STUDIENPROJEKT

Simulation des Extraktionsgitters eines Radiofrequenz-Ionentriebwerks mithilfe von IBSimu

Peter Preisler (8102359)

begleitet von

Jana Zorn

2. Juni 2024

Zusammenfassung

Elektrostatische Antriebssysteme stellen eine sehr treibstoffeffiziente Methode für Geschwindigkeitsänderungen im Weltraum dar. Denn dank ihrer hohen Ausstoßgeschwindigkeiten wird jedes Masseninkrement möglichst effizient zur Schuberzeugung genutzt. Besonders hohe Geschwindigkeiten werden bei Radiofrequenz-Ionentriebwerken (RIT) erreicht. In diesen laufen zwei nacheinander folgende Schritte ab, nämlich die Ionisation des Treibstoffs und die Beschleunigung der daraus gewonnenen Ionen durch ein Gittersystem. Um den Schub zu maximieren, sollen die Gitter einen Strahl möglichst geringer Divergenz erzeugen, da die transversalen Impulsanteile dem Schub nicht dienlich sind. Die Betrachtung der Ionen-Trajektorien in der Gitteranordnung wird auch Ionenoptik genannt, welche mithilfe von Simulationsprogrammen vor der Produktion der Gitter ausgelegt werden kann. In dieser Arbeit wird dazu der open-source Code IBSimu genutzt, welcher eine C++ Bibliothek darstellt, die alle Werkzeuge für die Simulation der Plasmaextraktion beinhaltet. Da die Gitter für das RAM-EP Projekt bereits gefertigt wurden, wird im Rahmen dieser Arbeit der Gitterabstand für verschiedene Arbeitspunkte im Bereich zwischen 2 und 4 Millimeter variiert. Eine detaillierte Analyse der Simulationsdaten ergibt, optimiert auf die Winkeldivergenz des Ionenstrahls, einen idealen Gitterabstand zwischen 3.1 und 3.3 Millimetern.

Inhaltsverzeichnis

1 Einführung

Dieses Studienprojekt beschäftigt sich mit der Simulation der Ionenoptik des Extraktionsgitters eines Radiofrequenz-Ionentriebwerks (RIT). Dafür ist die Ausführung in fünf Kapitel unterteilt, welche Schritt für Schritt an das Thema heran führen sollen. In diesem Kapitel soll die Einordnung und der Aufbau von RITs erläutert und die Simulation motiviert werden. Anschließend erfolgt eine Einführung in die Plasmaphysik, welche notwendig zum Verständnis der Eingabeparameter ist. Die C++ Bibliothek, welche der Simulation zugrunde liegt, soll anschließend genauer beschrieben werden, sodass im anschließenden Kapitel dann die Parameterstudie definiert werden kann. Abschließend wird dann das Ergebnis dieser Untersuchungen dargestellt.

1.1 Radiofrequenz-Ionentriebwerke

RITs gehören zur Klasse der elektrostatischen Triebwerke, welche ihren Schub durch die Coulomb-Kraft erzeugen. Damit stehen sie im Gegensatz zu elektrothermischen und elektromagnetischen Triebwerken, welche ihren Schub durch elektrisches Aufheizen des Treibstoffs bzw. mithilfe der Lorentzkraft erzeugen. Gemeinsam haben alle drei Arten von elektrischen Triebwerken, dass sie elektrische in kinetische Energie umwandeln, um so Schub zu erzeugen. Im Gegensatz dazu wird in chemischen Triebwerken die Reaktionsenergie des Treibstoffs mit einem Katalysator oder Oxidator ausgenutzt, um den Treibstoff zu erhitzen. Diese thermische Energie wird dann durch Expansion in kinetische Energie umgewandelt. Chemische Triebwerke müssen also immer den Umweg über die thermische Energie nehmen und sind deshalb durch die Größe der Reaktionsenergie und die Temperaturbeständigkeit der verwendeten Materialien begrenzt. Elektrostatische und -magnetische Triebwerke sind hingegen nur durch die elektrische Leistung begrenzt, welche auf Satelliten meist durch Solarpanele geliefert wird.

Ein weiterer großer Unterschied zwischen chemischen und elektrischen Triebwerken ist ihr Schub. Dieser entspricht der Kraft mit der eine Rakete beschleunigt wird und lässt sich durch $|T| = \dot{m} v_{ex}$ berechnen, wobei $\dot{m}$ der Massenfluss und v_{ex} die Ausstoßgeschwindigkeit des Treibstoffs ist. Während chemische Triebwerke meist Ausstoßgeschwindigkeiten von etwa 3 km/s erreichen, lässt sich der Massenfluss nahezu beliebig skalieren, weshalb die hohen Schübe erzeugt werden können, welche zur Überwindung der Atmosphäre beim Start notwendig sind. Elektrische Triebwerke erreichen hingegen deutlich höhere Ausstoßgeschwindigkeiten, können aufgrund

ihrer Leistungsbegrenzung allerdings keinen besonders hohen Schub erzeugen. Klarer wird dieser Umstand, wenn man sich vor Augen führt, dass die Energie in chemischen Triebwerken im Treibstoff selbst gespeichert ist, während elektrische Triebwerke diese Energie extern zuführen müssen, wobei der Treibstoff nur ein Mittel zum Zweck ist. Dank ihrer hohen Ausstoßgeschwindigkeiten sind elektrische Triebwerke allerdings sehr treibstoffeffizient, weil sie denselben Schub mit einem niedrigeren Massendurchfluss erzeugen können. Deshalb sind sie besonders für Anwendungen im Orbit interessant, da mit elektrischen Triebwerken wertvolle Startmasse eingespart werden kann, während Orbitalmanöver meist keine großen Schübe benötigen.

Im Folgenden soll der Aufbau und die Funktionsweise eines RITs genauer beschrieben werden. Grundlegend bestehen RITs aus zwei Bereichen. Zum einen aus der Ionisationskammer, in der ein Niedertemperaturplasma erzeugt und aufrecht erhalten wird und zum anderen aus dem Extraktionsgitter, welches Ionen aus dem Plasma hinaus in das Vakuum beschleunigt. Dieser grundlegende Aufbau ist auf Abbildung 1.1 schematisch dargestellt. Bei RITs wird die Ionisation des Treibstoffs durch eine induktiv an das Plasma gekoppelte Spule erreicht, welche mit Radiofrequenz betrieben wird. Die Spule ist um eine - häufig zylinderförmige - Ionisationskammer gewickelt, wodurch sie ein magnetisches Wechselfeld entlang der Zylinderachse erzeugt. Dieses Wechselfeld erzeugt wiederum ein elektrisches Wirbelfeld, welches die freien Elektronen innerhalb des Gases beschleunigt. Da die Ionen über eine deutlich größere Masse verfügen, können diese als stationär angenommen werden. Die Radiofrequenz muss nun so abgestimmt werden, dass die Elektronen mit den Atomen kollidieren, bevor sie durch das elektrische Feld wieder in die andere Richtung beschleunigt werden. Bei einer Kollision schlägt das freie Elektron ein Elektron aus der Atomhülle und ionisiert so den Treibstoff. Bei einer guten Auslegung wird dadurch ein stabiles Plasma erzeugt. Da sich diese Arbeit mit der Extraktion der Ionen beschäftigt, soll diese oberflächliche Beschreibung der Ionisationskammer ausreichen. Dem Extraktionsgitter wird hingegen das nächste Unterkapitel (1.2) gewidmet.

Zunächst sei aber noch auf ein weiteres wichtiges Teil des Triebwerks hingewiesen, nämlich den Neutralisator. Für dessen Notwendigkeit muss sich in Erinnerung gerufen werden, dass ein RIT für den Einsatz auf einem Satelliten ausgelegt ist. Durch die Extraktion der Ionen aus dem Plasma würde sich der Satellit mit der Zeit stark negativ aufladen und die Elektronik gefährden. Deshalb muss für jedes Ion auch ein Elektron vom Satelliten ausgestoßen werden. Dies wird mithilfe einer Hohlkathode erreicht, die neben dem Extraktionsgitter Elektronen in das Vakuum entsendet [1].

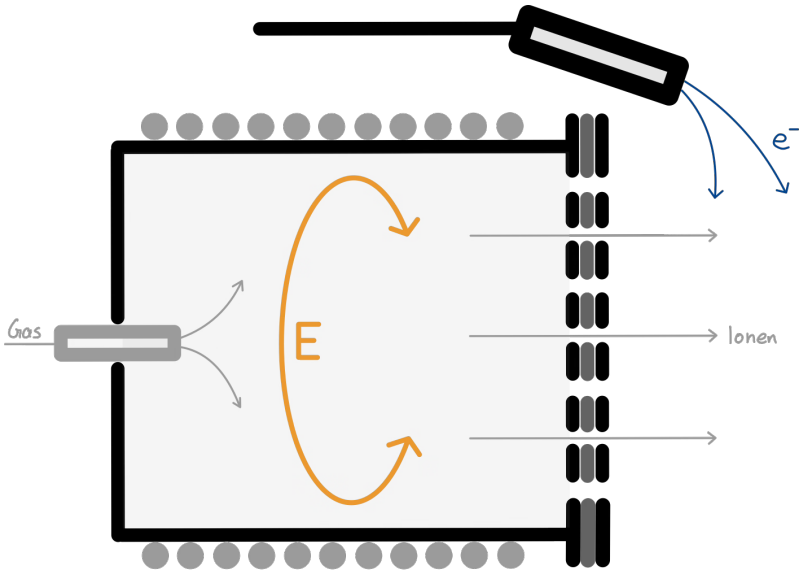


Abbildung 1.1: Schematischer Aufbau eines RITs bestehend aus Ionisationskammer und Extraktionsgittersystem. Der Treibstoff wird in die Ionisationskammer eingelassen, in der er anschließend durch ein elektrisches Wirbelfeld ionisiert wird. Die Gitter beschleunigen die Ionen ins Vakuum, wo sie wieder mit Elektronen kombinieren, die der externe Neutralisator entsendet. Abbildung inspiriert durch [1, S. 11].

Für das RAM-EP Projekt, in dessen Rahmen dieses Studienprojekt ausgearbeitet wird, soll die Nutzung von Restatmosphäre auf niedrigen Umlaufbahnen als Treibstoff für ein RIT erprobt werden. Das würde es niedrig fliegenden Satelliten ermöglichen, der atmosphärischen Reibung mithilfe eines RITs entgegenzuwirken, ohne dafür eigens Treibstoff mitführen zu müssen. So könnte die Missionsdauer niedrig fliegender Satelliten erheblich verlängert werden. Dies ist nur möglich, da RITs mehr oder weniger unabhängig vom Treibstoff operieren können, solange er sich ionisieren lässt. Im Rahmen des Experiments wird das Triebwerk - entsprechend der atmosphärischen Zusammensetzung - mit molekularem Stickstoff und Sauerstoff betrieben.

1.2 Extraktionsgitter

Zur Extraktion aus dem Plasma wird nicht nur eines, sondern meist drei Gitter verwendet. Diese werden hintereinander angeordnet und entsprechend ihrer Aufgabe benannt. Direkt an die Ionisationskammer schließt das **Abschirmgitter** (engl. screen grid) an, welches die Extraktion von der Ionisation trennt. Das Plasma bildet an dieser Barriere eine sogenannte Plasma-**Randschicht** aus, welche in Kapitel 2.2 genauer erläutert wird. In dieser Randschicht entsteht ein Potentialgefälle, da das Plasma aufgrund der physikalischen Prozesse in der Randschicht ein höheres Potential im Vergleich zum Abschirmgitter aufweist. Durch dieses werden die Ionen aus

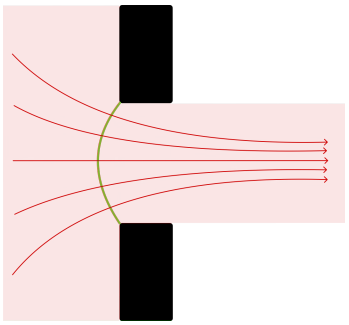


Abbildung 1.2: Schematische Darstellung des Plasma-Meniskus im Bereich eines Lochs. Beispielhaft sind einige Ionentrajektorien eingezeichnet, die senkrecht zur Äquipotentialfläche verlaufen.

dem Plasma in Richtung Gitter beschleunigt. Da das Gitter allerdings keine ausgedehnte Fläche ist, sondern Löcher aufweist, bildet sich im Bereich jedes Lochs der sogenannte Plasma-**Meniskus** aus, welcher die Äquipotentialfläche vom Potential des Abschirmgitters fortführt (siehe Abbildung 1.2). Da die Ionen senkrecht zum Meniskus beschleunigt werden, ist seine Form entscheidend für die Trajektorie der Ionen.

Hinter dem Abschirmgitter befindet sich das **Beschleunigungsgitter** (engl. acceleration grid), auf welches abschließend das **Abbremsgitter** (engl. deceleration grid) folgt. Die Wahl die Ionen nach der Beschleunigung nochmal abzubremsen, hat den Hintergrund einer besseren Strahlfokussierung. Außerdem wird durch diese Anordnung verhindert, dass die Elektronen aus dem Neutralisator zurück in das Triebwerk fliegen. Die Spannungen der Extraktionsgitter, im Vergleich zum Satelliten, werden gewöhnlich so gewählt, dass das Abschirmgitter eine hohe positive Spannung (etwa 1 bis 2 kV) und das Beschleunigungsgitter eine leicht negative Spannung (etwa -100 bis -200 V) aufweist. Hingegen liegt das Abbremsgitter gewöhnlich auf dem Satellitenpotential bei etwa 0 V. Die Spannung U, welche die Ionen der Masse M und Ladung q also durchlaufen, wird durch das Potential des Plasmas und des Abschirmgitters vorgegeben. Diese Spannung beschleunigt die Ionen gemäß Gleichung 1.1 auf ihre Ausstoßgeschwindigkeit.

$$v_{ex} = \sqrt{\frac{2qU}{M}} \tag{1.1}$$

Bei der Auslegung des Extraktionsgittersystems gilt es also für das Wechselspiel aus geometrischen Maßen und Spannungen eine ideale Anordnung zu finden, welche für verschiedene Arbeitspunkte des Plasmas eine bestmögliche Fokussierung liefert. Optimiert wird dabei die Form des elektrischen Feldes, welches für die Beschleunigung

der Ionen verantwortlich ist. Stellvertretend für dieses werden meist die Äquipotentialflächen in die Anordnung eingezeichnet, auf denen das elektrische Feld im rechten Winkel steht. Ähnlich wie in der geometrischen Optik soll so der *Strahlengang* der Ionen optimiert werden, weshalb diese Betrachtung auch **Ionenoptik** genannt wird [1].

Darüber hinaus wird die Auslegung eines Extraktionsgittersystems durch das Zusammenspiel von Ionentransparenz und Neutralgasaustritt bestimmt. Je größer das Verhältnis von Lochfläche zu Gitterfläche ist, desto größer ist die Ionentransparenz. Große Löcher bedeuten aber auch, dass viel Neutralgas aus dem Triebwerk entweichen kann und damit wertvoller Treibstoff verloren geht. Um dies zu vermeiden, wird der Lochradius des Abschirmgitters größer gewählt als der des Beschleunigungsgitters. Dadurch sieht das Neutralgas die effektive Lochfläche des Beschleunigungsgitters. Die Ionen werden hingegen durch die angelegte Spannung durch das Gittersystem geleitet und effektiv durch die Lochfläche des Abschirmgitters extrahiert. Auf diese Weise kann eine hohe Transparenz für Ionen und eine niedrige Transparenz für das Neutralgas erreicht werden [2, S. 189].

2 Theoretische Grundlage der Simulation

Simulationen sind ein wichtiges Werkzeug, um physikalische Prozesse im Vorfeld eines experimentellen Aufbaus zu modellieren. Mit ihrer Hilfe können wichtige Parameter bereits vor der Produktion der Bauteile abgeschätzt werden. Allerdings ist die Güte einer jeden Simulation maßgeblich von zwei Faktoren abhängig. Einerseits von der Genauigkeit der theoretischen Beschreibung des Problems und andererseits von den angenommenen Eingabeparametern. Um sinnvolle Ergebnisse zu erhalten, müssen die Eingabewerte möglichst nah mit den realen Bedingungen übereinstimmen. Deshalb werden in diesem Kapitel die verwendeten theoretischen Konstrukte und die daraus folgenden Simulationsparameter genauer erläutert.

2.1 Definition Plasma

Unvermeidbar vor einer genauen theoretischen, physikalischen Beschreibung ist die genaue Definition des beschriebenen Systems. Deshalb sollen nun grundlegende Eigenschaften von Plasmen geklärt und in mathematischen Formeln festgehalten werden.

Im Alltag eines jeden Menschen sind Aggregatzustände eine allgegenwärtige Konstante. Beispielsweise ist Wasser ein Molekül, welches regelmäßig in den verschiedenen Aggregatszuständen Anwendung findet. Am Morgen trinken viele Menschen eine heiße Tasse Kaffee oder Tee, für die Wasser aufgekocht und dabei teilweise in einen *gasförmigen* Zustand gebracht wird. Im Sommer wird hingegen *festes* Eis verwendet, um ein *flüssiges* Glas Wasser zu kühlen. Entscheidend für den vorliegenden Aggregatzustand ist nur die Energie, die eine bestimmte Menge des vorliegenden Stoffes besitzt. Dieser Mechanismus lässt sich einfach am Beispiel von Wasser (H_2O) erklären. Führt man der festen Kristallstruktur von Eis durch Erwärmung Energie zu, so lösen sich die starren Bindungen und das Wasser wird flüssig. Wird das Wasser weiter erhitzt, verdampft es und geht in den gasförmigen Zustand über. Entscheidend für einen Phasenübergang ist also das Aufbrechen von davor dominanten Bindungen.

Doch obwohl uns dieser in unserem Alltag nicht so präsent ist, gibt es noch einen vierten Aggregatszustand nach der Gasphase. Fügt man dem Gas immer weiter Energie hinzu, so werden Energien erreicht, bei denen Atome oder Moleküle ionisiert werden können. Das bedeutet, dass Elektronen, die zuvor an den Kern gebunden waren, aus der Hülle geschlagen werden und anschließend als freie Ladungsträger vorliegen. Der daraus resultierende Zustand aus ionisierten Atomen (oder Molekülen) und freien

Elektronen wird **Plasma** genannt. Dies führt uns zu einer anfänglichen Beschreibung des Plasmas als „quasineutrales Gas geladener und ungeladener Teilchen das kollektives Verhalten zeigt [3, S. 6]“. Denn obwohl lokal Ladungsträger voneinander getrennt werden, weist das Plasma auf der makroskopischen Ebene eine Neutralität auf.

2.1.1 Klassifikation von Plasmen

Während die anderen drei Aggregatszustände fest durch Phasenübergänge voneinander abgegrenzt sind, ist der Übergang zum Plasma fließend. Entscheidend ist dabei der Ionisationsgrad des Plasmas, welcher den Anteil von ionisierten Atomen an der Gesamtzahl der Atome angibt. Aus der Ionendichte n_i und der Dichte des Neutralgases n_n lässt sich der Ionisationsgrad X wie folgt berechnen:

$$X = \frac{n_i}{n_n + n_i} \tag{2.1}$$

Neben dem Ionsiationsgrad sind die Temperatur T und die Teilchendichte n zwei weitere essentielle Größen zur Beschreibung von Plasmen. Da für die Massen von Ionen und Elektronen gilt, dass $m_i \gg m_e$, werden die beschreibenden Größen der beiden Teilchen meist getrennt voneinander betrachtet. Aufgrund dieses Massenunterschieds erreichen Elektronen auch früher hohe Geschwindigkeiten, weshalb Elektronentemperatur und -dichte maßgeblich für das Verhalten des Plasmas sind.

Des Weiteren umfassen Plasmen sehr große Dichte- und Temperaturbereiche, weshalb sie zusätzlich in unterschiedliche Klassen unterteilt werden. Dabei kann nicht von einer einheitlichen *Plasmatheorie* gesprochen werden. Stattdessen müssen je nach Dichte und Temperatur unterschiedliche physikalische Modelle zugrunde gelegt werden.

Bei den meisten Plasmen handelt es sich um **ideale Plasmen**, bei denen die Ionen und Elektronen nur durch Stöße miteinander wechselwirken. Das ist gewährleistet, wenn die thermische Bewegungsenergie der Stoßpartner deutlich größer ist, als die „potentielle Energie im gegenseitigen Feld [4, S. 6]“. Aus der Ionendichte n errechnet sich der mittlere Teilchenabstand zu $n^{-\frac{1}{3}}$. Mit diesem lässt sich ein Ungleichung aufstellen, die jedes ideale Plasma erfüllen muss [4]:

$$\frac{3}{2} k_B T \gg \frac{e^2}{4\pi\epsilon_0} n^{\frac{1}{3}} \tag{2.2}$$

Dabei ist hier und im Folgenden k_B die Boltzmannkonstante, T die Temperatur, e die Elementarladung und ϵ_0 die Permittivität des Vakuums.

Eine weitere Unterscheidung wird zwischen **Nieder-** und **Hochtemperaturplasmen** getroffen. In Niedertemperaturplasmen weisen die schweren Teilchen, also die neutralen Atome und Ionen, bei niedrigen Drücken nur etwa Raumtemperatur auf. Dahingegen besitzen die Elektronen sehr hohe Temperaturen, um überhaupt Ionisation zu ermöglichen. Außerdem liegt in diesen Plasmen ein Ionisationsgrad $X \ll 1$ vor. Im Gegensatz dazu sind Hochtemperaturplasmen nahezu vollständig ionisiert und sowohl die Ionen, als auch die Elektronen weisen hohe Temperaturen auf. Während diese Art von Plasmen in der Fusionsforschung zum Einsatz kommt, sind die meisten technischen Plasmen, welche uns im Alltag beispielsweise in Lampen oder Fernsehbildschirmen begegnen Niedertemperaturplasmen [5].

Das Plasma des RITs wird in dieser Arbeit als ideales Niedertemperaturplasma behandelt.

2.1.2 Debye-Länge

Um die Quasineutralität zu gewährleisten, muss die Summe aus positiven und negativen Ladungsträgern innerhalb eines Volumenelements Null ergeben. Offensichtlich ist diese Bedingung nicht für beliebig kleine Volumenelemente gegeben, da sich auf mikroskopischer Ebene voneinander getrennte Ladungen befinden, die inhomogene elektrische Potentialfelder bewirken. Es stellt sich also die Frage, in welcher Größenordnung ein Volumenelement liegen muss, damit diese mikroskopischen Effekte vernachlässigt werden können. Hierbei kommt eine wichtige Eigenschaft des Plasmas ins Spiel; nämlich die Abschirmung von Ladungen. Diese bewirkt, dass Ladungen im Plasma kein Coulomb-Potential, sondern ein abgeschirmtes Potential ausbilden. Um die Frage also zu beantworten, ist die Größenordnung dieser Abschirmung zu bestimmen. Dazu muss die Poisson-Gleichung für die Ladungsverteilung, bestehend aus Punktladung und der Raumladung von Ionen und Elektronen, gelöst werden.

In einer ausführlichen Rechnung die [4, S. 7-9] entnommen werden kann, lässt sich das Potential einer solchen abgeschirmten Punktladung bestimmen:

$$\Phi(r) = \frac{e}{4\pi\epsilon_0}\frac{1}{r}\exp\left(-\frac{r}{\lambda_D}\right). \tag{2.3}$$

Dabei ist λ_D die **Debye-Länge**, welche ein Maß für die Abschirmung des Coulomb-

Potentials ist. Innerhalb einer Debye-Länge ist das Potential auf das $1/e$-fache des Coulomb-Potentials abgefallen.

$$\lambda_D = \sqrt{\frac{\epsilon_0 k_B T}{2 n_0 e^2}} \tag{2.4}$$

Die Debye-Länge liefert also eine Größenordnung für die Ausdehnung, die ein Plasma mindestens haben muss, um die obige Bedingung der Quasineutralität zu erfüllen. Wie die durchgeführte Rechnung aber zeigt, darf die Ladungstrennung auf mikroskopischer Ebene keinesfalls vernachlässigt werden. Diese ist beispielsweise dringend notwendig, um die Interaktion eines Plasmas mit elektromagnetischer Strahlung zu beschreiben.

Ferner lässt sich nun eine sogenannte **Debye-Kugel** mit dem Radius der Debye-Länge definieren. Damit das Plasma kollektives Verhalten aufweist, muss die Teilchenanzahl innerhalb der Debye-Kugel deutlich größer als Eins sein. Denn nur für eine große Teilchenzahl ist die statistische Beschreibung der Teilchen gerechtfertigt [3].

$$N_D = \frac{4}{3}\lambda_D^3 \pi n \gg 1 \tag{2.5}$$

2.1.3 Plasmafrequenz

Im vorangegangenen Kapitel hat die Betrachtung des statischen Verhaltens einer Punktladung zur Debye-Länge geführt. In diesem Kapitel soll nun das dynamische Verhalten des Plasmas nach kollektiven lokalen Störungen der Ladungsverteilung betrachtet werden.

Aufgrund der hohen Temperaturen innerhalb eines Plasmas sind die Teilchen (insbesondere die Elektronen) in ständiger Bewegung. Dies hat temporäre Ladungsverschiebungen zur Folge, die elektrische Gegenfelder erzeugen. Angenommen Elektronen und Ionen verschieben innerhalb eines lokalen Bereichs kollektiv gegeneinander. Dann überlappen sich die Ladungswolken immer noch zu einem Großteil, allerdings bilden sich an den Rändern lokale Überschussladungen aus (siehe Abbildung 2.1). Durch dieses Feld werden die Elektronen im quasineutralen Bereich beschleunigt und beginnen das Feld wieder auszugleichen. Allerdings schießen sie über einen Ausgleich hinaus und es bildet sich analog ein elektrisches Feld in die entgegengsetzte Richtung aus. Dadurch entsteht eine Oszillation, die mit der sogenannten **Plasmafrequenz** schwingt [4, S. 11f.]:

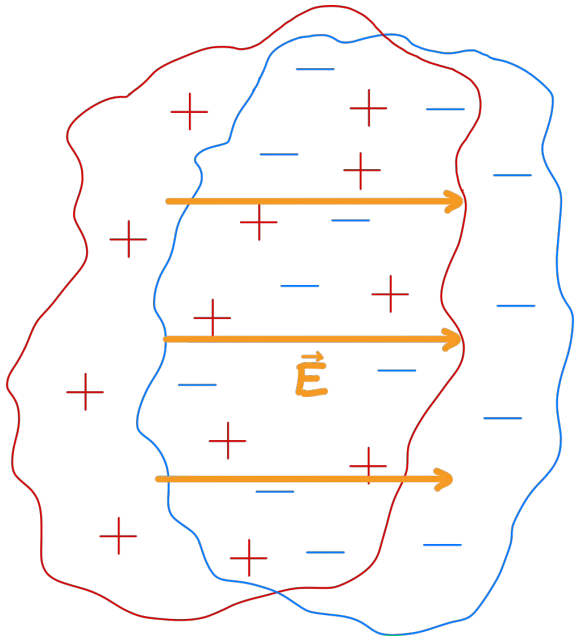


Abbildung 2.1: Eine lokale Ladungsverschiebung führt zur Ausbildung eines elektrischen Feldes im quasineutralen Plasma [6, S. 6].

$$\omega_{pe} = \sqrt{\frac{e^2 n_e}{\epsilon_0 m_e}} \tag{2.6}$$

Daraus lässt sich eine weitere wichtige Plasmaeigenschaft ableiten. Denn damit die elektromagnetischen Wechselwirkungen überwiegen, muss die Plasmafrequenz größer sein, als die Frequenz ν_{en}, in der Elektronen mit Neutralgasteilchen stoßen. Wäre diese Bedingung nicht erfüllt, so würde sich die Anordnung ähnlich zu reinem Neutralgas verhalten [7, S. 6].

2.1.4 Zusammenfassung

Die in den beiden Unterkapiteln diskutierten Bedingungen, sollen nun einmal übersichtlich zusammengefasst werden [3].

- $L \gg \lambda_D$
  Die Maße des Plasmas müssen die Debye-Länge deutlich übersteigen, damit die Bedingung der Quasineutralität erfüllt ist.

- $N_D \gg 1$
  Die Anzahl der Teilchen innerhalb der Debye-Kugel muss wesentlich größer als Eins sein, damit kollektives Verhalten gewährleistet ist.

- $\nu_{pe} \gg \nu_{ne}$ Die Plasmafrequenz muss erheblich größer als die Stoßfrequenz

zwischen Elektronen und Neutralgasteilchen sein, damit die Plasmadynamik gegenüber der Gasdynamik dominiert.

2.2 Plasmarandschichten

Nach der ausführlichen Definition des Plasmas in Kapitel 2.1, sollen nun die Phänomene innerhalb der Plasmarandschicht untersucht werden. Diese ist von Interesse, da jedes im Labor erzeugte Plasma räumlich begrenzt ist. Diese Begrenzung stört die Homogenität des Plasmas und ruft die Ausbildung eines elektrischen Potentialfeldes hervor. Im Kontext dieser Arbeit bildet sich eine solche Randschicht beispielsweise vor dem Extraktionsgitter aus. Demnach ist ein Verständnis der Vorgänge am Plasmarand essentiell für die Ermittelung der Simulationsparameter.

Im Folgenden wird also die Wechselwirkung zwischen einem quasineutalen Plasma und einer nicht geerdeten *floatenden* Oberfläche untersucht. Da die Elektronen aufgrund ihrer geringen Masse deutlich beweglicher sind als das Neutralgas und die Ionen, treffen diese als erstes auf die eingebrachte Oberfläche, wodurch die Quasineutralität lokal verletzt wird. Durch den Elektronenstrom wird sich die Oberfläche im Vergleich zum Plasma negativ aufladen. Dadurch entsteht eine Potentialdifferenz, dessen Gradient ein elektrisches Feld hervorruft. Dieses Feld wirkt dem Verlust an Elektronen entgegen und zieht Ionen aus dem Plasma, sodass sich ein Gleichgewicht einstellt. Dieser Gleichgewichtszustand soll nun weiter untersucht werden. In der folgenden Betrachtung wird das Potential an der Grenze zur Randschicht gleich Null gesetzt [3].

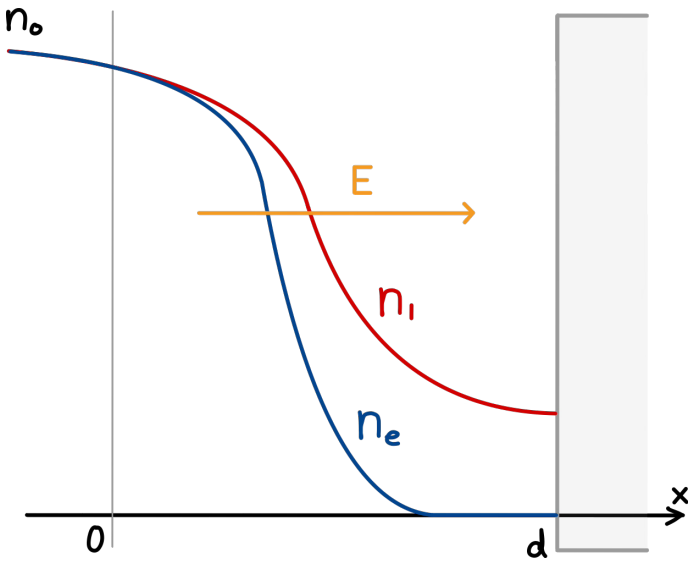


Abbildung 2.2: Die Verteilung der Ionen und Elektronen innerhalb der Randschicht. Da sich die Oberfläche negativ auflädt, werden die Ionen angezogen und die Elektronen abgestoßen. Die Gesamtladung innerhalb der Randschicht inklusive Oberfläche ist Null [3, S. 164].

2.2.1 Bohm-Geschwindigkeit

Für die elektrostatische Beschreibung der gegebenen Anordnung ist wieder die Poisson-Gleichung zu lösen, für die zunächst ein Ausdruck für die Dichte von Elektronen und Ionen gefunden werden muss. Während für die Elektronendichte eine Boltzmann-Verteilung angenommen werden kann, muss die Ionendichte über die Ladungs- und Energieerhaltung hergeleitet werden. Die daraus resultierende Differentialgleichung kann nur nummerisch gelöst werden, doch mithilfe eines Integraltricks lässt sich die DGL auf folgende Weise ausdrücken:

$$\frac{1}{2}\left(\frac{\mathrm{d}\Phi(x)}{\mathrm{d}x}\right)^2 = \frac{en_0}{\epsilon_0}\left[k_B T_e \exp\left(\frac{e\Phi(x)}{k_B T_e}\right) - k_B T_e + 2E_0\left(1 - \frac{e\Phi(x)}{E_0}\right)^{\frac{1}{2}} - 2E_0\right] \tag{2.7}$$

Obwohl die DGL auch nach dieser Umformung nicht analytisch lösbar ist, lässt sich aus der Gleichung eine Bedingung ableiten, welche für eine Lösung der DGL immer erfüllt sein muss. Denn wegen des quadratischen Terms auf der linken Seite, wird diese immer ≥ 0 sein. Dasselbe muss also auch für die Klammer auf der rechten Seite gelten, wodurch sich eine Ungleichung ergibt. In dieser kann anschließend die Exponential- und die Wurzelfunktion in einer Taylor-Reihe bis zur 2. Ordnung um

$$\frac{e\Phi(x)}{k_B T_e} = 0 \qquad \text{bzw.} \qquad \frac{e\Phi(x)}{E_0} = 0 \tag{2.8}$$

entwickelt werden. Vereinfacht man nun den resultierenden Term, so ergibt sich ein

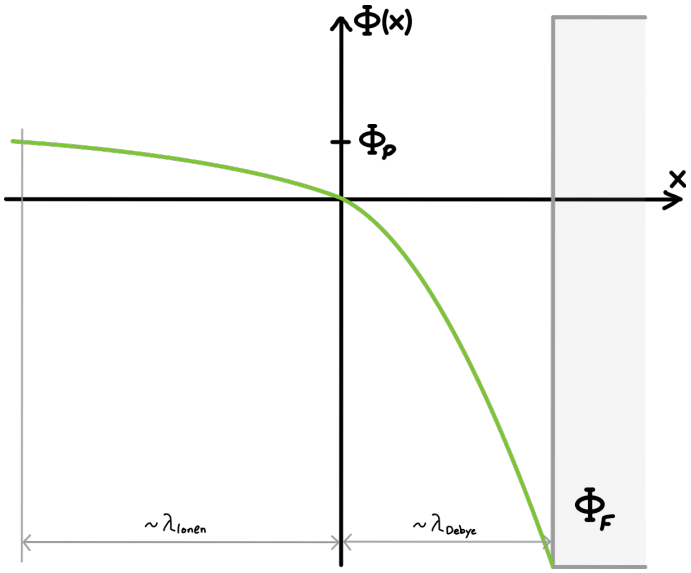


Abbildung 2.3: Diese Abbildung zeigt eine skizzenhafte Darstellung des Potentialverlaufs in der Randschicht, da die Poisson-Gleichung nie analytisch gelöst wurde. Während der Plasma Bulk auf dem Plasmapotential Φ_P liegt, herrscht auf der Oberfläche das Floating Potential Φ_F [3, S. 167].

Ausdruck für das untere Geschwindigkeitslimit, das auch **Bohm-Geschwindigkeit** genannt wird. Die ausführliche Herleitung dieser Größe kann [3, S. 163-166] entnommen werden.

$$v_0 \geq v_B := \sqrt{\frac{k_B T_e}{M}} \tag{2.9}$$

In den meisten Fällen ist dieser Ausdruck mit $v_0 = v_B$ erfüllt [7]. Die Ionen werden also mit der Bohm-Geschwindigkeit in die Plasmarandschicht eintreten, um den Gleichgewichtszustand aufrecht zu erhalten. Spannenderweise ist die Eintrittsgeschwindigkeit dabei nur von der Elekronentemperatur und der Ionenmasse abhängig. Innerhalb der Randschicht werden die Ionen dann weiter in x-Richtung beschleunigt, sodass sie senkrecht auf die Oberfläche auftreffen. Dieser Umstand ist sehr hilfreich für die Plasmaextraktion, da die Ionen bereits *von selbst* das Plasma verlassen. Demnach müssen sie an der Randschicht nur noch mithilfe einer Extraktionsspannung abgesaugt werden.

2.2.2 Plasmapotential

Obwohl die DGL für das Potential auch weiterhin nicht gelöst wird, lassen sich nun weitere Aussagen über den Potentialverlauf treffen. Denn damit die Ionen mit der Bohm-Geschwindigkeit in die Plasmarandschicht eintreten, müssen sie zuvor beschleunigt werden. Gerade bei Niedertemperaturplasmen kann man die thermische Geschwindigkeit der Ionen vernachlässigen und davon ausgehen, dass diese nur

durch eine Potentialdifferenz auf Geschwindigkeit gebracht werden. Der Bereich in dem die Beschleunigung der Ionen geschieht wird **Vorschicht** genannt. Ferner wird das beschleunigende Potential zwischen quasineutralem Plasma und Randschicht als **Plasmapotential** Φ_P bezeichnet, welches sich einfach aus der Energieerhaltung, zusammen mit dem oben gefundenen Ausdruck für die Bohm-Geschwindigkeit, herleiten lässt:

$$\Phi_P = \frac{k_B T_e}{2e} \tag{2.10}$$

Der Potentialabfall innerhalb der Randschicht (das sogenannte **Floating Potential**) lässt sich über die Betrachung des Teilchenflusses von Elektronen und Ionen auf die eingebrachte Oberfläche herleiten. Im Gleichgewichtsfall ist dieser identisch, was in einer ausführlichen Rechnung, die in [3, S. 168f.] ausgeführt ist, zum Floating Potential führt:

$$\Phi_F = -\frac{k_B T_e}{e} \ln \sqrt{\frac{M}{2\pi m}} \tag{2.11}$$

Aus diesen Ergebnissen kann nun der ungefähre Potentialverlauf skizziert werden (siehe Abbildung 2.3). Dabei ist zu berücksichtigen, dass der Betrag des Floatingpotentials um den Faktor $\ln(M/2\pi m)$ größer ist, als das Plasmapotential. Außerdem liegt die Größenordnung der Randschicht innerhalb einiger Debye-Längen, während die Vorschicht in der Größenordnung der mittleren freien Weglänge der Teilchen liegt. Der Potentialanstieg ist in der Vorschicht also wesentlich flacher [3].

2.3 Raumladungsgesetz

Die Betrachtung in Kapitel 2.2 hat gezeigt, dass Ionen innerhalb der Plasmaranschicht senkrecht auf die Oberfläche beschleunigt werden. Will man also Ionen aus dem Plasma extrahieren, so muss lediglich ein Loch in die Oberfläche geschnitten werden. Setzt man nun hinter diese Oberfläche eine zweite Oberfläche mit einem niedrigeren Potential, so werden die Ionen weiter beschleunigt. Macht man aus diesem einen Loch viele Löcher, so sind das Abschirm- und Beschleunigungsgitter geboren.

Interessant für die Ionenextraktion ist nun der maximale Strom an Ionen, der mithilfe der Gitter extrahiert werden kann. In einem Stromleiter besteht dem Ohm'schen Gesetz zufolge ein linearer Zusammenhang zwischen Spannung und Stromstärke ($U = R \cdot I$). Im Extraktionskanal wird das elektrische Feld hingegen durch die Raumladung der Ionen beeinflusst, sodass dieser lineare Zusammenhang aufbricht. Deshalb soll nun die Abhängigkeit zwischen Stromstärke und Spannung bei einer Feldextraktion mit zwei Elektroden ermittelt werden.

Vereinfachend wird dafür der Einfluss der ausgeschnittenen Löcher vernachlässigt und davon ausgegangen, dass das Potential im Bereich dieser senkrecht zur x-Achse konstant verläuft. Damit lässt sich das Problem als Plattenkondensator mit positiv geladenen Ionen im Zwischenraum betrachten. Wie für jedes elektrostatische Problem muss auch hier wieder die Poisson-Gleichung aufgestellt und gelöst werden. Unter der Bedingung, dass die Ionen der Masse M zwischen den Platten im Abstand d mit der Spannung U beschleunigt werden, ergibt sich dann das sogenannte **Child-Langmuir-Raumladungsgesetz** [3]:

$$j = \frac{4}{9}\epsilon_0\sqrt{\frac{2e}{M}}\frac{U^{\frac{3}{2}}}{d^2} \tag{2.12}$$

Dieses beschreibt die Begrenzung der Ionenstromdichte durch zwei entscheidende Parameter der Anordnung. Einerseits findet man den gesuchten Zusammenhang zwischen Strom und Spannung $I \propto U^{\frac{3}{2}}$. Für hohe Spannungen bewirkt eine Steigerung dieser also nur eine vergleichsweise geringe Erhöhung des Stroms. Andererseits zeigt das Child-Langmuir-Gesetz eine wichtige geometrische Limitierung des Extraktionsgitters auf, da auch der Gitterabstand d die Stromdichte begrenzt $j \propto \frac{1}{d^2}$. Je größer der Gitterabstand, desto mehr Raumladung passt zwischen die Gitter, wodurch das Potential stärker geschwächt wird.

Dabei ist es allerdings wichtig anzumerken, dass dieses Gesetz nur die Limitierung des Stroms durch die Extraktion selbst angibt. Der Strom ist aber auch durch die Bereitstellung einer ausreichenden Anzahl an Ladungsträgern durch das Plasma begrenzt. In einem RIT ist beispielsweise die geringe Ionendichte der limitierende Faktor, sodass die Stromstärke in der Anwendung geringer ausfällt, als das Child-Langmuir-Gesetz vorgibt.

2.4 Emittanz

Die Emittanz stellt eine wichtige Größe zur Beschreibung von Ionenstrahlen dar. Ob sich die Emittanz als Güteparameter anbietet wird in einem späteren Kapiteln noch diskutiert. Nichtsdestotrotz soll der Begriff in diesem Kapitel eingeführt werden.

Als Phasenraum wird allgemein der 6-dimensionale Raum bezeichnet, der durch die drei Orts- und Impulskoordinaten eines Teilchenensembles aufgespannt wird. Jedem Teilchen kann ein bestimmter Punkt in diesem Phasenraum zugeordnet werden. Da ein 6-dimensionaler Raum visuell nicht darstellbar ist und die Berechnungen

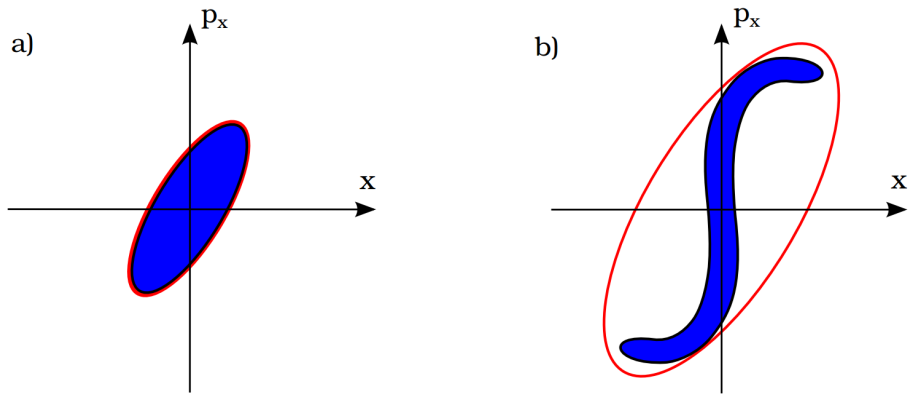


Abbildung 2.4: Darstellung des 1D Phasenraums für die allgemeinen Koordinaten x und p_x: 2D-Projektion eines Teilchenstrahls vor (a) und nach (b) Durchlauf durch ein nichtlineares optisches System. Es ist zu erkennen, wie sich die Fläche, welche die Teilchen einnehmen, nicht verändert, während sich die einhüllende Fläche vergrößert [7].

erheblich erschwert, wird bei Teilchenstrahlen der Phasenraum meist getrennt für den transversalen und longitudinalen Anteil betrachtet, wobei bei kontinuierlichen Strahlen der longitudinale Anteil vernachlässigt werden kann. Für radialsymmetrische Strahlen genügt es daher letztlich, den Raum zu betrachten, der durch den Abstand vom Zentrum r und die lokale Änderung des Impulses in r-Richtung relativ zum Impuls in Strahlrichtung $r' = p_r/p_x$ aufgespannt wird.

Plottet man nun alle Teilchen eines Strahls in dieses rr'-Koordinatensystem, so erhält man eine Verteilung im Phasenraum, die zeigt, wie homogen der Impuls der Teilchen über den Ort verteilt ist. Die Fläche, welche die Teilchen im Phasenraum einnehmen, ist nach dem Liouville-Theorem konstant, solange nur konservative Kräfte auf den Strahl einwirken. Die elektromagnetische Kraft ist konservativ, weshalb die Fläche im vorliegenden Fall der Ionenextraktion also konstant bleiben sollte. Dabei ist allerdings zu beachten, dass dieses Theorem für den 6-dimensionalen Raum gilt. Da der longitudinale Anteil im rr'-Diagramm nicht dargestellt ist, wird sich die Fläche bei einer Beschleunigung des Strahls in Strahlrichtung verkleinern. Denn durch eine Erhöhung von p_x verkleinert sich r', ohne das sich der transversale Impuls p_r verringert hat.

Als Emittanz bezeichnet man nun die Größe der Fläche, welche die Summe der Punkte im Phasenraum einhüllt (siehe Abbildung 2.4). In der Regel wird diese einhüllende Form so gewählt, dass sie nicht alle, sondern eine bestimmte Prozentzahl an Teilchen umschließt. Weit verbreitet ist dabei die Nutzung der RMS (Root Mean Squared) Emittanz, welche wie folgt definiert ist:

$$\epsilon_{RMS} = \sqrt{\langle r^2 \rangle \langle r'^2 \rangle - \langle r r' \rangle^2} \tag{2.13}$$

Um weitere Aussagen über die Beschaffenheit des Strahls treffen zu können, nutzt man anschließend eine einfache geometrische Form, wie eine Ellipse, als einhüllende Fläche. Setzt man nun für den Betrag dieser Fläche ϵ_{RMS} ein, so lassen sich die sogenannten Twiss-Parameter bestimmen, mit denen der Strahl nun genauer charakterisiert werden kann. Da diese jedoch im Rahmen dieses Studienprojektes keine weitere Anwendung finden, werden sie nicht weiter erläutert [7, 8].

3 Simulation mit IBSimu

3.1 Allgemeine Informationen

Den Ursprung hat der *Ion Beam Simulator* (IBSimu) an der Universität von Jyväskylä in Finnland, an der im Jahr 2004 eine Simulation für das Design einer Spaltstrahl Plasmaextraktion benötigt wurde. Der dafür eigens angefertigte Code wurde daraufhin am dortigen physikalischen Institut weiterentwickelt, modularisiert und als open-source Code im Jahr 2010 veröffentlicht. Federführend bei der Veröffentlichung des Codes war Taneli Kalvas, der im Jahr 2013 seine Doktorarbeit über seine Arbeit an und mit dem IBSimu veröffentlicht hat [7].

Dabei handelt es sich bei IBSimu nicht um ein eigenständiges Programm, sondern um eine C++ Bibliothek, dessen einzelne Modulbausteine sehr flexibel auf die jeweilige Anwendung angepasst werden können. Diese hohe Flexibilität bedingt allerdings eine geringe Anwenderfreundlichkeit. Denn die Bibliothek besitzt keine Benutzeroberfläche, sondern wird über den *command prompt* und selbst geschriebene Codedateien bedient. Ferner erweist sich die Installation auf einem Windows System als zusätzliche Hürde, da die Bibliothek in einer Linux Umgebung geschrieben wurde. Deshalb muss auf einem Windows System zunächst eine Linux Umgebung geschaffen werden. Im Anhang ist eine kompakte Anleitung zu Inbetriebnahme der Bibliothek zu finden.

3.2 Funktionsweise der Bibliothek

Bevor genauer auf die Verwendung von IBSimu eingegangen wird, soll in diesem Kapitel ein kurzer Überblick über die nummerischen Verfahren der Simulation gegeben werden. Dabei wird allgemein auf die Funktionsweise eingegangen, ohne aber zu tief ins Detail zu gehen. Dieses ist in der Doktorarbeit des Entwicklers [7] zu finden, auf der dieses Kapitel auch basiert.

Die Bibliothek IBSimu bietet die Möglichkeit zwei- oder dreidimensionale Simulationen von Plasmaextraktionen durchzuführen. Grundlage der Simulation ist dabei ein Gitter, welches über die Geometrie gelegt wird, um diese in diskrete Rechenpunkte aufzuteilen. Für jeden dieser sogenannten Knoten wird dann ein Vektor der elektrischen Feldstärke bestimmt, um daraus die Trajektorien der extrahierten Ionen oder Elektronen zu ermitteln.

Zur Implementation des Plasmas wird das in Kapitel 2.2 erläuterte Randschicht-Modell verwendet. Die Trajektorien der Ionen starten am Rand der Simulation, der als Grenze zwischen Vor- und Randschicht angenommen wird. Um Konvergenz zu gewährleisten, müssen die Ionen mindestens mit der Bohm-Geschwindigkeit in die Randschicht eintreten. Um den Einfluss des Plasmas zu simulieren, wird eine exponentiell abfallende Elektronendichte zur Simulation hinzugefügt:

$$\rho_e = en_0 \exp\left(\frac{e(\Phi - \Phi_P)}{k_B T_e}\right) \tag{3.1}$$

Während diese an der Grenzschicht, die auf dem Plasmapotential Φ_P liegt, noch die gesamte Ladung der Ionen kompensiert, wird tiefer im Potential immer weniger Raumladung ausgeglichen.

Zu Beginn jeder Simulation wird das Potential der jeweiligen Elektrodenanodnung über die Laplace-Gleichung $\Delta\Phi = 0$ ermittelt. Das elektrostatische Problem wird also zunächst ohne Berücksichtigung der Ionen, welche eine Raumladung einbringen würden, gelöst. Anschließend wird aus dem Potential über $\boldsymbol{E} = \nabla\Phi$ der Vektor der elektrischen Feldstärke an jedem Knoten bestimmt. Nun können die Teilchen in die Simulation eingesetzt und ihre Trajektorien berechnet werden. Aus diesen kann jedem Knoten eine neue Ladungsdichte zugeordnet werden. Da sich diese Ladung auf das elektrische Potential auswirkt, muss dieses nun neu berechnet werden. Diesmal muss dafür die Poisson-Gleichung $\Delta\Phi = -\rho/\epsilon_0$ gelöst werden, aus der sich wieder die elektrische Feldstärke ermitteln lässt. Ab hier wiederholt sich der Prozess solange, bis das Potential auf einen statischen Zustand konvergiert. Dann können die Simulationsdaten gespeichert und Teilchendiagnostiken durchgeführt werden. Der Ablauf ist auch nochmal auf dem Flussdiagramm in Abbildung 3.1 visualisiert.

Das Simulationsgitter wird über kartesische Koordinaten definiert, bei dem die Knoten in jeder Dimension um die Schrittweite h voneinander entfernt liegen:

$$x_i = x_{min} + h \cdot i \tag{3.2}$$

$$y_j = y_{min} + h \cdot j \tag{3.3}$$

$$z_k = z_{min} + h \cdot k \tag{3.4}$$

So kann jeder Knoten anstelle seiner x-, y- und z-Koordinate auch über die Indizes i, j und k definiert werden.

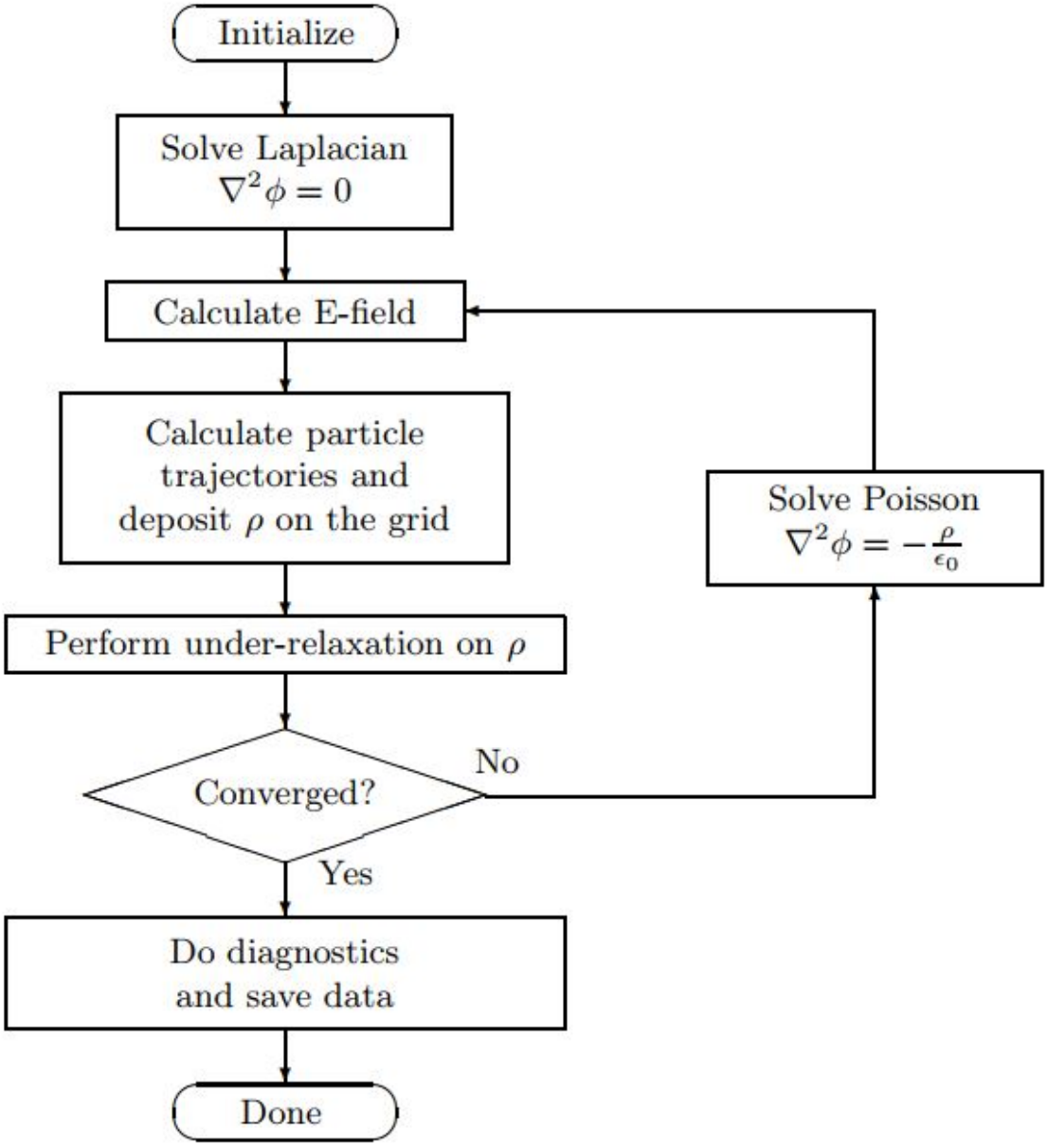


Abbildung 3.1: Flussdiagramm zum Ablauf einer Simulation, entnommen aus [7, S. 61].

Zur Lösung der oben genannten DGLs wird nun die Finite-Differenzen-Methode (FDM) angewandt. Bei dieser werden die Ableitungen diskretisiert, also über existierende Funktionswerte ausgedrückt. In diesem Fall werden dazu zwei Taylor-Reihen genutzt, um $f(x_0 + h)$ und $f(x_0 - h)$ um x_0 zu entwickeln:

$$f(x_0 + h) \approx f(x_0) + f'(x_0)h + \frac{f''(x_0)}{2}h^2 + \frac{f'''(x_0)}{6}h^3 \tag{3.5}$$

$$f(x_0 - h) \approx f(x_0) - f'(x_0)h + \frac{f''(x_0)}{2}h^2 - \frac{f'''(x_0)}{6}h^3 \tag{3.6}$$

Summiert man diese beiden Gleichungen auf, so streichen sich die Terme der ersten und dritten Ableitung weg. Stellt man die daraus resultierende Gleichung dann nach $f''(x_0)$ um, so ergibt sich ein Ausdruck für die zweite Ableitung einer Funktion:

$$f''(x_0) \approx \frac{f(x_0 - h) - 2f(x_0) + f(x_0 + h)}{h^2} \tag{3.7}$$

So lässt sich beispielsweise die Poisson-Gleichung auf eine diskrete Weise ausdrücken:

$$\Delta\Phi = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right)\Phi = -\frac{\rho}{\epsilon_0} \tag{3.8}$$

$$\Phi_{i-1,j,k} + \Phi_{i+1,j,k} + \Phi_{i,j-1,k} + \Phi_{i,j+1,k} + \Phi_{i,j,k-1} + \Phi_{i,j,k+1} - 6\Phi_{i,j,k} = -h^2\frac{\rho_{i,j,k}}{\epsilon_0} \tag{3.9}$$

Wie aus der Gleichung jedoch ersichtlich ist, hängen die Potentiale der unterschiedlichen Knoten voneinander ab. Deshalb müssen die Gleichungen als gemeinsames Set gelöst werden. Um diese Aufgabe für Knotenanzahlen von bis zu 10^7 in dreidimensionalen Simulationen zu lösen, müssen extrem effiziente Verfahren angewandt werden, die den Rahmen dieser Arbeit übersteigen. Es sei aber angemerkt, dass in IBSimu eine sogenannte **Proconditioned Biconjugate Gradient Stabalized Method** (BiCGSTAB) zur Lösung des elektrischen Potentials genutzt wird.

Da das elektrische Feld separat für jeden Knoten errechnet wird, ist die Lösung deutlich einfacher. Die Diskretisierung von $\boldsymbol{E} = \nabla\Phi$ wird analog zu der von Gleichung 3.7 durchgeführt. Auch für die Berechnung der Trajektorien der Ionen muss lediglich ein Set aus zwei gewöhnlichen Differentialgleichungen gelöst werden.

$$\frac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}t} = \boldsymbol{v} \tag{3.10}$$

$$\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \frac{q}{m}(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}) \tag{3.11}$$

Dazu kann das häufig verwendete Runge-Kutta Verfahren genutzt werden, welches

auch auf der Approximierung durch eine Taylor-Reihe beruht.

3.3 Aufbau der Simulation

Nachdem die IBSimu Bibliothek erfolgreich eingebunden wurde, kann der volle Umfang dieser verwendet werden. Dazu muss eine C++-Datei geschrieben werden, in der die Simulation definiert wird. Innerhalb dieses Codes kann auf die Funktionalitäten der Bibliothek zurückgegriffen werden, sodass eine kompakte Datei entsteht. Zusammen mit dem entsprechenden *Makefile* kann der Code anschließend compiliert werden, sodass ein ausführbares Programm vorliegt. Dabei ermöglicht IBSimu eine Trennung von Simulation und Analyse. Dies ist möglich, da die errechneten Simulationsdaten der Felder und Trajektorien abgespeichert und in einem separaten Programm wieder eingelesen werden können. Sinnvoll ist das gerade bei komplexen dreidimensionalen Simulationen, deren Daten vielseitig analysiert werden sollen. In dieser Arbeit wird die Trennung von Simulation und Analyse allerdings nicht genutzt, da der Rechenaufwand hier überschaubar ist und nur eine Analyse der Strahldivergenz durchgeführt werden muss.

Um eine Doppelung zu vermeiden, wird die Funktionsweise von IBSimu-Code direkt anhand der Gittersimulation für dieses Studienprojekt erklärt. Dabei wird aus Gründen der Verständlichkeit nicht auf jede einzelne Zeile eingegangen und teils innerhalb des Codes gesprungen. Deshalb ist für einen ganzheitlichen Überblick der vollständige kommentierte Programmcode im Anhang zu finden.

Im Folgenden wird also die Funktion zur Simulation der Gitteranordnung erläutert, der einzig die Anzahl an Iterationszyklen übergeben wird.

```
1 void simu(int cycles){
```

In dieser wird zunächst ein Objekt der Klasse *Geometry* kreiert, welches alle Informationen über die Geometrie des Simulationsbereichs vereint. Zur Übersichtlichkeit des Programmcodes ist die Definition der Geometrie in eine eigene Funktion ausgelagert, welche das *Geometry*-Objekt bei Aufruf als Rückgabewert liefert. In dieser Funktion werden zunächst einige entscheidende Parameter festgelegt. Ganz elementar ist hierbei die Wahl zwischen einer Simulation in einer, zwei oder drei Dimensionen. Auch eine Simulation in Zylinderkoordinaten ist möglich.

Zur Bewertung der Qualität der Extraktion wird beispielhaft die Ionenoptik einer einzelnen Gitteröffnung untersucht. Da jedes Loch des Gitters eine Zylindersymme-

trie aufweist, lässt sich das Problem in zwei Dimensionen lösen. Denn die Geometrie, das elektrische Feld und damit die Trajektorie der Teilchen, sind nur vom Abstand vom Zentrum des Lochs abhängig. Demnach wird als erster Parameter *MODE_CYL* gewählt.

```
Geometry geom( MODE_CYL,
```

Anschließend wird das Simulations-Mesh definiert, also das Gitter, welches eine vorgegebene Anzahl an Knoten enthält. Für jeden dieser Knoten wird später die Poisson-Gleichung gelöst. Folglich gibt das Mesh die Größe des Simulationsbereichs vor und ist deshalb direkt von der simulierten Geometrie abhängig. Der Code ist so geschrieben, dass der Anwender oben im Code die Breite (width), Höhe (height) und den Knotenabstand (nodeSpacing) jeweils in Millimetern angeben kann. Abhängig davon wird die Knotenanzahl in x- und y-Richtung gesetzt. In z-Richtung wird nur ein Knoten gesetzt, da es sich um eine zweidimensionale Simulation handelt.

Als weitere Parameter werden noch der Ursprung gesetzt und der Knotenabstand in Metern übergeben.

```
Geometry geom( MODE_CYL,
   Int3D(round(width/nodeSpacing) + 1, round(height/nodeSpacing) +
   1, 1),
   Vec3D(0,0,0),
   nodeSpacing * mm );
```

Nun werden die drei Gitter als Elektroden in die Geometrie integriert. Dazu wird jede Elektrode mithilfe des *FuncSolid* Konstruktors erstellt und als Objekt der Klasse *Solid* gespeichert. Dieses kann nun dem oben erstellten *geom* Objekt hinzugefügt werden. *FuncSolid* bekommt dafür eine Funktion übergeben, welche jeder möglichen Kombination aus x-, y- und z-Wert einen bool'schen Wahrheitswert zuordnet. Das Abschirmgitter beginnt beispielsweise in x-Richtung nach der Plasmarandschicht und ist eine *gridWidth* breit. In y-Richtung beginnt das Gitter, nachdem ein Spalt von der Größe des Lochradius ausgespart wurde. Diese einfachen bool'schen Ausdrücke sind für alle drei Gitter ins separaten Funktion ausgelagert.

```
Solid *s1 = new FuncSolid( screenGrid );
geom.set_solid( 7, s1 );
Solid *s2 = new FuncSolid( accelerationGrid );
geom.set_solid( 8, s2 );
Solid *s3 = new FuncSolid( decelerationGrid );
geom.set_solid( 9, s3 );
```

Abschließend wird jeder *boundary* ein Potential und eine Grenzbedingung zugeordnet. Eine *boundary* bezeichnet dabei die sechs Grenzen des Simulationsbereichs (1-6) und die drei hinzugefügten Elektroden bzw.Gitter (7-9). Bei den Grenzbedingungen kann zwischen einer *Dirichlet-* und einer *Neumann-Bedingung* gewählt werden. Bei Dirichlet werden die betreffenden Grenzknoten fest auf das vorgegebene Potential gesetzt, während bei Neumann gefordert wird, dass die Ableitung des Potentials an der Grenze gleich einem bestimmten Wert ist. Exemplarisch werden die Grenzen gezeigt, bei denen das Potential ungleich Null gesetzt wird, also für x_{min}, das Abschirm- und das Beschleunigungsgitter. Damit ist die Definition der Geometrie abgeschlossen.

```
geom.set_boundary( 1, Bound(BOUND_DIRICHLET,
    sGridVoltage + electronTemp * tempToPotential) );
geom.set_boundary( 7, Bound(BOUND_DIRICHLET,   sGridVoltage) );
geom.set_boundary( 8, Bound(BOUND_DIRICHLET,   aGridVoltage) );
```

Zurück in der *simu*-Funktion werden, basierend auf den oben definierten Knoten des Simulationsgitters, Felder für das Potential, die Ladung, das elektrische Feld und das magnetische Feld kreiert. Letzteres ist notwendig, falls die Anordnung von einem externen magnetischen Feld durchsetzt wird. Es muss allerdings in jedem Fall deklariert sein, sodass es im feldfreien Fall auf Null belassen wird. Da das elektrische und magnetische Feld nicht gekoppelt sind, hat das unbenutzte Feld keinen Einfluss auf die Geschwindigkeit der Simulation.

```
EpotField epot( geom );
MeshScalarField scharge( geom );
MeshVectorField bfield;
EpotEfield efield( epot );
```

Anschließend werden Randbedingungen für das elektrische Feld gesetzt. Dafür wird ein Array mit 6 Elementen definiert, welches abwechselnd erst die Bedingung für das Minimum und dann die für das Maximum für alle drei Raumrichtungen festlegt (also x_{min}, x_{max}, y_{min} etc.). Über alle Simulationsränder hinaus soll das Feld mithilfe der letzten drei Potentialwerte extrapoliert werden. Einzig an der y_{min} Grenze soll das Feld einem symmetrischen Potential entsprechend gespiegelt werden, da wir ein rotationssymmetrisches System betrachten. Das bedeutet, dass das elektrische Feld an der Grenze selbst Null ist und darüber hinaus antisymmetrisch gespiegelt wird (also $E(-y) = -E(y)$).

```
field_extrpl_e efldextrpl[6] = { FIELD_EXTRAPOLATE,
   FIELD_EXTRAPOLATE, FIELD_SYMMETRIC_POTENTIAL, FIELD_EXTRAPOLATE,
```

```
    FIELD_EXTRAPOLATE, FIELD_EXTRAPOLATE };
efield.set_extrapolation( efldextrpl );
```

Nachdem die in Kapitel 3.2 erwähnte Methode zur Lösung des Potentialfeldes ausgewählt wird,

```
EpotBiCGSTABSolver solver( geom );
```

wird das Plasma initialisiert. Dazu wird eine Methode auf den soeben definierten *solver* angewandt, die diesem das Plasmapotential und das initiale Plasmavolumen übergibt. Dazu wird ein Grenzwert auf einer der drei Koordinatenachsen (hier der x-Achse) gesetzt, bis zu welchem sich das Plasma initial ausdehnt. Dieses initiale Plasma ist noch linear verteilt, also quasineutral im vorausgesetzten Volumen. Nachdem später das erste Mal die Ionentrajektorien berechnet wurden, wird das Plasmamodell zu dem in Kapitel 2.2 beschriebenen geändert.

```
InitialPlasma initp( AXIS_X, pOffset * mm);
solver.set_initial_plasma(sGridVoltage + plasmaPotential, &initp);
```

Als nächstes wird die Particle Data Base für Zylinderkoordinaten initialisiert, welche die Ionen in den Simulationsbereich einbringt und deren Trajektorien berechnet. Anschließend wird auch hier die Zylindersymmetrie berücksichtigt und eine Spiegelung aller Trajektorien an der y_{min} boundary gefordert.

```
ParticleDataBaseCyl pdb( geom );
bool pmirror[6] = { false, false, true, false, false, false };
pdb.set_mirror( pmirror );
```

Damit sind alle für die Simulation notwendigen Klassen eingebunden, sodass der in Abbildung 3.1 visualisierte Simulationsablauf beginnen kann. Es wird also eine for-Schleife geöffnet, die für eine vorgegebene Anzahl an Zyklen durchlaufen wird. Sie wird also nicht nach Erreichen einer bestimmten Konvergenz-Bedingung abgebrochen. Das liegt daran, dass eine Simulation mit IBSimu aufgrund ihrer diskreten nummerischen Natur zu keiner eindeutigen Lösung führen wird. Beispielsweise können die Trajektorien der Teilchen beginnen zu oszillieren, wenn die Teilchen vermehrt über einige Knoten fliegen. Denn dann deponieren sie dort eine hohe Raumladung, die sie beim nächsten Schritt von diesen Knoten ablenkt. Dadurch wird die Raumladung dort wieder sinken, wodurch im nächsten Schritt die entgegengesetzte Bewegung der Ionen hervorgerufen wird. Obwohl im Code dafür eine Relaxation der Raumladung implementiert ist, beobachtet man auch bei der hier durchgeführten Simulation ein nicht vollständig konvergentes Verhalten [7, S. 89f.]. Deshalb wird

eine Konvergenz der Simulation angenommen, sobald nur noch kleine Änderungen der Observablen auftreten. In einer Versuchsreihe mit repräsentativen Werten hat sich eine Iterationsanzahl von 10-15 Schritten als sichere Abschätzung bewiesen und gegenüber der Implementation eines Abbruchkriteriums durchgesetzt.

Da der generelle Ablauf der Simulation bereits in Kapitel 3.2 ausführlich dargestellt wurde, soll hier nur auf die Definition des Ionenstrahls über die **Particle Data Base** genauer eingegangen werden. Dafür werden die Ionen durch Masse, Ladung und mittlere kinetische Energie definiert. Letztere lässt sich über die Bohm-Geschwindigkeit errechnen, da der Ort an dem die Ionen eingebracht werden, der Grenze zwischen Vor- und Randschicht des Plasmas entspricht. Die Energie errechnet sich dann zu:

$$E = \frac{1}{2} M v_B^2 = \frac{1}{2} M \frac{k_B T_e}{M} = \frac{k_B T_e}{2}. \tag{3.12}$$

Dabei ist $k_B T_e$ als die Elektronentemperatur in eV definiert. Außerdem wird die Richtung des Strahls durch einen Vektor definiert, zu dem die Ionen im 90° Winkel austreten und diesem eine Stromdichte zugeordnet. Letztlich kann auch die Anzahl an simulierten Partikeln angegeben werden. Dabei ist es wichtig anzumerken, dass es sich bei den Partikeln nicht um einzelne Ionen, sondern um virtuelle Teilchen handelt. Auf diese wirkt eine Kraft entsprechend der Ladung und Masse, jedoch trägt jedes dieser Partikel einen bestimmten Anteil des Strahlstromes. Entsprechend dieses Anteils wird nach der Berechnung auch die Raumladung auf die Knoten verteilt.

```
1 pdb.add_2d_beam_with_energy(numberOfParticles, currentDensity,
2             charge, mass, electronTemp / 2,
3             0.0, 0.0, 0.0, 0.0, 0.0, height * mm);
```

Ein anderer wichtiger Punkt innerhalb der Iteration, ist der Wechsel des Plasmamodells nach dem ersten Schritt. Denn bei der Initialisierung des Plasmas wurde das Plasma zunächst quasineutral im gesamten Volumen verteilt. Nachdem die Ionen einmal durch die Geometrie geflogen sind, kann allerdings zu dem oben erläuterten Plasmamodell der Randschicht gewechselt werden. Dafür müssen dem *solver* die Gesamtladung des Strahls, die Elektronentemperatur und das Potential übergeben werden. Für das Potential wird zur Spannung des Abschirmgitters noch das Floatingpotential, welches in Kapitel 2.2 hergeleitet wurde, addiert. Der Faktor der zwischen Potential und Elektronentemperatur steht, ist konstant und errechnet sich zu

$$|\Phi_F| = \frac{k_B T_e}{e} \ln \sqrt{\frac{M}{2\pi m}} = \frac{k_B T_e}{e} \cdot 4.22. \tag{3.13}$$

```
if( i == 1 ) {
    double rhoe = pdb.get_rhosum();
    solver.set_pexp_plasma( rhoe, electronTemp,
    sGridVoltage + electronTemp * tempToPotential );}
```

Nach dem kompletten Durchlauf dieser for-Schleife sind die Felder und Trajektorien für die gegebene Gitteranordnung vollständig bestimmt. Nun kann Strahldiagnostik durchgeführt werden und ein Plot des Simulationsbereichs angefertigt werden. In Abbildung 3.2 ist ein solcher Plot für eine beliebige Parameterkombination zu sehen.

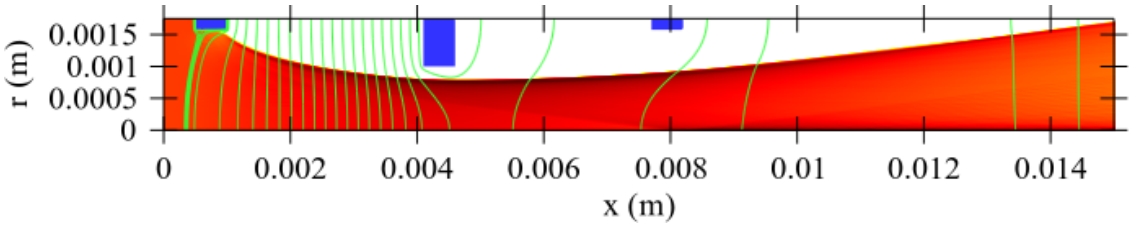


Abbildung 3.2: Abbildung der Ionenextraktion für $T_e = 7.68$ eV und einen Gitterabstand von 3.1 mm.

4 Definition der Parameterstudie

Die im vorangegangenen Kapitel aufgebaute Gittersimulation soll nun für unterschiedliche Gitterabstände variiert werden. Für jeden Abstand soll dann die Güte der Fokussierung mithilfe der Emittanz und dem Divergenzwinkel bestimmt werden, um den bestmöglichen Gitterabstand zu finden. Da die Ionenquelle bei unterschiedlichen Arbeitspunkten des Plasmas betrieben wird, soll der ideale Gitterabstand auch für veränderte Elektronentemperaturen ermittelt werden. Zum Schluss sollen die Ergebnisse für die unterschiedlichen Arbeitspunkte verglichen und eine endgültige Wahl für den besten Gitterabstand getroffen werden.

4.1 Wahl der Simulationsparameter

Wie oben bereits ausgeführt wurde, ist die korrekte Wahl der Eingabeparameter entscheidend für die Güte der Simulation. Deshalb sollen hier einmal alle Annahmen basierend auf der in Kapitel 2 eingeführten Theorie zum Plasma aufgeschlüsselt werden.

Zunächst werden die Parameter erläutert, welche die geometrische Anordnung bestimmen. Da die drei Gitter bereits gefertigt wurden, sind die Stegbreiten, die Radien der Löcher und die Breite der Gitter bereits festgesetzt. Die Höhe des Simulationsbereichs ergibt sich dann aus der Summe von Lochradius und halber Stegbreite. Die Breite wird hingegen so gewählt, dass der Ionenstrahl nach Austritt aus der Gitteranordnung in ausreichender Länge simuliert wird. Nicht festgelegt durch die gefertigten Gitter ist ihr Abstand, weshalb dieser in der Simulation variiert wird.

Anschließend muss noch der Plasma-Offset, also der Länge zwischen der linken Simulationsgrenze und dem Beginn des Abschirmgitters. Wie in Kapitel 2.2 erwähnt wurde, liegt die Größenordnung der Plamarandschicht im Bereich einiger Debye-Längen; also nur wenigen μm. Da sich durch das Loch allerdings der einleitend beschriebene Plasma-Menisukus ausbildet, muss der Offset genügend Raum bieten, um diesen einzuschließen. Gleichzeitig sollte er nicht zu groß gewählt werden, um die Konvergenz der Simulation nicht zu beeinträchtigen. Der Abstand wird also so gewählt, dass der Meniskus gerade ausreichend Platz zur Ausbildung hat.

Tabelle 4.1: Geometrische Eingabeparameter, die durch das produzierte Gitter vorgegeben sind.

Parameter	**Variablenname**	**Größe in mm**
Höhe des Simulationsbereichs	height	15
Breite des Simulationsbereichs	width	1.75
Radius des Abschirm- und Abbremsgitters	sdGridRadius	1.575
Radius der Beschleunigungsgitters	aGridRadius	1.0
Gitterbreite	gridWidth	0.5
Gitterabstand	gridSpacing	variabel
Plasma Offset	pOffset	0.5

Nun müssen also noch die für die Definition des Ionenstrahls notwendigen Parameter festgelegt werden. Da das RAM-EP Projekt die Nutzung von Restatmosphäre als Treibstoff demonstrieren soll, werden sowohl atomare Stickstoff-, als auch Sauerstoffionen separat voneinander simuliert. Für Ladung und Masse werden also die entsprechenden Werte eingesetzt. Die Anzahl an Simulationspartikel wird so gewählt, dass sich eine gute *Abtastrate* bei keiner zu großen Rechenleistung ergibt. Die mittlere Energie der Ionen ist, wie in Gleichung 3.12 gezeigt wurde, nur von der Elektronentemperatur abhängig.

Zur Bestimmung der Stromdichte wird ein rückwärts gerechneter Ansatz verwendet. Es ist bekannt, welcher Strom I für das gesamte Triebwerk erreicht werden soll. Dieser teilt sich auf die N Löcher des Extraktionsgitters auf, weshalb durch jedes der Löcher ein Strom von $I_L = I \ / \ N$ fließen muss. Durch Angabe des Gesamtstroms I und der Anzahl der Löcher N lässt sich demnach die Stromdichte für einen Kanal berechnen, dessen Abschirmgitter den Radius r_S besitzt:

$$j = \frac{I}{r_S^2 \pi N} \tag{4.1}$$

Entscheidend für die Extraktion ist auch das Potential der Elektroden, wobei für das Abbremsgitter eine Spannung von 0 V angenommen wird. Die Spannungen der beiden anderen Gitter sind in Tabelle 4.2 aufgeführt.

Tabelle 4.2: Aufstellung der weiteren Eingabeparameter für die Definition des Strahls und die Spannung der Gitter.

Parameter	**Variablenname**	**Wert mit Einheit**
Virtuelle Teilchen	numberOfParticles	1000
Ladung der Ionen	charge	1.0 e
Masse der Ionen	mass	14/16 u
Gesamtstrom	totalCurrent	0.3 A
Anzahl der Löcher	numberOfHoles	499
Spannung des Abschirmgitters	sGridVoltage	2.0 kV
Spannung des Beschleunigungsgitters	aGridVoltage	-200 V

Die Simulation ist durch die in den beiden Tabellen aufgeführten Größen vollständig definiert. Aus diesen lässt sich auch der Potentialabfall innerhalb der Randschicht des Plasmas entsprechend Gleichung 3.13 berechnen. Das Potential muss sich in Abhängigkeit vom Arbeitspunkt in dem das Plasma betrieben wird ändern, um einen konstanten Strom zu gewährleisten. Der Zusammenhang erklärt sich, wenn man die Stromdichte der Ionen an der Grenze zwischen Vor- und Randschicht betrachtet. Nimmt man vereinfacht an, dass die Ionen und Elektronendichte an der Grenze identisch und gleich der im Inneren des Plasmas sind, so ergibt sich mit Gleichung 2.9:

$$j_i = e n_e v_B \propto n_e \sqrt{T_e} \tag{4.2}$$

Wird nun der Massenfluss gesenkt, so sinkt die Anzahl an ionisierbaren Atomen und entsprechend die Elektronendichte. Um dennoch denselben Strom extrahieren zu können, muss die Elektronentemperatur zwangsläufig steigen. Da diese Betrachtung nur eine Abschätzung der Abhängigkeit darstellt, wurden die Arbeitspunkte des Plasmas in einem Programm simuliert und der Graph in Abbildung 4.1 erstellt. Aus diesen Simulationsdaten ergeben sich nun verschiedene Elektronentemperaturen für die jeweils der bestmögliche Gitterabstand ermittelt werden soll.

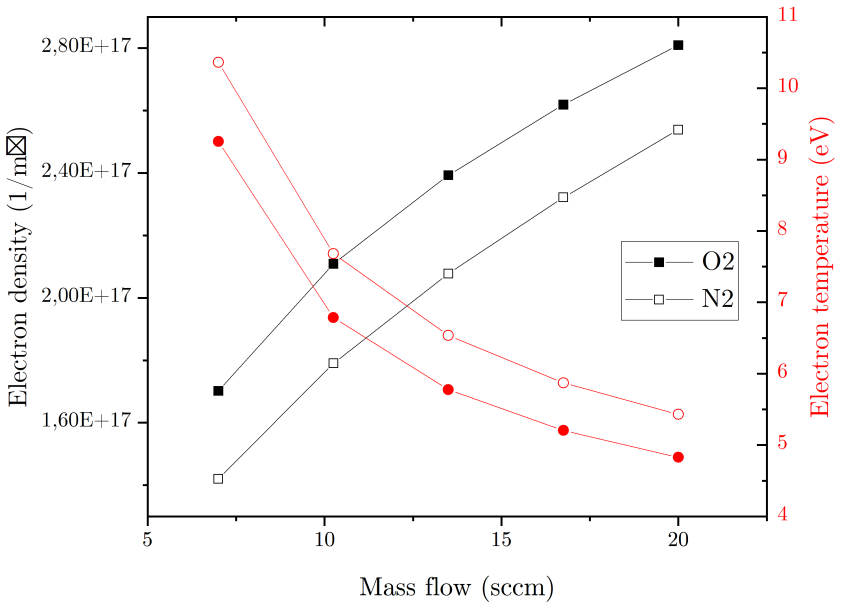


Abbildung 4.1: Dieser Graph stellt die Abhängigkeit der Elektronentemperatur und -dichte von dem Massenfluss durch das Triebwerk, bei konstantem Strahlstrom, dar. Wie erwartet, steigt die Elektronendichte zusammen mit dem Massenfluss linear, während die Elektronentemperatur entsprechend abgesenkt wird, um einen konstanten Strahlstrom beizubehalten.

Beim Aufbau der Simulation fällt auf, dass keine der Größen von der Elektronendichte abhängt. Das liegt daran, dass der Ionenstrahl nicht inhärent durch die Eigenschaften des Plasmas bestimmt wird, sondern - wie oben erklärt - über den Strahlstrom rückwärts festgesetzt wird. Diese Variante wurde gewählt, da der Strahlstrom bei den errechneten Arbeitspunkten als Konstante festgelegt wurde. Dennoch ergibt sich dadurch eine schwammige Definition der Stromdichte, da davon ausgegangen wird, dass der extrahiere Strahl rechtwinklig aus dem definierten Ionenstrahl ausgeschnitten wird. In Realität können aber je nach Potentialverlauf auch mehr oder weniger Ionen eingesammelt werden, wodurch der Strom am Gitterausgang leicht variiert. Allerdings ist diese Variation des Stroms nur vergleichsweise klein, weshalb an diesem Weg zur Bestimmung der Stromdichte festgehalten wurde.

4.2 Physikalische Grenzen für den Gitterabstand

Um zu gewährleisten, dass der ideale Gitterabstand innerhalb physikalisch möglicher Grenzen liegt, müssen der minimale und maximale Gitterabstand definiert werden, um so ein Intervall für die Variation des Parameters zu finden.

Der minimale Gitterabstand wird grundlegend durch die Durchschlagspannung zwischen Abschirm- und Beschleunigungsgitter festgelegt. Denn sobald ein Durchschlag erfolgt kann Strom von einem Gitter zum anderen fließen und die beschleunigende Potentialdifferenz bricht ein. Die maximale Spannung ist allerdings eine schwer zu bestimmende Größe, da es sich bei der Ionenextraktion um ein dynamisches System handelt. Zwischen den Platten befindet sich entsprechend kein Vakuum, sondern sowohl ausströmendes Neutralgas, als auch der Ionenstrahl. Typischerweise wird das maximale elektrische Feld zwischen den beiden Gittern so gewählt, dass es weniger als der Hälfte des Feldes für einen Vakuumdurchschlag entspricht. In der Literatur findet man, dass das theoretisch festgesetzte Maximum in der Praxis nicht erreicht wird. Als Abschätzung bietet sich ein Wert für Xenonatome aus [2, S. 200] an. Bei einer Spannung zwischen den Gittern von 2 kV und einem Gitterabstand von 2mm wäre eine maximale Ionenstromdichte von 150 $\mathrm{A/m^2}$ theoretisch möglich. In dem hier betrachteten Triebwerk liegt hingegen eine Dichte von etwa 77 $\mathrm{A/m^2}$, also etwa die Hälfte vom gegebenen Maximum, vor. Da anfängliche Tests überdies gezeigt haben, dass die Ionenoptik für Gitterabstände von weniger als 2 mm nur äußerst schlechte Strahlengänge aufweist, wurde sich entschieden den minimalen Gitterabstand bei der Variation auf **2 mm** festzusetzen.

Zur Berechnung des maximal möglichen Gitterabstands zwischen Abschirm- und Beschleunigungsgitter wird das Child-Langmuir-Raumladungsgesetz verwendet, welches in Kapitel 2.3 eingeführt wurde. Dazu wird die Formel nach dem Gitterabstand d umgestellt.

$$d = \frac{2}{3} U^{\frac{3}{4}} \sqrt{\frac{\epsilon_0}{j}} \sqrt{\frac{2e}{M}} \tag{4.3}$$

Setzt man in diese nun die Stromdichte und die Spannung zwischen den beiden Gittern ein, so ergibt sich ein maximaler Gitterabstand von etwa **4.3 mm**. Dieser Wert legt also den maximalen Abstand für die Parametervariation fest. Zur Berechnung des Wertes wurde der Abstand der ersten beiden Gitter untersucht, da das verwendete Gesetz nur für diesen Fall gilt. Bei seiner Herleitung wurde nämlich angenommen, dass die Ionen ohne Geschwindigkeit starten und in dem Feld beschleunigt werden. Zwischen den beiden hinteren Gittern liegt hingegen eine abbremsende Spannung gegenüber den Ionen mit einer initialen Geschwindigkeit an. Hier ergibt sich also gar kein Ausdruck für eine maximale Stromdichte, weil die Raumladung nur das abbremsende Feld abschwächt.

4.3 Durchführung und Diagnostik

Nun kann die Funktion zur Simulation einer gegebenen Anordnung (Kapitel 3.3) mithilfe der physikalisch begründeten Eingabeparameter (Kapitel 4.1) für die verschiedenen Gitterabstände und Arbeitspunkte (Kapitel 4.2) ausgeführt werden. Dank der allgemeinen Definition der Simulationsfunktion muss dafür lediglich über den zu variierenden Parameter iteriert werden.

Entscheidend ist nun, welche Messwerte aus den Simulationsdaten ermittelt werden. Dafür muss überlegt werden, auf welche Größe der Gitterabstand optimiert werden soll. Im Endeffekt soll die Ionenoptik so ausgelegt werden, dass aus den extrahierten Ionen der größtmögliche Schub gewonnen werden kann. Dazu sollten die Ionen möglichst genau in Richtung des Schubs, also orthogonal zum Gittersystem, ausgestoßen werden. Der Anteil des Impulses, der parallel zum Gitter wirkt wird sich aufgrund der Radialsymmetrie aufheben und dadurch die Effizienz des Triebwerks absenken. Es muss also ein Maß gewählt werden, welches die Fokussierung des Strahls bemisst.

Dazu bietet sich einerseits die Emittanz des Strahls an. Wie in Kapitel 2.4 dargelegt, kann mithilfe der Emittanz eine Aussage über die Uniformität des Strahls getroffen werden. Die RMS Emittanz ϵ_{RMS} gibt dabei eine Art Streumaß für die Strahlverteilung an. Wie oben allerdings bereits erwähnt wurde, ist die Emittanz für eine größere Geschwindigkeit in Strahlrichtung automatisch kleiner. Deshalb ist sie keine gute Vergleichsgröße zwischen den unterschiedlichen Arbeitspunkten, da die Ionen je nach Arbeitspunkt das Triebwerk mit unterschiedlichen Geschwindigkeiten verlassen.

Andererseits kann innerhalb der Simulation direkt auf die Daten der Ionentrajektorien zurückgegriffen werden. Daraus lässt sich auch direkt der Divergenzwinkel des Strahls berechnen. Dabei ist es essentiell, wie die Winkel der verschiedenen Trajektorien gemittelt werden. Hier wurde sich dazu entschieden, den Quotienten v_r/v_x über die Stromdichte der jeweiligen Trajektorie zu gewichten und anschließend zu mitteln. Dadurch berücksichtigt die gewählte Mittlung, dass eine Trajektorie, die weiter von der Strahlachse entfernt liegt, ein größeres Gewicht hat. Denn diese liegt bei der Rücktransformation in Zylinderkoordinaten auf einem Kreis mit einem größeren Radius.

Die IBSimu Bibliothek bietet die Möglichkeit der Strahldiagnostik über Auslesen der simulierten Daten aus der Particle Data Base. Die daraus gewonnenen Da-

ten der Trajektorien können einer Methode der Klasse **Emittance** übergeben werden, welche daraus die RMS Emittanz nach der oben aufgeführten Formel errechnet. Bei der Berechnung wird auch das gewichtete Mittel von r' gebildet, welches die Klasse ebenfalls ausgeben kann. Die beiden Werte für Emittanz und Winkel ($r' = tan(\Theta/2)$)werden also für jede Parameterkombination berechnet und in einem Array gespeichert. Aus diesem werden die Werte am Ende der Simulation in einer .txt Datei abgespeichert.

Zusätzlich zu Emittanz und Divergenzwinkel, werden auch die Anzahl der Kollisionen mit den Simulationsgrenzen und den Elektroden in einem Textdokument abgespeichert. Gitterabstände, bei denen es zu direkten Kollisionen mit dem Beschleunigungsgitter kommt, sind aus Gründen der Effizienz und Abnutzung nicht sinnvoll. Da diese Bedingung für jeden Arbeitspunkt gelten muss, wird der kleinste Gitterabstand herausgesucht, bei dem es bei einem der Arbeitspunkte noch zu einer Kollision kommt. Dieser und alle größeren Abstände werden anschließend aus den Plots zur Auswertung im folgenden Kapitel entfernt, da sie durch diese Kollisionsbedingung bereits ausgeschlossen werden. Der minimale Gitterabstand mit Kollision wurde für Stickstoff zu 3.7 mm und für Sauerstoff zu 3.6 mm bestimmt.

Zur Analyse der Daten wird ein Python Code genutzt, der im Anhang zu finden ist. Mit ihm werden die generierten Textdateien in einen Dataframe eingelesen und daraufhin als Heatmap geplottet. Die Heatmap hat den Gitterabstand auf der x- und die verschiedenen Elektronentemperaturen auf der y-Achse. Die Colormap stellt dann für jede Kombination aus Abstand und Temperatur den Wert der Emittanz, bzw. des Divergenzwinkels dar. Diese Plots werden im folgenden Kapitel zur Auswertung und Ermittlung der Ergebnisse verwendet.

5 Auswertung

5.1 Ergebnisse

In diesem Kapitel wird die Auswertung der Simulationsergebnisse anhand übersichtlicher Plots dargestellt. Dabei werden die Ergebnisse anhand der Graphen für Stickstoff erläutert. Die Werte für die Simulation mit Sauerstoff werden analog ermittelt und deshalb am Ende des Kapitels kurz zusammengefasst. Die entsprechenden Plots sind dem Anhang zu entnehmen.

Auf Grundlage der in den vorangegangen Kapiteln ausgeführten Begründung der Eingabeparameter wird davon ausgegangen, dass die Simulation physikalisch sinnvolle Ergebnisse liefert. Da der Autor auf den Aufbau des Lösungsverfahrens des Simulationsprogramms selbst keinen Einfluss hat, muss auf die Vorarbeit von [7] vertraut werden. Auf mögliche Schwachstellen der Simulation wird in der folgenden Diskussion eingegangen.

Sowohl die Emittanz, als auch der Divergenzwinkel werden an drei verschiedenen Ebenen des Simulationsbereichs untersucht. Zwei davon sind für alle Gitterabstände an bestimmten Positionen im Simulationsbereich fixiert, nämlich bei 10.5 mm und 12 mm. Die dritte Ebene ist hingegen auf einen Abstand von 2 mm hinter dem Ende des Abbremsgitters gesetzt und variiert somit mit der Variation des Gitterabstands. Die drei Ebenen liegen damit für alle Gitterabstände immer hinter dem Abbremsgitter und sind beispielhaft in Abbildung 5.1 visualisiert.

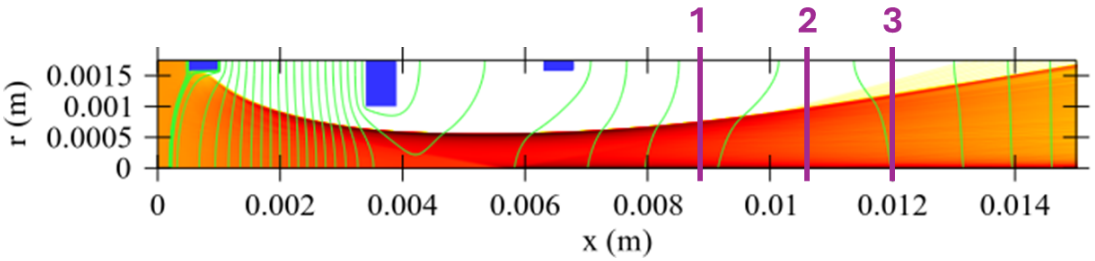


Abbildung 5.1: Auf dieser Abbildung sind für T_e=7.68 eV und d=2.4 mm die drei Ebenen eingezeichnet, an denen die Daten zur Berechnung von Emittanz und Divergenzwinkel erhoben werden. Dabei ist **1** die variable Ebene, **2** die bei 10.5 mm und **3** die bei 12 mm

Da auf den Strahl nach dem Abbremsgitter, außer der Abstoßung durch die eigene Raumladung, keine weiteren Kräfte mehr wirken, ist davon auszugehen, dass die Emittanz für alle drei Ebenen jeweils für die gleiche Parameterkombination konstant ist. Diese Erwartung wird von der Auswertung bestätigt. Deshalb wird hier nur ein

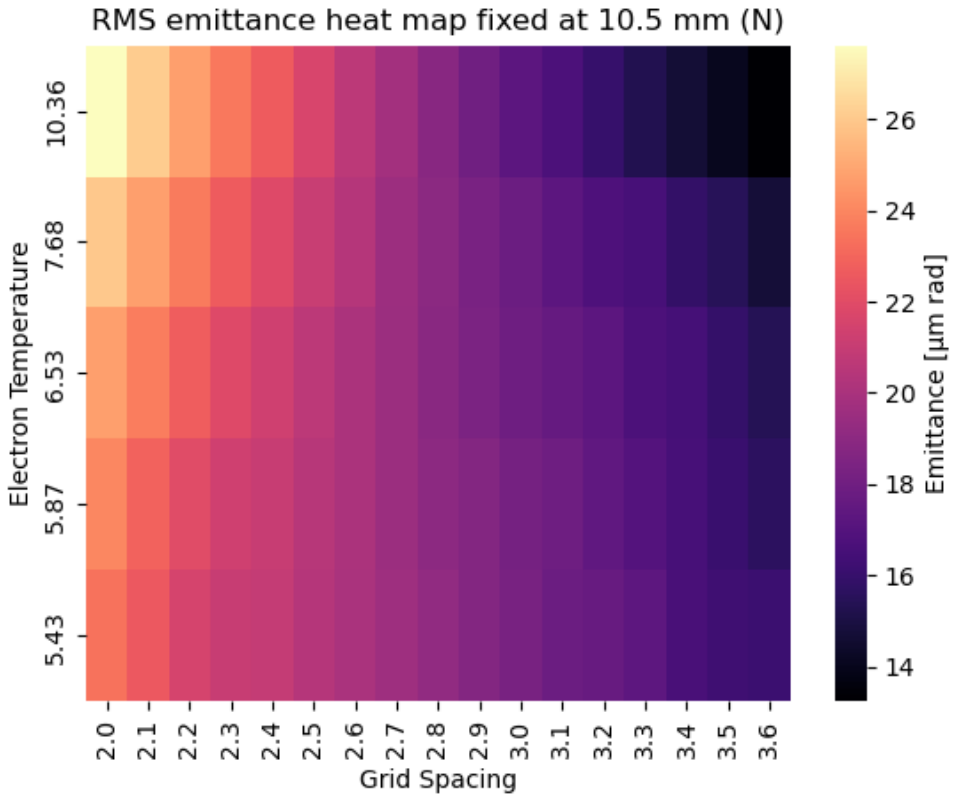
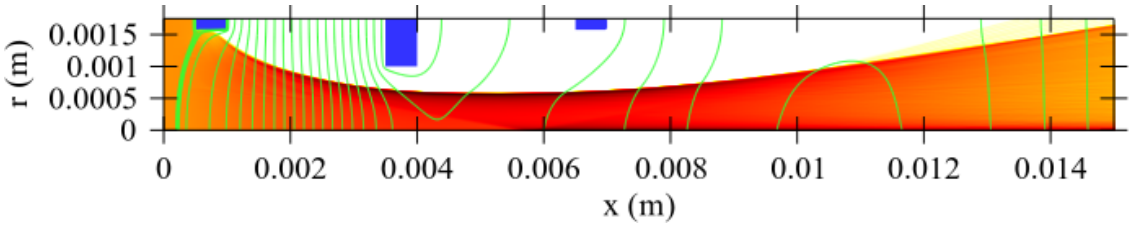


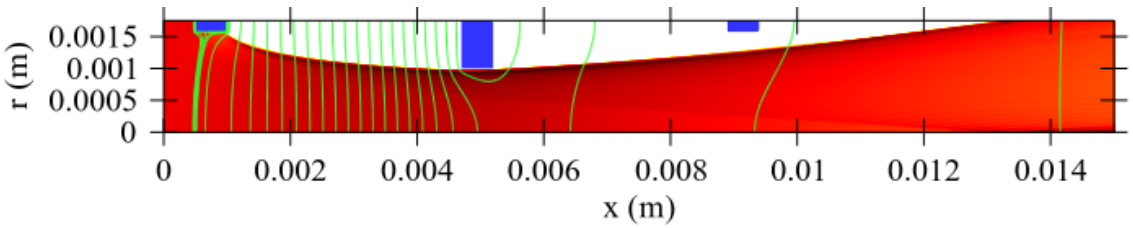
Abbildung 5.2: RMS Emittanz bei 10.5 mm für eine Variation der Elektronentemperatur und den Gitterabstand. Die Heatmap ordnet jeder Parameterkombination einen Emittanzwert zu. Es ist zu erkennen, dass dieser für höhere Temperaturen und größere Gitterabstände abnimmt.

Plot gezeigt, während die weiteren Graphen im Anhang zu finden sind.

Der Plot zeigt zwei Abhängigkeiten, auf welche nun genauer eingegangen wird. Einerseits zeigt der Graph, dass die Emittanz für größere Elektronentemperaturen abnimmt. Der Grund dafür wurde in den vorangegangenen Kapiteln bereits angedeutet. Da die Bohmgeschwindigkeit für größere Temperaturen zunimmt, haben die Ionen bei hohen Temperaturen eine größere Endgeschwindigkeit. Weil die Definition $r' = p_r/p_x$ gilt, nimmt r' für einen höheren Impuls nach Durchlaufen der Anordnung ab. Das bedeutet, dass der Emittanz Plot nur innerhalb einer Temperatur sinnvoll ausgewertet werden kann. Denn hier ist die zweite Abhängigkeit sichtbar, welche zeigt, dass die Emittanz für steigenden Gitterabstand abnimmt. Das liegt dran, dass der Plasmameniskus und alle weiteren Äquipotentiallinien zwischen Abschirm- und Beschleunigungsgitter für größere Gitterabstände immer gerader werden. Dadurch weisen die Ionentrajektorien keinen so großen Bauch auf und fliegen homogener durch die Anordnung (siehe Abbildung 5.3). Das Ergebnis, den Gitterabstand so zu



(a) Mit einem Gitterabstand von 2.5 mm



(b) Mit einem Gitterabstand von 3.7 mm

Abbildung 5.3: Vergleich der Ionentrajektorien für dieselbe Elektronentemperatur bei sehr verschiedenen Gitterabständen. Es ist zu erkennen, dass die Äquipotentiallinien für den größeren Gitterabstand deutlich gerader verlaufen und die Ionen deshalb homogener durch die Anordnung fliegen.

wählen, dass die Ionen gerade so an dem Beschleunigungsgitter vorbei fliegen, ist allerdings mit Vorsicht zu betrachten, da die Ionen in Realität eine Verteilung der Startgeschwindigkeit aufweisen und deshalb ein gewisser Sicherheitsabstand eingehalten werden sollte.

Für den gewichteten Divergenzwinkel sehen die Plots ganz anders aus und zeigen auch über die verschiedenen Temperaturen hinweg einen klaren Trend. Während sich für kleine und große Gitterabstände vergleichsweise große Divergenzwinkel ergeben, gibt es für die fixierten Auswertungsebenen im Bereich zwischen 3.1 mm und 3.4 mm für alle Temperaturen einen minimalen Winkel von etwa 8° bis 9.5°. Dabei liefert die Auswertung bei 10.5 mm etwas kleinere ideale Gitterabstände, als die Auswertung bei 12 mm. Dennoch geben sie gemeinsam einen klaren Rahmen für den idealen Gitterabstand vor.

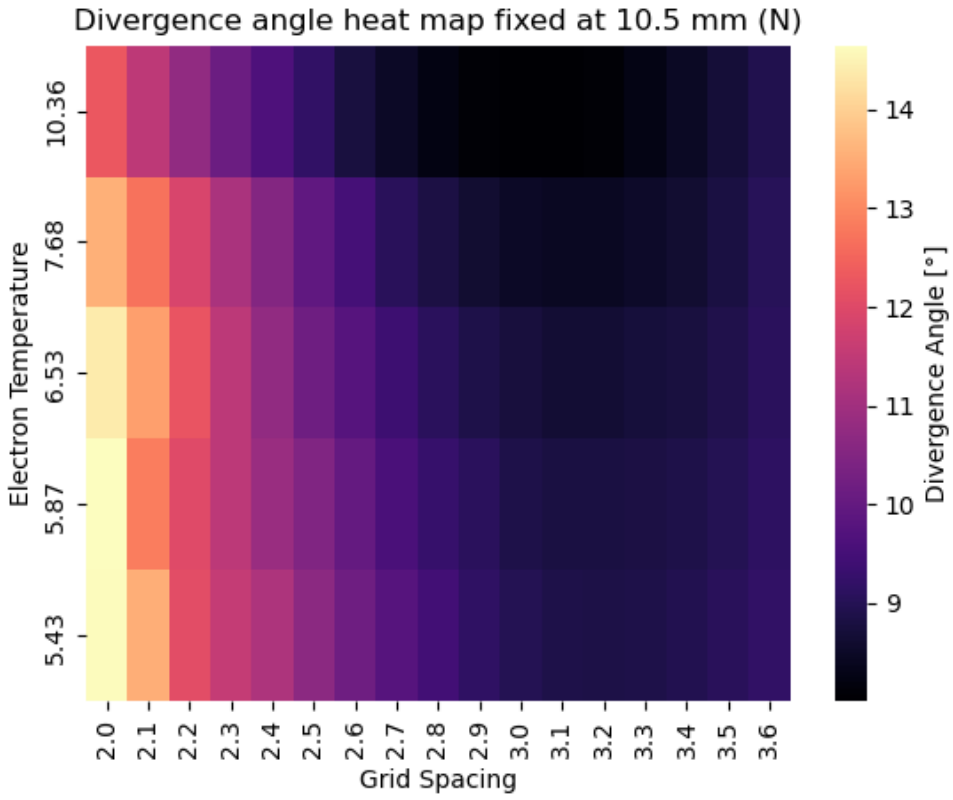


Abbildung 5.4: Darstellung des Divergenzwinkels bei 10.5 mm für eine Variation der Elektronentemperatur und den Gitterabstand. Die Heatmap ordnet jeder Parameterkombination einen Wert für den mittleren gewichteten Divergenzwinkel zu. Es ist zu erkennen, dass sich für einen Gitterabstand von 3.1 mm - 3.2 mm ein minimaler Divergenzwinkel für alle Elektronentemperaturen ergibt.

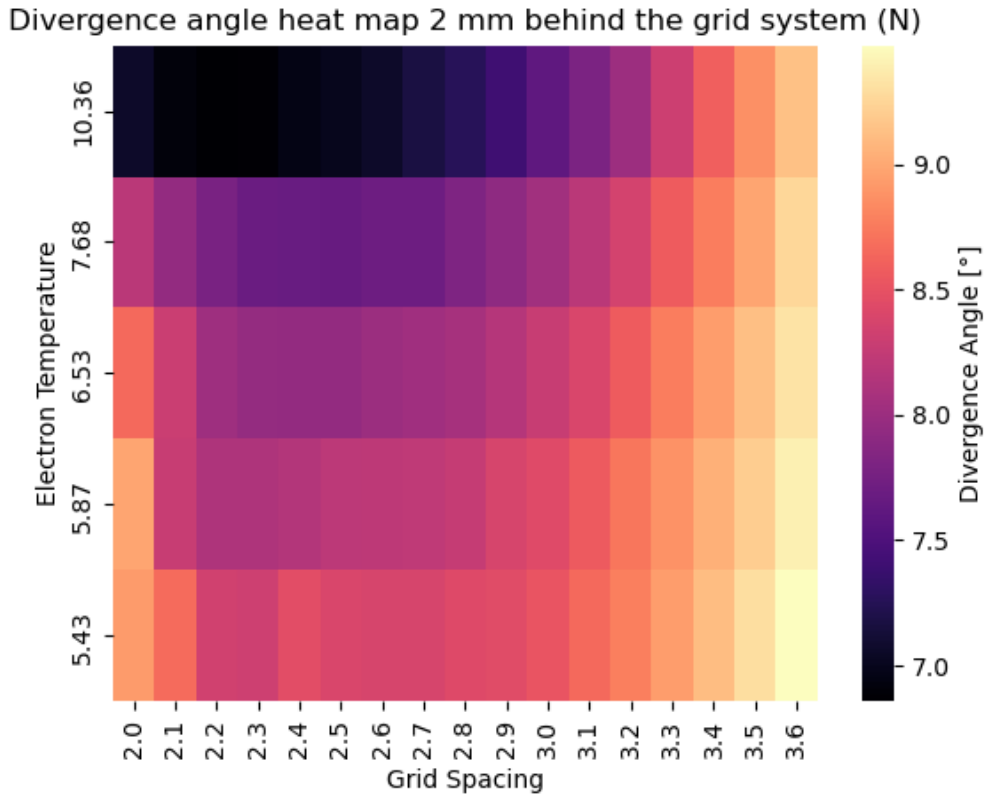


Abbildung 5.5: Plot des Divergenzwinkels analog zu den oben beschriebenen Darstellungen. Nun allerdings mit den Werten der variablen Auswertungsebene 2 mm hinter dem Ende des Gittersystems. Hier ergibt sich scheinbar ein widersprüchliches Ergebnis und ein idealer Gitterabstand zwischen 2.2 mm und 2.5 mm. Allerdings sind diese Werte durch die variable Lage der Auswertungsebene verzerrt, wie im Text genauer erläutert wird.

Ein Ausreißer ist hingegen die Auswertung der variabel gesetzten Ebene (siehe Abbildung 5.5). Diese hat ihr Minimum vielmehr bei einem Abstand von etwa 2.2 mm bis 2.5 mm. Auch ist der Divergenzwinkel hier deutlich niedriger und bewegt sich in einem Bereich von 7° bis 9°. Da die variable Auswertungsebene für Gitterabstände ab etwa 3.3 mm im Bereich der fixierten Ebenen liegt, liefert sie dort identische Ergebnisse. Es muss nun allerdings geklärt werden, warum die Ergebnisse bei kleineren Gitterabständen geringer ausfallen. Ein Blick auf Abbildung 5.3 zeigt, dass sich für alle Gitterabstände, im Bereich zwischen etwa 10 mm und 12 mm eine Quelle des elektrischen Feldes befindet. Die daraus resultierenden Äquipotentiallinien zwischen Strahl und Abbremsgitter bedeuten, dass auf die Ionen direkt nach dem Gitter eine Kraft wirkt, die sie weiter in transversaler Richtung beschleunigt. Da für kleine Gitterabstände die Auswertungsebene noch am Anfang dieses defokussierenden Bereichs liegt, ergibt sich dort ein scheinbar niedrigerer Divergenzwinkel. Folglich resultiert das Minimum für diesen Plot nicht aus dem Wert des Gitterabstands, sondern aus

der Lage der Auswertungsebene. Aus diesem Grund wurde die Entscheidung getroffen, mit den zwei fest gesetzten Ebenen den Bereich der Quelle des elektrischen Feldes bestmöglich abzudecken, um belastbare Ergebnisse zu erhalten.

Für atomaren Sauerstoff ergeben sich analoge Plots, bei denen sich einzig der Bereich des Minimums um etwa 0.1 mm zu kleineren Gitterabständen verschiebt. Hier liegt der ideale Bereich also zwischen 3.0 mm und 3.3 mm. Zur Kombination der Ergebnisse für die Simulation mit Stickstoff und Sauerstoff kann dann die Schnittmenge aus beiden gebildet werden. Daraus ergibt sich ein idealer Gitterabstand von **3.1 mm bis 3.3 mm**. Dabei ist es empfehlenswert eher einen Gitterabstand in der Nähe der unteren Grenze des ermittelten, idealen Bereichs zu wählen, da die Ionen sich für kleinere Abstände weniger stark an das Beschleunigungsgitter annähern. So kann eine schnellere Abnutzung des Gitters vermieden werden.

5.2 Diskussion

Im Rahmen dieses Studienprojekts wurde sich bemüht alle Eingabeparameter physikalisch zu begründen, um so eine möglichst realitätsgetreue Simulation zu schaffen. Dennoch sind im Verlauf der Simulation Punkte aufgefallen, welche für eine zukünftige Steigerung der Genauigkeit einer ausführlicheren Betrachtung bedürfen. Deshalb sollen diese Punkte in diesem Kapitel für eine möglichst gute Transparenz übersichtlich aufgeführt werden.

Bei der Definition des Ionenstrahls wurde in dieser Arbeit darauf verzichtet eine transversale und longitudinale Verteilung der Ionentemperatur bzw. Geschwindigkeit hinzuzufügen. Demnach startet der Strahl mit vollkommener Homogenität. Die Einführung einer solchen Abweichung könnte eine realistische Anzahl an Kollisionen mit dem Gittersystem ergeben. Das würde beispielsweise die Obergrenze des Gitterabstands weiter herabsetzen.

Wie im vorangegangenen Kapitel bereits gezeigt wurde, hat die Wahl der Auswertungsebene einen großen Einfluss auf die Werte des mittleren Divergenzwinkels. Im Idealfall sollte die Ebene so gelegt werden, dass sie sich im Mittelpunkt der Quelle des elektrischen Feldes innerhalb der Raumladung befindet. So würde die Kraft auf die Ionen, welche auch auf das Gittersystem wirkt, berücksichtigt werden, während die weiteren Raumladungseffekte, welche keine Kraftwirkung auf das Triebwerk haben, ignoriert werden. Dadurch könnte eine physikalisch begründete Lage der Auswertungsebene gefunden werden.

Hinzu kommt noch die in Kapitel 4.1 angesprochene Wahl des Plasma Offsets, also der Länge zwischen der linken Simulationsgrenze und dem Beginn des Abschirmgitters. Innerhalb dieses Abstands muss genügend Platz für die Ausbildung des Plasmameniskuses sein. Allerdings gerät die Simulation in Konvergenzprobleme, wenn der Offset zu groß gewählt wird, da, trotz der implementierten Relaxation, Oszillationen der Raumladung entstehen [7, S. 89]. Gerade für niedrige Startgeschwindigkeiten der Ionen (also niedrige Elektronentemperaturen) wirkt sich dieser Effekt auf die Bestimmung von Emittanz und Divergenzwinkel aus.

Obwohl die aufgeführten Punkte Einfluss auf die Güte der Simulation haben, kann mithilfe des geschriebenen Codes ein Bereich für den idealen Gitterabstand bestimmt werden. Dieser Abstand von **3.1 mm bis 3.3 mm** kann als Ausgangspunkt für Experimente am realen Extraktionsgitter genutzt werden. Eine solche Messung kann

anschließend auch zum Abgleich zwischen Experiment und Simulation genutzt werden. Außerdem zeigt diese Arbeit beispielhaft die Nutzung der Simulationsbibliothek IBSimu auf. Ausgehend von dieser Ausarbeitung kann die Bibliothek für weitere Anwendungen verwendet werden.

Literatur

[1] K. Holste et al., *Ion thrusters for electric propulsion: Scientific issues developing a niche technology into a game changer*, Review of Scientific Instruments **91** (2020).

[2] D. M. Goebel und I. Katz, *Fundamentals of Electric Propulsion* (John Wiley & Sons, 2008).

[3] A. Keudell, *Einführung in die Plasmaphysik*, Vorlesungsskript (2021).

[4] U. Stroth, *Plasmaphysik*, 2. Aufl. (Springer, 2018).

[5] A. Keudell und V. Schulz-von der Gathen, *Einführung in die Plasmaphysik II: Niedertemperaturplasmen*, Vorlesungsskript (2023).

[6] U. Motschmann et al., *Plasmaphysik*, Vorlesungsskript (2018).

[7] T. Kalvas, *Development and use of computational tools for modelling negative hydrogen ion source extraction systems*, Diss. (University of Jyväskylä, 2013).

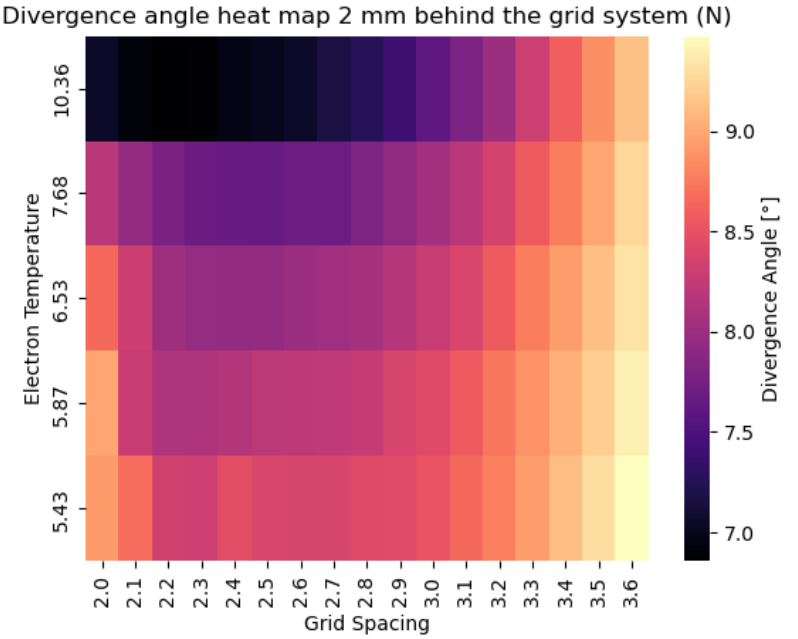
[8] T. Green, *Intense ion beams*, Reports on Progress in Physics **37**, 1257 (1974).

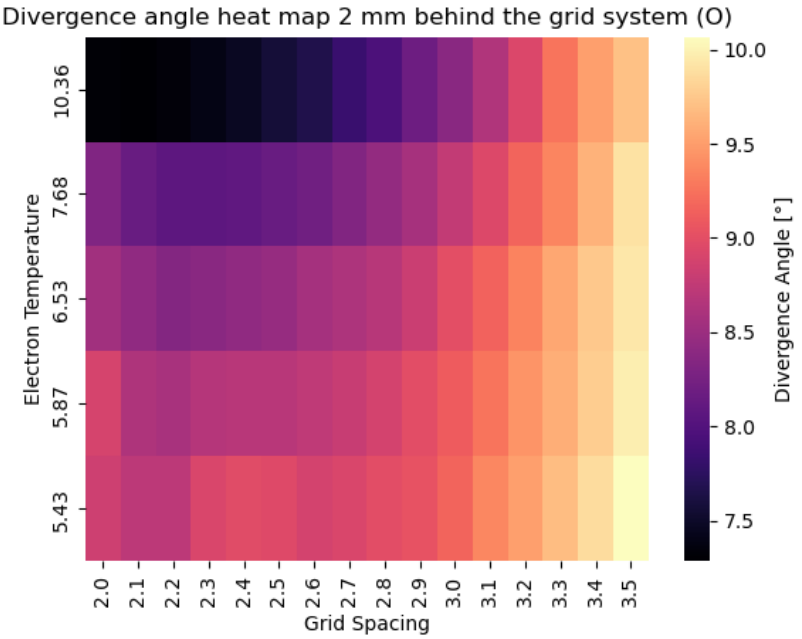
Danksagung

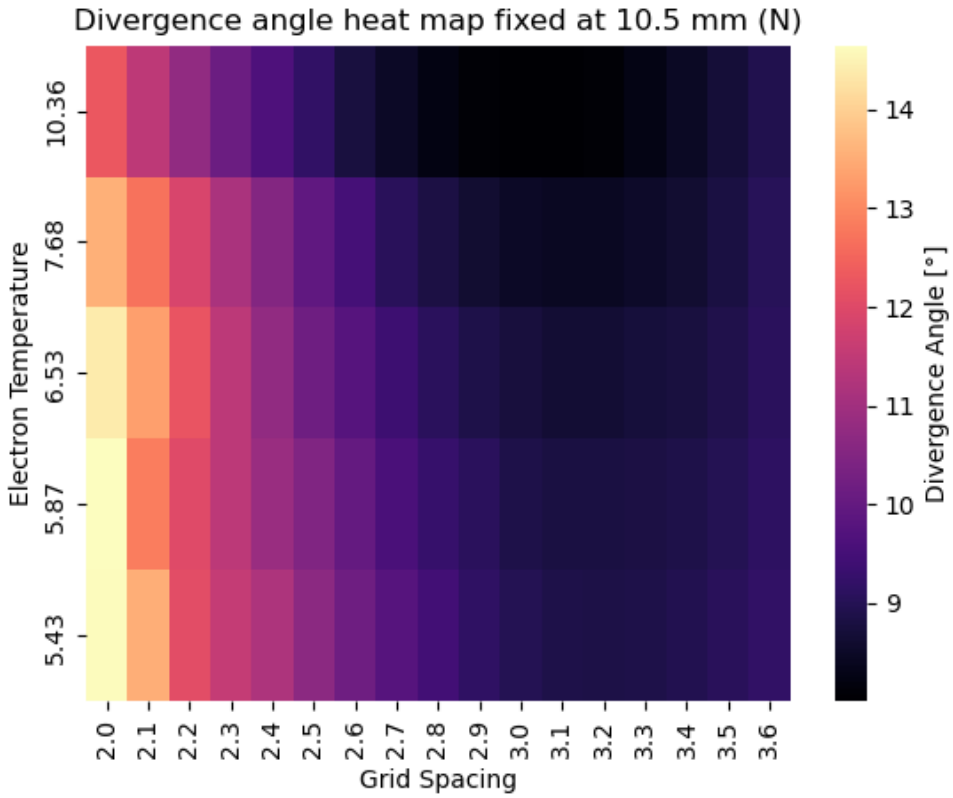
Zum Abschluss dieser Arbeit möchte ich mich bei meiner Betreuerin Jana Zorn bedanken, die mich mit dem nötigen Input versorgt und durch das Projekt geleitet hat. Auch Kristof Holste möchte ich Dank aussprechen, da er jederzeit für Nachfragen zur Verfügung stand. Vielen Dank auch an Kalle Bräumer, der mich in die Theorie zu den Plasmarandschichten eingeführt hat und sich meinen kritischen Nachfragen gestellt hat. Besonders möchte ich mich bei meinem sehr geschätzten Kommilitonen Lorenz Saalmann bedanken, der mich tatkräftig bei der Installation von IBSimu und dem Aufbau der Simulation unterstützt hat.

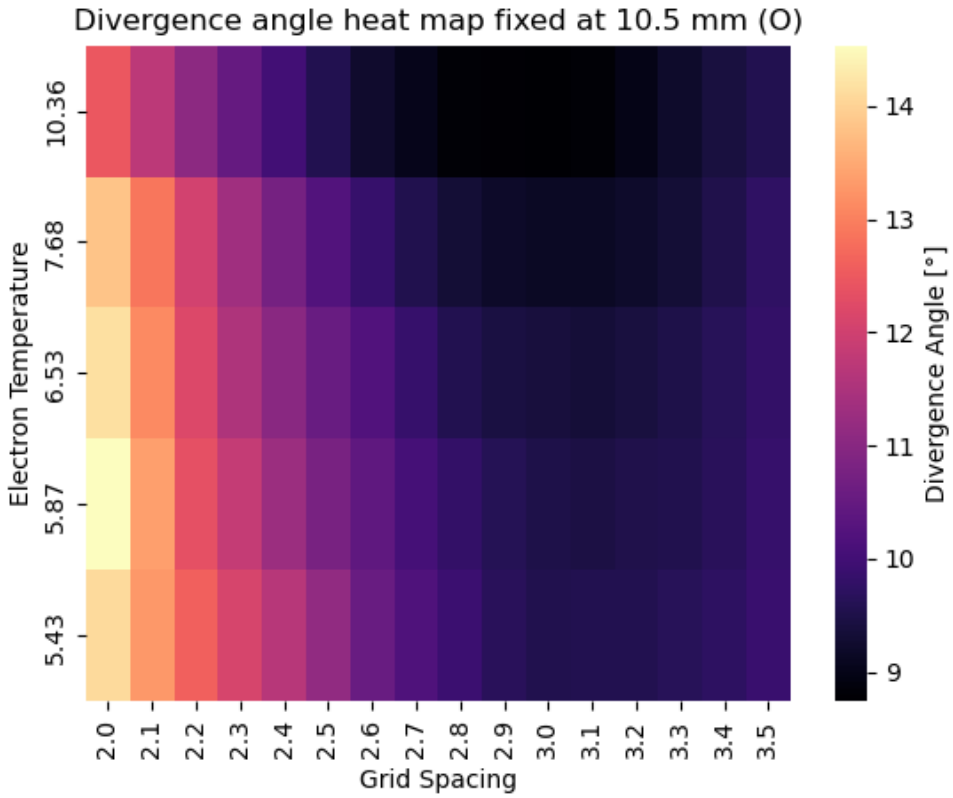
A Anhang

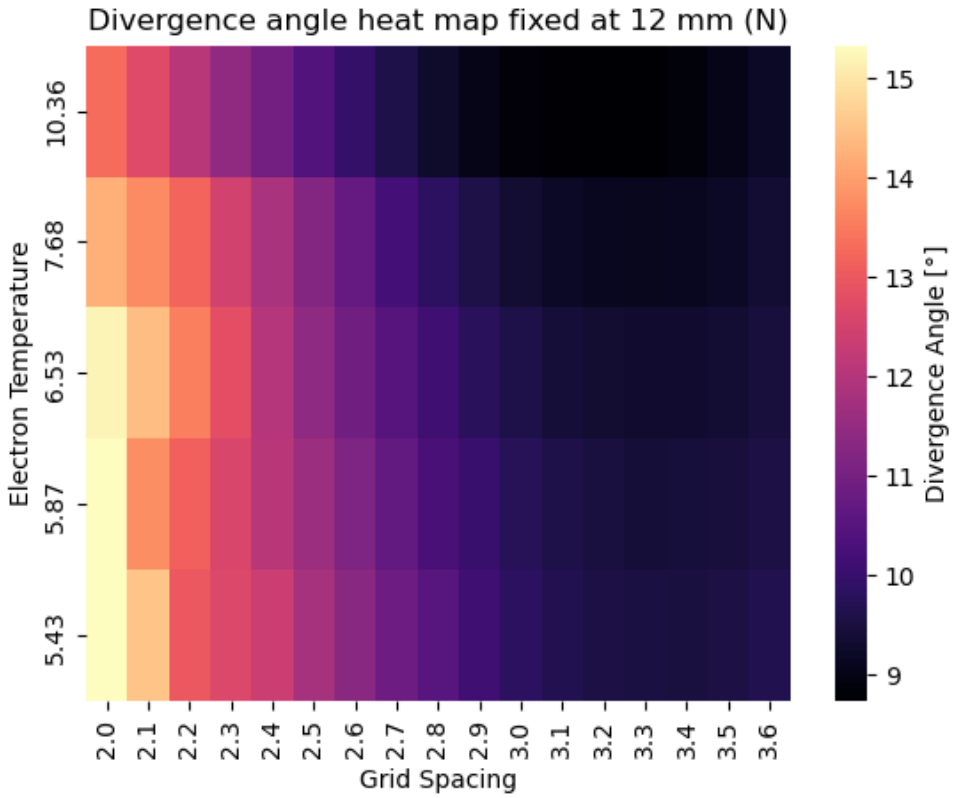
A.1 Alle Plots zur Auswertung von Emittanz und Divergenzwinkel

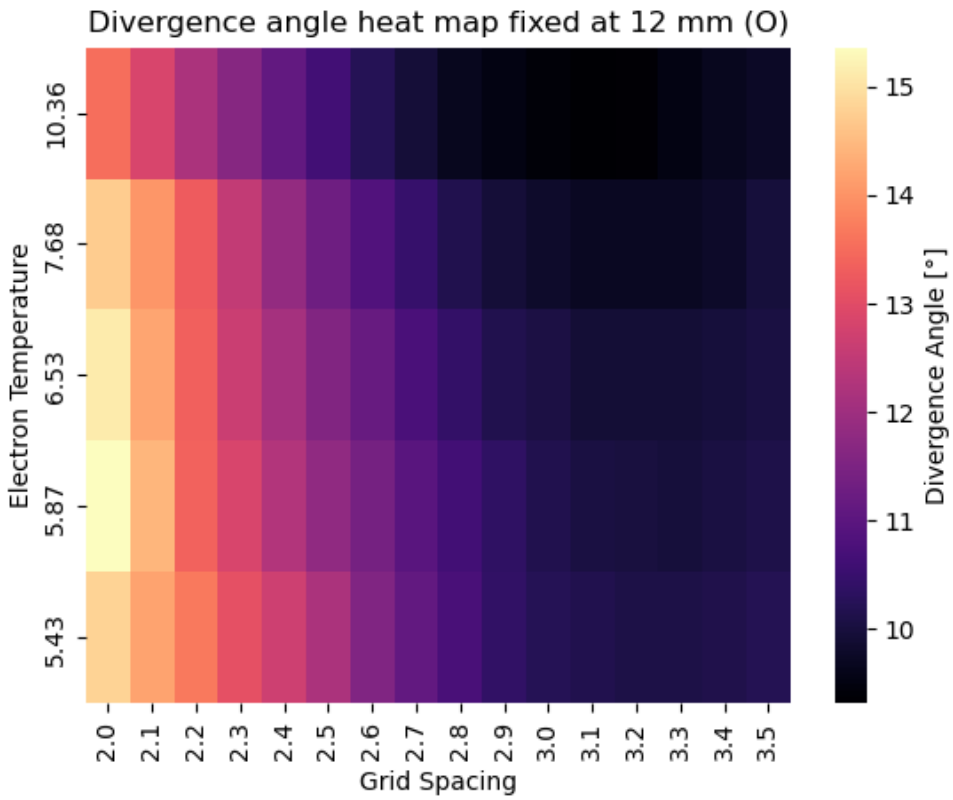


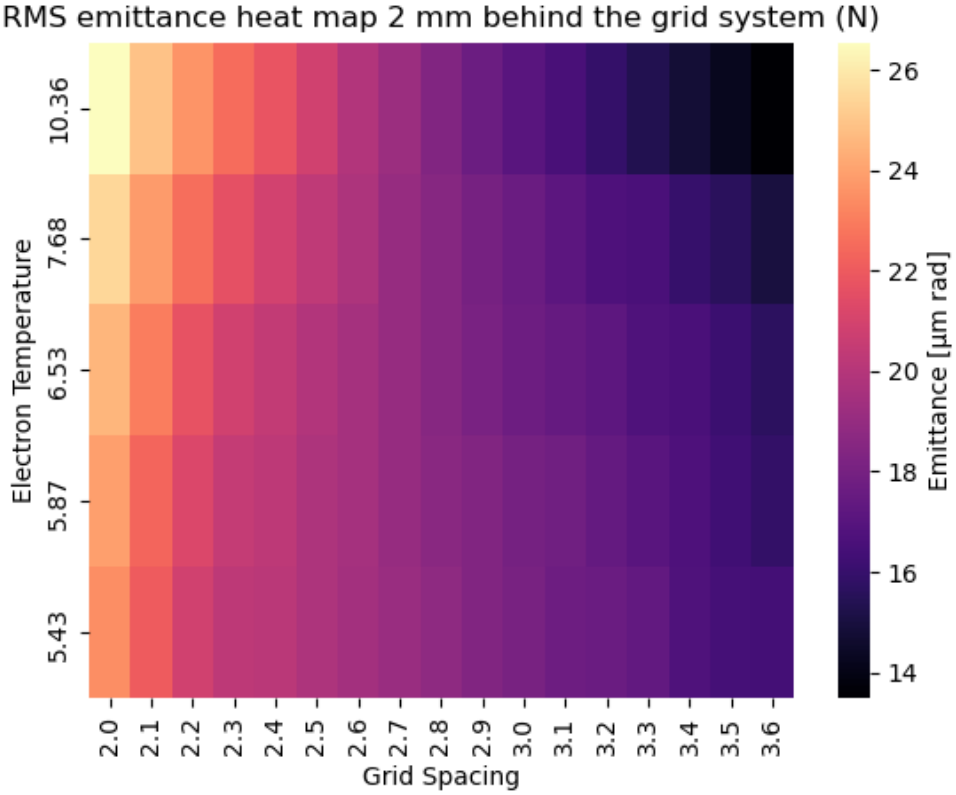


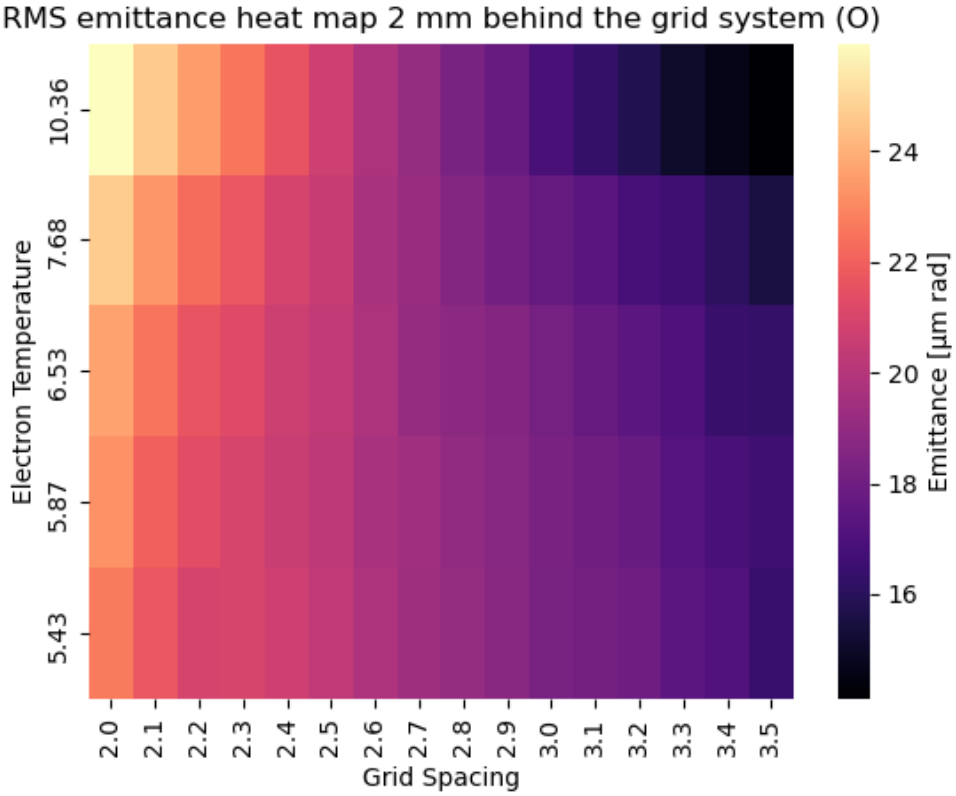

Divergence angle heat map fixed at 10.5 mm (N)
Electron Temperature
10.36
7.68
6.53
5.87
5.43
2.0
2.1
2.2
2.3
2.4
2.5
2.6
2.7
2.8
2.9
3.0
3.1
3.2
3.3
3.4
3.5
3.6
Grid Spacing
Divergence Angle [°]
14
13
12
11
10
9

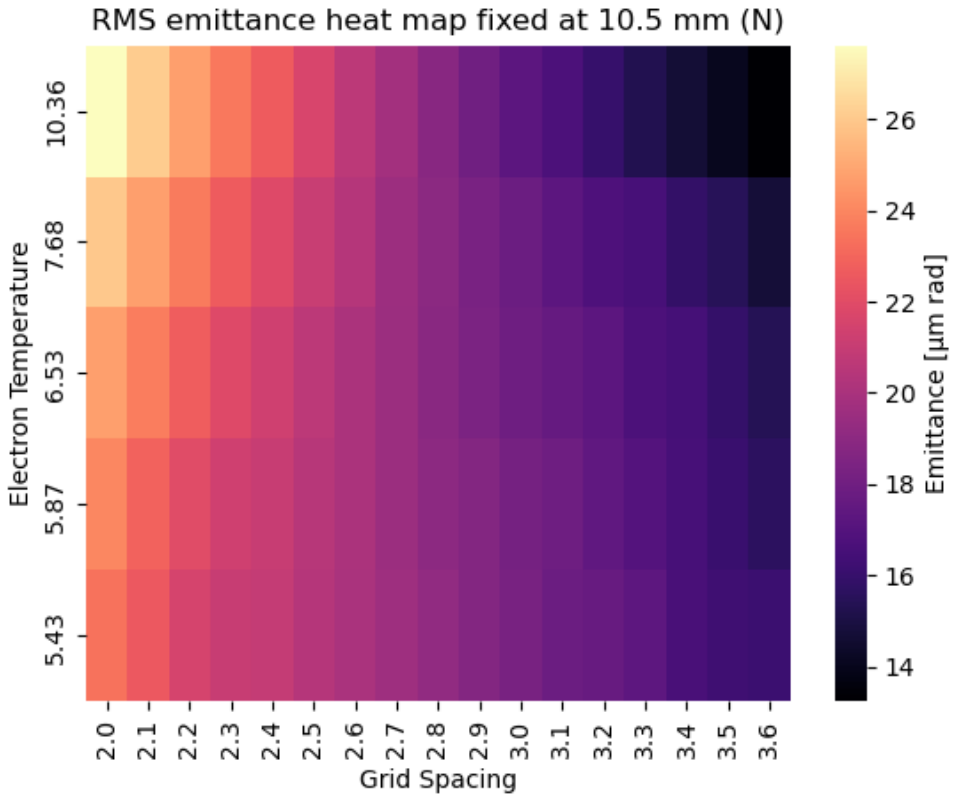

Divergence angle heat map fixed at 10.5 mm (O)
Electron Temperature
10.36
7.68
6.53
5.87
5.43
2.0
2.1
2.2
2.3
2.4
2.5
2.6
2.7
2.8
2.9
3.0
3.1
3.2
3.3
3.4
3.5
Grid Spacing
Divergence Angle [°]
14
13
12
11
10
9

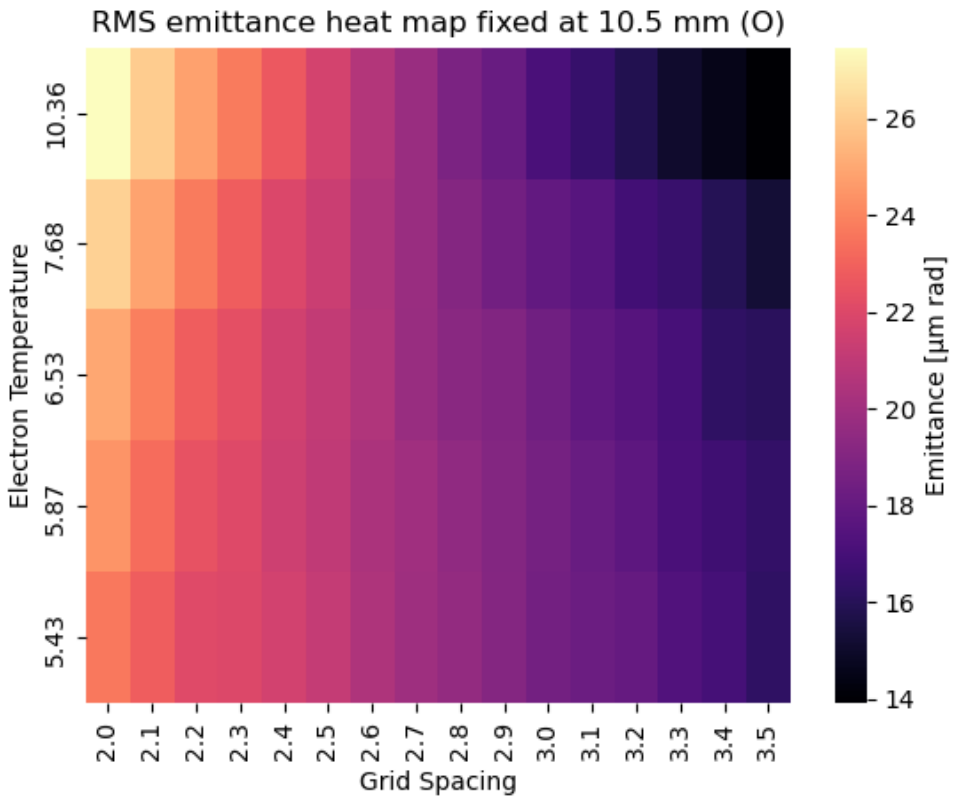

Divergence angle heat map fixed at 12 mm (N)
Electron Temperature
10.36
7.68
6.53
5.87
5.43
2.0
2.1
2.2
2.3
2.4
2.5
2.6
2.7
2.8
2.9
3.0
3.1
3.2
3.3
3.4
3.5
3.6
Grid Spacing
15
14
13
12
11
10
9
Divergence Angle [°]

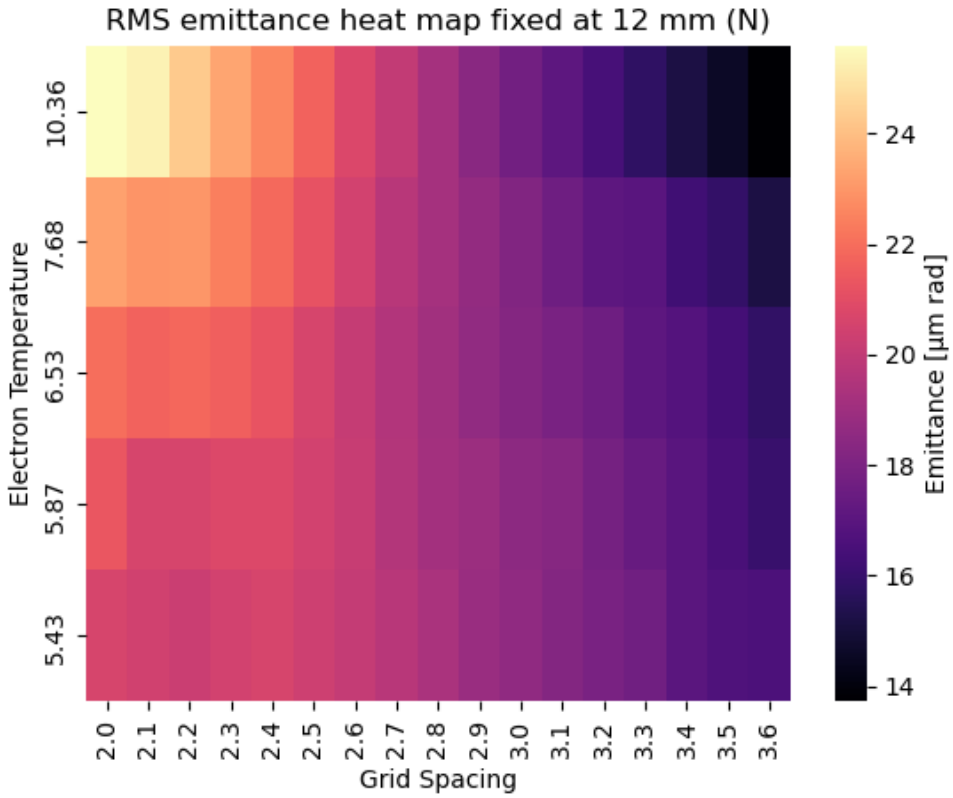

Divergence angle heat map fixed at 12 mm (O)
Electron Temperature
10.36
7.68
6.53
5.87
5.43
2.0
2.1
2.2
2.3
2.4
2.5
2.6
2.7
2.8
2.9
3.0
3.1
3.2
3.3
3.4
3.5
Grid Spacing
15
14
13
12
11
10
Divergence Angle [°]

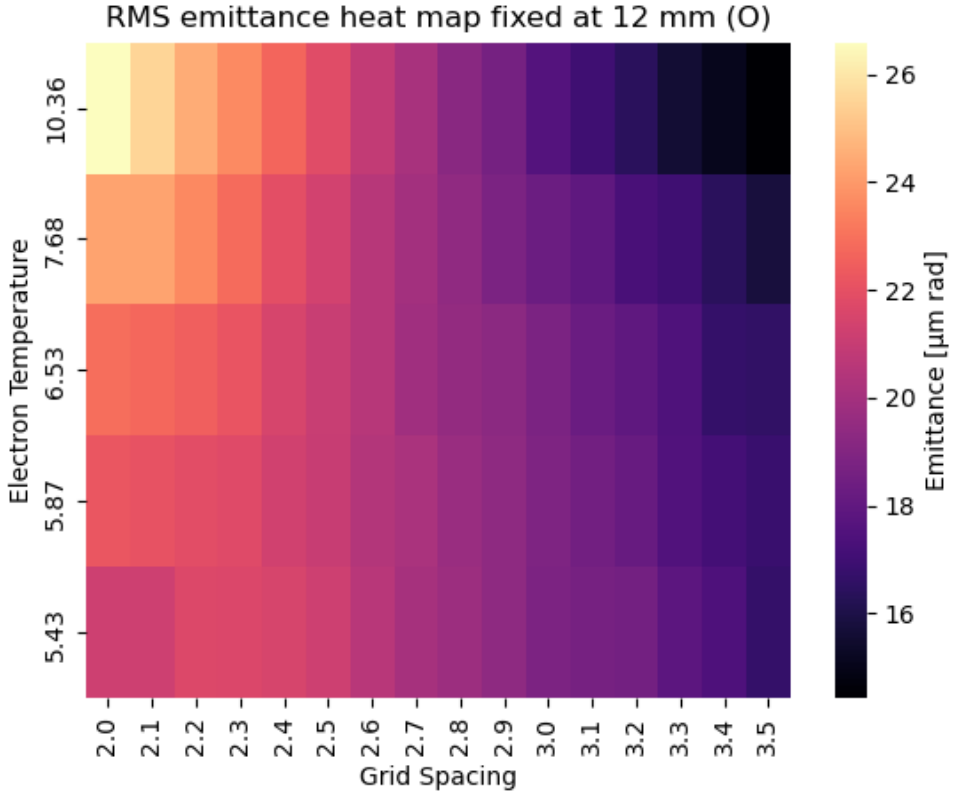

RMS emittance heat map 2 mm behind the grid system (N)
Electron Temperature
10.36
7.68
6.53
5.87
5.43
Grid Spacing
2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1 3.2 3.3 3.4 3.5 3.6
Emittance [µm rad]
26
24
22
20
18
16
14


RMS emittance heat map 2 mm behind the grid system (O)
Electron Temperature
10.36
7.68
6.53
5.87
5.43
Grid Spacing
2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1 3.2 3.3 3.4 3.5
Emittance [µm rad]
24
22
20
18
16


RMS emittance heat map fixed at 10.5 mm (N)
Electron Temperature
10.36
7.68
6.53
5.87
5.43
Grid Spacing
2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1 3.2 3.3 3.4 3.5 3.6
Emittance [µm rad]
26 24 22 20 18 16 14


RMS emittance heat map fixed at 10.5 mm (O)
Electron Temperature
10.36
7.68
6.53
5.87
5.43
Grid Spacing
2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1 3.2 3.3 3.4 3.5
Emittance [µm rad]
26 24 22 20 18 16 14


RMS emittance heat map fixed at 12 mm (N)
Electron Temperature
10.36
7.68
6.53
5.87
5.43
2.0
2.1
2.2
2.3
2.4
2.5
2.6
2.7
2.8
2.9
3.0
3.1
3.2
3.3
3.4
3.5
3.6
Grid Spacing
24
22
20
18
16
14
Emittance [µm rad]


RMS emittance heat map fixed at 12 mm (O)
Electron Temperature
10.36
7.68
6.53
5.87
5.43
2.0
2.1
2.2
2.3
2.4
2.5
2.6
2.7
2.8
2.9
3.0
3.1
3.2
3.3
3.4
3.5
Grid Spacing
26
24
22
20
18
16
Emittance [µm rad]

A.2 IBSimu Installation

Der folgende Anhang soll als Schritt für Schritt Anleitung für die Installation der IBSimu Bibliothek auf einem Windows Computer dienen. Da gerade die Software-Branche in ständiger Entwicklung ist, veralten solche Anleitungen leider schnell. Beispielsweise ist die Anleitung auf der Webseite von IBSimu aus dem Jahr 2016 und deshalb nicht mehr umsetzbar. Dieser Anhang bringt sie nun auf den Stand von 2024.

Zunächst wird zunächst die Einrichtung einer Linux-ähnlichen Umgebung in Windows beschrieben, da die IBSimu Bibliothek für eine solche geschrieben wurde. Der Autor von IBSimu empfiehlt hierfür die Software MSYS2, welche auf der offiziellen Webseite heruntergeladen werden kann[1]. Dabei ist es dringend zu empfehlen den Standardinstallationspfad `C:/msys64` beizubehalten. Anschließend ist das UCRT64 Terminal zu öffnen, um zunächst den C++ Compiler zu installieren. Dazu wird folgendes Kommando verwendet:

```
$ pacman -S mingw-w64-ucrt-x86_64-gcc
```

Nun ist die MSYS Umgebung mit MinGW und gcc Compiler einsatzbereit.

Als nächstes müssen einige Bibliotheken installiert werden, auf welchen der Code von IBSimu aufbaut. Dies geschieht erneut direkt über das Terminal mit dem *pacman* Befehl. Dabei ist darauf zu achten, dass jeweils die ucrt64-Version der Bibliotheken installiert wird:

```
$ pacman -S mingw-w64-ucrt-x86_64-pkg-config
$ pacman -S mingw-w64-ucrt-x86_64-emacs
$ pacman -S mingw-w64-ucrt-x86_64-gtk3
$ pacman -S mingw-w64-ucrt-x86_64-gsl
$ pacman -S mingw-w64-ucrt-x86_64-suitesparse
$ pacman -S mingw-w64-ucrt-x86_64-opencsg
$ pacman -S make
```

Anschließend muss die IBSimu Bibliothek selbst installiert werden. Die aktuelle Version ist auf der IBSimu-Homepage[2] verlinkt und kann als zip-Datei heruntergeladen werden. Diese ist in den neuen Ordner `C:/msys64/home/username/scr` zu entpacken. Nach einem Neustart des Terminals müssen nacheinander folgende Kommandos zur Einrichtung der Bibliothek ausgeführt werden:

[1] `https://www.msys2.org/`, letzter Zugriff: 14.01.2024
[2] `https://ibsimu.sourceforge.net/download.html`, letzter Zugriff: 15.01.2024

```
$ cd src/libibsimu-1.0.6/
$ ./configure -prefix=/home/username
$ make
$ make check
$ make install
```

Dabei kann es passieren, dass Fehlermeldungen auf fehlende Bibliotheken hinweisen, wodurch die Einrichtung abgebrochen wird. Die Befehle zum Download dieser Pakete können durch eine kurze Internet-Suche der Form *MSYS2 ucrt64 [insert library]* schnell gefunden werden.

Nun ist IBSimu im Prinzip einsatzbereit. Bevor jedoch das erste Programm geschrieben wird, sollte der Pfad zur Bibliothek in der *PKG_CONFIG_PATH* Variable hinterlegt werden. Der Befehl dazu kann manuell vor jeder Nutzung ausgeführt, oder im *.bashrc*, welches unter `C:/msys64/home/username` zu finden ist, hinterlegt werden. Dann wird die Variable jedes Mal bei Neustart des Terminals direkt ausgeführt. Der Befehl lautet:

```
export PKG_CONFIG_PATH=$PATH:/home/username/src/libibsimu-1.0.6
```

A.3 Python Code zur Auswertung

Beispielhaft wird hier der Code zur Analyse der Emittanz der Simulation mit Stickstoff dargestellt. Die Darstellung der anderen Heatmaps erfolgt analog.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import colormaps
import seaborn as sns

def plot_and_save_heatmap( dataframe, title, colorbar_label,
    png_name, colormap ):
     plt.close()
     plt.axis = sns.heatmap(dataframe.iloc[::-1,:-4],
         cbar_kws={'label': colorbar_label}, cmap = colormap)
     plt.xlabel("Grid Spacing")
     plt.xticks(rotation = 90)
     plt.ylabel("Electron Temperature")
     plt.title(title)
     plt.savefig('N_' + png_name)
     plt.show()
     return

emit_df1 = pd.read_csv('C:/msys64/home/peter/IBSimu-ExtractionGrid/
    RAM-EP_Grid/variableSimuCYL/2mm_var_emittance_mum_rad.txt',
                delim_whitespace = True)

emit_df1.columns = [str(2.0 + 0.1 * i)[:3] for i in range(21)]

plot_and_save_heatmap(emit_df1,
    "RMS emittance heat map 2 mm behind the grid system (N)",
    "Emittance [$\mu$m rad]",
    "emit_2_var.png",
    "magma")
```

A.4 Ausführlicher Simulationscode

Der gesamte, kommentierte C++ Code, der für die Simulation geschrieben wurde ist auf den folgenden Seiten angehängt. Auf Anfrage beim Autor kann der Zugang zum Git Respository mit allen Simulationsdateien angefragt werden.

IBSimu-ExtractionGrid\RAM-EP_Grid\variableSimuCYL\variableSimuCYL.cpp

```cpp
#include <cstdlib>
#include <sstream>
#include <fstream>
#include <iomanip>
#include <limits>
#include <functional>
#include "epot_bicgstabsolver.hpp"
#include "geometry.hpp"
#include "func_solid.hpp"
#include "epot_efield.hpp"
#include "meshvectorfield.hpp"
#include "particledatabase.hpp"
#include "ibsimu.hpp"
#include "error.hpp"
#include "geomplotter.hpp"
#include "particlediagplotter.hpp"
#include "fielddiagplotter.hpp"
#include "gtkplotter.hpp"
#include "cairo.h"

// Count the number of parameter changes for a parameter sweep.
int parameterChanges = 0;
int eTempChanges = 0;

// Define several arrays to collect the collision, emittance and angle data.
int collisions[40][8];
float emittance1[5][40];
float emittance2[5][40];
float emittance3[5][40];
float angle1[5][40];
float angle2[5][40];
float angle3[5][40];

float mm = 0.001;                   // Conversion factor from millimeter to meter.
float uOverMe = 1822.89;            // Atomic mass unit divided by the electron mass.

// Define simulation area in milimeters.
float width = 15.0;                 // Set width of simulation area.
float height = 1.75;                // Set height of the simulation area (dependent on grid geometry).
float nodeSpacing = 0.025;          // Set Spacing between simulation nodes.

// Setting grid parameters in milimeters.
float pOffset = 0.5;                 // Length of the plasma in front of the screen grid.
float sdGridRadius = 3.15 / 2;       // Radius of screen and deceleration grid.
float aGridRadius = 2.0 / 2;         // Radius of acceleration grid.
float sdGridWidth = 0.5;             // Width of screen and deceleration grid
float aGridWidth = 0.5;              // Width of acceleration grid
float gridSpacing = 2.8;             // Spacing between grids.

// Set additional parameters.
float electronTemp = 7.68;           // Electon temperature in [eV].
float eTempArray[5] = {5.43,         // Array with the electron temperature of different plasma operating points.
5.87, 6.53, 7.68, 10.36};            // in [eV]
```

```
54  int numberOfParticles = 2000;       // Number of simulated particles.
55  float charge = 1.0;                 // Ion charge in [e].
56  float mass = 14.0;                  // Ion mass in [u].
57  float totalCurrent = 0.3;           // beam current in [A].
58  int numberOfHoles = 499;       // Number of holes of the grid.
59
60  // Calculate beam current density.
61  int currentDensity = round( totalCurrent / (pow(sdGridRadius * mm, 2) * 3.14159 *
    numberOfHoles) );
62
63  // Calculate conversion factor from electron temperature to floating potential [V/eV].
64  float tempToPotential = log(sqrt((mass/6.28319) * uOverMe));
65
66  // Define grid voltages.
67  float sGridVoltage = 2.0e3;         // Screen grid voltage in [V].
68  float aGridVoltage = -200;          // Acceleration grid voltage in [V].
69
70  // Define geometry of screen grid
71  bool screenGrid( double x, double y, double z )
72  {
73      bool xBounds = pOffset * mm <= x && x <= ( pOffset + sdGridWidth ) * mm;
74      bool yBounds = sdGridRadius * mm <= y && y <= height * mm;
75      return( xBounds && yBounds );
76  }
77
78  // Define geometry of acceleration grid
79  bool accelerationGrid( double x, double y, double z )
80  {
81      float xOffset = pOffset + sdGridWidth + gridSpacing;
82      bool xBounds = xOffset * mm <= x && x <= ( xOffset + aGridWidth ) * mm;
83      bool yBounds = aGridRadius * mm <= y && y <= height * mm;
84      return( xBounds && yBounds );
85  }
86
87  // Define geometry of deceleration grid
88  bool decelerationGrid( double x, double y, double z )
89  {
90      float xOffset = pOffset + sdGridWidth + aGridWidth + 2 * gridSpacing;
91      bool xBounds = xOffset * mm <= x && x <= ( xOffset + sdGridWidth ) * mm;
92      bool yBounds = sdGridRadius * mm <= y && y <= height * mm;
93      return( xBounds && yBounds );
94  }
95
96  // Creates the simulation geometry according to the globally defined parameters.
97  Geometry createGeometry( void ){
98
99      /*  Creates the simulation geometry with help of the class Geometry.
100         MODE_CYL: Set geometry to a cylindical mode.
101         Int3D(int, int, int): Define Number of Nodes in x, y, and z direction.
102         Vec3D(0,0,0): Defines origin of the geometry.
103         nodeSpacing * mm: Defines the distance between nodes.
104     */
105     Geometry geom(  MODE_CYL,
106                     Int3D(round(width/nodeSpacing) + 1, round(height/nodeSpacing) + 1,
    1),
107                     Vec3D(0,0,0),
108                     nodeSpacing * mm );
109
```

```
    // Add electrodes to the geometry and set boundary numbers (1-6 reserved for simulation box boundaries).
    Solid *s1 = new FuncSolid( screenGrid );
    geom.set_solid( 7, s1 );
    Solid *s2 = new FuncSolid( accelerationGrid );
    geom.set_solid( 8, s2 );
    Solid *s3 = new FuncSolid( decelerationGrid );
    geom.set_solid( 9, s3 );

    // Set the voltage and boundary condition of each boundary
    geom.set_boundary( 1, Bound(BOUND_DIRICHLET,   sGridVoltage + electronTemp * tempToPotential) );
    geom.set_boundary( 2, Bound(BOUND_DIRICHLET,   0.0) );
    geom.set_boundary( 3, Bound(BOUND_NEUMANN,     0.0  ) );
    geom.set_boundary( 4, Bound(BOUND_NEUMANN,     0.0  ) );
    geom.set_boundary( 7, Bound(BOUND_DIRICHLET,   sGridVoltage) );
    geom.set_boundary( 8, Bound(BOUND_DIRICHLET,   aGridVoltage) );
    geom.set_boundary( 9, Bound(BOUND_DIRICHLET,   0.0) );
    geom.build_mesh();

    return geom;
}

/*  Plots a PNG of the geometry, trajectories and potential field of the simulation area.
    The creation of an interactive plot with a graphical user interface can be activated.
*/
void plotPNG(Geometry &geom, EpotField &epot, EpotEfield &efield, ParticleDataBase &pdb, MeshScalarField &scharge,
             std::string title, bool useGTK = false)
{
    if (!useGTK) {
        MeshScalarField tdens( geom );
        pdb.build_trajectory_density_field( tdens );
        GeomPlotter gplotter( geom );
        gplotter.set_size( 800, 600 );
        gplotter.set_font_size( 20 );

        gplotter.set_epot( &epot );
        std::vector<double> eqlines;
        eqlines.push_back( 2e3 - 4.0 );
        eqlines.push_back( 2e3 -2.0 );
        eqlines.push_back( 2e3 );
        eqlines.push_back( 2e3 + 2.0 );
        eqlines.push_back( 2e3 + 4.0 );
        gplotter.set_eqlines_manual( eqlines );
        gplotter.set_particle_database( &pdb );
        gplotter.set_particle_div( 0 );
        gplotter.set_trajdens( &tdens );
        gplotter.set_fieldgraph_plot( FIELD_TRAJDENS );
        gplotter.fieldgraph()->set_zscale( ZSCALE_RELLOG );
        gplotter.plot_png( title );
    } else {
        MeshScalarField tdens( geom );
        pdb.build_trajectory_density_field( tdens );
        GTKPlotter plotter( {}, {} );
        plotter.set_geometry( &geom );
        plotter.set_epot( &epot );
        plotter.set_efield( &efield );
```

```
        plotter.set_trajdens( &tdens );
        plotter.set_scharge( &scharge );
        plotter.set_particledatabase( &pdb );
        plotter.new_geometry_plot_window();
        plotter.run();
    }

}

//  Calculates the rms emittance and average divergence angle at a given level.
void calculateEmittance(ParticleDataBase &pdb, float *emittance, float *angle, float
level)
{
    std::string numTextGrid = std::to_string(gridSpacing);
    std::string numTextTemp = std::to_string(electronTemp);

    // Define a vector that holds all diagnostics data that is needed to calculate the
emittance.
    std::vector<trajectory_diagnostic_e> diagnostics;
    diagnostics.push_back(DIAG_R);
    diagnostics.push_back(DIAG_RP);
    diagnostics.push_back(DIAG_CURR);

    // Perform trajectory diagnsotics at the given level.
    TrajectoryDiagnosticData tdata;
    pdb.trajectories_at_plane(tdata, AXIS_X, level, diagnostics);

    // Save the calculated values of v_r/v_x to an array
    Emittance tanAngle(tdata(0).data(), tdata(1).data(), tdata(2).data());
    angle[parameterChanges] = tanAngle.xpave();

    // Mirror the diagnostics data to calculate the rms emittance and save it to an
array
    tdata.mirror(AXIS_R, 0);
    Emittance emit(tdata(0).data(), tdata(1).data(), tdata(2).data());
    // Save emittance in array for later output
    emittance[parameterChanges] = emit.epsilon();
}

// Main simulation function
void simu(int cycles)
{
    // Creates the simulation geometry according to the globally defined parameters
    Geometry geom = createGeometry();

    // Create a Potenial-, Charge-, Magnetic-, and Electric Field
    EpotField epot( geom );
    MeshScalarField scharge( geom );
    MeshVectorField bfield;
    EpotEfield efield( epot );

    // Set extrapolation conditions for the electric field for all simulation
boundaries
    field_extrpl_e efldextrpl[6] = { FIELD_EXTRAPOLATE, FIELD_EXTRAPOLATE,
                    FIELD_SYMMETRIC_POTENTIAL, FIELD_EXTRAPOLATE,
                    FIELD_EXTRAPOLATE, FIELD_EXTRAPOLATE };
    efield.set_extrapolation( efldextrpl );

    // Initialize the Poisson Equation solver
    EpotBiCGSTABSolver solver( geom );
```

```
221
222     // Initialize the plasma by setting its initial volume and voltage
223     InitialPlasma initp( AXIS_X, ( pOffset + 0.5) * mm);
224     solver.set_initial_plasma(sGridVoltage + electronTemp * tempToPotential, &initp);
225
226     // Initialize the ParticleDataBase which contains the simulated Ions and their
    trajectories
227     ParticleDataBaseCyl pdb( geom );
228     bool pmirror[6] = { false, false, true, false, false, false };
229     pdb.set_mirror( pmirror );
230
231     /*  Main iteration loop that continues until convergence is assumed (after a
    ceratin number of cycles).
232         In the first iteration, the Laplace Equation is solved, which gives the
    potential field of the geometry
233         without the ion beam (i.e without space charge deposition).
234         Thereafter, the following steps will be repeated until convergence:
235             1. Calculate the ion trajectories and deposit space charge at the
    simulation nodes.
236             2. Solve the Poisson Equation to get the potential field and recalculate
    the electric field.
237     */
238     for( int i = 0; i < cycles; i++ ) {
239
240         // Redefine Plasma to a non-linear model after the first iteration with the
    initial plasma.
241         if( i == 1 ) {
242             double rhoe = pdb.get_rhosum();
243             /*  Change to the exponential plasma model, by setting the total electron
    charge, electron
244                 temperature and floating potential */
245             solver.set_pexp_plasma( rhoe, electronTemp, sGridVoltage + electronTemp *
    tempToPotential );
246         }
247
248         // Solve the Poisson Equation and recalculate the electic field
249         solver.solve( epot, scharge );
250         efield.recalculate();
251
252         //  Clear the particle database and add a new beam with the globally defined
    parameters.
253         pdb.clear();
254         pdb.add_2d_beam_with_energy(numberOfParticles, currentDensity, charge, mass,
255                                     electronTemp / 2, 0.0, 0.0, 0.0, 0.0, 0.0,
256                                     height * mm);
257         // Calculate the trajectory of all added particles through the geometry.
258         pdb.iterate_trajectories( scharge, efield, bfield );
259
260     }
261
262     // Save all boundary collisions for the given parameter in an array
263     for(uint32_t a = 0; a < pdb.get_statistics().number_of_boundaries(); a++){
264         collisions[parameterChanges][a] = pdb.get_statistics().bound_collisions(a+1);
265     }
266
267     // Calculate the ion beam's emittance at the end of the acceleration grid.
268     calculateEmittance( pdb, emittance1[eTempChanges], angle1[eTempChanges],
269                         (pOffset + 2 * sdGridWidth + 2 * gridSpacing + aGridWidth +
    2.0) * mm);
270
271     // Calculate the ion beam's emittance at 10.5mm.
```

```
272     calculateEmittance( pdb, emittance2[eTempChanges], angle2[eTempChanges],
273                         10.5 * mm);
274 
275     // Calculate the ion beam's emittance at 12mm.
276     calculateEmittance( pdb, emittance3[eTempChanges], angle3[eTempChanges],
277                         12.0 * mm);
278 
279     // Plot the simulation area as a PNG with the electron temperature and grid spacing
    included in the name.
280     gridSpacing = round(gridSpacing * 10) / 10;
281     std::string numTextGrid = std::to_string(gridSpacing);
282     std::string numTextTemp = std::to_string(electronTemp);
283     plotPNG(geom, epot, efield, pdb, scharge, numTextTemp.substr(0, numTextTemp.find("
    .") + 3) + "_eV_geom_" +
284     numTextGrid.substr(0, numTextGrid.find(".") + 2) + "_mm.png", false);
285 }
286 
287 // Performs a parameter sweep, for any given simulation parameter
288 void doParameterSweep(float *parameter, float from, float to, float step) {
289 
290     std::string eTempText = std::to_string(electronTemp);
291 
292     // Sweep through the parameter and execute the simulation in every iteration.
293     for(*parameter = from; *parameter <= to; *parameter += step){
294         simu(11);
295         parameterChanges++;
296     }
297 
298     /*  Create a table with the number of collisions to each boundary for the different
    parameter values.
299         Print it to the console and a .txt file.
300     */
301     std::ofstream dout( "collisions_" + eTempText.substr(0, eTempText.find(".") + 3) +
    "_eV.txt",
302                         std::ios_base::trunc);
303     ibsimu.message(1) << "\nCollisions per Boundary (by number) for the given parameter
    (dimension in first column) \n";
304     dout              << "Collisions per Boundary (by number) for the given parameter
    (dimension in first column) \n";
305     ibsimu.message(1) << "\t1\t2\t3\t4\t5\t6\t7\t8\t9 \n";
306     dout              << "\t1\t2\t3\t4\t5\t6\t7\t8\t9 \n";
307     for(int i = 0; i < parameterChanges; i++){
308         std::string output;
309         for(int b = 0; b < 9; b++){
310             output += std::to_string(collisions[i][b]) + "\t";
311         }
312         ibsimu.message(1) << (from + i * step) << "\t" << output << "\n";
313         dout              << (from + i * step) << "\t" << output << "\n";
314     }
315 
316     dout.close();
317 }
318 
319 // Save the emittance and angle data from the given arrays to a .txt file.
320 void saveEmittanceAndAngleData(std::string filename, float emittance[5][40], float
    angle[5][40], float start, float step)
321 {
322     ibsimu.message(1) << "Saving " << filename << " emittance and angle data to txt
    file" << "\n";
323 
324     // Create files.
```

```
    std::ofstream emitout(filename + "_emittance_mum_rad.txt", std::ios_base::trunc);
    std::ofstream angleout(filename + "_angle_rad.txt", std::ios_base::trunc);
    for (int i = 0; i < eTempChanges; i++)
    {
        std::string output1;
        std::string output2;
        for (int a = 0; a < parameterChanges; a++)
        {
            output1 += std::to_string(emittance[i][a] * 1e6) + "\t";
            output2 += std::to_string(angle[i][a]) + "\t";
            if (i == 0)
            {
                emitout << "\t\t" << (start + a * step);
                angleout << "\t\t" << (start + a * step);
            }
        }
        // Output emittance and angle for every temerature and grid spacing
        emitout << "\n" << (eTempArray[i]) << "\t" << output1;
        angleout << "\n" << (eTempArray[i]) << "\t" << output2;
    }
    emitout.close();
    angleout.close();
}

int main( int argc, char **argv )
{
    try {
        ibsimu.set_message_threshold( MSG_VERBOSE, 1 );
        ibsimu.set_thread_count( 16 );

        // Set grid spacing sweep parameters.
        float start = 2.0f;
        float end = 4.0f;
        float step = 0.1f;

        // Iterate grid spacing sweep over all electron temperatures.
        for(int i = 0; i < 5; i++){
            parameterChanges = 0;
            electronTemp = eTempArray[i];
            doParameterSweep(&gridSpacing, start, end, step);
            eTempChanges++;
        }

        saveEmittanceAndAngleData("2mm_var", emittance1, angle1, start, step);
        saveEmittanceAndAngleData("10.5mm_fixed", emittance2, angle2, start, step);
        saveEmittanceAndAngleData("12mm_fixed", emittance3, angle3, start, step);

    } catch ( Error e ) {
        e.print_error_message( ibsimu.message( 0 ) );
        exit( 1 );
    }

    return( 0 );
}
```