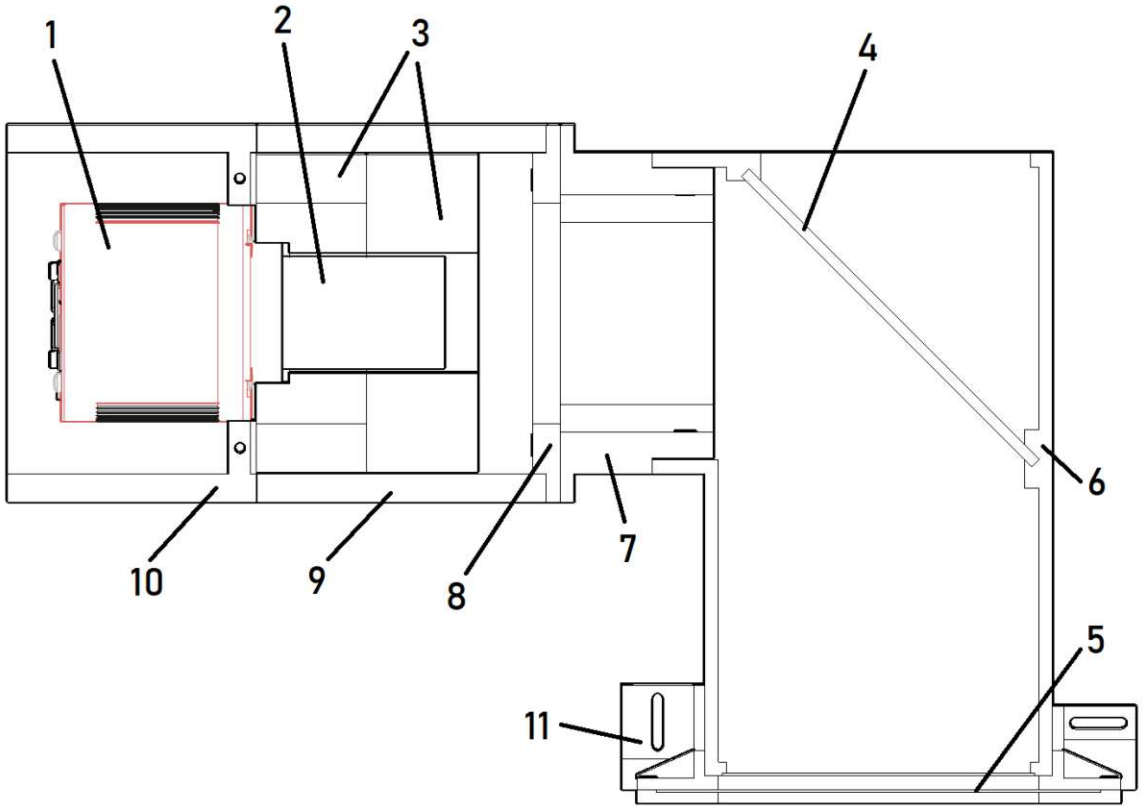
A.1 Probentisch



Nr.	Bauteil	Stk.	Lieferant	Referenz
1	Lineartisch	1	Amazon	Anhang A.4.2
2	Schrittmotor	1	Amazon	Anhang A.4.1
3	Montageplatte	1	Eigenfertigung	Abbildung 4.3
4	Distanzhalter	1	Eigenfertigung	Abbildung C.4
5	Montagesteg	1	Eigenfertigung	Abbildung C.5
6	Endschalterplatte	2	Eigenfertigung	Abbildung C.3
7	Endschalter	2	RS Components	Omron V-166-1C5
8	D-Sub Gehäuse	1	Eigenfertigung	Abbildung C.6
9	D-Sub DE-9 Stecker	1	RS Components	Phoenix Contact 1688379
10	Drehtisch	1	Standa	Anhang A.4.3
11	Adapterplatte	1	Eigenfertigung	Abbildung C.2
12	Schleppkette	1	RS Components	igus 06.10.018.0
13	Halterung für Schleppkette	1	RS Components	igus 06.10.12PZ

Abbildung A.1: Bezeichnung der einzelnen Komponenten mit schematischer Darstellung des Probentisches

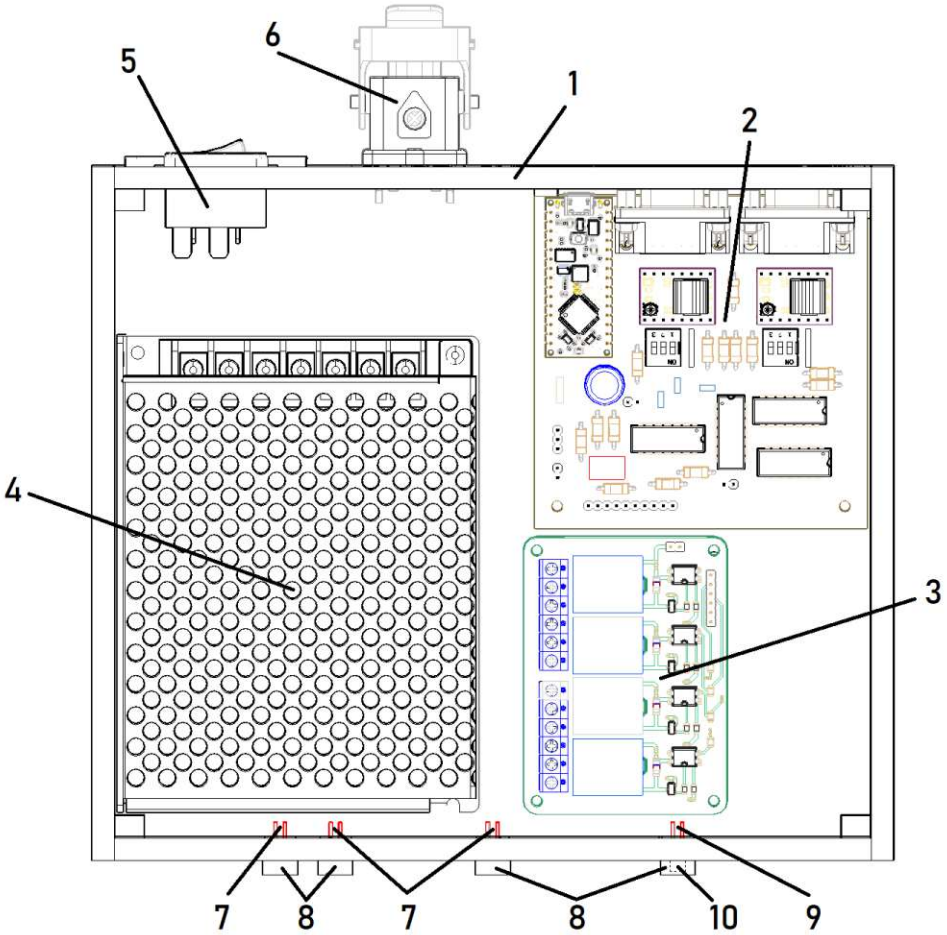
A.2 Detektoreinheit



Nr.	Bauteil	Stk.	Lieferant	Referenz
1	Kamera	1	Amazon	Anhang A.4.6
2	Objektiv	1	Amazon	Anhang A.4.7
3	Bleiabschirmung	1	Eigenfertigung	Anhang C.3
4	Spiegel	1	Baumarkt	
5	Szintillator	1	N/A	Abschnitt 4.6
6	Basisteil Spiegel	1	Eigenfertigung	Abbildung C.14
7	Mittelteil	1	Eigenfertigung	Abbildung C.16
8	Basisplatte Kamera	1	Eigenfertigung	Abbildung C.17
9	Abdeckung	1	Eigenfertigung	Abbildung C.18
10	Klemmdeckel	1	Eigenfertigung	Abbildung C.19
11	Szintillator Halterung	1	Eigenfertigung	Abbildung C.20

Abbildung A.2: Bezeichnung der einzelnen Komponenten mit schematischer Darstellung der Detektoreinheit

A.3 Steuerung



Nr.	Bauteil	Stk.	Lieferant	Referenz
1	Gehäuse	1	Eigenfertigung	Anhang C.2
2	Steuerplatine	1	Eigenfertigung	Anhang B.2
3	Relais Modul	1	az-delivery	4-Relais Modul 5V
4	Schaltnetzteil	1	RS Components	Mean Well RS-75-12
5	Netzstecker	1	RS Components	Bulgin BVA01/Z0000/01
6	Steckverbinder	1	RS Components	Epic Contact 10422500&10421000
7	LED (grün)	3	RS Components	Kingbright THT LED L-53GD
8	LED Sockel	1	RS Components	Kingbright LED-Halter RTF-5020
9	LED (grün/rot)	1	RS Components	Kingbright THT LED L-59EGW-CA
10	Taster	1	RS Components	RS PRO 734-6735

Abbildung A.3: Bezeichnung der einzelnen Komponenten mit schematischer Darstellung der Steuerung

Bauteil	Bezeichnung	Stk.	Referenz
Mikrocontroller-Board	Arduino Nano Every	1	Anhang A.4.4
Motortreiber	DRV8825	1	Anhang A.4.5
Kohleschicht Widerstand ±5 % 0,25 W	1 kΩ	4	
	2,5 kΩ	1	
	22 kΩ	3	
	100 kΩ	3	
	150 kΩ	3	
Elektrolyt Kondensator ±20 % 63 V	1 µF	3	
	1 µF	1	
Diode DO-35	1N914TR	2	
NAND-Gatter	74HCT00N	1	
AND-Gatter	74HCT08N	1	
Schmitt-Trigger invertierend	74HCT14N	1	
OR-Gatter	74HCT32N	1	
Stiftleiste 2.54mm, gerade, 1-reihig	20-polig	1	
DIP-Schalter	3-stellig	2	
Printtaster	6 x6 x5 mm	1	
Printklemme 5 mm	2-polig	1	
D-Sub Steckverbinder abgewinkelt	DE-09 Buchse	1	
	DE-15 Buchse	1	

Tabelle A.1: Auflistung der Bauteile zur Bestückung der Steuerplatine

A.4 Datenblätter

A.4.1 Schrittmotor

Amazon. *Rtelligent 42A02C Schrittmotor*. URL: `https://www.amazon.de/Rtelligent-Schrittmotor-Arduino-Stepping-3D-Druck/dp/B083PWQ887?th=1` (besucht am 07. 02. 2024)

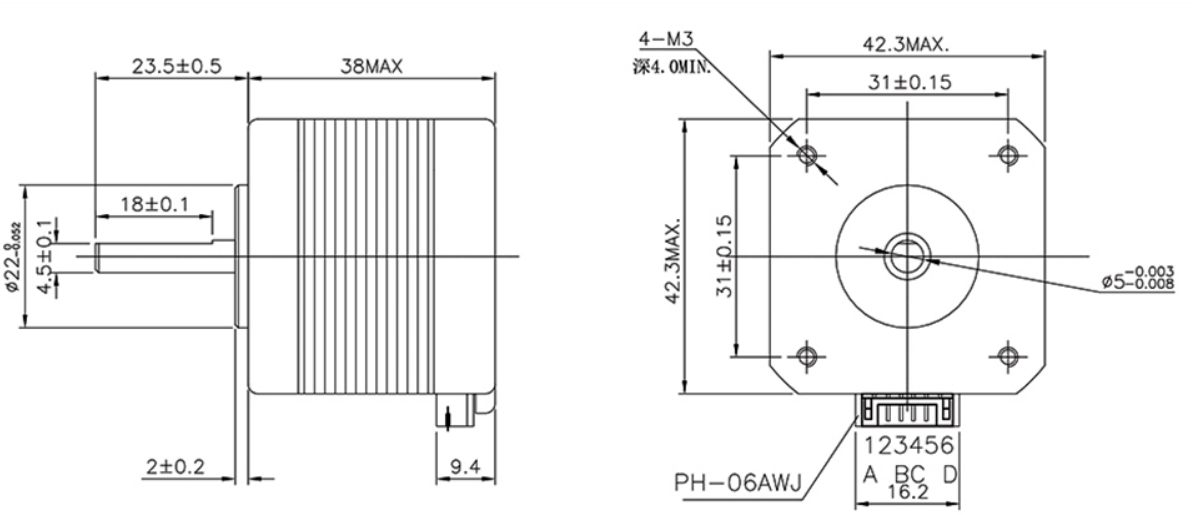


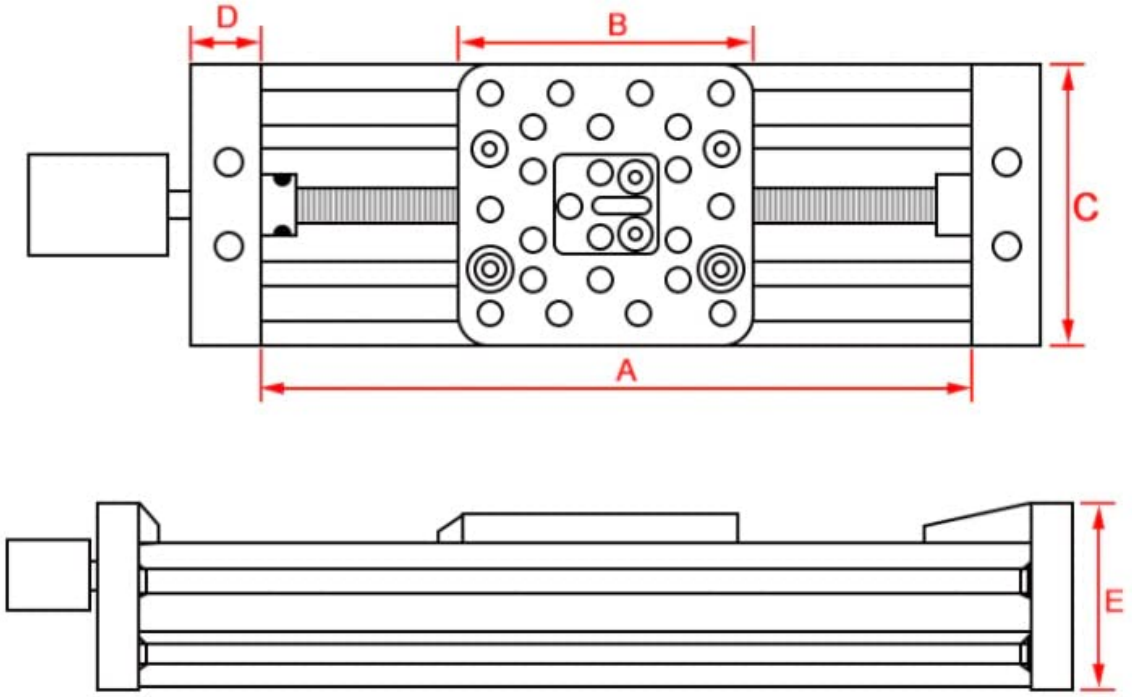
Abbildung A.4: Konstruktionszeichnung Schrittmotor 42A02C

Parameter	Value
Manufacturer	Rtelligent
Model	42A02C
Size	NEMA 17
Phase number	2
Rated voltage	DC 3,6 V
Phase resistance (20°)	2,4 Ω/phase
Holding torque	≥42 N cm
Sterring (shaft extension)	A-AB-B-clockwise
Maximum no-load operating frequency	≥1900 PPS
Electrical strength	AC 600 V mA^{-1} s^{-1}
Moment of inertia	57,3 g cm^2
Step angle	1,80 ± 0,09°
Rated current	DC 1,5 A/phase
Phase inductance (1 kHz)	3,7 mH/phase
Positioning torque	1,5 N cm REF.
Maximum no-load starting frequency	≥1500 PPS
Insulation resistance	≥100 MΩ (DC 500 V)
Insulation grade	Class B
Quality	255 g REF.

Tabelle A.2: Datenblatt Schrittmotor

A.4.2 Lineartisch

Amazon. *Linear Table with Screw Slide*. Silber 400mm. URL: `https://www.amazon.de/dp/B089D9D9SG/ref=sspa_dk_detail_0?pd_rd_i=B089D9D9SG&pd_rd_w=srTus&content-id=amzn1.sym.ae2317a0-2175-4285-af64-66539858231f&pf_rd_p=ae2317a0-2175-4285-af64-66539858231f&pf_rd_r=3D0F03ZVQWNEWKPBB2XF&pd_rd_wg=zOGJZ&pd_rd_r=56a21082-8965-460b-b04c-b81fca17b97e&s=industrial&sp_csd=d2lkZ2V0TmFtZT1zcF9kZXRhaWw&smid=A0867S1490VMY&th=1` (besucht am 07. 02. 2024)



Effective journey	A	B	C	D	E
308,85 mm	400 mm	77,15 mm	80 mm	12 mm	55 mm

Abbildung A.5: Abmessungen des Lineartisches

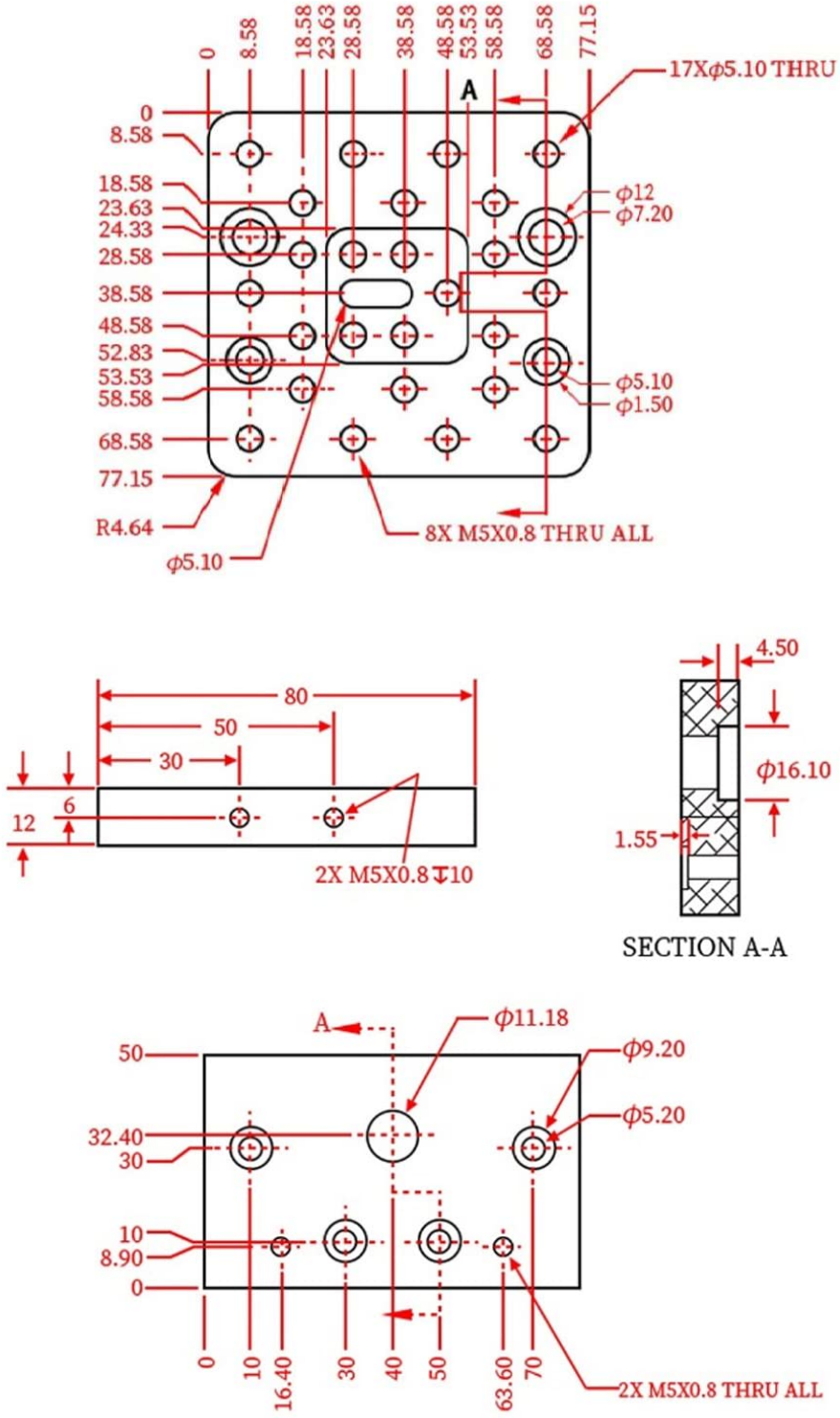


Abbildung A.6: Detailzeichnungen der Lineartischkomponenten

A.4.3 Drehtisch

Standa Ltd. *8MR174-11 - Motorized Rotation Stage*. URL: `https://www.standa.lt/products/catalog/motorised_positioners?item=68` (besucht am 07. 02. 2024)

Parameter	Value
Manufacturer	Standa LTd.
Model	8MR174-11-28
Rotation Range	360°
Resolution in full step	0,9′ (0,015°)
Uni-directional repeatability	0,4′
Bi-directional repeatability	1,6′
Wobble	1′
Eccentricity	10 µm10 µm
Load capacity horizontal	4 kg
Load capacity radial	1,5 kg
Torque	0,5 N m
Integrated cable length	1,6 m
Motor connector	DE-15(M)
Stepper motor	bipolar, 200 (1,8°) steps, 6,8 Ω, 0,67 A
Weight	0,4 kg
Mechanical home reference switch	pushed is "closed"

Tabelle A.3: Datenblatt Drehtisch

Pin	Funktion
1	B2
2	B1
3	A1
4	A2
5-7	n.c.
8-9	Taster Referenzposition
10-15	n.c.

Tabelle A.4: Pinbelegung des D-Sub DE-15 Steckers am Drehtisch

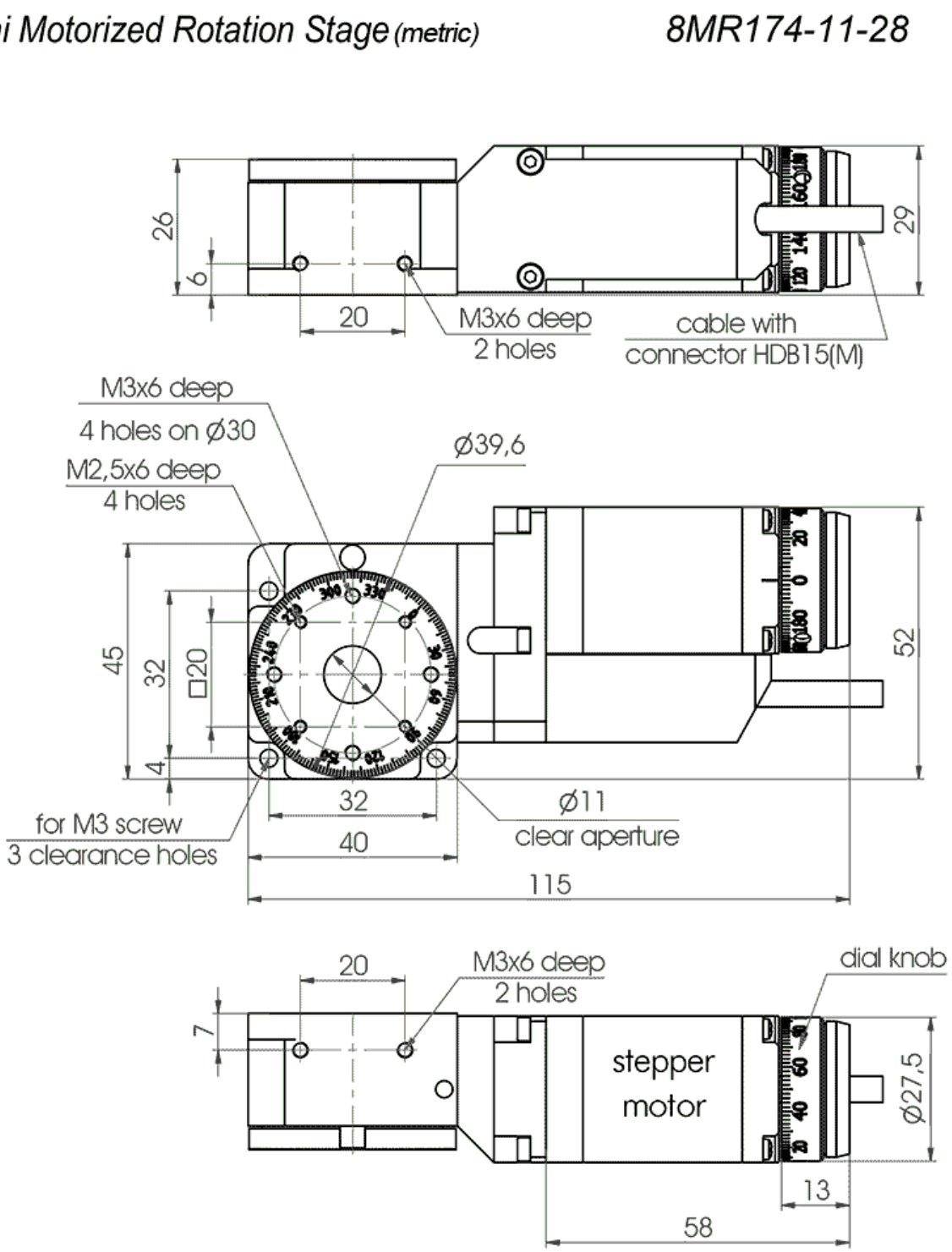


Abbildung A.7: Konstruktionszeichnung Drehtisch Standa 8MR174-11-28

A.4.4 Mikrocontroller Board

Arduino DOCS. *Arduino Nano Every*. URL: `https://docs.arduino.cc/hardware/nano-every` (besucht am 07. 02. 2024)

Board	Name	Arduino Nano Every
	SKU	ABX00028
Microcontroller	ATMega4809	
USB connector	Micro USB	
Pins	Built-in LED Pin	13
	Digital I/O Pins	14
	Analog input pins	8
	PWM pins	5
Communication	UART	RX/TX
	I2C	A4 (SDA), A5 (SCL)
	SPI	D11 (COPI), D12 (CIPO), D13 (SCK). Use any GPIO for Chip Select (CS).
Power	I/O Voltage	5V
	Input voltage (nominal)	7-18V
	DC Current per I/O Pin	15 mA
Clock speed	Processor	ATMega4809 16 MHz
Memory	ATmega328P	6 kB SRAM, 48 kB flash, 256 B EEPROM
Dimensions	Weight	5 g
	Width	8 mm
	Length	45 mm

Tabelle A.5: Daten Arduino Nano Every

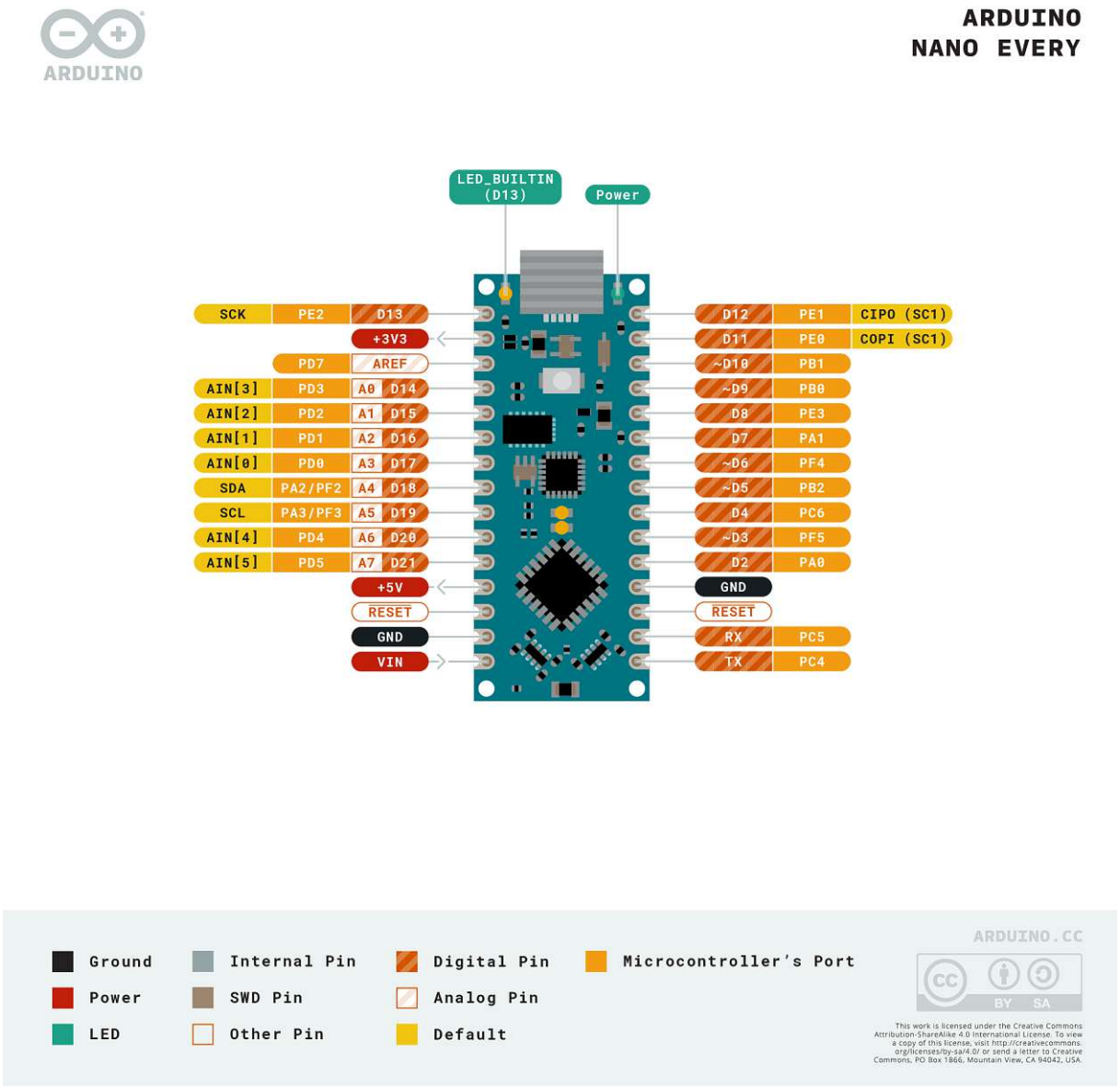


Abbildung A.8: Arduino Nano Every Pinout

A.4.5 Motortreiber Modul

Pololu Corporation. *DRV8825 Stepper Motor Driver Carrier*. URL: `https://www.pololu.com/product/2133` (besucht am 07. 02. 2024)

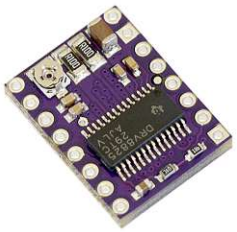


Abbildung A.9: DRV8825 Driver Module

Parameter	Value
Model	DRV8825
Minimum operating voltage	8.2 V
Maximum operating voltage	45 V
Continuous current per phase	1.5 A2
Maximum current per phase	2.2 A3
Minimum logic voltage	2.5 V4
Maximum logic voltage	5.25 V4
Microstep resolutions	full, 1/2, 1/4, 1/8, 1/16, and 1/32

Tabelle A.6: Daten DRV8825 Driver Module

MODE0	MODE1	MODE2	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	1/4 step
High	High	Low	1/8 step
Low	Low	High	1/16 step
High	Low	High	1/32 step
Low	High	High	1/32 step
High	High	High	1/32 step

Tabelle A.7: Microstepping-Modi des DRV8825

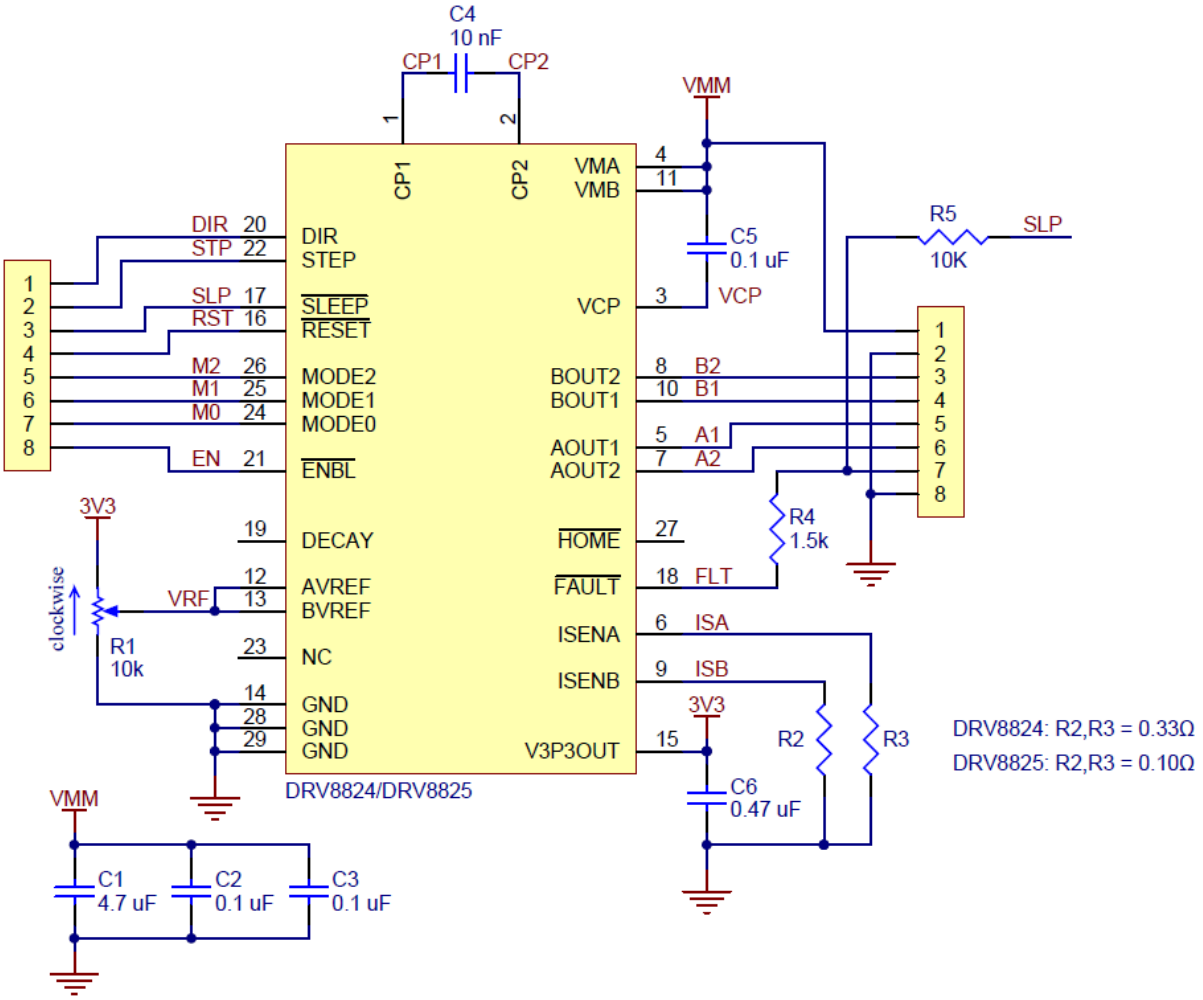


Abbildung A.10: Interner Aufbau des DRV8825 Driver Module

A.4.6 Astronomiekamera

ZWO Company. *ASI294MM Pro*. URL: `https://astronomy-imaging-camera.com/product/asi294mm-pro/` (besucht am 07. 02. 2024)

Parameter	Value
Manufacturer	ZWO Company
Model	ASI294MM Pro
Sensor	SONY IMX492 CMOS
Diagonal	23,2 mm
Resolution	11,7 MPixel 4144 × 2822 Pixel
Pixel size	4,63 µm
Image area	19,2 × 13 mm
Max FPS at full resolution	19 FPS
Shutter	rolling shutter
Exposure range	$3{,}2 \cdot 10^{-6}$ bis 2000 s
Read noise	$1{,}2 \cdot 10^{-8}$
QE peak	about 90 %
Full well	66,5 ke
ADC	14 bit
DDR3 buffer	256 MB
Interface	USB3.0/USB2.0
Adapters	T2 M42 × 0,75)
Protect window	AR window
Dimensions	78 mm Diameter
Weight	410 g
Back focus distance	6,5 mm
Cooling	regulated two stage TEC
Delta T	35 °C - 40 °C (based on 30°C ambient temperature)
Cooling power consumption	max 3 A at 12 V
Support OS	Windows, Linux & Mac OSX
Maximum working temperature	40 °C
Storage temperature	−10 °C ~ 60 °C
Working relative humidity	20 % ~ 80 %
Storage relative humidity	20 % ~ 95 %

Tabelle A.8: Datenblatt Astronomiekamera

Abbildung A.11: Rückansicht der Astronomiekamera ZWO ASI294MM Pro mit Anschlüssen

A.4.7 Objektiv

AliExpress.com. *Fujian CCTV 35mm f1.7 Lens C Mount*. URL: https://de.aliexpress.com/item/1005005143329809.html?spm=a2g0o.productlist.main.7.4ea7594ctfUI67&algo_pvid=77015846-ba2d-49b5-af52-d5707f3bcbd7&algo_exp_id=77015846-ba2d-49b5-af52-d5707f3bcbd7-3&pdp_npi=4%40dis%21EUR%2131.24%2123.43%21%21%2132.84%2124.63%21%402101c5b117072990541046338ef8c8%2112000035018195147%21sea%21AT%213436433299%21&curPageLogUid=VmPaTlru0PZX&utparam-url=scene%3Asearch%7Cquery_from%3A (besucht am 07. 02. 2024)

Parameter	Value
Manufacturer	Fujian
Model	35MM F/1.7 CCTV Lens
Focal length	35 mm
Aperture	$f/1{,}7$
Angle of view	13,8°
Object distance	30 cm - ∞
Size	40 mm × 59 mm
Mount type	C-Mount (1″ - 32)

Tabelle A.9: Datenblatt Objektiv

Abbildung A.12: Kameraobjektiv 35MM F/1.7 CCTV Lens

B Schaltpläne

B.1 Steuerung

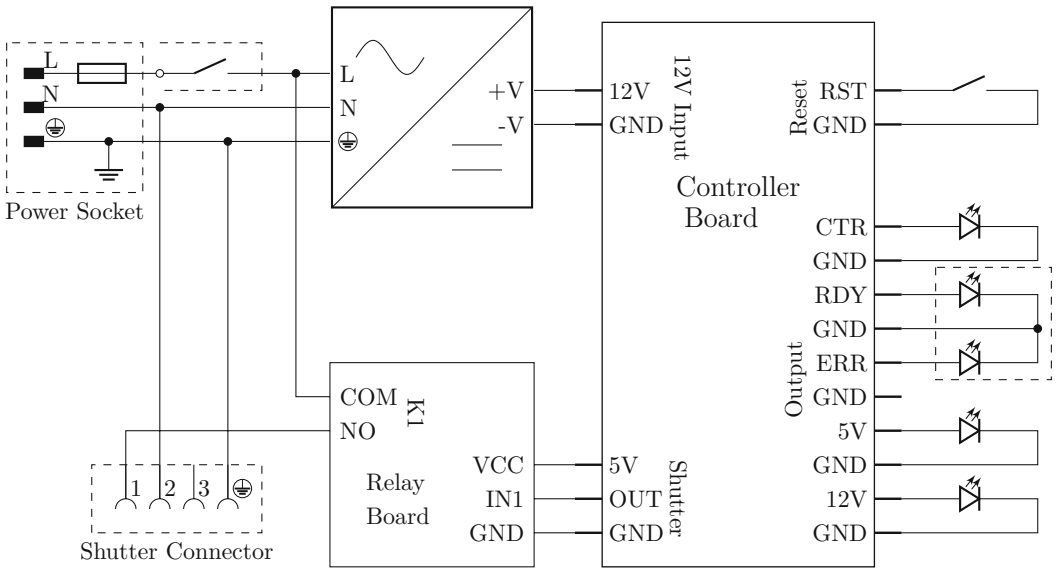


Abbildung B.1: Schaltplan der Steuerung mit den Verbindungen zu den einzelnen Komponenten

B.2 Steuerplatine

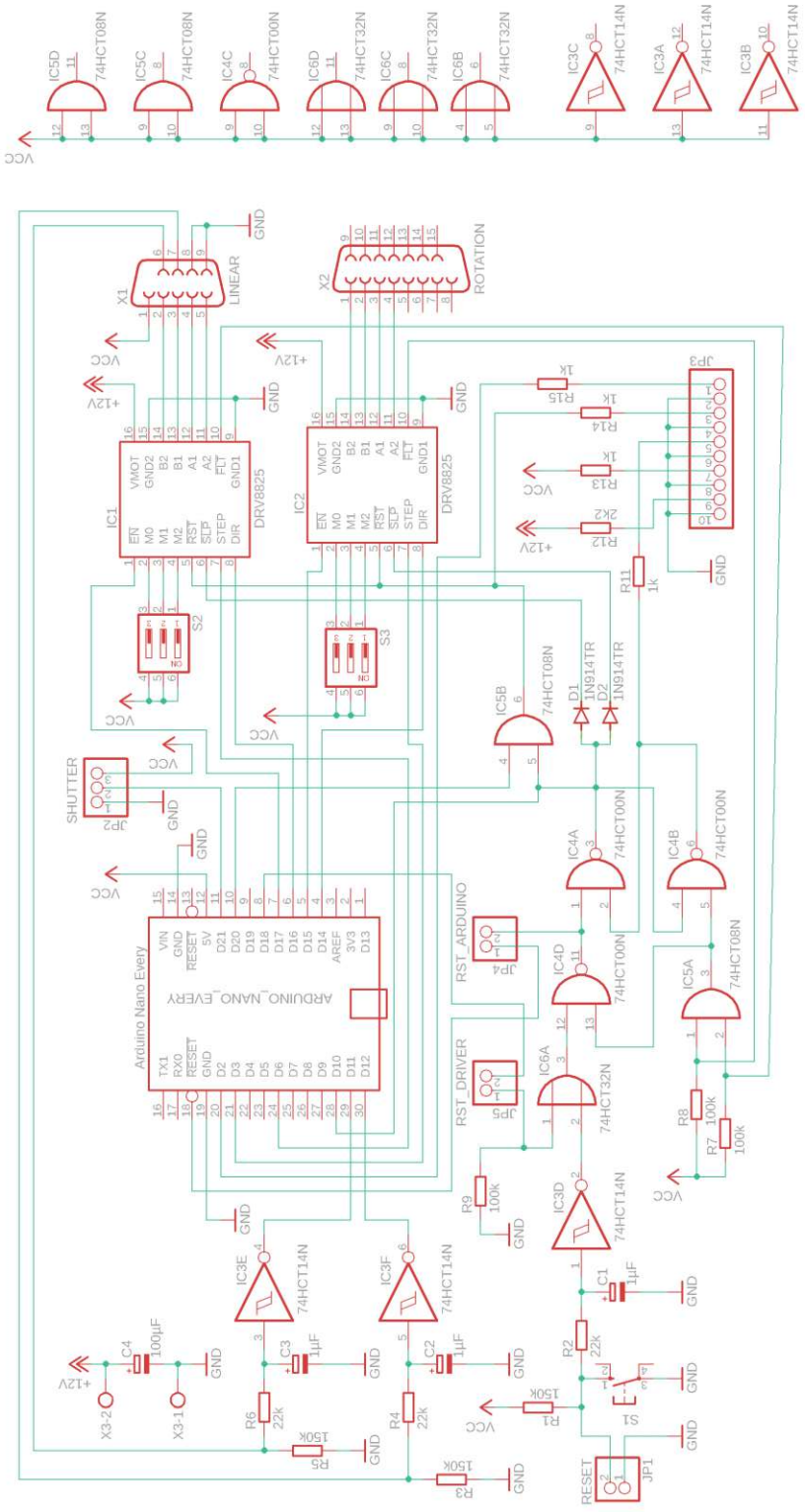


Abbildung B.2: Schaltung der Steuerplatine

Bauteil	Wert/Typ
C1	1 µF
C2	1 µF
C3	1 µF
C4	100 µF
D1	1N914TR
D2	1N914TR
IC1	DRV8825
IC2	DRV8825
IC3	74HCT14N
IC4	74HCT00N
IC5	74HCT08N
IC6	74HCT32N
JP1	Stiftleiste 1x02
JP2	Stiftleiste 1x03
JP3	Stiftleiste 1x10
JP4	Stiftleiste 1x02
JP5	Stiftleiste 1x02
R1	150 kΩ
R2	22 kΩ
R3	150 kΩ
R4	22 kΩ
R5	150 kΩ
R6	22 kΩ
R7	100 kΩ
R8	100 kΩ
R9	100 kΩ
R11	1 kΩ
R12	2,2 kΩ
R13	1 kΩ
R14	1 kΩ
R15	1 kΩ
S1	Printtaster
S2	DIP-Schalter
S3	DIP-Schalter
X1	D-Sub DE-9
X2	D-Sub DE-15
X3	Printklemme

Tabelle B.1: Bestückungsliste Steuerplatine

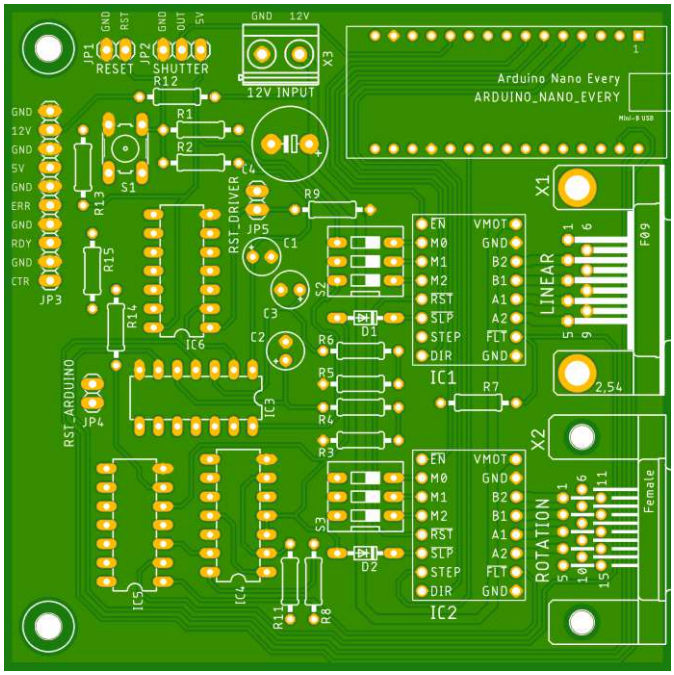


Abbildung B.3: Oberansicht der Steuerplatine

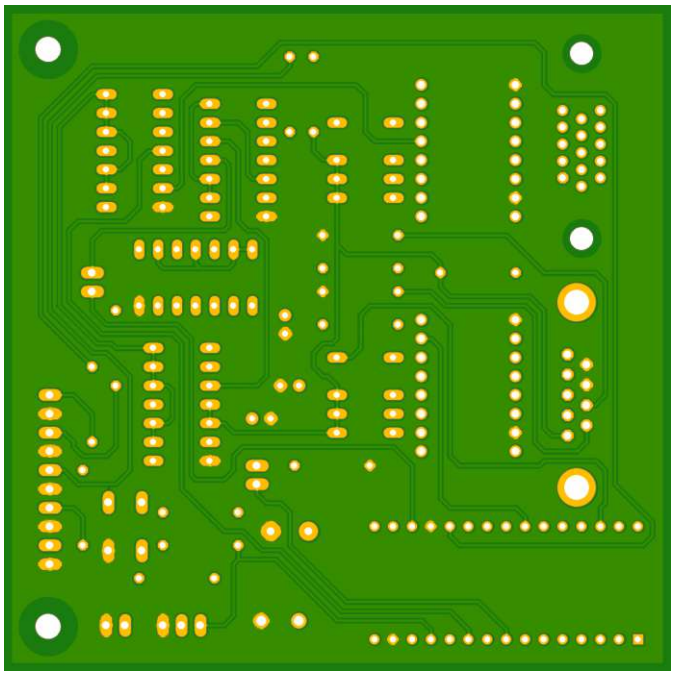


Abbildung B.4: Unteransicht der Steuerplatine

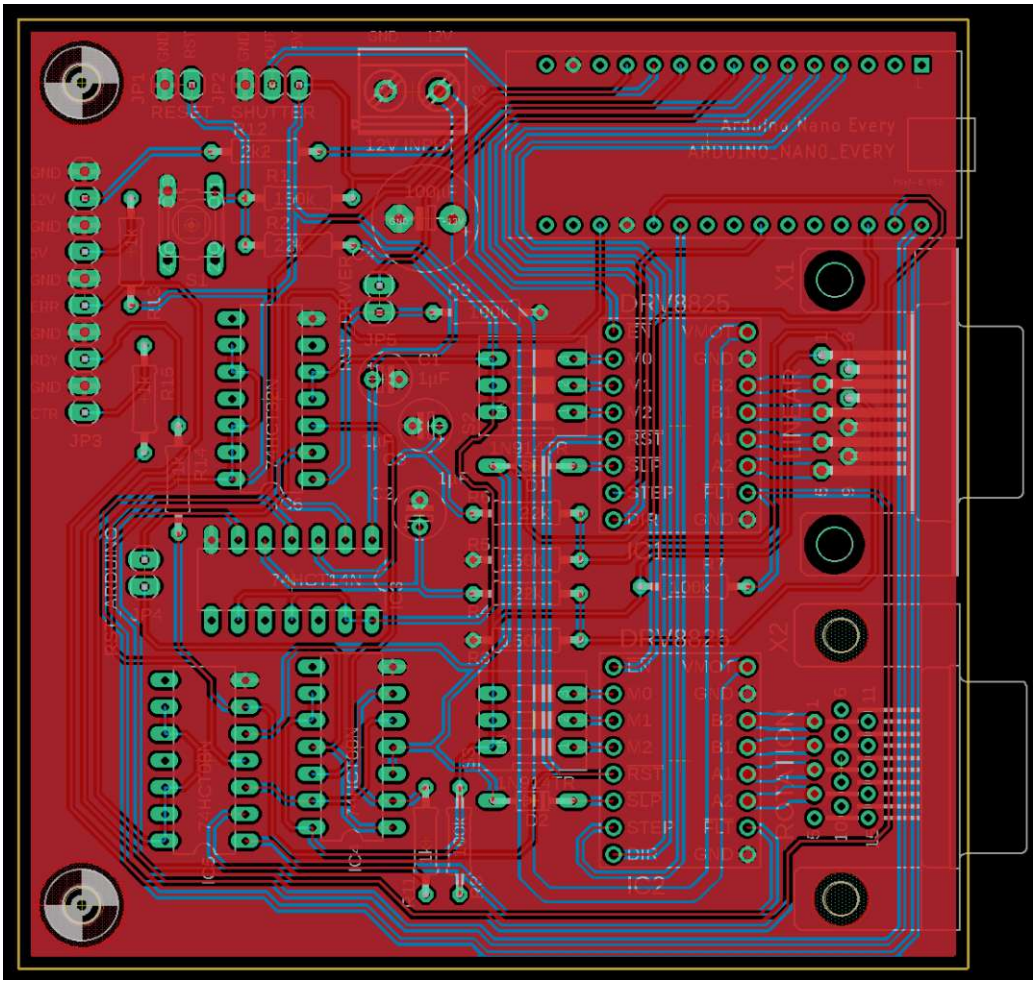


Abbildung B.5: Layout der Steuerplatine mit Top und Bottom Layer

Anschluss	Beschreibung
12V	12 V-Versorgung
5V	5 V-Versorgung für das Relais Modul
OUT	Shuttersignal für das Relais Modul
RST	Reseteingang für den Fehlerspeicher
12V	LED 12 V-Versorgung bereit
5V	LED 5 V-Versorgung bereit
ERR	Störung Motortreiber vorhanden
RDY	Betriebsbereit
CTR	Arduino bereit
LINEAR	Anschluss für den Lineartisch Tabelle B.3
ROTATION	Anschluss für den Rotationstisch Tabelle B.4

Tabelle B.2: Beschreibung der Anschlüsse der Steuerplatine

Pin	Funktion
1	5 V
2	B2
3	B1
4	A1
5	A2
6	D11
7	D12
8	GND
9	GND

Tabelle B.3: Pinbelegung der D-Sub DE-9 Steckverbindung für die Linearbewegung

Pin	Funktion
1	B2
2	B1
3	A1
4	A2
5-15	n.c.

Tabelle B.4: Pinbelegung der D-Sub DE-15 Steckverbindung für die Rotationsbewegung

C Konstruktionszeichnungen

C.1 Probentisch

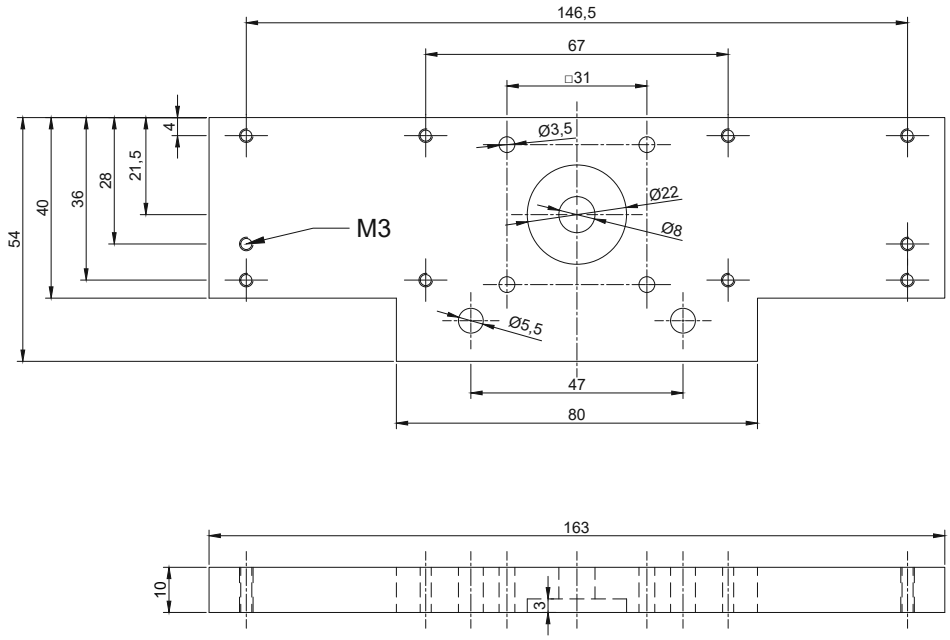


Abbildung C.1: Montageplatte für Motor, Steckergehäuse und Schleppkette

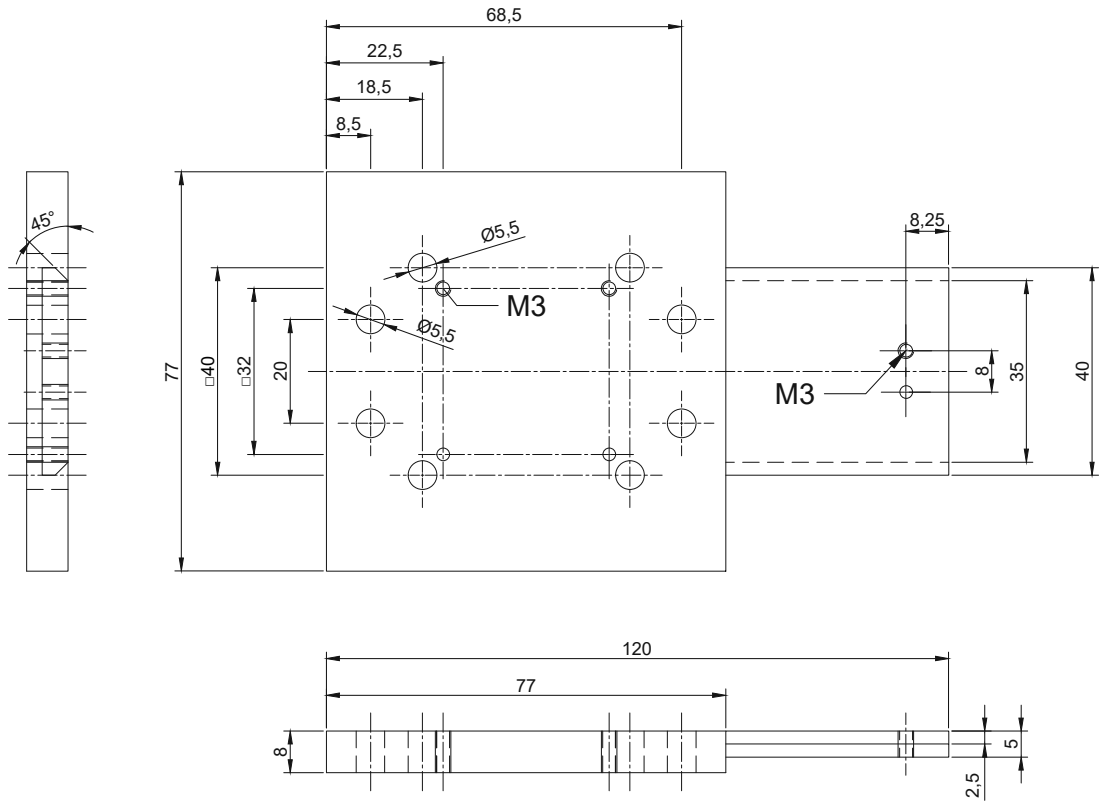


Abbildung C.2: Adapterplatte zwischen Drehtisch und Schlitten

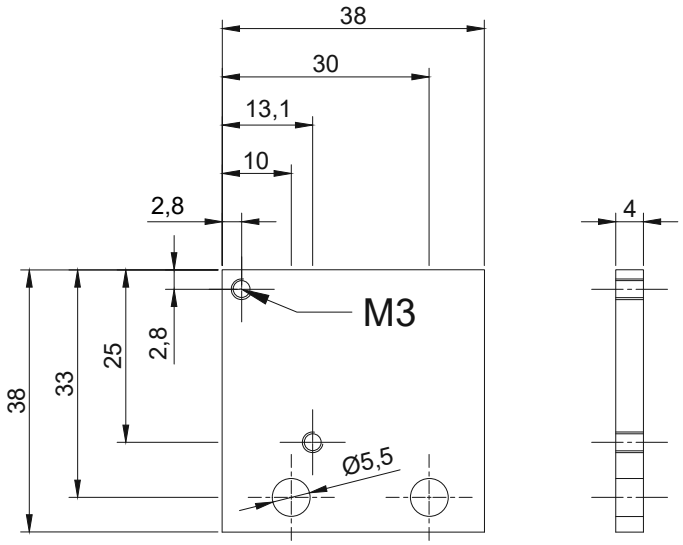


Abbildung C.3: Platte zur Befestigung des Endschalters

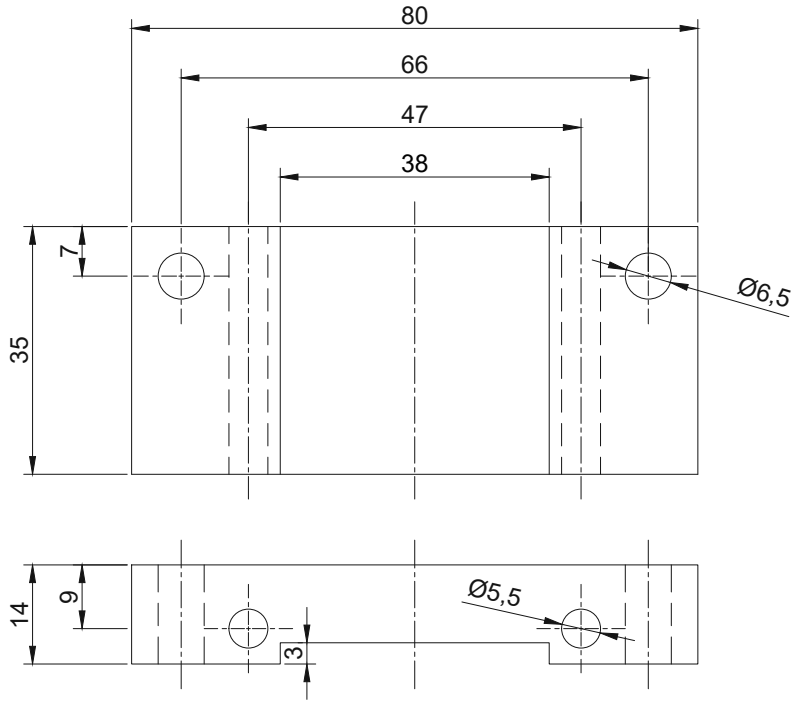


Abbildung C.4: Distanzhalter zwischen Montageplatte und Lineartisch für die Kupplung

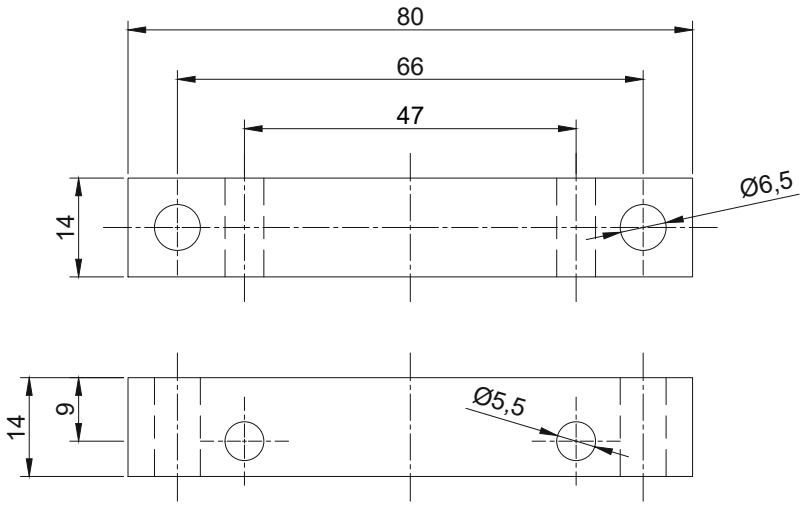


Abbildung C.5: Montagesteg für den Lineartisch

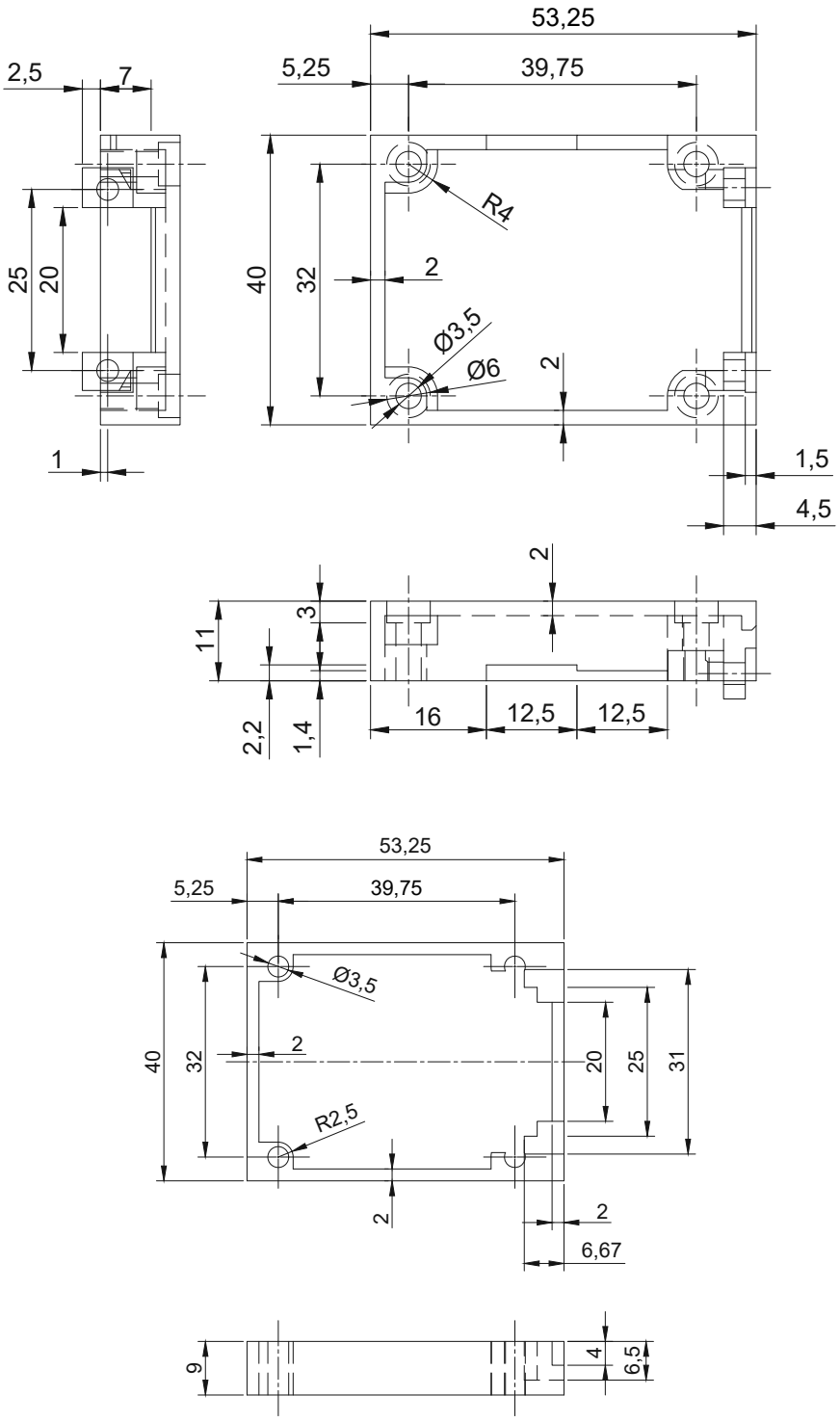


Abbildung C.6: Gehäuse für den D-Sub DE-9 Stecker, bestehend aus zwei Teilen

C.2 Gehäuse Steuerung

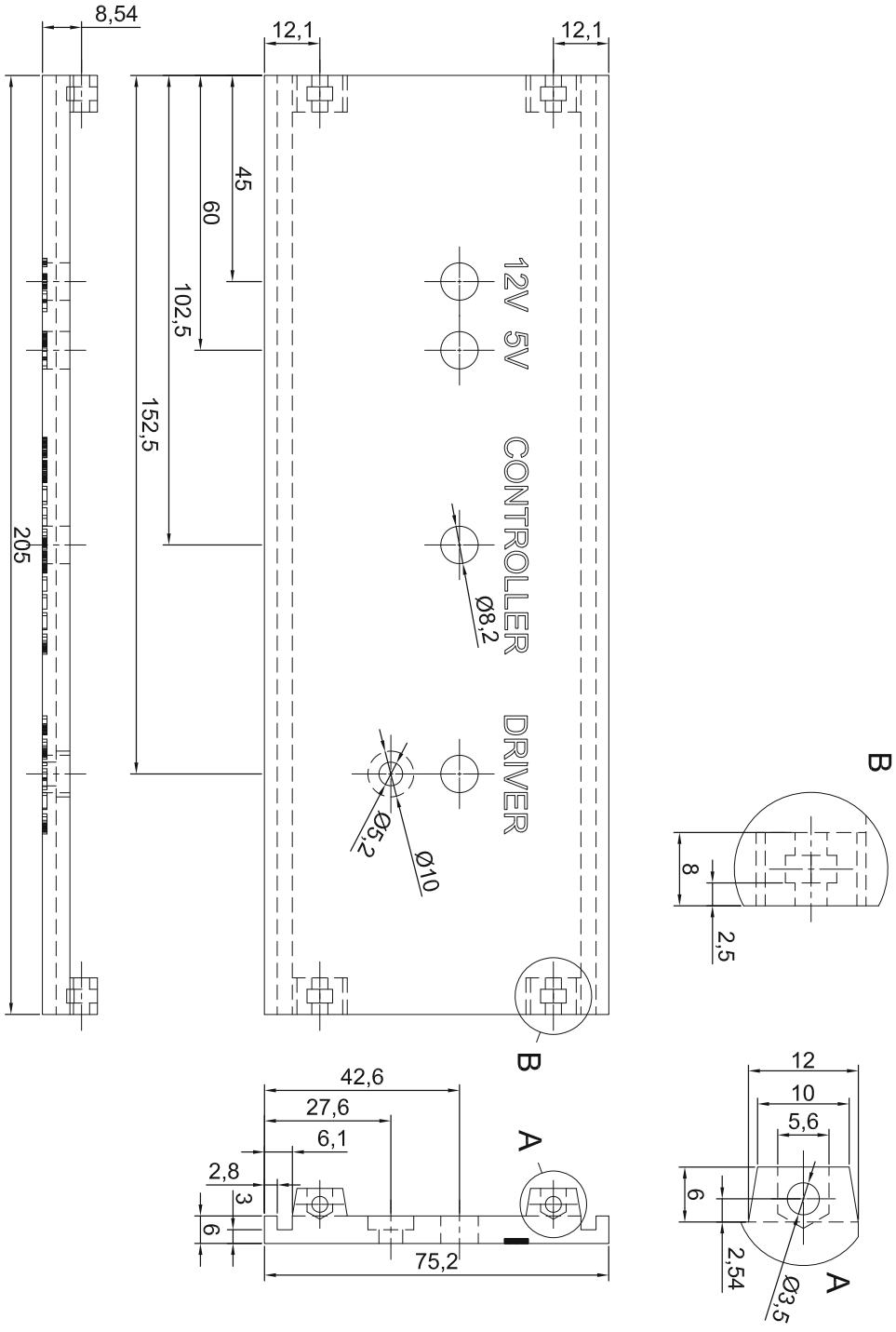


Abbildung C.7: Vorderseite des Steuerungsgehäuses mit Löchern für die LED Sockel und den Taster

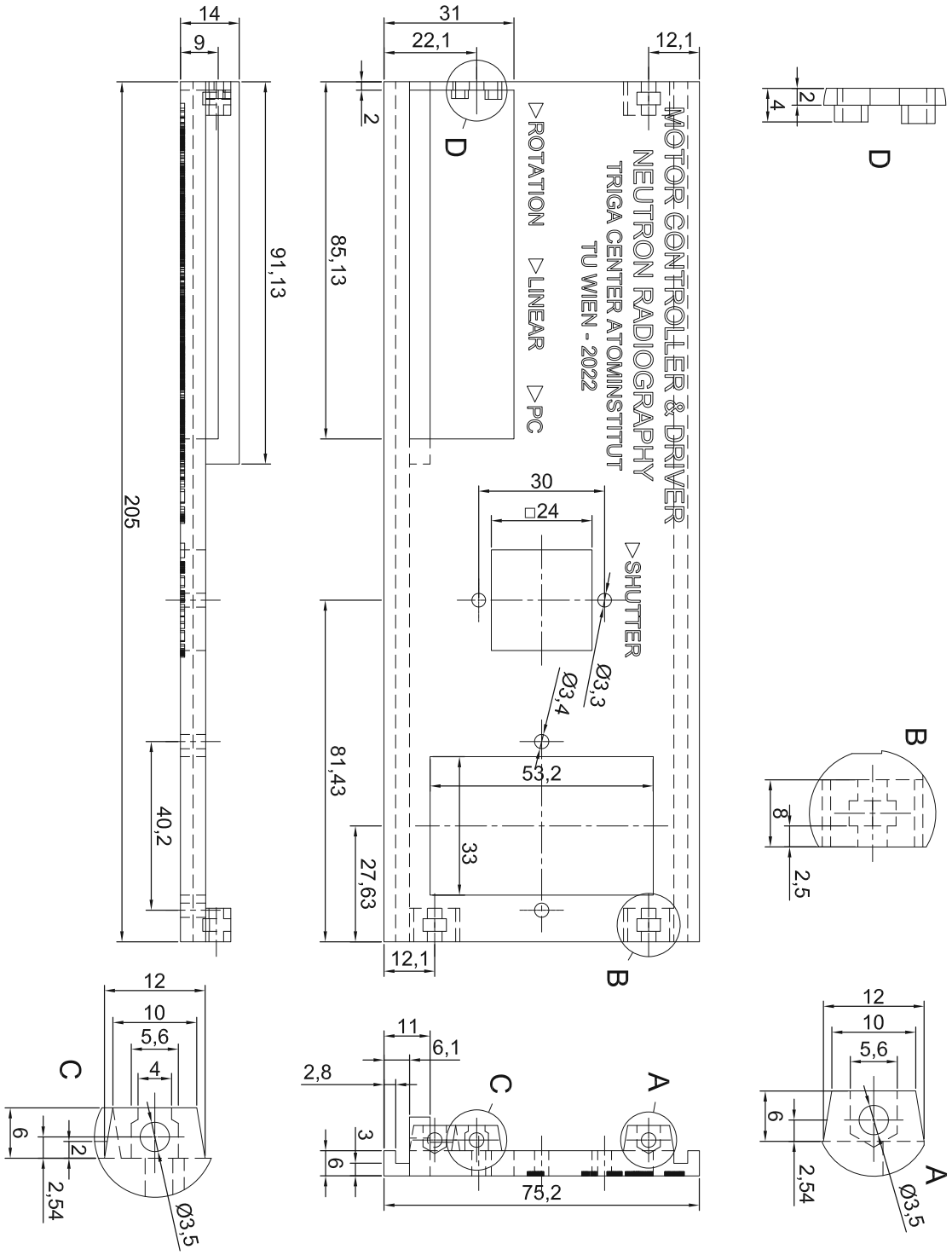


Abbildung C.8: Rückseitiges Paneel des Steuerungsgehäuses mit Aussparungen für den Netzstecker, den Steckverbinder für den Shutter und die Steuerplatine

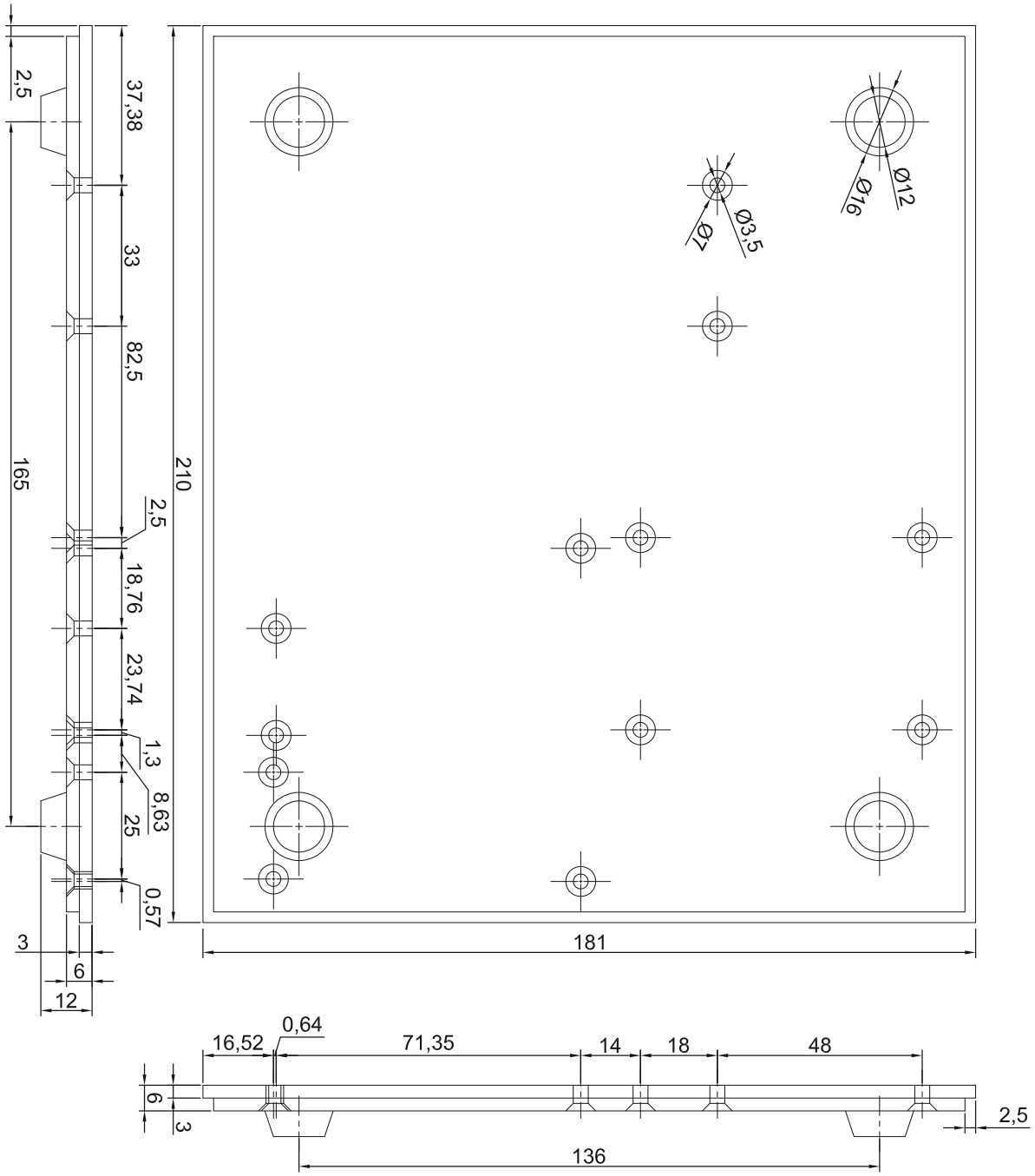


Abbildung C.9: Unterseite des Steuerungsgehäuses mit Löchern für die Montage der Steuerplatine, des Netzteils und des Relais Moduls

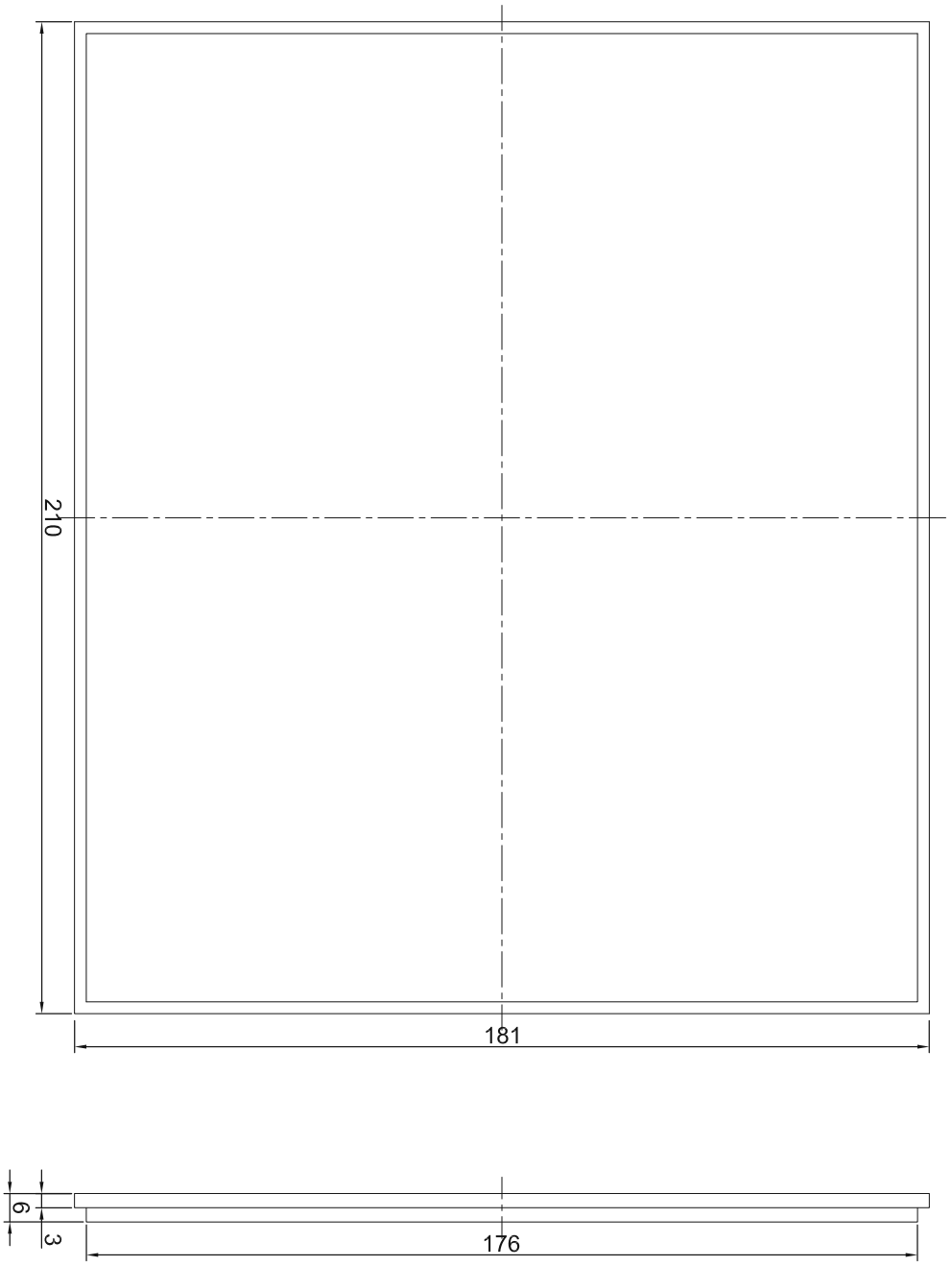


Abbildung C.10: Oberseite des Steuerungsgehäuses

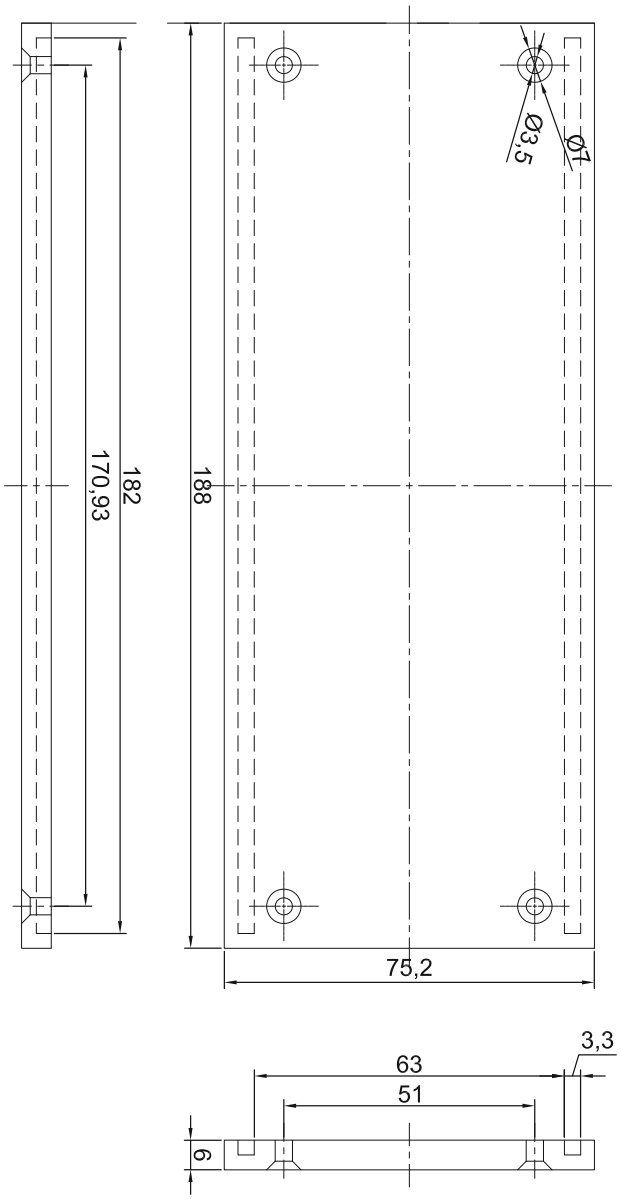


Abbildung C.11: Seitenteil des Steuerungsgehäuses

C.3 Abschirmung Detektoreinheit

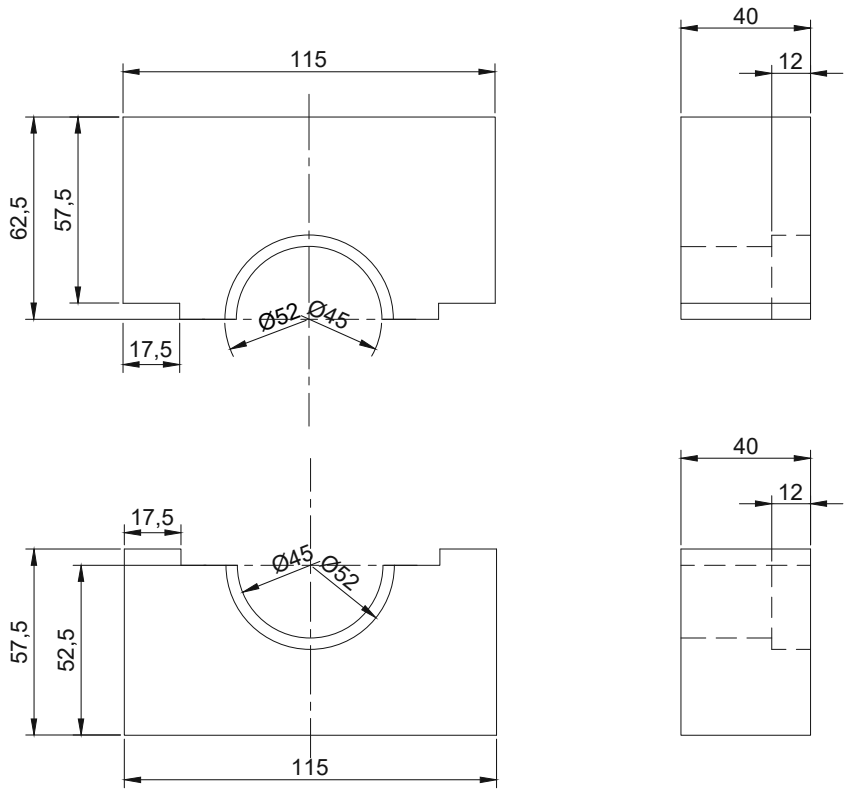


Abbildung C.12: Teil 1 der Bleiabschirmung für die Kamera

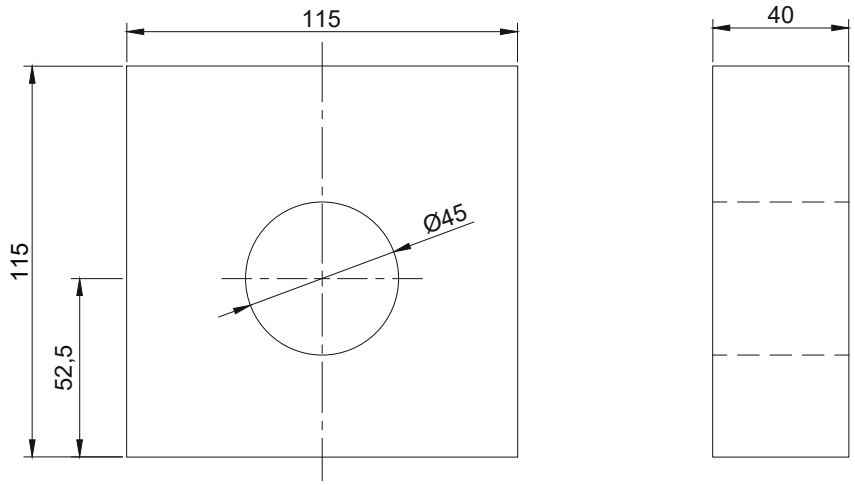


Abbildung C.13: Teil 2 der Bleiabschirmung für die Kamera

C.4 Gehäuse Detektoreinheit

Abbildung C.14: Basisteil der Detektoreinheit mit Aufnahme für den Spiegel

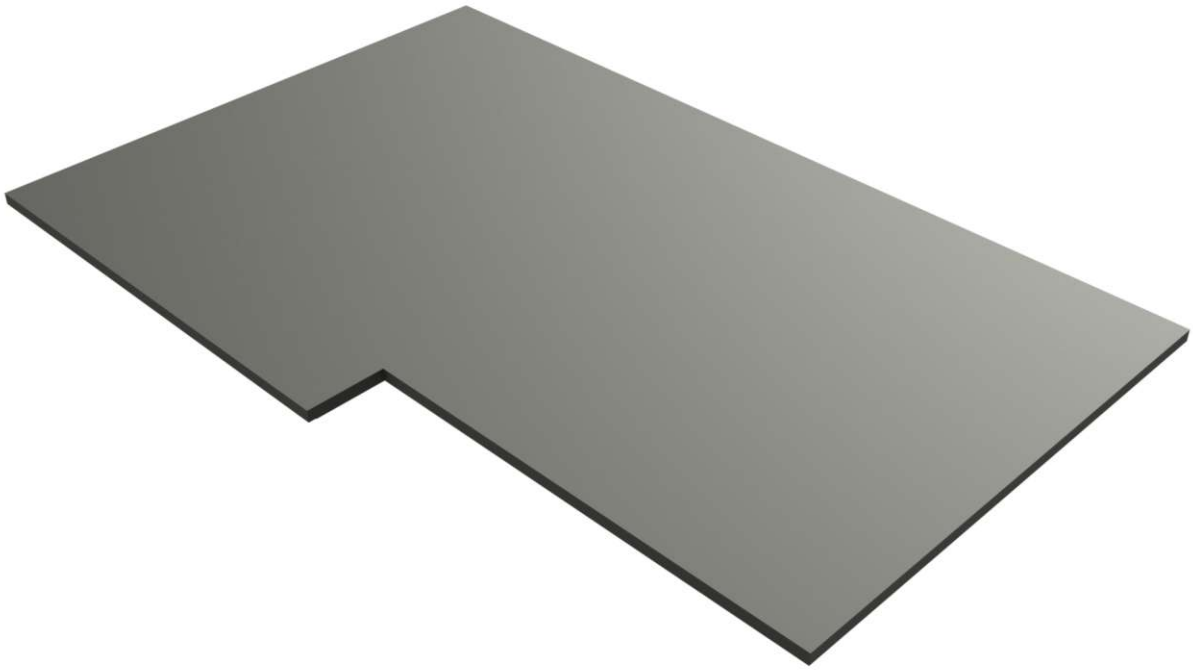


Abbildung C.15: Deckel des Basisteils mit Aufnahme für den Spiegel

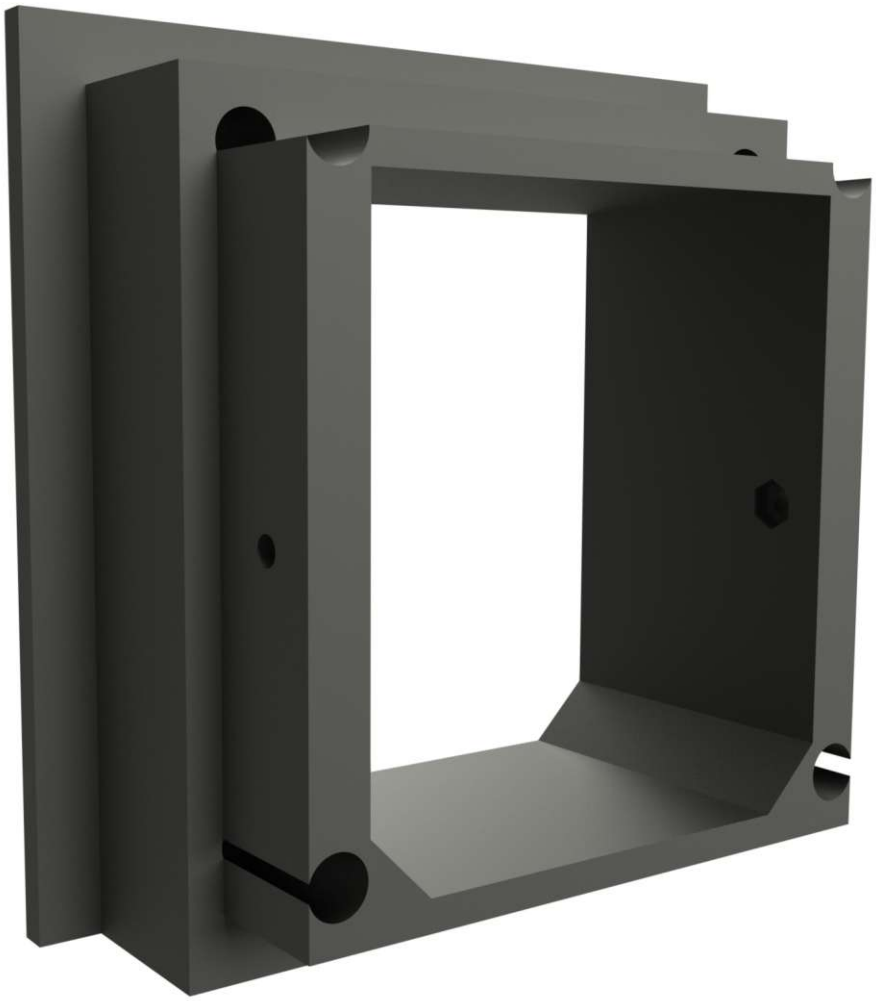


Abbildung C.16: Mittelteil der Detektoreinheit

Abbildung C.17: Basisplatte zur Aufnahme der Kamera

Abbildung C.18: Abdeckung des Objektivs und der Abschirmung

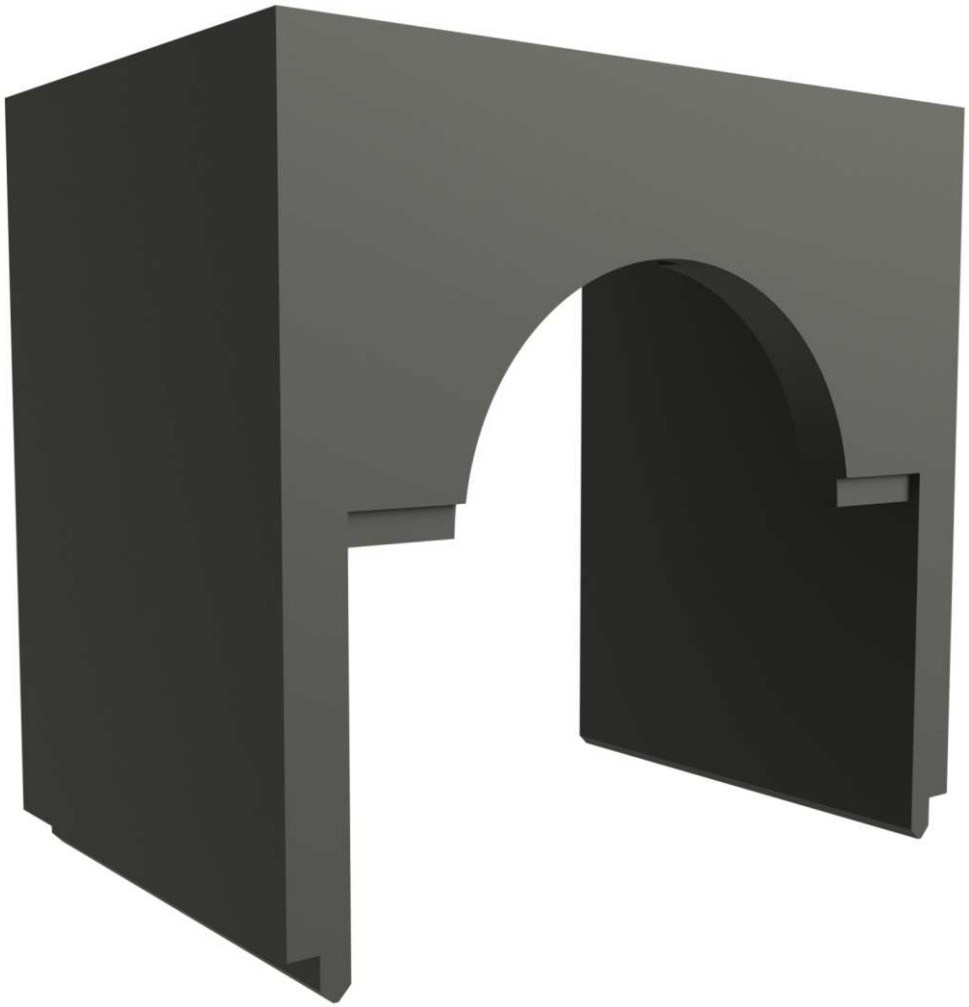


Abbildung C.19: Klemmdeckel zur Befestigung der Kamera

Abbildung C.20: Halterung des Szintillators an der Detektoreinheit

D Quelltexte

D.1 Arduino Sketch

Der Sketch für den Arduino Nano Every wurde mit der Entwicklungsumgebung Arduino IDE in der Programmiersprache C++ erstellt.

```
/*************************************************************************
** Radiography Controller
** Program Filename: controller_sketch.ino
** Author: Clemens Trunner
** Date:
** Description:
** This sketch is the software for the new radiography controller at the TU Wien Triga Reactor.
** It is made for an Arduino Nano Every based on an ATmega4809 and is used to control two stepper motors for sample manipulation and a relay for opening the shutter.
** For this reason, the sketch combined with the Arduino generates a square wave signal for each of the two stepper motors and an on/off signal for the shutter relay.
** One stepper motor is for linear and the second is for rotational movement.
** Besides the three signals, the Arduino produces other signals for controller and motor drivers.
** The data transfer between the Arduino and the measurement computer is made by serial communication.
*************************************************************************/

/*************************************************************************
** definition of the I/O pins using the Arduino pinout
*************************************************************************/
#define READY_PIN 20  //controller ready LED signal, Pin 20 of the Arduino Nano Every or pin PD4 on ATmega4809
#define LIN_STEP_PIN 6    //square wave signal for linear movement, Pin 6 of the Arduino Nano Every or pin PF4 on ATmega4809
#define LIN_DIR_PIN  16   //direction signal for linear movement, Pin 16 of the Arduino Nano Every or pin PD1 on ATmega4809
#define LIN_ENBL_PIN  17  //enable signal for linear movement, Pin 17 of the Arduino Nano Every or pin PD0 on ATmega4809
#define ROT_STEP_PIN  3   //square wave signal for rotational movement, Pin 3 of the Arduino Nano Every or pin PF5 on ATmega4809
```

```
#define ROT_DIR_PIN  14   //direction signal for rotational movement, Pin 14 ↙
    ↳ of the Arduino Nano Every or pin PD3 on ATmega4809
#define ROT_ENBL_PIN  15  //enable signal for rotational movement, Pin 4 of ↙
    ↳ the Arduino Nano Every or pin PD2 on ATmega4809
#define STATUS_LED_PIN 2  //status motor driver LED signal, Pin 2 of the ↙
    ↳ Arduino Nano Every or pin PA0 on ATmega4809
#define SHUTTER_RELAY_PIN 21  //shutter signal, Pin 21 of the Arduino Nano ↙
    ↳ Every or pin PD5 on ATmega4809
#define DRIVER_RESET_PIN 18 //reset driver error memory, Pin 18 of the ↙
    ↳ Arduino Nano Every or pin PA2 on ATmega4809
#define DRIVER_ERROR_PIN 10 //error from driver error memory, Pin 2 of the ↙
    ↳ Arduino Nano Every or pin PB1 on ATmega4809
#define LIMIT_OUT_PIN 11  //signal from limit switch out position, Pin 11 of ↙
    ↳ the Arduino Nano Every or pin PE0 on ATmega4809
#define LIMIT_IN_PIN  12  //signal from limit switch in position, Pin 12 of ↙
    ↳ the Arduino Nano Every or pin PE1 on ATmega4809

/*****************************************************************************
** define global variables
*****************************************************************************/
volatile int rot_step_count = 0;  //step counter variable to count the steps ↙
    ↳ of the clock signal
volatile int rot_step_setpoint = 0; //set point variable for the total ↙
    ↳ number of steps that are required

/*****************************************************************************
** initialize timer/counter A0 (LED blinking)
*****************************************************************************/
void initTCA0()
{
  PORTMUX.TCAROUTEA = PORTMUX_TCA0_PORTA_gc;  //select port A as output
  TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV1024_gc; //set prescaler to 1024
  TCA0.SINGLE.CTRLB = TCA_SINGLE_WGMODE_FRQ_gc; //select frequency mode
  TCA0.SINGLE.CMP0  = 0x0A00; //set compare value at 2560 to get 3Hz signal
}

/*****************************************************************************
** start timer/counter A0 (LED blinking)
*****************************************************************************/
void startTCA0()
{
  TCA0.SINGLE.CTRLB |= TCA_SINGLE_CMPOEN_bm;  //enable waveform on output pin
  TCA0.SINGLE.CTRLA |= TCA_SINGLE_ENABLE_bm;  //enable TCA0
}

/*****************************************************************************
** stop timer/counter A0 (LED blinking )
```

```
*************************************************************************/
void stopTCA0()
{
  TCA0.SINGLE.CTRLB &= ~(TCA_SINGLE_CMPOEN_bm); //disable waveform on output ↙
    ↳ pin
  TCA0.SINGLE.CTRLA &= ~(TCA_SINGLE_ENABLE_bm); //disable TCA0
}

/*************************************************************************
** initialize timer/counter B0 (linear movement)
*************************************************************************/
void initTCB0()
{
  PORTMUX.TCBROUTEA |= PORTMUX_TCB0_bm; //select Port F as output
  TCB0.CTRLA = TCB_CLKSEL_CLKTCA_gc;  //use TCA0 as clocksource
  TCB0.CTRLB = TCB_CNTMODE_PWM8_gc;  //select 8-Bit PWM mode and enable ↙
    ↳ waveform output
  TCB0.CCMPL = 0x18;  //set CCMPL value at 24 to get 1kHz signal
  TCB0.CCMPH = 0x0C;  //set CCMPH value at 12 to get 50% duty cycle
}

/*************************************************************************
** start timer/counter B0 (linear movement)
*************************************************************************/
void startTCB0()
{
  TCB0.CTRLB |= TCB_CCMPEN_bm;  //enable waveform on output pin
  TCB0.CTRLA |= TCB_ENABLE_bm;  //enable TCB0
}

/*************************************************************************
** stop timer/counter B0 (linear movement)
*************************************************************************/
void stopTCB0()
{
  TCB0.CTRLB |= TCB_CCMPEN_bm;  //enable waveform on output pin
  TCB0.CTRLA  &= ~(TCB_ENABLE_bm);  //disable TCB0
}

/*************************************************************************
** initialize timer/counter B1 (rotational movement)
*************************************************************************/
void initTCB1()
{
  PORTMUX.TCBROUTEA |= PORTMUX_TCB1_bm; //select Port F as output
  TCB1.CTRLA = TCB_CLKSEL_CLKTCA_gc;  //use TCA0 as clocksource
  TCB1.CTRLB = TCB_CNTMODE_PWM8_gc; //select 8-Bit PWM mode and make ↙
```

```
    ↳ waveform output available
  TCB1.CCMPL = 0xFF;  //set CCMPL value at 255 to get 60Hz signal
  TCB1.CCMPH = 0x80;  //set CCMPH value at 128 to get 50% duty cycle
  TCB1.INTCTRL |= TCB_CAPT_bm;  //enable interrupt
}

/*****************************************************************************
** start timer/counter B1 (rotational movement)
*****************************************************************************/
void startTCB1()
{
  TCB1.CTRLB |= TCB_CCMPEN_bm;  //enable waveform on output pin
  TCB1.CTRLA |= TCB_ENABLE_bm;  //enable TCB1
}

/*****************************************************************************
** stop timer/counter B1 (rotational movement)
*****************************************************************************/
void stopTCB1()
{
  TCB1.CTRLB &= ~(TCB_CCMPEN_bm); //disable waveform on output pin
  TCB1.CTRLA &= ~(TCB_ENABLE_bm); //disable TCB1
}

/*****************************************************************************
** interrupt service routine of TCB1 (rotational movement)
*****************************************************************************/
ISR(TCB1_INT_vect)
{
  stepCounter();  //call function to count steps and stop movement if necessary
  TCB1.INTFLAGS |= TCB_CAPT_bm; //clear the interrupt flag
}

/*****************************************************************************
** initialize real-time counter (wait for shutter)
*****************************************************************************/
void initRTC()
{
  RTC.CTRLA = RTC_PRESCALER_DIV1_gc; //set prescaler
  RTC.CLKSEL = RTC_CLKSEL_INT1K_gc; //use 1024Hz clocksource
  RTC.PER = 0x0600;                 //set overflow value at 1536 to wait 1,5s
  RTC.INTCTRL |= RTC_OVF_bm;        //enable overflow interrupt
}

/*****************************************************************************
** start real-time counter (wait for shutter)
*****************************************************************************/
```

```
void startRTC()
{
  RTC.CTRLA |= RTC_RTCEN_bm; //enable RTC
}

/*************************************************************************
** interrupt service routine of RTC (wait for shutter)
*************************************************************************/
ISR(RTC_CNT_vect)
{
  stopShutter(); //call function to indicate shutter stopped moving
  RTC.CTRLA &= ~(RTC_RTCEN_bm); //disable RTC
  RTC.INTFLAGS |= RTC_OVF_bm; //clear the interrupt flag in the interrupt
    ↳ flags register to enable the interrupt again
}

/*************************************************************************
** initialize Port E interrupt (limitation switches)
*************************************************************************/
void initPORTEINT()
{
  PORTE.INTFLAGS |= PIN0_bm;  //Pin 0 as interrupt source (LIMIT_OUT_PIN)
  PORTE.INTFLAGS |= PIN1_bm;  //Pin 1 as interrupt source (LIMIT_IN_PIN)
  PORTE.PIN0CTRL = PORT_ISC_RISING_gc;  //raise interrupt of Pin 0 on rising
    ↳ edge
  PORTE.PIN1CTRL = PORT_ISC_RISING_gc;  //raise interrupt of Pin 1 on rising
    ↳ edge
}

/*************************************************************************
** interrupt service routine of Port E (limitation switches)
*************************************************************************/
ISR(PORTE_PORT_vect)
{
  posReached();
  PORTE.INTFLAGS |= PIN0_bm;  //clear the interrupt flag of Pin 0
  PORTE.INTFLAGS |= PIN1_bm;  //clear the interrupt flag of Pin 1
}

/*************************************************************************
** initialize port B interrupt (driver error)
*************************************************************************/
void initPORTBINT()
{
  PORTB.INTFLAGS |= PIN1_bm;  //Pin 1 as interrupt source (DRIVER_ERROR_PIN)
  PORTB.PIN1CTRL = PORT_ISC_RISING_gc; //raise interrupt on rising edge
}
```

```
/*****************************************************************************
** interrupt service routine of port B (driver error)
*****************************************************************************/
ISR(PORTB_PORT_vect)
{
 driverError(); // call driver error
 PORTB.INTFLAGS = PIN1_bm;  //clear the interrupt flag
}

/*****************************************************************************
** start linear motion
*****************************************************************************/
void startLin(bool direction)
{
  /* check which direction is requested and if it is possible */
  if(!checkPosition(LIMIT_OUT_PIN) && direction){
    Serial.println("moving_out"); //report "moving_out" via serial port
  }
  else if(!checkPosition(LIMIT_IN_PIN) && !direction){
    Serial.println("moving_in"); //report "moving_in" via serial port
  }
  else{
    return;
  }
  digitalWrite(LIN_DIR_PIN, direction); //set direction of driver
  digitalWrite(LIN_ENBL_PIN, LOW); //enable driver
  startTCB0();  //call function to start step signal
  startTCA0();  //call function to start LED blinking
}

/*****************************************************************************
** stop linear motion
*****************************************************************************/
void stopLin()
{
  digitalWrite(LIN_ENBL_PIN, HIGH); //disable driver
  stopTCB0(); //call function to stop step signal
  stopTCA0(); //call function to stop LED blinking
  Serial.println("lin_stopped");
}

/*****************************************************************************
** check position of sample
*****************************************************************************/
bool checkPosition(int pin)
{
```

```
  /* check if requested limitation switch is pressed */
  if(digitalRead(pin)){
    /* check if outer position is requested*/
    if(pin == LIMIT_OUT_PIN){
      Serial.println("pos_out"); //report "pos_out" via serial port
    }
    /* check if inner position is requested*/
    if(pin == LIMIT_IN_PIN){
      Serial.println("pos_in"); //report "pos_in" via serial port
    }
    return true;  //if limitation switch is pressed, return true
  }else{
      return false; //if limitation switch is not pressed, return false
  }
}

/*****************************************************************************
** limit switch pressed
*****************************************************************************/
void posReached()
{
  stopLin();  //call function to stop linear motion
  checkPosition(LIMIT_IN_PIN);  //call function to check if inner position ↙
    ↳ was reached
  checkPosition(LIMIT_OUT_PIN); //call function to check if outer position ↙
    ↳ was reached
}

/*****************************************************************************
** start rotational motion
*****************************************************************************/
void startRot(bool direction, int set_point)
{
  rot_step_setpoint = set_point; //set global setpoint variable to required ↙
    ↳ value
  rot_step_count=0; //set global count variable to 0
  Serial.println((String)"start_rot_"+rot_step_setpoint); //report ↙
    ↳ "start_rot_XXXX" via serial port
   (ROT_DIR_PIN, direction); //set direction of driver
  digitalWrite(ROT_ENBL_PIN, LOW); //enable driver
  startTCB1();  //call function to start step signal
  startTCA0(); //call function to start LED blinking
}

/*****************************************************************************
** stop rotational motion
*****************************************************************************/
```

```
void stopRot()
{
  digitalWrite(ROT_ENBL_PIN, HIGH);  //disable driver
  stopTCB1(); //call function to stop step signal
  stopTCA0();  //call function to stop LED blinking
  Serial.println("rot_stopped"); //report "rot_stopped" via serial port
}

/*****************************************************************************
** rotation step counter
*****************************************************************************/
void stepCounter()
{
  rot_step_count++; //increment count by 1
  /* check if counter has reached setpoint  */
  if(rot_step_count>=rot_step_setpoint){
    stopRot(); //stop rotation when setpoint variable is reached
  }
}

/*****************************************************************************
** start shutter movement
*****************************************************************************/
void startShutter(bool direction)
{
  /* check which direction is requested */
  if (direction){
    Serial.println("start_close_shutter"); //report "start_close_shutter" ↙
    ↳ via serial port
  }
  else{
    Serial.println("start_open_shutter"); //report "start_open_shutter" via ↙
    ↳ serial port
  }
  digitalWrite(SHUTTER_RELAY_PIN, direction); //set the shutter output
  startRTC(); //call function to start shutter timer
  startTCA0(); //call function to start LED blinking
}

/*****************************************************************************
** stop shutter
*****************************************************************************/
void stopShutter()
{
  stopTCA0();  //call function to stop LED blinking
  /* check which position was required */
  if(digitalRead(SHUTTER_RELAY_PIN)){
```

```
    Serial.println("shutter_closed"); //report "shutter_closed" via serial port
  }
  else{
    Serial.println("shutter_opened"); //report "shutter_opened" via serial port
  }
}

/*****************************************************************************
** stop all motion
*****************************************************************************/
void stopAll()
{
  stopLin();  //call function to stop linear motion
  stopRot();  //call function to stop rotational motion
  startShutter(HIGH);  //call function to close shutter
  Serial.println("all_stopped"); //report "all_stopped" via serial port
}

/*****************************************************************************
** driver error occured
*****************************************************************************/
void driverError()
{
  stopAll(); //call function to stop all motion
  Serial.println("driver_error"); //report "driver_error" via serial port
}

/*****************************************************************************
** handle input string
**
** input string keyword:
** connect - check if controller is responding
** move_in - move sample in
** move_out - move sample out
** stop_lin - stop linear motion of the sample
** rot_cw+XXXX - rotate sample clockwise by XXXX steps
** rot_ccw+XXXX - rotate sample counterclockwise by XXXX steps
** stop_rot - stop rotation of the sample
** open_shutter - open shutter
** close_shutter - close shutter
** stop_lin - stop all motion and close shutter
*****************************************************************************/
void handleInputString(String input_command)
{
  int delimiter_index;  //position of the delimiter
  int requested_steps;  //number of steps passed
  input_command.trim(); //removing all white spaces from input string
```

```
delimiter_index = input_command.indexOf("+"); //find position of delimiter
/* check if delimiter is in input string */
if(delimiter_index>0)
{
  requested_steps = input_command.substring(delimiter_index+1).toInt();
  //separate steps from input string and convert to integer
  input_command = input_command.substring(0,delimiter_index); //keep
  remaining input string after separating steps information
}
/* check if input string equals keyword*/
if(input_command.equals("connect"))
{
  Serial.println("connected");  //report "connected" via serial port
}
else if(input_command.equals("move_in"))
{
  startLin(LOW);  //call function to start moving in
}
else if(input_command.equals("move_out"))
{
  startLin(HIGH); //call function to start moving out
}
  if(input_command.equals("stop_lin"))
  {
  stopLin();  //call function to stop linear motion
}
else if(input_command.equals("rot_cw"))
{
  startRot(HIGH, requested_steps);  //call function to start rotation
  clockwise
  }
else if(input_command.equals("rot_ccw"))
{
  startRot(LOW, requested_steps); //call function to start rotation
  counterclockwise
}
else if(input_command.equals("stop_rot"))
{
  stopRot();  //call function to stop rotational motion
}
else if(input_command.equals("open_shutter"))
{
  startShutter(LOW); //call function to open shutter
}
else if(input_command.equals("close_shutter"))
{
  startShutter(HIGH);  //call function to close shutter
```

```
  }
  else if(input_command.equals("stop_all"))
  {
    stopAll();  //call function to stop all motion
  }
}

/*****************************************************************************
** set up serial connection
*****************************************************************************/
void serialSetup()
{
  Serial.begin(9600);
}

/*****************************************************************************
** set up I/O pins
*****************************************************************************/
void pinSetup()
{
  /* declare pins as output */
  pinMode(LIN_STEP_PIN, OUTPUT);
  pinMode(LIN_DIR_PIN, OUTPUT);
  pinMode(LIN_ENBL_PIN, OUTPUT);
  pinMode(ROT_STEP_PIN, OUTPUT);
  pinMode(ROT_DIR_PIN, OUTPUT);
  pinMode(ROT_ENBL_PIN, OUTPUT);
  pinMode(STATUS_LED_PIN, OUTPUT);
  pinMode(SHUTTER_RELAY_PIN, OUTPUT);
  pinMode(READY_PIN, OUTPUT);
  /* set initial value to output pins */
  digitalWrite(READY_PIN, LOW);
  digitalWrite(LIN_ENBL_PIN, HIGH);
  digitalWrite(ROT_ENBL_PIN, HIGH);
  digitalWrite(STATUS_LED_PIN, LOW);
  digitalWrite(SHUTTER_RELAY_PIN, HIGH);
  /* declare pins as input and enable internal pullup resistor*/
  pinMode(LIMIT_OUT_PIN, INPUT_PULLUP);
  pinMode(LIMIT_IN_PIN, INPUT_PULLUP);
  pinMode(DRIVER_ERROR_PIN, INPUT_PULLUP);
}

/*****************************************************************************
** set up timers and interrupts
*****************************************************************************/
void timerInterruptSetup()
{
```

```
  cli();  //disable global interrupt
  initTCA0(); //call function to initialize timer/counter A0
  initTCB0(); //call function to initialize timer/counter B0
  initTCB1(); //call function to initialize timer/counter B1
  initRTC();  //call function to initialize Real Timer Counter
  initPORTEINT(); //call function to initialize interrupt on Port E
  initPORTBINT(); //call function to initialize interrupt on Port B
  sei();  //enable global interrupt
}

/*************************************************************************
** set up function
*************************************************************************/
void setup()
{
  pinSetup(); //call function to set up I/O pins
  serialSetup();  //call function to set up serial connection
  timerInterruptSetup();  //call function to set up timers and interrupts
  Serial.println("controller_ready"); //report "controller_ready" via serial ↙
    ↳ port
  digitalWrite(READY_PIN, HIGH); //set controller ready LED signal high
  digitalWrite(STATUS_LED_PIN, HIGH);  //set status motor driver LED signal ↙
    ↳ high
}

/*************************************************************************
** loop function
*************************************************************************/
void loop()
{
  /* check if data is available on serial port */
  if (Serial.available() > 0)
  {
    handleInputString(Serial.readStringUntil('\n'));  //call function to ↙
    ↳ handle input string with string from serial port as argument
  }
}
```

Quelltext D.1: Software für die Steuerplatine zur Verwendung mit dem verbauten Arduino Nano Every

D.2 Paket imgctrl

Das Paket `imgctrl` ist eine in Python geschriebene Sammlung an Modulen, welche die Softwareschnittstelle des Computers zur Steuerung darstellt. Das Paket besteht aus den beiden Modulen `ctrlcom` und `ctrlcmd`.

```
''' Module to communicate with the neutron imaging controller

Description:
    Creates the possibility to communicate with the controller.
    Provides classes and methods to connect, disconnect, send  and
    receive data from controller.
Author: Clemens Trunner
Date Created: September 22, 2023
Date Modified:
Version: 1.0
Python Version:
Dependencies: serial, serial.tools, time, threading
License:
'''

import serial
from serial.tools import list_ports
import time
import threading


class Observer():
    ''' Implementing of observer design pattern

    The observer design pattern creates the possibility to attach observers
    to a list maintained by a subject. At a state change, the subject can
    easily notify the observer.
    '''

    def __init__(self):
        ''' Initialize Observer class

        Initializes the Observer class and defines an empty instance list.
        '''
        self._methods = []

    def attach(self, method):
        ''' Attach observer to subject

        Attaches an observer object to the subject by appending it to the
        list.
```

```
        Parameters
        ----------
        method : object
            Object to be attached to subject.
        '''
        if method not in self._methods:
            self._methods.append(method)

    def detach(self, method):
        ''' Detach observer from subject

        Detaches an observer object from the subject by removing it from the
        list.

        Parameters
        ----------
        method : object
            Object to be detached from subject.
        '''
        if method in self._methods:
            self._methods.remove(method)

    def call(self, *args, **kwargs):
        ''' Call observers attached to subject

        Calls all observers that are attached to the subject.

        Parameters
        ----------
        *args : arbitrary arguments list
            Arguments to be passed to another object.
        **kwargs : arbitrary keyword arguments list
            Keyword arguments to be passed to another object.
        '''
        for method in self._methods:successful deutsch
            method(*args, **kwargs)


class _SerialCommunication():
    ''' Manage serial communication with controller

    Provides a set of methods to operate and maintain the serial
    communication with the controller. Allows to connect and
    disconnect, send and receive data and wait for specific data.
    '''
```

```
    def __init__(self):
        ''' Initialize _SerialCommunication class

        Initializes the communication class. Starts an instance of the
        Observer class. Defines instance attributes, such as a variable for
        received messages, an empty list for serial ports, a boolean
        variable to stop the receiving thread and the receiving thread
        itself. Searches and stores available ports in the list after defining.
        '''
        self._receive_observer = Observer()
        self._message = ""
        self._ports = []
        self.stop_receive_data_thread = False
        self.receive_thread = threading.Thread(target = self.__receive)
        # Search for qualified ports in all listed ports
        for port in serial.tools.list_ports.comports():
            # If name of port contains 'ACM' or 'USB', append it to ports
            if ('ACM' or 'USB') in port.device:
                self._ports.append(port.device)

    def _connected(self):
        ''' Check connection to controller

        Checks the serial connection to controller.

        Returns
        ----------
        bool
            State of serial connection
        '''
        try:
            return self.serial.is_open
        except:
            return False

    def __start_receive(self):
        ''' Start receive thread

        Starts receiving data by setting the stop attribute false and
        starting the receive thread.

        Returns
        ----------
        bool
            Life state of thread
        '''
        self.stop_receive_data_thread = False
```

```
        self.receive_thread.start()
        return self.receive_thread.is_alive()

    def __stop_receive(self):
        ''' Stop receive thread

        Stops receiving data by setting the stop attribute true, cancel
        reading data from serial port and join thread to terminate it safe.

        Returns
        ----------
        bool
            Inverted state of thread
        '''
        self.stop_receive_data_thread = True
        self.serial.cancel_read()
        self.receive_thread.join(timeout = 2)
        return not self.receive_thread.is_alive()

    def __receive(self):
        ''' Receive data from serial port

        Receives data by linewise reading from serial port in loop. If data
        is read in, it is decoded. After data preparation is done, call
        observers about new data. If stop attribute is true, exit the loop.
        '''
        while self._connected():
            input = self.serial.readline()
            if self.stop_receive_data_thread:
                break
            if input:
                # Decode input data with utf-8 encoding and remove escape ↙
↳ sequence
                self._message = input.decode('utf-8', ↙
↳ 'ignore').replace('\r\n', '')
                self._receive_observer.call(self._message)

    def __write(self, data):
        '''Write to serial port

        Writes encoded data from parameter to serial port.

        Parameters
        ----------
        data : str
            Data to be written via serial port
```

```
        Returns
        ----------
        bool
            Result of writing data
        '''
        if self._connected():
            try:
                # Add escape sequence to data
                data = data + "\n"
                # Write utf-8 encoded data to serial port
                self.serial.write(bytes(data.encode('utf-8', 'ignore')))
            except serial.SerialTimeoutException:
                return False
            else:
                return True
        else:
            return False

    def __wait(self, response, timeout):
        '''Wait for response

        Waits for response in received data. Stops waiting when time exceeds.

        Parameters
        ----------
        response : str
            Response to be waited for
        timeout : int
            Time to wait for a response

        Returns
        ----------
        bool
            Result of waiting
        '''
        # Build expire time from timeout
        exittime=time.time()+timeout
        while self._message != response:
            # Break loop and stop waiting when time exceeds expire time
            if time.time() > exittime:
                return False
        return True

    def _send(self, data, response = None, timeout = None):
        ''' Send data to controller

        Sends data to controller using writing method. If response and
```

```
        timeout are passed, waits for data to match response.

        Parameters
        ----------
        data : str
            Data to be sent to controller
        response : str
            Response to be waited for
        timeout : int
            Time to wait for a response

        Returns
        ----------
        bool
            Result of sending
        '''
        # If no timeout or response is passed, just write data
        if timeout is None or response is None:
            return self.__write(data)
        # If timeout is type integer and response is type string, write data
↪ and wait for response
        elif isinstance(timeout, int) and isinstance(response, str):
            if self.__write(data):
                    # If writing was successful, wait for response
                    return self.__wait(response, timeout)
            else:
                    return False
        else:
            return False

    def _connect(self, port):
        ''' Connect to controller

        Connects to controller via serial port. If connecting to port was
        successful, start receiving data.

        Parameters
        ----------
        port : str
            Name of port to connect with

        Returns
        ----------
        bool
            Result of connecting
        '''
        try:
```

```
            # Open exclusive serial port with baudrate 9600
            self.serial = serial.Serial(port, 9600, timeout = 0.5,
    ↪ write_timeout = 1, exclusive = True)
        except:
            return False
        else:
            return self.__start_receive()

    def _disconnect(self):
        ''' Disconnect from controller

        Disconnects from controller by stopping to receive data and closing
        serial connection.

        Returns
        ----------
        bool
            Result of disconnecting
        '''
        if self._connected():
            if self.__stop_receive():
                try:
                    self.serial.close()
                except:
                    return False
                else:
                    return True
            else:
                return False
        else:
            return True
```

Quelltext D.2: Modul `ctrlcom` als Teil des Pakets `imgctrl`

```
'''Module for specifying commands using methods of a class

Description:
    Defines a specific method for each command to control the neutron
    imaging controller. These command methods and their class build an
    advanced interface to operate the controller.
Author: Clemens Trunner
Date Created: September 22, 2023
Date Modified:
Version: 1.0
Python Version:
```

```
Dependencies: controller_communication
License:
'''

import ctrlcom

class Commander(_SerialCommunication):
    ''' Interface to command the controller

    Links the textbased commands of the controller to callable methods.
    '''

    def __init__(self):
        ''' Initialize Commander class

        Initializes the Commander class. Starts an instance of the
        Observer class to notify observers when sending instructions.
        Redefines instance attributes, receive observer and serial ports list.
        '''
        _SerialCommunication.__init__(self)
        self.send_observer = Observer()
        self.receive_observer = self._receive_observer
        self.ports = self._ports

    def _send(self, instruction, *args, **kwargs):
        ''' Send instructions to controller

        Notifies the observers subscribed to sending of the instruction
        and calls the _send method of _SerialCommunication class with its
    ↳ arguments.

        Parameters
        ----------
        instruction : str
            Command to be sent to controller
        response : str
            Response to be waited for (needs timeout parameter)
        timeout : int
            Time to wait for a response

        Returns
        ----------
        bool
            Result of sending
        '''
        self.send_observer.call(instruction)
        return _SerialCommunication._send(self, instruction, *args, **kwargs)
```

```
    def connect(self, *args, **kwargs):
        ''' Connect to controller

        Calls _connect method of _SerialCommunication class with its arguments.

        Parameters
        ----------
        port : str
            Name of port to connect with

        Returns
        ----------
        bool
            Result of connecting
        '''
        return _SerialCommunication._connect( *args, **kwargs)

    def disconnect(self):
        ''' Disconnect from controller

        Calls _disconnect method of _SerialCommunication class.

        Returns
        ----------
        bool
            Result of disconnecting
        '''
        return self._disconnect()

    def connected(self):
        ''' Check connection to controller

        Calls the _connected method of _SerialCommunication class to check
    ↳ connection status.

        Returns
        ----------
        bool
            State of serial connection
        '''
        return self._connected()

    def linear_out(self, *args, **kwargs):
        ''' Move sample out

        Moves the sample out of the beam using the linear table.
```

```
        Parameters
        ----------
        timeout : int
            Time of timeout in seconds

        Returns
        ----------
        bool
            Result of movement, false if timeout occures
        '''
         return self._send("move_out", "pos_out", *args, **kwargs)

    def linear_in(self, *args, **kwargs):
        ''' Move sample in

        Moves the sample in the beam using the linear table.

        Parameters
        ----------
        timeout : int
            Time of timeout in seconds

        Returns
        ----------
        bool
            Result of movement, false if timeout occures
        '''
        return self._send("move_in", "pos_in",*args, **kwargs)

    def rotate_clockwise(self, *args, step = None, **kwargs):
        ''' Rotate sample clockwise

        Rotates the sample clockwise using the rotation table.

        Parameters
        ----------
        step : int
            Steps of rotation indicating rotation angle
        timeout : int
            Time of timeout in seconds

        Returns
        ----------
        bool
            Result of movement, false if timeout occures
        '''
```

```
        # Check if step is given and has correct type
        if type(step) is int:
            return self._send("rot_cw+" + str(step), "rot_stopped", *args, ↙
↳ **kwargs)
        else:
            return False

    def rotate_counter_clockwise(self, *args, **kwargs):
        ''' Rotate sample counterclockwise

        Rotates the sample counterclockwise using the rotation table.

        Parameters
        ----------
        step : int
            Steps of rotation indicating rotation angle
        timeout : int
            Time of timeout in seconds

        Returns
        ----------
        bool
            Result of movement, false if timeout occures
        '''
        # Check if step is given and has correct type
        if type(step) is int:
            return self._send("rot_ccw+" + str(step),"rot_stopped", *args, ↙
↳ **kwargs)
        else:
            return False

    def open(self, *args, **kwargs):
        ''' Open shutter

        Opens shutter to unleash the neutron beam.

        Parameters
        ----------
        timeout : int
            Time of timeout in seconds

        Returns
        ----------
        bool
            Result of movement, false if timeout occures
        '''
        return self._send("open_shutter", "shutter_opened", *args, **kwargs)
```

```
    def close(self, *args, **kwargs):
        ''' Close shutter

        Closes shutter to distract the neutron beam.

        Parameters
        ----------
        timeout : int
            Time of timeout in seconds

        Returns
        ----------
        bool
            Result of movement, false if timeout occures.
        '''
        return self._send("close_shutter", "shutter_closed", *args, **kwargs)

    def stop_all(self, *args, **kwargs):
        ''' Stop all movement

        Stops linear and rotational movement and closes shutter.

        Parameters
        ----------
        timeout : int
            Time of timeout in seconds

        Returns
        ----------
        bool
            Result of movement, false if timeout occures
        '''
        return self._send("stop_all", "all_stopped", *args, **kwargs)

    def stop_rotate(self, *args, **kwargs):
        ''' Stop rotational movement

        Stops rotation movement of sample

        Parameters
        ----------
        timeout : int
            Time of timeout in seconds

        Returns
        ----------
```

```
        bool
            Result of movement, false if timeout occures
        '''
        return self._send("stop_lin", "lin_stopped", *args, **kwargs)

    def stop_linear(self, *args, **kwargs):
        ''' Stop linear movement

        Stops linear movement of sample.

        Parameters
        ----------
        timeout : int
            Time of timeout in seconds

        Returns
        ----------
        bool
            Result of movement, false if timeout occures
        '''
        return self._send("stop_rot", "rot_stopped", *args, **kwargs)
```

Quelltext D.3: Modul `ctrlcmd` als Teil des Pakets `imgctrl`

Tabellenverzeichnis

Abbildungsverzeichnis

Quellen

[1] Ian S Anderson, Robert L McGreevy und Hassina Z Bilheux. *Neutron imaging and applications : a reference for the imaging community*. eng. New York, NY: Springer, 2009. ISBN: 1441946195.

[2] Chinmoy Bhattacharya. „2-D and 3-D image reconstruction from backprojections of underwater objects by ocean acoustic tomography (OAT)“. In: (Juni 2012).

[3] James Chadwick und M. Goldhaber. „The nuclear photoelectric effect“. eng. In: *Proceedings of the Royal Society of London. Series A, Mathematical and physical sciences* 151.873 (1935), S. 479–493. ISSN: 0080-4630.

[4] Clinton Davisson und Lester H Germer. „Diffraction of Electrons by a Crystal of Nickel“. eng. In: *Phys. Rev.* 30 (Dez. 1927), S. 705–740.

[5] Wolfgang Demtröder. *Experimentalphysik : 3. Atome, Moleküle und Festkörper*. ger. 5., neu bearbeitete und aktualisierte Auflage. Berlin Heidelberg: Springer Spektrum, 2016. ISBN: 3662490935.

[6] Nicolás Di Luozzo, Michael Schulz und Marcelo Fontana. „Imaging of boron distribution in steel with neutron radiography and tomography“. eng. In: *Journal of materials science* 55.18 (2020), S. 7927–7937. ISSN: 0022-2461.

[7] Particle Data Group u. a. „Review of Particle Physics“. In: *Progress of Theoretical and Experimental Physics* 2022.8 (Aug. 2022). ISSN: 2050-3911. URL: `https://doi.org/10.1093/ptep/ptac097` (besucht am 07. 02. 2024).

[8] Paul Scherrer Institut. *Neutron Imaging Setup*. URL: `https://www.psi.ch/de/niag/neutron-imaging-setup` (besucht am 07. 02. 2024).

[9] Anders P Kaestner. „MuhRec—A new tomography reconstructor“. eng. In: *Nuclear instruments & methods in physics research. Section A, Accelerators, spectrometers, detectors and associated equipment* 651.1 (2011), S. 156–160. ISSN: 0168-9002.

[10] Helmut Kaiser und Helmut Rauch. „De Broglie wave optics: neutrons, atoms and molecules“. eng. In: Ludwig Bergmann und Clemens Schaefer. *Optics of waves and particles*. Berlin [u.a.]: Walter de Gruyter, 1999. ISBN: 3110143186.

[11] Avinash C Kak und Malcolm Slaney. *Principles of computerized tomographic imaging*. eng. New York, NY: IEEE Pr., 1988. ISBN: 0879421983.

[12] Hartmut Kallmann. „Neutron radiography“. In: *Research; a journal of science and its applications* 1.6 (1948), S. 254–260.

[13] Glenn F Knoll. *Radiation detection and measurement*. eng. 4. ed.. Hoboken, NJ: Wiley, 2010. ISBN: 0470131489.

[14] Jiri Kopecky u. a. *Atlas of neutron capture cross sections*. Techn. Ber. International Atomic Energy Agency, 1997. URL: https://www-nds.iaea.org/ngatlas2 (besucht am 07. 02. 2024).

[15] Otto Peter. „Neutronen-Durchleuchtung“. In: *Zeitschrift für Naturforschung A* 1.10 (1946), S. 557–559.

[16] Tushar Roy. „Basic Principles of Neutron Radiography and Tomography“. In: *Neutron Imaging: Basics, Techniques and Applications*. Hrsg. von Dinesh K. Aswal, Partha S. Sarkar und Yogesh S. Kashyap. Singapore: Springer, 2022. ISBN: 978-981-16-6273-7.

[17] Ernest Rutherford. „Bakerian Lecture: Nuclear constitution of atoms“. eng. In: *Proceedings of the Royal Society of London. Series A, Containing papers of a mathematical and physical character* 97.686 (1920), S. 374–400. ISSN: 0950-1207.

[18] Rahul Satija u. a. „In situ neutron imaging technique for evaluation of water management systems in operating PEM fuel cells“. eng. In: *Journal of power sources* 129 (Apr. 2004), S. 238–245. ISSN: 0378-7753.

[19] Burkhard Schillinger. „An affordable image detector and a low-cost evaluation system for computed tomography using neutrons, x-rays, or visible light“. eng. In: *Quantum beam science* 3.4 (2019), S. 21. ISSN: 2412-382X.

[20] Varley F Sears. *Neutron optics : an introduction to the theory of neutron optical phenomena and their applications*. eng. Oxford series on neutron scattering in condensed matter. New York [u.a.]: Oxford Univ. Pr., 1989. ISBN: 0195046013.