Hochschule für Wirtschaft und Recht Berlin
Berlin School of Economics and Law

HZB Helmholtz Zentrum Berlin

Bachelorthesis

Kamera-Einbindung in ein verteiltes Steuerungssystem

vorgelegt am 9. August 2022

•

Fachbereich Duales Studium Wirtschaft / Technik
Hochschule für Wirtschaft und Recht Berlin

Name:	Almut Erdmann
Ausbildungsbetrieb:	Helmholtz-Zentrum Berlin für Materialien und Energie
Studienbereich:	Technik
Fachrichtung:	Informatik
Studiengang:	Informatik
Studienjahrgang:	2019
Erstgutachter:	Dr.-Ing. Günther Rehm
Zweitgutachter:	Prof. Dr. Rainer Höhne

Abstract

Um einen reibungslosen Betrieb zu garantieren, wird während des laufenden Betriebs des Elektronenspeicherrings BESSY II der beschleunigte Elektronenstrahl durchgängig überwacht.
Die vorliegende Arbeit beschäftigt sich mit der Frage, ob es eine alternative Möglichkeit gibt, die Kontrolle des Elektronenstrahls von BESSY II über Kameras in das Software-Paket EPICS einzubinden. Um die Frage zu beantworten, wurde der Versuch unternommen, eine solche alternative Möglichkeit in Python umzusetzen und zu testen.
Eine Beispielkamera konnte erfolgreich angebunden werden. Beim Analysieren der Bilddaten wurde ein Fit-Algorithmus optimiert, der Informationen über den zu kontrollierenden Elektronenstrahl erfasst. Die Einbindung in das Prozessleitsystem von EPICS konnte mit Hilfe von pythonSoftIOC umgesetzt werden.
Erste Tests im Testbetrieb waren erfolgreich. Der Implementierungsversuch wurde damit als alternative Möglichkeit zur Einbindung der Kontrolle des Elektronenstrahls von BESSY II über Kameras in EPICS eingestuft.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Hintergrund

Die Energiewende ist eine große politische sowie technische Herausforderung und steht im Fokus der Bundespolitik. Die Energieversorgung auf erneuerbare Energien umzustellen, ist ein entscheidender Schritt, um dem Klimawandel entgegenzuwirken und sich von geopolitischen Abhängigkeiten zu befreien. [Pre] Forschung, vor allem im Bereich Energie und Materie, ist dafür von besonderer Bedeutung.
Einen großen Beitrag in diesen Forschungsbereichen leisten die Wissenschaftler:innen der „Berliner Elektronenspeicherring-Gesellschaft für Synchrotronstrahlung m.b.H." (BESSY). An der Synchrotronstrahlungsquelle BESSY II untersuchen sie chemische Prozesse und innovative Materialien bis ins kleinste Detail. Mit der vom Sychrotron erzeugten weichen Röntgenstrahlung analysieren sie nanometerdünne Schichten. Mit den gewonnenen Erkenntnissen, vor allem im Bereich der effektiven Nutzung von Sonnenenergie, legen sie den Grundstein für die Forschung zur Gewinnung erneuerbarer Energien. [Lan20]
Die vorliegende Arbeit beschäftigt sich mit einer Erweiterung des Steuerungssystems von BESSY II, weshalb im Folgenden BESSY II und die Software für das Steuerungssystem (EPICS) genauer vorgestellt werden sollen.

1.1.1 Synchrotronstrahlungsquelle BESSY II

BESSY II ist eine Synchrotronstrahlungsquelle der dritten Generation, die elektromagnetische Strahlung für Forschungszwecke erzeugt. Die Anlage mit ihrem 240 m umfänglichen Speicherring steht in Berlin-Adlershof. Ihr Schwerpunkt, die Erzeugung von weicher Röntgenstrahlung, ist einmalig in Deutschland. [Hel] [Lan20]

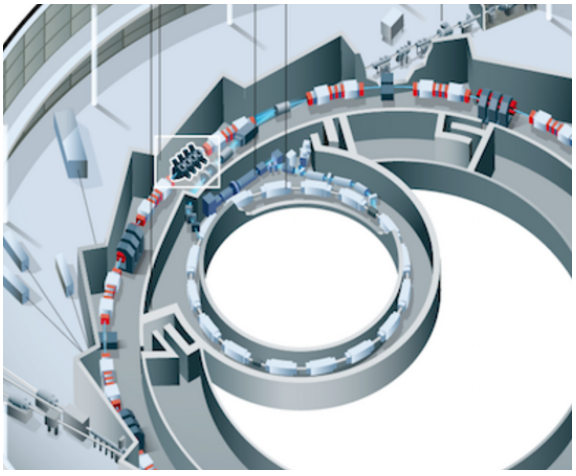


Abbildung 1: BESSY II. Bildquelle: Ausschnitt aus „Aufbau von BESSY II" [Lan20]

Um die weiche Röntgenstrahlung zu erzeugen, werden in BESSY II Elektronen beschleunigt und im Speicherring auf ei-

ner Kreisbahn gehalten. BESSY II wird deshalb auch als „Teilchenbeschleuniger“ bezeichnet.
Die Elektronen werden zu Beginn in einem linearen Vorbeschleuniger auf hohe Geschwindigkeiten gebracht. Daraufhin werden sie in einem Synchrotron (siehe Abb. 1 innerer Kreis) beschleunigt und dann in den evakuierten Ring mit nahezu Lichtgeschwindigkeit gebündelt eingespeist (siehe Abb. 1 äußerer Kreis). Um die Elektronen auf ihrer Kreisbahn zu halten, werden sie von Magneten immer wieder umgelenkt. Die bei der Umlenkung entstehende Synchrotronstrahlung kann an den vom Speicherring tangential abgehenden Experimentierstationen von den Wissenschaftler:innen genutzt werden. [Lan20]

Das Helmholtz-Zentrum Berlin für Materialien und Energie (HZB) ist für den Betrieb der Anlage zuständig. Damit zählt BESSY II zu den Großgeräten der Helmholtz-Gemeinschaft Deutscher Forschungszentren. Im Jahr werden durchschnittlich 2700 Besuche von Gastforschenden aus dem In- und Ausland verzeichnet. Sie schätzen die hohe Zuverlässigkeit und Stabilität der Röntgenstrahlung für die Experimente. Während der Laufzeit von BESSY überwachen Mitarbeiter:innen vom HZB die Qualität des Elektronenstrahls. Sie überprüfen im Kontrollraum die an verschiedenen Bildschirmen angezeigten Statusinformationen von BESSY II. [Hel]
Eine der Hauptaufgaben des HZBs ist, es die Software, mit der der Betrieb von BESSY II überwacht und gesteuert werden kann, zu pflegen. Sie muss regelmäßig gewartet und weiterentwickelt werden. Die vorliegende Bachelorarbeit befasst sich mit einer solchen Weiterentwicklung. Dabei wird das bereits vorhandene mit EPICS implementierte Prozessleistsystem weiter ausgebaut. Das Ergebnis soll einen kleinen Beitrag zu der komplexen Steuerung von BESSY II leisten.

1.1.2 EPICS

Bei großen Experimenten wie am BESSY II müssen die vielen involvierten Geräte und Recheneinheiten miteinander verknüpft und kontrolliert werden. Diese Problematik wird u.a. auch vom „Internet of Things“ (IoT, auf deutsch Internet der Dinge) aufgegriffen. Auch beim IoT geht es darum, viele verschiedene Recheneinheiten, Sensoren und Speichermöglichkeiten miteinander zu verbinden. [Gre21, S.35 f.] Der Anwendungsbereich vom IoT erstreckt sich von Smarthomes über Verkehrsanalyse bis hin zur Steuerung von Geräten in der Industrie.
Heute gibt es viele verschiedene Bausteine, die IoT unterstützen. Gerade die Entwicklung von Mikrochips, Netzwerkprotokollen und Clouds haben dazu beigetragen, dass IoT in seiner Vielfältigkeit möglich ist.

In den 1980ern stand IoT jedoch noch am Anfang seiner Entwicklungsgeschichte. Damals gab es z.B. erste Versuche, alltägliche Gegenstände mit Sensoren, Rechen- und Verbindungseinheiten zu versehen. Diese Ansätze waren aber noch nicht so ausgereift, um ein ganzes Großexperiment ähnlich dem zu BESSY II durchzuführen. Deshalb wurde damals in Großlaboren nach einer eigenen Lösung gesucht und EPICS (Experimental Physics and Industrial Control System), eine Software für Prozessleistsysteme (engl. distributed control system), entwickelt. Mit Hilfe von EPICS können viele verschiedene Geräte miteinander verknüpft und über das Abfragen von Prüfständen einige hundert mal pro Sekunde überwacht werden.
Ursprünglich entstanden ist EPICS Ende der achtziger Jahre in Zusammenarbeit von Mitarbeiter:innen des *Los Alamos National Laboratory* und des *Argonne National Laboratory*, wurde aber seitdem immer weiter auch von anderen Großlaboren ausgebaut. Besonders durch das Aufheben der Lizenzbeschränkungen ca. zehn Jahre nach der Veröffentlichung wurde der Ausbau weiter vorangetrieben. EPICS wird heute sowohl in der Wirtschaft, als auch weiterhin in der Wissenschaft, so auch am BESSY II, eingesetzt und kontinuierlich verbessert. [KGLT94]

1.1.3 Architektur

Die Architektur von EPICS gibt Aufschluss über dessen Funktionsweise und soll deshalb im Folgenden genauer beleuchtet werden (siehe Abb. 2).

Die Basis für EPICS legen die Input/Output Controller (IOC, siehe Input-Output Abb.2). Sie erfassen die Statusinformationen der in das Prozessleitsystem integrierten Messgeräte und Recheneinheiten. Auf ihnen baut die verteilte Laufzeitdatenbank (distributed run time database) auf. Die Datenbank ist für das Speichern von Daten zuständig. Ihr Anwendungsbereich ist jedoch umfassender als der einer klassischen Datenbank. Sie stellt weitere Funktionen wie die Schleifensteuerung oder Alarmerkennung zur Verfügung. Diese werden dann von anderen Subsystemen wie dem Alarmmanager oder dem

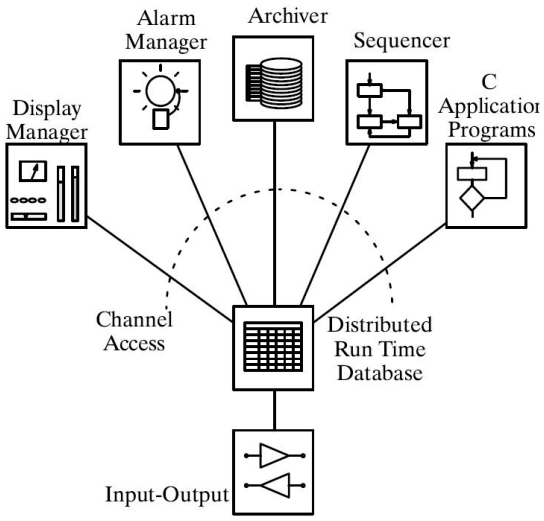


Abbildung 2: Architektur EPICS. Bildquelle: Figure 1 in [DKK91]

Sequencer benutzt. [DKK91] Dabei ist besonders zu beachten, dass es sich hierbei um eine verteilte Datenbank handelt. Bei verteilter Datenverarbeitung wird ein Problem in mehrere Aufgaben aufgeteilt, die jeweils von einer oder mehreren Recheneinheiten gelöst werden. [Pat19] Auch hier wird das Kontrollsystem mit der Datenbank auf die verschiedenen IOC verteilt, sodass mehrere IOC gemeinsam für das Kontrollsystem arbeiten. Deshalb wird das Prozessleitsystem von EPICS als verteiltes Steuerungssystem bezeichnet.
Die in Abb. 2 gezeigten Subsysteme greifen über den Channel-Access auf die verteilte Datenbank zu. Ähnlich einer Server-Client-Verbindung oder Anwendungen im IoT werden hier über die Netzwerkprotokolle TCP/IP und UDP/IP Daten übertragen. In dem Subsystem Display Manager können z.B. die Daten aus der Datenbank über einen Channel-Access ausgelesen und in grafischen Oberflächen für den Nutzer angezeigt werden. [DKK91] Solche grafischen Oberflächen sind z.B. im Kontrollraum für BESSY II zu finden.
Vorteilig bei der Verwendung von EPICS ist der niedrige Programmieraufwand. Das mit EPICS implementierte Prozessleitsystem muss nur noch an das Großexperiment angepasst und nicht mehr von Grund auf neu entwickelt werden. Die EPICS-Umgebung unterstützt außerdem Systemerweiterungen auf allen Ebenen und ermöglicht den Benutzer:innen, andere Systeme zu integrieren oder das System an Bedürfnisse anzupassen. [DHK+94]

Mit Hilfe von EPICS lassen sich Prozessleitsysteme erstellen. Anwendungen dieser Systeme unterstützten bei der Datenerfassung, Kontrolle, Regelung, Optimierung von Geräten während des Betriebes. Mit ihnen können die vielen Geräte und Recheneinheiten vernetzt, über Kontrollstrukturen und Feedbacklösungen der Status der Geräte überprüft und hohe Datenmengen erfasst und verarbeitet werden. [DHK+94] Für die vorliegende Arbeit ist vor allem der IOC von EPICS relevant. Das für BESSY II vorhandene Prozessleitsystem soll um einen weiteren solcher IOC erweitert werden.

2 Problemstellung

Um einen reibungslosen Betrieb zu garantieren, wird während des laufenden Betriebs von BESSY II der beschleunigte Elektronenstrahl durchgängig überwacht (siehe 1.1.1 Seite 1). Dazu wird der Elektronenstrahl in einen Lichtstrahl umgewandelt und mit Hilfe von CCD-Kameras abgebildet (im Detail 3.2 Seite 8). Aus den sich daraus ergebenden Messdaten werden die Statusinformationen abgeleitet und im Kontrollraum an Bildschirmen sichtbar gemacht.

Bisher werden die Kameras mit LabView einer Software von *National Instruments* ausgelesen. Die mit dem LabView-Programm erzeugte Benutzeroberfläche wird mit Hilfe einer Bildschirmübertragung im Kontrollraum angezeigt. Realisiert wird das durch einen Remote-Zugriff auf den LabView-Programm ausführenden Rechner. Zusätzlich werden vom LabView-Programm Statusinformationen ermittelt und an EPICS weitergeleitet.
Aus diesem Aufbau ergeben sich ein Reihe von Nachteilen. Ein Nachteil ist z.B., dass durch die Remote-Übertragung keine vollständige Einbindung in EPICS erfolgt. Ein weiterer finanzieller Nachteil sind die steigenden Lizenzpreise von National Instruments für spezifische Anbindungen wie zu den aktuell verwendeten Kameras. Lizenzen müssen zum Teil für jedes neue Gerät, in diesem Fall jede neue Kamera, neu erworben werden, was sich nachteilig auf einen weiteren Ausbau der vorhandenen Kontrollstrukturen auswirkt. Auch kann die Bestätigung von Lizenzen über externe Netzwerke zu Schwierigkeiten führen, da u.a. aus Sicherheitsgründen BESSY II möglichst von Außenverbindungen getrennt läuft. Dabei ist die Anbindung der Kamera nicht spezifisch an LabView gebunden. Der Kamerahersteller bietet verschiedene andere Möglichkeiten, eine Kamera auszulesen. In der vorliegenden Arbeit soll eine solche Möglichkeit untersucht werden.
Die Frage, der in dieser Bachelorarbeit nachgegangen wird, lautet daher: Gibt es eine alternative Möglichkeit, die Kontrolle des Elektronenstrahls von BESSY II über Kameras in EPICS einzubinden?

In Beantwortung dieser Frage ist das Ziel der Arbeit, eine alternative Lösung zu der LabView-Programmumsetzung auszuarbeiten und zu testen. Diese Lösung soll auf der Programmiersprache Python basieren, eine Analyse der Daten des Elektronenstrahls beinhalten, in EPICS integrierbar sein und Möglichkeiten zum Ausbau bieten. Mit ihr soll der Elektronenstrahl nachweislich ohne LabView kontrollierbar sein.

2.0.1 Anforderungen

Aus dieser Zielsetzung können die folgenden Anforderungen an das finale Python-Skript abgeleitet werden.

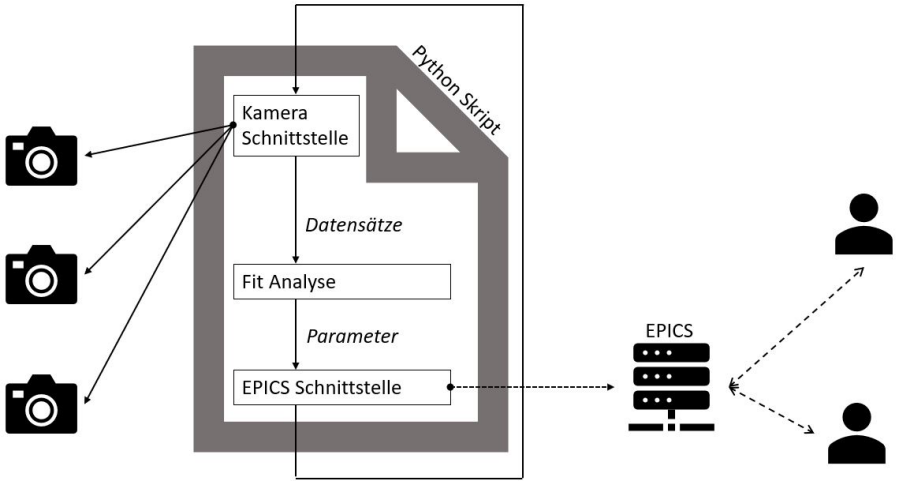


Abbildung 3: Skizze zu den Anforderungen

Die Daten einer Kamera sollen über ihre Gigabit-Ethernet-Schnittstelle ausgelesen werden. Wünschenswert aber nicht erforderlich wäre es, wenn auch mehrere verschiedene Kameras betrachtet werden könnten.
Im nächsten Schritt sollen die erhaltenen Datensätze aufbereitet und unter Zuhilfenahme eines Fit-Algorithmus analysiert werden. Die sich aus der Analyse ergebenden Statusinformationen sollen daraufhin über eine Schnittstelle zu EPICS an das Prozessleitsystem von BESSY II weitergeleitet werden. Die eingespeisten Informationen können dann von Mitarbeiter:innen des HZBs z.B. im Kontrollraum ausgelesen werden. Es wäre dann kein Remote-Zugriff mehr nötig (siehe Abb. 2.0.1).

Eine weitere wichtige Anforderung ist die ausführliche Dokumentation der Arbeitsschritte. Die Einbindung verschiedener Bibliotheken in Python soll auch später noch nachvollziehbar sein. Zudem soll auf die Strukturierung des Programmcodes besonders Wert gelegt werden. Der Code soll für Nachfolgende, die weiter daran arbeiten, gut verständlich sein und Möglichkeiten zur Erweiterung bieten.
Das Auswerten von Lichtstrahlen mit Hilfe von Kameras ist nicht nur für den Betrieb von BESSY II interessant. Wenn der Code leicht anzupassen ist, könnten auch andere Strahlungsquellen davon profitieren. Des Weiteren soll auf die Performanz der Datenauswertung ein besonderes Augenmerk gelegt werden. Die Kameras können mehrere Bilder pro Sekunde produzieren und EPICS genauso häufig den Status überprüfen. Deshalb ist die Datenauswertung so weit wie möglich zu optimieren.

3 Umsetzung

Nachdem zuvor Zielsetzung und Anforderungen erläutert wurden, folgen nun der Prozess der Vorbereitung, die Erarbeitung der einzelnen Umsetzungsschritte und deren Zusammenführung bis hin zur Implementierung von Anforderungen, die sich während der Entwicklung ergeben haben.

3.1 Vorbereitung

Die Problemstellung wurde in Zusammenarbeit mit dem Betreuer der Bachelorarbeit Hr. Dr.-Ing. Rehm erarbeitet. Um ein besseres praktisches Verständnis zu bekommen, wurde BESSY II in Adlershof besucht. Es folgte eine Einarbeitung in die Thematik EPICS und das Vertrautmachen mit den Standards für die Kameraschnittstelle.
Um das Ziel zu erfüllen, von LabView unabhängig zu sein, wurden das bisherige Programm und die Schnittstelle von LabView zwar mit einbezogen, aber nicht als Vorlage verwendet. Mit der Entwicklerin der bisherigen Lösung, Fr. Inés Seiler, wurde sich im Laufe des Arbeitsprozesses immer wieder ausgetauscht.
Für ein abgesichertes Arbeiten wurde ein GitHub-Repository für das Projekt angelegt und die Arbeitszeit über gepflegt.

Es gab verschiedene Überlegungen zur Gestaltung des Arbeitsprozesses. Bei der Analyse von Bildern ist es naheliegend, diese auch grafisch darzustellen. Erste Ideen, dafür eine Benutzeroberfläche zu entwickeln, wurden verworfen. Die einzelnen Programmierschritte wurden stattdessen in Jupyter Notebooks festgehalten. Jupyter Notebook ist eine Entwicklungsumgebung für Python, mit der einzelne Codeblöcke ausgeführt und mit Textblöcken kombiniert werden können. Die Umgebung ist ideal, um Programmcode zu erklären, erlaubt aber vor allem schnelles Ausprobieren ähnlich einer interaktiven Konsole [PM22]. Entstandene Grafiken werden unkompliziert eingebunden, im Dokument gespeichert und müssen beim erneuten Öffnen des Dokuments nicht neu berechnet werden. Daher ist eine Visualisierung der Daten mit Jupyter Notebooks genauso möglich, spart aber die Zeit, die eine Benutzeroberflächenimplementierung kosten würde.

Wie in Abbildung 3 ersichtlich gibt es drei große Hauptumsetzungsschritte: das Anbinden der Kamera, das Auswerten der Daten und die EPICS-Schnittstelle. Alle drei Schritte konnten unabhängig voneinander betrachtet und einzeln untersucht

werden, bevor sie zu einem großen Gesamtkonzept zusammengefügt wurden. Mit Hilfe von bereits erwähnten Jupyter Notebooks wurden für die einzelnen Schritte die dazugehörigen Bibliotheken und Installationsbesonderheiten dokumentiert und wichtige Einstellungen und Funktionsweisen untersucht. Daraufhin wurden die einzelnen Schritten mit PyCharm, einer Programmierunmgebung für Python, zusammengeführt. Weiterführende Umsetzungsschritte wurden implementiert und Tests durchgeführt.

3.2 Kameraanbindung

Der erste große Umsetzungsschritt ist die Kameraanbindung. Um die Anbindung der CCD-Kamera mit Gigabyte-Ethernet-Schnittstelle umsetzen zu können, wurde eine Beispielkamera zur Probe ausgelesen. Hierbei wurde erst die Schnittstelle getestet und dann der Aufbau um die Kamera so verändert, dass die entstandenen Aufnahmen einer echten Aufnahme im Betrieb möglichst ähnlich sehen. Mit den entstandenen Probebilder konnten daraufhin erste Daten analysiert werden.

Bei der Beispielkamera handelt es sich um eine „Prosilica GT1290“ vom Hersteller Allied Vision. Sie kann 1456x1936 Pixel große, 12 oder 14 Bit tiefe, monochrome Bilder mit einer maximalen Geschwindigkeit von 33,3 Bildern pro Sekunde auslesen. [All21] Das gleiche Modell wird auch momentan zur Datenaufnahme im Betrieb verwendet.
Der Sensor in der Kamera ist ein CCD-Sensor. Auf Basis von CCD-Sensoren wurden in den späten 60er Jahren erste digitale Kameras entwickelt. Heute ist die Technologie relativ ausgereift. Hoch perfomante CCD-Kameras, wie sie häufig in der Wissenschaft eingesetzt werden, liefern inzwischen effiziente Lichterkennung. [Oxf] Der CCD-Sensor bestimmt außerdem, dass es sich bei der Kamera um ein serielles Auslesegerät handelt. In diesem Fall werden die Informationen über Ethernet mit einer Datenrate von bis zu einem Gigabit pro Sekunde übertragen.

3.2.1 Kameraschnittstelle

Für das Auslesen der Kamera im Python-Code wird eine Schnittstelle benötigt. Die Schnittstelle soll dafür sorgen, dass sich im Python-Code mit der Kamera verbunden werden kann und die Bildaufnahmen für die weitere Verarbeitung ausgelesen werden können. Eine weitere Anforderung an die Kameraschnittstelle ist, dass Kameraeinstellungen verstellbar sein sollen. Kameraeigenschaften, wie Belichtungseinstellungen oder Auslöser via externem Triggersignal, sollen für verschiedene Anwen-

dungszwecke der Kamera anpassbar sein. Solche Schnittstellen sind bereits implementiert. Sie mussten jedoch noch getestet und auf die Anforderungen überprüft werden.

3.2.1.1 Vimba Python API

Die erste untersuchte und bereits implementierte Schnittestelle nennt sich Vimba. Vimba ist die vom Hersteller mitgelieferte Software für die Kameraansteuerung. Sie ist jedoch in den Programmiersprachen C, bzw C++ implementiert. Um die Programmierung von Allied Vision Kameras mit weniger Codezeilen in Python zu ermöglichen, gibt es dazu eine Anwendungsschnittstelle (auf englisch Application Programming Interface - API) in Python, die die verschiedenen Programmiersprachen verbindet. Die Vimba Python API bietet dieselben Funktionen wie die Software, auf der sie aufbaut. [All20] So können u.a. Kameraeinstellungen angepasst und Bilder im numpy-Array Format ausgelesen werden, was sich gut für Datenverarbeitung anbietet (mehr dazu 3.3) .
Für die Schnittstelle spricht zudem eine gut gepflegte Dokumentation.

3.2.1.2 Harvester

Eine weitere Schnittstelle zur Kamera kann mit der Python Bibliothek Harvester hergestellt werden. Harvester ist im Vergleich zur Vimba Python API eine öffentliche, von Firmen unabhängige Bibliothek, die von einem internationalen Team von motivierten Mitwirkenden entwickelt wurde. [The]
Sowohl die Vimba Python API als auch Harvester basieren auf den Normen des sogenannten GenICam (Generic Interface for Cameras)-Standard, wobei Harvester mehr darauf ausgelegt ist, möglichst viele Hersteller und Geräte zu unterstützen. Die Vimba Python API hingegen ist speziell für Nutzer:innen von Allied Vision konzipiert.
Die Motivation hinter dem GenICam-Standard ist, dass nicht für jedes neues Kameramodell eine von Grund auf neue Schnittstelle entworfen werden muss. Der seit 2006 immer weiter entwickelte GenICam-Standard ist ein erfolgreicher, effizienter, branchenweiter Ansatz, um Bildverarbeitungsanwendungen schneller entwickeln zu können. Ein einmal implementierter Quellcode kann Dank des Standards für andere Projekte mit unterschiedlichen Kameras und Sensoren wiederverwendet werden. [All17]

Der Standard setzt sich aus mehreren Modulen zusammen. Standardisierte Funktionen für die Wiederverwendung von Code definiert das SFNC (Standard Features Naming Convention) Modul. Es definiert sowohl die Bezeichnungen der Kameraeigenschaften als auch, was über diese Eigenschaften eingestellt werden kann. Zum Beispiel ist als Bezeichnung für das Starten einer Bilderfassung nur „AcquisitionStart“ erlaubt. Trotzdem ist die SFNC flexibel genug, um auch herstellerspezifische Merkmale unterzubringen.

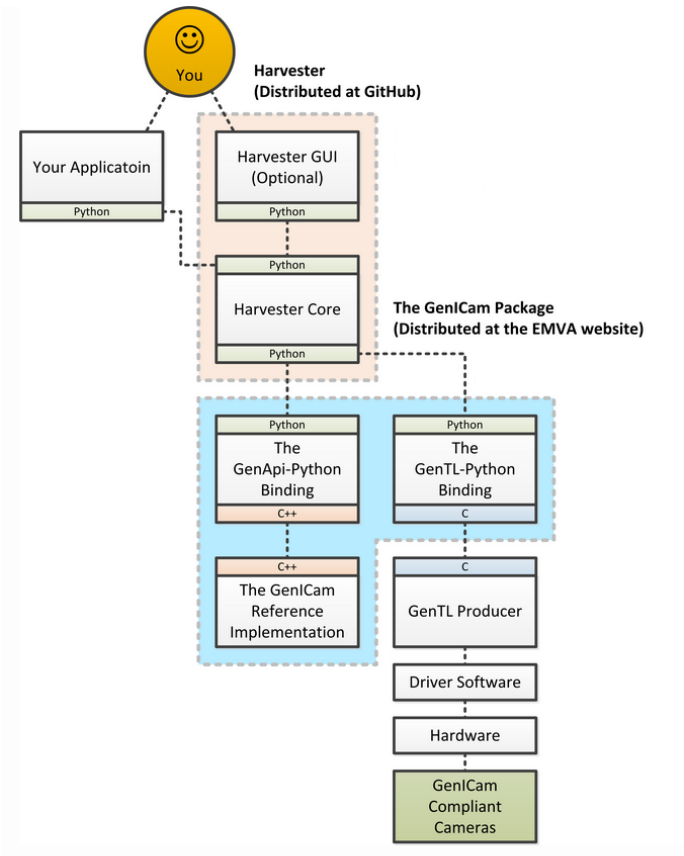


Abbildung 4: Architektur Harvester.
Bildquelle: [The]

Weiterhin entscheidende Module für die Umsetzung einer unabhängigen Schnittstelle sind die GenAPI und die GenTL. Die GenAPI ist eine Anwendungsschnittstelle, die die Steuerung der Kamerafunktionen durch den/die Benutzer:in ermöglicht. Die GenTL (Generic Transport Layer) hingegen übernimmt die physikalischen Aspekte. Sie ermöglicht z.B. das Auflisten aller verfügbaren Kameras und sorgt für den Transport der Bilddaten von der Kamera (dem GenTL-Produzenten) zum Host (dem GenTL-Konsumenten).
Der GenICam-Standard mit seinen Modulen ist inzwischen in Hunderttausenden von verkauften Kameras integriert und gewinnt weiterhin an Bedeutung. So wird immer mehr Kund:innen und Entwickler:innen bewusst, dass durch das unkomplizierte Tauschen von Kameras bessere Leistungen erbracht oder Kosten gespart werden können. [All17] [Die]Die European Machine Vision Association EMVA sorgt außerdem dafür, dass die Entwicklung des Standards durch aktiv beitragende Unternehmen weiterhin vorangetrieben wird.
Von der EMVA wird auch eine Python-Schnittstelle zu der GenAPI und der GenTL zu Verfügung gestellt. [EMV] Diese ist wiederum Grundlage für Harvester. In Abbildung 4 sind die soeben erläuterten verschiedene Ebenen, auf denen Harvester aufbaut, zu erkennen und zwar von den GenICam kompatiblen Kameras bis hin zum Harvester-Core.

Sowohl die Vimba Python API als auch die Harvester Bibliothek erfüllen die Anforderungen für die Kameraschnittstelle. Für eine Entscheidung zwischen den beiden Bibliotheken ist eine weitere zu Beginn gestellte Anforderung nach Flexibilität ausschlaggebend gewesen. Wie in 2.0.1 aufgeführt soll der Code am Ende auch für andere ähnliche Aufbauten von Nutzen sein. Harvester ist vom Hersteller unabhängig, dementsprechend flexibler in seinem Einsatzbereich und damit die präferierte Wahl.
Die Kamera konnte auch mit Harvester angesteuert werden. Einstellung konnten an den Anwendungszweck angepasst werden, aber die Bilderfassung funktionierte nicht zuverlässig. Ohne erkennbaren Regeln zu folgen, wurde häufig nach dem Neustart der Kamera ein Bild erhalten, aber dann stundenlang nicht mehr. Um das Problem zu lösen, wurden zuerst mit Fr. Seiler (siehe 3.1) jegliche Kameraeinstellungen überprüft und getestet. Das Problem blieb weiterhin bestehen. Als auch nach tagelangem tieferen Einarbeiten in Harvester keine Lösung gefunden werden konnte, wurde eine alternative Umsetzung mit der Vimba Python API implementiert. Eine weitere Fehlersuche in den in Abbildung 4 abgebildeten unter Harvester liegenden Ebenen hätte nicht in den zeitlichen Rahmen der Bachelorarbeit gepasst und musste auf spätere Weiterentwicklungen verschoben werden. Um aber solche späteren Entwicklungen vorzubereiten, wurde entschieden, die Bilderfassung im finalen Programmcode möglichst austauschbar zu gestalten.

3.2.2 Erfassen von beispielhaften Aufnahmen

Nachdem die Bilderfassung über die Kameraschnittstelle realisiert werden konnte, wurde der Aufbau um die Kamera so verbessert, dass ein möglichst ähnliches Bild zu einem Originalbild während des Betriebes entsteht.

Damit verständlich wird, wie ein solches Originalbild aussieht, sollen im Folgenden die zwei verschiedene Möglichkeiten der Bildaufnahme zur Elektronenstrahlkontrolle erläutert werden. Abbildung 5 zeigt eine vereinfachte Darstellung von BESSY II (siehe Abb. 1). Mit dem rechten rötlichen Quadrat in der Abbildung ist die erste Möglichkeit verdeutlicht. Hier werden die Elektronen auf fluoreszierende Schirme geleitet, die das für die

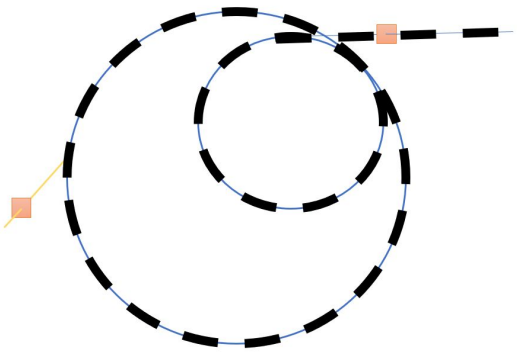


Abbildung 5: Platzieren der Kamera in BESSY II

Kameras sichtbare Licht erzeugen. Das für das Bild verwendete Elektronenbündel ist danach nicht mehr zu gebrauchen. Deshalb wird dieser Aufbau häufig hinter dem linearen Vorbeschleuniger verwendet, bevor die Elektronen sich in den Kreisen bewegen.
Um die kreisenden Elektronen zu kontrollieren, gibt es eine zweite Möglichkeit der Bildaufnahme (siehe rötliches Quadrat links in Abb. 5). Hierbei wird die Strahlung aufgenommen, die auch an den Experimentierstationen verwendet wird. Je nach Aufbau wird das Licht mit verschiedenen Optiken aufbereitet. Die Aufbereitung mit einer Lochoptik hat zum Beispiel zur Folge, dass der Elektronenstrahl in verschiedenen Ausprägungen mehrfach abgebildet wird (siehe Abb.7), wohingegen bei der ersten Möglichkeit nur ein Profil zu erkennen ist (siehe Abb. 6).
In beiden Fällen ist auf der Kamera das transversale Profil des Elektronenstrahls zu sehen. Die Bilddaten können zur Analyse verwendet werden, solange die Proportionen des abgebildeten Profils mit der Realität übereinstimmen.

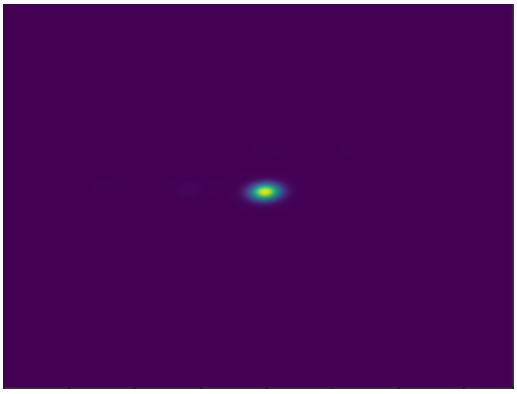


Abbildung 6: Elektronenstrahl hinter fluoreszierendem Schirm. Bildquelle: siehe 3.3 eigene Aufnahme an MLS

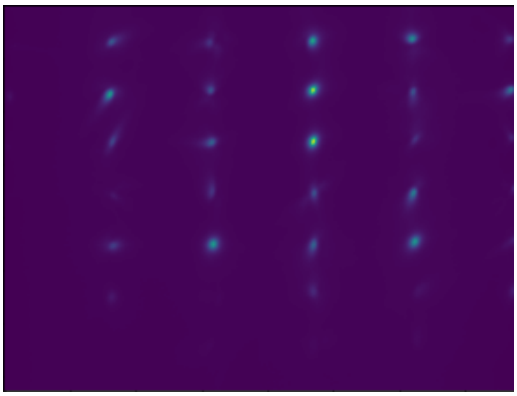
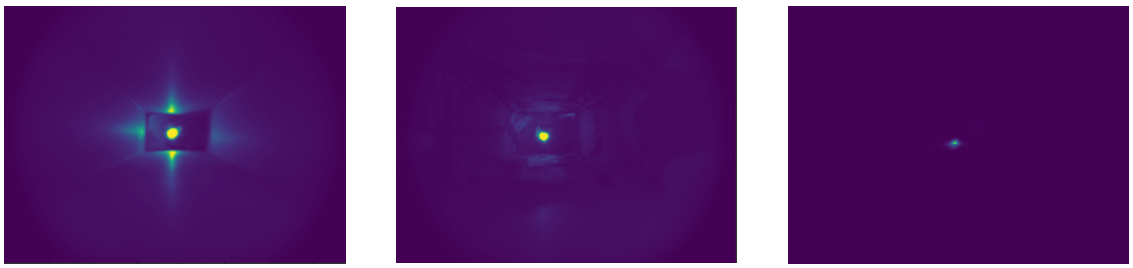


Abbildung 7: Elektronenstrahl hinter Lochoptik. Bildquelle: HZB interne Dokumentation

Die entstehenden Bilder liegen in Grautönen vor. Das ist darauf zurückzuführen, dass die verwendeten Kameramodelle nur monochrone Aufnahmen machen (siehe Absatz 3.2). Um eine bessere Visualisierung durch Kontrast zu erzeugen, wurden in der vorliegenden Arbeit alle Aufnahmen eingefärbt.
In Abbildung 6, 7 ist zu erkennen, dass ein Großteil eines Bildes zur Elektronenstrahlkontrolle dunkel erscheint. Es gibt einzelne maximal helle Pixel im Bild, die auf den Abbildungen aber auf Grund der schwachen Auflösung nicht zu erkennen sind. Diese sogenannten Hotpixel müssen in der Datenauswertung berücksichtigt werden.
Die auf den Abbildungen erkennbaren hellen Punkte sind Abbilder des Elektronenstrahls. Um einen solchen hellen Punkt auf einem dunklem Hintergrund mit der Beispielkamera zu erzeugen, musste die lichtsensitive Kamera gut abgedunkelt

werden. Dafür wurde sie in einen Tunnel aus Pappe gelegt (das entstandene Bild siehe Abb. 8a). Der mit einer kleinen Lampe erzeugte Lichtstrahl, der den Elektronenstrahl simuliert, reflektierte jedoch an den Außenwänden des Tunnels. Deshalb wurde dieser mit dunklem Stoff ausgelegt (siehe Abb. 8b). Weitere Restreflektionen konnten verhindert werden, indem die Lichtstärke des simulierenden Lichtstrahl durch Wechseln der Lichtquelle und durch weiteres Abdunkeln mit Stoff verringert wurde (siehe Abb. 8c). Der helle Punkt auf einer solchen Aufnahme ähnelt einer Elektronenstrahlabbildung hinter einer Lochoptik.
Mit diesem zufriedenstellenden Aufbau konnten eine erste Reihe von Daten für die Auswertung erhoben werden. Durch Bewegung des Lichtstrahls konnte sogar die zu kontrollierende Bewegung des Elektronenstrahl simuliert werden.



(a) im Tunnel (b) absorbierender Tunnel (c) simulierte Bilddaten

Abbildung 8: Entwicklung der Probeaufnahmen

3.3 Datenanalyse

Die von der Kamera produzierten Daten können den Elektronenstrahl visualisieren, eine genauere Analyse der Daten kann jedoch weitere Details hervorbringen. Für die Steuerung von BESSY II werden bisher nicht nur die Positionswerte des Elektronenstrahls auf dem Kamerabild in Betracht gezogen, sondern auch die Verteilung des Elektronenstrahls. Um die bisherige Umsetzung mit LabView ablösen zu können, soll auch dieser Teil in Python implementiert werden. Im Folgenden wird erläutert, wie aus den Bilddaten spezifische Informationen extrahiert werden, die dann an das Prozessleitsystem weitergegeben werden können.

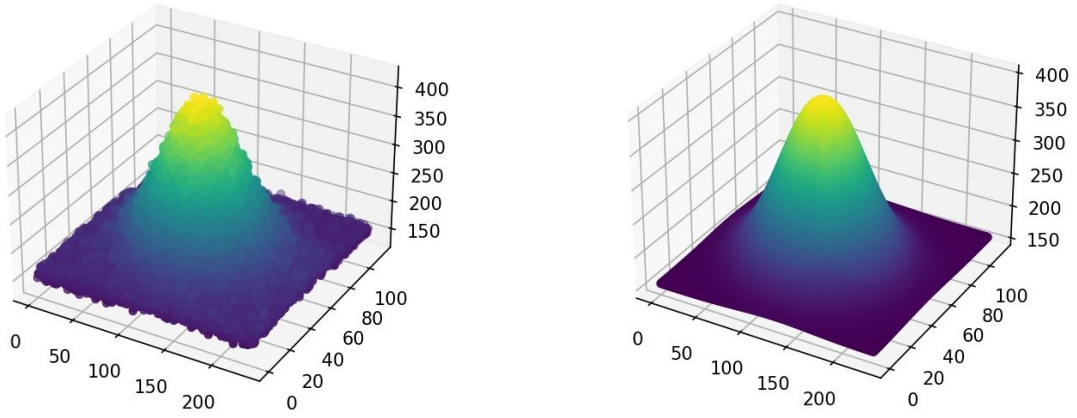
Die entworfene Datenanalyse wurde mit Bilddaten getestet. Zu Beginn beruhten diese Daten auf den in Abschnitt 3.2.2 beschriebenen Aufnahmen. Die Verteilung um den hellen Punkt konnte jedoch nicht zufriedenstellend simuliert werden. Es konnten keine Daten während des Betriebes von BESSY II aufgenommen werden, da diese Arbeit in einer Zeit entstanden ist, in der BESSY II nicht in Betrieb war.
In Adlershof gibt es ein weiteres nicht ganz so großes Experiment wie BESSY II, bei dem über Kameras ein Elektronenstrahl überwacht wird. Die Metrology Light Source

(MLS) ist ein spezieller Niedrigenergiespeicherring für die Metrologie mit Synchrotronstrahlung, an der Präzisionsmessungen und Kalibrierungen im Spektralbereich vom fernen Infrarot bis zum extremen Ultraviolett angeboten wird. Auch hier wird ein Vorbeschleuniger betrieben, an dem mit Hilfe von fluoreszierenden Schirmen der Elektronenstrahl sichtbar gemacht und von Kameras aufgenommen wird. [Phy]
Während eines Testlaufes an der MLS war es möglich, die entsprechende Kamera mit der Vimba Python API anzusteuern, auszulesen und die Bilddaten für spätere Testzwecke abzuspeichern. Der auf den Bilddaten sichtbare Lichtfleck für den Elektronenstrahl enthält die gleichen Informationen, die eine Aufnahme an BESSY II enthalten würde.

Die Bilddaten liegen in Form eines numpy-Arrays vor. Ein Array ist eine zentrale Datenstruktur in der Python Bibliothek numpy. Bei einem Array werden die Werte in einem Gitter angeordnet, das Informationen über die Rohdaten enthält und festlegt, wie ein Element zu lokalisieren und interpretieren ist. Die Elemente im Gitter können auf verschiedene Weise indiziert werden und sind alle vom gleichen Datentyp. [Num] [Wei19, S.865 ff.]

3.3.1 Fit-Analyse

Kernstück der Datenanalyse ist eine Fit-Analyse. Bei einer Fit-Analyse wird ausgenutzt, dass bekannt ist, wie die Daten gedeutet werden können. So ist bei der Analyse der Bilddaten des Projektes schon bekannt, dass die Abbildung des Elektronenstrahls einer gaußschen Normalverteilung ähnelt. Das bedeutet, die Helligkeitswerte der Datenpixel ähneln einer 3D-Projektion einer gaußschen Glocke (siehe Abb. 9a). Die mathematische Funktion zur Beschreibung einer gaußschen Glocke ist bekannt. In dieser Formel gibt es Parameter, die die Eigenschaften der Glocke bestimmen. Sie beeinflussen u.a. wie hoch und breit die Glocke ist. Ziel einer Fit-Analyse ist es, die Parameter einer Glocke so zu bestimmen, dass diese den Bilddaten möglichst ähnlich sehen (siehe Abb. 9b).



(a) Beispiel für ein mögliches Elektronenstrahlabbild

(b) An die Bilddaten von 9a angepasste gaußsche Glocke

Abbildung 9: 3D-Projektion von Bilddaten und angepasster gaußscher Glocke

Um die Parameter zu bestimmen, wird bei der Fit-Analyse ein Fit-Algorithmus benutzt. Er nimmt sich die von der Funktion unabhängigen Daten (in diesem Fall die x- und y-Werte der Bilddaten), die Bilddaten und die vermutete Funktion (auch als Modell bezeichnet) und bestimmt die besten Parameter, mit der die Funktion die Bilddaten beschreibt. Dazu werden im Algorithmus verschiedene Parameterwerte ausprobiert. Mit den unabhängigen Daten und dem Model mit seinen aktuellen Parameterdaten werden Testdaten erzeugt. Diese Testdaten werden mit den Bilddaten verglichen. Sind sich Testdaten und Bilddaten ähnlich genug, liefert der Fit-Algorithmus die Parameter als Ergebnis zurück. Sind die Testdaten noch zu weit von dem Bilddaten entfernt, werden die Parameter weiter verändert, bis zufriedenstellende Testdaten entstehen (siehe Abb. 10). Der Fit-Algorithmus bekommt also Bilddaten, deren x- und y- Werte und ein Modell und liefert die besten Parameter für dieses Modell auf die Bilddaten angewandt.

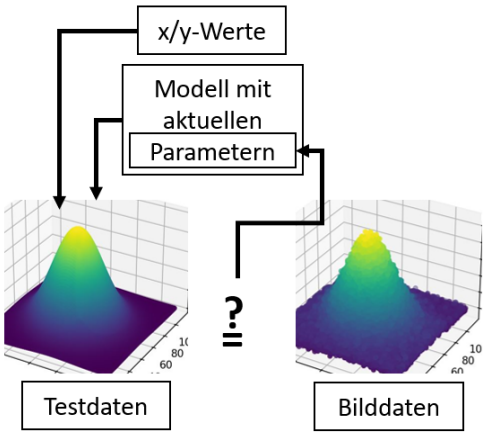


Abbildung 10: Funktionsweise des Fit-Algorithmus

Die x- und y- Werte sind nicht in den von der Kamera gelieferten Bilddaten enthalten. Sie müssen explizit erstellt werden und haben die gleiche Arraystruktur wie die Bilddaten, nur dass an Stelle der Helligkeitswerte Koordinaten stehen (siehe Abb. 11).

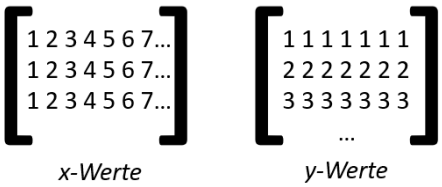


Abbildung 11: Vereinfachte x- bzw. y-Werte

Bevor die Bilddaten vom Algorithmus verarbeitet werden können, müssen sie gegebenenfalls zusätzlich aufbereitet werden. In Abschnitt 3.2.2 wurde bereits erwähnt, dass Hotpixel bei der Auswertung mitbeachtet werden müssen. Diese einzelnen Pixel mit Maximalwerten sind auf die technischen Gegebenheiten der Kamera zurückzuführen und verfälschen das Ergebnis. Eine erfolgreiche Lösung, um Ausreißer in einem Bild zu eliminieren, ist ein sogenannter Median-Filter. Bei einem Median-Filter wird jedes Pixel des Bildes der Reihe nach durchgegangen und dessen Nachbarn betrachtet, um zu entscheiden, ob das Pixel repräsentativ für seine Umgebung ist oder nicht. Der Pixelwert wird durch den Median seiner benachbarten Pixel ersetzt. [RFW] [Wei19, S.893 f.] Die Hotpixel sind nicht repräsentativ für ihre Umgebung. Ihr Wert wird durch den Median ihrer Umgebung ersetzt und sticht dadurch nicht mehr hervor.
Durch den Filter werden die Bilddaten verändert. Je stärker die Änderung zwischen benachbarten Pixeln, desto größer die Auswirkung des Filters auf den Informationsgehalt der Bilddaten. Eine Elektronenstrahlabbildung, die sich nur über eine geringe Pixelmenge erstreckt, könnte stark verfälscht werden. Deswegen ist es sinnvoll, eine Möglichkeit zur Medianfilterung in der Datenanlyse zu bieten, die von Benutzer:innen ein- und ausgestellt werden kann. Hinzu kommt der zeitliche Aspekt der Medianfilterung. Testungen haben ergeben, dass die Median-Filterung für die gesamten Bilddaten der Beispielkamera bis zu 87ms dauern kann, für kleinere Ausschnitte z.B.300x450 Pixel nur 4ms. Der Filter sollte also überlegt eingesetzt werden.

In Python gibt es verschiedene Möglichkeiten, Fit-Algorithmen umzusetzen. Es existieren ausführliche Projekte, wie „Non-Linear Least-Squares Minimization and Curve-Fitting for Python“ (LMFIT), die sich damit beschäftigen, verschiedene wiederverwendbare Modelle zu entwerfen und zur Analyse mit Fit-Algorithmen zu verwenden. [MN] Das in

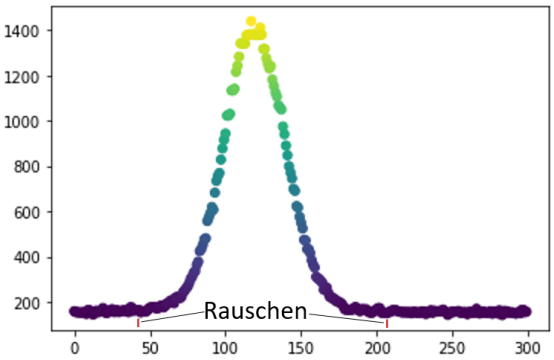


Abbildung 12: Basisrauschen

diesem Projekt implementierte 3DGaussmodel stand erst in engerer Wahl und wurde dann aber auf Hinweis von Hr. Dr.-Ing. Rehm verworfen. Es war nicht in der Lage, die Rotation einer gaußschen Glocke mit einzubeziehen. Stattdessen wurde ein eigenes Modell entwickelt, das nicht nur Rotation, sondern auch das Basisrauschen einer Kamera mit in Betracht zieht. Unter Basisrauschen werden die Werte verstanden, die vermeintlich schwarz sind, aber nicht dem niedrigsten Wert 0 entsprechen. Um eine zu Null konvergierende gaußsche Glocke zu bekommen, muss dieses Rauschen herausgerechnet werden (siehe Abb.12).
Das eigene Modell hat damit die folgenden Fit-Parameter: Amplitude, Position des x-Maximaum, Position des y-Maximum, x-Varianz, y-Varianz, Rotationswinkel und Rauschen. Diese Parameter werden von dem Fit-Algorithmus curvefit von der Python Bibliothek scipy.optimize gefunden.
Beim Testen der Funktionsweise des Modells und des Algorithmus wurde besonders der Performanz der Datenanalyse Beachtung geschenkt. Aus diesem Grund folgte eine Untersuchung von Optimierungsansätzen, für die weitere Tests durchgeführt wurden.
Unter Berücksichtigung dieser Ansätze entstand ein erster Entwurf eines Klassendiagrammes für die Datenanalyse. Im Folgenden wird zuerst auf die Optimierung und dann den Klassendiagrammentwurf eingegangen.

3.3.2 Optimierung

Die Bildanalyse soll möglichst für jedes ausgelesene Bild ausgeführt und damit möglichst mehrmals pro Sekunde durchlaufen werden. Es wäre ungünstig, wenn die Analyse sekundenlang läuft oder zu keinem Ergebnis kommt. Der Erfolg eines Fit-Algorithmus kann durch mehrere Ansätze begünstigt werden. So haben u.a. die Menge der Daten, Wertebereiche der Parameter und Startparameterwerte einen Einfluss auf das Resultat. Inwieweit diese Einflüsse zur Optimierung des Fit-Algorithmus beitragen können, soll im Folgenden untersucht werden.

3.3.2.1 Reduzieren der Datenmenge

Die Kamera kann bis zu 33 Bilder pro Sekunde aufnehmen, d. h. es müssten 93 Mio Pixel pro Sekunde ausgewertet werden. Soll die Auswertung beschleunigt werden, lohnt es sich am Anfang der Analyse, die Menge der Daten pro Bild zu reduzieren. Um Informationen über den Elektronenstrahl zu erhalten, reicht es aus, sich nur einen hellen Punkt, der den Elektronenstrahl abbildet, in seiner unmittelbaren Umgebung anzuschauen. Alle anderen Pixel werden dann nicht mehr benötigt. Hinzu

kommt, dass bei der Bildaufnahme hinter einer Lochoptik (siehe Abb. 7) der für die Analyse genutzte Bereich so festgelegt werden muss, dass nur noch ein einziger heller Punkt ausgewertet wird. Deshalb ist es sinnvoll, dass dieser Bereich manuell von Nutzer:innen bestimmt werden kann. Der erste Schritt die Analyse schneller zu gestalten, ist dementsprechend das große Bild in einen von Nutzer:innen definierten und für die Analyse relevanten Bereich zuzuschneiden, der nur noch eine Elektronenstrahl Abbildung enthält. Ein solcher Bereich wird in der Bildanalyse auch als „region of interest“ (kurz ROI) bezeichnet. [Erd]

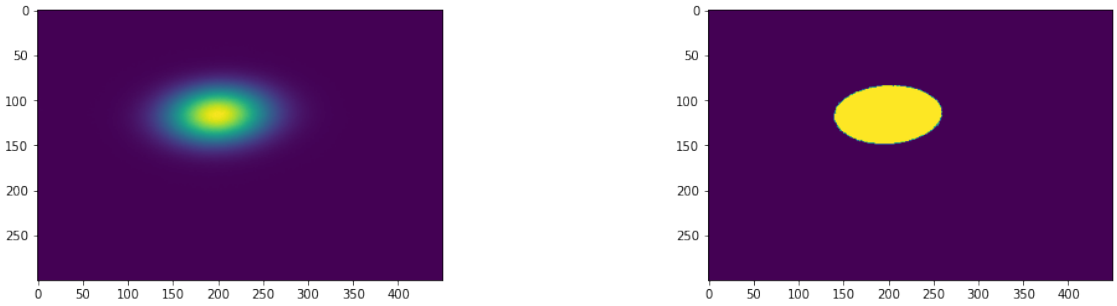
Der manuell von Nutzer:innen ausgewählte Bereich könnte eventuell immer noch sehr groß und damit ineffizient sein. Es ist aber auch nicht sicher, dass der Elektronenstrahl immer genau an der gleichen Stelle in der Bildaufnahme abgebildet ist. Eine eng umschließende ROI, aus der sich der Elektronenstrahl hinaus bewegt, könnte ebenfalls Probleme bereiten.
Es gab Überlegungen, die ROI immer um die Elektronenstrahlabbildung mitlaufen zu lassen. Der Bereich würde sich also an die Position des Strahls anpassen. Diese Variante wurde jedoch abgelehnt, da es dann außer zu Beginn nie eine Möglichkeit für Nutzer:innen gäbe, die ROI während des Betriebs zu beeinflussen. Stattdessen wurde sich mit Hr. Dr.-Ing. Rehm darauf verständigt, die ROI weiterhin manuell zu bestimmen, dann aber innerhalb der ROI den Strahl algorithmisch zu finden und noch einmal um den Strahl herum einen Bereich zu definieren.
Sollte sich der Elektronenstrahl an den Rand der ROI bewegen, kann ein Hinweis an den/die Nutzer:in geschickt werden, sodass diese:r die ROI aktiv verbessert. Ein solcher Hinweis ist wichtig für die Kontrolle des Strahls. Insofern bringt diese Variante nicht nur durch die Verringerung der Datenmenge Laufzeitvorteile mit sich, sondern bietet weitere Kontrollmechanismen. Auch eine gewisse Flexibilität für die Position des Strahls bleibt erhalten, schließlich kann sich der Strahl innerhalb der ROI bewegen.
Die Implementierung einer ROI gestaltet sich unkompliziert. Für das Finden des Elektronenstrahls in der ROI hingegen soll im Folgenden auf zwei verschiedene Ansätze eingegangen werden. Clustering und ein Labelansatz sollen untersucht und verglichen werden.

Clustering

Algorithmisch zu bestimmen, wo sich die Abbildung des Elektronenstrahls in der ROI befindet, lässt sich u.a. mit einer Clusteranalyse realisieren. Die Clusteranalyse ist eine in der Statistik verwendete Methode zur Verarbeitung von Daten. Dabei werden Elemente in Gruppen (Clustern) organisiert, je nachdem, wie ähnlich sie einander sind. [Qua]

Es gibt eine breite Auswahl an Clustering-Algorithmen, die sich vor allem darin unterscheiden, auf welchem Weg sie bestimmen, wie ähnlich zwei Datenpunkte sind. In diesem Fall besteht ein Datenpunkt aus einem Helligkeitswert und einer x- und y- Position im Bild. Je ähnlicher die Helligkeit und je näher sie im Koordinatensystem beieinander liegen, desto ähnlicher die zwei Datenpunkte. Genauso wie für den Fit-Algorithmus müssen die Werte für die x- und y-Position explizit erstellt werden.

Für Clustering-Algorithmen ist die Python Bibliothek scikit-learnarn eine weit verbreitete Möglichkeit. Sie liefert eine Vielzahl ausgefeilter Clustering-Algorithmen. [PVG+11] Eine gute Laufzeit verspricht der K-Means-Algorithmus. Er ist laut Dokumentation universell einsetzbar und geeignet für flache Geometrie mit kleinerer Clusternanzahl. [sci] Erste Tests wurden mit diesem Algorithmus durchgeführt. Ein beispielhafter Test ist in Abbildung 13 dargestellt. Das Elektronenstrahlabbild wurde in den Tests verlässlich gefunden.



(a) Beispielhafte ROI

(b) Clustering Ergebnis

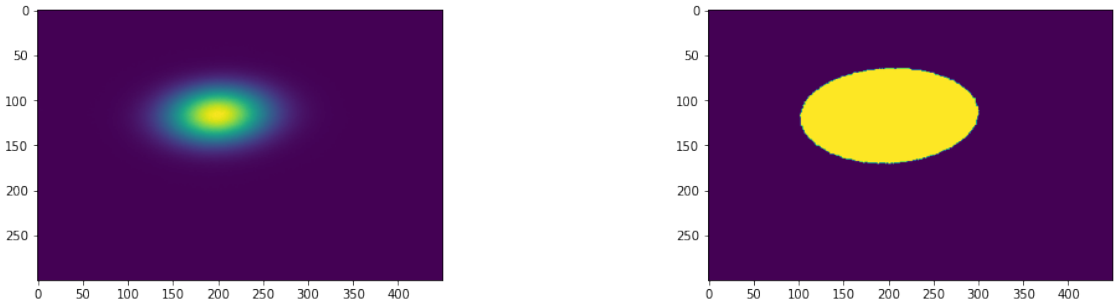
Abbildung 13: Clustering Beispiel. In 13a wurden die in 13b abgebildeten Cluster gefunden. Das Elektronenstrahl-Cluster ist in 13b gelb markiert

Labelansatz

Aus Nachforschungen ergab sich ein weiterer Ansatz, um das Abbild der Elektronen in den Bilddaten zu finden. Entscheidend für diesen Ansatz ist es, dass es sich bei den Daten um Bilddaten handelt, die von ihrer Struktur im numpy-Array aus schon Positionsinformationen enthalten. Speziell für Bilddaten gibt es die Python Bibliothek ndimage.

Beim Labelansatz wird die monochrome Abbildung des Elektronenstrahls nach der Helligkeit ihrer Pixel gefiltert. Ist ein Pixel heller als ein bestimmter Schwellenwert für den Elektronenstrahl, steht im Ergebnis Array eine 1, ansonsten eine 0. Mit Hilfe der ndimage-Funktion label können jetzt alle zusammenhängenden benachbarten Elektronenstrahlpixel bestimmt werden. Dabei müssen nicht explizit x- oder y-Werte erzeugt werden. Die Funktion label kennt die Arraystruktur des Bildes und kann auf dessen Basis benachbarte Pixel bestimmen. Das erfordert weniger Rechen-

leistung, als die Bestimmung von Ähnlichkeit beim Clustering.



(a) Beispielhafte ROI

(b) Ergebnis des Labelansatzes

Abbildung 14: Labelansatz Beispiel. In 14a konnten die in 14b abgebildeten zusammenhängenden Bereiche gefunden werden. Zur Elektronenabbildung gehörige Werte sind in 14b gelb markiert

Das Resultat ist ein numpy-Array in Bilddatenform, das alle gefundenen zusammenhängenden Bereiche darstellt. Einer der Bereiche ist die Elektronenstrahlabbildung. Bei einem guten Schwellenwert wird nur dieser eine Bereich gefunden. Gibt es mehrere Bereiche wird der Elektronenstrahl im größten Teilbereich vermutet. (siehe Abb. 14)
Erste Tests mit dem Labelansatz waren erfolgreich. Die Verlässlichkeit von diesem Verfahren Verlässlichkeit hängt stark vom Schwellenwert ab. Ein guter Schwellenwert liefert ein zuverlässiges Ergebnis. Deshalb sollte er bei späterer Verwendung des Ansatzes besonders beachtet werden. Außerdem bei dem Ansatz zu berücksichtigen ist, dass Ausreißer z.B. mit dem Median-Filter eliminiert werden müssen, weil diese sonst das Ergebnis verfälschen würden.

Die beschriebenen Verfahren - Clustering und Labelansatz - sind beide funktionierende Lösungsansätze um die Position der Elektronenstrahlabbildung zu bestimmen und einzugrenzen. Für einen Performanzvergleich wurden mit dem Python Modul timeit eine zeitliche Überprüfung durchgeführt. Beide Verfahren wurden mit der gleichen 300x450 Pixel großen ROI (siehe Abb. 13a und 14b) getestet und dabei mit dem timeit Modul überwacht. Um das Elektronenstrahlabbild zu finden, brauchte Clustering 412 ms und der Labelansatz 9 ms. Damit war der Labelansatz trotz Zeit für den Median-Filter deutlich schneller.
Auch bei den weiteren Clustering Tests zeichnete sich ab, dass K-Means zwar zu den schnelleren Clustering-Algorithmen gehört, aber bei größeren Bilddaten mehrere Sekunden brauchen kann, um zu einem Ergebnis zu kommen. Das wäre deutlich zu viel Zeit für einen Algorithmus, der möglichst 33 mal pro Sekunde die Daten für eine weitere Verarbeitung aufbereiten soll. Clustering-Algorithmen sind leistungsstark, wenn es darum geht, Zusammenhänge in komplexen Daten zu finden. Da hier

die Ähnlichkeit der helleren Daten eigentlich nur ein Mittel zum Zweck ist, um den Elektronenstrahl zu finden, sind Clustering-Algorithmen für diesen Anwendungsfall zu aufwendig und zeitintensiv.
Deshalb wurde der Labelansatz zum Finden der Elektronenstrahlabbildung weiterverfolgt. Im nächsten Schritt wurde für diesen Ansatz getestet, wie er sich zeitlich auf die Fit-Analyse auswirkt. Dazu wurden von einem Bild verschieden große ROIs erzeugt und deren Analysezeit mit und ohne zusätzliche Reduzierung durch den Labelansatz verglichen (siehe Tabelle 1).

Größe der ROI in Pixel	**Analysezeit mit Labelansatz**	**Analysezeit ohne Labelansatz**
1456x1936	146 ms	3.09 s
1000x1450	71,3 ms	1,68 s
300x450	14,4 ms	126 ms
150x200	5,73 ms	9,75 ms

Tabelle 1: Zeitliche Auswirkung des Labelansaztes

Es ist zu erkennen, dass je größer die ROI ist, desto stärker die Optimierung durch den Labelansatz. Wenn die ROI so groß ist, wie das ursprüngliche Bild (erste Zeile in Tabelle 1), benötigt eine normale Analyse sogar Sekunden, um zu einem Ergebnis zu kommen. Mit dem Labelansatz und dem gleichen Bild dauert die optimierte Analyse nur 146ms. Damit ist bestätigt, dass es durch den Einsatz des Labelansatzes möglich ist, die Laufzeit der Datenanalyse zu verringern.
Auf die Qualität der Fit-Analyse hat der Labelansatz kaum eine Auswirkung. Erst bei dem kleinsten Bereich (letze Zeile in Tabelle 1), der die Elektronenstrahlabbildung eng umschließt, konnten Unterschiede in den Fit-Parametern festgestellt werden.

Bildbereiche

Bevor weiter darauf eingegangen wird, was zusätzlich zur Verkleinerung der Datenmenge unternommen werden kann, nochmal eine kurze Zusammenfassung für die verschiedenen Bildbereiche, die durch das Zuschneiden entstehen (siehe Abb. 15). Der größte Bereich sind die eigentlichen Bilddaten, in der Abbildung als *image* markiert. Ein Teil der Bilddaten wird vom Benutzer als *roi* festgelegt. In der *roi* befindet sich das Elektronen-

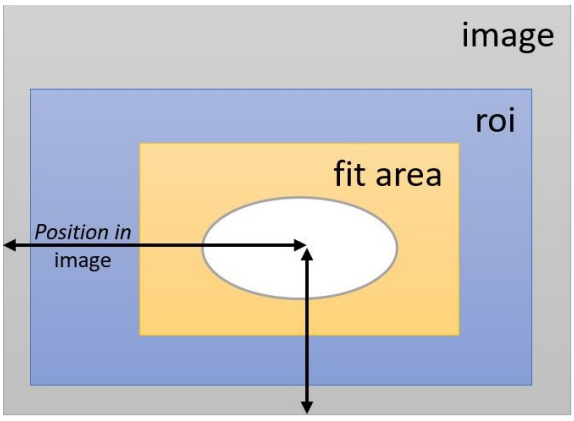


Abbildung 15: Bildbereiche bei der Datenanalyse

strahlabbild (in Abb. 15 ellipsenförmig dargestellt). Unter Verwendung des Labelansatzes kann es lokalisiert und enger umschnitten werden. Der entstandene Bereich ist als *fit area* beschriftet, da dies der Bereich ist, der später vom Fit-Algorithmus analysiert wird.
Bei dem Verkleinern des Bereichs ist zu beachten, dass auch die x- und y-Werte für den Fit-Algorithmus mit angepasst werden müssen. Analysiert der Fit-Algorithmus die *fit area*-Daten, sind die resultierenden Parameter auf diesen Bereich bezogen. Die positionbestimmenden Parameter müssen also auf den Image-Bereich zurückgerechnet werden.

Sampling

Zusätzlich zum Verkleinern der Bereiche kann die Datenmenge weiter reduziert werden, indem nur eine Auswahl der Daten verwendet wird. Dieser Prozess wird auch als Sampling bezeichnet. Als eine mögliche Umsetzung von Sampling wurden die Daten aufgereiht und nur jeder n-te Datenpunkt für die Analyse verwendet. In Abbildung 16 sind die Datensätze für verschiedene Werte von n zu zusehen.

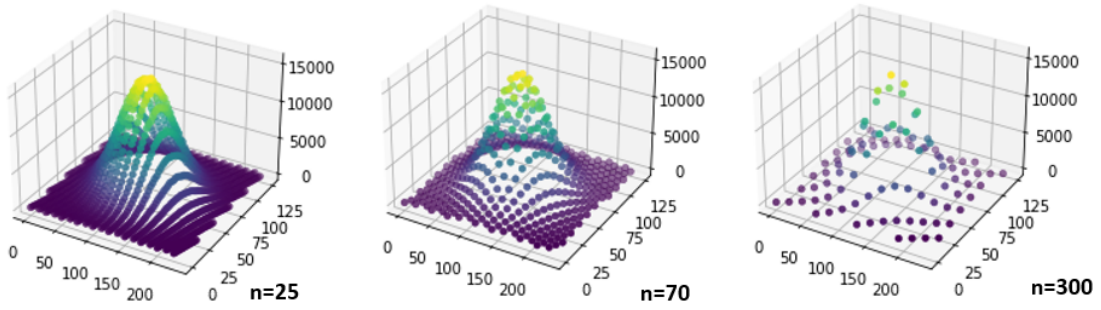


Abbildung 16: Sampling Datensätze mit verschiedenen Werten von n

Um zu überprüfen wie sehr die Analyse durch Sampling beschleunigt werden kann, wurden mit dem timeit Modul die Analysezeiten für verschiedene Werte von n bestimmt (siehe Abb. 17). Aus den Tests ging hervor, dass eine Verbesserung der Analysezeit erzielt werden kann, aber bei weiterer Verfeinerung nicht mehr zielführend ist. Hinzu kommt, dass Sampling, im Vergleich zum Labelansatz, stärkere Auswirkungen auf die Qualität der Fit-Analyse hat. Die Position im Bild, die Varianzen und der Rotationswinkel sind weniger betroffen. Vor allem Amplitude und Rauschen

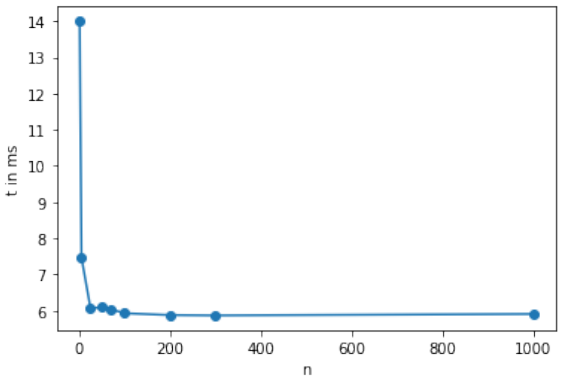


Abbildung 17: Analysezeit für verschiedene Werte von n

verändern sich leicht bei den verschiedenen Samplingstufen. Deshalb ist Sampling wohlüberlegt einzusetzen und sollte benutzerdefiniert erfolgen.
Die Tests zum Sampling sind nur beispielhaft. Sie zeigen, dass eine Optimierungspotenzial durch Sampling besteht. Sie zeigen aber auch, dass es eine Grenze gibt, an der noch weniger Daten nicht mehr die Analyse beschleunigen. Für weitere Optimierung könnte es von Interesse sein, andere Samplingansätze zu untersuchen, die noch zielgerichteter Daten aus der Menge der Daten wählen.

Zusammengefasst beschleunigt das Verkleinern der Datenmenge den Fit-Algorithmus. Die Datenmenge kann durch das Verkleinern des Bildbereichs durch den/die Nutzer:in, mit dem Labelansatz, aber auch durch weiteres Auswählen von Daten durch Sampling verringert werden.

3.3.2.2 Wertebereiche

Einen weiteren Einfluss auf die Qualität des Fit-Algorithmus hat das Festlegen von Wertebereichen (Boundaries) für die Fit- Parameter. Der Nutzen von Boundaries wird z.B. an dem Fit-Parameter Rotation des Elektronenstrahls deutlich. Der Fit-Algorithmus sucht einen möglichst passenden Winkel, mit dem er eine optimale Ähnlichkeit erzielt. Nun können aber mehrere Winkel eine optimale Lösung erzeugen. Ist das Bild um 360° gedreht, entsteht sogar genau das gleiche Optimum. Für die Kontrolle des Strahls besteht jedoch ein praktischer Unterschied, wenn der Strahl sich um 360° dreht oder gar nicht. Deshalb empfiehlt es sich, den Winkel in seinem Wertebereich einzuschränken, sodass sich der Elektronenstrahl nur noch um 45° in beide Richtungen drehen kann.
Tests ergaben, dass mit der Nutzung von Boundaries der Winkel viel deutlicher dem zu erwartenden Wert entspricht. Die Zeit für die Analyse verlängert sich jedoch auf mehr als das Doppelte. Diese Zeitzunahme spricht gegen das Verwenden von Boundaries bei der Analyse während des Betriebes.

3.3.2.3 Startwerte

Dem Fit-Algorithmus können nicht nur Boundaries übergeben werden, sondern auch Startwerte für die Parameter. Je näher die Startwerte an den optimalen Werten liegen, desto höher die Chance, dass optimale Werte schnell gefunden werden.
Bei ersten Testversuchen mit ungefähren Startwerten konnten im Unterschied zu den Wertebereichen keine Verzögerungen festgestellt werden.
Die nacheinander während des Betriebes erfassten Bilddaten sehen sich sehr ähnlich.

Es ist unwahrscheinlich, dass sich die resultierenden Fit-Parameter von zwei aufeinanderfolgenden Bilddaten von Grund auf unterscheiden. Deshalb lohnt es sich, die resultierenden Parameter des vorangegangenen Bildes zu speichern und der nächsten Fit-Analyse als Startwerte zu übergeben.
Ein besonderer Fall ist der erste Durchlauf. In Absprache mit Hr. Dr.-Ing. Rehm wurde vorausgesetzt, dass die erste Datenanalyse länger dauern kann, als alle darauf folgenden Analysen. Die resultierenden Fit-Parameter des ersten Durchlaufs können dadurch gute Startwerte für die nächste Analyse bilden. Deshalb wird für die ersten Startwerte eine begründete Vermutung durchgeführt und die erste Fit-Analyse durch Boundaries eingeschränkt.
Diese Optimierung hat den Vorteil, dass die entstandenen Startwerte für das zweite Bild nicht aus dem definierten Bereich hinaus gehen. Der Startwert für den Rotationswinkel kann für das zweite Bild nicht größer sein als vorher festgelegt. Bei der Analyse des zweiten Bildes und allen darauf folgenden Analysen werden die vorangegangenen resultierenden Parameter als Orientierungshilfe verwendet. Da die Bilder sich nur geringfügig unterscheiden, ist es unwahrscheinlich, dass sich der Rotationswinkel noch einmal stark z.B. um 360° verändert.
Eine Reihe von Tests mit vielen aufeinander folgenden Analysen konnte nachweisen, dass sich das in Abschnitt 3.3.2.2 beschriebene Rotationsproblem dadurch erübrigt.

Die erläuterten Startwerte sind ein Teil der Optimierung des Fit-Algorithmus. Weiterhin kann die Genauigkeit des Fit-Resultates durch Boundaries beim ersten Durchlauf verbessert werden. Die Laufzeit des Fit-Algorithmus verringert sich außerdem, wenn die zu analysierende Datenmenge vorher verkleinert wurde.

3.3.3 Entwurf DataAnalyzer

Um die Datenanalyse mit ihren verschiedenen Optimierungsschritten zu ordnen, wurde der Programmcode mit Hilfe von Klassen strukturiert. In Anlehnung an Abbildung 15 entstand der in Abbildung 18 abgebildete Entwurf.

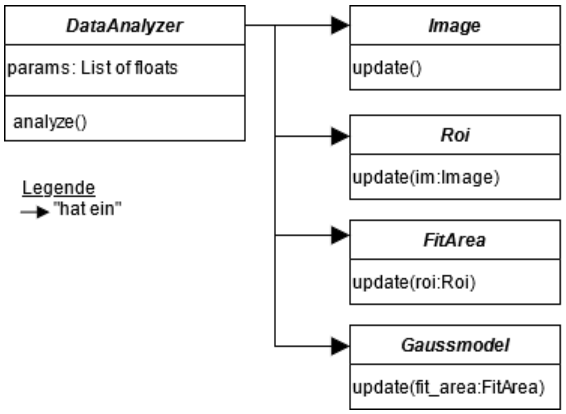


Abbildung 18: UML-Klassendiagramm mit ausgewählten Attributen und Methoden für die Datenanalyse

Bei einer Bildanalyse wird die Methode `analyze()` in der Klasse `DataAnalyzer` aufgerufen. Alle anderen aufgeführten Klassen führen die Methode `update()` aus und verarbeiten das Bild zu neuen Bildbereichen. Mit Hilfe der Klasse `Gaussmodel` wird dann der Fit-Algorithmus durchgeführt und die Fit-Parameter in der Liste `params` aktualisiert. Die Klasse `DataAnalyzer` konnte nach diesem Entwurf umgesetzt und getestet werden. Sie enthält die oben beschriebene optimimierte Fit-Analyse der Bilddaten und ist dabei nach Bildbereichen gegliedert.

3.4 EPICS-Schnittstelle

Die den Elektronenstrahl beschreibenden Parameter sollen nicht nur berechnet werden, sondern auch für Mitarbeiter:innen des HZBs im Kontrollraum von BESSY II sichtbar sein. Wie in Abschnitt 2.0.1 dargelegt, sollen sie dafür an ein mit EPICS erstelltes Prozessleitsystem weitergegeben werden. Um innerhalb des Python-Interpreters mit dem Prozessleitsystem arbeiten zu können, gibt es eine weitere Schnittstelle, die Python Bibliothek pythonSoftIOC. Mit dieser Bibliothek ist es möglich, ein EPICS IOC von einem Python-Interpreter aus zum Laufen zu bringen.
Darüber hinaus können repräsentative Datensätze (Prozess variables - PVs) erstellt werden. Die PVs basieren auf dem Channel-Acces-Kompontente von EPICS (siehe Abb. 2). Ein PV kann entweder auf einen Befehl des Prozessleitsystems reagieren oder einen Wert enthalten, der vom Prozessleitsystem ausgelesen werden kann. Ein solcher Wert ist zum Beispiel ein Parameter, der den Elektronenstrahl beschreibt. Der vom Python-Interpreter gestartete IOC kann eine PV erstellen und immer wieder aktualisieren. Über den Display Manager via Channel-Access können die Mitarbeiter:innen sich die Werte des PVs anzeigen lassen. [Diaa]
Genauso wie die Kameraschnittstellen in Abschnitt 3.2.1 wurde auch die EPICS-

Schnittstelle untersucht. Die dabei entdeckten Besonderheiten sollen im Folgenden genauer beleuchtet werden.

pythonSoftIOC ist entscheidend für die Wahl des Python-Interpreters. Die vorangegangenen Umsetzungsschritte können in Python 3.10 laufen. Die aktuellste Python Version, die pythonSoftIOC unterstützt, ist Python 3.9.. Die anderen Umsetzungsschritte wurden infolgedessen noch einmal mit dem 3.9 Kernel getestet. Als dadurch keine Funkionalitäten verloren gingen, wurde Python 3.9 als Version für das Projekt festgelegt.
Eine weitere Besonderheit von pythonSoftIOC ist, dass es das Prinzip der Gleichzeitigkeit verwendet. Dafür verwendet es die Python Bibliotheken cothread oder asyncio. cothread und asyncio beruhen auf der gleichen Art der Umsetzung von Gleichzeitigkeit. Die beiden Klassen unterscheiden sich u.a. durch verschiedene Benennung und dadurch, dass cothread nicht für Windows geeignet ist. Da das Projekt auf einem Windows-Rechner entwickelt wurde, fiel deshalb die Entscheidung zwischen den beiden Bibliotheken leicht. Die ausgewählte asyncio Bibliothek ist entscheidend für die Funktionsweise von pythonSoftIOC, weshalb im Folgenden nochmal auf asyncio und das Prinzip der Gleichzeitigkeit eingegangen werden soll.

3.4.1 Gleichzeitigkeit

Wenn es darum geht, Programme besonders schnell oder effizient laufen zu lassen, dann kommt häufig der Begriff der Gleichzeitigkeit (auf engl. cuncurrency) ins Spiel. Er bezieht sich auf das gleichzeitige Ausführen von Anweisungsfolgen. Statt immer sequenziell nur einer Anweisungsfolge nachzugehen, wird sich mit mehreren Folgen gleichzeitig beschäftigt.
Es gibt dabei verschiedene Arten für die Implementierung von Gleichzeitigkeit in Python. Multiprocessing ist die einzige tatsächlich parallele Umsetzung. Die einzelnen Anweisungsfolgen laufen auf verschiedenen Prozessoren mit eigenem Python-Interpreter. [Wei19, S.933 ff.]
Beim Multitasking hingegen laufen die Anweisungsfolgen nur scheinbar parallel auf einem einzigen Prozessor. Es werden Teile der Anweisungsfolgen hintereinander ausgeführt, sodass mehrere Anweisungsfolgen scheinbar gleichzeitig laufen.
Man unterscheidet hierbei in präemtives und kooperatives Multitasking. Beim präemtiven Multitasking entscheidet das jeweilige Betriebssystem, wann eine Anweisungsfolge unterbrochen und ein weitere ausgeführt wird. Die Python Bibliotheken asyncio und cothread gehören zum kooperativen Mutlitasking. Hier entscheiden die

Art der Gleichzeitigkeit	Entscheidung zum Wechsel	Anzahl der Prozessoren
präemtives Multitasking	Das Betriebssystem entscheidet von außen, wann Tasks gestoppt und weitergeführt werden.	1
kooperatives Multitasking	Die Tasks entscheiden, wann sie die Kontrolle abgeben.	1
Multiprocessing	Die Prozesse laufen alle parallel auf verschiedenen Prozessoren.	viele

Tabelle 2: Arten der Umsetzung von Gleichzeitigkeit im Vergleich

einzelnen Anweisungsfolgen (Tasks) selbst, wann sie soweit sind, dass eine weitere Anweisungsfolge ausgeführt werden kann. (siehe Tabelle 2) [And]

Multitasking, vor allem kooperatives Multitasking, ist besonders effizient, wenn es darum geht, an Input/Output gebundene Anweisungsfolgen gleichzeitig laufen zu lassen. So kann z.B. die Bilderfassungsfunktion anderen Funktionen den Vorrang lassen, während sie darauf wartet, dass die Kamera ein Bild geladen hat. Es ist also nachzuvollziehen, warum sich die Entwickler:innen von softioc für diese Art der Umsetzung von Gleichzeitigkeit entschieden haben.

Ein wichtiger Bestandteil für die Funktion von asyncio ist die sogenannte Ereignisschleife. Sie ist für die Steuerung der Task zuständig. Die Ereignisschleife kennt jeden Task und weiß, in welchem Zustand er sich gerade befindet. Es gibt eine Reihe von Zuständen, in denen sich die Anweisungsfolgen befinden können. Für ein besseres Verständnis werden diese im Folgenden zu den zwei Zuständen *bereit* und *wartend* vereinfacht. Der Zustand *bereit* zeigt an, dass eine Aufgabe bereit ist, erledigt zu werden, während der Zustand *wartend* bedeutet, dass die Aufgabe auf die Beendigung eines externen Vorgangs wartet, z. B. auf das Laden eines Bildes von der Kamera.
Die Ereignisschleife wählt einen der *bereiten* Tasks aus und lässt ihn sich ausführen. Dieser Task hat die vollständige Kontrolle, bis er die Kontrolle an die Ereignisschleife zurückgibt. Wird die Kontrolle an die Ereignisschleife zurückgegeben, evaluiert diese erneut, welche Tasks *bereit* sind, und bestimmt den nächsten auszuführenden Task. Dabei wird u.a. mit berücksichtigt, wie lange die Tasks schon auf ihre Ausführung warten. Dieser Ablauf wiederholt sich so lange, bis alle Tasks zu Ende ausgeführt wurden.
Entscheidend ist dabei, dass die Tasks niemals die Kontrolle unabsichtlich abgeben. Sie werden nie mitten in einer Operation unterbrochen. Dadurch können Ressourcen einfacher zwischen Tasks genutzt werden, und es kommt nicht zu Zugriffskollidierungen, wie das bei präemtiven Multitasking der Fall sein kann.

Auch in pythonSoftIOC wird eine Ereignisschleife erzeugt. Ihr können weitere nutzerdefinierte Tasks hinzugefügt werden. Damit erkannt werden kann, dass es sich um einen Task handelt, muss die entsprechende Funktion mit `async` markiert werden. Innerhalb der Funktion wird mit `await` markiert, an welcher Stelle die Funktion die Kontrolle zurück gibt. [And] [Diab]
Wenn also ein Task implementiert werden soll, der immer wieder einen Parameter setzen und dann die Kontrolle wieder zurück geben soll, könnte das wie folgt aussehen:

```
1 async def update():
2     while True:
3         parameter.set(parameter.get() + 1)
4         await asyncio.sleep(1)
```

Listing 3.1: Bespielprogrammcode für die Umsetzung von asyncio

Die Funktionsweise von pythonSoftIOC wurde auch für das Projekt getestet. Dazu wurden zunächst vom Python 3.9-Interpreter aus eine Reihe von PVs erzeugt und getestet. Zum Testen wurde die Software „control system studio Phoebus“ verwendet, in der benutzerdefinierte Standardeinstellungen für die Verwendung u.a. in den HZB-Einrichtungen MLS und BESSY festgelegt sind. Darauf folgend wurde das gleichzeitige Laufen von EPICS über pythonSoftIOC im Zusammenhang mit der Kamera getestet. Es wurde sichergestellt, dass sowohl die Kamera- als auch die EPICS-Schnittstelle gleichzeitig ausgelesen werden kann. Damit stand der Verwendung von pythonSoftIOC nichts mehr im Wege.

3.5 Zusammenführen der Umsetzungsschritte

Die drei Hauptumsetzungsschritte - Anbinden der Kamera, Datenanalyse und EPICS-Schnittstelle - konnten erfolgreich getestet werden. Im nächsten Schritt wurden diese unabhängigen Programmteile zusammengefügt. Vor dem eigentlichen Zusammenfügen des Programmcodes wurde zuerst eine Architektur für den Code entworfen. Diese sollte, wie in den Anforderungen (siehe Absatz 2.0.1) spezifiziert, besonders anpassungsfähig und erweiterbar sein.
Die Festlegung auf die Programmiersprache Python bestimmt das verwendete Programmierparadigma – objektorientierte Programmierung. Ein bekanntes Ablaufmodell für eine objektorientierte Software-Entwicklung, an dem sich für das Projekt orientiert wurde, einhält die folgenden Phasen: Analyse, Entwurf, Programmierung. [Wei19, S.323 ff.] Nachdem mit der Vorbereitung und den ersten Umset-

zungsschritten eine ausführliche Analysephase durchlaufen wurde, folgte in der Entwurfsphase ein objektorientierter Entwurf, der in mit einem UML-Klassendiagramm festgehalten wurde. Daraufhin wurde der Entwurf implementiert, wobei bei der Detailimplementierung Teile des Codes von den vorherigen Umsetzungsschritten übernommen werden konnten.

3.5.1 Entwurf Klassendiagramm

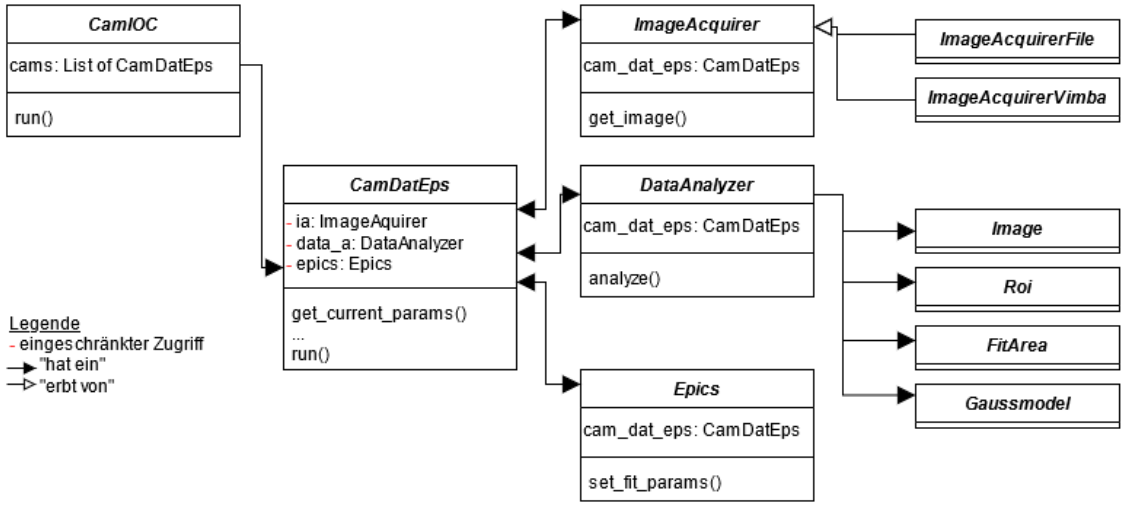


Abbildung 19: UML-Klassendiagramm mit ausgewählten Attributen und Methoden

Aus den in Abb. 3 dargestellten Anforderungen wurden eine Reihe von Klassen für die Implementierung abgeleitet (siehe Abb. 19). `CamIOC` steht für „Camera Input/Output Controler“, angelehnt an den IOC von EPICS, und kann mehrere Kameras und deren Bildverarbeitung verwalten. Kernstück ist dafür eine Liste von `CamDatEps`-Instanzen. Jede der Instanzen ist während des Betriebs für eine Kontrollkamera und deren Auswertung zuständig. Die Klasse `CamDatEps` besitzt eine Kameraschnittstelle `ImageAcquirer`, eine Klasse zur Datenauswertung `DataAnalyzer` und eine EPICS-Schnittstelle `Epics`.

Auch wenn unabhängig getestet und in verschiedenen Skripten implementiert, interagieren `Epics`, `DataAnalyzer` und `ImageAcquirer` im finalen Einsatz miteinander. Um zu verhindern, dass Zugriffe zwischen den Klassen unkontrolliert bzw. schwer nachzuvollziehen sind, laufen alle Interaktionen über die Klasse `CamDatsEps`. `CamDatEps` fungiert als Vermittler zwischen den drei anderen Klassen und die drei Klassen bleiben unabhängig. Das ist besonders von Vorteil, wenn es darum geht, spätere Änderungen hinzu zu fügen. Statt in den verschiedenen Skripten nachschauen zu müssen, ob eine Änderung noch an anderen Stellen eine Auswirkung hat, sind alle vermittelnden Funktionen, wie `get_current_params()`, in `CamDatEps` übersichtlich

an derselben Stelle zu finden.
Diese Lösung für den Austausch birgt jedoch die Gefahr, dass der Zugriff auf die Klasse `CamDatEps` von deren Attributen inkorrekt ausgenutzt wird. Alle Bemühungen für Unabhängigkeit wären vergebens, wenn z.B. im `Epics`-Programmcode über `CamDatEps` ohne Vermittlungsfunktion direkt auf `DataAnalyzer`-Werte zugriffen werden würde. Um das zu verhindern, wurde dem Geheimnisprinzip von objektorientierter Programmierung gefolgt. Bei dem Geheimnisprinzip (auf engl. information hiding) geht es um Zugriffsbeschränkungen und die Sichtbarkeit von Attributen. [Wei19, S.296] Ziel ist es, schon durch im Code festgehaltene Regeln spätere unerwünschte Implementationen zu vermeiden. Das ist besonders von Bedeutung, wenn das Projekt von verschiedenen Entwickler:innen weitergeführt wird.
In anderen Programmiersprachen wie z.B. Java oder C++ können Attribute und Funktionen in einer Klasse mit Zugriffsberechtigungen versehen werden. In Python wird das Geheimnisprinzip nicht unmittelbar unterstützt. Es gibt jedoch als Ersatz die Konvention, interne Variablen- und Methodennamen mit ein bis zwei Unterstrichen anzufangen und damit begrenzt zu schützen. [Wei19, S.285 f.]
In dem vorliegendem Projekt wurde mit dieser Konvention der Zugriff von außerhalb auf die Attribute `ia`, `data_a` und `epics` der Klasse `CamDatEps` eingeschränkt (in Abb. 19 rot markiert). Dadurch ist es nicht mehr ohne Weiteres möglich, von `Epics` direkt auf die Instanz der `DataAnalyzer`-Klasse `data_a` zuzugreifen. Die Unabhängigkeit der beiden Klassen wird durch im Programmcode festgelegte Regeln gewahrt.

Ein weiteres Konzept der objektorientierten Programmierung, das für die Entwicklung des Klassendiagramms von Bedeutung war, ist Polymorphie. Wie in Abschnitt 3.2.1 deutlich wird, gibt es für die Anbindung der Kamera verschiedene Schnittstellen, wie die Vimba Python API und Harvester, und damit verschiedene Möglichkeiten für die Bilderfassung. Weiterhin ist es von Vorteil, wenn die Bilderfassung durch das Laden bereits aufgenommener Aufnahmen für Testzwecke simuliert werden kann.
Ein Lösungsansatz hierfür wurde mit dem Konzept der Polymorphie entworfen. [Joh] [Wei19, S.290 ff.] Um eine höhere Flexibilität zu erreichen, wird eine polymorphe Klasse erzeugt, hier die Klasse `ImageAcquirer`. Die funktional wichtigen Methoden werden erst in den von `ImageAcquirer` erbenden Klassen implementiert. So ist die Funktion `get_image()` in der Klasse `ImageAcquirer` leer und wird in `ImageAcquirerVimba` mit der Vimba Python API und in `ImageAcquirerFile` durch das Auslesen von Bilddateien umgesetzt. Die Klasse `ImageAcquirer` kann damit flexibel verschiedene Formen ihrer erbenden Klassen annehmen.
Sollte eine weitere Klasse mit der Harvester-Schnittstelle implementiert werden, müsste diese nur von der Klasse `ImageAcquirer` erben und deren funktional wich-

tigen Methoden implementieren. Dann könnte sie genauso flexibel eingesetzt werden.

Mit diesem Entwurf wurde den Anforderungen nach Erweiterbarkeit und der Möglichkeit zur Anpassung entsprochen. Der Entwurf hat sich bei der Implementierung weiterer Umsetzungsschritte (siehe Abschnitt 3.6) bewährt, da das bestehende Konzept erhalten bleibt und nur noch um weitere Komponenten erweitert werden konnte.

3.5.2 Programmstruktur

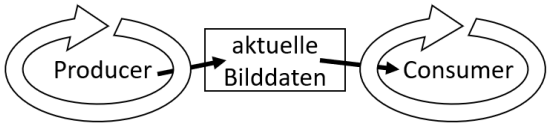


Abbildung 20: Consumer-Producer-Entwurfsschema für die Programmstruktur

In Abschnitt 3.4.1 deutete sich bereits an, dass nicht nur die Klassenstruktur sondern auch eine Programmstruktur für den Ablauf im Programm wichtig ist. Die Kamera sollte unabhängig davon, wie schnell ihre Aufnahmen verarbeitet werden, möglichst aktuelle Bilder liefern. Datenverarbeitung und und Bilderfassung werden deshalb gleichzeitig ausgeführt.
Für den Informationsaustausch zwischen den beiden Komponenten bietet das Consumer-Producer-Entwurfsmuster eine gute Hilfestellung. Bei diesem Entwurfsmuster gibt es zwei programmatische Akteure, den Consumer und den Producer. Der Producer produziert Daten und der Consumer verarbeitet die Daten. Beide arbeiten gleichzeitig in voneinander unabhängigen Schleifen, zwischen denen die Daten über ein Übertragungsobjekt transferiert werden. [Tin18]
Dieses Entwurfsmuster hat den Vorteil, dass der Producer durchgängig in seiner Geschwindigkeit Daten produzieren kann und nicht auf deren Verarbeitung warten muss. In dem vorliegenden Projekt ist die Klasse `ImageAcquirer` für das Produzieren der Daten zuständig und `DataAnalyzer` und `Epics` für das Verarbeiten der Daten. Sie erhalten deshalb beide ihre eigenen Schleifen, die mit Hilfe von asyncio (siehe Abschnitt 3.4.1) gleichzeitig durchlaufen werden. Die Verbindung zur Kamera kann dadurch durchgängig erhalten bleiben und muss nicht für jedes neue Bild erneut aufgebaut werden (siehe Abb. 20).

Wird die Funktion `run()` in `CamIOC` ausgeführt, wird für jede Kamera eine Consumer- und eine Producerschleife erzeugt, die mit Hilfe von `asyncio` alle gleichzeitig arbeiten. Um sowohl die Programmstruktur mit diesen Schleifen und die Klassenstruktur zu

testen, wurden zuerst die Klassen mit ihren wichtigsten Eigenschaften vereinfacht implementiert. Nachdem erste Tests erfolgreich waren, wurden die Details der Klassen weiter implementiert und erneut getestet (siehe Abschnitt 3.6.4).

3.6 Weiterführende Umsetzungsschritte

3.6.1 Fehlermeldung

In der ersten Version des Codes wurden Fehlermeldungen und Hinweise durch print-Anweisungen in den Terminal des Python-Interpreters umgesetzt. Da während des Betriebes keine Konsole angezeigt wird, empfiehlt es sich, die Meldungen an das Kontrollleitsystem weiterzuleiten. Dadurch werden sie auch während des Betriebs für Mitarbeiter:innen im Kontrollraum von BESSY II sichtbar. Beide Schleifen, sowohl die Consumer- als auch die Producerschleife, können Fehler melden. Die verschiedenen Fehler sollen über zwei verschiedene PVs an das Prozessleitsystem gesendet werden. Diese weiterführende Anforderung konnte erfolgreich implementiert werden.

3.6.2 Kontrollparameter

Während der Implementierung für die Datenanalyse (siehe Abschnitt 3.3) wurde eine Reihe von Benutzereinstellungen definiert. Für eine bessere Handhabung wäre es von Vorteil, wenn diese Einstellungen nicht im Programmcode festgehalten werden, sondern vom Kontrollraum aus geändert werden könnten. So könnte z.B. die ROI während des Betriebes als Reaktion auf eine Fehlermeldung, dass sich der Elektronenstrahl an den Rand der ROI bewegt, von Mitarbeiter:innen verbessert werden.
In pythonSoftIOC gibt es die Möglichkeit, über PVs Änderungsfunktionen mit Parametern, die neuen Informationen enthalten, aufzurufen. Diese Funktionen wurden über weitere Vermittlungsfunktionen in der Klasse `CamDatEps` so erweitert, dass die Änderungsinformationen an die Klasse `DataAnalyzer` weitergegeben werden. Die Nutzung von Kontrollparametern konnte so implementiert werden, ohne dass die Klassen `Epics` und `DataAnalyzer` ihre Unabhängigkeit verlieren.

3.6.3 Initialisierungsdateien

Bei wissenschaftlichen Experimenten ist es wichtig, gleiche Experimente unter gleichen Bedingungen noch einmal neu durchführen zu können. Dafür müssen die Bedingungen dokumentiert und wiederherstellbar sein. Weiterhin soll der Programmcode möglichst für verschiedene Aufbauten flexibel einsetzbar sein. Wenn also Einstellungen für die Datenanalyse oder die Kamera fest im Programmcode verankert sind, würde das nicht einem Anspruch an ein wissenschaftliches Experiment entsprechen. In der bereits existierenden Lösung mit LabView gab es die Möglichkeit, Einstellungen und Initialisierungswerte über Initialisierungsdateien einzulesen. Die Dateien beruhten dabei auf einem LabView-Standard, der so nicht mehr für das Projekt in Frage kommt. Mit dem LabView-Standard gab es zusätzlich die Möglichkeit, nicht alle Einstellungen gleichzeitig zu implementieren, sondern nur Teile davon wieder zu verwenden.

Als Dateiformat für eine neue Lösung wurde JSON ausgewählt. Dieses Dateiformat basiert auf der Programmiersprache JavaScript, ist aber programmiersprachenunabhängig. [Wei19, S.754 ff.] Mit der Python Bibliothek `json` lassen sich aus JSON-Dateien Python-Datenstrukturen mit ihren Werten erzeugen und abspeichern. Der Inhalt einer solchen JSON-Datei ist in Abbildung 21 zu erkennen.

```
{
    "camera": {
        "features": {
            "ExposureAuto": "Off"
        },
        "cam_id": "DEV_000F31024A32"
    },
    "data_a": {
        "control_params_values": {
            "roi_x_start": 800,
            "roi_x_stop": 1250,
            "roi_y_start": 600,
            "roi_y_stop": 900,
            "factor": 0.5,
            "threshold": 708,
            "median_flt": true,
            "sampled": 0
        }
    },
    "epics": {
        "device_name": "CAMERA",
        "control_params": {
            "roi_x_start": "AO_ROI_X_START",
            "roi_x_stop": "AO_ROI_X_STOP",
            "roi_y_start": "AO_ROI_Y_START",
            "roi_y_stop": "AO_ROI_Y_STOP",
            "factor": "AO_FACTOR",
            "threshold": "AO_THRESHOLD",
            "median_flt": "AO_MEDIAN_FLT",
            "sampled": "AO_SAMPLED"
        }
    }
}
```

Abbildung 21: Beispielhafte JSON-Initialisierungsdatei

Wird eine neue `CamDatEps`-Instanz hinzugefügt, kann dieser eine aus der Initialisierungsdatei erzeugte Datenstruktur übergeben werden. So bekommt in dem von Abb. 21 ausgehenden Beispiel die neue `DataAnalyzer`-Instanz alle Informationen übergeben, die unter `data_a` zu finden sind.
Innerhalb des Initialisierungsprozesses delegieren die Klassen die Umsetzung immer weiter in ihre Unterklassen. In dem Beispiel würde die `DataAnalyzer`-Instanz Informationen zur ROI an die neue `Roi`-Instanz weitergeben.

Für die eigenen Initialisierungsparameter wird überprüft, ob in den übergebenen Initialisierungsinformationen ein entsprechender Wert vorhanden ist oder ob ein Defaultwert verwendet werden muss. Die Defaultwerte sind fest im Programmcode oder berechnen sich selbst aus den Bilddaten. Dadurch entsteht die gleiche Flexibilität, die auch die LabView-Lösung geboten hat. Ist irgendeine Eigenschaft oder Einstellung nicht in der Initialisierungsdatei definiert, wird auf den Defaultwert zurückgegriffen. Einstellungen können über Initialisierungsdateien festgelegt werden, der Programmcode funktioniert aber auch ohne.
Bei dem Initialisieren der Kontrollparameter muss beachtet werden, dass die Parameter über Vermittlungsfunktionen in der Klasse `CamDatEps` zur Klasse `Epics` weitergeleitet werden, damit auch die anfänglichen Kontrollparameter mit dem Prozessleitsystem angezeigt werden können.

Mit der Implementierung zum Laden aus Initialisierungsdateien bietet es sich an, ein Speichern des aktuellen Zustandes hinzuzufügen. Dafür wurde eine weitere PV in der Klasse `Epics` hinzugefügt, bei deren Änderung eine Speicherung aller aktuellen Einstellungen in einer JSON-Datei ausgelöst wird. Einstellungen werden dabei unabhängig davon gespeichert, ob es sich aktuell um Default- oder Benutzereinstellungen handelt.
Die Kameraeinstellungen seien an dieser Stelle noch einmal seperat erwähnt. Eine Kamera kann alle ihre aktuellen Eigenschaften liefern, diese Liste ist jedoch unübersichtlich lang. Deshalb wurde sich dafür entschieden, beim Speichern nur die über die anfängliche Initialisierungsdatei übergebenen Kameraeinstellungen zu speichern. Die Lesbarkeit der neuen Initialisierungsdatei bleibt dadurch erhalten.

3.6.4 Tests

Um die Einsatzfähigkeit der implementierten Gesamtlösung zu überprüfen, wurde sie weiteren Tests unterzogen. So wurde u.a. an der MLS eine Kamera angebunden und deren Daten ausgewertet. Mit Phoebus, der in Abschnitt 3.4.1 erwähnten Software zum Auslesen von PVs, konnte bestätigt werden, dass die Fit-Parameter ausgerechnet und an EPICS weitergeleitet werden. Zudem können während des Betriebes Einstellungen für die Analyse verändert werden. Beim Verbindungsabbruch der Kamera wird ein Fehler angezeigt. Nach dem Wiederherstellen der Verbindung beginnt das Erfassen der Daten erneut, ohne dass alles neu gestartet werden muss.

In einem weiteren Test wurde überprüft, ob mehrere Kameras angeschlossen werden können. Da mehrere Kameras nicht zur Verfügung standen, wurden die Kameras durch mehrere `ImageAcquirerFile`-Instanzen simuliert. Ihre Analyseergebnisse konnten mit verschiedenen PV Namen ausgelesen werden.

Weiterhin wurde der Arbeitsspeicher während der Programmausführung überprüft. Großexperimente wie BESSY II können tagelang in Betrieb sein. Ein langsam ansteigender Arbeitsspeicherverbrauch könnte bei diesen langen Zeiten zu Problemen führen. Mit dem Python Modul `psutil` wurde beispielhaft für eine Stunde der Prozess des laufenden Programmes überwacht. Dabei ergab sich tatsächlich eine langsame Erhöhung des genutzten Arbeitsspeichers. Nach Ursachen für dieses Problem sollte in der weiteren Entwicklung gesucht werden, sodass ein begrenzter Arbeitsspeicherverbrauch gewährleistet werden kann.

4 Auswertung

Die vorliegende Arbeit hat sich mit der Frage beschäftigt, ob es eine alternative Möglichkeit gibt, die Kontrolle des Elektronenstrahls von BESSY II über Kameras in EPICS einzubinden. Um die Frage zu beantworten, wurde der Versuch unternommen, eine solche alternative Möglichkeit in Python umzusetzen und zu testen.

Bei der Umsetzung erfolgte zunächst eine Einarbeitung in die drei verschiedenen Thematiken der Kameraanbindung, Datenanalyse und EPICS. Daraufhin wurden erste Teilschritte implementiert und dokumentiert. Die Kamera konnte erfolgreich angebunden werden, wobei auf die Flexibilität verschiedener Hersteller verzichtet wurde. Mit Hilfe der Vimba Python API war es möglich, verlässlich Bilddaten auszulesen.
Beim Analysieren der Bilddaten wurde ein Fit-Algorithmus verwendet, der Informationen über den zu kontrollierenden Elektronenstrahl erfasst. Der Fit-Algorithmus wurde optimiert. Ein Ansatz war dabei das Verkleinern der Datenmenge durch den/-die Nutzer:in oder algorithmisch durch Sampling und Finden des Elektronenstrahls in einem Bereich. Weitere Optimierungen konnten durch das Verwenden von Startwerten und Boundaries durchgeführt werden.
Die Einbindung in das Prozessleitsystem konnte mit Hilfe von pythonSoftIOC umgesetzt werden. Dabei wurde sich mit der Thematik der Gleichzeitigkeit bei der Verwendung von asyncio auseinander gesetzt.
Beim Zusammenführen der einzelnen Teilschritte konnte eine Klassenstruktur entworfen werden, die offen für spätere Erweiterungen ist und Möglichkeiten zur Anpassung bietet. Der Programmablauf orientiert sich an dem Consumer-Producer-Entwurfsmuster.
Die Programmstruktur ermöglicht eine unkomplizierte Implementierung weiterer Umsetzungsschritte, wie das Weitergeben von Fehlermeldungen über EPICS und Initialisierungsdateien.

4.1 Evaluation

Erste Tests u.a. an der MLS konnten bestätigen, dass die implementierte Lösung alle Anforderungen erfüllt. Eine Kamera konnte angebunden werden. Ihre Daten konnten analysiert und an das Prozessleitsystem weiter gegeben werden, ganz ohne die Verwendung von LabView. Damit kann die in der vorliegenden Arbeit beschriebene Lösung als alternative Möglichkeit gesehen werden, um die Kontrolle des Elektronenstrahls von BESSY II über Kameras in EPICS einzubinden.

4.2 Ausblick

Bevor die erarbeitete Software im Betrieb genutzt werden kann, sollte jedoch das Problematik des ansteigenden Arbeitsspeichers gelöst werden. Auch wäre es sinnvoll, sie weiteren Tests zur Betriebsfähigkeit und Integration zu unterziehen. Schließlich ist die Elektronenstrahlkontrolle essenziell für den Betrieb von BESSY II.
Auch wäre es interessant, die Laufzeit und Qualität der alternativen Lösung mit der LabView-Lösung zu vergleichen.
Weiterhin könnte die Optimierung der Datenanalyse noch vertieft werden. Weitere Ansätze zur Reduzierung der Datenmenge könnten untersucht werden oder bereits existierende Ansätze mit Performanzbetrachtung weiter verbessert werden. Auch wäre es denkbar, dass durch das Verlagern der Datenanalyse auf andere Hardware, wie Grafikkarten oder FPGAs, noch Verbesserungen erzeugt werden können.
Für eine vom Kamerahersteller unabhängige Lösung könnte zudem eine Harvester-Schnittstelle implementiert und eingebunden werden.

Literaturverzeichnis

[All17] Allied Vision Technologies GmbH (2017). *GenICam—yet another standard?* Allied Vision Technologies GmbH, Stadtroda, Germany.

[All20] Allied Vision Technologies GmbH (2020). *Vimba Python Manual 1.0.0.* Allied Vision Technologies GmbH, Stadtroda, Germany.

[All21] Allied Vision Technologies GmbH (2021). *GIGE VISION CAMERAS Prosilica GT Technical Manual.* Allied Vision Technologies GmbH, Stadtroda, Germany.

[And] Anderson, J. pythonsoftioc. https://realpython.com/python-concurrency/. Abgerufen: 21.07.2022.

[DHK+94] Dalesio, L. R., Hill, J. O., Kraimer, M., Lewis, S., Murray, D., Hunt, S., Watson, W., Clausen, M., and Dalesio, J. (1994). The experimental physics and industrial control system architecture: past, present, and future. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 352(1-2):179–184.

[DKK91] Dalesio, L. R., Kozubal, A., and Kraimer, M. (1991). Epics architecture. Technical report, Los Alamos National Lab., NM (United States).

[Diaa] Diamond Light Source. pythonsoftioc. https://dls-controls.github.io/pythonSoftIOC/master/index.html. Abgerufen: 21.07.2022.

[Diab] Diamond Light Source. What are the differences between asyncio and cothread? https://dls-controls.github.io/pythonSoftIOC/master/explanations/asyncio-cothread-differences.html. Abgerufen: 21.07.2022.

[Die] Dierks, F. Genicam standard. page 57. https://www.emva.org/standards-technology/genicam/genicam-downloads/.

[EMV] EMVA – European Machine Vision Association. Genicam. https://www.emva.org/standards-technology/genicam/. Abgerufen: 05.07.2022.

[Erd] Erdem, K. Understanding region of interest (roi pooling). https://erdem.pl/2020/02/understanding-region-of-interest-ro-i-pooling. Abgerufen: 27.07.2022.

[Gre21] Greengard, S. (2021). *The internet of things*. MIT press.

[Hel] Helmholtz-Zentrum Berlin für Materialien und Energie. Röntgenquelle bessy ii. https://www.helmholtz-berlin.de/forschung/quellen/bessy/index_de.html. Abgerufen: 2022-06-01.

[Joh] Johnson, J. Polymorphism in programming. https://www.bmc.com/blogs/polymorphism-programming/l. Abgerufen: 20.07.2022.

[KGLT94] Knott, M., Gurd, D., Lewis, S., and Thuot, M. (1994). Epics: A control system software co-development success story. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 352(1-2):486–491.

[Lan20] Landschaft der Forschungsinfrastrukturen (2020). Bessy ii – universelle photonenquelle. https://fis-landschaft.de/materie/bessy-ii/. Abgerufen: 2022-06-01.

[MN] Matthew Newville, Till Stensitzki, R. O. u. a. Non-linear least-squares minimization and curve-fitting for python. https://lmfit.github.io/lmfit-py/index.html. Abgerufen: 26.07.2022.

[Num] NumPy Developers. Numpy: the absolute basics for beginners. https://numpy.org/doc/stable/user/absolute_beginners.html. Abgerufen: 25.07.2022.

[Oxf] Oxford Instruments. Ccd, emccd or scmos: Choosing the right scientific camera for your research. https://andor.oxinst.com/learning/view/article/scientific-digital-cameras. Abgerufen: 2022-07-04.

[Pat19] Patel, D. T. (2019). *Distributed Computing for Internet of Things (IoT)*. IGI Global. Kapitel 4.

[PVG+11] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Phy] Physikalisch-Technische Bundesanstalt. Metrology light source. https://www.ptb.de/cms/ptb/fachabteilungen/abt7/ptb-sr/mls.html. Abgerufen:

25.07.2022.

[PM22] Pina Merkert, W. D. u. J. M. (2022). Codefabriken - die python-entwicklungsumgebung für sie. *c't magazin für computer technik*, 5:26–30.

[Pre] Presse- und Informationsamt der Bundesregierung. Energiewende im Überblick. https://www.bundesregierung.de/breg-de/themen/klimaschutz/energiewende-im-ueberblick-229564. Abgerufen: 2022-06-01.

[Qua] Qualtrics. What is cluster analysis and when should you use it? https://www.qualtrics.com/uk/experience-management/research/cluster-analysis/. Abgerufen: 27.07.2022.

[RFW] R. Fisher, S. Perkins, A. W. and Wolfart, E. Median filter. https://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm. Abgerufen: 26.07.2022.

[sci] scikit-learn developers. Clustering. https://scikit-learn.org/stable/modules/clustering.html. Abgerufen: 27.07.2022.

[The] The GenICam Committee. harvesters-util 0.1.0 project description. https://pypi.org/project/harvesters-util/#about-harvester. Abgerufen: 01.07.2022.

[Tin18] Tinca, I. (2018). The evolution of the producer-consumer problem in java. https://dzone.com/articles/the-evolution-of-producer-consumer-problem-in-java. Abgerufen: 01.08.2022.

[Wei19] Weigend, M. (2019). *Python 3. Lernen und professionell anwenden. Das umfassende Praxisbuch.* mitp Verlag. 8. Auflage.

Ehrenwörtliche Erklärung

Ich erkläre ehrenwörtlich:

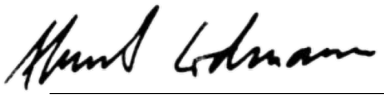
1. dass ich meine Bachelor-Thesis selbstständig verfasst habe,

2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe,

3. dass ich meine Bachelor-Thesis bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Berlin, 05.08.2022 — Ort, Datum

 — Almut Erdmann